UNIVERSITY OF CALIFORNIA SAN DIEGO

Robust and Efficient Deep Learning for Multimedia Generation and Recognition

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Electrical Engineering (Computer Engineering)

by

Shehzeen Samarah Hussain

Committee in charge:

Professor Farinaz Koushanfar, Chair
Professor Shlomo Dubnov
Professor Tara Javidi
Professor Ryan Kastner

2023

The Dissertation of Shehzeen Samarah Hussain is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

DEDICATION

To my dearest friends and family.

EPIGRAPH

The future depends on some graduate student who is deeply suspicious of everything I have said.

— Geoffrey Hinton.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to begin by expressing my deepest and most sincere gratitude to my PhD advisor and committee chair Professor Farinaz Koushanfar. Her resolute guidance, profound wisdom, and innovative ideas have been nothing short of invaluable throughout every step of my doctoral journey. I consider myself incredibly fortunate to have had the privilege of being mentored by such a remarkable and steadfast female researcher, whose dedication and support have been the very cornerstone of my academic achievement. Under her tutelage, I have not only grown academically but have also been inspired to embrace challenges and push the boundaries of knowledge in my field. Professor Koushanfar's unwavering belief in my potential has instilled in me a newfound confidence and determination to pursue excellence in research, and I will carry the lessons she imparted with me throughout my professional journey.

I am truly indebted to the remarkable community of professors I had the privilege of meeting during my time at UCSD. My heartfelt appreciation goes to Professor Shlomo Dubnov and Professor Julain McAuley for their invaluable feedback and collaboration on numerous research projects. Their expertise and guidance have been instrumental in shaping the quality and direction of my work, and I am immensely grateful for the opportunity to learn from them. I am equally thankful to Professor Ryan Kastner, whose collaboration on my research and presence on my committee have been of immense value. I would also like to thank Professor Gary Cottrell for his constructive feedback on my research and Professor Tara Javidi for her support by serving on my PhD committee.

Words fail to express the depth of my gratitude to Paarth Neekhara, whose collaboration throughout my PhD journey has been an invaluable source of inspiration and strength. This research was made possible by many brainstorming sessions, brimming with his brilliant ideas and unwavering support. My heartfelt thanks go to him for being an exceptional partner in this research endeavor.

I would like to express my sincere appreciation to the following researchers across academia and industry, whose contribution was crucial to the research included in this dissertation:

Javier Duarte, Mojan Javaheripi, Xinqiao Zhang, Nojan Sheybani, Malhar Jere, Jinglong Du, Brian Dolhansky, Joanna Bitton, Cristian Canton Ferrer, Van Nguyen, Shuhua Zhang, Erik Visser, Jocelyn Huang, Jason Li, Boris Ginsburg. I am also grateful for the privilege of meeting some remarkable scholars and colleagues at UCSD, who shared the same passion for knowledge and discovery: Lorraine Hossain, Siam Hussain, Huili Chen, Mohammad Samragh, Sadegh Riazi, Jung-woo Chang, Ruisi Zhang, Nasimeh Heydaribeni, Zahra Ghodsi, Seira Hidano, Soheil Shabgahi, Shashank Balla, Yaman Jandali, Rishabh Ranjan and Angelique Taylor. Their presence and shared enthusiasm fostered a supportive and inspiring environment, that nurtured both personal growth and learning. Additionally, I would like to extend my sincerest gratitude to the UCSD ECE graduate student affairs department, particularly Teresa Chiu for creating a welcoming and supportive environment for me, where I always felt comfortable approaching her with any concerns or challenges I faced.

Finally, I would like to acknowledge the unwavering support and understanding of my friends and family throughout this challenging yet rewarding journey. My parents Zakia Sultana, Zahid Hussain, brother Saquib Hussain and friends Risana Nahreen Malik, Shehtaz Huq, Salwa Hoque, Omid Meh, Navin Rahman, Ruha Aziz, Prakhar Pandey, Palash Agarwal, Soham Shah, Mathew Sam, Erik Seetao, Barsha Dash, Safwan Haque, Safwan Saif, Chitula Chipimo, Jialei Xu, Hannah Rosen and Frida Moller. Their encouragement, patience, and belief in my abilities have sustained me during moments of self-doubt and have given me the strength to persevere.

Chapter 2 contains material found in the following two papers (1) *ReFace: Adversarial Transformation Networks for Real-time Attacks on Face Recognition Systems*. IEEE/IFIP International Conference on Dependable Systems and Networks, 2023. Hussain, Shehzeen; Huster, Todd; Mesterharm, Chris; Neekhara, Paarth; Koushanfar, Farinaz. (2) *Universal Adversarial Perturbations for Speech Recognition Systems*. Interspeech, 2019. Neekhara, Paarth; Hussain, Shehzeen; Pandey, Prakhar; Dubnov, Shlomo; McAuley, Julian; Koushanfar, Farinaz. The dissertation author was the primary investigator and author of paper (1). The dissertation author and Paarth Neekhara made equal contributions to the work done in paper (2).

Chapter 3 is a reprint of the material as it appears in two papers (1) *Adversarial Reprogramming of Text Classification Neural Networks*. Empirical Methods in Natural Language Processing, 2019. Neekhara, Paarth; Hussain, Shehzeen; Dubnov, Shlomo; Koushanfar, Farinaz. (2) *Cross-modal Adversarial Reprogramming*. IEEE Winter Conference on Applications of Computer Vision, 2022. Neekhara, Paarth; Hussain, Shehzeen; Du, Jinglong; Dubnov, Shlomo; Koushanfar, Farinaz; McAuley, Julian. The dissertation author and Paarth Neekhara made equal contributions to this work.

Chapter 4 is a reprint of the material as it appears in *WaveGuard: Understanding and Mitigating Audio Adversarial Examples*. USENIX Security Symposium, 2021. Hussain, Shehzeen; Neekhara, Paarth; Dubnov, Shlomo; McAuley, Julian; Koushanfar, Farinaz. The dissertation author was the primary investigator and author of this paper.

Chapter 5 is a partial reprint of the material as it appears in *FastWave: Accelerating Autoregressive Convolutional Neural Networks on FPGA*. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2019. Hussain, Shehzeen; Javaheripi, Mojan; Neekhara, Paarth; Kastner, Ryan; Koushanfar, Farinaz. The dissertation author was the primary investigator and author of this paper.

Chapter 6 contains material found in the following two papers. (1) *Expressive Neural Voice Cloning*. Neekhara, Paarth; Hussain, Shehzeen; Dubnov, Shlomo; Koushanfar, Farinaz; McAuley, Julian. Asian Conference on Machine Learning 2021. (2) *Controllable Speech Synthesis with Iterative Refinement using Self Transformations*. 2023. Neekhara, Paarth; Hussain, Shehzeen; Ranjan, Rishabh; Dubnov, Shlomo; Koushanfar, Farinaz; McAuley, Julian. Currently under review for publication. The dissertation author and Paarth Neekhara made equal contributions to this work.

Chapter 7 contains material found in the following two papers. (1) *Adversarial Deepfakes: Evaluating Vulnerability of Deepfake Detectors to Adversarial Examples*. Hussain, Shehzeen; Neekhara, Paarth; Jere, Malhar; Koushanfar, Farinaz; McAuley, Julian. IEEE Winter Conference on Applications of Computer Vision, 2021. (2) *Exposing Vulnerabilities of Deepfake Detection*

Systems with Robust Attacks. Hussain, Shehzeen; Neekhara, Paarth; Dolhansky, Brian; Bitton, Joanna; Canton, Cristian; McAuley, Julian; Koushanfar, Farinaz. ACM Journal on Digital Threats: Research and Practice, Vol 3, 2022. The dissertation author was the primary investigator and author of these papers.

Chapter 8 is a reprint of the material as it appears in *FastStamp: Accelerating Neural Steganography and Digital Watermarking of Images on FPGAs*. IEEE/ACM International Conference on Computer-Aided Design, 2022. Hussain, Shehzeen; Sheybani, Nojan; Neekhara, Paarth; Zhang, Xinqiao; Duarte, Javier; Koushanfar, Farinaz. The dissertation author was the primary investigator and author of this paper.

This dissertation was supported, in parts, by the Charles Lee Powell Foundation Fellowship, NSF TILOS AI institute award number 2112665, NSF-CNS award number 2016737, ARO (W911NF1910317), ARO MURI (W911NF20S0009), SRC-Auto (2019-AU-2899), DoD UCR W911NF2020267 (MCA S-001364), Defense Advanced Research Projects Agency (DARPA) contract HR00112090093, and in part, by the U.S. Government.

VITA

2014	B.A. in Physics, Mount Holyoke College
2015	B.S. in Electrical Engineering, University of Massachusetts Amherst
2019	M.S. in Electrical Engineering (Computer Engineering), University of California San Diego
2023	Ph.D in Electrical Engineering (Computer Engineering), University of California San Diego

PUBLICATIONS

Shehzeen Hussain, Todd Huster, Chris Mesterharm, Paarth Neekhara, Farinaz Koushanfar "ReFace: Adversarial Transformation Networks for Real-time Attacks on Face Recognition Systems", to appear in Proceedings of IEEE International Conference on Dependable Systems and Networks (DSN), 2023

Shehzeen Hussain*, Paarth Neekhara*, Jocelyn Huang, Jason Li, Boris Ginsburg "ACE-VC: Adaptive and Controllable Voice Conversion using Explicitly Disentangled Self-supervised Speech Representations", to appear in Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2023

Jung-Woo Chang, Nojan Sheybani, **Shehzeen Hussain**, Mojan Javaheripi, Seira Hidano, Farinaz Koushanfar "NetFlick: Adversarial Flickering Attacks on Deep Learning Based Video Compression", in Proceedings of International Conference on Learning Representations (ICLR) Workshop on ML4IoT, 2023

Shehzeen Hussain*, Nojan Sheybani*, Paarth Neekhara*, Xinqiao Zhang, Javier Duarte, Farinaz Koushanfar "FastStamp: Accelerating Neural Steganography and Digital Watermarking of Images on FPGAs", in Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2022

Shehzeen Hussain*, Paarth Neekhara*, Brian Dolhansky, Joanna Bitton, Cristian Canton Ferrer, Julian McAuley, Farinaz Koushanfar "Exposing Vulnerabilities of Deepfake Detection Systems with Robust Attacks", in Proceedings of ACM Journal on Digital Threats Research and Practices (DTRAP), 2022

Shehzeen Hussain, Van Nguyen, Shuhua Zhang, Erik Visser "Multi-task Voice Activated Framework using Self-supervised Learning", to appear in Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2022

Paarth Neekhara*, **Shehzeen Hussain***, Jinglong Du, Shlomo Dubnov, Farinaz Koushanfar, Julian McAuley "Cross-modal Adversarial Reprogramming", in Proceedings of IEEE Winter Conference on Applications of Computer Vision (WACV), 2022

Paarth Neekhara*, **Shehzeen Hussain***, Shlomo Dubnov, Farinaz Koushanfar, Julian McAuley "Expressive Neural Voice Cloning", in Proceedings of Asian Conference on Machine Learning (ACML), 2021

Shehzeen Hussain*, Paarth Neekhara*, Shlomo Dubnov, Julian McAuley, Farinaz Koushanfar "WaveGuard: Understanding and mitigating audio adversarial examples", in Proceedings of USENIX Security (USENIX), 2021

Shehzeen Hussain*, Paarth Neekhara*, Malhar Jere, Julian McAuley, Farinaz Koushanfar "Adversarial DeepFakes: Evaluating Vulnerability of Deepfake Detectors to Adversarial Examples", in Proceedings of IEEE Winter Conference on Applications of Computer Vision (WACV), 2021

Paarth Neekhara, **Shehzeen Hussain**, Shlomo Dubnov, Farinaz Koushanfar "Adversarial Reprogramming of Text Classification Neural Networks", in Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP), 2019

Paarth Neekhara*, **Shehzeen Hussain***, Prakhar Pandey, Shlomo Dubnov, Julian McAuley, Farinaz Koushanfar "Universal Adversarial Perturbations for Speech Recognition Systems", in Proceedings of Interspeech 2019

Shehzeen Hussain, Mojan Javaheripi, Paarth Neekhara, Ryan Kastner, Farinaz Koushanfar "FastWave: Accelerating Autoregressive Convolutional Neural Networks on FPGA", in Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2019

ABSTRACT OF THE DISSERTATION

Robust and Efficient Deep Learning for Multimedia Generation and Recognition

by

Shehzeen Samarah Hussain

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California San Diego, 2023

Professor Farinaz Koushanfar, Chair

Deep Neural Networks (DNNs) have transformed the field of multimedia generation and recognition by replacing traditional hand-engineered systems in domains like vision, speech and text. This is because DNNs can operate end-to-end and model complex dependencies yielding state-of-the-art results on several generation and recognition benchmarks. However, there are three key challenges that need to be addressed for the practical, secure and reliable deployment of DNN-based media processing systems: 1) Robustness: DNNs are vulnerable to adversarial attacks, 2) Data-Requirement: DNNs often require large amounts of labelled data, 3) Compute-Efficiency: DNNs require extensive compute and resources.

My research focuses on addressing the above three challenges of DNN based multimedia

generation and recognition systems. On the robustness side, I first analyze practical vulnerabilities of DNN-based recognition systems and then propose a robust defense framework that can reliably identify adversarial inputs using perceptually informed input transformations. To address the challenge of data-requirement, I develop training frameworks that can effectively adapt foundation models trained using self-supervised learning for recognition and synthesis tasks in a data-efficient manner. Finally, to address the challenge of compute-efficiency, I propose acceleration methods using hardware-software codesign that significantly reduce the latency and resource-requirement while preserving the synthesis quality of DNN generators.

Chapter 1

Introduction

Multimedia recognition and multimedia synthesis are two interconnected yet distinct fields within the domain of multimedia processing. While both involve the analysis and understanding of multimedia data, they differ in their primary objectives and methodologies. Multimedia recognition focuses on the task of automatically identifying and categorizing different types of media content. It involves the development of algorithms and models that can accurately recognize and classify images, videos, audio, and textual information. For instance, in image recognition, the goal is to determine the content of an image, such as identifying objects, scenes, or specific visual patterns. Similarly, in speech recognition, the aim is to transcribe spoken words in an audio signal into written text, enabling applications like voice assistants or transcription services. Other examples of multimedia recognition include video action recognition, sentiment analysis in text, or music genre classification.

On the other hand, multimedia synthesis involves the generation or creation of new multimedia content based on given inputs or desired specifications. It encompasses algorithms and techniques for generating realistic or novel media content, such as images, videos, audio, and text. For instance, in image synthesis, generative models like generative adversarial networks (GANs) [7] can generate realistic images that resemble a given style or set of attributes. Text-to-speech synthesis algorithms can convert written text into natural-sounding speech. Video synthesis techniques can be used to generate new videos by altering existing footage or creating

entirely new visual sequences. These examples highlight the creative aspect of multimedia synthesis, enabling applications like virtual reality, content generation, or artistic expression.

Deep Neural Networks (DNNs) have had a profound impact on the field of multimedia recognition and synthesis, revolutionizing the way we approach and solve complex problems in this domain. With their ability to learn and extract intricate patterns and representations from large amounts of data, neural networks have significantly improved the accuracy and efficiency of multimedia recognition tasks. They have surpassed traditional hand-engineered approaches by automatically learning hierarchical features and capturing contextual dependencies, resulting in state-of-the-art performance in tasks such as image classification, object detection, speech recognition, and natural language processing. Moreover, neural networks have facilitated the development of sophisticated multimedia synthesis techniques by enabling the generation of highly realistic and novel media content. Generative models like GANs and variational autoencoders (VAEs) have demonstrated remarkable capabilities in image and video synthesis, speech synthesis, and even text generation. The advent of neural networks has propelled the field forward, unlocking new possibilities and pushing the boundaries of what can be achieved in multimedia recognition and synthesis.

Although neural networks have made remarkable contributions and achieved significant advancements in the field of multimedia recognition and synthesis, they are not without their limitations. Their main limitations can be categorized into three key challenges: vulnerability to adversarial examples, large data requirements, and substantial compute and resource demands. Despite their numerous strengths and successes, it is essential to acknowledge and address these limitations in order to fully harness the potential of neural networks in multimedia applications.

Challenge 1: Limited Robustness: DNNs are vulnerable to adversarial examples, which are carefully crafted inputs designed to deceive the network into making incorrect predictions. These examples contain imperceptible perturbations that can lead to significant changes in the network's output. This poses a critical concern for the reliability and robustness of neural networks, particularly in safety-critical applications. Developing effective defense mechanisms

to mitigate the impact of adversarial attacks is crucial to ensure the trustworthiness and security of neural network-based systems.

Challenge 2: Large Data Requirement: DNNs heavily rely on large amounts of labelled data for training, in order to achieve state-of-the-art performance on benchmark tasks. Gathering and annotating extensive datasets can be a resource-intensive and time-consuming process. Additionally, in domains where labeled data is scarce or expensive to obtain, the reliance on large-scale labeled datasets becomes a significant limitation.

Challenge 3: Compute and Resource Requirement: The compute and resource requirements of neural networks are substantial. Training DNNs with millions or even billions of parameters can be computationally expensive, requiring high-performance hardware and extensive computational resources. Deployment of these models on resource-constrained devices or within real-time systems can pose significant challenges due to their demanding nature. Developing efficient model architectures, optimizing algorithms, and exploring hardware accelerators are crucial for reducing the compute and resource requirements while maintaining or improving performance.

The research presented in this dissertation provides solutions to addresses the above challenges and aims to facilitate the practical deployment of deep learning models for multimedia processing. The subsequent sections detail my research contributions in tackling these three challenges.

1.1 Robustness of Deep Learning Models

Ensuring the robust and reliable deployment of DNNs has become crucial, particularly as they are increasingly utilized in safety-critical applications like autonomous vehicles, medical diagnosis, personal devices and security systems. To address the robustness challenges of deep learning models to adversarial inputs, in my research, I first assess the vulnerability of DNNs to adversarial attacks in different threat scenarios and data domains. Having exposed vulnerabilities

Figure 1.1. Research Overview: My research investigates robust and efficient deep learning techniques for multimedia generation and recognition.

and security threats to deep learning models, I then propose novel and robust frameworks to defend against adversarial attacks.

1.1.1 Vulnerabilities of Deep Learning Models to Adversarial Attacks

While past research has demonstrated that DNNs are vulnerable to adversarial examples, crafting such inputs often requires an iterative optimization approach thereby introducing significant latency. In this dissertation, I present my research on unveiling new real-time threats against DNNs that can more practically exploit vulnerabilities of these networks leading to incorrect outputs during inference. In Chapter 2, I describe two such real-time adversarial attack algorithms that I have developed to study the vulnerabilities of DNN-based face recognition and speech recognition systems. These attacks are based on input transformation functions that are either trained as a feed-forward neural network or a universal additive perturbation, such that they cause the victim model to mis-predict a given input. In Chapter 7, I describe my work *Adversarial Deepfakes* which extends real-time adversarial attacks to target DNN based Deepfake media detection systems. This renders such detection systems obsolete in safety-critical real-world environments, allowing synthetically-generated media to convincingly bypass detection and appear as authentic media on surveillance footage or media sharing platforms.

In Chapter 3, I describe my work that studies another related DNN vulnerability termed *Adversarial Reprogramming*. In adversarial reprogramming, the task of the attacker is to repurpose a given DNN for an alternate task using computationally inexpensive input transformation functions. In this setting, I first develop a method to repurpose sequence classification networks for alternate sequence classification tasks. Next, I consider a more challenging threat scenario of cross-domain adversarial reprogramming and demonstrate that image classification models can be effectively reprogrammed for text and sequence classification tasks using simple input transformation functions.

1.1.2 Defenses for Mitigating Adversarial Attacks

Having unveiled the above vulnerabilities of DNNs, my research develops two defense frameworks to guard DNNs against adversarial attacks in the speech and image domain. To mitigate adversarial attacks in the speech domain, I propose *WaveGuard*, the first formal defense framework for protecting automatic speech recognition models against adversarial inputs, even in challenging *adaptive attack* settings. Chapter 4 of this dissertation details the *WaveGuard* defense framework, which leverages the observation that model predictions for adversarial inputs are unstable while those for benign inputs are robust to small changes in the input. Therefore, *WaveGuard* applies input transformation functions to audio inputs and analyzes the transcriptions of original and transformed audio to differentiate between adversarial and benign inputs. *WaveGuard* achieves state-of-the-art performance in detecting audio adversarial samples, operates in real-time, and boasts low computational overhead, making it readily deployable to safeguard real-world automatic speech recognition models.

To mitigate adversarial attacks in the image and video domain, I propose *FastStamp* which leverages secure and proactive watermarking to preserve the authenticity of real media. Chapter 8 of this dissertation details the *FastStamp* encoder-decoder network which is trained end-to-end to embed a secure and verifiable secret message into images and videos at the time of their capture from a device and establish media authenticity. The secret watermark is designed to

be recoverable when benign transformations such as compression, color and contrast adjustments are applied. However, if a facial manipulation (e.g. DeepFake) is applied to the image/video, the watermark should break. By ensuring this selective fragility of the watermark, we can reliably prove the authenticity of real and original media. *FastStamp* stands out for its combination of parameter efficiency and real-time performance, surpassing other neural image watermarking models in both robust and semi-fragile watermarking, and achieving state-of-the-art results.

1.2 Compute Efficient Design for Neural Media Synthesis

DNNs are computationally expensive and are typically represented by millions of trainable parameters. The recent advances in deep learning are suggesting a shift towards even larger neural network architectures which further increases the size and computation requirement for training and inference of DNNs. Typically these models are deployed on high performance GPUs, which makes them suitable for only cloud-hosted servers with a GPU grid. To achieve more efficient deployment of deep learning models in resource constrained settings such as IoT, personal and edge devices, my research looks into acceleration of neural networks using hardware-software codesign. I propose solutions to optimize deep learning execution on diverse hardware platforms, enabling more hardware-friendly DNN model design and accelerating inference speed by leveraging customizable and reprogrammable platforms.

This dissertation presents two frameworks that accelerate DNN model inference on programmable hardware platforms such as FPGAs, which allows for more efficient neural network performance on smaller edge devices and addresses the practical challenge of real-world neural network deployment. In Chapter 5.3.3, I describe FastWave, a general purpose accelerator for autoregressive convolutional neural networks (CNNS). Autoregressive CNNs are commonly used for sequence generation tasks like speech and text synthesis. While such networks can be trained efficiently using parallel computation, they have high latency during inference due to their autoregressive nature. FastWave addresses the inference challenges for autoregressive

CNNs using hardware-software codesign. We implement an efficient inference algorithm on FPGA and achieve 66 times faster generation speed compared to CPU implementation and 11 times faster generation speed than GPU implementation for a popular DNN-based autoregressive speech synthesis model.

In Chapter 8, I describe *FastStamp*, an acceleration framework for CNN-based U-Net models. U-Net models are commonly used for image synthesis tasks such as text-to-image synthesis, image-to-image translation, image watermarking and steganography. In FastStamp, we propose a parameter-efficient U-Net architecture that matches the performance of much larger U-Net models for the task of image watermarking, while being significantly faster and resource efficient. We then design an FPGA-based accelerator framework to further improve the model throughput and power consumption by leveraging data parallelism and customized computation paths. Our best design enables real-time image watermarking and achieves 68 times faster inference as compared to GPU implementations of prior DNN based watermarking encoder, while consuming significantly less power.

1.3 Data Efficient Training for Neural Media Synthesis

Aside from being computationally expensive, DNNs also typically require large amounts of training data. Therefore, learning to solve tasks which have limited training data is another common challenge for deep learning researchers. To address this issue of data efficiency, I have proposed training solutions that leverage self-supervised learning (SSL) for solving tasks with limited training data. Self-supervised learning methods aim to learn meaningful representations from large amounts of unlabelled data in various domains including text, speech and vision. While previous research has shown that SSL representations can be useful for downstream recognition tasks, the efficacy of such representations for generation tasks had not been explored. In my research, I propose methods that utilize SSL representations for controllable synthesis tasks in the speech domain. Particularly, I propose methods for generating natural and expressive

speech, that enable creating a digital clone of a new voice in data-limited settings and using only a few reference audio samples. The techniques I develop using SSL, enable DNNs to learn meaningful representations of speech in a language-agnostic manner from readily available unlabeled audios. These DNNs can be fine-tuned on smaller amount of data from unseen target languages, to generate contextualized representations which are highly effective for synthesizing speech in the respective target language.

Chapter 6 of this dissertation describes two speech synthesis frameworks which I have developed that can synthesize natural-sounding speech for a new speaker using just a few seconds of training data of the given speaker. First, I propose a data-efficient text-to-speech (TTS) synthesis framework to generate natural-sounding and expressive speech for a new speaker using zero-shot and few-shot learning methods. To achieve the goal of data-efficient TTS synthesis, we condition a base text-to-speech synthesizer with speaker embeddings derived from a speaker verification model and heuristically derived pitch information. Our proposed method significantly outperforms baseline methods on metrics evaluating speech naturalness and expressivity for a new speaker using just a few seconds of audio of the target speaker.

In addition to my work on TTS synthesis, I also develop methods for speech-to-speech voice conversion, where the goal is to convert the voice of a given speech utterance to match the vocal qualities of a target speaker. To this end, I first propose a multi-task finetuning strategy to derive speech representations describing content and speaker characteristics from a pre-trained SSL model. To disentangle content and speaker representations, we propose a training strategy based on Siamese networks that encourages similarity between the content representations of the original and pitch-shifted audio. Next, we develop a synthesis model with pitch and duration predictors that can effectively reconstruct the speech signal from its decomposed representation. Our framework allows controllable and speaker-adaptive synthesis to perform zero-shot any-to-any voice conversion.

Finally, I propose a data-efficient speech synthesis system that can be trained in a completely text-free manner using imperfectly disentangled SSL representations. By removing

the dependence on text, we design universal language-independent models that do not require transcribed speech files for training and is particularly useful for low-resource languages where we have limited audio data and often without parallel text transcripts. To allow explicit control over the content and speaker characteristics during synthesis, we develop a training strategy to iteratively improve the synthesizer, by challenging its capabilities using self-synthesized training examples. That is, during training, we utilize the current state of the synthesizer to generate voice-converted variants of a given utterance to be used as inputs for the reconstruction task. We demonstrate that incorporating such self-synthesized training examples improves model performance as compared to a model trained solely on heuristically perturbed inputs. Our framework is trained without any text and is applicable to a range of tasks such as zero-shot voice conversion, voice conversion across different languages, and controllable speech synthesis with pitch and pace modifications. Our framework achieves state-of-the-art results in voice conversion for both seen and unseen speakers, evaluated on naturalness, speaker similarity, and intelligibility metrics. Such expressive voice cloning systems that operate in real-time are useful in empowering individuals who have lost their ability to speak. In multilingual settings where data is generally scarce, my approach enables DNNs to generate more accurate and contextually relevant speech. Additionally, my work can aid in education and schooling systems to empower individuals with reading and visual disabilities.

Part I

Robust Deep Learning

Chapter 2

Vulnerabilities of DL to Adversarial Attacks

Recent advances in adversarial Deep Learning (DL) have opened up a largely unexplored surface for malicious attacks jeopardizing the integrity of autonomous DL systems. Machine learning models like image/video classifiers, face recognition systems and speech recognition systems are vulnerable to adversarial attacks. Adversarial attacks refer to carefully crafted perturbations applied to input data that are imperceptible to human observers but can lead to erroneous predictions or misclassification by the machine learning model. The implications of this threat are significant, particularly in real-world settings where machine learning models are deployed.

While DNNs have been shown to be vulnerable to adversarial examples, crafting such adversarial inputs often requires an iterative optimization approach assuming white-box access to the model. In real world settings, the weights of the victim DNN models can be easily kept private thereby preventing a white-box attack. Moreover, the added latency of the adversarial example generator can be too high to pose any practical threat. For my research on evaluating DNN vulnerabilities, I consider a more practical attack scenario and seek to answer the following question: Can adversarial examples be generated in real-time and bypass unseen models in a black-box setting?

In this chapter, I first provide a brief background of adversarial example generation

using first-order gradients. Next, I describe two attack frameworks that I developed to generate adversarial examples in real-time. The first framework trains an Adversarial Transformation Network (ATN) to generate adversarial inputs given a benign input. We train the ATN to target face-recognition models by formulating training objectives that focus on the embedding space and optimizing metrics that degrade the identification and verification performance of such models. Once trained the ATN can generate highly transferable adversarial inputs in real-time. The second attack framework proposes an algorithm to generate universal adversarial perturbations for speech recognition systems. The algorithm can generate a single quasi-imperceptible perturbation that can be added to any audio signal to cause a victim speech recognition model to mistranscribe an audio signal.

2.1 Adversarial Examples

Adversarial examples are intentionally designed inputs to a machine learning (ML) model that cause the model to make a mistake [8]. Prior work has shown a series of first-order gradient-based attacks to be fairly effective in fooling DNN based models in image [9, 10, 11, 12, 13, 14, 15], audio [1, 16, 2] and text [17, 18, 19] domains. The objective of such adversarial attacks is to find a good trajectory that (i) maximally changes the value of the model's output and (ii) pushes the sample towards a low-density region as indicated by the magenta arrow in Figure 2.1. This is equivalent to the ML model's first-order gradients with respect to input features. Aside from utilizing only first-order gradients, prior work [20] shows that the adversary can additionally utilize the second order gradients and take iterative steps to craft the adversarial sample as shown by black arrows in Figure 2.1(b). Such an objective can be used in adaptive attack [20] settings where an attacker has some knowledge of a defense existing to enhance robustness of the ML model.

Adversarial attacks can be categorized as either black-box or white-box, depending on the underlying threat model. In the white-box scenario, the attacker has complete access to

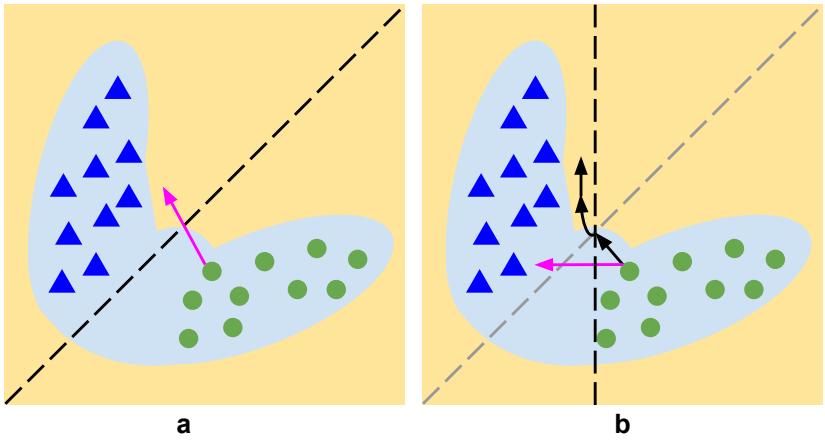


Figure 2.1. (a) First order attack achieved by following the gradient direction (magenta arrow) across the boundary into low density region. (b) Multi-step second-order attacks (black arrows) can still generate successful adversarial samples against robust models.

the victim's DNN, including its model architecture and parameters. This allows them to utilize gradient-based optimization techniques in order to discover the perturbation. In the black-box attack scenario, the adversary does not have access to the internals of the DNN. As such, proposed attacks use black-box optimization algorithms [21, 22] or surrogate models [23, 24, 2] to find an effective perturbation.

2.2 Vulnerabilities of DL based Face Recognition

Face recognition and verification systems are widely used for identity authentication in government surveillance, military applications, public security settings such as airports, hotels, banks as well as smartphones to unlock applications. Over recent years, Convolutional Neural Networks (CNNs) have achieved state-of-the-art results on several face recognition and verification benchmarks outperforming traditional computer vision algorithms that rely on hand engineered features. With the widespread adoption of face recognition models in surveillance and other security sensitive applications, careful vulnerability analysis is imperative to ensure their safe deployment.

Several works have shown that deep neural networks (DNNs) are vulnerable to adversarial

examples, causing the model to make an incorrect prediction with higher confidence [20, 13, 25, 26, 9]. Particularly, past attacks [27, 28] on face recognition systems have garnered immense media attention [29, 30] by utilizing projected gradient descent (PGD) [31] based approaches to achieve high fooling success rates. However, designing such adversarial examples requires the adversary to solve an optimization problem for each input. This makes the attack impractical in real-time since the adversary would need to re-solve the data-dependent optimization problem from scratch for every new input. The aforementioned methods for generating adversarial examples may cause a timing bottleneck that could hinder real-time image uploads, making it impractical to deploy such attacks. This bottleneck becomes even more pronounced when dealing with videos, where adversarial examples need to be generated for multiple frames per second. For example, during surveillance real-time face recognition models operate on live video streams from security cameras or webcam interfaces. In order to expose any real-time security vulnerabilities in such systems, it is necessary to generate an adversarial video stream in real-time as well.

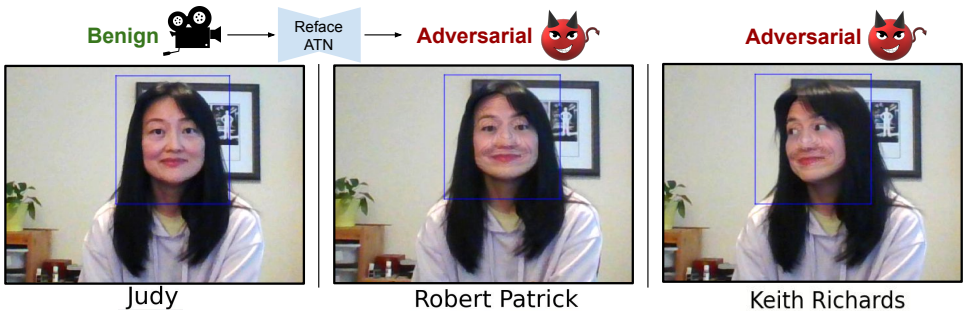


Figure 2.2. Real-time ReFace attack on a face recognition model operating on a live video stream. ReFace uses an Adversarial Transformation Network to inject adversarial perturbations into the video frames causing the face recognition model to mis-predict the identity of the subject in the video.

To generate adversarial attacks against classification systems in real-time, some past works, such as Adversarial Transformation Networks (ATNs) [32], have attempted to learn a perturbation function with a neural network. ATNs are encoder-decoder neural networks that are trained to generate an adversarial image directly from an input image without having to

perform multiple forward-backward passes on the victim classification model during inference, thereby making the attack possible in real time. However, ATNs have only been explored for classification tasks. The training objective studied thus far for an ATN is to push the classifier's output outside the decision boundary of the correct class. Unlike a classification model, where model outputs are class probabilities, the output space of a typical face recognition system is an embedding vector. A face recognition system is trained to cluster the embeddings of the same identity together in the embedding space while ensuring they are well separated from the embeddings of other identities. Therefore when attacking such a setup, the attack objective requires the adversary to target the embedding space rather than the decision boundaries of the classifier.

To perform attacks on face recognition models, we first develop training objectives that target the embedding space of face recognition models and optimize metrics that degrade the identification and verification performance of such models. To minimize perceptibility of our perturbations, we incorporate Learned Perceptual Image Patch Similarity L_{pips} perceptual loss [33] in addition to the L_∞ constraint during training. Next, to perform real-time attacks, we design a new ATN based on the U-net [34] architecture, since U-nets have been notably effective in many prior image-to-image translation tasks [35, 36]. We find that while a U-net based ATN can generate real-time adversarial examples, the attack performance falls short as compared to per-image gradient based attacks such as PGD [31] at the same magnitude of adversarial perturbation. This is because gradient-based attacks generate highly tailored adversarial examples that are optimized on a single image. We address the performance gap between ATN and PGD attacks through neural architectural improvements to our ATN model which we describe in Section 2.2.3.

Having bridged the gap with gradient based attacks on seen victim models, we evaluate the transferability of our adversarial samples to unseen models. Since ATNs are trained on a diverse set of images, we find that perturbations generated from an ATN are more transferable to unseen architectures as compared to per-input PGD attack, while being much faster to compute.

To further improve our attack transferability, we adapt our ATN training framework to target an ensemble of face recognition models with various backbone architectures. Our best ATN attacks on unseen models successfully reduce the performance of face recognition models to the level of random guessing or worse. We present a demo video of our attack in real-time on our project webpage [1] with sample images presented in Figure 2.2. Finally, we demonstrate our attack effectiveness against cloud-hosted face recognition APIs in a complete black-box setting. The technical contributions of our work are as follows:

- We propose a real-time attack framework to study the robustness of face recognition systems and demonstrate that our proposed ATN can synthesize adversarial examples several orders of magnitude faster than existing attacks on face recognition systems while achieving comparable attack success metrics as past works. To the best of our knowledge this is the first real-time attack on face recognition systems, in contrast to previous works which perform gradient based attacks or study real-time attack only in the classification domain.

- We bridge the performance gap between real-time ATN attacks and PGD attacks by developing a Residual U-net architecture that allows us to effectively increase the capacity of the ATN (Section 2.2.3). Our ResU-Net ATN approaches PGD performance in white-box attacks and outperforms PGD on black-box transfer attacks.

- We develop and release a benchmarking library for face recognition models (Section 2.2.4), implemented in the PyTorch framework. This allows us to evaluate our attacks on diverse set of architectures and loss functions. This library may be used to develop more robust face recognition models and to provide benchmarks of models' performance in an adversarial setting.

- We demonstrate the effectiveness of our real-time attacks on commercial face recognition

[1] Demo video: https://refaceattack.github.io/

services such as Amazon Face Rekognition and Microsoft Azure Face. Our attacks reduce face identification accuracy from 82% to 16.4% for AWS SearchFaces and face verification accuracy from 91% to 50.1% for Microsoft Azure.

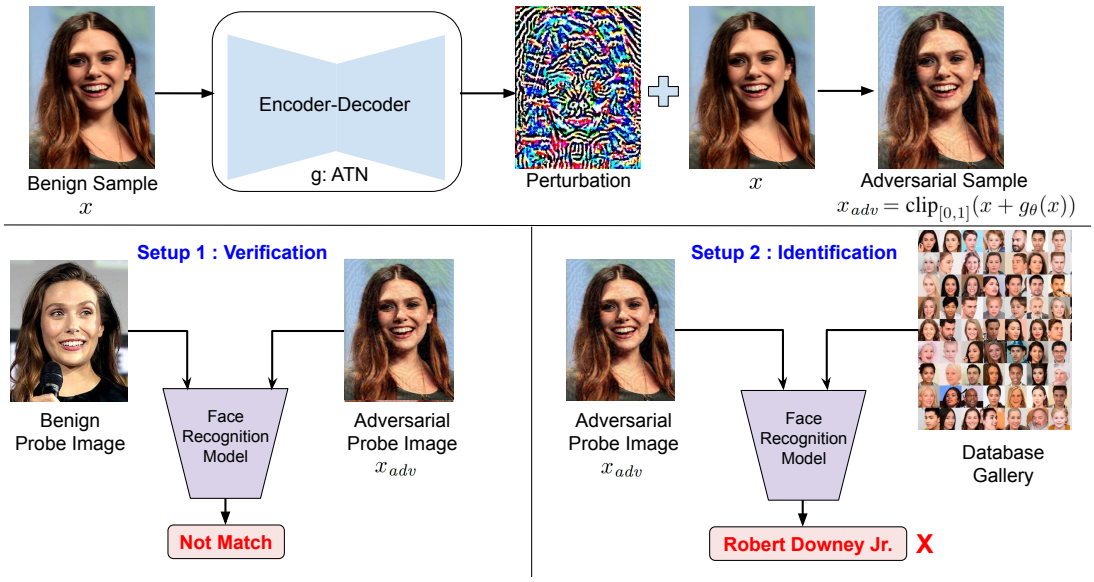


Figure 2.3. Overview of ReFace adversarial perturbation generator (top) and attack application on face verification and identification systems (bottom).

2.2.1 ReFace: Adversarial Transformation Networks for Real-time Attacks on Face Recognition Systems

In this section, I describe our proposed approach: ReFace [37], a real-time, highly-transferable attack framework on face recognition models leveraging Adversarial Transformation Networks (ATNs). Past attacks on face recognition models require the adversary to solve an input-dependent optimization problem using gradient descent making the attack impractical in real-time. Such adversarial examples are also tightly coupled to the victim model and are not as successful in transferring to different models. We find that the white-box attack success rate of a pure U-Net ATN falls substantially short of gradient-based attacks like PGD on large face recognition datasets. We therefore develop a new architecture for ATNs that closes this gap while maintaining a 10000$\times$ speedup over PGD. Furthermore, we find that at a given perturbation

magnitude, our ATN adversarial perturbations are more effective in transferring to new face recognition models than PGD. We demonstrate that our attacks transfer effectively to models with different architectures, loss functions, and training procedures. ReFace attacks can successfully deceive commercial face recognition services via transfer attack and reduce face identification accuracy from 82% to 16.4% for AWS SearchFaces API and Azure face verification accuracy from 91% to 50.1%.

2.2.2 Background and Related Work

Adversarial Examples for Real-time Attacks

Prior work on attacks have demonstrated that adversarial examples can circumvent state-of-the-art image classification models while remaining indistinguishable from benign images for humans [13, 25, 31, 38, 39, 9, 15]. However many of these works are gradient based attacks, which cannot be performed in real-time. To address this limitation, the authors of UAPs [40] demonstrated that there exist universal *input-agnostic* perturbations which when added to any image will cause the image to be misclassified by a victim network. The existence of such perturbations poses a threat to machine learning models in practical settings since the adversary may simply add the same pre-computed perturbation to a new image and cause misclassification in real-time. Also addressing the real-time challenge, the authors of [32] designed Adversarial Transformation Networks (ATNs) that follow an encoder-decoder architecture and output an adversarial perturbation for each input image, without having to compute gradients from the victim classification model during inference [32]. Unlike UAPs, ATNs generate input-specific perturbations. However these ATN attacks are specific to image classification tasks and cannot be directly used to attack face recognition models that use task-specific model and loss functions as opposed to the standard cross-entropy loss used by classifiers.

Facial Recognition Systems

Unlike typical classification algorithms, facial recognition systems do not have a fixed set of classes. Instead, a face recognition system must establish a person's identity and can operate in two different modes 1) *face verification* or 2) *face identification*. A verification system establishes whether or not a person is who they say they are (i.e., the person claims an identity and the system tries to prove whether or not that claim is true). On the other hand, an identification system attempts to establish a person's identity from scratch i.e. the system tries to associate a person with an identity from a set of identities in the system's database. Both problems are generally solved with metric learning [41, 42, 43]. Recognition algorithms learn an embedding of face images in which the distance between embedding vectors indicates whether or not the vectors came from the same identity. Given this learned embedding function, face verification amounts to applying a threshold to the distance between two embedding vectors, and face identification amounts to ranking images by closeness to the query image. While most state-of-the-art facial recognition algorithms use this general approach, they use different strategies to learn the metric. DeepID [42] is a CNN based network which follows a training process where each unique training identity is treated as a separate class and the network is trained using softmax loss. The second-to-last layer then serves as the embedding, which is effective on unseen identities. SphereFace [43] builds on DeepID by replacing traditional softmax loss with angular softmax loss. Angular softmax ensures that Euclidean distance in the embedding space produces optimal decision boundaries between identities. Similarly, ArcFace [44] loss is also adopted by recent state-of-the-art face recognition models, and these models are used as the main testbench in recent literature [28, 45] to study the effectiveness of adversarial attacks. In our work, we study the effectiveness of adversarial attacks using ATNs on face recognition models trained with SphereFace, DeepID and ArcFace loss, in addition to black-box models trained with unseen loss functions.

Adversarial Attacks on Face Recognition

While several works have studied adversarial attacks on face recognition models, these are relatively fewer in literature as compared to image classification attacks. Some prior works include physical adversarial examples in the form of objects such as glasses [46, 47] and hats [45] that can fool models to make wrong prediction on the person wearing the object. The authors of [48] attempt to target face "classification" networks which operate differently from face recognition networks that perform face verification and identification. Prior works such as [49, 50, 28, 27] generate adversarial examples for face recognition systems by optimizing the perturbation for each image using white-box access to a face-recognition model. One such attack Face-Off [27] demonstrates that it is possible to generate adversarial faces by optimizing in the model embedding space using PGD [31] and CW [13] attack, however reports an attack run-time from 6 seconds to 373 seconds per image while using 2 GPUs. The authors of Face-Off evaluate the strongest attack algorithms to perform adversarial attack on Face Recognition models. They report that PGD algorithm used in their experiments is stronger than CW attack. This finding is corroborated in other papers [51]. Another gradient based attack Lowkey [52] generates image-specific adversarial samples for face recognition models and demonstrates their transferability to public cloud provider APIs, however reports an attack run-time of 32 seconds per image. To generate adversarial examples in black-box settings, the authors of [49] utilize an evolutionary optimization technique, but require at least 1,000 queries to the target face recognition system before a realistic adversarial face can be synthesized. Similarly, the more recently proposed black-box attack by [53] on face recognition systems requires at least 1700 queries to generate successful attacks. The time for generating adversarial examples using the above techniques can potentially bottleneck real-time image upload making the attacks impractical for deployment. The timing bottleneck gets even more significant for videos in which we need to generate adversarial examples for several frames per second. In contrast, we propose a framework to adversarially modify query images in real-time, such that the performance of

face recognition models deteriorate significantly in both white-box and transfer based black-box attack settings. Unlike past works on face recognition, our proposed approach enables real-time attacks on video streams which we demonstrate via successful attacks over web-cam interfaces (demo video linked in the second page).

2.2.3 ReFace Methodology

Victim Models

A typical face recognition pipeline first detects and crops faces. Next, they map each cropped image x to an embedding vector y using $F : x \mapsto y$. Typically, such models are trained on a dataset of facial images and identity labels, with the objective of clustering embeddings of images from the same identity together and ensuring separability between embeddings of images from different identities. State-of-the-art face recognition models are commonly trained with objectives that effectively optimize a cosine distance metric e.g. SphereFace [41], DeepID [42] or ArcFace [44] loss. During inference, a face recognition model can be used for one of the following goals:

1. Verification - A face recognition model can be used to verify whether two images belong to the same person or not. In this setting, the model compares the embeddings of two probe images and reports a match if the distance between the embeddings of the two models is below a certain threshold.

2. Identification - In this setting, the face recognition system tries to associate a person with an identity from a set of identities in *gallery images* stored in the system's database. When presented with a *probe image*, the system compares the embedding of the probe image with the gallery images to find the closest matching neighbour in the gallery and determine the identity of the probe image.

In our work, we attack CNN-based face recognition models in real time and assess the success rate against both of the above goals. To simplify experimentation, we do not include the detection and cropping step in our attacks pipeline. Instead we use the pre-cropped images

provided by standard datasets.

Attack Objective

Threat Model: Given benign facial input images, our goal is to adversarially modify the inputs in real-time, such that the modified inputs cause the face recognition model to mispredict the embedding vectors, thereby degrading the verification and identification performance of the face recognition model. In order to adversarially modify each image, we design a perturbation generator that operates in real-time to add a quasi-imperceptible adversarial perturbation to the given input image. When attacking a *face verification* system, we adversarially perturb one of the two probe images. In this attack setting, our goal is to reduce the true recall rate of the verification system (performance on positive pairs). When attacking a *face identification* system, we assume the probe images have been adversarially perturbed while the dataset of gallery images is benign. In this attack setting, our goal is to lower the recognition rate of the face identification system.

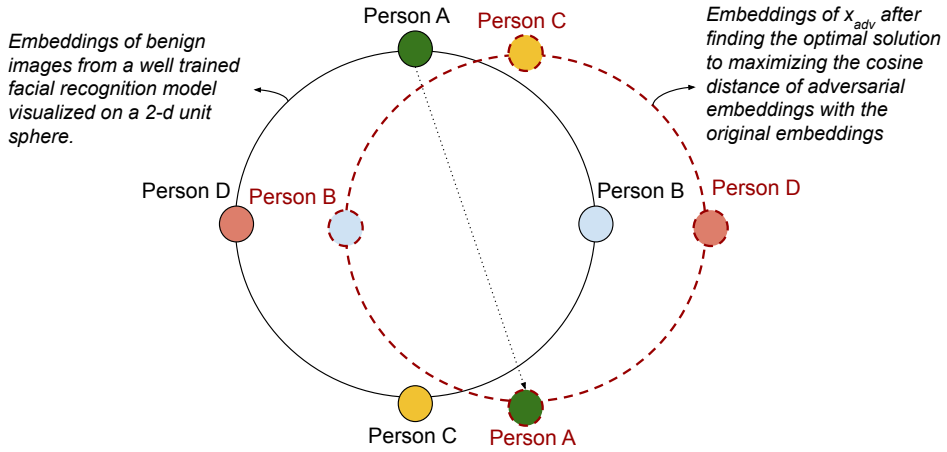


Figure 2.4. Visualizing the optimum solution to our attack objective: Our attack objective pushes the originally predicted embedding vectors to the opposite end of the unit sphere thereby hampering the performance of the face-recognition model.

Problem Formulation: To achieve the above objectives, we train a perturbation generator g_θ, parameterized by θ, which takes as input an image x and generates an adversarial perturbation $g_\theta(x)$ that can be added to x to synthesize an adversarial example x_{adv}. The optimization objective of g_θ is to maximize the cosine distance the embeddings of the adversarial and original image,

while constraining the amount of the perturbation added to the image. This is different from the objective for fooling classification systems, where the commonly used objective for untargeted attacks is to maximize the cross-entropy loss with the correct label. L_p norm is a widely used distance metric for measuring the distortion between the original and adversarial inputs. Prior works [25] recommend constraining the maximum distortion of any individual pixel using the L_∞ norm. To further reduce the perceptibility of the perturbation we incorporate L_{pips} [33] loss during training. L_{pips} distance measures the visual similarity between two images by comparing the embeddings from a pre-trained CNN model.

Mathematically, our attack objective is as follows:

$$\begin{gathered} \forall_{x \in X} \; maximize \; [d(F(x_{adv}), F(x)) - \lambda L_{pips}(x_{adv}, x)] \\ where \; x_{adv} = \text{clip}_{[0,1]}(x + g_\theta(x)) \\ s.t \; ||g_\theta(x)||_\infty < \varepsilon \end{gathered} \tag{2.1}$$

where $d(F(x_{adv}), F(x))$ is the cosine distance between embeddings of the adversarial and original image and λ is the loss coeffecient for L_{pips}. In Figure 2.4 we illustrate how an optimum solution to the above problem of maximizing the cosine distance completely degrades the performance of a face recognition model. A visualization of such embedding clusters for a hypothetical case of four individuals on a 2-D unit sphere is shown on the left in Figure 2.4. If we were to find the optimum solution to our attack objective in an unbounded attack setting, the embeddings clusters for adversarial images will move to the opposite end of the unit sphere (to maximize the cosine distance). This clearly results in hampering both verification and identification performance of the model since the embeddings of benign and adversarial examples are completely rotated to the opposite ends in the unit sphere.

In our work, we model g_θ as a neural encoder-decoder architecture called an Adversarial Transformation Network (ATN) (Section 2.2.3).

ATN: Adversarial Transformation Network

An ATN is a neural network trained to produce adversarial images, with the form $g_\theta : \mathscr{X} \to \mathscr{X}$. Since the network only needs one forward pass to compute the perturbation, it is less expensive than an iterative gradient-based optimization procedure. We obtain an adversarial image from a benign image using the neural network N_θ as follows:

$$g_\theta(x) = \varepsilon \cdot \tanh(N_\theta(x)) \tag{2.2}$$

With this formulation we enforce the constraint $||x_{adv} - x||_\infty < \varepsilon$ since the output of tanh is bounded between $[-1, 1]$.

Algorithm 1. Ensemble attack training procedure

Inputs: Victim Models $\mathbb{F} = F_1, \ldots F_n$, image dataset X
Output: Perturbation engine (g_θ) parameters θ
HyperParams: Learning rate α, L_∞ bound ε, L_{pips} loss coefficient λ
Initialize ATN: N_θ
Batch training images: $X_{batched} \leftarrow Batch(X)$
for *epoch* in 0 to N_{epochs} **do**
  for x in $X_{batched}$ **do**
    $x_{adv} \leftarrow \text{clip}_{[0,1]}(x + \varepsilon \cdot \tanh(N_\theta(x)))$
    $loss \leftarrow 0$
    for F_i in F **do**
      $loss \leftarrow loss + (-d(F_i(x), F_i(x_{adv})))$
    $loss \leftarrow loss / len(F)$
    $loss \leftarrow loss + \lambda L_{pips}(x_{adv}, x)$
    $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta(loss)$
return θ

We train the ATN to generate adversarial examples using the procedure described in Algorithm 1. Our ATN can be trained to target one or more face recognition models in the model set $\mathbb{F}$. During each mini-batch iteration, we generate a batch of adversarial images from the ATN and compute the cosine distance between embeddings of benign and adversarial images. We accumulate the loss for all models in the set $\mathbb{F}$ and can optionally add the L_{pips} loss to minimize the perceptibility of the adversarial perturbation. Finally, we backpropogate through all models

in the set $\mathbb{F}$ to compute the gradient of the loss with respect to the parameters θ of the ATN and update the ATN parameters using mini-batch gradient descent with a learning rate α. Targeting an ensemble of face recognition models during training can result in more transferable adversarial attacks. In our experiments, we verify this hypothesis and demonstrate that ATNs trained to target an ensemble of models result in better transferability to unseen models.

The search for an effective ATN architecture

The input and output domains of the ATN have the same spatial dimension, so a logical choice for the network architecture is a U-net [34]. U-nets are commonly used for several image-to-image translation problems. The architecture consists of several down-sampling layers followed by an equal number of up-sampling layers. The feature maps from the down-sampling layers have skip connections that are concatenated to the up-sampling layers with matching resolution. Previous work with ATNs used different architectures, but in our preliminary experiments, we found that U-nets were far more effective than alternate architectures at the same level of perturbation.

However, we still found that there was a large gap between a U-net based ATN and an iterative gradient-based white-box attack, *even on the training data*. This is illustrated in Figure 2.6 in our experiments comparing PGD-30 (i.e., 30 iterations of PGD) to the U-net ATN. From the universal approximation theorem [54], a neural network could in principle represent a close approximation of the PGD-30 function. As this neural network would have lower training loss than the U-net ATN, it appears that this architecture is *underfitting*. We therefore explored ways to add capacity to the ATN. We found that adding layers and making the layers wider both led to small gains in performance with diminishing returns.

One feature of the U-net is that every layer changes the spatial resolution. The deeper layers of the U-net necessarily operate at very low spatial resolutions. Intuitively, it may be useful to be able to express complex hierarchical functions at higher resolutions. We developed a new Residual U-net architecture, illustrated in Figure 2.5, that replaces individual convolution

and transpose convolution layers in a U-net with groups of residual blocks. We use 2-layer pre-activation blocks with ReLU and batch normalization. One skip connection per downsample is carried over to the decoder, which allows arbitrary numbers of residual blocks at each step. We denote the number of blocks in each group as a vectors **E** and **D** for the encoder and decoder, respectively. While similar architectures have been proposed in the past [55, 56], they are not widely used and have not been used in adversarial perturbation literature.

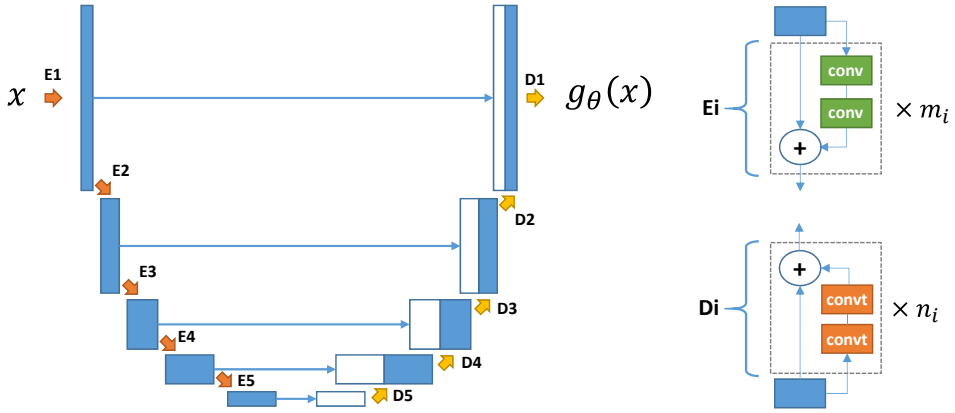


Figure 2.5. Residual U-net architecture: We replace the strided convolutions and transposed convolutions in the U-Net architecture with residual blocks. Each residual block contains multiple convolutions (in the encoder) or transposed convolution (in the decoder) layers.

We found that adding layers in this architecture was considerably more effective than in the pure U-net ATN. We performed an architecture search to find an effective balance between computational cost and attack effectiveness. The optimal architecture from this process had five downsampling steps with $\mathbf{E} = [1, 1, 2, 3, 5]$ and $\mathbf{D} = [1, 1, 1, 1, 1]$. We use a base width of 64 channels and double the width at each downsample step except for the last. Using this ResU-net architecture, the ATN approached the performance of PGD-30 (plotted in Figure 2.6a) with roughly 10,000$\times$ less run-time. We refer the readers to the code-base included in our supplementary material for the precise model implementation.

2.2.4 Experiments

We perform experiments to evaluate our proposed attack in both white-box and transfer-attack settings. We perform the attack at different levels of adversarial perturbations and study how factors such as victim model architecture, loss functions and random initialization affect the success rate of the attacks. We also perform a timing analysis and demonstrate that our attacks can be performed in real-time and achieve a high success rate in both white-box and transfer attack settings. We compare our attack performance against state-of-the-art adversarial attacks on face recognition models such as Face-Off [27] and Fawkes [28] that utilize the PGD attack algorithm. Finally, we perform our transfer attack on black-box public APIs (Amazon Rekognition and Microsoft Azure) and demonstrate that our attacks can significantly reduce both the verification and identification performance of such APIs.

Dataset and Models

We develop a benchmarking framework in PyTorch to evaluate both white-box and transfer attack performance of adversarial examples generated using our ATNs. Our experiments are designed to examine how factors such as network architecture, training loss functions, and random initialization affect the transferability of attacks. We used two main CNN architectures for the face recognition models: pre-activation ResNet [57] and Inception-v4 [58]. Within these architectures, we varied the number of blocks leading to networks ranging from 22 to 118 layers which were trained with three different loss functions: DeepID [42], SphereFace [41] and ArcFace [44]. The dimension of the output embedding vector for all test-bed models is 512. The face recognition models are trained on the training partition of the VGGFace2 dataset [59]. We start with the standard crops provided by the dataset and perform random resized cropping for data augmentation during training. VGGFace2 dataset contains 3.31 million images across 9131 identities which is larger and more diverse as compared to other face recognition datasets such as FaceScrub [60] or UMDFaces [61]. We choose VGGFace2 dataset for training the test-bed face recognition models because models trained on larger and more diverse datasets are more

robust and generalize better to unseen images [59].

Table 2.1. Victim model sets used for conducting our attack evaluations. Experiments are conducted on both single and ensemble model sets. The verification and identification metrics are averages over the whole model set reported on the *clean* unperturbed VGGFace2 test set.

		Networks		*Verification*		*Identification*
Name	Architecture	# Models	Loss	V-AUC	V-Acc.	R1-Acc.
RN-SF-1	ResNet	1	SphereFace	0.99	95.2	84.4
RN-DID-1	ResNet	1	DeepID	0.98	93.3	78.0
IN-SF-1	InceptionNet	1	SphereFace	0.99	94.4	78.5
RN-AF-1	ResNet	1	ArcFace	0.98	92.8	89.0
RN-SF-6	ResNet	6	SphereFace	0.99	94.3	82.0
RN-DID-6	ResNet	6	DeepID	0.98	93.0	77.4
IN-SF-4	InceptionNet	4	SphereFace	0.99	94.4	78.9
RN-AF-4	ResNet	4	ArcFace	0.98	93.3	90.0

Table 2.1 presents the test-bed of face recognition models that we use for training our ATNs. The model sets comprise single and ensemble versions for each architecture, enabling us to evaluate the efficacy of ensemble attacks on unknown models. For ensemble models, the reported metrics are averaged over all individual models in the ensemble. For the training and testing of the ATNs, we utilize the VGGFace2 validation set that was not used in the training of the test-bed face recognition models. Specifically, we partitioned the validation set of VGGFace2 into two distinct subsets, each containing a comparable number of images and non-overlapping identities. The resulting subsets comprise a training subset of 84,953 images and a testing subset of 84,443 images, used for training and testing the ATN models.

Evaluation Metrics

We evaluate the performance of face recognition models on both verification and identification tasks with the metrics described below.

Face Verification Metrics: For each identity in the test set, we prepare all possible pairs of distinct images that have the same identity. To keep our problem balanced, we randomly sample an equal number of non-matching pairs. On the test set of VGGFace2, this creates a total of

Table 2.2. White-box and transfer attack results of ATN attack at $\varepsilon = 0.03$. A lower value for all three metrics indicates a more successful attack. The diagonal entries in each of the three tables represents a white-box attack while all other entries represent a transfer (black-box) attack.

			Single Defender Models				Ensemble Defender Models			
			RN-SF-1	RN-DID-1	IN-SF-1	RN-AF-1	RN-SF-6	RN-DID-6	IN-SF-4	RN-AF-4
Verification AUC	*No Attack*	Clean	0.99	0.98	0.99	0.98	0.99	0.98	0.99	0.98
	Single Model Attack	RN-SF-1	0.03	0.59	0.87	0.77	0.59	0.71	0.89	0.71
		RN-DID-1	0.58	0.04	0.88	0.72	0.64	0.55	0.88	0.68
		IN-SF-1	0.75	0.77	0.03	0.82	0.77	0.79	0.54	0.80
		RN-AF-1	0.60	0.62	0.88	0.04	0.68	0.72	0.90	0.60
	Ensemble Model Attack	RN-SF-6	0.06	0.12	0.35	0.17	0.07	0.14	0.37	0.19
		RN-DID-6	0.13	0.07	0.45	0.18	0.10	0.08	0.44	0.18
		IN-SF-4	0.37	0.43	0.05	0.45	0.31	0.43	0.05	0.46
		RN-AF-4	0.15	0.14	0.34	0.06	0.12	0.15	0.38	0.08
Verification Accuracy	*No Attack*	Clean	95.20%	93.30%	94.40%	92.80%	94.30%	93.00%	94.40%	93.30%
	Single Model Attack	RN-SF-1	48.78%	59.29%	67.90%	63.43%	64.34%	65.47%	82.70%	68.20%
		RN-DID-1	62.54%	49.45%	82.28%	64.12%	64.40%	60.63%	81.58%	69.54%
		IN-SF-1	71.21%	68.91%	48.43%	72.12%	70.90%	69.98%	63.64%	71.41%
		RN-AF-1	61.34%	60.12%	68.10%	49.21%	67.34%	66.62%	83.45%	64.57%
	Ensemble Model Attack	RN-SF-6	48.37%	48.47%	53.44%	49.89%	48.17%	48.32%	53.53%	48.87%
		RN-DID-6	48.85%	48.67%	57.20%	50.23%	48.13%	48.57%	56.44%	49.35%
		IN-SF-4	50.79%	50.98%	47.93%	51.21%	49.46%	50.86%	47.99%	51.21%
		RN-AF-4	49.53%	50.12%	58.12%	48.73%	49.62%	48.92%	57.41%	48.45%
Rank-1 Accuracy	*No Attack*	Clean	84.40%	78.00%	78.50%	89.00%	82.00%	77.40%	78.90%	90.00%
	Single Model Attack	RN-SF-1	0.05%	12.60%	43.45%	18.32%	17.96%	19.36%	43.96%	17.56%
		RN-DID-1	16.36%	0.02%	43.10%	19.87%	17.41%	12.57%	41.01%	18.76%
		IN-SF-1	30.31%	26.14%	0.02%	32.88%	26.75%	25.59%	16.36%	27.65%
		RN-AF-1	17.21%	13.10%	46.37%	0.03%	18.45%	21.23%	45.46%	15.21%
	Ensemble Model Attack	RN-SF-6	0.10%	0.22%	5.31%	0.98%	0.09%	0.22%	5.05%	0.43%
		RN-DID-6	0.72%	0.05%	8.59%	1.03%	0.27%	0.05%	7.62%	0.45%
		IN-SF-4	2.20%	2.20%	0.01%	3.10%	1.03%	2.03%	0.01%	2.41%
		RN-AF-4	0.81%	0.34%	6.43%	0.05%	0.31%	0.27%	8.12%	0.04%

917,692 verification tests where half have a pair of images with matching identities (positive labels) and half have different identities (negative labels). Given this binary classification problem, we report the following metrics:

1. Verification AUC (V-AUC): We use the cosine distance between the embedding of the two images along with the verification label to generate a Receiver Operating Characteristic curve (ROC). Our metric is the standard area under the ROC curve (AUC).

2. Verification Accuracy (V-Acc.): To determine the accuracy, we need a threshold for the cosine distance, across which the example is labelled positive or negative. For each model, we set this

to *equal error rate* threshold of the model on the (clean) VGGFace2 validation set.

Face Identification Metric: We use the VGGFace2 test set to create a random gallery with 100 unique identities. For each of these 100 identities, we select a probe image with one of the identities appearing in the gallery and compute its distance to each image in the gallery. This creates 100 identification tests. We repeat this gallery test on 1000 random galleries to create a total of 100,000 identification tests. When evaluating attacks, we perturb the probe image and leave the gallery unmodified. We report the *Rank-1 Accuracy (R-1)*, which is the percentage of tests where the image in the gallery with the minimum distance to the probe image has the same identity as the probe image.

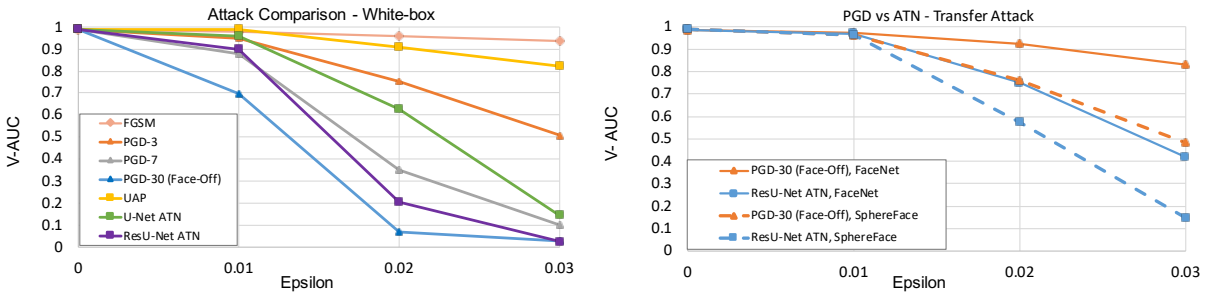


Figure 2.6. Comparison of PGD and ATN based attacks. (a) compares white-box attacks on the single RN-SF-1 model. (b) compares transfer attacks optimized on the six RN-SF-6 models and evaluated on two different models.

Baseline Attacks

We compare the effectiveness of ATN attacks against three alternate attacks:

1. Universal Adversarial Perturbations (UAP): UAP is a single input-agnostic perturbation vector that can be added to all images to fool the victim models. UAP can be formulated as a simplified ATN where the ATN formulation reduces to : $g_\theta(x) = \varepsilon \cdot \tanh(\theta_{h\times w\times c})$. That is, instead of modeling ATN as a neural network, the ATN is modelled using a perturbation vector $\theta_{h\times w\times c}$ which is trained using the same procedure given by Algorithm 1.

2. Fast Gradient Sign Method (FGSM): FGSM [25] attack obtains an adversarial example for an image by obtaining the gradient of the optimization objective with respect to the

image and then perturbing the image in the direction of the gradient with step size ε. That is, $x_{adv} = clip_{[0,1]}(x + \varepsilon \cdot sign(\nabla_x L(x)))$ where $L(x)$ is the optimization objective given by Equation 2.1.

3. Face-Off [27] / Projected Gradient Descent (PGD): PGD [31] attack is a multi-step iterative variant of the FGSM attack. Unlike ATNs and UAPs, PGD attack requires several forward and backward passes through a victim face-recognition model to find an adversarial perturbation that is highly optimized for a single image. We perform PGD attack as a baseline because it has been commonly adopted by past attacks such as Fawkes [28], Face-Off [27], [51] and achieves highest white-box attack success rates. In our experiments *PGD-n* refers to PGD with *n* iterations.

2.2.5 Results

We train six ATN models each targeting one of the model sets listed in Table 2.1. The ATNs are trained using mini-batch gradient descent with a batch size 32 for 500K iterations using Adam optimizer [62] with a learning rate $2e$-4. Our primary evaluation is conducted using the ResU-Net ATN architecture described in Section 2.2.3 at max L_∞ distortion $\varepsilon = 0.03$ in $[0,1]$ pixel scale. We present the white-box and transfer attack results of our primary evaluation in Table 2.2. Additionally, we present the results and comparisons for the pure U-Net architecture in Section 2.2.5 and comparison against alternate attacks at $\varepsilon = [0.01, 0.02, 0.03]$ in Figure 2.6. Finally, we present examples of adversarial images generated at $\varepsilon = [0.03]$ from each of the techniques in Figure 2.7.

Single Model Attack vs Ensemble Attack

Aversarial perturbations trained on an ensemble of victim models exhibit better transferability across model architectures than those trained on a single model. That is, the attack success metrics on unseen models for ATNs trained on ensemble models (RN-SF-6, RN-DID-6 and IN-SF-4) are significantly better than ATNs trained on single models (RN-SF-1, RN-DID-1

and IN-SF-1 respectively). The only difference amongst the models in an ensemble is their weight initialization. It is interesting to note that this difference in weight initialization offers enough variance in the model set to train significantly more generalizable perturbations, at the same level of distortion as compared to the single-model attacks.

ATN vs. PGD based attacks

State-of-the-art attacks such as Fawkes [28] and Face-Off [27] utilize PGD based attack against face recognition systems. As such, we implement the PGD based attack algorithm proposed in Face-Off on our models and architectures to compare the effectiveness of ATNs and PGD on both seen and unseen models. We optimize PGD-30 and ATN attacks on the same surrogate models and perform the attack on a random subset of 10,000 images from the test set. For a fair comparison, we drop the L_{pips} term from the loss and train purely to maximize the cosine distance with an L_∞ constraint. Figure 2.6a shows white-box attack success rate of PGD and ATN attacks on the RN-SF-1 model. As discussed in Section 2.2.3, the Residual U-Net ATN architecture developed by us provides a large improvement over a basic U-net architecture and bridges the white-box performance gap between the ATN and PGD-30.

We also performed an ensemble attack on the six models from RN-SF-6. Running PGD-30 against six surrogate models simultaneously took more than three seconds per image, while the ATN's forward pass was the same complexity as other experiments - more than 10,000$\times$ faster than PGD-30. Table 4.4 reports the timing comparison of ATN and PGD attacks.

In addition to being fast, ATNs learn attacks that generalize effectively to new models. We evaluated how well the perturbed images transferred to two different models. First, we evaluated against a ResNet+SphereFace model that is similar to the RN-SF-6 models, but has a different number of layers. Second, we evaluated against an open source model from the FaceNet repository[2]. This model uses a different architecture (Inception ResNet), loss (DeepID) and training procedure. We did not do any parameter tuning based on this model, so it serves as an

[2]https://github.com/davidsandberg/facenet

Table 2.3. Average Wall-Clock time in seconds required for generating a single adversarial image on GPU (Nvidia Titan X) and CPU platforms using different attacks. Time for RN-SF-1 process indicates the forward pass computation time for a single ResNet Face Recognition model.

	Avg Wall-Clock Time (seconds)	
Process	GPU	CPU
RN-SF-1	2.93e-2	1.02e-1
ATN	2.83e-3	5.67e-2
UAP	1.89e-4	5.39e-3
PGD	3.73	365.2

independent validation of the transferability of our attacks.

Figure 2.6b compares the attacks at different L_∞ thresholds. As expected, transferring an attack from RN-SF-6 to the FaceNet model was more difficult than the ResNet+SphereFace model. However, in both cases the ATN attack is effective at $\varepsilon = 0.02$ and transfers much better than PGD to the new models.

Figure 2.7. Sample adversarial images generated by ReFace attack at $\varepsilon = 0.03$ and their benign counterparts.

ATN vs. UAP

We find that attacks utilizing ATNs outperform the UAP attacks at the same level of perturbation. Since the goal of finding a single input-agnostic perturbation is more challenging than finding one perturbation per image, a higher amount of distortion is required for a successful attack using UAPs as compared to the ATN based attacks. This is indicated by less successful attack metrics (higher V-AUC) from UAPs in Figure 2.6a. However, it is important to note that UAPs pose a significant threat to face recognition models since they can be easily shared amongst attackers and are simpler to implement as compared to ATNs.

ATN Architecture Complexity vs Performance

To study the relationship between the architecture of the ATN model, computational cost and attack effectiveness, we train different ATN architectures to attack the RN-SF-1 face-recognition model. First, we consider different variants of the U-net architecture by progressively increasing the base channel-width from 16 to 64 and the number of downsampling/upsampling layers from 3 to 5. Next, we consider ResU-net architecture with 5 downsampling/upsamping layers with the default 2, 3, and 5 residual blocks in the last three downsampling layers (the default ATN model for all experiments described in Section 5.6). We compare the number of parameters, inference time, and attack effectiveness of the different architectures in Table 2.4. We find that scaling up the architecture complexity and size helps improve attack performance with a marginal increase in the average wall-clock time which is real-time for all ATN attacks and several orders of magnitude faster than PGD-based attack (provided in Table 4.4). Our proposed ResU-net architecture allows adding intermediate residual blocks to increase the number of parameters without requiring additional downsampling/upsampling layers or increasing the base-channel width.

Table 2.4. Model size, inference time and attack effectiveness comparison for different architectures of the ATN model. Inference time is reported as the average wall clock time for a single image on a single Nvidia Titan X GPU and CPU. Attack effectiveness is reported at $\varepsilon = 0.03$.

	Model Size			*Time (seconds)*		*Attack Performance*
ATN Arch.	*#Layers*	*#channels*	*#Params*	GPU	CPU	Ver. AUC
U-net	3	32	$337k$	$1.01e$-3	$6.12e$-3	0.80
U-net	3	64	$1.45m$	$1.25e$-3	$1.35e$-2	0.75
U-net	5	16	$1.04m$	$1.37e$-3	$1.05e$-2	0.35
U-net	5	32	$4.17m$	$1.45e$-3	$1.76e$-2	0.21
U-net	5	64	$16.6m$	$1.73e$-3	$3.12e$-2	0.15
ResU-net	5	64	$48.6m$	$2.83e$-3	$5.67e$-2	0.03

2.2.6 Vulnerabilities of Public Face Recognition APIs

We demonstrate the effectiveness of our attacks against commercial face recognition systems. These systems are black-box, proprietary, and are abstracted away through a web-based API. We evaluate our perturbations against the Amazon (AWS) Rekognition and Microsoft Azure Face services.

Face Verification: In this setting, we target the *CompareFaces* API in AWS and the *verify_face_to_face* API in the Azure Face client. We prepare a total of 1000 image pairs (500 positive and 500 negative pairs sampled randomly from the VGGFace2 test set) and report the verification metrics in Table 2.5.

Face Identification: We target the *SearchFaces* API in AWS Rekognition. The API accepts a gallery of N faces $x_1, x_2, x_3 ... x_N$ and a query image x_q, and returns similar faces to the query image from those in the gallery, ranked in order of similarity to the query image. We generate a gallery of 500 benign faces each with unique identities randomly sampled from the VGGFace2 test set and 500 adversarial samples by adversarially perturbing alternate images of the same identities as those in the gallery, resulting in a total of 500 trials. We report the Rank-1 accuracy of this experiment in Table 2.5.

Table 2.5. ATN attack results at $\epsilon = 0.03$ on AWS and Azure face recognition APIs. The ATN was trained jointly on RN-SF-6 and IN-SF-4. Recall(%) indicates the verification accuracy on only the positive pairs in the evaluation set. For verification, we use the default match threshold 0.5 for both AWS and Azure.

	Verification				*Identification*
	V-Acc. (%)		Recall (%)		Rank-1 Acc. (%)
Input type	AWS	Azure	AWS	Azure	AWS
Clean images	95.5	91.0	91.0	83.0	82.0
Ensemble ATN	64.7	50.1	30.2	2.1	16.4

For attacking the above APIs, we train an ATN jointly on the ensemble of RN-SF-6 and IN-SF-4 models. Training the attack against an ensemble of different architectures helps achieve more effective attack generalization against black-box models. As reported by the results in Table 2.5, for images perturbed by our Ensemble ATN, we achieve a significant drop in both verification and identification metrics as compared to the API performance on Clean images. This is indicated by the lower verification accuracy, recall rate and identification accuracy for the Ensemble ATN attack.

2.2.7 Conclusion

In this section I described our proposed framework ReFace, a real-time, highly transferable attack on face recognition models based on Adversarial Transformation Networks. Using our Residual U-Net ATN model, we bridge the performance gap between ATN and gradient-based PGD attacks while being several orders of magnitude faster than PGD attacks. Unlike prior work, our method enables real-time attacks on video streams which we demonstrate via successful attacks on face recognition over web-cam interfaces. We demonstrate that adversarial examples generated using ATNs can effectively bypass face recognition systems in both white-box and black-box transfer attack settings. Our work bridges the attack effectiveness gap between real-time ATN attacks and PGD attacks by developing a Residual U-net architecture that allows us to

effectively increase the capacity of the ATN. Our ResU-Net ATN approaches PGD performance in white-box attacks and outperforms PGD on black-box transfer attacks. Adversarial examples generated from our framework can bypass commercial face recognition APIs in a complete black-box setting and reduce face identification accuracy from 82% to 16.4%. Our extensive experiments validate that ReFace attacks can effectively target the embedding space of face recognition models, and therefore serve as a strong benchmark to investigate the adversarial robustness of future models.

2.3 Vulnerabilities of DL based Speech Recognition

Machine learning agents serve as the backbone of several speech recognition systems, widely used in personal assistants of smartphones and home electronic devices (e.g. Apple Siri, Google Assistant). Traditionally, Hidden Markov Models (HMMs) [63, 64, 65, 66, 67, 68] were used to model sequential data but with the advent of deep learning, state-of-the-art speech recognition systems are based on DNNs [4, 69, 70, 71].

However, several studies have demonstrated that DNNs are vulnerable to adversarial examples [25, 20, 13, 26, 9]. An adversarial example is a sample from the classifier's input domain which has been perturbed in a way that is intended to fool a victim machine learning (ML) model. While the perturbation is usually imperceptible, such an adversarial input can mislead neural network models deployed in real-world settings causing it to output an incorrect class label with higher confidence.

A vast amount of past research in adversarial machine learning has shown such attacks to be successful in the image domain [72, 9, 10, 73, 11]. However, few works have addressed attack scenarios involving other modalities such as audio. This limits our understanding of system vulnerabilities of many commercial speech recognition models employing DNNs, such as Amazon Alexa, Google Assistant, and home electronic devices like Amazon Echo and Google Home. Recent studies that have explored attacks on automatic speech recognition (ASR) systems

[74, 1, 75, 76], have demonstrated that adversarial examples exist in the audio domain. The authors of [1] proposed targeted attacks where an adversary designs a perturbation that can cause the original audio signal to be transcribed to any phrase desired by the adversary. However, calculating such perturbations requires the adversary to solve an optimization problem for each data-point they wish to mis-transcribe. This makes the attack in-applicable in real-time since the adversary would need to re-solve the data-dependent optimization problem from scratch for every new data-point.

Universal Adversarial Perturbations [40] have demonstrated that there exist universal *image-agnostic* perturbations which when added to any image will cause the image to be mis-classified by a victim network with high probability. The existence of such perturbations poses a threat to machine learning models in real world settings since the adversary may simply add the same pre-computed universal perturbation to a new image and cause mis-classification.

Contributions: In this chapter, we address the question "Do universal adversarial perturbations exist for neural networks in audio domain?" We demonstrate the existence of universal audio-agnostic perturbations that can fool DNN based ASR systems.[3] We propose an algorithm to design such universal perturbations against a victim ASR model in the *white-box setting*, where the adversary has access to the victim's model architecture and parameters. We validate the feasibility of our algorithm, by crafting such perturbations for Mozilla's open source implementation of the state-of-the-art speech recognition system DeepSpeech [71]. Additionally, we discover that the generated universal perturbation is transferable to a significant extent across different model architectures. Particularly, we demonstrate that a universal perturbation trained on DeepSpeech can cause significant transcription error on a WaveNet [4] based ASR model.

[3] Sound Examples: universal-audio-perturbation.herokuapp.com

2.3.1 Universal Adversarial Perturbations for Real-time Attacks on Speech Recognition Systems

In this section, I discuss the existence of universal adversarial audio perturbations that cause mis-transcription of audio signals by automatic speech recognition (ASR) systems. I describe our proposed algorithm to find a single quasi-imperceptible perturbation, which when added to any arbitrary speech signal, will most likely fool the victim speech recognition model. Our experiments demonstrate the application of our proposed technique by crafting audio-agnostic universal perturbations for the state-of-the-art ASR system – Mozilla DeepSpeech. Additionally, we show that such perturbations generalize to a significant extent across models that are not available during training, by performing a transferability test on a WaveNet based ASR system.

2.3.2 Background and Related Work

Adversarial Attacks in the Audio Domain: Adversarial attacks on ASR systems have primarily focused on *targeted attacks* to embed carefully crafted perturbations into speech signals, such that the victim model transcribes the input audio into a specific malicious phrase, as desired by the adversary [74, 1, 77, 75, 78]. Prior works [75, 78] demonstrate successful attack algorithms targeting traditional speech recognition models based on HMMs and GMMs, that operate on Mel Frequency Cepstral Coefficient (MFCC) representation of audio. For example, in Hidden Voice Commands [75], the attacker uses inverse feature extraction to generate obfuscated audio that can be played over-the-air to attack ASR systems. However, obfuscated samples sound like random noise rather than normal human perceptible speech and therefore come at the cost of being fairly perceptible to human listeners. Additionally, these attack frameworks are not end-to-end, which render them impractical for studying the vulnerabilities of modern ASR systems – that are entirely DNN based.

In recent work [1], Carlini *et al.* propose an end-to-end white-box attack technique to craft adversarial examples, which transcribe to a target phrase. Similar to the work in images,

they propose a gradient-based optimization method that replaces the cross-entropy loss function used for classification, with a Connectionist Temporal Classification (CTC) loss [79] which is optimized for time-sequences. The CTC-loss between the target phrase and the network's output is backpropagated through the victim neural network and the MFCC computation, to update the additive adversarial perturbation. The adversarial samples generated by this work are quasi-perceptible, motivating a separate work [80] to minimize the perceptibility of the adversarial perturbations using psychoacoustic hiding.

Designing adversarial perturbations using all the above mentioned approaches requires the adversary to solve a data dependent optimization problem for each input audio signal the adversary wishes to mis-transcribe, making them ineffective in a real-time attack scenario. In other words, targeted attacks must be customized for each segment of audio, a process that cannot yet be done in real-time. The existence of universal adversarial perturbations (described below) can pose a more serious threat to ASR systems in real-world settings since the adversary may simply add the same pre-computed universal adversarial perturbation to any input audio and fool the DNN based ASR system.

Universal Adversarial Perturbations: The authors of [40] craft a single universal perturbation vector which can fool a victim neural network to predict a false classification output on the majority of validation instances. Let $\hat{k}(x)$ be the classification output for an input x that belongs to a distribution μ. The goal is to find a perturbation v such that: $\hat{k}(x+v) \neq \hat{k}(x)$ for "most" $x \in \mu$. This is formulated as an optimization problem with constraints to ensure that the universal perturbation is within a specified p-norm and is also able to fool the desired number of instances in the training set. The proposed algorithm iteratively goes over the training dataset to build a universal perturbation vector that pushes each data point to its decision boundary. The authors demonstrate that it is possible to find a quasi-imperceptible universal perturbation that pushes most data points outside the correct classification region of a victim model. More interestingly, the work demonstrates that the universal perturbations are transferable across models with different architectures. The perturbation produced using one network such as VGG-16 can also be used to

fool another network e.g. GoogLeNet showing that their method is doubly universal. *Universal adversarial perturbations* for images focuses on the goal of mis-classification and cannot directly be applied to the more challenging goal of mis-transcription by Speech Recognition System. In our work we address this challenge and solve an alternate optimization problem to adapt the method for designing universal adversarial perturbations for ASR systems.

2.3.3 Methodology

Threat Model

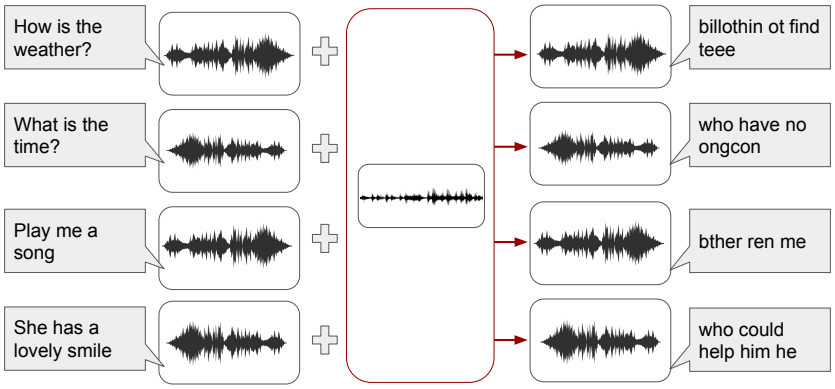


Figure 2.8. Threat Model: We aim to find a single perturbation which when added to any arbitrary audio signal, will most likely cause an error in transcription by a victim Speech Recognition System

We aim to find a universal audio perturbation, which when added to any speech waveform, will cause an error in transcription by a speech recognition model with high probability. For the success of the attack, the error in the transcription should be high enough so that the transcription of the perturbed signal (adversarial transcription) is incomprehensible and the original transcription cannot be deduced from the adversarial transcription. As discussed in [1], the transcription *"test sentence"* mis-spelled as *"test sentense"* does little to help the adversary. To make the adversary's goal challenging, we report success only when the Character Error Rate (CER) or the normalized Levenshtein distance *(Edit Distance)* [81] between the original and adversarial transcription is greater than a particular threshold. Formally, we define our threat model as follows:

Let μ denote a distribution of waveforms and C be the victim speech recognition model that transcribes a waveform x to $C(x)$. The goal of our work is to find perturbations v such that:

$$CER(C(x), C(x+v)) > t \text{ for "most" } x \in \mu$$

Here, $CER(x,y)$ is the edit distance between the strings x and y normalized [81] by the *length* of x i.e

$$CER(x,y) = \frac{EditDistance(x,y)}{length(x)}$$

The threshold t is chosen as 0.5 for our experiments i.e., we report success only when the original transcription has been *edited* by at least 50% of its length using character *removal, insertion, or substitution* operations.

The universal perturbation signal v is chosen to be of a fixed length and is cropped or zero-padded at the end to make it equal to length of the signal x.

Distortion Metric

To quantify the distortion introduced by some adversarial perturbation v, an l_∞ metric is commonly used in the space of images. Following the same convention, in the audio domain [13], the loudness of the perturbation can be quantified using the dB scale, where $dB(v) = \max_i(20.\log_{10}(v_i))$. We calculate $dB_x(v)$ to quantify the relative loudness of the universal perturbation v with respect to an original waveform x where:

$$dB_x(v) = dB(v) - dB(x)$$

Since the perturbation introduced is quieter than the original signal, $dB_x(v)$ is a negative value, where smaller values indicate quieter distortions. In our results, we report the average relative loudness: $dB_x(v)$ across the whole test set to quantify the distortion introduced by our universal perturbation.

Problem Formulation and Algorithm

Our goal to find a quasi-imperceptible universal perturbation vector v such that it mis-transcribes *most* data points sampled from a distribution μ. Mathematically, we want to find a perturbation vector v that satisfies:

1. $\|v\|_\infty < \varepsilon$

2. $\underset{x\sim\mu}{P}(CER(C(x), C(x+v) > t)) \geq \delta$.

Here ε is the maximum allowed l_∞ norm of the perturbation, δ is the desired attack success rate and t is the threshold CER chosen to define our success criteria.

To solve the above problem, we adapt the universal adversarial perturbation algorithm proposed by [40] to find universal adversarial perturbations for the goal of *mis-transciption* of speech waveforms instead of *mis-classification* of data (images). Let $X = x_1, x_2, \ldots, x_m$ be a set of speech signals sampled from the distribution μ. Our Algorithm (4) goes over the data-points in X iteratively and gradually builds the perturbation vector v. At each iteration i, we seek a minimum perturbation Δv_i, that causes an error in the transcription of the current perturbed data point $x_i + v$. We then add this additional perturbation Δv_i to the current universal perturbation v and clip the new perturbation v, if necessary, to satisfy the constraint $\|v\|_\infty < \varepsilon$.

Algorithm 2. Universal Adversarial Perturbations for Speech Recognition Systems

input: Training Data Points X, Validation Data Points X_v Victim Model C, allowed distortion level ε, desired success rate δ
output: Universal Adversarial Perturbation vector v
Initialize $v \leftarrow 0$
while $SuccessRate(X_v) < \delta$ **do**
    for each data point $x_i \in X$ **do**
        if $CER(C(x_i + v + r), C(x_i)) < t$ **then**
            Compute min perturbation that mis-transcribes $x_i + v$: $\Delta v_i \leftarrow \arg\min_r \|r\|_2$ s.t.: $CER(C(x_i + v + r), C(x_i)) > t$
            Update and clip universal perturbation v: $v = Clip_{v,\varepsilon}(v + \Delta v_i)$

At each iteration we need to solve the following optimization problem, that seeks a minimum (under *l2* norm) additional perturbation Δv_i, to mis-transcribe the current perturbed audio signal $x_i + v$:

$$\Delta v_i \leftarrow \arg\min_r \|r\|_2 \text{ s.t. } CER(C(x_i + v + r), C(x_i)) > t \tag{2.3}$$

It is non-trivial to solve the above optimization in its current form. In [40], the authors try to solve a similar optimization problem for the goal of *mis-classification* of data points. They approximate its solution using DeepFool [82] which finds a minimum perturbation vector that pushes a data point to its decision boundary. Since we are tackling a more challenging goal of *mis-transcription* of signals where we have decision boundaries for each audio frame across the time axis, the same idea cannot be directly applied. Therefore, we approximate the solution to the optimization problem given by Equation 2.3 by solving a more tractable optimization problem:

$$\begin{aligned} &\text{Minimize } J(r) \text{ where} \\ &J(r) = c\|r\|^2 + L(x_i + v + r, C(x_i)) \\ &\text{s.t. } \|v + r\|_\infty < \varepsilon \\ &\text{where } L(x,y) = -CTCLoss(f(x), y) \end{aligned} \tag{2.4}$$

In other words, to mis-transcribe the signal, we aim to maximize the CTC-Loss between the predicted probability distributions of the perturbed signal $f(x_i + v + r)$ and the original transcription $C(x_i)$ while having a regularization penalty on the $l2$ norm of r. Since this a non-convex optimization problem, we approximate its solution using iterative gradient sign

method [83]:

$$
\begin{aligned}
r_0 &= \overrightarrow{0} \\
r_{N+1} &= Clip_{r+v,\varepsilon}\{r_N - \alpha sign(\Delta_{r_N} J(r_N)\}
\end{aligned}
\tag{2.5}
$$

Note that the error J is back-propagated through the entire neural network and the MFCC computation to the perturbation vector r. We iterate until we reach the desired CER threshold t for a particular data point x_i. The regularization constant c is chosen through hyper-parameter search on a validation set to find the maximum success rate for a given magnitude of allowed perturbation.

2.3.4 Experiments

We demonstrate the application of our proposed attack algorithm on the pre-trained *Mozilla DeepSpeech* model [71, 84]. We train our algorithm on the Mozilla Common Voice Dataset [71] which contains 582 hours of audio across 400,000 recordings in English. We train on a randomly selected set X containing 5,000 audio files from the training set and evaluate our model on both the training set X and the entire unseen validation set of the Mozilla Common Voice Dataset. We analyze the effect of the size of the set X below. The length of our universal adversarial perturbation is fixed to 150,000 samples which corresponds to around 9 seconds of audio at 16 KHz. The universal adversarial perturbations are trained using our proposed algorithm 4 with a learning rate $\alpha = 5$ and the regularization parameter c set to 0.5.

Evaluation: We utilize two metrics: *i) Mean CER* - Character Error Rate averaged over the entire test set and *ii) Success Rate* to evaluate our universal adversarial perturbations. We report success on a particular waveform, if the *CER* between the original and adversarial transcription (Section 2.3.3) is greater than 0.5. The amount of perturbation is quantified using mean relative distortion $dB_x(v)$ over the test set (Refer to Section 2.3.3).

Table 2.6. Results of our algorithm for different allowed magnitude of universal adversarial perturbation

	Training Set (X)			Test Set		
$\|\mathbf{v}\|_\infty$	**Mean $\mathbf{dB_x(v)}$**	**Success Rate (%)**	**Mean CER**	**Mean $\mathbf{dB_x(v)}$**	**Success Rate (%)**	**Mean CER**
100	-42.03	57.46	0.63	-41.86	56.13	0.64
150	-38.51	72.78	0.81	-38.34	72.49	0.82
200	-36.01	83.27	0.92	-35.84	80.47	0.95
300	-32.49	89.52	1.10	-32.32	89.06	1.11
400	-30.18	90.60	1.06	-29.82	88.24	1.07

2.3.5 Results

Table 2.6 shows the results of our algorithm for different allowed magnitude of universal adversarial perturbation on both the training set X and the unseen Test Set. Both the success rate and the Mean Character Error Rate (CER) increase with increase in the maximum allowed perturbation. We achieve a success rate of 89.06 % on the validation set, with the mean distortion metric $dB_x(v) \approx -32dB$. To interpret the results in context, $-32dB$ is roughly the difference between ambient noise in a quiet room and a person talking [85, 1] . We encourage the reader to listen to our adversarial samples and their corresponding transcriptions on our web page (link in the footnote of the first page)

Figure 2.9 (a) shows the success rate and mean edit distance compared to the size of the training set X for maximum allowed perturbation $\|v\|_\infty = 200$ (Mean $dB_x(v) = -36.01$). We observe that it is possible to train our proposed algorithm on very few examples and achieve reasonable success rates on unseen data. For example, training on just 1000 examples can achieve a success rate of 80.47 % on the test set.

Effectiveness of universal perturbations

In order to assess the vulnerability of the victim Speech Recognition System to our attack algorithm, we compare our universal perturbation with random (uniform) perturbation

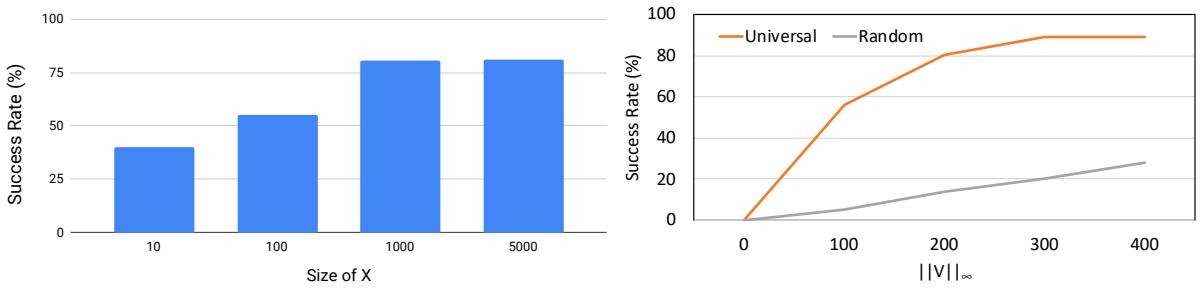


Figure 2.9. (a) Attack Success Rate on the test set vs. the number of audio files in the training set X (b) Success Rate vs $\|v\|_\infty$ of universal and random perturbations.

having the same magnitude of distortion (same $\|v\|_\infty$) as our universal adversarial perturbation. Figure 2.9 (b) shows the plot of success rate vs. the magnitude of the perturbation for each of these perturbations. It can be seen that universal adversarial perturbations are able to achieve high success rate with very low magnitude of distortion as compared to a random noise perturbation. For example, for allowed perturbation $\|v\|_\infty = 100$ our universal perturbation achieves a success rate of 65% which is substantially higher than the success rate of random noise. This implies that for the same magnitude of distortion, distorting an audio waveform in a random direction is significantly less likely to cause mis-transcription as compared to distorting the waveform in the direction of universal perturbation. Our results support the hypothesis discussed in [40], demonstrating that universal adversarial perturbations exploit geometric correlations in the decision boundaries of the victim model.

Table 2.7. Results of the same universal adversarial perturbation on two victim models: Wavenet and Mozilla DeepSpeech. The universal perturbation was trained on the DeepSpeech model.

		Wavenet		**Mozilla DeepSpeech**	
$\|\mathbf{v}\|_\infty$	**Mean dBx(v)**	**Success Rate (%)**	**Mean CER**	**Success Rate (%)**	**Mean CER**
150	-38.34	**26.97**	**0.37**	72.49	0.82
200	-35.84	**31.18**	**0.40**	80.47	0.95
300	-32.32	**42.05**	**0.47**	89.06	1.11
400	-29.82	**63.28**	**0.60**	88.24	1.07

Cross-model Transferability

We perform a study on the transferability of adversarial samples to deceive ML models that have not been used for training the universal adversarial perturbation, i.e., their parameters and network structures are not revealed to the attacker. We train universal adversarial perturbations for Mozilla DeepSpeech and evaluate the extent to which they are valid for a different ASR architecture based on WaveNet [4]. For this study, we use a publicly available pre-trained model of WaveNet [86] and evaluate the transcriptions obtained using clean and adversarial audio for the same unseen validation dataset as used in our previous experiments. Our results in Table 2.7 indicate that our attack is transferable to a significant extent for this particular setting. Specifically, when the mean $dB_x(v) = -29.82$, we are able to achieve a 63.28% success rate while attacking the WaveNet based ASR model. This result demonstrates the practicality of such adversarial perturbations, since they are able to generalize well across data points and architectures.

2.3.6 Conclusion

In this section, I described the existence of audio-agnostic adversarial perturbations for speech recognition systems. We demonstrate that our audio-agnostic adversarial perturbation generalizes well across unseen data points and to some extent across unseen networks. Our proposed end-to-end approach can be used to further understand the vulnerabilities and blind spots of deep neural network based ASR system, and provide insights for building more robust neural networks.

2.4 Acknowledgements

Chapter 2 contains material found in the following two papers (1) *ReFace: Adversarial Transformation Networks for Real-time Attacks on Face Recognition Systems*. IEEE/IFIP International Conference on Dependable Systems and Networks, 2023. Hussain, Shehzeen; Huster,

Todd; Mesterharm, Chris; Neekhara, Paarth; Koushanfar, Farinaz. (2) *Universal Adversarial Perturbations for Speech Recognition Systems*. Interspeech, 2019. Neekhara, Paarth; Hussain, Shehzeen; Pandey, Prakhar; Dubnov, Shlomo; McAuley, Julian; Koushanfar, Farinaz. The dissertation author was the primary investigator and author of paper (1). The dissertation author and Paarth Neekhara made equal contributions to the work done in paper (2).

Chapter 3

Vulnerabilities of DL to Adversarial Reprogramming

3.1 Vulnerabilities of DL based Text Recognition

While neural network based machine learning models serve as the backbone of many text and image processing systems, recent studies have shown that they are vulnerable to adversarial examples. Traditionally, an adversarial example is a sample from the classifier's input domain which has been perturbed in such a way that is intended to cause a machine learning model to misclassify it. While the perturbation is usually imperceptible, such an adversarial input results in the neural network model outputting an incorrect class label with higher confidence. Several studies have shown such adversarial attacks to be successful in both the continuous input domain [72, 9, 87, 73, 11] and discrete input spaces [88, 89, 90].

Adversarial Reprogramming [91] is a new class of adversarial attacks where a machine learning model is repurposed to perform a new task chosen by the attacker. The proposed attack is interesting because it allows an adversary to move a step beyond mere mis-classification of a victim network's output onto having the control to repurpose the network fully. The authors demonstrated how an adversary may repurpose a pre-trained ImageNet [92] model for an alternate classification task like classification of MNIST digits or CIFAR-10 images without modifying the network parameters. Since machine learning agents can be reprogrammed to perform unwanted actions as desired by the adversary, such an attack can lead to theft of computational resources

such as cloud-hosted machine learning models. Besides theft of computational resources, the adversary may perform a task that violates the code of ethics of the system provider.

The adversarial reprogramming approach proposed by [91] trains an additive contribution θ to the inputs of the neural network to repurpose it for the desired alternate task. This approach assumes a white-box attack scenario where the adversary has full access to the network's parameters. Also, the adversarial program proposed in this work is only applicable to tasks where the input space of the the original and adversarial task is continuous.

In our work, we propose a method to adversarially *repurpose* neural networks which operate on sequences from a *discrete* input space. The task is to learn a simple transformation (adversarial program) from the input space of the adversarial task to the input space of the neural network such that the neural network can be repurposed for the adversarial task. We propose a context-based vocabulary remapping function as an adversarial program for sequence classification networks. We propose training procedures for this adversarial program in both white-box and black-box scenarios. In the white-box attack scenario, where the adversary has access to the classifier's parameters, a Gumbel-Softmax trick [93] is used to train the adversarial program. Assuming a black-box attack scenario, where the adversary may not have access to the classifier's parameters, we present a REINFORCE [94] based optimization algorithm to train the adversarial program.

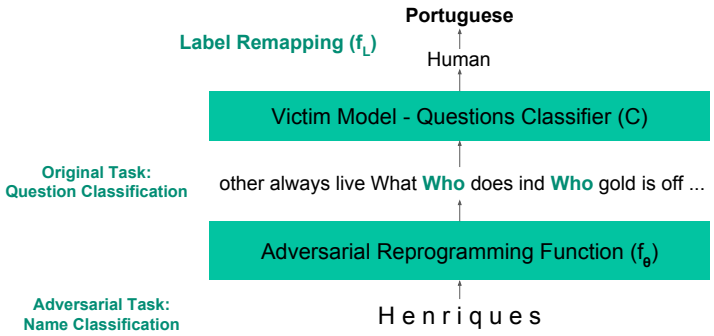


Figure 3.1. Example of Adversarial Reprogramming for Sequence Classification. We aim to design and train the adversarial reprogramming function f_θ, such that it can be used to repurpose a pre-trained classifier C, for a desired adversarial task.

We apply our proposed methodology on various text classification models including Recurrent Neural Networks such as LSTMs and bidirectional LSTMs, and Convolutional Neural Networks (CNNs). We demonstrate experimentally, how these neural networks trained on a particular (original) text classification task can be repurposed for alternate (adversarial) classification tasks. We experiment with different text classification datasets given in table 3.1 as candidate original and adversarial tasks and adversarially reprogram the aforementioned text classification models to study the robustness of the attack.

3.1.1 Adversarial Reprogramming of Text Classification Neural Networks

In this work, we develop methods to repurpose text classification neural networks for alternate tasks without modifying the network architecture or parameters. We propose a context based vocabulary remapping method that performs a computationally inexpensive input transformation to reprogram a victim classification model for a new set of sequences. We propose algorithms for training such an input transformation in both white box and black box settings where the adversary may or may not have access to the victim model's architecture and parameters. We demonstrate the application of our model and the vulnerability of neural networks by adversarially repurposing various text-classification models including LSTM, bi-directional LSTM and CNN for alternate classification tasks.

3.1.2 Background and Related Work

Adversarial Examples

Traditionally, adversarial examples are intentionally designed inputs to a machine learning model that cause the model to make a mistake [11]. These attacks can be broadly classified into *untargeted* and *targeted* attacks. In the untargeted attack scenario, the adversary succeeds as long as the victim model classifies the adversarial input into *any* class other than the correct class, while in the *targeted* attack scenario, the adversary succeeds only if the model classifies

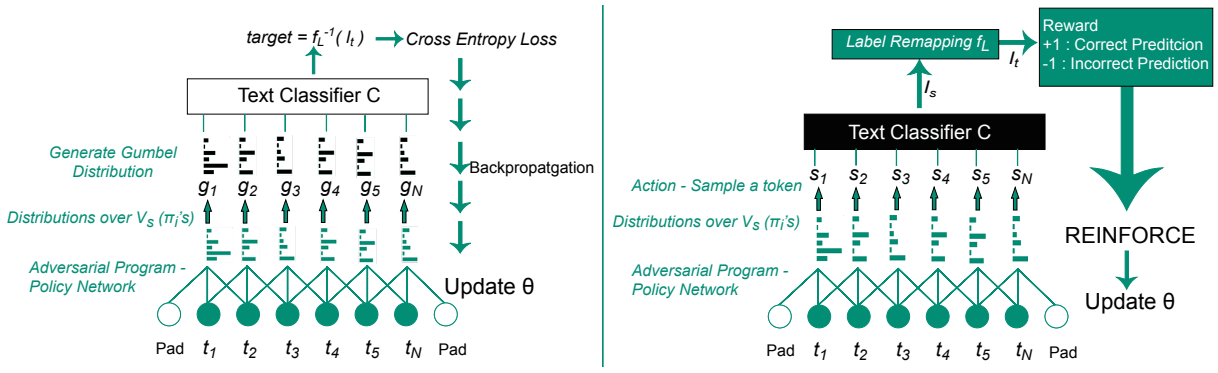


Figure 3.2. Adversarial Reprogramming Function and Training Procedures. **Left:** White-box Adversarial Reprogramming using gumbel softmax distributions. **Right:** Black-box Adversarial Reprogramming using REINFORCE algorithm.

the adversarial input into a specific incorrect class. In both these scenarios, the intent of the adversary is usually malicious and the outcome of the victim model is still limited to the original task being performed by the model.

Adversarial attacks of image-classification models often use gradient descent on an image to create a small perturbation that causes the machine learning model to mis-classify it [72, 95]. There has been a similar line of adversarial attacks on neural networks with discrete input domains [88, 90], where the adversary modifies a few tokens in the input sequence to cause mis-classification by a sequence model. In addition, efforts have been made in designing more general adversarial attacks in which the same modification can be applied to many different inputs to generate adversarial examples [11, 40]. For example, authors [32] trained an Adversarial Transformation Network that can be applied to all inputs to generate adversarial examples targeting a victim model or a set of victim models. In this work, we aim to learn such universal transformations of discrete sequences for a fundamentally different task: *Adversarial Reprogramming* described below.

Adversarial Reprogramming

Adversarial Reprogramming [91] introduced a new class of adversarial attacks where the adversary wishes to repurpose an existing neural network for a new task chosen by the

attacker, without the attacker needing to compute the specific desired output. The adversary achieves this by first defining a hard-coded one-to-one label remapping function h_g that maps the output labels of the adversarial task to the label space of the classifier f; and learning a corresponding adversarial reprogramming function $h_f(.;\theta)$ that transforms an input $(\tilde{X})$ [1] from the input space of the new task to the input space of the classifier. The authors proposed an adversarial reprogramming function $h_f(.;\theta)$, for repurposing ImageNet models for adversarial classification tasks. An adversarial example X_{adv} for an input image $\tilde{X}$ can be generated using the following adversarial program: [2]

$$X_{adv} = h_f(\tilde{X};\theta) = \tilde{X} + \tanh(\theta)$$

where $\theta \in \mathbb{R}^{n\times n\times 3}$ is the learnable weight matrix of the adversarial program (where n is the ImageNet image width). Let $P(y|X)$ denote the probability of the victim model predicting label y for an input X. The goal of the adversary is to maximize the probability $P(h_g(y_{adv})|X_{adv})$ where y_{adv} is the label of the adversarial input X_{adv}. The following optimization problem that maximizes the log-likelihood of predictions for the adversarial classification task, can be solved using backpropagation to train the adversarial program parameterized by θ:

$$\hat{\theta} = argmin_\theta \left(-\log P(h_g(y_{adv})|X_{adv}) + \lambda||\theta||_2^2\right) \tag{3.1}$$

where λ is the regularization hyperparameter. Since the adversarial program proposed is a trainable additive contribution θ to the inputs, it's application is limited to neural networks with a continuous input space. Also, since the the above optimization problem is solved by back-propagating through the victim network, it assumes a white-box attack scenario where the adversary has gained access to the victim model's parameters. In this work, we describe how we can learn a simple transformation in the *discrete space* to extend the application of adversarial

[1] $\tilde{X}$ is an ImageNet size $(n \times n \times 3)$ padded input image

[2] Masking ignored because it is only a visualization convenience

reprogramming on *sequence classification* problems. We also propose a training algorithm in the black-box setting where the adversary may not have access to the model parameters.

Transfer Learning

Transfer Learning [96] is a study closely related to Adversarial Reprogramming. During training, neural networks learn representations that are generic and can be useful for many related tasks. A pre-trained neural network can be effectively used as a feature extractor and the parameters of just the last few layers are retrained to realign the output layer of the neural network for the new task. Prior works have also applied transfer learning on text classification tasks [97, 98]. While transfer learning approaches exploit the learnt representations for the new task, they cannot be used to repurpose an exposed neural network for a new task without modifying some intermediate layers and parameters.

Adversarial Reprogramming reprogramming studied whether it is possible to keep all the parameters of the neural network unchanged and simply learn an input transformation that can realign the outputs of the neural network for the new task. This makes it possible to repurpose *exposed* neural network models like cloud-based photo services to a new task where transfer learning is not applicable since we do not have access to intermediate layer outputs.

3.1.3 Methodology

Threat Model

Consider a sequence classifier C trained on the original task of mapping a sequence $s \in S$ to a class label $l_S \in L_S$ i.e $C : s \mapsto l_S$. An adversary wishes to repurpose the original classifier C for the adversarial task C' of mapping a sequence $t \in T$ to a class label $l_T \in L_T$ i.e $C' : t \mapsto l_T$. The adversary can achieve this by hard-coding a one-to-one label remapping function:

$$f_L : l_S \mapsto l_T$$

that maps an original task label to the new task label and learning a corresponding adversarial reprogramming function:

$$f_\theta : t \mapsto s$$

that transforms an input from the input space of the adversarial task to the input space of the original task. The adversary aims to update the parameters θ of the adversarial program f_θ such that the mapping $f_L(C(f_\theta(t)))$ can perform the adversarial classification task $C' : t \mapsto l_T$. Note that the victim model's parameters will be kept frozen while training the reprogramming function f_θ.

Adversarial Reprogramming Function

The goal of the adversarial reprogramming function $f_\theta : t \mapsto s$ is to map a sequence t to s such that it is labeled correctly by the classifier $f_L(C)$.

The tokens in the sequence s and t belong to some vocabulary lists V_S and V_T respectively. We can represent the sequence s as $s = s_1, s_2, .., s_N$ where s_i is the vocabulary index of the i_{th} token in sequence s in the vocabulary list V_S. Similarly sequence t can be represented as $t = t_1, t_2, .., t_N$ where t_i is the vocabulary index of the i_{th} token of sequence t in the vocabulary list V_T.

In the simplest scenario, the adversary may try to learn a vocabulary mapping from V_T to V_S using which each t_i can be independently mapped to some s_i to generate the adversarial sequence. Such an adversarial program has limited potential since the representational capacity of such a reprogramming function is very limited. We experimentally support this hypothesis by showing how such a transformation has limited potential for the purpose of adversarial reprogramming.

A more sophisticated adversarial program can be a sequence to sequence machine translation model [99] that learns a translation $t \mapsto s$ for adversarial reprogramming. While theoretically this is a good choice, it defeats the purpose of adversarial reprogramming. This is

because the computational complexity of training and using such a machine translation model would be similar if not greater than that of a new sequence classifier for the adversarial task C'.

The adversarial reprogramming function should be computationally inexpensive but powerful enough for adversarial repurposing. To this end, we propose a context-based vocabulary remapping model that produces a distribution over the target vocabulary at each time-step based on the surrounding input tokens. More specifically, we define our adversarial program as a trainable 3-d matrix $\theta_{k\times|V_T|\times|V_S|}$ where k is the context size. Using this, we generate a probability distribution π_i over the vocabulary V_S at each time-step i as follows:

$$h_i = \sum_{j=0}^{k-1} \theta[j, t_{i+\lfloor k/2 \rfloor - j}] \tag{3.2}$$

$$\pi_i = softmax(h_i) \tag{3.3}$$

Both h_i and π_i are vectors of length $|V_S|$. To generate the adversarial sequence s we sample each s_i independently from the distribution π_i i.e $s_i \sim \pi_i$

In practice, during training, we implement this adversarial program as a single layer of 1-d convolution over the sequence of one-hot encoded vectors of adversarial tokens t_i with $|V_T|$ input channels and $|V_S|$ output channels with k-length kernels parameterized by $\theta_{k\times|V_T|\times|V_S|}$.

White-box Attack

In the white-box attack scenario, we assume that the adversary has gained access to the victim network's parameters and architecture. To train the adversarial reprogramming function f_θ, we use an optimization objective similar to equation 3.1. Let $P(l|s)$ denote the probability of predicting label l for a sequence s by classifier C. We wish to maximize the probability $P(f_L^{-1}(l_t)|f_\theta(t))$ which is the probability of the output label of the classifier being mapped to the correct class l_t for an input t in the domain of the adversarial task. Therefore we need to solve the following log-likelihood maximization problem:

$$\hat{\theta} = argmin_{\theta}(-\sum_{t} log(P(f_L^{-1}(l_t)|f_{\theta}(t)))) \tag{3.4}$$

Note that that the output of the adversarial program $s = f_{\theta}(t)$ is a sequence of discrete tokens. This makes the above optimization problem non-differentiable. Prior works [100, 101, 90] have demonstrated how we can smoothen such an optimization problem using the Gumbel-Softmax [93] distribution.

In order to backpropagate the gradient information from the classifier to the adversarial program, we smoothen the generated tokens s_i using Gumbel-Softmax trick as per the following:

For an input sequence t, we generate a sequence of Gumbel distributions $g = g_1, g_2, .., g_N$. The n_{th} component of distribution g_i is generated as follows:

$$g_i^n = \frac{\exp((\log(\pi_i^n) + r_n)/temp)}{\sum_j \exp((\log(\pi_i^j) + r_j)/temp)}$$

where π_i is the softmax distribution at the i_{th} time-step obtained using equation 3.3, r_n is a random number sampled from the Gumbel distribution [102] and $temp$ is the temperature of Gumbel-Softmax.

The sequence then passed to the classifier C is the sequence g which serves as a soft version of the one-hot encoded vectors of s_i's. Since the model is now differentiable, we can solve the following optimization problem using backpropagation:

$$\hat{\theta} = argmin_{\theta}(-\sum_{t} log(P(f_L^{-1}(l_t)|g)))$$

During training the temperature parameter is annealed from some high value t_{max} to a very low value t_{min}. The details of this annealing process for our experiments have been included in the appendix.

Black-box Attack

In the black-box attack scenario, the adversary can only query the victim classifier C for labels (and not the class probabilities). We assume that the adversary has the knowledge of the vocabulary V_S of the victim model. Since the adversarial program needs to produce a discrete output to feed as input to the classifier C, it is not possible to pass the gradient update from the classifier $f_L(C)$ to the adversarial program θ using standard back-propagation. Also, in the black-box attack setting it is not possible to back-propagate the cross entropy loss through the classifier C in the first place.

We formulate the sequence generation problem as a Reinforcement Learning problem [103, 104, 105] where the adversarial reprogramming function is the policy network. The state of the adversarial program is a sequence $t \in T$ and an action of our policy network is to produce a sequence of tokens $s \in S$. The adversarial program parameterized by θ, models the stochastic policy $\pi_{adv}(s|t;\theta)$ such that a sequence $s \in S$ may be sampled from this policy conditioned on $t \in T$. We use a simple reward function where we assign a reward +1 for a correct prediction and -1 for an incorrect prediction using the classifier $f_L(C)$ where f_L is the label remapping function and C is the classifier. Formally:

$$r(t,s) = \begin{cases} +1, & f_L(C(s)) = l_t \\ -1, & f_L(C(s)) \neq l_t \end{cases}$$

The optimization objective to train the policy network is the following:

$$\max_{\theta} J(\theta) \quad \text{where,} \quad J(\theta) = \mathbb{E}_{\pi_{adv}}[r(t,s)]$$

Following the REINFORCE algorithm [94] we can write the gradient of the expectation with respect to θ as per the following:

$$
\begin{aligned}
\nabla_\theta J &= \nabla_\theta \left[\mathop{\mathbb{E}}_{\pi_{adv}} [r(t,s)] \right] \\
&= \nabla_\theta \left[\sum_s \pi_{adv}(s|t;\theta) r(t,s) \right] \\
&= \sum_s \pi_{adv}(s|t;\theta) \nabla_\theta \log(\pi_{adv}(s|t;\theta)) r(t,s) \\
&= \mathop{\mathbb{E}}_{\pi_{adv}} [r(t,s) \nabla_\theta \log(\pi_{adv}(s|t;\theta))] \\
&= \mathop{\mathbb{E}}_{\pi_{adv}} [r(t,s) \nabla_\theta \log(\pi_{adv}(s_1,..,s_N|t;\theta))] \\
&= \mathop{\mathbb{E}}_{\pi_{adv}} \left[r(t,s) \nabla_\theta \log(\prod_i \pi_{adv}(s_i|t;\theta)) \right] \\
&= \mathop{\mathbb{E}}_{\pi_{adv}} \left[r(t,s) \sum_i \nabla_\theta \log(\pi_{adv}(s_i|t;\theta)) \right]
\end{aligned}
$$

Note that $\pi_{adv}(s_i|t;\theta)$ is the same as π_i defined in equation 3.3 which can be differentiated with respect to θ. The expectations are estimated as sample averages. Having obtained the gradient of expected reward, we can use mini-batch gradient ascent to update θ with a learning rate α as: $\theta \leftarrow \theta + \alpha \nabla_\theta J$.

3.1.4 Experiments

Datasets and Classifiers

We demonstrate the application of the proposed reprogramming techniques on various text-classification tasks. In our experiments, we design adversarial programs to attack both word-level and character-level text classifiers. Additionally, we aim to adversarially repurpose a character-level text classifier for a word-level classification task and vice-versa. To this end, we choose the following text-classification datasets as candidates for the original and adversarial classification tasks:

1. Surname Classification Dataset **(Names-18, Names-5) [106]**: The dataset categorizes surnames from 18 languages of origin. We use a subset of this dataset *Names-5* containing Names

Table 3.1. Summary of datasets. $|V|$ denotes the vocabulary size of each dataset.

Data Set	# Classes	Train Samples	Test Samples	$\lvert V \rvert$	Avg Length
Names-18	18	115,028	28,758	90	7.1
Names-5	5	3632	909	66	6.5
Questions	6	4361	1091	1205	11.2
Arabic	2	1600	400	955	9.7
IMDB	2	25,000	25,000	10000	246.8

from 5 classes: *Dutch, Scottish, Polish, Korean* and *Portuguese*, as a candidate for adversarial task in the experiments.

2. Experimental Data for Question Classification **(Questions) [107]**: categorizes around 5500 questions into 6 classes: *Abbreviation, Entity, Description, Human, Location, Numeric*.

3. Sentiment Classification Dataset [108] **(Arabic)** : contains 2000 binary labeled tweets on diverse topics such as politics and arts.

4. Large Movie Review Dataset **(IMDB) for sentiment classification [109]**: contains 50,000 movie reviews categorized into binary class of positive and negative sentiment.

The statistics of the above mentioned datasets have been given in Table 3.1. We train adversarial reprogramming functions to repurpose various text-classifiers based on Long Short-Term Memory (LSTM) network [110], bidirectional LSTM network [111] and Convolutional neural network [112] models. All the aforementioned models can be trained for both word-level and character-level classification. We use character level classifiers for *Names-18* and *Names-5* datasets and word-level classifiers for *IMDB*, *Questions* and *Arabic* datasets. We use randomly initialized word/character embeddings which are trained along with the classification model parameters. For LSTM, we use the output at last timestep for prediction. For the Bi-LSTM, we combine the outputs of the first and last time step for prediction. For the Convolutional Neural Network we follow the same architecture as [112]. Additionally, to understand the effectiveness of adversarial reprogramming, we train a single layer CNN with convolutional filter width 5 and discuss the results of this experiment in Section 3.1.5.

Table 3.2. Test accuracy of various classification models. We use character-level models for *Names-5* and *Names-18* and word-level models for all other tasks. 1-CNN is a single layer CNN model with filter width 5.

Data Set	LSTM	Bi-LSTM	CNN	1-CNN
Names-18	97.84	97.84	**97.88**	74.04
Names-5	**99.88**	**99.88**	99.77	71.51
Questions	96.70	**98.25**	98.07	83.77
Arabic	87.25	**88.75**	88.00	74.75
IMDB	86.83	89.43	**90.02**	83.32

Experimental Setup

As described in the methodology section, the label remapping function f_L we use, is a one-to-one mapping between the labels of the original task and the adversarial task. Therefore it is required to apply the constraint that the number of classes of the adversarial task are less than or equal to the number of classes of the original task. We choose *Names-5, Arabic* and *Question Classification* as candidates for the adversarial tasks and repurpose the models allowed under this constraint. We use context size $k = 5$ for all our experiments.

In white-box attacks, we use the Gumbel-Softmax based approach described in the methodology to train the adversarial program. The details of the temperature annealing process are included in table 1 of the supplementary material. For black-box attacks, we use the REINFORCE algorithm described in methodology, on mini-batches of sequences. Since the action space for certain reprogramming problems, (eg. reprogramming of IMDB classifier) is large ($|V_S| = 10000$), we restrict the output of the adversarial program to most frequent 1000 tokens in the vocabulary V_S. We use Adam optimizer [113] for all our experiments. Hyperparameter details of all our experiments are included in table 1 of the supplementary material.

Table 3.3. Adversarial Reprogramming Experiments: The accuracies of white-box and black-box reprogramming experiments on different combinations of original task, adversarial task and model. White-box on Random Network column presents results of the white-box attack on an untrained neural network.

			Test Accuracy (%)		
Original Task	Adversarial Task	Victim Model	Black Box	White Box	White Box on Random Network
Questions	Names-5	LSTM	80.96	97.03	44.33
		Bi-LSTM	**93.51**	**99.66**	63.14
		CNN	88.90	99.22	93.06
Questions	Arabic	LSTM	73.50	87.50	50.00
		Bi-LSTM	81.75	83.50	70.00
		CNN	**82.25**	**87.25**	76.25
Names-18	Questions	LSTM	68.56	95.23	28.23
		Bi-LSTM	**94.96**	97.15	80.01
		CNN	71.03	**97.61**	33.45
Names-18	Arabic	LSTM	**83.00**	84.75	51.50
		Bi-LSTM	78.75	84.25	69.25
		CNN	80.75	**86.50**	60.00
IMDB	Arabic	LSTM	80.75	**88.25**	50.50
		Bi-LSTM	83.25	86.75	84.00
		CNN	**84.00**	87.00	84.25

3.1.5 Results and Discussion

The accuracies of all adversarial reprogramming experiments have been reported in Table 3.3. To interpret the results in context, the accuracies achieved by the LSTM, Bi-LSTM and CNN text classification models on the adversarial tasks can be found in table 3.1.

We demonstrate how character-level models trained on Names-18 dataset can be repurposed for word-level sequence classification tasks like Question Classification and Arabic Tweet Sentiment Classification. Similarly, word-level classifiers trained on Question Classification Dataset can be repurposed for the character-level Surname classification task. Interestingly, classifiers trained on IMDB Movie Review Dataset can be repurposed for Arabic Tweet Sentiment

Classification even though there is a high difference between the vocabulary size (10000 vs 955) and average sequence length(246.8 vs 9.7) of the two tasks. It can be seen that all of the three classification models are susceptible to adversarial reprogramming in both white-box and black-box setting.

White-box based reprogramming outperforms the black-box based approach in all of our experiments. In practice, we find that training the adversarial program in the black-box scenario requires careful hyper-parameter tuning for REINFORCE to work. We believe that improved reinforcement learning techniques for sequence generation tasks [104, 103] can make the training procedure for black-box attack more stable. We propose such improvement as a direction of future research.

To assess the importance of the original task on which the network was trained, we also present results of white-box adversarial reprogramming on untrained random network. Our results are coherent with similar experiments on adversarial reprogramming of untrained ImageNet models [91] demonstrating that adversarial reprogramming is less effective when it targets untrained networks. The figures in Table 3.3 suggest that the representations learned by training a text classifier on an original task, are important for repurposing it for an alternate task. However another plausible reason as discussed by reprogramming is that the reduced performance on random networks might be because of simpler reasons like poor scaling of network weight initialization making the optimization problem harder.

Complexity of Reprogramming Function

As motivated earlier in Section 3.1.3, computational efficiency of the adversarial program is critical for adversarial reprogramming to be of interest to an adversary. If the adversary can achieve the desired results using a computationally inexpensive classifier, it defeats the purpose of adversarial reprogramming. To understand if this is the case, we train a one-layer CNN with the same convolutional filter width as our adversarial program and average the activations across all time-steps to classify a given phrase. The results of such a classifier on various datasets have

been reported in Table 3.2. We can observe that our white-box attack on pre-trained networks, outperforms this classifier in all scenarios (refer to Table 3.3). Our best black-box attacks also outperform a one-layer CNN for all adversarial tasks. This experiment demonstrates that the reprogramming function exploits the learned feature representation of the victim model. Also, the observation that adversarial reprogramming is significantly less effective on randomly initialized untrained networks further reinforces the importance of utilizing a trained victim model.

Since the reprogramming function is a context based vocabulary remapping function, we can implement it as a look-up table that maps a combination of k tokens from the vocabulary V_t to a token in the source vocabulary V_s. The time complexity for transforming a sentence t to an adversarial sentence is just $O(length(t))$.

Adversarial Sequences

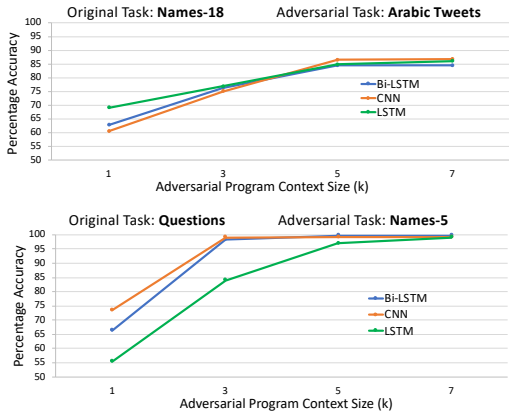
Figure 3.3 (a) shows some adversarial sequences generated by the adversarial program for Names-5 Classification while attacking a CNN trained on the Question Classification dataset. A sequence t in the first column is transformed into the adversarial sequence s in the second column by the trained adversarial reprogramming function. While these adversarial sequences may not make semantic or grammatical sense, it exploits the learned representation of the classifier to map the inputs to the desired class. For example, sequences that should be mapped to HUMAN class have words like *Who* in the generated adversarial sequence. Similarly, sequences that should be mapped to LOCATION class have words like *world, city* in the adversarial sequence. Other such interpretable transformations are depicted via colored text in the adversarial sequences of Figure 3.3 (a).

Effect of Context Size

By varying the context size k of the convolutional kernel $\theta_{k\times|V_T|\times|V_S|}$ in our adversarial program we are able to control the representational capacity of the adversarial reprogramming function. Figure 3.3 (b) shows the percentage accuracy obtained when training the adversar-

Adversarial Task Sequence (t) (Names-5)	Adversarial Program Output (s) (Question Classification)	Prediction by Classifier	Mapped Class	Actual Class
Ryoo	white sport substance animal All off ..	ENTITY	Korean	Korean
Houtum	player video exp abb What does off is off ..	ABBREVIATION	Dutch	Dutch
Winogrodzki	manner France manner video def oil def reason desc What do All off ..	DESCRIPTION	Polish	Polish
Murphy	world live exp city What university All is off ..	LOCATION	Scottish	Scottish
Paulissen	player stars along abb abb exp exp always abb What is off ..	ABBREVIATION	Dutch	Dutch
Henriques	other always live What Who does ind Who gold is off ..	HUMAN	Portuguese	Portuguese
Maly	world attend home abb home is off ..	LOCATION	Scottish	Polish
Kasprzak	does exp exp def manner does reason What does off ..	DESCRIPTION	Polish	Polish
Ferreiro	e-mail Who ind exp Who ind university university gold off ..	HUMAN	Portuguese	Portuguese
Hong	sport cremat substance university is off ..	ENTITY	Korean	Korean

(a)



(b)

Figure 3.3. (a) Adversarial sequences generated by our adversarial program for Names-5 Classification (adversarial task), when targeting CNN trained for Question Classification (original task). (b) Accuracy vs Context size (k) plots for 3 classification models on 2 different adversarial reprogramming tasks.

ial program with different context sizes k on two different adversarial tasks: Arabic Tweets Classification and Name Classification. Using a context size $k = 1$ reduces the adversarial reprogramming function to simply a vocabulary remapping function from V_S to V_T. It can be observed that the performance of the adversarial reprogramming model at $k = 1$ is significantly worse than that at higher values of k. While higher values of k improve the performance of the adversarial program, they come at a cost of increased computational complexity and memory required for the adversarial reprogramming function. For the adversarial tasks studied in this paper, we observe that $k = 5$ is a reasonable choice for context size of the adversarial program.

3.2 Cross-modal Adversarial Reprogramming

Transfer learning [96] and adversarial reprogramming [91] are two closely related techniques used for repurposing well-trained neural network models for new tasks. Neural networks when trained on a large dataset for a particular task, learn features that can be useful across multiple related tasks. Transfer learning aims at exploiting this learned representation for adapting a pre-trained neural network for an alternate task. Typically, the last few layers of a neural network are modified to map to a new output space, followed by fine-tuning the network parameters

on the dataset of the target task. Such techniques are especially useful when there is a limited amount of training data available for the target task.

Adversarial reprogramming shares the same objective as transfer learning with an additional constraint: the network architecture or parameters cannot be modified. Instead, the adversary can only adapt the input and output interfaces of the network to perform the new adversarial task. This more constrained problem setting of adversarial reprogramming poses a security challenge to neural networks. An adversary can potentially re-purpose cloud-hosted machine learning (ML) models for new tasks thereby leading to theft of computational resources. Additionally, the attacker may reprogram models for tasks that violate the code of ethics of the service provider. For example, an adversary can repurpose a cloud-hosted ML API for solving captchas to create spam accounts.

Prior works on adversarial reprogramming [91, 19, 114, 115] have demonstrated success in repurposing Deep Neural Networks (DNNs) for new tasks using computationally inexpensive input and label transformation functions. One interesting finding of [91] is that neural networks can be reprogrammed even if the training data for the new task has no resemblance to the original data. The authors empirically demonstrate this by repurposing ImageNet [116] classifiers on MNIST [117] digits with shuffled pixels showing that transfer learning does not fully explain the success of adversarial reprogramming. These results suggest that neural circuits hold properties that can be useful across multiple tasks which are not necessarily related. Hence neural network reprogramming not only poses a security threat, but also holds the promise of more reusable and efficient ML systems by enabling shared compute of the neural network backbone during inference time.

In existing work on adversarial reprogramming, the target adversarial task has the same data domain as the original task. Recent work has shown that network architectures based on the transformer model can achieve state-of-the-art results on language [118], audio [119] and vision [120] benchmarks suggesting that transformer networks serve as good inductive biases in various domains. Given this commonality between the neural architectures in different

domains, an interesting question that arises is whether we can perform cross-modal adversarial reprogramming: For example, Can we repurpose a vision transformer model for a language task?

Cross-modal adversarial reprogramming increases the scope of target tasks for which a neural network can be repurposed. In this work, we develop techniques to adversarially reprogram image classification networks for discrete sequence classification tasks. We propose a simple and computationally inexpensive adversarial program that embeds a sequence of discrete tokens into an image and propose techniques to train this adversarial program subject to a label remapping defined between the labels of the original and new task. We demonstrate that we can reprogram a number of image classification neural networks based on both Convolutional Neural Network (CNN) [121] and Vision Transformer [120] architectures to achieve competitive performance on a number of sequence classification benchmarks. Additionally, we show that it is possible to conceal the adversarial program as a perturbation in a real-world image thereby posing a stronger security threat. The technical contributions of this paper are summarized below:

- We propose Cross-modal Adversarial Reprogramming, a novel approach to repurpose ML models originally trained for image classification to perform sequence classification tasks. To the best of our knowledge, this is the first work that expands adversarial reprogramming beyond the data domain of the original task.

- We demonstrate the feasibility of our method by re-purposing four image classification networks for six different sequence classification benchmarks covering sentiment, topic, and DNA sequence classification. Our results show that a computationally-inexpensive adversarial program can leverage the learned neural circuits of the victim model and outperform word-frequency based classifiers trained from scratch on several tasks studied in our work.

- We demonstrate for the first time the threat imposed by adversarial reprogramming to the transformer model architecture by repurposing the Vision Transformer model for six

different sequence classification tasks. The reprogrammed transformer model outperforms alternate architectures on five out of six tasks studied in our work.

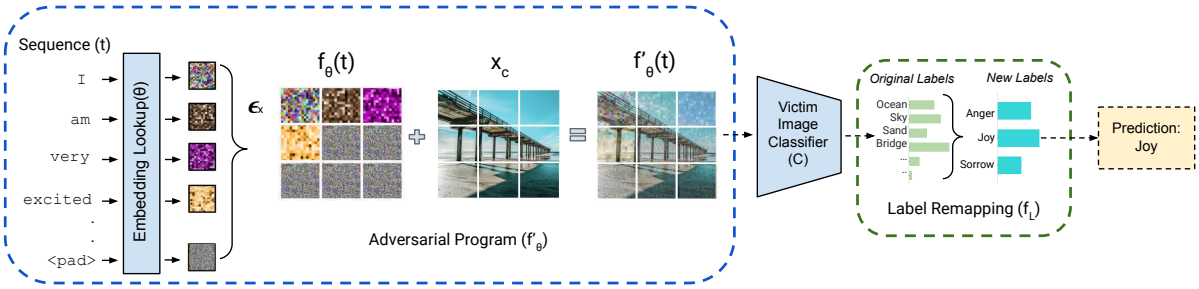


Figure 3.4. Schematic overview of cross-modal adversarial reprogramming method.

3.2.1 Background and Related Work

Adversarial Reprogramming

Neural networks have been shown to be vulnerable to adversarial examples [11, 9, 87, 40, 90, 122, 2, 123, 38, 39] which are slightly perturbed inputs that cause victim models to make a mistake. Adversarial Reprogramming was introduced by [91] as a new form of adversarial threat that allows an adversary to repurpose neural networks to perform new tasks, which are different from the tasks they were originally trained for. The proposed technique trains a single adversarial perturbation that can be added to all inputs in order to re-purpose the target model for an attacker's chosen task. The adversary achieves this by first defining a hard-coded one-to-one label remapping function that maps the output labels of the adversarial task to the label space of the classifier; and learning a corresponding adversarial reprogramming function that transforms an input from the input space of the new task to the input space of the classifier. The authors demonstrated the feasibility of their attack algorithm by reprogramming ImageNet classification models for classifying MNIST and CIFAR-10 data in a white-box setting, where the attacker has access to the victim model parameters.

While the above attack does not require any changes to the victim model parameters or architecture, the adversarial program proposed [91] is only applicable to tasks where the input

space of the the original and adversarial task is continuous. To understand the feasibility of attack in a discrete data domain, [19] proposed methods to repurpose *text* classification neural networks for alternate tasks, which operate on sequences from a discrete input space. The attack algorithm used a context-based vocabulary remapping method that performs a computationally inexpensive input transformation to reprogram a victim classification model for a new set of sequences. This work was also the first in designing algorithms for training such an input transformation function in both white-box and black-box settings—where the adversary may or may not have access to the victim model's architecture and parameters. They demonstrated the success of their proposed reprogramming functions by adversarially re-purposing various text-classification models including Long Short Term Memory networks (LSTM) [110], bi-directional LSTMs [111] and CNNs [124] for alternate text classification tasks.

Recent works [114, 115] have argued that reprogramming techniques can be viewed as an efficient training method and can be a superior alternative to transfer learning. Particularly [115] argue that one of the major limitations of current transfer learning techniques is the requirement of large amounts of target domain data, which is needed to fine-tune pre-trained neural networks. They demonstrated the advantage of instead using reprogramming techniques to repurpose existing ML models for alternate tasks, which can be done even when training data is scarce. The authors designed a black-box adversarial reprogramming method, that can be trained iteratively from input-output model responses, and demonstrated its success in repurposing ImageNet models for medical imaging tasks such as classification of autism spectrum disorders, melanoma detection, etc.

All of these existing reprogramming techniques are only able to reprogram ML models when the data domain of the target adversarial task and the original task are the same. We address this limitation in our work by designing adversarial input transformation functions that allow image classification models to be reprogrammed for sequence classification tasks such as natural language and protein sequence classification.

Transformers and Image Classifiers

While Convolutional Neural Networks (CNNs) have long achieved state-of-the-art performance on vision benchmarks, the recently proposed Vision Transformers (ViTs) [120] have been shown to outperform CNNs on several image classification tasks. Transformers [118] are known for achieving state-of-the-art performance in natural language processing (NLP). In order to train transformers for image classification tasks, the authors [120] divided an image into patches and provide the sequence of linear embeddings of these patches as an input to a transformer. Image patches are treated the same way as tokens (words) in an NLP application and the model is trained on image classification in a supervised manner. The authors report that when ViTs are trained on large-scale image datasets, they are competitive and also outperform state-of-the-art models on multiple image recognition benchmarks.

Since transformers can model both language and vision data in a similar manner, that is, as a sequence of embeddings, we are curious to investigate whether a vision transformer can be reprogrammed for a text classification task. In the process, we find that CNN network architectures can also be reprogrammed to achieve competitive performance on discrete sequence classification tasks. In the next section, we discuss our cross-modal adversarial reprogramming approach.

3.2.2 Methodology

Problem Definition

Consider a victim image classifier C trained for mapping images $x \in X$ to a label $l_X \in L_X$. That is,

$$C : x \mapsto l_X$$

An adversary wishes to repurpose this victim image classifier for an alternate text classification task C' of mapping sequences $t \in T$ to a label $l_T \in L_T$. That is,

$$C' : t \mapsto l_T$$

To achieve this goal, the adversary needs to learn appropriate mapping functions between the input and output spaces of the original and the new task. We solve this by first defining a label remapping f_L that maps label spaces of the two tasks: $f_L : l_X \mapsto l_T$; and then learning a corresponding adversarial program f_θ that maps a sequence $t \in T$ to an image $x \in X$ i.e., $f_\theta : t \mapsto x$ such that $f_L(C(f_\theta(t)))$ acts as the target classifier C'.

We assume a white-box adversarial reprogramming setting where the adversary has complete knowledge about architecture and model parameters of the victim image classifier. In the next few sections we describe the adversarial program f_θ, the label remapping function and the training procedure to learn the adversarial program.

Cross-modal Adversarial Program

The goal of our adversarial program is to map a sequence of discrete tokens $t \in T$ to an image $x \in X$. Without loss of generalizability, we assume $X = [-1,1]^{h \times w \times c}$ to be the scaled input space of the image classifier C where h, w are the height and width of the input image and c is the number of channels. The tokens in the sequence t belong to some vocabulary list V_T. We can represent the sequence t as $t = t_1, t_2, \ldots, t_N$ where t_i is the vocabulary index of the i_{th} token in sequence t in the vocabulary list V_T.

When designing the adversarial program it is important to consider the computational cost of the reprogramming function f_θ. This is because if a classification model that performs equally well can be trained from scratch for the classification task C' and is computationally cheaper than the reprogramming function, it would defeat the purpose of adversarial reprogramming.

Keeping the above in mind, we design a reprogramming function that looks up embed-

dings of the tokens t_i and arranges them as contiguous patches of size $p \times p$ in an image that is fed as input to the classifier C. Mathematically, the reprogramming function f_θ is parameterized by a learnable embedding tensor $\theta_{|V_T| \times |p| \times |p| \times |c|}$ and performs the transformation $f_\theta : t \mapsto x$ as per Algorithm 3.

Algorithm 3. Cross-modal Adversarial Program f_θ

Input: Sequence $t = t_1, t_2, \ldots, t_N$
Output: Reprogrammed image $x_{h \times w \times c}$
Parameters: Embedding tensor $\theta_{|V_T| \times |p| \times |p| \times |c|}$
$x \leftarrow 0_{h \times w \times c}$
for each t_k in t **do**
  $i \leftarrow \lfloor (k \times p)/h \rfloor$
  $j \leftarrow (k \times p) \bmod w$
  $x[i : i+p, j : j+p, :] \leftarrow \tanh(\theta[t_k, :, :, :])$
return x

The patch size p and image dimensions h, w determine the maximum length of the sequence t that can be encoded into the image. We pad all the input sequences t all the way up to the maximum allowed sequence length with a padding token to fill up the reprogrammed image and clip any sequences longer than the maximum allowed length from the end. More details about the hyper-parameters can be found in our experiments section.

Concealing the adversarial perturbation: Most past works on adversarial reprogramming have considered an unconstrained attack setting, where the reprogrammed image does not necessarily need to resemble a real-world image. However, as noted by [91], it is possible to conceal the reprogrammed image in a real-world image by constraining the output of the reprogramming function. We can conceal the reprogrammed image as an additive perturbation to some real-world base image x_c by defining an alternate reprogramming function f'_θ as follows:

$$f'_\theta(t) = Clip_{[-1,1]}(x_c + \varepsilon . f_\theta(t)) \tag{3.5}$$

Since the output of the original reprogramming function f_θ is bounded between $[-1, 1]$, we can control the L_∞ norm of the added perturbation using the parameter $\varepsilon \in [0, 1]$.

Computational Complexity: As depicted in Figure 8.2, during inference, the adversarial program only looks up embeddings of the tokens in the sequence t and arranges them in an image tensor which can optionally be added onto a base image. Asymptotically, the time complexity of this adversarial program is linear in terms of the length of the sequence t. Since there are no matrix-vector multiplications involved in the adversarial program, it is computationally equivalent to just the embedding layer of a sequence-based neural classifier. Therefore the inference cost of the adversarial program is significantly less than that of a sequence-based neural classifier. Table 1 in our supplementary material compares the wall-clock inference time for a sequence of length 500 for our adversarial program and various sequence-based neural classifiers used in our experiments.

Label Remapping and Optimization Objective

Past work [91, 19, 115] on adversarial reprogramming assume that the number of labels in the target task are less than than the number of labels in the original task. In our work, we relax this constraint and propose label remapping functions for both of the following scenarios:

1. Target task has fewer labels than the original task: Initial works on adversarial reprogramming defined a one-to-one mapping between the labels of the original and new task [91, 19]. However, recent work [115] found that mapping multiple source labels to one target label helps improve the performance over one-to-one mapping. Our preliminary experiments on cross-modal reprogramming confirm this finding, however, we differ in the way the final score of a target label l_t is aggregated—[115] obtained the final score for a target label as the mean of the scores of the mapped original labels. We found that aggregating the score by taking the maximum rather than the mean over the mapped original labels leads to faster training. Another advantage of using max reduction is that during inference, we can directly map the original predicted label to our target label without requiring access to probability scores of any other label.

Consider a target task label l_t, mapped to a subset of labels $L_{S_t} \subset L_S$ of the original task

under the many-to-one label remapping function f_L. We obtain the score for this target task label as the maximum of the scores of each label $l_i \in L_{S_t}$ by classifier C. That is,

$$Z'_{l_t}(t) = \max_{l_i \in L_{S_t}} Z_{l_i}(f_\theta(t)), \tag{3.6}$$

where $Z_k(x)$ and $Z'_k(t)$ represent the score (before softmax) assigned to some label k by classifier C and C' respectively.

To define the label remapping f_L, instead of randomly assigning m source labels to a target label, we first obtain the model predictions on the base image x_c (or a zero image in the case of an unbounded attack) and sort the labels by the obtained scores; We then assign the the highest scored source labels to each target label using a round-robin strategy until we have assigned m source labels to each target label.

Note that while we need access to individual class scores during training (where we assume a white-box attack setting), during inference we can simply map the highest predicted label to the target label using the label remapping function f_L without having to know the actual scores assigned to different labels.

2. Original task has fewer labels than the target task: In this scenario, we map the probability distribution over the original labels to a distribution over target labels to class scores for the target label space using a learnable linear transformation. That is,

$$Z'(t) = \theta'_{|L_T| \times |L_X|} \cdot softmax(Z(f_\theta(t))). \tag{3.7}$$

Here $Z'(t)$ is a vector representing class scores (logits) for the target label space. $\theta'_{|L_T| \times |L_X|}$ are the learnable parameters of the linear transformation that are optimized along with the parameters of the reprogramming function f_θ. Note that unlike the previous scenario, in this setting, we assume that we have access to the probability scores of the original labels during both training and inference.

Optimization Objective: To train the parameters θ of our adversarial program, we use a cross-entropy loss between the target label and the model score predictions obtained as per Equation 3.6 or Equation 3.7. We also incorporate an L_2 regularization loss for better generalization on the test set and to encourage more imperceptible perturbation in the case of our bounded attack. Therefore our final optimization objective is the following:

$$P_{l_t} = softmax(Z'(t))_{l_t}$$
$$E(\theta) = -\sum_{t \in T} \log(P_{l_t}) + \lambda ||\theta||_2^2.$$

Here λ is the regularization hyper-parameter and P_{l_t} is the predicted class probability of the correct label l_t for sequence t. We use mini-batch gradient descent using an Adam optimizer [113] to solve the above optimization problem on the dataset of the target task.

3.2.3 Experiments

Victim Image Classifiers

To demonstrate cross-modal adversarial reprogramming, we perform experiments on four neural architectures trained on the ImageNet dataset. We choose both CNNs and the recently proposed Vision Transformers (ViT) [120] as our victim image classifiers. While CNNs have long achieved state-of-the-art performance on computer-vision benchmarks, the recently proposed ViTs have been shown to outperform CNNs on several image classification tasks. We choose the ViT-Base [120], ResNet-50 [125], InceptionNet-V3 [126] and EfficientNet-B7 [127] architectures. The details of these architectures are listed in Table 3.4. We perform experiments on both pre-trained and randomly initialized networks.

Datasets and Reprogramming Tasks

In this work, we repurpose the aforementioned image classifiers for several discrete sequence classification tasks. We wish to analyze the performance of cross-modal adversarial re-

Table 3.4. Victim image classification networks used for adversarial reprogramming experiments. We include the number of parameters of each model and also the Top-1 and Top-5 test accuracy achieved on the ImageNet benchmark.

				Accuracy (%)	
Model	Abbr.	Type	# Params	Top-1	Top-5
ViT-Base	ViT	Transformer	86.9M	84.2	97.2
ResNet-50	RN-50	CNN	25.6M	79.0	94.4
InceptionNet-V3	IN-V3	CNN	23.8M	77.5	93.5
EfficientNet-B4	EN-B4	CNN	19.3M	83.0	96.3

Table 3.5. Statistics of the datasets used for reprogramming tasks. We also include the test accuracy of both neural network based and TF-IDF based benchmark classifiers trained from scratch on the train set.

	Dataset Statistics						*Accuracy (%)*			
						Avg	*Neural Methods*		*TF-IDF*	
Dataset	Task Type	# Classes	# Train	# Test	Token	Length	Bi-LSTM	1D-CNN	unigram	n-gram
Yelp	Sentiment	2	560,000	38,000	word	135.6	95.94	95.18	92.50	92.93
IMDB	Sentiment	2	25,000	25,000	word	246.8	89.43	90.02	88.52	88.43
AG	Topic	4	120,000	7,600	word	57.0	91.45	92.09	90.92	90.69
DBPedia	Topic	14	560,000	70,000	word	47.1	97.78	98.09	97.12	97.16
Splice	DNA	3	2,700	490	neucleobase	60.0	93.26	83.87	51.42	72.24
H3	DNA	2	13,468	1,497	neucleobase	500.0	86.84	85.43	75.68	78.89

programming for different applications such as understanding language and analyzing sequential biomedical data. Biomedical datasets e.g. splice-junction detection in genes, often have fewer training samples than language based datasets and we aim to understand whether such limitations can adversely affect our proposed reprogramming technique.

Sentiment analysis and topic classification are popular NLP tasks. However, analyzing the underlying semantics of the sequence is often not necessary for solving these tasks since word-frequency based statistics can serve as strong discriminatory features. In contrast, tasks like DNA-sequence classification requires analyzing the sequential semantics of the input and simple frequency analysis of the unigrams or n-grams does not achieve competitive performance on these tasks. To evaluate the effectiveness of adversarial reprogramming in both of these

scenarios, we consider the following tasks and datasets in our experiments:

Sentiment Classification Datasets

1. Yelp Polarity Dataset **(Yelp)** [124]: This is a dataset consisting of reviews from Yelp for the task of sentiment classification, categorized into binary classes of positive and negative sentiment.

2. Large Movie Review Dataset **(IMDB)** [109]: This is a dataset for binary sentiment classification of positive and negative sentiment from highly polar IMDB movie reviews.

Topic Classification Datasets

1. AG's News Dataset **(AG)** [124]: is a collection of more than 1 million news articles. News articles have been gathered from more than 2000 news sources and contains 4 classes: *World, Sports, Business, Sci/Tech*.

2. DBPedia Ontology Dataset **(DBPedia)** [124]: consists of 14 non-overlapping categories from DBpedia 2014. The samples consist of the category and abstract of each Wikipedia article.

DNA Sequence Classification Datasets

1. Splice-junction Gene Sequences (**Splice**): This dataset [128, 129] was curated for training ML models to detect splice junctions in DNA sequences. In DNA, there are two kinds of splice junction regions: Exon-Intron (EI) junction and Intron-Exon (IE) junction. This dataset contains sample DNA sequences of 60 base pair length categorized into 3 classes: "EI" which contains exon-intron junction, "IE" which contains intron-exon junction, and "N" which contain neither EI or IE regions.

2. Histone Protein Occupancy in DNA (**H3**): This dataset from [130, 131] indicates whether certain DNA sequences wrap around H3 histone proteins. Each sample is a sequence with a length of 500 neucleobases. Positive samples contain DNA regions wrapping around histone proteins while negative samples do not contain such DNA regions.

The statistics of these datasets are included in Table 3.5. To benchmark the performance that can be achieved on these tasks, we train various classifiers from scratch on the datasets for

each task. We consider both neural network based classification models and frequency-based statistical models (such as TF-IDF) as our benchmarks. We use word-level tokens for sentiment and topic classification tasks and neucleobase level tokens for DNA sequence classification tasks.

The TF-IDF methods can work on either unigrams or n-grams for creating the feature vectors from the input data. For the n-gram model, we consider n-grams up to length 3 and choose the value of n that achieves the highest classification accuracy on the hold-out set. We train a Stochastic Gradient Descent (SGD) classifier to classify the feature vector as one of the target classes. Additionally, we train DNN based text-classifiers: Bidirectional Long Short Term Memory networks (Bi-LSTM) [111, 110] and 1D CNN [112] models from scratch on the above tasks. We use randomly initialized token embeddings for all classification models, which are trained along with the network parameters. For Bi-LSTMs, we combine the outputs of the first and last time step for prediction. For the Convolutional Neural Network we follow the same architecture as [112]. The hyper-parameter details of these classifiers and architecture have been included in Table 2 of the supplementary material.

We report the accuracies on the test set of the above mentioned classifiers in Table 3.5. We find that while both neural and frequency based TF-IDF methods work well on sentiment and topic classification tasks, neural networks significantly outperform frequency based methods on DNA sequence classification tasks. This is presumably because the latter require structural analysis of the sequence rather than relying on keywords.

Experimental Details

Input image size and patch size: The ViT-Base model utilized in our work is trained on images of size 384×384 and works on image patches of size 16×16. For all our experiments, we fix the input image size to be 384×384. When we use a patch of size 16×16 for encoding a single token in our sequence, it allows for a maximum of 576 tokens to be encoded into a single image. In our initial experiments we found that using larger patch sizes for smaller sequences leads to higher performance on the target task, since it encodes a sequence in a spatially larger

area of the image. Therefore, we choose our patch size as the largest possible multiple of 16 that can encode the longest sequence in our target task dataset. We list the patch size p used for different tasks in Table 3.6.

Table 3.6. Results (% Accuracy on the test set) of adversarial reprogramming experiments targeting four image classification models for six sequence classification tasks.

		Unbounded								*Bounded ($L_\infty = 0.1$)*			
		Pre-trained				*Randomly Initialized*				*Pre-Trained*			
Task	p	ViT	RN-50	IN-V3	EN-B4	ViT	RN-50	IN-V3	EN-B4	ViT	RN-50	IN-V3	EN-B4
Yelp	16	92.82	**93.29**	89.19	93.47	92.73	68.50	65.56	52.97	88.57	81.32	81.33	81.23
IMDB	16	86.76	85.60	80.67	87.26	**88.38**	81.08	52.87	50.26	82.07	72.28	71.22	81.42
AG	16	**91.59**	89.88	89.78	90.46	91.45	82.37	50.43	24.87	86.49	83.26	78.93	84.03
DBPedia	32	**97.62**	96.31	95.70	96.77	97.56	30.12	52.87	19.61	92.79	80.64	81.46	79.53
Splice	48	**95.31**	94.48	95.10	92.04	54.13	48.57	91.22	50.20	95.10	94.27	94.89	91.55
H3	16	**82.57**	78.16	80.29	80.16	77.02	73.00	64.20	51.17	76.62	72.01	75.55	75.42

Training hyper-parameters: We train each adversarial program on a single Titan 1080i GPU using a batch size of 4. We set the learning rate as 0.001 for the unbounded attacks and $0.001 \times \varepsilon^{-1}$ for our bounded attacks (Equation 3.5). We set the L_2 regularization hyper-parameter $\lambda = 1e-4$ for all our experiments and train the adversarial program for a maximum 100k mini-batch iterations in the unbouned attack setting and for 200k mini-batch iterations in the bounded attack setting. We map 10 original labels to each target label in the scenario when there are fewer labels for the target task than for the original task. We point the readers to our codebase for precise implementation.[3]

3.2.4 Results

Pre-trained vs untrained victim models

Experimental results of our proposed cross-modal reprogramming method are reported in Table 3.6. In these experiments, the original task has more labels than the target task so we

[3]https://github.com/paarthneekhara/multimodal_rerprogramming

use the label remapping function given by Equation 3.6. We first consider the unbounded attack setting, where the output of the adversarial program does not need to be concealed in a real-world image. For these experiments, we use the reprogramming function f_θ described in Algorithm 3. We also note that the primary evaluation of past reprogramming works [91, 19, 115] is done in an unbounded attack setting.

When attacking pre-trained image classifiers, we achieve competitive performance (as compared to benchmark classifiers trained from scratch, reported in Table 3.5) across several tasks for all victim image classification models. To assess the importance of pre-training the victim model on the original dataset, we also experiment with reprogramming untrained randomly initialized networks. Randomly initialized neural networks can potentially have rich structure which the reprogramming functions can exploit. Prior works [132, 133] have shown that wide neural networks can behave as Gaussian processes, where training specific weights in the intermediate layers is not necessary to perform many different tasks. However, in our experiments, we find that for CNN-based image classifiers, reprogramming pre-trained neural networks performs significantly better than reprogramming randomly initialized networks for all tasks. This is consistent with the findings of prior reprogramming work [91] which reports that adversarial reprogramming in the image domain is more effective when it targets pre-trained CNNs. For the ViT model, we find that we are able to obtain competitive performance on sentiment and topic classification tasks when reprogramming either randomly initialized or pre-trained models. Particularly, we find that reprogramming untrained vision transformers provides the highest accuracy on the IMDB classification task. However, for DNA sequence classification tasks (Splice and H3) that require structural analysis of the sequence rather than token-frequency statistics, we find that reprogramming pre-trained vision transformer model performs significantly better than a randomly initialized transformer model.

The ViT model outperforms other architectures on 5 out of 6 tasks in the unbounded attack setting. In particular, for the task of splice-junction detection in gene sequences, reprogramming a pre-trained ViT model outperforms both TF-IDF and neural classifiers trained from scratch.

For sentiment analysis and topic classification tasks, which primarily require keyword detection, some reprogramming methods achieve competitive performance as the benchmark methods reported in Table 3.5.

Additionally, to assess the importance of the victim classifier for solving the target task, we study the extent to which the task can be solved without the victim classifier and using only the adversarial reprogramming function with a linear classification head. We present the results and details of this experiment in Table 3 of our supplementary material.

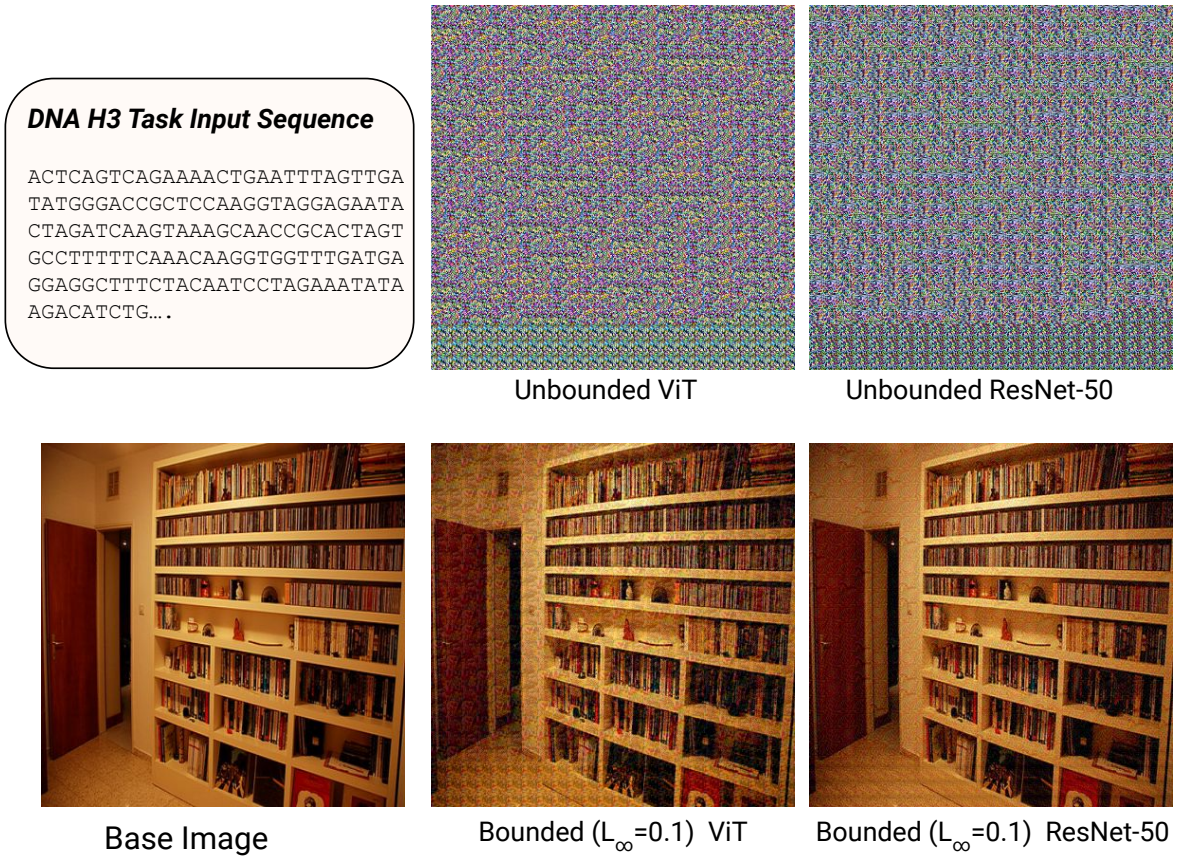


Figure 3.5. Example outputs of our adversarial reprogramming function in both unbounded (top) and bounded (bottom) attack settings while reprogramming two different pre-trained image classifiers for a DNA sequence classification task (H3).

Concealing the adversarial perturbation: To conceal the output of the adversarial program in a real-world image, we follow the adversarial reprogramming function defined in Equation 3.5. We randomly select an image from the ImageNet dataset (shown in Figure 3.5) as

the base image x_c and train adversarial programs targeting different image classifiers for the same base image. We present the results at $L_\infty = 0.1$ (on a 0 to 1 pixel value scale) distortion between the reprogrammed image and the base image x_c on the right side of Table 3.6. It can be seen that for some drop in performance, it is possible to perform adversarial reprogramming such that the input sequence is concealed in a real-world image. Figure 1 in our supplementary material shows the accuracy on three target tasks for different magnitudes of allowed perturbation, while reprogramming a pre-trained ViT model.

Target task has more labels than original task

In a practical attack scenario, the adversary may only have access to a victim image classifier with fewer labels than the target task labels. To evaluate adversarial reprogramming in this scenario, we constrain the adversary's access to the class-probability scores of just q labels of the ImageNet classifier. We choose the most frequent q ImageNet labels as the original labels, that can be accessed by the adversary; and perform our experiments on two tasks from our datasets, which have the highest number of labels—AG News (4 labels) and DBPedia (14 labels). We use the label remapping function given by Equation 3.7, and learn a linear transformation to map the predicted probability distribution over the q original labels to the target task label scores.

We demonstrate that we are able to perform adversarial reprogramming even in this more constrained setting. We achieve similar performance as compared to our many-to-one label remapping scenario reported in Table 3.6 when q is close to the number of labels in the target task. This is because we learn an additional mapping function for the output interface, which can potentially lead to better optimization. However as a downside, this setting requires access to all q class probability scores for predicting the adversarial label, while in the previous many-to-one label remapping scenario, we only need to know the highest-scored original label for mapping it to one of the adversarial labels.

Table 3.7. Results of adversarial reprogramming when the target task has more labels than the original task. The access of the adversary is constrained to class-probabilities of q labels of the original (ImageNet) task. This evaluation is done on pre-trained networks in an unbounded attack setting.

			Accuracy (%)			
Dataset	# Labels	q	ViT	RN-50	IN-V3	EN-B4
AG	4	3	**89.42**	87.18	86.66	89.18
DBPedia	14	3	**96.34**	83.16	84.17	92.95
DBPedia	14	10	**98.01**	96.84	94.88	97.16

3.2.5 Conclusion

In this work, we extend adversarial reprogramming, a new class of adversarial attacks, to target text classification neural networks. Our results demonstrate the effectiveness of such attacks in the more challenging black-box settings, posing them as a strong threat in real-world attack scenarios. We demonstrate that neural networks can be effectively reprogrammed for alternate tasks, which were not originally intended by a service provider. Our proposed end-to-end approach can be used to further understand the vulnerabilities and blind spots of deep neural network based text classification systems. We recommend future work to study the scope of adversarial reprogramming for other NLP applications such as machine translation, text to speech synthesis and text to image synthesis where the input space is discrete.

We propose Cross-modal Adversarial Reprogramming, which for the first time demonstrates the possibility of repurposing pre-trained image classification models for sequence classification tasks. We demonstrate that computationally inexpensive adversarial programs can repurpose neural circuits to non-trivially solve tasks that require structural analysis of sequences. Our results suggest the potential of training more flexible neural models that can be reprogrammed for tasks across different data modalities and data structures. More importantly, this work reveals a broader security threat to public ML APIs that warrants the need for rethinking existing security primitives.

3.3 Acknowledgements

Chapter 3 is a reprint of the material as it appears in two papers (1) *Adversarial Reprogramming of Text Classification Neural Networks*. Empirical Methods in Natural Language Processing, 2019. Neekhara, Paarth; Hussain, Shehzeen; Dubnov, Shlomo; Koushanfar, Farinaz. (2) *Cross-modal Adversarial Reprogramming*. IEEE Winter Conference on Applications of Computer Vision, 2022. Neekhara, Paarth; Hussain, Shehzeen; Du, Jinglong; Dubnov, Shlomo; Koushanfar, Farinaz; McAuley, Julian. The dissertation author and Paarth Neekhara made equal contributions to this work.

Chapter 4

WaveGuard: Understanding and Mitigating Audio Adversarial Examples

The recent surge in adversarial attacks on deep learning based automatic speech recognition (ASR) systems threatens the wide-scale adoption of these technologies in safety-critical applications. In this chapter, I describe our proposed solution WaveGuard: a framework for detecting adversarial inputs that are crafted to attack ASR systems. Our framework incorporates audio transformation functions and analyses the ASR transcriptions of the original and transformed audio to detect adversarial inputs.[1] We demonstrate that our defense framework is able to reliably detect adversarial examples constructed by four recent audio adversarial attacks, with a variety of audio transformation functions. With careful regard for best practices in defense evaluations, we analyze our proposed defense and its strength to withstand adaptive and robust attacks in the audio domain. We empirically demonstrate that audio transformations that recover audio from perceptually informed representations can lead to a strong defense that is robust against an adaptive adversary even in a complete white-box setting. Furthermore, WaveGuard can be used out-of-the box and integrated directly with any ASR model to efficiently detect audio adversarial examples, without the need for model retraining.

Speech serves as a powerful communication interface between humans and machine learning agents. Speech interfaces enable hands-free operation and can assist users who are

[1] Audio Examples: https://waveguard.herokuapp.com

visually or physically impaired. Research into machine recognition of speech is driven by the prospect of offering services where humans interact naturally with machines. To this end, automatic speech recognition (ASR) systems seek to accurately convert a speech signal into a transcription of the spoken words, irrespective of a speaker's accent, or the acoustic environment in which the speaker is located [134]. With the advent of deep learning, state-of-the-art speech recognition systems [69, 135, 71] are based on Deep Neural Networks (DNNs) and are widely used in personal assistants and home electronic devices (e.g. Apple Siri, Google Assistant).

The popularity of ASR systems has brought new security concerns. Several studies have demonstrated that DNNs are vulnerable to adversarial examples [25, 20, 13, 26, 9]. While previously limited to the image domain, recent attacks on ASR systems [74, 80, 1, 75, 76, 3, 2, 136, 137], have demonstrated that adversarial examples also exist in the audio domain. An audio adversarial example can cause the original audio signal to be transcribed to a target phrase desired by the adversary or can cause significant transcription error by the victim ASR model.

Due to the existence of these vulnerabilities, there is a crucial need for defensive methods that can be employed to thwart audio adversarial attacks. In the image domain, several works have proposed input transformation based defenses [138, 139, 140, 141, 142] to recover benign images from adversarially modified images. Such inference-time adversarial defenses use image transformations like feature squeezing, JPEG compression, quantization, randomized smoothing (etc.) to render adversarial examples ineffective. While such defenses are effective in guarding against non-adaptive adversaries, they can be bypassed in an adaptive attack scenario where the attacker has partial or complete knowledge about the defense.

Another line of defense in the image domain is based on training more robust neural networks using adversarial training or by introducing randomization in network layers and parameters. Such defenses are comparatively more robust under adaptive attack scenarios, however they are significantly more expensive to train as compared to input transformation based defenses that can be employed directly at the model inference stage. Although input transformation based defenses are shown to be broken for image classifiers, the same conclusion

cannot be drawn for ASR systems without careful evaluation. This is because an ASR system is a more complicated architecture as compared to an image classification model and involves several individual components: an acoustic feature extraction pipeline, a neural sequence model for processing the time-series data and a language head for predicting the language tokens. This pipeline makes it challenging to craft robust adversarial examples for ASR systems that can reliably transcribe to a target phrase even when the input is transformed and reconstructed from some perceptually informed representation.

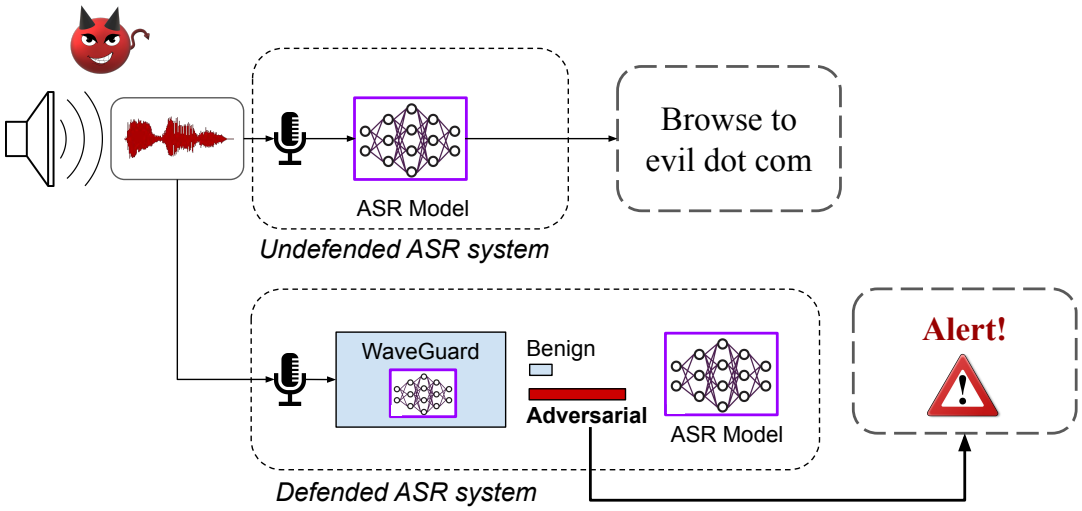


Figure 4.1. Depiction of an undefended ASR system and an ASR system defended by WaveGuard in the presence of a malicious adversary. The ASR system defended by WaveGuard detects the adversarial input and alerts the user.

WaveGuard: In this section, we analyze the effectiveness of audio transformation based defenses for detecting adversarial examples for speech recognition systems. We first design a general framework for employing audio transformation functions as an adversarial defense for ASR systems. Our framework transforms the given audio input x using an input transformation function g and analyzes the ASR transcriptions for the input x and $g(x)$. The underlying idea for our defense is that model predictions for adversarial examples are unstable while those for benign examples are robust to small changes in the input. Therefore, our framework labels an input as adversarial if there is a significant difference between the transcriptions of x and $g(x)$.

We first study five different audio transformations under different compression levels against non-adaptive adversaries. We find that at optimal compression levels, most input transformations can reliably discriminate between adversarial and benign examples for both targeted and untargeted adversarial attacks on ASR systems. Furthermore, we achieve higher detection accuracy in comparison to prior work [143, 144] in adversarial audio detection. However, this evaluation does not provide security guarantees against a future adaptive adversary who has knowledge of our defense framework. To evaluate the robustness of our defense against an adaptive adversary, we propose a strong white-box adaptive attack against our proposed defense framework. Interestingly, we find that some input transformation functions are robust to adaptive attack even when the attacker has complete knowledge of the defense. Particularly, the transformations that recover audio from perceptually informed representations of speech prove to be more effective against adaptive-attacks than naive audio compression and filtering techniques. Following are the explicit contributions of this work:

- We develop a formal defense framework (Section 4.2) for detecting audio adversarial examples against ASR systems. Our framework uses input transformation functions and analyses the transcriptions of original and transformed audio to label the input as adversarial or benign.

- We evaluate different transformation functions for detecting recently proposed and highly successful targeted [1, 3] and untargeted [2] attacks on ASR systems. We study the trade-off between the hyper-parameters of different transformations and the detector performance and find an optimal range of hyper-parameters for which the given transformation can reliably detect adversarial examples (Section 4.5).

- We demonstrate the robustness of our defense framework against an adaptive adversary who has complete knowledge of our defense and intends to bypass it. We find that certain input transformation functions that reduce audio to a perceptually informed representation cannot be easily bypassed under different allowed magnitudes of perturbations. Particularly,

we find that Linear Predictive Coding (LPC) and Mel spectrogram inversion are more robust to adaptive attacks as compared to other transformation functions studied in our work (Section 4.6).

- We investigate transformation functions for the goal of recovering the original transcriptions from an adversarial signal. We find that for certain attacks and transformation functions, we can recover the original transcript with a low Character Error Rate. (Section 4.5.2)

4.1 Background and Related Work

4.1.1 Adversarial Attacks in the Audio Domain

Adversarial attacks on ASR systems have primarily focused on *targeted attacks* to embed carefully crafted perturbations into speech signals, such that the victim model transcribes the input audio into a specific malicious phrase, as desired by the adversary [74, 1, 77, 75, 78]. Such attacks can for example cause a digital assistant to incorrectly recognize commands it is given, thereby compromising the security of the device. Prior works [75, 78] demonstrate successful attack algorithms targeting traditional speech recognition models based on HMMs and GMMs [63, 64, 65, 66, 67, 68]. For example, in Hidden Voice Commands [75], the attacker uses inverse feature extraction to generate obfuscated audio that can be played over-the-air to attack ASR systems. However, obfuscated samples sound like random noise rather than normal human perceptible speech and therefore come at the cost of being fairly perceptible to human listeners.

In more recent work [1] involving neural network based ASR systems, Carlini *et al.* propose an end-to-end white-box attack technique to craft adversarial examples, which transcribe to a target phrase. Similar to work in images, they propose a gradient-based optimization method that replaces the cross-entropy loss function used for classification, with a Connectionist Temporal Classification (CTC) loss [79] which is optimized for time-sequences. The CTC-loss

between the target phrase and the network's output is backpropagated through the victim neural network and the Mel Frequency Cepstral Coefficient (MFCC) computation, to update the additive adversarial perturbation. The authors in this work demonstrate 100% attack success rate on the Mozilla DeepSpeech [71] ASR model. The adversarial samples generated by this work are quasi-perceptible, motivating a separate work [80] to minimize the perceptibility of the adversarial perturbations using psychoacoustic hiding. Further addressing the imperceptibility of audio attacks, Qin *et al.* [3] develop effectively imperceptible audio adversarial examples by leveraging the psychoacoustic principle of auditory masking. In their work [3], the imperceptibility of adversarial audio is verified through a human study, while retaining 100% targeted attack success rate on the Google Lingvo [135] ASR model.

Targeted attacks, such as those described above, cannot be performed in real-time since it requires the adversary to solve a data-dependent optimization problem for each data-point they wish to mis-transcribe. To perform attacks in real-time, the authors of [2] designed an algorithm to find a single quasi-imperceptible universal perturbation, which when added to any arbitrary speech signal, causes mis-transcription by the victim speech recognition model. The proposed algorithm iterates over the training dataset to build a universal perturbation vector, that can be added to any speech waveform to cause an error in transcription by a speech recognition model with high probability. This work also demonstrates transferability of adversarial audio samples across two different ASR systems (based on DeepSpeech and Wavenet), demonstrating that such audio attacks can be performed in real-time even when the attacker does not have knowledge of the ASR model parameters.

Physical attacks. Adversarial attacks to ASR Systems have also been demonstrated to be a real-world threat. In particular, recently developed attack algorithms have shown success in attacking physical intelligent voice control (IVC) devices, when playing the generated adversarial examples over-the-air. The recently developed *Devil's Whisper* [137] demonstrated that adversarial commands embedded in music samples and played over-the-air using speakers, are able to attack popular IVC devices such as Google Home, Google Assistant, Microsoft Cortana

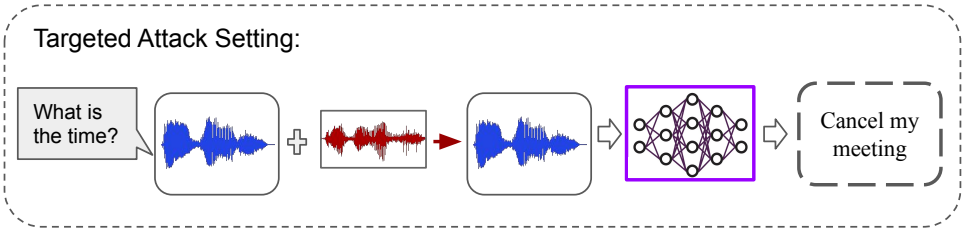


Figure 4.2. In the targeted attack setting the adversary solves a data-dependent optimization problem to find an additive perturbation, such that a victim ASR model transcribes the adversarial input audio to a target phrase as desired by the adversary.

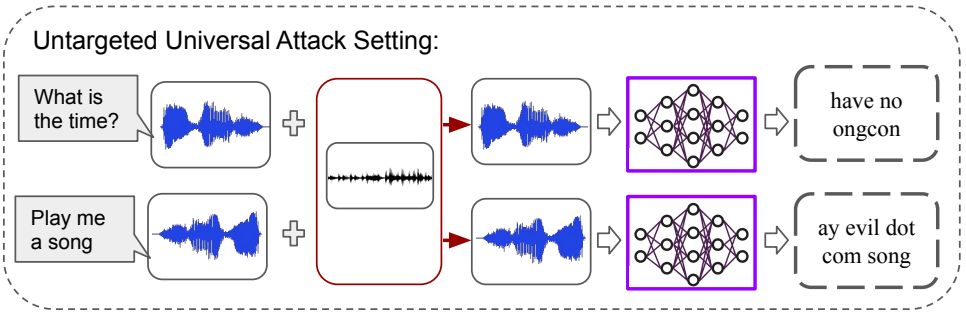


Figure 4.3. In untargeted universal attacks the adversary computes a single universal perturbation which when added to any arbitrary audio signal, will likely cause errors in transcription by a victim ASR.

and Amazon Alexa with 98% of target commands being successful. They utilize a surrogate model approach to generate transferable adversarial examples that can attack a number of unseen target devices. However, as noted by the authors, physical attacks are very sensitive to various environmental factors, such as the volume when playing adversarial examples, the distance between the speaker and the victim IVC device, as well as the brand of speakers, that can render the attack unsuccessful. Qin *et al.* [3] designed robust, physical-world, over-the-air audio adversarial examples by constructing perturbations, which remain effective in attacking the Google Lingvo ASR model [135] even after applying environmental distortions. Such robust adversarial examples are crafted by incorporating the noise simulation during the training process of the perturbation. In our work, we evaluate our defense against the robust attack proposed in [3] on the Google Lingvo ASR model. We find that while such examples are more robust to small input changes as compared to previously proposed targeted attacks [1], they can still be

easily distinguished from benign audio samples using our defense framework.

4.1.2 Principles of Defense and Adaptive Attacks in the Image Domain

To strengthen the reliability of deep learning models in the image domain, a significant amount of prior work has proposed defenses to adversarial attacks [138, 139, 142, 140, 145, 146]. However, most of these defenses were only evaluated against non-adaptive attacks or using a "zero-knowledge" threat model, where the attacker has no knowledge of the defense existing in the system. Such defenses offer bare-minimum security and in no way guarantee that they can be secure against future attacks [95, 147]. Accurately evaluating the robustness of defenses is a challenging but important task, particularly because of the presence of adaptive adversaries [20, 147, 148, 149]. An adaptive adversary is one that has partial or complete knowledge of the defense mechanism in place and therefore adapts their attack to what the defender has designed [147, 150, 148].

Many prior works on defenses are variants of the same idea: pre-process inputs using a transform, e.g. randomized cropping, rotation, JPEG compression, randomized smoothing, auto-encoder transformation, that can remove the adversarial perturbation from the input. However, such defenses are shown to be vulnerable to attack algorithms that are partially or completely aware of the defense mechanism [20, 151]. In [20], the authors show that the input-transformation function can be substituted with a differentiable approximation in the backward pass in-order to craft adversarial examples that are robust under the given input-transform. In [151], the authors craft adversarial examples that are robust over a given distribution of transformation functions, which guarantees robustness over more than one type of transform.

Solely analyzing a defense against a non-adaptive adversary gives us a false sense of security. Therefore, the authors of [147] provided several guidelines to ensure completeness in the evaluation of defenses to adversarial attacks. The authors recommend using a threat model with an "infinitely thorough" adaptive adversary, who is capable of developing new optimal attacks against the proposed defense. They recommend applying a diverse set of attacks to

any proposed defense, with the same mindset of a future adversary. However, such defense guidelines have not been applied to the audio domain and many of the proposed ASR defenses have not carried out thorough evaluations against adaptive adversaries. In our work, we follow these guidelines and evaluate our ASR defense against the strongest non-adaptive and adaptive adversaries.

4.1.3 Defenses in the Audio Domain

In comparison to the image domain, only a handful of studies have proposed defenses to adversarial attacks in the audio domain. Prior work on defenses for speech recognition models have focused on both audio pre-processing techniques [143, 152] and utilizing temporal dependency in speech signals [144] to detect adversarial examples.

Yang *et al.* in [144] proposed a defense framework against three attack methods targeting state-of-the-art ASR models such as Kaldi and DeepSpeech. The proposed defense framework checks if the transcription of the first k-sized portion of the audio waveform (t_1) is similar to the first k-sized transcription of the complete audio waveform (t_2). A sample is identified as adversarial when the two transcriptions are dissimilar, i.e., the Character Error Rate (CER) or Word Error Rate (WER) between t_1 and t_2 is higher than a predefined threshold. The authors further study the effectiveness of their defense in an adaptive attack scenario, where the attacker has partial knowledge of the defense framework. In their strongest adaptive attack scenario, they vary the portion k_D used by the defense and evaluate the cases where the adaptive attacker uses a the same/different portion k_A.

However, recent work [149] has re-evaluated temporal dependency frameworks and demonstrated them to be ineffective in detecting adversarial perturbations in the audio domain. The authors of [149] designed attacks that were able to fool the proposed detector in [144] with 100% accuracy, and further report that the adaptive evaluations conducted in [144] are incomplete. In the adaptive attack designed by [149], the CTC loss function used by the attacker incorporates different values of k_A and is therefore able to bypass the temporal dependency

detector with minimal added perturbation to audio.

Aside from proposing the temporal-dependency defense for detection, the authors of [144] also study the effectiveness of various input transformation functions in recovering the original transcription from the adversarial counterpart. To this end, they perform experiments with transformation functions such as quantization, down-sampling, local smoothing and auto-encoder reformation of signals. They report that these methods are ineffective in recovering the correct transcription of audio signals. In our work, we will evaluate some of these transformations for the goal of detecting adversarial examples as opposed to recovering benign examples. However, we report that for some attack types, most transformation based defenses are able to recover the benign audio transcription with low CER.

Rajaratnam *et al.* [143] also studied the use of pre-processing techniques such as audio compression, band-pass filtering, audio panning and speech coding as a part of both isolated and ensemble methods for detecting adversarial audio examples generated by a single targeted attack [148]. While they report high detection performance against the targeted adversarial attack proposed by [148], their techniques were not evaluated in an adaptive attack setting and therefore do not provide security guarantees against a future adversary. Given the difficulty of performing defense evaluations, in our work, we perform additional experiments with various input transformation functions to validate or refute the security claims made in existing papers.

4.2 WaveGuard Methodology

4.2.1 Threat Model

Adversarial attacks in the audio domain can be classified broadly into two categories: *targeted* and *untargeted* attacks. In targeted attacks the goal of the adversary is to add a small perturbation to an audio signal such that it causes the victim ASR to transcribe the audio to a given target phrase. In untargeted attacks the goal is simply to cause significant error in transcription of the audio signal so that the original transcription cannot be deciphered.

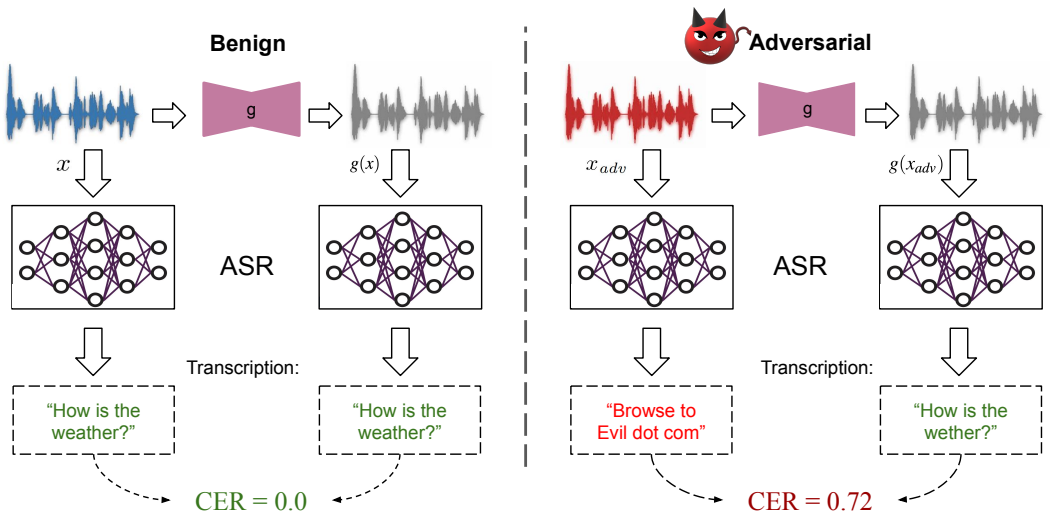


Figure 4.4. WaveGuard Defense Framework: Input audio x is processed using an audio transformation function g to obtain $g(x)$. Next, ASR transcriptions or x and $g(x)$ are compared. An input is classified as *adversarial* if the difference between the transcriptions of x and $g(x)$ exceeds a particular threshold.

The common goal across both targeted and untargeted attack is to cause mis-transcription of the given speech signal while keeping the perturbation imperceptible. Therefore, we define an audio adversarial example x_{adv} as a perturbation of an original speech signal x such that the Character Error Rate (CER) between the transcriptions of the original and adversarial examples from an ASR C is greater than some threshold t. That is,

$$CER(C(x), C(x_{adv})) > t \tag{4.1}$$

and the distortion between x_{adv} and x is constrained under a distortion metric δ as follows:

$$\delta(x, x_{adv}) < \varepsilon. \tag{4.2}$$

Here, $CER(x, y)$ is the edit distance [81] between the strings x and y normalized by the length of the strings i.e.,

$$CER(x, y) = \frac{EditDistance(x, y)}{max(length(x), length(y))}. \tag{4.3}$$

L_p norms are popularly used to quantify the distortion δ between the original and adversarial example in the image domain. Following prior works [1, 2] on audio adversarial attacks, we use an L_∞ norm on the waveforms to quantify the distortion between the adversarial and the original signal.

4.2.2 WaveGuard Defense Framework

The goal of our defense is to correctly detect adversarially modified inputs. The underlying hypothesis for our defense framework is that the network predictions for adversarial examples are often unstable and small changes in adversarial inputs can cause significant changes in network predictions. In the image domain, it has been shown that several input transformation techniques [138, 139, 140, 141] such as JPEG compression, randomized smoothing and feature squeezing can render adversarial perturbations ineffective. This is because such input transformations introduce an additional perturbation in the input that can dominate the carefully added adversarial perturbation. On the other hand, predictions for the original (benign) inputs are usually robust to small random perturbations in the input.

Based on this hypothesis, we propose the following defense framework for detecting audio adversarial examples: For a given audio transformation function g, input audio x is classified as adversarial if there is significant difference between the transcriptions $C(x)$ and $C(g(x))$:

$$d(C(x), C(g(x))) > t \tag{4.4}$$

where d is some distance metric between the two given texts and t is a detection threshold. In our work we use the Character Error Rate (CER) as the distance metric d. z An overview of the defense is depicted in Figure 8.5. Note that unlike [144], the goal using an input transformation g is not to recover the original transcription of an adversarial example, but to detect if an example is adversarial or benign by observing the difference in the transcriptions of x and $g(x)$.

In this work, we study various input transformation functions g as candidates for our

defense framework. We evaluate our defense against four recent adversarial attacks [148, 3, 2] on ASR systems. One of the main insights we draw from our experiments is that in the non-adaptive attack setting, most audio transformations can be effectively used in our defense framework to accurately distinguish adversarial and benign inputs. This result is consistent with the success of input-transformation based defenses in the image domain.

However, in order to use a defense reliably in practice, the defense must be secure against an adaptive adversary who has knowledge of the defense. For an adaptive attack setting, we find that certain input transformations are more robust to attacks than others. Particularly, the transformations which compress audio to perceptually informed representations cannot be easily bypassed even when the attacker has complete knowledge of the defense. This finding is in contrast to the image domain where most input transformation based defenses have been shown to be broken under robust or adaptive adversarial attacks. We elaborate on our adaptive attack scenario and the results in Section 4.6 and Section 4.6.3.

4.3 Input-transformation functions

We study the following audio transformations as candidates for the input transformation function g:

4.3.1 Quantization-Dequantization

Several works in the image domain [141, 153, 154], have used quantization based defenses to neutralize the effect of adversarial perturbations. Since adversarial pertubations to audio have small amplitudes, quantization can help reomve added perturbations. In this study, we employ quantization-dequantization in our defense framework, where each waveform sample is quantized to q bits and then reconstructed back to floating point to produce the output approximation of the original input data.

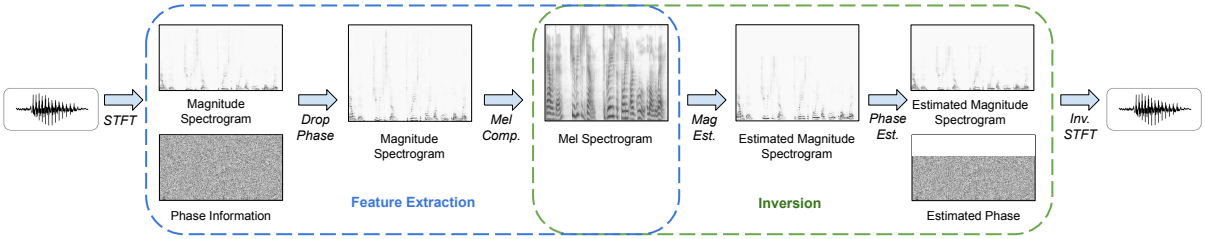


Figure 4.5. Steps involved in the Mel extraction and inversion transform (Section 4.3.4).

4.3.2 Down-sampling and Up-sampling

Discarding samples from a waveform during down-sampling could remove a significant portion of the adversarial perturbation, thereby disrupting an attack. To study this effect, we down-sample the original waveform (16 kHz in our experiments), to a lower sampling rate and then estimate the waveform at its original sampling rate using interpolation. We perform this study for a number of different down-sampling rates to find an optimal range of sampling rates for which the defense is effective.

4.3.3 Filtering

Filtering is commonly applied for noise cancellation applications such as removing background noise from a speech signal. It is intuitive to study the effect of filtering in order to remove adversarial noise from a speech signal. In this work, we use low-shelf and high-shelf filters to clean a given signal. Low-shelf and high-shelf filters are softer versions of high-pass and low-pass filters respectively. That is, instead of completely removing frequencies above or below some thresholds, shelf filters boost or reduce their amplitude. For noise removal, we use a low-shelf filter to reduce the amplitude of frequencies below a threshold and a high-shelf filter to reduce the amplitude of frequencies above a threshold.

In our experiments we first compute the spectral centroid of the audio waveform: Each frame of a magnitude spectrogram is normalized and treated as a distribution over frequency bins, from which the mean (centroid) is extracted per frame. We then compute the median centroid frequency (C) over all frames and set the high-shelf frequency threshold as $1.5 \times C$ and low-shelf

frequency threshold as $0.1 \times C$. We then reduce the amplitude of frequencies above and below the respective thresholds using a negative gain parameter of -30.

4.3.4 Mel Spectrogram Extraction and Inversion

Mel spectrograms are popularly used as an intermediate audio representation in both text-to-speech [155, 156, 157] and speech-to-text [158, 159] systems. While reduction of the waveform to a Mel spectrogram is a lossy compression, the Mel spectrogram is a perceptually informed representation that mostly preserves the audio content necessary for speech recognition systems. We use the following Mel spectrogram extraction and inversion pipeline for disrupting adversarial perturbations in our experiments:

Extraction: We first decompose waveforms into time and frequency components using a Short-Time Fourier Transform (STFT). Then, the phase information is discarded from the complex STFT coefficients leaving only the magnitude spectrogram. The linearly-spaced frequency bins of the resultant spectrogram are then compressed to fewer bins which are equally-spaced on a logarithmic scale (usually the Mel scale [160]). Finally, amplitudes of the resultant spectrogram are made logarithmic to conform to human loudness perception, then optionally clipped and normalized to obtain the Mel spectrogram.

Inversion: To invert the Mel spectrogram into a listenable waveform, the inverse of each extraction step is applied in reverse. First, logarithmic amplitudes are converted to linear ones. Then the magnitude spectrogram is estimated from the Mel spectrogram using the approximate inverse of the Mel transformation matrix. Next, the phase information is estimated from the magnitude spectrogram using a heurisitc algorithm such as Local Weighted Sum (LWS) [161] or Griffin Lim [162]. Finally, the inverse STFT is used to render audio from the estimated magnitude spectrogram and phase information.

We hypothesize that reconstructing audio from a perceptually informed representation can potentially remove the adversarial perturbation while preserving the speech content that is perceived by the human ear. While some speech recognition systems also use Mel spectrogram

features, we find that reconstructing audio from the *compressed* Mel spectrograms introduces enough distortion in the original waveform, such that the ASR Mel features of the newly reconstructed audio are different from the original audio. The distortion in the reconstructed audio is introduced by the magnitude estimation and phase estimation steps depicted in Figure 4.5. In order to bypass a defense involving Mel extraction and inversion, an adaptive attacker will need to craft a perturbation that can be retained in the compressed Mel spectrogram representation, making it challenging to keep the perturbation imperceptible. In our adaptive attack experiments in Section 4.6.3 we demonstrate that even when the attacker uses a differentiable implementation of the Mel extraction and inversion pipeline, it cannot easily be bypassed without introducing a clearly perceptible adversarial noise in the signal.

4.3.5 Linear Predictive Coding (LPC)

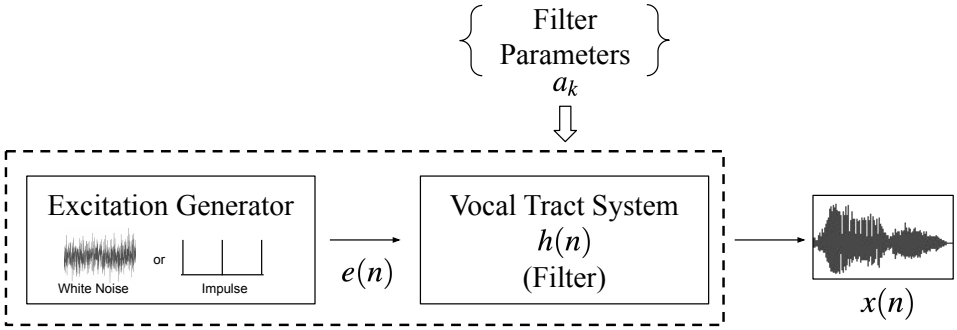


Figure 4.6. Model for linear predictive analysis of speech signals.

Linear Predictive Coding (LPC) is a speech encoding technique that uses a source-filter model based on a mathematical approximation of the human vocal tract. The model assumes that a source signal $e(n)$ (which models the vocal chords) is passed as input to a resonant filter $h(n)$ (that models the vocal tract) to produce the resultant signal $x(n)$. That is:

$$x(n) = h(n) * e(n) \tag{4.5}$$

The source excitation $e(n)$ can either be quasi-periodic impulses (during voiced speech) or

random noise (during unvoiced speech). Both these source excitation sources are spectrally flat implying that all spectral information is modeled in the filter parameters.

LPC assumes a p^{th} order all-pole filter $h(n)$ which means that each waveform sample is modelled as a linear combination of p previous values. That is,

$$x(n) = \Sigma_{k=1}^{k=p} a_k x(n-k) + e(n). \tag{4.6}$$

The basic problem of LPC analysis is to estimate the filter parameters a_k. Since the source signal is assumed to be an impulse train or random white noise, the problem is formulated as minimizing $||e(n)||^2$ which is the power of the excitation signal. This reduces the parameter-estimation problem to a linear regression problem in which the goal is to minimize:

$$\textit{minimize: } \langle ||e(n)||^2 \rangle = \langle (x(n) - \Sigma_{k=1}^{k=p} a_k x(n-k))^2 \rangle \tag{4.7}$$

Here, $\langle\rangle$ denotes averaging over finite number of waveform samples. In practice, a long time-varying signal is divided into overlapping windows of size w and LPC coeffecients a_k are estimated for each window by solving the above linear regression problem. To re-synthesize the signal from the estimated coefficients, we use a random-noise excitation signal. In our experiments, we use 25 millisecond windows with 12.5 millisecond overlap. We experiment with different numbers of the LPC coeffecients which control the compression level of the original signal.

Since LPC models the human vocal tract system, it preserves the phonetic information of speech in the filter parameters. Bypassing a defense involving LPC transform, would require the adversary to add an adversarial perturbation that can be preserved in the LPC filter coeffecients; thereby requiring the adversary to modify the phonetic information in speech. We empirically demonstrate that the LPC transform is the most robust against an adaptive adversary amongst all the transforms studied in this paper.

4.4 Experimental Setup

We evaluate our defense against the following recent audio adversarial attacks on speech recognition systems [1, 2, 3]:

- **Carlini:** Attack introduced in [1]. This is a white-box targeted attack on the Mozilla Deepspeech [71] ASR system, where the attacker trains an adversarial perturbation by minimizing the CTC loss between the target transcription and the ASR's prediction. This attack minimizes the L_∞ norm of the adversarial perturbation to constrain the amount of distortion.

- **Qin-I:** Imperceptible attack described in [3]. This is another white-box targeted attack that focuses on ensuring imperceptibility of the adversarial perturbation by using psycho-acoustic hiding. The victim ASR for this attack is Google Lingvo [135].

- **Qin-R:** Robust attack described in [3]. This attack incorporates input transformations during training of the adversarial perturbation which simulate room environments. This improves the attack robustness in real world settings when played over the air. The victim ASR for this attack is Google Lingvo [135].

- **Universal:** We implement the white-box attack described in [2]. This is an untargeted attack which finds an input-agnostic perturbation that can cause significant disruption in the transcription of the adversarial signal. In our work, we follow the algorithm provided by the authors and craft universal perturbation with an L_∞ bound of 400 (for 16-bit audio wave-forms with sample values in the range -32768 to 32768). The victim ASR for this attack is Mozilla DeepSpeech [71].

4.4.1 Dataset and Attack Evaluations

We conduct all our experiments on the Mozilla Common Voice dataset, which contains 582 hours of audio across 400,000 recordings in English. The audio data is sampled at 16 kHz.

Table 4.1. Adversarial commands used for constructing targeted adversarial examples.

Target Adversarial Commands
”browse to evil dot com”
”hey google cancel my medical appointment”
”hey google”
”this is an adversarial example”

Table 4.2. Evaluations for each input transformation defense against various non-adaptive attacks. We use two objective metrics: AUC score and Attack Detection Accuracy for evaluation (higher values are better for both metrics).

		AUC Score				*Detection Accuracy*			
Defense	Hyper-params	Carlini	Universal	Qin-I	Qin-R	Carlini	Universal	Qin-I	Qin-R
Downsampling - Upsampling	6000 kHz	1.00	0.91	1.00	1.00	100%	88%	100%	100%
Quantization - Dequantization	6 bits	0.99	0.92	1.00	0.93	98.5%	88%	99%	95%
Filtering	(Section 4.3.3)	1.00	0.92	1.00	1.00	99.5%	86%	100%	100%
Mel Extraction - Inversion	80 Mel-bins	1.00	0.97	1.00	1.00	100%	92%	100%	100%
LPC	LPC order 20	1.00	0.91	1.00	1.00	100%	83%	100%	100%

We evaluate on the same subset of the Mozilla Common Voice dataset, as used in [1], that is, the first 100 examples from the Mozilla Common Voice test set. We construct adversarial examples on this dataset using each of the attacks described above. In the targeted attack scenario, we randomly choose one of the target phrases listed in Table 4.1 and follow the attack algorithms to create 100 pairs of original and adversarial examples for each attack type. For the untargeted universal attack, we train the universal perturbation on the same subset of Mozilla Common Voice examples with L_∞ distortion bound of 400.

Attack evaluations: We achieve 100% attack success rate for *Carlini* and *Qin-I* attacks. For *Qin-R*, the attack achieves 47% success rate (similar to that reported in the paper [3]) on 100 examples. In our experiments when recreating the *Universal* attack, we achieve an attack success rate of 81% using the same criteria as described in [2] i.e., the attack is considered successful when the CER between original and adversarial transcriptions is greater than 0.5.

4.4.2 Evaluation Metrics

As described in Section 4.2.2, in our detection framework, we label an example as *adversarial* or *benign* based on the CER between x and $g(x)$. The decision threshold t controls the true positive rate and false positive rate of our detector. Following standard procedure to evaluate such detectors [144], we calculate the *AUC score* - Area Under the ROC curve. A higher AUC score indicates that the detector has more discriminative power against adversarial examples.

Additionally, we also report the *Detection Accuracy* which is calculated by finding the best detection threshold t on a separate set containing 50 adversarial and benign examples.

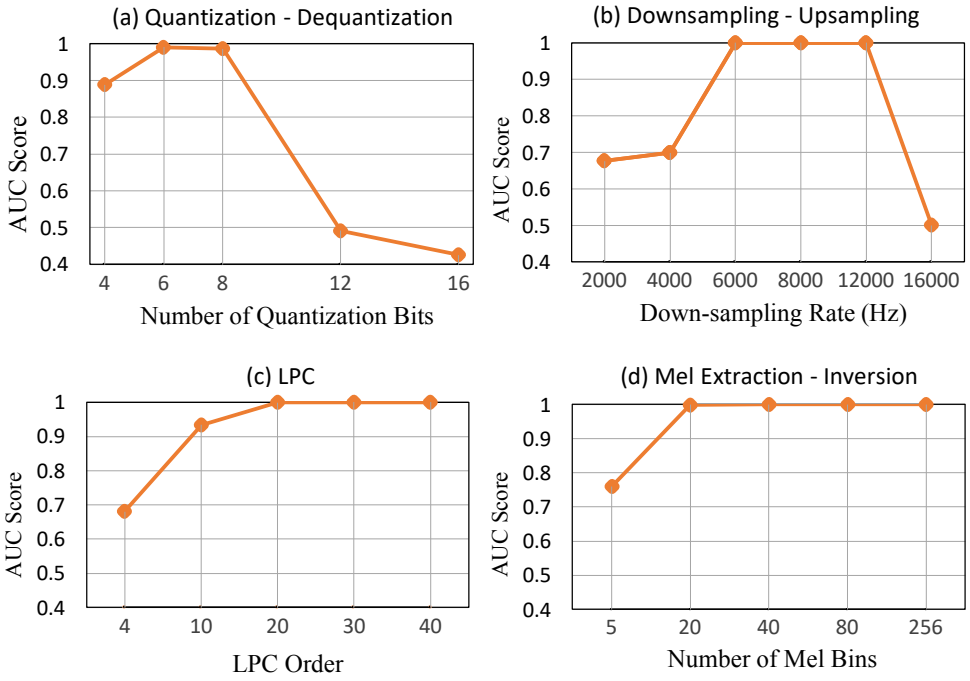


Figure 4.7. Detection AUC Scores against *Carlini* attack at varying compression levels for the following transforms: (a) Quantization - Dequantization; (b) Downsampling - Upsampling; (c) Linear Predictive Coding (LPC); and (d) Mel Spectrogram Extraction- Inversion.

4.5 Evaluation against Non Adaptive Attacks

The various input transformation functions we consider can be parameterized to control the compression level of the transformation. There is a trade-off between the compression level and the discriminative power of the detector. At low compression levels the transformation may not eliminate the adversarial perturbation. In contrast, at very high compression levels, even the benign signals may become significantly distorted causing substantial change in their transcriptions. Keeping this in mind, we perform a search over the hyper-parameters for different audio transforms. The AUC score of the detector against the *Carlini* attack for different transformation functions at varying compression levels is depicted in Figure 4.7. For most transformations, we observe the expected pattern where the defense is effective at some optimal compression levels and the AUC falls at very high or low compression levels. The Mel-inversion pipeline is effective for a wide range of *Mel-bins* possibly due to the distortion introduced by the phase estimation step during the inversion stage. For the *Filtering* transform we do not perform a hyper-parameter search and use the transformation parameters described in Section 4.3.3.

4.5.1 Attack Detection Scores

Based on the above described search, we find the optimal hyper-parameters for each of the transforms and report the detection scores against all the attacks in Table 4.2. We observe that at optimal compression levels, all the input transforms listed in Section 4.3 can achieve high discriminative performance against adversarial examples. As compared to targeted adversarial examples, it is harder to detect examples with universal adversarial perturbations. This is because universal perturbations attempt to distort the original transcription rather than targeting a very different phrase. Interestingly, we find that the defense is effective even against the *Qin-R* attack which incorporates noise simulation during training and leads to adversarial examples that are robust to small changes. We elaborate on this result in the following Section.

4.5.2 Analysis of undefended and defended transcriptions

In Figure 4.8 we provide comparisons of Mean CER between transcriptions of audio before and after passing through a given transformation function (g) for both benign ($orig$) and adversarial examples (adv). Additionally, we also calculate the CER between the transcriptions of the defended adversarial example and its benign counterpart: $CER(orig, g(adv))$.

The discriminative power of the detector is indicated by the difference between $CER(orig, g(orig))$ (blue) and $CER(adv, g(adv))$ (red). A high difference between the red and blue bar graphs in Figure 4.8 indicates easier detection of adversarial examples. From these results we can observe that detecting the *Qin-I* attack is easier than detecting the *Carlini* [1] attack. We can further deduce that detecting *Universal* attacks is generally more difficult for any given transformation function compared to the *Carlini* and *Qin-I* attacks.

The metric $CER(orig, g(adv))$ helps evaluate the ability of the transformation function to recover the original transcript from the adversarial audio. A low $CER(orig, g(adv))$ indicates better recovery of the original transcript. We find that for the imperceptible attack *Qin-I*, the recovery rate of the original transcript is higher than any other attack indicating that the adversarial perturbation is unstable to small changes in inputs.

The *Qin-R* attack has a lower $CER(adv, g(adv))$ for most transformations as compared to *Qin-I* which suggests that the adversarial perturbation generated by the *Qin-R* attack is relatively more robust to input transformations. Also, recovering the original transcription is much harder as compared to *Qin-I* and is indicated by higher $CER(orig, g(adv))$ values. However, there is still a significant difference between the blue and red bar graphs for *Qin-R*, which can be used to discriminate between adversarial and benign samples. This result is consistent with the high detection accuracy reported in Table 4.2, since the transformations are successful in disrupting the adversarial perturbations.

We provide a few sample transcriptions from our experiments in Table 4.3. The green commands indicate the transcriptions from benign audio samples, while the red transcriptions

Table 4.3. Sample transcriptions of un-transformed(x) and transformed audio($g(x)$) for both benign and adversarial examples.

Attack	Adversarial Command (C(x_adv))	Defended Command (C(g(x_adv)))					Benign Command (C(x))
		Down-Up	Quant	Filter	Mel	LPC	
Carlini	**"browse to evil dot com"**	i'm sure i didn't know whenc set's talking about	"i'm sure i don't know what you' talking about"	"srown to withe cot gom"	"i'm sure i don't know what you'e talking about"	"absure i don't know what you' talking about"	**"i'm sure i don't know what you're talking about"**
Qin-I	**"hey google"**	"this is no place for you"	"this is no place for you"	"but it is no place for you"	"this is no place for you"	"this is no place for you"	**"this is no place for you"**
Qin-R	**"hey google cancel my medical appointment"**	"ah you hahogum he hath a home and not far called the man pulling there"	"hey de laggle cancel my medical appointment"	"he hated the loggal cly anticone not a particle of appointment"	"lady galogolfe and lygam amethurical appointment"	"and when i had never he ankle a handful for my little appointment"	**"he did find it soon after dawn and not far from the sand pits"**
Universal	**"there ae little ied ne callyuack"**	"wa didn't i call you back"	"why didn't i call you back"	"lodidn't i call you back"	"why didn't i call you back"	" litwoted no col yo back"	**"why didn't o call you back"**
	Benign Command (C(x))	Defended Command (C(g(x)))					
		Down-Up	Quant	Filter	Mel	LPC	
	"i'm sure i don't know what you're talking about"	"i'm sure i don't know what you're talking about"	"i'm sure i don't know what you're talking about"	"i'm sure i don't know what you're talking about"	"i'm sure i don't know what you're talking about"	"i'm sure i don't know what you're talking about"	

refer to adversarial commands from each attack type. Overall, the results in Figure 4.8 and Table 4.3 demonstrate that the ability to recover benign commands is dependent on the type of attack and varies for each input transformation function.

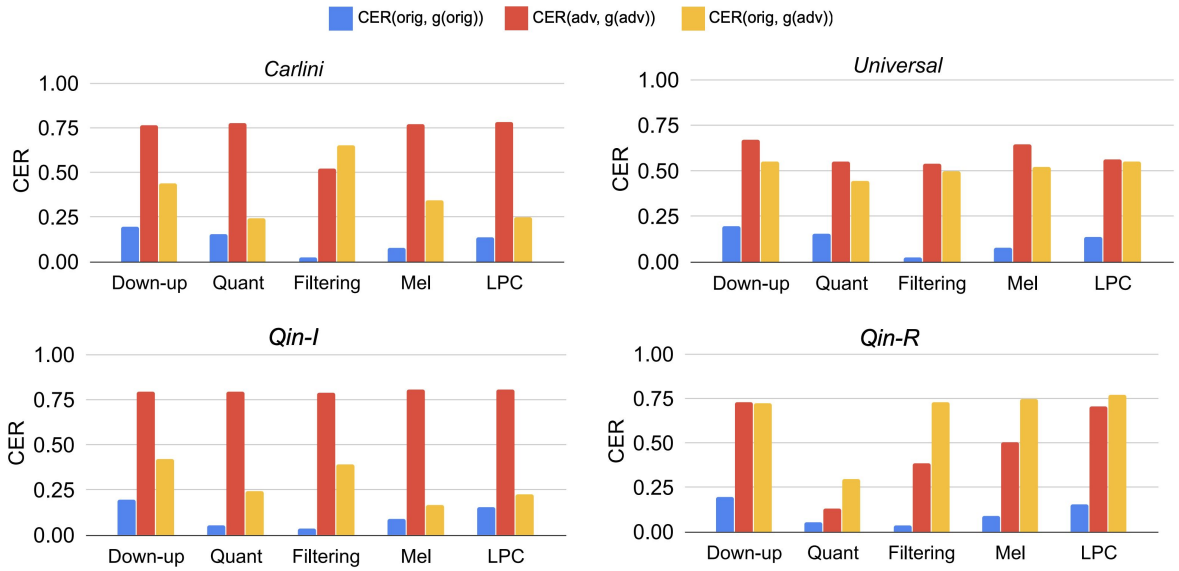


Figure 4.8. Mean Character Error Rate (CER) is measured between ASR transcriptions of un-transformed (x) and transformed ($g(x)$) audio for original and adversarial pairs crafted using various attacks.

4.5.3 ROC for Detection under Non-Adaptive Attacks

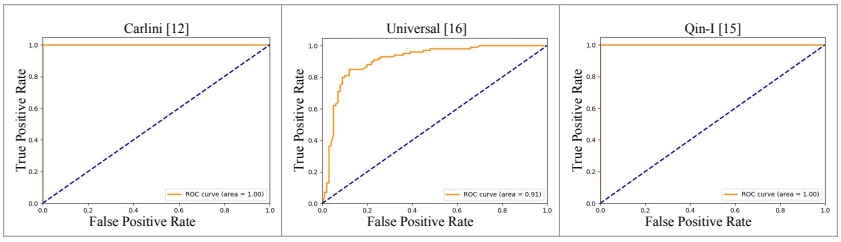
We provide the Receiver Operating Characteristic (ROC) curves for our detection of non-adaptive adversarial attacks using various transformation functions against three different adversarial attacks in Figure 4.9. The AUC scores are reported in Table 4.2 in Section 4.5.1 and included with each of the plots below. A *true positive* implies an example that is adversarial and is correctly identified as adversarial.

4.5.4 Timing analysis

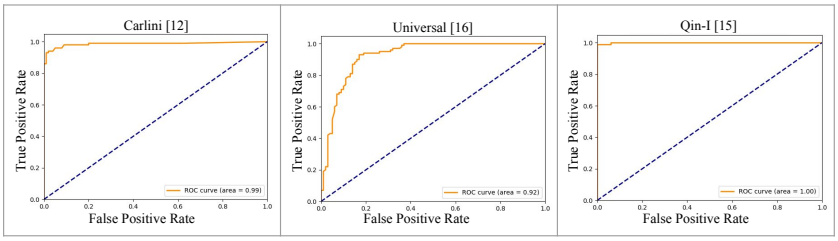
To implement our defense framework in practice, we have to perform two forward passes through our ASR model to obtain the transcriptions $C(x)$ and $C(g(x))$. It is ideal to parallelize these two forward passes, so that the only computational overhead introduced by the defense is that of the transformation function g. Table 4.4 provides the average Wall-Clock time in seconds of each transformation function averaged over the 100 audio files (entire test set). Since our transformation functions were implemented solely on CPU, we provide timing comparisons for all implementations on the Intel Xeon CPU platform. The average inference time over the test set for Mozilla Deepspeech ASR model is 2.540 seconds and that of Google Lingvo ASR model is 4.212 seconds on the Intel Xeon CPU Platform.

Table 4.4. Average Wall-Clock time in seconds required for transcription of audio by ASR models and each transformation function on Intel Xeon CPU platform. The Wall-Clock time is averaged over the entire test set.

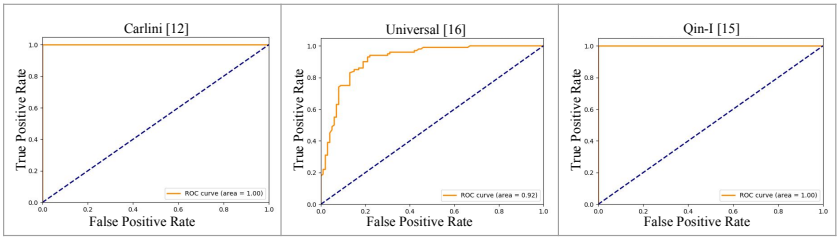
Process	Avg. Wall-Clock time (s)
Deepspeech ASR	2.540
Lingvo ASR	4.212
Downsampling-Upsampling	0.148
Quantization-Dequantization	0.001
Filtering	0.035
Mel Extraction - Inversion	0.569
LPC	0.781



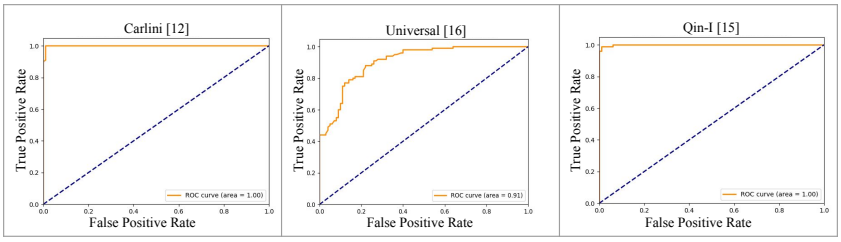
(a) Downsampling-upsampling



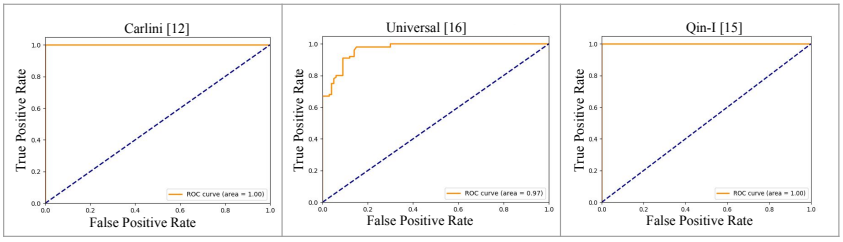
(b) Quantization



(c) Filtering



(d) Linear Predictive Coding (LPC)



(e) Mel Extraction - Inversion

Figure 4.9. Detection ROC curves for different transformation functions against three attacks (Carlini [1], Universal [2], Qin-I [3]) in the non-adaptive attack setting.

4.5.5 Thresholds for Attack Detection Accuracy

Table 4.5 lists the detection thresholds (t) for various transformation functions for the two ASR systems studied in our work. We choose 50 original examples (separate from the first 100 used for evaluation) and construct 50 adversarial examples using each of the attack. This results in 100 adversarial-benign example pairs for DeepSpeech (constructed using Carlini [1] and Universal [2] attacks) and 100 adversarial-benign example pairs for Google Lingvo (constructed using Qin-I and Qin-R attacks [3]). Using this dataset, we obtain the threshold that achieves the best detection accuracy for each defense separately for the two ASRs. The AUC metric is threshold independent. We do not change the threshold for adaptive attack evaluation and use the same threshold as listed in Table 4.5.

Table 4.5. Detection Threshold when using each transformation function in WaveGuard framework for DeepSpeech and Lingvo ASR systems.

Defense	Threshold - DeepSpeech	Threshold - Lingvo
Downsampling - Upsampling	0.48	0.48
Quantization - Dequantization	0.44	0.26
Filtering	0.32	0.31
Mel Extraction - Inversion	0.33	0.31
LPC	0.38	0.46

4.6 Adaptive Attack

While our defense framework can accurately discriminate adversarial from benign examples for existing attacks, it only offers security in a "zero-knowledge" attack scenario where the attacker is not aware of the defense being present. As motivated in Section 4.1.2, in order to use our defense framework reliably in practice, it is important to evaluate it against an adaptive adversary who has complete knowledge of the defense and intend to design a perturbation that can bypass the defense mechanism.

In the adaptive attack setting, we will focus on the more impactful targeted attack scenario,

where the adversary designs an adversarial perturbation that causes the victim ASR system to transcribe the input audio into a specific target phrase. In order to bypass the proposed defense framework, the adversary must craft an adversarial perturbation such that the transcription of $C(x_{adv})$ and $C(g(x_{adv}))$ match closely with each other and the target transcription τ. Therefore, to craft such a perturbation δ, the adversary aims to optimize the following problem:

$$\textit{minimize:} \;\; |\delta|_\infty + c_1 \cdot \ell(x+\delta, \tau) + c_2 \cdot \ell(g(x+\delta), \tau)$$

where, $\ell(x', t) = \text{CTC-Loss}(C(x'), t)$ and c_1 and c_2 are hyper-parameters that control the weights of the respective loss terms. Since optimization process over the L_∞ metric is often unstable [1], we modify our optimization objective as follows:

$$\begin{aligned} \textit{minimize:} \;\; & c \cdot |\delta|_2^2 + c_1 \cdot \ell(x+\delta, \tau) + c_2 \cdot \ell(g(x+\delta), \tau) \\ & \textit{such that } |\delta|_\infty < \varepsilon \end{aligned} \tag{4.8}$$

4.6.1 Gradient Estimation for Adaptive Attack

To solve the optimization problem given by equation 4.8 using gradient descent, the attacker must back-propagate the CTC-Loss through the ASR model and the input transformation function g. In case a differentiable implementation of g is not available, we use the Backward Pass Differentiable Approximation (BPDA) technique [20] to craft adversarial examples. That is, during the forward pass we use the exact implementation of the transformation function as used in our defense framework. During the backward pass, we use an approximate gradient implementation of the transformation g. We first perform the adaptive attack using the straight-through gradient estimator [20]. That is, we assume that the gradient of the loss with respect to the input x to be the same as the gradient of the loss with respect to $g(x)$:

$$\nabla_x \ell(g(x))|_{x=\hat{x}} \approx \nabla_x \ell(x)|_{x=g(\hat{x})} \,. \tag{4.9}$$

In our experiments, we find that the straight-through estimator is effective in breaking the Quantization-Dequantization and Filtering transformation functions at low perturbation levels. However, using a more accurate gradient estimate can lead to a stronger attack. Specifically for the Mel-inversion and LPC transformations, we find that using a straight-through gradient estimator does not work for solving the above optimization problem (Equation 4.8). We discuss our results of using a straight-through gradient estimator for LPC transform in Appendix 4.6.3. Also, using a straight-through estimator for the Downsampling-Upsampling transform results in high distortion for adversarial perturbations. Therefore, we implement differentiable computational graphs for the following three transforms in TensorFlow:

Downsampling-Upsampling: We use TensorFlow's bi-linear resizing methods to first downsample the audio to the required sampling rate and then re-estimate the signal using bi-linear interpolation.

Mel inversion: For the Mel inversion transform we use TensorFlow's STFT implementation to obtain the magnitude spectrogram, then perform the Mel transform using matrix multiplication with the Mel basis, and estimate the waveform using the iterative Griffin-Lim [162] algorithm implemented in TensorFlow [163].

LPC transform: We replicate the LPC analysis and synthesis process in TensorFlow. Specifically, for each overlapping window in the original waveform, we first estimate LPC coefficients by solving the linear regression problem given by Equation 4.7. Next, for the reconstruction process, we generate the residual excitation signal using the exact same implementation as used in our defense. We also fix the random seed of the excitation generator in both our defense and our adaptive attacks for a complete knowledge white box attack scenario. Finally, we implement auto-regressive filtering of the residual signal with the LPC coefficients for that window, and combine the filtered signal for each overlapping window to generate the transformed audio.

Note that for all the adaptive attacks, we use the original defense implementations in the forward pass and use the differentiable implementation only during the backward pass.

4.6.2 Adaptive Attack Algorithm

Algorithm 4 details our adaptive attack implementation. We closely follow the targeted attack implementation in [1] and incorporate the optimization objective of our adaptive attack specified by Equation 4.8 and BPDA. We choose $c_1 = c_2 = 1$ since both loss terms have the same order of magnitude. Following the default open source implementation of [1], we do not penalize L_2 distortion. We optimize for 5000 iterations and use a learning rate of 10. Any time the attack succeeds, we re-scale the perturbation bound by a factor of 0.8 to encourage less distorted (quieter) adversarial examples.

Algorithm 4. Adaptive attack algorithm

Initialize $rescaleFactor \leftarrow 1$
Initialize $\delta \leftarrow 0$
Initialize $bestDelta \leftarrow null$
for *iterNum* in 1 to *MaxIters* **do**
    $loss \leftarrow c \cdot |\delta|_2^2 + c_1 \cdot \ell(x+\delta, t) + c_2 \cdot \ell(g(x+\delta), t)$
    $\nabla\delta \leftarrow BPDA(loss, \delta)$
    $\delta \leftarrow \delta - \alpha\ sign(\nabla\delta)$
    $\delta \leftarrow rescaleFactor * clip_\varepsilon(\delta)$
    if $C(x+\delta) = C(g(x+\delta)) = \tau$ **then**
        $bestDelta \leftarrow \delta$
        $rescaleFactor \leftarrow rescaleFactor \times 0.8$
if *bestDelta* is *null* **then**
    $bestDelta \leftarrow \delta$
return $(x + bestDelta)$

4.6.3 Adaptive Attack Evaluation

In this section, we test the limits of our defense and evaluate the breaking point for each transformation function through adaptive attacks in white box setting. We conduct adaptive attack evaluations on the same dataset used in our previous experiments. The victim ASR for the adaptive attack is the Mozilla DeepSpeech model. In order to evaluate the imperceptibility of adversarial perturbations, we quantify the distortion of adversarial perturbations as follows.

Table 4.6. Adaptive attack evaluations against different transformation functions. ε_∞ is the initial L_∞ bound used and δ_∞ is the mean L_∞ norm of the perturbations obtained after applying the adaptive attack algorithm. Bolded values indicate the δ_∞ required to completely break a particular defense.

	Distortion metrics			*Attack Performance*				*Detection Scores*	
Defense	ε_∞	$\lvert\delta\rvert_\infty$	$dB_x(\delta)$	SR (x_{adv})	SR ($g(x_{adv})$)	$CER(x_{adv},\tau)$	$CER(g(x_{adv}),\tau)$	AUC	Acc.
None	500	81	-45.3	100%	-	0.00	-	-	-
Downsampling - Upsampling	500	**342**	-32.7	100%	78%	0.00	0.05	0.31	50.0%
Quantization - Dequantization	500	**215**	-36.7	100%	81%	0.00	0.01	0.11	50.0%
Filtering	500	**92**	-44.1	91%	72%	0.01	0.02	0.45	50.0%
Mel Extraction - Inversion	500	500	-29.4	34%	0%	0.11	0.44	0.97	95.5%
LPC	500	500	-29.4	43%	0%	0.06	0.51	0.94	86.0%
Mel Extraction - Inversion	1000	1000	-23.5	53%	0%	0.05	0.34	0.92	84.0%
LPC	1000	1000	-23.5	72%	0%	0.01	0.29	0.77	72.5%
Mel Extraction - Inversion	4000	**2461**	-15.1	100%	31%	0.00	0.08	0.48	50.0%
LPC	4000	**2167**	-16.7	100%	73%	0.0	0.03	0.21	50.0%

Distortion Metrics and Relative Loudness: We first implement adaptive attacks using an initial distortion bound $|\varepsilon|_\infty = 500$. Note that we are using a 16-bit waveform representation which means that the waveform samples are in the range -32768 to 32768. An L_∞ distortion of 500 is fairly perceptible although it does not completely mask the original signal.[2] Along with the L_∞ norm of the perturbation, we report another related metric $dB_x(\delta)$ [1, 2] that measures the relative loudness of the perturbation with respect to the original signal in Decibels(dB). The metric $dB_x(\delta)$ is defined as follows:

$$\begin{aligned} dB(x) &= max_i 20\log_{10}(x_i) \\ dB_x(\delta) &= dB(\delta) - dB(x) \end{aligned} \tag{4.10}$$

The more negative $dB_x(\delta)$ is, the quieter is the adversarial perturbation. For comparison, -31 dB is roughly the difference between ambient noise in a quiet room and a person talking [1]. While we start with an initial L_∞ (ε_∞) bound of 500 in our experiments, the final distortion norm (δ_∞) can be much smaller than the initial bound. This is because our optimization objective penalizes high distortion amounts and our algorithm re-scales the perturbation bound by a factor of 0.8

[2]Audio Examples: https://waveguard.herokuapp.com

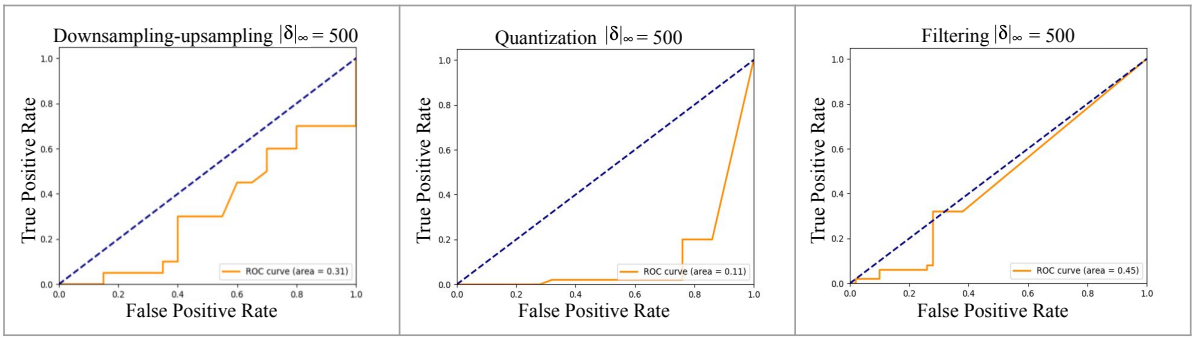
every time the attack succeeds.

Generally, prior work on attacks to ASR systems apply particular attention to minimize perturbation distortions, in order to encourage imperceptibility of adversarial audio. Towards this goal of generating imperceptible adversarial examples, Qin et al. [3] and Universal [2] generate examples with maximum allowed distortion of L_∞ = 400, while Carlini et al. [1] generate examples with maximum distortion of L_∞ = 100. However for conducting our adaptive attack evaluation, since we aim to test the breaking point of each transformation function, we generate adversarial perturbations at much higher L_∞ bounds (500, 1000, 4000) that are significantly more audible to the human ear.

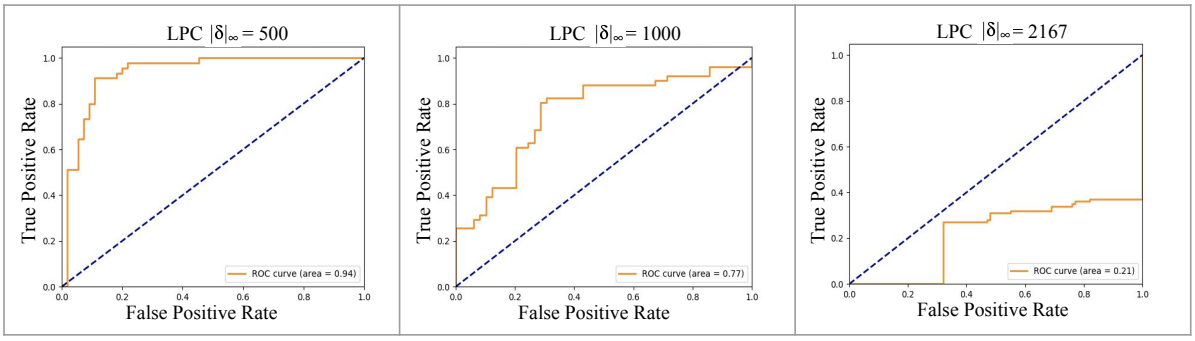
Table 4.6 presents the results for our adaptive attack against various input transformation functions. We provide the Receiver Operating Characteristic (ROC) of the detector in the adaptive attack settings for different transformation functions under different magnitudes of perturbation in Figure 4.10. A *true positive* implies an example that is adversarial and is correctly identified as adversarial. We evaluate the adaptive attacks on two aspects: 1) *Attack Performance:* How successful was the adaptive attack in its objective? 2) *Detection Scores:* How effective is our detector for the adversarial audios generated by the attack?

For the adaptive attacks against the *Downsampling - upsampling*, *Quantization - Dequantization* and *Filtering* transforms, we achieve low CER between the target transcription and transcriptions for x_{adv} and $g(x_{adv})$ ($CER(x_{adv}, \tau)$ and $CER(g(x_{adv}))$ respectively). This makes it harder for the detector to discriminate between adversarial and benign samples thereby resulting in a drastic drop in detector AUC and accuracy scores as compared to the non-adaptive scenario. Amongst these three transformations, bypassing *Downsampling-upsampling* requires the highest amount of perturbation ($\delta_\infty = 342$) indicating that it serves as a more robust defense transformation as compared to *Quantization-Dequantization* and *Filtering*. The columns $SR(x_{adv})$ and $SR(g(x_{adv}))$ indicate the percentage of examples that transcribed exactly to the target phrase for the un-transformed and transformed adversarial inputs respectively.

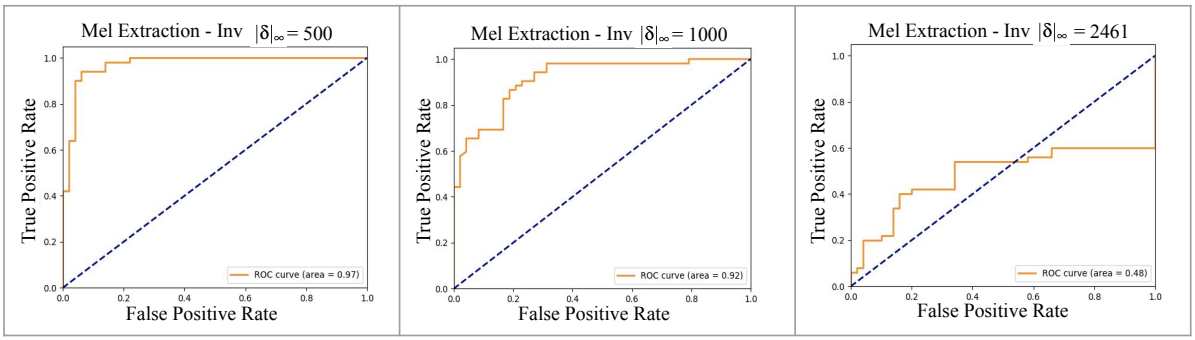
The calibration of the detection threshold depends on the use case of the ASR system—



(a) Downsampling-upsampling, Quantization and Filtering



(b) Linear Predictive Coding (LPC)



(c) Mel Extraction - Inversion

Figure 4.10. Detection ROC curves for different transformation functions against adaptive attacks (Section 4.6.3) with various magnitudes of adversarial perturbation ($|\delta|_\infty$).

for a user facing ASR system, the number of legitimate commands would usually be very high as compared to the number of adversarial commands. Therefore, the false positive rate needs to be extremely low for such ASR systems. As shown in Figure 4.9 (Appendix 4.5.3), in the non-adaptive attack scenario, we are able to achieve a very high true positive rate at 0% false positive rate for the targeted adversarial attacks (Carlini and Qin-I) for all transformation

functions. Therefore a low detection threshold can be reliable against non-adaptive adversaries and also not interfere with the user experience. In the adaptive attack scenario, while both LPC and Mel inversion achieve higher AUC scores as compared to other transforms, Mel inversion transform gives the highest true positive rate at extremely low false positive rates. Therefore, amongst the transformation functions studied in our work, Mel Extraction and Inversion serves as the best defense choice for user facing ASR systems.

Robustness of perceptually informed representations: For both Mel extraction-inversion and LPC transformations, although we observe a drop in the detector scores as compared to the non-adaptive attack setting, we are not able to completely bypass the defense using the initial distortion bound $\varepsilon_\infty = 500$. Note that a perturbation higher than this magnitude, has $dB_x(\delta) > -29$ which is more audible than ambient noise in a quiet room ($dB_x(\delta) = -31$) [85, 148]. In order to test the limit at which the defense breaks, we successively increase the allowed magnitude of perturbation. We are able to completely break the defense (AUC ≤ 0.5) at $\delta_\infty = 2479$ and $\delta_\infty = 2167$ for Mel extraction-inversion and LPC transforms respectively. These perturbations are more than $6\times$ higher than that required to break any of the other transformation functions studied in our work and more than $25\times$ higher than that required to fool an undefended model. At such distortion levels, the perturbation is very perceptible to the human ear so it defeats the purpose of an adversarial audio attack. This suggests that using perceptually informed intermediate representations prove to be more robust against adaptive attacks as compared to naive compression and decompression techniques.

Figure 4.11 reports the same metrics as those reported in Figure 4.8 for the adaptive attack scenario with an initial $\varepsilon_\infty = 500$. The $CER(adv, g(adv))$ (red bar) drops below $CER(orig, g(orig))$ (blue bar) for *Downsampling-upsampling*, *Quantization-Dequantization* and *Filtering* transforms thereby breaking these defenses. In contrast, the red bar for *Mel extraction-inversion* and *LPC* based defense is much higher than the blue bar indicating that the defense is more robust under this adaptive attack setting.

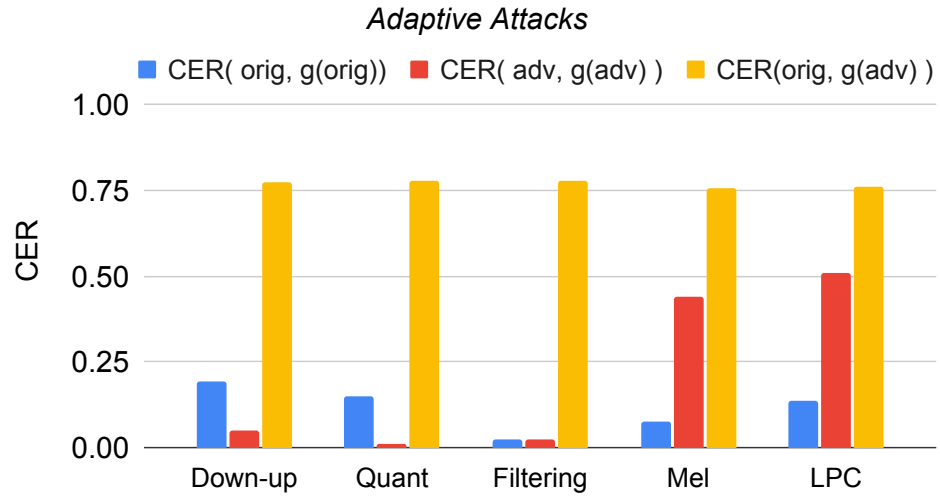


Figure 4.11. Mean CER between the ASR transcriptions of un-transformed (x) and transformed ($g(x)$) audio for adaptive attacks with an initial distortion $\varepsilon_\infty = 500$.

Straight-through Gradient Estimator for LPC

We find that the LPC transform cannot be broken in an adaptive attack scenario using BPDA attack with a straight-through gradient estimator (i.e assuming identity function as the gradient of transformation function g during the backward pass). In our experiments, we started with an initial ε_∞ of 2000, and increased the initial distortion bound to 16000 but did not observe any improvement in the attack performance as the detector was still able to identify adversarial audio with 100% accuracy. Therefore, using our BPDA attack algorithm, we do not arrive at a solution in which both x and $g(x)$ transcribe to the target phrase even with a high amount of allowed distortion. This motivated us to design stronger adaptive attacks with differentiable LPC (Section 4.6.1) to find distortion bounds over which LPC transforms are not able to reliably detect adversarial examples.

Table 4.7. Evaluation of LPC transform against straight-through gradient estimator.

	Distortion metrics			*Attack Performance*		*Detection Scores*	
Defense	ε_∞	$\lvert\delta\rvert_\infty$	$dB_x(\delta)$	$CER(x_{adv},\tau)$	$CER(g(x_{adv}),\tau)$	AUC	Acc.
LPC	2000	2000	-15.9	0.31	0.85	1.0	100%
LPC	16000	16000	2.1	0.34	0.85	1.0	100%

Table 4.8. Evaluation of Mel Extraction - Inversion and LPC transform defense against perturbations targeting an undefended DeepSpeech ASR model at different levels of magnitude.

	Distortion metrics		*Attack Performance*		*Detection Scores*	
Defense	$\lvert\delta\rvert_\infty$	$dB_x(\delta)$	$CER(x_{adv},\tau)$	$CER(g(x_{adv}),\tau)$	AUC	Acc.
LPC	1000	-23.5	0.0	0.80	0.99	98.5%
LPC	2000	-17.4	0.0	0.83	0.99	99.0%
LPC	4000	-11.4	0.0	0.81	0.99	97.0%
LPC	8000	-5.4	0.0	0.91	0.99	99.0%
Mel Ext - Inv	1000	-23.5	0.0	0.81	0.99	98.5%
Mel Ext - Inv	2000	-17.4	0.0	0.88	0.99	97.5%
Mel Ext - Inv	4000	-11.4	0.0	0.89	0.99	98.0%
Mel Ext - Inv	8000	-5.4	0.0	0.92	0.99	98.5%

4.7 Evaluation of Transfer Attacks from an Undefended Model

We additionally evaluate the robustness of Mel extraction-inversion and LPC transformations against transfer attacks from an undefended model. We craft targeted adversarial examples using [1] for DeepSpeech ASR at different perturbation levels by linearly scaling the perturbation to have the desired L_∞ norm. Table 4.8 shows the evaluations of transfer attack at different perturbation levels. We find that attacks targeting undefended models do not break the defense using these two transformation functions even at high perturbation levels. This is because the transcription of $g(x_{adv})$ is significantly different from the target transcription and transcription of x_{adv} even at high perturbation levels thereby allowing our detector to consistently detect the adversarial samples.

4.8 Discussion

Do learnings from adversarial defenses in the image domain transfer over to the audio domain? We find that not all learnings about input-transformation based defenses in the image domain transfer to the speech recognition domain. It has been shown that input-transformation based adversarial defenses can be easily bypassed using robust or adaptive attacks for image classification systems [20, 151]. However, an ASR system is a substantially different architecture as compared to an image classification model. ASR systems operate on time-varying inputs and map each input frame to a language token. Since they rely on Recurrent Neural Networks (RNNs), the token prediction for each frame also depends on other frames in the signal. For targeted attacks, that are robust to a transformation g, we need to find an adversarial example x_{audio} such that both x_{audio} and $g(x_{audio})$ map to the target language tokens across all time-steps. On the other hand, for the image classification problem, the adaptive attack goal is simpler: Find an image x_{image}, such that both x_{image} and $g(x_{image})$ map to the same class label. Therefore, in our adaptive attack experiments, we need to add significant amount of perturbation to bypass the defense even for simple transformation functions. We also find that adversarial attacks targeting undefended ASR models do not transfer to defended models even at high perturbation levels, in contrast to results reported in the image domain [149]. Details of this experiment are provided in Section 4.7.

4.9 Conclusion

In this chapter, I have presented our proposed solution *WaveGuard*, a framework for detecting audio adversarial inputs, to address the security threats faced by ASR systems. Our framework incorporates audio transformation functions and analyzes the ASR transcriptions of the original and transformed audio to detect adversarial inputs. We demonstrate that WaveGuard can reliably detect adversarial inputs from recently proposed and highly successful targeted and untargeted attacks on ASR systems. Furthermore, we evaluate WaveGuard in the presence

of an *adaptive* adversary who has complete knowledge of our defense. We find that only at significantly higher magnitudes of adversarial perturbation, which are audible to the human ear, can an adaptive adversary bypass transformations that compress input to perceptually informed audio representations. In contrast, naive audio transformation functions can be easily bypassed by an adaptive adversary using small inaudible amounts of perturbations. This makes transformations such as LPC and Mel extraction-inversion more robust candidates for defense against audio adversarial attacks.

4.10 Acknowledgements

Chapter 4 is a reprint of the material as it appears in *WaveGuard: Understanding and Mitigating Audio Adversarial Examples*. USENIX Security Symposium, 2021. Hussain, Shehzeen; Neekhara, Paarth; Dubnov, Shlomo; McAuley, Julian; Koushanfar, Farinaz. The dissertation author was the primary investigator and author of this paper.

Part II

Efficient Neural Media Synthesis

Chapter 5

Compute Efficient Design for Neural Media Synthesis

The goal of autoregressive neural networks is modeling the predictors of future values of time series sequences given their past. Autoregressive convolutional neural networks (CNNs) have been widely exploited for sequence generation tasks such as audio synthesis, language modeling and neural machine translation. These architectures use a stack of temporal convolutional layers to model sequences and have achieved exemplary results in audio [4, 164, 165, 166] and text domains [167, 168] with respect to both estimating the data distribution and generating high-quality samples. Wavenet [4] is an example of autoregressive CNN, used for modelling audio for applications such as text-to-speech (TTS) synthesis and music generation. WaveNet has been rated by human listeners to provide substantially more natural sounding audio when compared to the best existing parametric and concatenative systems in TTS applications for both English and Mandarin[4]. Alongside achieving state-of-the art results in the audio synthesis, autoregressive CNN architectures based on ByteNet [167] are prominent for natural language modeling tasks like text generation and machine translation.

Generally, both autoregressive convolutional neural networks (CNNs) and Recurrent Neural Networks (RNNs) [169] are widely popular for sequence modelling tasks. The main advantage of CNN based models is that they can achieve higher parallelism during training and can capture longer time-dependencies as compared to RNN based models [170, 171]. This

allows scaling up CNN based model to deeper architectures and larger datasets thereby achieving state-of-the-art results in speech and text synthesis. However, this comes at a cost of slower inference, since the naive inference implementation of autoregressive CNNs has exponential time complexity in terms of the depth of the network. To overcome this problem, Fast-Wavenet [172] exploits the temporal dependencies by caching redundant computations using fixed-length convolutional queues and thus makes the generation time linear in terms of depth of the network. Such efforts have made it feasible to use autoregressive CNNs for practical sequence generation applications, as an alternative to RNN-based models.

While the fast inference algorithm provides significant speed-up in sequence synthesis over the naive implementation, the inference is still slower than real-time, even on a high-end GPU. For examples, it takes around 120 seconds to generate 1 second of audio using Fast-Wavenet implementation on a NVIDIA TITAN Xp GPU. Prior studies have shown FPGAs to be successful in accelerating the inference of pre-trained neural networks by providing custom data paths to achieve high parallelism. Despite the considerable body of research dedicated to accelerating neural networks for tasks like image classification [173, 174], speech recognition [175, 176], and language modeling using RNNs [177], limited attention has been given to speech and text synthesis.

This work presents pioneering solutions for accelerating CNNs on FPGA platforms, specifically for sequence synthesis in the audio and text domains. First I describe our proposed solution FastWave [178], the first accelerator platform for autoregressive convolutional neural networks, which primarily targets speech synthesis. Next, I detail the extension of FastWave to text synthesis, by redesigning the input and output interfaces of the network and introduce new network components like the embedding layer (Section 5.5.5) and activation functions to make the transition from audio to text model. These details are listed in Section 5.6.3. Furthermore in this chapter, I systematically examine and address the challenges related to resource and runtime efficiency when leveraging autoregressive CNN models for accelerating speech and text synthesis on FPGA. The goal is to optimize the generation of sequential multimedia data such as audio and

text, by efficiently utilizing resources on hardware platforms and minimizing processing time. The primary challenges in deploying autoregressive CNN inference on FPGA are designing modules for dilated convolutional layers, buffers for storing redundant computations using convolutional queues, and dealing with the large depth of these networks which is necessary to maintain high synthesis quality especially in audio. In this work, I address these challenges of deploying large autoregressive convolutional models on FPGAs and perform a wide range of application and architectural optimizations. Furthermore, I comprehensively analyze and compare the performance of both audio and text synthesis implementations on FPGA with the CPU and GPU counterparts. The explicit contributions of this study are as follows:

- Creation of FastWave, the first accelerator platform for autoregressive CNNs. Our platform is generalizable and maintains high performance across different temporal input domains such as audio and text. We deploy a fast inference algorithm on Xilinx XCVU13P FPGA which achieves **11** times faster generation speed than a high-end GPU and **66** times faster audio generation speed than a high-end CPU. For text synthesis, the speed up is **29** times faster than a high-end GPU and **175** times faster than a high-end CPU.

- Development of reconfigurable basic blocks pertinent to autoregressive convolutional networks i.e., dilated causal convolutional layers, convolutional queues, embedding layer and fully connected layer. Our operations are powered by a fully-customizable matrix-multiplication engine that uses two levels of parallelism controlled by tunable parameters.

- Optimization of our accelerator for limited resource deployment settings: We perform accuracy aware pruning targeting the memory bottlenecks of long convolutional queues in deep autoregressive CNNs. We demonstrate that we can preserve the model correctness and performance gains across two different FPGA platforms.

- Exploration of the design space using different architectural and application optimizations, as well as comparing the performance and resource usage. We present extensive evaluation

of throughput and power efficiency for our fully optimized and baseline designs. We integrate a high level of parallelism along with a pipelined design to maximize system throughput.

5.1 FastWave: Accelerating Autoregressive Convolutional Neural Networks

In this section, we describe the first accelerator platform *FastWave* that we developed for autoregressive convolutional neural networks, and address the associated design challenges. We design the Fast-Wavenet inference model in Vivado HLS and perform a wide range of optimizations including fixed-point implementation, array partitioning and pipelining. Our model uses a fully parameterized parallel architecture for fast matrix-vector multiplication that enables per-layer customized latency fine-tuning for further throughput improvement. Our experiments comparatively assess the trade-off between throughput and resource utilization for various optimizations. Our best WaveNet design on the Xilinx XCVU13P FPGA that uses only on-chip memory, achieves 66$\times$ faster generation speed compared to CPU implementation and 11$\times$ faster generation speed than GPU implementation.

5.2 Prior Work on Accelerating DNNs for FPGAs

Prior works have made significant efforts in compressing Deep Neural Networks (DNNs) to support fast energy-efficient applications. However, recent research on DNNs is still increasing the depth of models and introducing new architectures, resulting in higher number of parameters per network and higher computational complexity. Other than CPUs and GPUs, FPGAs are becoming a platform candidate to achieve energy efficient neural network computation [179, 180, 181, 174, 173, 182, 183]. Several works [179, 184, 185] have efficiently mapped DNN computations to different FPGA resources, thereby achieving higher energy efficiency compared to optimized multi-core CPU and GPU implementations. Alternately, some prior works [173, 186] have proposed model adaptation techniques such as parameter encoding, to customize DNNs

for the hardware platforms, thereby reducing the total memory footprint and computational burden of DNNs. Equipped with the necessary hardware for basic DNN operations, FPGAs are able to achieve high parallelism and utilize the properties of neural network computation to remove unnecessary logic. Algorithm explorations also show that neural networks can be simplified to become more hardware friendly without sacrificing the accuracy of the model. Therefore, it has become possible to achieve increased speedup and higher energy efficiency on FPGAs compared to CPU and GPU platforms [187, 179, 188] while maintaining state-of-the-art accuracy. However, these works on FPGA acceleration have mainly focused on neural network based classifiers. Most of these works, focus on accelerating conventional CNN architectures that operate in the image domain. They utilize techniques such as adaptive network quantization, matrix-vector multiplication optimizations and parallelization which tackle challenges pertinent to FPGA acceleration of conventional CNN architectures.

Some prior efforts have also been made in FPGA acceleration of speech recognition, classification and language modelling using RNNs [183, 175, 177]. A recent work [189] presents an acceleration framework for 1-d CNN networks that are used as classifiers or audio transcribers. However, the challenges in *generation* of sequences with *long-term dependencies*, particularly in audio and text domain have not been addressed in any of these prior works. Sequence generation models that can capture long-term dependencies often use autoregressive CNN architectures based on WaveNet or Bytenet. Such networks, present their own set of optimization challenges — naive inference implementation of such networks has many redundant computations at each time-step. These redundant computations can be cached to significantly speed up the inference time at the cost of memory overhead. Also, the memory footprint of such networks increases exponentially with the increasing number of layers thereby making their implementation in resource constrained settings more challenging. In this work, we address such challenges pertinent to autoregressive CNNs i.e., dilated causal convolutional layers, convolutional queues, embedding layers and propose a general purpose FPGA accelerator for such architectures.

5.3 Background and Preliminaries

In this section, we provide a background on autoregressive CNNs. We cover two popular autoregressive CNNs in audio and text domain, WaveNet [4] and ByteNet [167] respectively. We first elaborate on the 1D convolution operation as it is the core computation performed in these models. Next, we explain the general architecture of autoregressive CNNs and specifics of WaveNet and ByteNet models. Finally, we go over the time-efficient inference algorithm Fast-Wavenet that can be generally applied to any autoregressive CNN with dilated convolutional layers.

5.3.1 1D Convolution

The 1D convolution operation is performed by sliding a one dimensional kernel over a 1D input signal. Each output value at position i is produced by calculating the dot product of the kernel and the overlapping values of the input signal, starting from position i. More formally, for an input vector a of length n and a kernel k of length m, the 1D convolution is calculated as follows:

$$(a * k)_i = \sigma_{j=1}^{m} k_j \times a_{i-j+\frac{m}{2}} \tag{5.1}$$

where i is an arbitrary index in the output vector, which has a total length of $n-m+1$. The subscripts denote the indices of the kernel/input vectors.

5.3.2 Autoregressive CNNs

Autoregressive Neural Networks are popularly used for sequence generation tasks which rely on ancestral sampling i.e. the predictive distribution for each sample in the sequence is conditioned on all previous ones. While RNNs are popular autoregressive models, they do not exhibit high receptive field making them unsuitable for modeling sequences with long-term dependencies like audio [170]. In contrast, autoregressive CNN based models use a stack of dilated convolutional layers to achieve higher receptive field necessary for modeling sequences

with long-term dependencies. In this section, we discuss two popular autoregressive CNNs for audio and text domains.

WaveNet: WaveNet [4] is an autoregressive CNN that produces raw audio waveforms by directly modeling the underlying probability distribution of audio samples. This has led to state-of-the-art performance in text-to-speech synthesis [164, 190, 191, 192], speech recognition [193], and other audio generation settings [4, 165, 166]. Popular cloud based TTS synthesis systems such as Google Now and Google Assistant, that produce natural sounding speech, are built on WaveNet architecture [166, 190]. The Wavenet architecture aims to model the conditional probability among subsequent audio samples. The joint probability distribution of waveform sample points $x = x_0, x_1, ..., x_T$ can be written as: $P(x|\lambda) = \prod_{t=1}^{T} P(x_t|x_{t-1}, .., x_0, \lambda)$ where λ denotes the learnable parameters of Wavenet model. During inference, next-sample audio (x_t) generation is performed by sampling from the conditional probability distribution given all of the previous samples, $P(x_t|x_{t-1}, ..., x_1, x_0, \lambda)$.

One possible method for modeling the probability density is via a stack of causal convolutional layers as depicted in Figure 5.1a. The input passes through this stack of convolutional layers and gated activation functions and finally through a softmax layer to get the posterior probability $P(x_t|x_{t-1}, ..., x_1, x_0)$. The downside of this approach is that in order to model long temporal dependencies from samples far in the past, the causal convolutional network requires either many layers or large filters to increase the receptive field. In general, the receptive field is calculated as $\#\,of\ layers + filter_{size} - 1$ which gives a receptive field of 5 in the architecture shown in Figure 5.1. To address this problem, WaveNet leverages dilated convolutions [194, 195] which deliver higher receptive fields without significant increase in computational cost of the model. Dilated convolution is equivalent to performing convolutions with dilated filters where the size of the filter is expanded by filling the empty positions with zeros. In practice, this is achieved by performing a convolution where the filter skips input values with a certain step.

Fig 5.1b illustrates a network with dilated causal convolutions for dilation values of 1, 2, 4, and 8. Here, the input nodes are shown with color blue and the output is shown with

orange. Each edge in the graph correspond to a 1-dimensional convolution (See section 5.3.1), more generally a matrix multiplication. Due to the binary tree structure of the network, the time complexity of computing output at each time-step is $\mathcal{O}(2^L)$ where L is the number of layers in the network, which gets highly undesirable as L increases. Similarly, the total memory required to store the inputs, output, and the intermediate layer features is $\mathcal{O}(2^L)$.

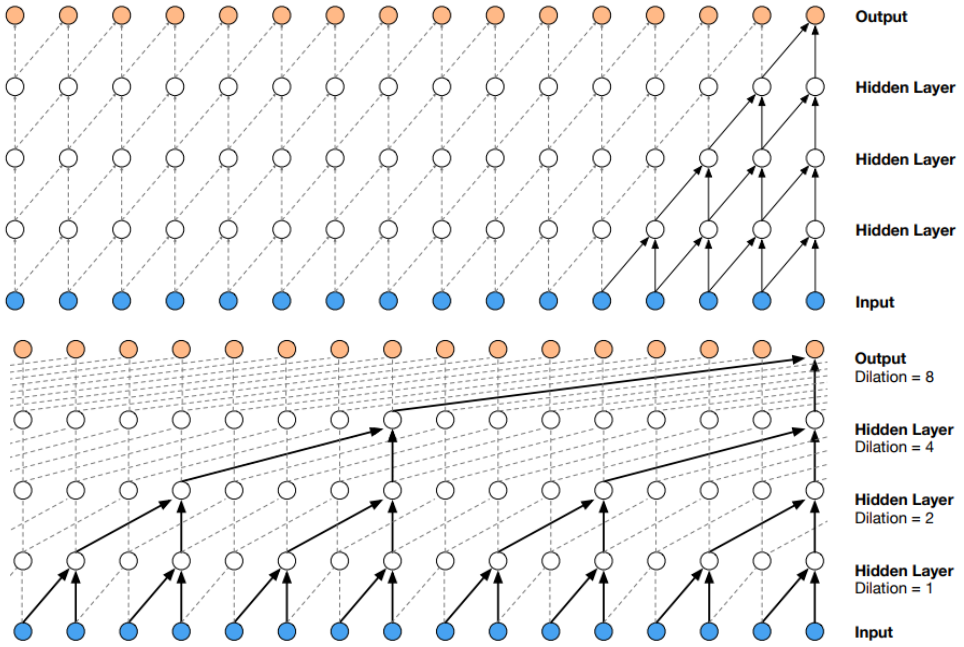


Figure 5.1. a. (Left) Stacked causal convolution layers without any dilations. **b.** (Right) Stacked causal 1-d convolution layers with increasing dilation. Figures from WaveNet paper [4].

ByteNet: ByteNet [167] is another example of autoregressive CNN used for natural language modelling tasks. The ByteNet model is composed of an encoder and decoder, both of which are stack of dilated 1-d convolutional layers. The encoder operates over the source sequence and the decoder operates over the encoded representation to generate the target sequence. ByteNet model can either be used for machine translation task, or for the task of language modelling. For the latter, only the ByteNet decoder is used which is a stack of causal 1-d covolutional layers similar to the WaveNet architecture. The authors demonstrate that ByteNet can achieve state of the art results in character level language modelling.

5.3.3 Fast Inference Algorithm for Autoregressive CNNs

The naïve inference implementation of autorgressive CNNs in Figure 5.1 has many redundant computations when generating a new sample, that is, it recomputes activations that have been already computed for generating previous samples. Fast-Wavenet [172] proposed an efficient algorithm that caches these recurrent activations in queues instead of recomputing them from scratch while generating a new sample. The algorithm uses a per-layer first-in-first-out queue to cache the states to be used in future timestamps.

The queue size at each layer is determined by its corresponding dilation value. Figure 5.2 demonstrates an example 4-layer network and their corresponding queues. For the first layer, the dilation value is 1 and therefore the corresponding queue ($Q1$) only keeps one value. Similarly, the output layer has a dilation value of 8, which means that its queue ($Q4$) will store 8 recurrent values. By removal of redundant computations due to the queue storing mechanism, the computational complexity of Fast-Wavenet is $\mathcal{O}(L)$ where L is the number of layers. The overall memory requirement for queues as well as the intermediate values remains the same as the naïve implementation, i.e., $\mathcal{O}(2^L)$.

The basic queue operations performed in the Fast-Wavenet are as follows (refer to Figure 5.2):

1. **Pop phase**: The oldest recurrent states are popped off the queues in each layer and fed as input to the generation model. These popped off states and the current input are operated with the convolutional kernel to compute the current output and the new recurrent states.

2. **Push Phase**: Newly calculated recurrent states (orange dots) are pushed to the back of their respective layer queues to be used in future time stamps.

Maintaining the convolutional queues in the above manner allows us to handle the sparse convolutional operation and avoid redundant computations and makes the generation algorithm linear in terms of length of the sequence. This algorithm is applicable for any autoregressive

CNN including both WaveNet and ByteNet.

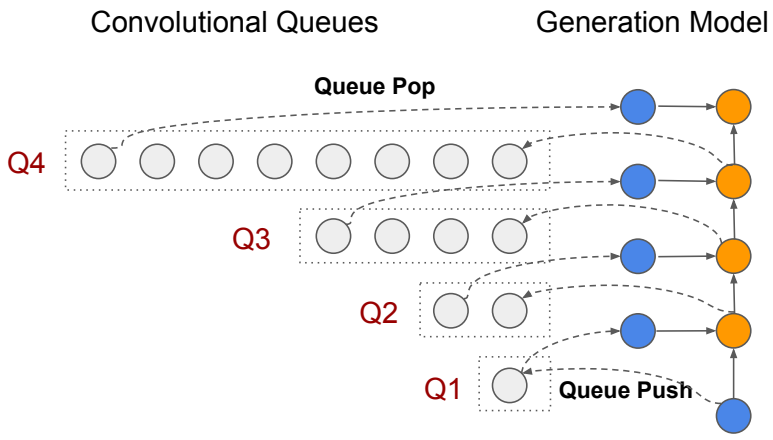


Figure 5.2. Basic queue operations (Push and Pop) performed in Fast inference algorithm to achieve linear time in audio generation.

5.4 Methodology

Our primary objective is to accelerate the inference of autoregressive CNNs for sequence generation on FPGAs. We train two such models as candidates for our acceleration framework: one for the task of audio synthesis and one for the task of text synthesis. Due to the high temporal data density in audio, audio models require a larger receptive field to capture long-term dependencies as compared to text. This in turn requires the network to be very deep which increases the computational and storage cost of such models. When designing an accelerator for such models, it is important to be aware of the system restrictions, particularly those of memory access bandwidth [179, 173]. Accessing off-chip memory is expensive and can limit the throughput of our network, making it important to compress DNNs into an optimal model for efficient inference.

Design Flow: We start with an open source TensorFlow implementation of the Fast-Wavenet algorithm for audio synthesis. We adapt this model for text synthesis by adding an embedding layer and modifying the network architecture. We train the two models - one for the task of audio synthesis and one for the task of language modelling. We save the weights of the

convolutional and fully connected layers of our trained model which are used in the inference stage for generating sequences. We implement the fast inference algorithm in NumPy without using any high level deep learning libraries. This implementation serves as a bridge between the high level TensorFlow and the low level Vivado HLS implementation in C++. On the FPGA platform, we then perform design space exploration on the audio synthesis model since it is a more resource and computation heavy architecture. We apply the best design optimizations on the text synthesis model and evaluate the performance across different hardware platforms: two different FPGA boards, CPU and GPU.

5.4.1 Model Architecture and Training on GPU

We use an open-source TensorFlow implementation of Fast-Wavenet [172] to pre-train our audio and text synthesis networks as follows:

Audio Synthesis: For audio synthesis, the network architecture we use is a stack of two dilated convolutional blocks. Each block consists of 14 convolutional layers with kernel size (filter width) = 2 and dilation increasing in powers of 2 (Table 5.1). Therefore each of the kernels is a 3-dimensional array of shape $2 \times input channels \times output channels$. The number of output channels in each layer is 128 in the baseline implementation. After each convolutional layer there is a *tanh* activation function which serves as the non-linearity in our model as used in the original WaveNet paper [4]. A *tanh* activation normalizes values between -1 and 1 and also allows us to better utilize fixed point data-types in the Vivado implementation without compromising on accuracy.

After the two convolutional blocks, we have a single fully connected layer which maps the activation of size 128 from the last convolutional layer to an output vector of size 256 followed by a softmax normalization layer. The output after the softmax layer is the generated distribution. The target audio is quantized linearly between -1 and 1 into 256 values. The one-hot representation of each sample of size 256 serves as the target distribution at each time-step. The cross entropy loss between the generated and target distribution is back-propagated to train the

Table 5.1. Audio synthesis model architecture: This model uses 2 blocks of dilated convolutional layers with 14 layers each. The column *Queue Size* denotes the number of floating point numbers stored in each queue and is equal to $QueueLength \times InputChannels$.

Block No.	Layer No.	Filter Width	Queue Length	Input Channels	Output Channels	Queue Size
1	1	2	1	1	128	1
1	2	2	2	128	128	256
1	3	2	4	128	128	512
1	4	2	8	128	128	1024
1	5	2	16	128	128	2048
1	6	2	32	128	128	4096
1	7	2	64	128	128	8192
1	8	2	128	128	128	16384
1	9	2	256	128	128	32768
1	10	2	512	128	128	65536
1	11	2	1024	128	128	131072
1	12	2	2048	128	128	262144
1	13	2	4096	128	128	**524288**
1	14	2	8192	128	128	**1048576**
2	1	2	1	128	128	128
2	2	2	2	128	128	256
2	3	2	4	128	128	512
2	4	2	8	128	128	1024
2	5	2	16	128	128	2048
2	6	2	32	128	128	4096
2	7	2	64	128	128	8192
2	8	2	128	128	128	16384
2	9	2	256	128	128	32768
2	10	2	512	128	128	65536
2	11	2	1024	128	128	131072
2	12	2	2048	128	128	262144
2	13	2	4096	128	128	**524288**
2	14	2	8192	128	128	**1048576**

convolutional kernels and weights of the fully connected layer.

Text synthesis: For text synthesis, our neural network architecture utilizes a stack of two dilated convolutional blocks with 10 layers each (Table 5.2). Since text data has lower temporal density as compared to audio, we need a comparatively lower receptive field for language modelling. This is aligned with the architecture of the ByteNet decoder being smaller than the WaveNet audio model. We train a character level neural network architecture on a subset of Shakespearean text with a vocabulary size of 58. To process the inputs, we add an embedding layer with 128 as the embedding dimension. The embedded input is then passed into the stack of dilated

Table 5.2. Text synthesis model architecture: The column *Queue Size* denotes the number of floating point numbers stored in each queue and is equal to $QueueLength \times InputChannels$.

Block No.	Layer No.	Filter Width	Queue Length	Input Channels	Output Channels	Queue Size
1	1	2	1	128	128	128
1	2	2	2	128	128	256
1	3	2	4	128	128	512
1	4	2	8	128	128	1024
1	5	2	16	128	128	2048
1	6	2	32	128	128	4096
1	7	2	64	128	128	8192
1	8	2	128	128	128	16384
1	9	2	256	128	128	32768
1	10	2	512	128	128	65536
2	1	2	1	128	128	128
2	2	2	2	128	128	256
2	3	2	4	128	128	512
2	4	2	8	128	128	1024
2	5	2	16	128	128	2048
2	6	2	32	128	128	4096
2	7	2	64	128	128	8192
2	8	2	128	128	128	16384
2	9	2	256	128	128	32768
2	10	2	512	128	128	65536

convolutional layers. After each convolutional layer we use the *ReLU* activation function since it provides higher accuracy than *tanh* for the task of language modelling. The final fully connected layer in the network maps the output of size of 128 to our vocabulary of size 58.

5.4.2 Optimizing the Design for Different FPGAs

Deep autoregressive CNNs have high memory footprint which posse a significant challenge in designing the FPGA accelerator. Specifically, the primary memory bottle-neck is not the parameters of the convolutional kernels, but convolutional queues which cache the intermediate outputs of the convolutional layers to be used for future predictions. The size of these queues increases exponentially with the depth of the block in the neural network. As highlighted in Table 5.1, the 14-th convolutional queue in each of the blocks stores $1,048,576$ floating point numbers ($\approx 33Mb$). To address this challenge, we take the following measures:

To address this memory challenge, we utilize both BRAMs and URAMs available on the

XCVU13P FPGA. We store all convolutional queues on the BRAMs by default and off-load the 14th convolutional queue of each block onto the URAMs on our board. In this way, we are able to utilize only on-chip memory and achieve higher bandwidth without compromising on audio quality.

The above approach may not be applicable to smaller FPGA boards, e.g., Xilinx Virtex UltraScale VCU108, with lower BRAM capacity and no URAM memory allocation. The baseline architecture shown in Table 5.1 uses 128 channels in each layer. Using 32-bit floating point numbers would result in the memory utilization of the 13-th and 14-th queue in both blocks to exceed the available on-chip memory. In order to fit the inference model on such FPGAs without using any external memory, we *prune* our neural network while maximally preserving accuracy. Recent research on high performance NN accelerators have proposed to either reduce the number of weights [196] or the bit-width used for weight representation [197], which can also help reduce the computation and storage complexity. Pruning is the process of reducing the parameters of convolutional networks, by removing redundancies of the network before invoking inference. This allows us to reduce the number of computational resources required and thereby conduct inference faster with higher power efficiency.

A very effective pruning approach when trading accuracy for the size and the speed, is to reduce the number of channels in each convolutional layer. In our work, we successfully pruned our baseline implementation to reduce the number of channels for every layer. We started with evaluating weights in every layer, and sorting the weights according to increasing order of their $l2$ norm. We then reduced the channels to the desired value by removing the components with the least magnitude. Pruning effectively reduces the total memory budget required for convolutional queues from $\approx 134Mb$ to $\approx 40Mb$. We retrained our model with fewer channels and were able to obtain similar audio quality with minimal prediction error using a lower learning rate. A quantitative comparison between the quality of audio generated from the pruned and original architecture is provided in the Results Section 5.6.

5.4.3 Accelerator Design Overview

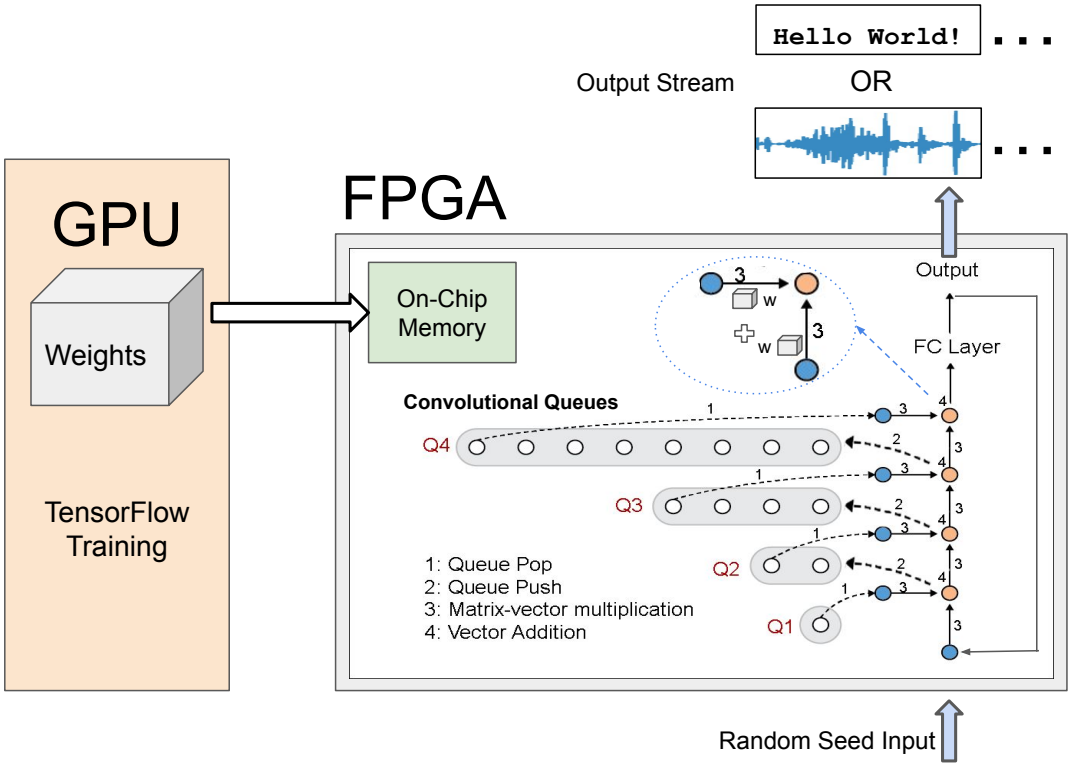


Figure 5.3. Acceleration Methodology for Autoregressive CNN synthesizing audio and text.

The primary objective of our system is to generate an output stream given a seed input. Figure 5.3 shows the overview of our accelerator design. Given a seed input, our system generates an output stream in an autoregressive manner, one-sample at a time. The output sample produced at each time-step is fed back as input to generate the next output sample. During each cycle, as the input goes through all the convolutional layers, the corresponding convolutional queues are updated using push and pop operations as explained in Section 5.3. It is important to note that the entire model including the convolutional queues and the parameters does not use any off-chip memory and are stored in the BRAM and URAM available on the FPGA board. We describe the details of implementing the convolution operations, queue updates and output generation using the fully connected layers in the following section.

5.5 Implementation Details

Our design is composed of 5 main elements: (i) The *dilated convolution* layers, (ii) *the queue control unit*, (iii) the *fully-connected* layer, (iv) the *matrix multiplication engine*, and (v) the *network description module*. An additional element (vi) the *embedding* layer is only necessary for text synthesis models and is invoked by our API when accelerating the text synthesis model.

5.5.1 Optimization of Dilated Convolutional Layer

As explained in the Section 5.3.3, Fast-WaveNet leverages queues to implement the dilated convolutional layers. A convolution of size $= 2$ is used in the WaveNet architecture, and can be implemented as two matrix-vector multiplications followed by vector addition in the manner explained below. Notations used for our variables along with the shapes are listed below:

IC_n : Number of Input channels of layer n.

OC_n : Number of Output channels of layer n.

$O[n]_{(OC_n \times 1)}$: Output of convolutional layer n.

$K[n]_{(2 \times OC_n \times IC_n)}$: Convolutional kernel of layer n.

$Q[n]_{(queueLength \times IC_n)}$: Convolutional queue of layer n.

$$O_1[n] = K[n][0]_{(OC_n \times IC_n)} \times Q[n][0]_{(IC_n \times 1)}$$

$$O_2[n] = K[n][1]_{(OC_n \times IC_n)} \times O[n-1]_{(IC_n \times 1)}$$

$$O[n]_{OC_n \times 1} = O_1[n]_{(OC_n \times 1)} + O_2[n]_{(OC_n \times 1)}$$

In other words, we matrix-multiply the first component of the convolutional kernel with the first element of the queue, and the 2nd component of the kernel with the previous layer's output and then add the two products to obtain the output of any layer. The details of the matrix-vector multiplication engine have been provided in Section 5.5.4.

The output of the convolution layer is then passed to *tanh* activation function. We use the CORDIC implementation available in Vivado HLS math library for applying *tanh* allowing us to optimize our design and memory usage. The output of the dilated convolution module is a vector of length equal to the number of layer output channels.

5.5.2 Cyclic Queue Buffer Unit

In order to reduce the number of operations, Fast-Wavenet aims to remove redundant convolution operations by caching previous calculations in a Queue, thereby reducing the complexity of synthesis to O(L) time. This means that after performing a convolutional operation, we push the compute into the end of the queue and pop the out the first element. These push and pop operations are shown in figure 5.3. As described above the queue in each layer $Q[n]$ is a 2-d array of shape $QueueLength \times InputChannels$. The $QueueLength$ depends on the $dilationFactor$ of the layer and is equal to $2^{dilationFactor}$. We aim to fit our queue computations in the on-chip memory BRAMs and URAMs. Our baseline queue implementation in Vivado HLS used shift operations to perform pop and push functionalities of a queue. The longest queue in our model is of size 8192×24. The shifting of a large number of elements in the queue resulted in very high latency.

To make queue push and pop operations computationally efficient, we implemented our queues using fixed length circular arrays for each layer. This is a lot more efficient than shifting all the elements present in the queue. The push and pop operations are reduced to just overwriting one column of our circular array which is indexed using modulo $QueueLength$ index.

5.5.3 Optimization of Fully-connected Layer

The fully connected layer in WaveNet is a linear layer after all the convolutional layers. This layer is characterized by a weight matrix $W_{channels \times OutputSize}$ and a bias vector $b_{1 \times OutputSize}$. The fully connected layer performs the following operation on $ConvOut$: the output of the last

convolution layer:

$$FinalOutput = ConvOut \times W + b$$

In our design, the weight matrix W has shape 128×256 and bias b has shape 1×256. We use arg-max sampling on the final vector of length 256 to obtain the quantized output value between -1 and 1.

5.5.4 Optimization with Matrix Multiplication Engine

The most computationally-intensive operation in DNN execution is matrix-vector multiplication. FPGAs are equipped with DSP units which offer a high computation capacity together with the reconfigurable logic. The basic function of a DSP unit is Multiplication Accumulation (MAC). Layers in a convolutional neural network take as input a vector $X_{N\times 1}$ and compute the output vector $Y_{M\times 1}$ as formulated below:

$$Y = f(WX + b) \tag{5.2}$$

where f(.) is a nonlinear function, $W_{M\times N}$ is the 2D matrix of the weights and $b_{M\times 1}$ is a vector of bias values. As can be seen, each layer is computing a vector-matrix multiplication and a vector-vector addition. In order to optimize the design and make efficient use of the DSP blocks, we proposed a parallelized approach to convert layer computations into multiple MAC operations. Figure 5.4 (a) presents our method to parallelize the matrix-vector multiplication computations.

We define two levels of parallelism for our engine which control the parallel computations with parameters *num_parallel_in* and *num_parallel_out*, denoting the level of parallelism in the input and output, respectively. For the first level of parallelism, multiple rows of the weight matrix are processed simultaneously by dividing it into chunks, each having *num_parallel_out* rows. In each round, a chunk of the weights matrix is copied to one of the weight buffers while the other weight buffer is fed into the *dot product* modules together with a copy of the input vector. The iterations end when all rows of the weight matrix have been processed. For the second level

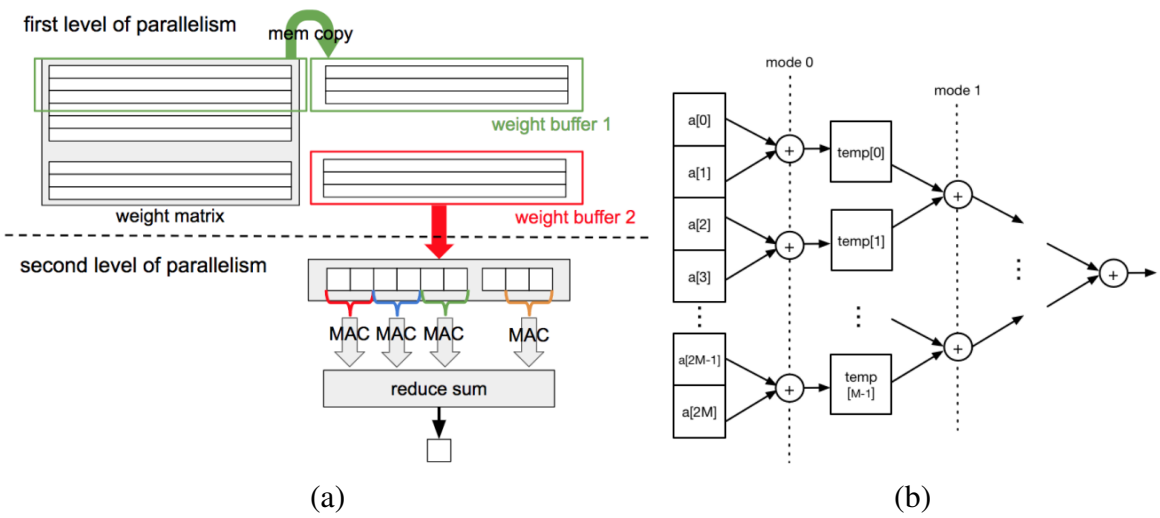


Figure 5.4. (a) Schematic representation of the matrix multiplication engine and the corresponding parallelization factors. (b) Realization of the tree-based vector reduction algorithm.

of parallelism, each *dot-product* function partitions its input vectors into *num_parallel_in* chunks and concurrently executes MAC operations over the partitioned subsets. The accumulated results of the subsets are then added together within the *reduce_sum* function to compute the final output. The *reduce_sum* module performs a tree-base reduction algorithm as outlined in Figure 5.4 (b). The reduction function takes an array of size $2M$ as its input (array a) and oscillates between 2 different modes. In mode 0, the function reduces a by using temp as a temporary array. In mode 1, temp is reduced using a. The result is returned based on the final mode.

The aforementioned parameters *num_parallel_in* and *num_parallel_out* are individually defined for each of the layers to enable fine-tuning according to the per-layer requirements. Due to the limited number of available resources on the FPGA platform, it is not possible to define high parallelization factors for all layers. As such, we give priority to layers with higher computational complexity, i.e., higher number of input and output channels, by instantiating their corresponding matrix multiplication engines with larger parallelization parameters.

Table 5.3. Design space exploration on Xilinx XCVU13P FPGA. We report the resource utilization of each of our designs. The percentages reported indicate percentage of resources utilized by the design.

	Resource Utilization				
	BRAM (Mb)	**URAM (Mb)**	**FF (K)**	**LUT (K)**	**DSP48E**
Design / Available Resources	94.5	360	3456	1728	12288
FloatingPointBaseline	93 (98%)	144 (40%)	35 (1%)	86 (5%)	288 (2%)
FloatingPointCQ	93 (98%)	144 (40%)	35 (1%)	83 (5%)	330 (3%)
FloatingPointPipeline	93 (99%)	144 (40%)	231 (7%)	231 (13%)	475 (4%)
FixedPointUnrolling	79 (84%)	144 (40%)	22 (1%)	146 (8%)	660 (5%)
FixedPointMME (Best) - Audio	90 (96%)	144 (40%)	425 (12%)	1669 (97%)	540 (4%)
FixedPointMME (Best) - Text	32 (34%)	0	138 (4%)	432 (25%)	1068 (8%)

5.5.5 Optimization of Embedding Layer

An embedding layer is used to numerically encode discrete inputs like characters in text synthesis models and is not utilized by audio synthesis models. Therefore, we design an embedding layer module for our accelerator for the purpose of text synthesis. In this module, we design the embedding operation as a lookup function to find the embedded vector for a given character in an embedding matrix. For this use case, the embedding matrix has dimensions *Vocabulary Size* $\times$ *Embedding Dimension*. The embedding dimension is a hyper-parameter set to 128 and serves as the number of input channels in our text synthesis network. We use pipelining to optimize the latency of this operation.

5.5.6 Network Description Module

In this module, we implement the overall architecture of our network as a stack of dilated conventional layers and perform queue update operations followed by a fully connected layer. This module instantiates the corresponding function for each network layer and manages the layer inter-connections. Since each layer is independently instantiated, we can use custom dilation, channels and parallelization parameters for each layer. After the last fully connected layer, to make audio generation deterministic we use arg-max sampling. This allows us to bypass

Table 5.4. Design space exploration on Xilinx XCVU13P FPGA. We report the performance and correctness of each of our designs. MSE and LSD are measured by comparing the generated audio from FPGA against corresponding GPU implementations. *Acc.* indicates the prediction accuracy for text synthesis

	Performance			Correctness		
	Latency	**Clock-Cycle Time (ns)**	**Through-put (Hz)**	**MSE**	**LSD**	**Acc.(%)**
FloatingPointBaseline	12110989	8.83	9.4	0	0	-
FloatingPointCQ	6170104	8.83	18.4	0	0	-
FloatingPointPipeline	612952	8.88	183.7	0	0	-
FixedPointUnrolling	293914	8.75	388.8	0.006	0.104	-
FixedPointMME (Best) - Audio	78275	8.66	1475.2	0.006	0.104	-
FixedPointMME (Best) - Text	28932	6.52	5301.2	0	-	96.1

the final softmax layer since we can directly apply the arg-max function on the output of our final fully connected layer.

5.6 Results and Experiments

In this section, we evaluate the effect of our optimizations, namely cyclic queues, pipelining, loop unrolling and customized matrix multiplication engine, by conducting extensive design space exploration. Our design experiments are synthesized for the Xilinx XCVU13P board using Xilinx Vivado HLS 2017.4. In particular, we discuss the experimental techniques applied to reduce resource utilization and latency of our baseline implementation. We further provide a comprehensive comparison of our best designs with CPU and GPU implemented baselines in terms of throughput and power efficiency.

5.6.1 Evaluation Metrics

To evaluate the accuracy of our implementation we compare the output generated from our FPGA implementation with the golden output generated by the TensorFlow GPU implementation for the same initial seed.

For audio, since the outputs are continuous values, we use the MSE and Log-Spectral

Distance metrics to compare any two audio signals x_1, x_2 of the same length. For text we use the prediction accuracy to evaluate the correctness of our discrete outputs. We describe these metrics below:

- **Mean Squared Error (MSE)**: The mean squared error (MSE) between any two given signals x_1, x_2 is the mean squared error between their representations in time domain as a sequence of floating point numbers. That is, $MSE = mean((x_1 - x_2)^2)$. The MSE losses reported are from the comparison of the entire waveform i.e. the total mean squared error from all 32000 samples.

- **Log-Spectral Distance (LSD)**: The log-spectral distance [198] is a commonly utilized metric, obtained as the root mean square error between the normalized log-spectra of given signals. Given two signals x_1, x_2, we calculate log-spectral distance between them as follows:

$$\begin{aligned} ps_1 &= (abs(stft(x_1)))^2 \\ ps_2 &= (abs(stft(x_2)))^2 \\ ls_1 &= normalize(log(ps_1)) \\ ls_2 &= normalize(log(ps_2)) \\ LSD &= RMSE(ls_1, ls_2) \end{aligned} \tag{5.3}$$

  Here ps_1, ps_2 are the power spectra and ls_1, ls_2 are the normalized log spectra of signals x_1, x_2 respectively. The normalization is performed across all frequencies in the log spectrograms.

- **Prediction Accuracy using teacher forcing (Acc.)**: To evaluate correctness of discrete outputs of our text synthesis inference, we use the accuracy metric in which the output of the TensorFlow implementation serves as the ground-truth. Note that we use teacher forcing for this evaluation, i.e. we feed in the ground truth inputs for time-steps 1 through $n-1$ and evaluate the correctness of the n_th output. We estimate the accuracy of predictions

over a sequence of 1000 characters.

- **Qualitative Evaluation**: Along with the quantitative results, we also provide log spectrogram visualizations of the audio signal generated using our FPGA implementation and the golden-output audio signal generated from the TensorFlow implementation in Figure 5.5.

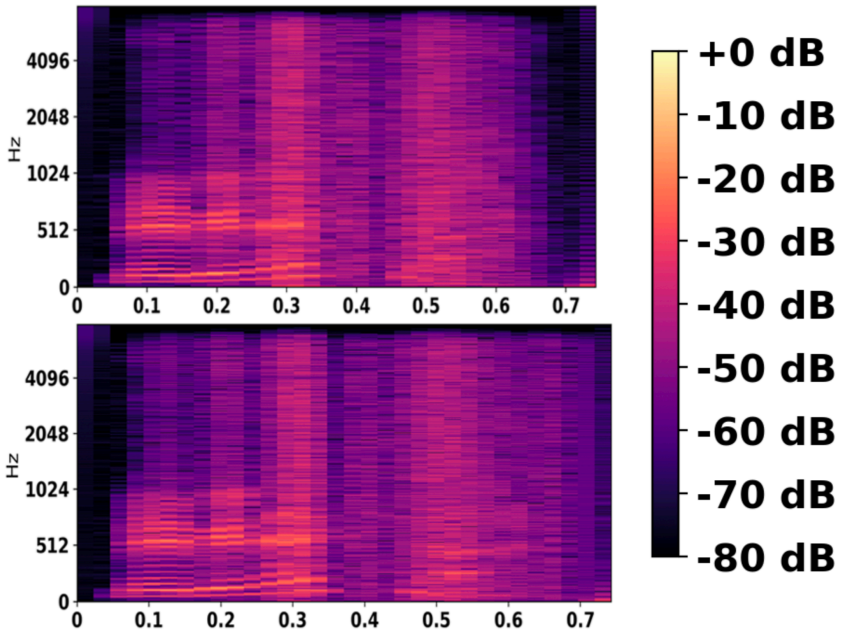


Figure 5.5. Log-Spectrograms of the 2-second audio generated from the TensorFlow implementation (top) and FPGA FixedPointMME design implementation (bottom).

5.6.2 Design Space Exploration

We implement several variants of our designs to study the effect of various optimization techniques in isolation and in combination with other techniques. We perform the design space exploration on the Xilinx XCVU13P FPGA. The resource utilization, performance (throughput) and error in the generated audio, for each of the following designs have been reported in Table 5.3 and Table 5.4 and Figure 5.6. Throughput measures the number of sequence tokens generated per second by our implementation of an autoregressive model. For text, each token corresponds to one character and in the case of audio, one second of audio contains 16000 samples if audio is

sampled at 16KHz. Since the audio synthesis model is computationally more expensive than the text synthesis model, we perform the design optimization on the audio model and apply the best design principles on the text model.

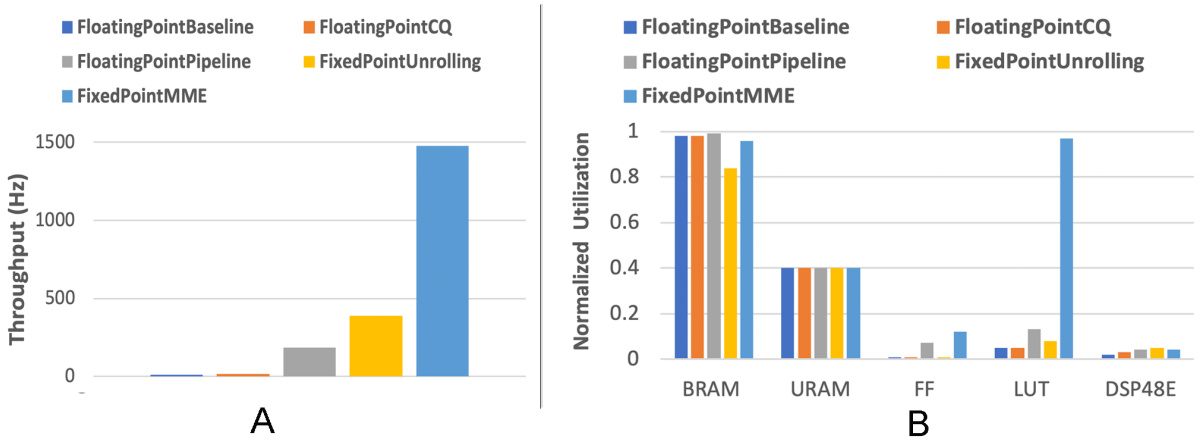


Figure 5.6. Design space exploration for the audio synthesis model on the Xilinx XCVU13P FPGA. **A:** Throughput (Number of Samples generated per second) of different designs. **B:** Normalized Resource Utilization of different designs.

Baseline Floating Point Implementation *(FloatingPointBaseline)*

The baseline design of our network is comprised of modules to implement the basic functionality of each layer, queue, initialization of weights from stored data files and forward propagation. We use a array-shifting implementation of queue which results in a fairly high latency as shown in Table 5.4 because of the very long queues (length = 8192 and 4096) in the **13th** and **14th** layers of our design. For matrix vector multiplication we use simple for loops without any optimization.

Floating Point + Cyclic Queue *(FloatingPointCQ)*

In this design, we replace our shifting based queue implementation with a cyclic queue implementation that uses dynamic indexing to produce the same effect as push and pop operations. This helps reduce latency substantially since shifting operations in the longer queues was the bottleneck in our baseline design. The resource utilization however, stays almost the same as our baseline design.

Floating Point + Cyclic Queue + Pipelining *(FloatingPointPipeline)*

In this design, we modify the above design and add pipelining pragma in the dot product computation and queue update operations. Pipelining the above design helped increasing the throughput substantially at the cost of higher resource utilization.

Fixed Point + Cyclic Queue + Unrolling *(FixedPointUnrolling)*

Including both Cyclic Queue and Pipelining optimization, we switch to fixed point operations from floating point operations. Since the order of magnitude of our kernels, inputs, activations and outputs is nearly the same, we keep a common data-type across all of them. After some experimentation, we found that Loop Unrolling outperforms pipelining in terms of both resource utilization and throughput for fixed point data-types. We use *loop unrolling factor = 8* for the inner loop of our dot product and also the queue update operations. We observe a trade-off between precision and resource utilization for different fixed point bit width and chose `ap_fixed<27,8>` (8 bits for the integer and 19 bits for the fractional part)since it gives reasonable MSE under the constraints of resources.

Fixed Point + Matrix Multiplication Engine *(FixedPointMME - Best)*

For our best design, we use fixed-point implementation in a parallelized approach to convert layer computations into multiple MAC operations (refer to Section 5.5.4 for details). For the first dilated convolution layer we set *num_parallel_out* and *num_parallel_in* as 1 since the number of input channels is just 1. For all other layers, including the fully connected layer we set *num_parallel_out* as 8 and *num_parallel_in* as 4 to get the best throughput under the constraint of available resources.

5.6.3 Design Modifications for Text Synthesis

For our text synthesis model, we use the best design found using our design space exploration on the audio model and change the input and output interfaces to adapt the model for

Table 5.5. Accelerator results on Xilinx Virtex UltraScale VCU108 FPGA (Smaller board). We report the resource utilization for each design implementation. The percentages reported indicate percentage of resources utilized by the design.

	Resource Utilization			
	BRAM (Mb)	**FF (K)**	**LUT (K)**	**DSP48E**
Design / Available Resources	60.8	1075	537.6	768
FloatingPointCQPruned	2881 (83%)	335 (43%)	43323 (4%)	87108 (16%)
FixedPointUnrollingPruned	2456 (71%)	660 (85%)	26441 (2%)	125206 (23%)
FixedPointMME (Best) - Audio	53 (87%)	957 (89%)	65 (12%)	684 (91%)
FixedPointMME (Best) - Text	42 (69%)	22 (2%)	135 (25%)	276 (35%)

Table 5.6. Accelerator results on Xilinx Virtex UltraScale VCU108 FPGA (Smaller board). We report performance and measured error in generation for each design implementation. MSE and LSD are measured by comparing the generated audio from FPGA against corresponding GPU implementations.

	Performance			Correctness		
	Latency	**Clock-Cycle Time (ns)**	**Throughput (Hz)**	**MSE**	**LSD**	**Acc.%**
FloatingPointCQPruned	1935599	8.75	59.04	0	0	-
FixedPointUnrollingPruned	182218	8.75	627.19	0.007	0.110	-
FixedPointMME (Best) - Audio	100169	8.75	1140.9	0.007	0.110	-
FixedPointMME (Best) - Text	52612	8.66	2194.8	0	-	96.1

characters instead of audio. More specifically:

- We implement the embedding table lookup for the input layer. We use a loop unrolling 4 in the loop that reads the embedding vector. We change the number of input channels of the first convolutional layer to 128.

- We use *ReLU* instead of *tanh* as the activation function for the convolutional layers. As discussed earlier, we found the network to converge better for the *ReLU* activation function for the text synthesis task. Implementing *ReLU* is straightforward because it simply gates the input based on the sign of the input. We found this also leads to a higher per layer throughput as compared to the audio model (shown in Table 5.4) leading to significantly faster synthesis as compared to the GPU and CPU implementations.

- We change the final fully connected layer to map to our vocabulary of 58 characters.
- Since, the text synthesis network is smaller than the audio model, we increase the bit-width of our fixed-width datatype to 32 bits and use 16 bits for integer and 16 bits for the fractional part. Using this fixed width datatype, we are able to achieve 96.1% prediction accuracy against the golden output obtained using the TensorFlow implementation.

The resource utilization, performance and correctness metrics for the text synthesis network are reported in Table 5.3 and Table 5.4.

5.6.4 Design Optimization for smaller FPGA Platforms

As discussed in Section 5.4.2, in-order to fit the model on a smaller board: Xilinx Virtex UltraScale VCU108 FPGA, we need to perform accuracy aware pruning that targets the memory bottleneck of deeper architectures. While we are able to fit our text synthesis model on the smaller board without pruning the network, for the audio model, we replace the baseline architecture with the pruned architecture. The high BRAM utilization in the baseline design is mainly due to the queues in layer 13 and 14 containing 4096×128 and 81292×128 floating point numbers respectively. We pruned the network so that the channels required in these queues get reduced from 128 to 24. Upon retraining the pruned model, we observe only a slight increase in prediction error where MSE increases from 0.006 to 0.007, while achieving a significantly lower memory footprint. This helped reduce both the BRAM utilization and also increased the throughput since the number of scalar multiplications and accumulation operations got reduced. We also reduce the parallelization factors of the matrix multiplication engine to fit both the text and audio synthesis models on the smaller board. For the first dilated convolution layer we set *num_parallel_out* and *num_parallel_in* as 1 since the number of input channels is just 1. For all other layers, including the fully connected layer we set *num_parallel_out* as 4 and *num_parallel_in* as 2 to get the best throughput under the constraint of available resources. The resource utilization, performance and correctness of our designs on the Xilinx Virtex UltraScale

VCU108 FPGA are reported in Table 5.5 and 5.6 respectively.

5.6.5 Performance and Power Analysis

Table 5.7 illustrates the performance and power consumption for our implemented designs and a highly optimized CPU and GPU implementation. We benchmark the optimized Tensorflow implementation of Fast-Wavenet on two GPUs: NVIDIA TITAN Xp and Nvidia Tesla V100. The CPU implementation is the NumPy inference program written by us and optimized fully. We measure the power consumption for the GPU benchmarks using the NVIDIA power measurement tool (*nvidia-smi*) running on *Linux* operating system which is invoked during program execution. For our FPGA implementations, we synthesize our designs using Xilinx Vivado v2017.4. We then integrate the synthesized modules accompanied by the corresponding peripherals into a system-level schematic using Vivado IP Integrator. The frequency is set to 150 MHz and power consumption is estimated using the synthesis tool. We perform this evaluation on two FPGA platforms: the larger Xilinx XCVU13P (FPGA 1) and the smaller Xilinx Virtex UltraScale VCU108 (FPGA 2) platforms.

Table 5.7. Power Consumption and Wall-Clock time required when generating 1-second audio for different implementations on different hardware platforms. FPGA 1 refers to Xilinx XCVU13P and FPGA 2 refers to Xilinx Virtex UltraScale VCU108.

Implementation	**Time (in seconds) for 1-Second Audio Generation**	**Power (W)**
CPU (Numpy)	732	
GPU - NVIDIA Titan Xp (TensorFlow)	120	70.0
GPU - NVIDIA Tesla V100 (TensorFlow)	85	66.0
FPGA 1 - FloatingPointPipeline	87	10.2
FPGA 1 - FixedPointUnrolling	41	7.6
FPGA 1 - FixedPointMME (Best-Audio)	**11**	**23**
FPGA 2 - FixedPointMME (Best-Audio)	14	9.2

As shown in Table 5.7, our best FPGA implementation achieves **11**× speed-up in audio generation while being **3**× more power efficient as compared to NVIDIA Titan Xp GPU. As compared to a NumPy based CPU implementation, our best design is **66**× faster. For text

Table 5.8. Power Consumption and Wall-Clock time for networks required when generating 16000 characters using our text synthesis network on different hardware platforms. FPGA 1 refers to Xilinx XCVU13P and FPGA 2 refers to Xilinx Virtex UltraScale VCU108.

Implementation	Time (in seconds) for generating 16000 characters	Power (W)
CPU (Numpy)	525	
GPU - NVIDIA Titan Xp (TensorFlow)	86	69.0
GPU - NVIDIA Tesla V100 (TensorFlow)	61	66.0
FPGA 1 - FixedPointMME (Best-Text)	**3.0**	**17.6**
FPGA 2 - FixedPointMME (Best-Text)	7.3	8.5

synthesis, the speedup is even higher because of the efficient *ReLU* implementation on the FPGA platforms. As shown in Table 5.8, our best accelerator design achieves **29**× speed-up in text generation while being **4**× more power efficient as compared to NVIDIA Titan Xp GPU. Additionally, our accelerator design achieves **175**× faster generation speed compared to NumPy based CPU implementation for text synthesis.

5.7 Conclusion

In this chapter, I have described the design and development of our proposed solution FastWave: the first accelerator platform for deep autoregressive convolutional neural networks. While prior works have proposed algorithms for making the inference of such networks faster on GPUs and CPUs, they do not exploit the potential parallelism offered by FPGAs. We have developed a systematic approach to accelerate the inference of autoregressive CNNs by optimizing their fundamental computational blocks and utilizing only on-chip memory. We additionally optimize the design for smaller resource constrained FPGA devices by effectively pruning the memory bottleneck of deep autoregressive CNNs. We demonstrate the effectiveness of using various FPGAs for applications like audio and text synthesis by achieving a significant speed-up over prior efforts on CPU and GPU based implementations.

5.8 Acknowledgements

Chapter 5 is a partial reprint of the material as it appears in *FastWave: Accelerating Autoregressive Convolutional Neural Networks on FPGA*. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2019. Hussain, Shehzeen; Javaheripi, Mojan; Neekhara, Paarth; Kastner, Ryan; Koushanfar, Farinaz. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Data Efficient Training for Neural Speech Synthesis

In recent years, there has been a significant advancement in speech synthesis using neural networks. Synthesizing natural-sounding speech using neural networks typically requires several hours of speech data for the target speaker. The process of collecting such high-quality speech data and training neural networks is often very costly and time consuming. The process requires recording a new speaker's voice, cleaning up the recorded data and training the spectrogram synthesis and vocoder models. Since many speaking characteristics for a given language are shared amongst different speakers, it is desirable to reuse the knowledge of a pre-trained TTS model when learning to synthesize the voice of a new speaker.

To address the above challenge, my research investigates methods that allow high-quality speech synthesis in data-limited settings. More precisely, I propose training frameworks based on transfer learning to effectively adapt pre-trained speech representation extractors and synthesizers for new tasks. In this chapter, I describe the speech synthesis frameworks that I have developed to allow speech generation using only a few seconds of data of the target speaker.

In Section 6.1, I describe the first framework that tackles the problem of *Voice Cloning* where the goal is to perform text-to-speech synthesis for a target speaker in a limited data settings. While previously proposed voice cloning systems can generate speech from text for a new speaker, they leave out control over various style aspects of speech. My proposed framework

addresses this challenge by using explicit conditioning with style information.

In Section 6.2, I discuss a data-efficient voice conversion framework. Voice conversion is the task of modifying a speech signal from a source speaker to match the vocal qualities of the target speaker. While traditional voice conversion systems [199, 200] rely on parallel training data with multiple speakers saying the same sentence, there has been a recent surge in voice conversion systems trained on non-parallel multi-speaker datasets [201, 202, 203, 204]. In order to perform non-parallel voice conversion, it is necessary to disentangle speech into representations describing the linguistic content and speaker characteristics. Synthesizing speech from these disentangled features allows voice conversion by swapping the speaker embedding of a given utterance with a target speaker. The motivation behind developing such voice conversion systems is to remove the dependence on text as training data. This results in universal language-independent models that do not require transcribed speech files for training and is particularly useful for low-resource languages, where there is a scarcity of audio training data and often a lack of parallel text transcripts. In my research, I propose a voice conversion framework that addresses the challenges in existing voice conversion systems and achieves state-of-the-art results by disentangling content and speaker information features from representations learned using self-supervised learning.

6.1 Expressive Neural Voice Cloning

The goal of voice cloning is commonly formulated as learning to synthesize the voice of an unseen speaker using only a few seconds of transcribed or untranscribed speech. This is typically done by embedding speaker-dependent information from the available speech samples of the new speaker, and conditioning a trained multi-speaker Text-to-Speech (TTS) model on the derived speaker embedding [205, 206]. While such a system can achieve promising results in closely retaining speaker-specific characteristics in the cloned speech, it does not offer control over other aspects of speech that are not contained in the text or the speaker-specific embedding.

These aspects include variation in tone, speaking rate, emphasis and emotions.

Adapting multi-speaker TTS models for voice cloning requires scaling up model training to a large multi-speaker TTS dataset, containing several minutes of transcribed speech from thousands of speakers. High speaker diversity in the training data is important to achieve generalization on unseen speakers [205, 206]. The goal of our voice conversion framework is to perform TTS synthesis for an unseen speaker with control over the style aspects of generated speech. As a first step in this direction, we train a TTS model conditioned on speaker encodings and latent style tokens [207] on a large multi-speaker dataset. While this model is able to generate voices for unseen speakers, we find that the results fall short in terms of speech naturalness and style control during synthesis. Our results suggest that learning meaningful latent style aspects is difficult when training on a large multi-speaker dataset containing speech with mostly neutral style and expressions.

To address problem of disentangling style and speaker characteristics on a large multi-speaker dataset containing mostly style-neutral speech, we propose a voice cloning model that is conditioned on both latent and heuristically derived style information. Specifically, we condition our TTS synthesis model on (i) text, (ii) speaker encoding (iii) pitch contour of the target speech and (iv) latent style tokens [207]. By conditioning synthesis on various style aspects and speaker embeddings derived from the target speech, we are able to train a model that offers fine-grained style control for synthesized speech. To adapt inference for an unseen speaker, we can either perform zero-shot inference or fine-tune the synthesis model on the limited text and speech pairs for the new speaker. Through both quantitative and qualitative evaluations, we demonstrate that our proposed model can make a new voice express, emote, sing or copy the style of a given reference speech.

6.1.1 Voice Cloning Framework

Our expressive voice cloning framework is a multi-speaker TTS model that is conditioned on speaker encodings and style aspects of speech. Style conditioning in expressive TTS models

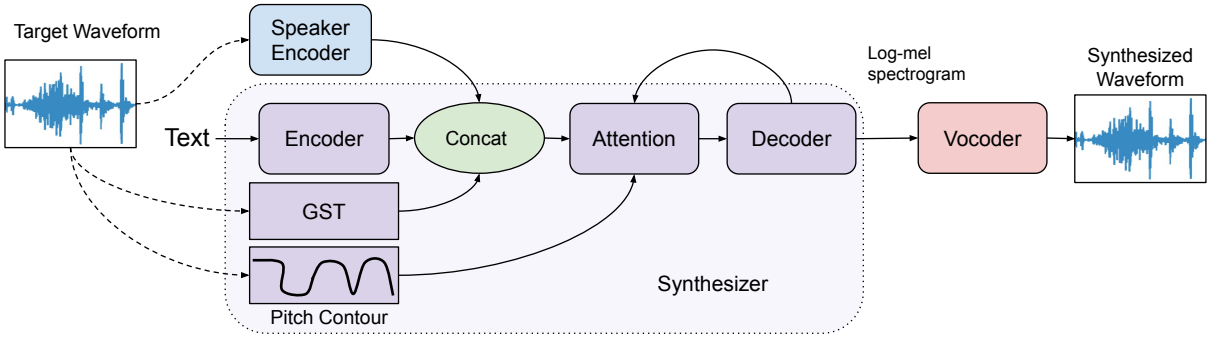


Figure 6.1. Expressive Voice Cloning Model: Tacotron-2 TTS model conditioned on speaker and style characteristics derived from the target audio of a given text. At inference time, the model can be provided independent references for style and speaker encodings to achieve expressive voice cloning.

is popularly done by learning a dictionary of latent style vectors called Global Style Tokens (GST) [207]. While GSTs can learn meaningful latent codes when trained on a dataset with high variation in expressions, we empirically find that it offers limited style control when trained on a large multi-speaker dataset with mostly neutral prosody.

Signal processing heuristics like the Yin algorithm [208] can derive the fundamental frequency contour (pitch contour) and voicing decisions from speech, which can be useful for expressive speech synthesis. We find that using a combination of latent and heuristically derived style information in the TTS model not only provides fine-grained control over the style aspects of synthesized speech, but also scales up to a large multi-speaker dataset to produce more natural sounding audio for an unseen speaker.

Speaker Encoder

Speaker conditioning in multi-speaker TTS models is usually done using a lookup in the speaker embedding matrix which is randomly initialized and trained end-to-end with the synthesizer. While such a framework learns speaker-specific information via the embedding vectors, synthesis cannot be generalized to unseen speakers. To adapt the multi-speaker TTS model for the goal of voice cloning, the speaker embedding layer can be replaced with a speaker encoder that derives speaker specific information from the target waveform. In this setting,

the speaker encoder can obtain embeddings for speakers not seen during training using a few reference speech samples. To obtain meaningful embeddings, the speaker encoder should be trained to discriminate between different speakers for the task of speaker verification [209]. We follow the speaker encoder architecture described in [209, 210]. The speaker encoder is trained to optimize a generalized end-to-end speaker verification loss [209], that encourages high cosine similarity between embeddings from same speaker and low similarity between different speaker embeddings. During inference, each utterance is broken into smaller segments of 1,600 ms with 1,000 ms overlap between consecutive segments. The final embedding is estimated by averaging the embedding of each individual segment.

Mel-Spectrogram Synthesizer

The goal of our synthesis model is to disentangle the style and speaker-specific information in speech by conditioning our TTS synthesis model on the speaker encoding and various style aspects. To this end, we adapt the synthesis model used in Mellotron [211] for the task of voice cloning. Mellotron is a multi-speaker TTS model that extends Tacotron 2 GST [207] by additional conditioning on pitch contours and speaker embeddings. To adapt Mellotron for voice cloning, we remove the speaker embedding layer and replace it with the speaker encoder network described in Section 6.1.1.

At its core, our synthesis model based on Tacotron 2 [155], is an LSTM based sequence-to-sequence model composed of an encoder that operates on a sequence of characters and a decoder that generates the individual frames of the mel spectrogram while attending over the encoded representations. Along with the encoded representation for text, we concatenate the speaker encoding (obtained from the speaker encoder) and the GST embedding at each time-step. The GST embedding is obtained by querying a dictionary of latent style vectors with the target mel-spectrogram using a multi-headed attention mechanism described in [207]. Decoding occurs in an autoregressive manner where we synthesize one mel spectrogram frame at a time by providing the fundamental frequency (from the pitch contour) and the mel spectrogram of the

previous frame as the input to the decoder. The pitch contours are derived from the target speech using the Yin algorithm with harmonicity thresholds between 0.1 and 0.25.

In this way, we can factor mel-spectrogram synthesis into the following variables: *text (t)*, *speaker encoding (s)*, *pitch contour (f_0)* and *latent style embedding obtained from GST (z)*. Formally, our synthesizer is a generative model $g(t, s, f_0, z; W)$ that is parameterized by trainable weights W, trained to optimize a loss function L that penalizes the differences between the generated and ground truth mel spectrogram. That is,

$$\min_{W} \mathbb{E}_{(t_i,a_i)\sim D} \{L(g(t_i, s_i, f_{0i}, z_i; W), mel_i)\} \tag{6.1}$$

where D is the dataset containing text and audio pairs (t_i, a_i). The variables $(s_i, f_{0i}, z_i, mel_i)$ are all derived from the target waveform a_i. For the loss function L, we use the L2 loss between the generated and ground truth mel spectrograms.

During training, the synthesizer learns another latent variable: the attention map between the encoder and decoder states which captures the alignment between text and audio. Following the notation used in [211], we call this latent variable *rhythm*, since it controls the timing aspects of synthesized speech. Note that unlike other style aspects which can be obtained directly from a_i, deriving *rhythm* requires both text and audio (t_i, a_i). In our experiments, we obtain the *rhythm* by using our synthesizer as a forced-aligner. That is, for a given text and audio pair, we derive the attention map between the encoder and decoder states by doing a forward pass through our model using teacher forcing. Therefore, during inference, our synthesizer g can be explicitly conditioned on rhythm r derived from some text and audio pair: $g(t, s, f_0, z, \boldsymbol{r}; W)$.

While the style aspects are obtained from the target waveform of the same speaker during training, we can use a different reference audio and text pair during inference. For example, we can transfer the pitch contour and rhythm of a style reference audio S from a different speaker to

the voice of a given target speaker T as follows:

$$mel = g(t_S, s_T, f_{0S}, z_T, r_S; W) \tag{6.2}$$

The output *mel* should have the same pitch and rhythm as the style reference S and should retain the latent style aspects and voice of the target speaker T.

Additionally, to assess the importance of pitch contours during training, we train another TTS model that is conditioned only on the latent style aspects obtained using GST. We use the same Tacotron2 architecture and GST module as our proposed model. Formally, this alternative synthesizer $g(t,s,z;W)$ is trained to optimize the same objective as Equation 6.1:

$$\min_{W} \mathbb{E}_{(t_i,a_i)\sim D} \{L(g(t_i, s_i, z_i; W), mel_i)\} \tag{6.3}$$

We refer to this alternative model as *Tacotron2 + GST* in our experiments. Similar to our proposed system, this model can also be additionally conditioned on rhythm. Since we are not explicitly conditioning the model on pitch contours, we expect the pitch variation in speech to be captured as part of the latent style tokens. We empirically demonstrate that using only latent style representation on a large multi-speaker dataset with neutral prosody offers limited style control and audio naturalness.

Vocoder:

For decoding the synthesized mel-spectrograms into listenable waveforms, we use the WaveGlow [212] model trained on the single speaker Sally dataset [211]. An advantage of WaveGlow over WaveNet [4] is that it allows real-time inference, while being competitive in terms of audio naturalness. The same vocoder model is used across all experiments and datasets. We find that the vocoder model trained on a single speaker generalizes well across all speakers in our datasets.

6.1.2 Cloning Techniques: Zero-Shot and Model Adaptation

We adopt the following two approaches for cloning the voice of a new speaker from a few transcribed or untranscribed speech samples:

Zero-Shot: For zero-shot voice cloning, we derive the speaker embedding by taking the mean followed by L-2 normalization of the speaker encodings of the individual samples of the target speaker. Since speaker encodings are obtained directly from the waveforms, we do not require audio transcriptions of the new speaker for zero-shot voice cloning.

Model Adaptation: When transcribed samples of a new speaker are available, we can fine-tune our synthesis model using the text and audio pairs. As shown in Neural Voice Cloning [205], fine-tuning can significantly improve the speaker similarity metrics of the cloned speech. Also, the authors of [205] observe that fine-tuning the whole synthesis model is faster and more effective than fine-tuning only the speaker embedding layer since more degrees of freedom are allowed in the whole model adaptation. Our preliminary experiments on model adaptation suggested the same. We hypothesize the reason for this is that fine-tuning the last-few layers of the synthesis model is essential, if not sufficient, to adapt the synthesizer to the speaker-specific speech characteristics. Therefore, we study the following two model adaptation techniques: **Adaptation whole -** Fine-tune all the parameters of the synthesis model on the text and audio pairs of the new speaker. **Adaptation decoder -** Fine-tune only the decoder parameters of the synthesis model. The advantage of only adapting the decoder parameters is that it requires fewer speaker-specific model parameters and a shared encoder can be used across all speakers in a real-world deployment setting. In both of the above adaptation settings, we fine-tune our model for 100 to 200 iterations using Adam optimizer with a learning rate of $1e-4$.

6.1.3 Experiments on Expressive Voice Cloning

We train our mel-spectrogram synthesis model on the clean subset of the publicly available Libri-TTS [213] dataset—*train-clean-100* and *train-clean-360*. This clean subset

contains around 245 hours of speech across 1151 speakers sampled at 24 kHz. We filter out utterances longer than 10 seconds and resample waveforms to 22050 Hz. The speaker embedding layer is replaced with our speaker encoding network which is kept frozen during training. We use a validation set with 250 examples and train the model using a batch size of 32 and an initial learning rate of 5e-4. We use an Adam optimizer [113] to update the weights and anneal the learning rate to half its value every 50k mini-batch iterations. For the *Tacotron 2 + GST* model, we use the same Tacotron 2 architecture and GST hyper-parameters as our proposed model. Training for the proposed model and the *Tacotron 2 + GST* model converged in 210,000 and 185,000 mini-batch iterations respectively and took around 4 seconds per iteration on a single Nvidia Titan 1080 GPU. The Resemblyzer speaker encoder [210, 214] used in our experiments is trained on the VoxCeleb [215], VoxCeleb2 [216] and LibriSpeech-other [217] datasets containing a total of 8.4k speakers. The authors of [214] report a 4.5% Equal Error Rate (EER) for the task of speaker verification using this speaker encoder on their internal test set.

To evaluate our cloning techniques objectively in terms of style and speaker disentanglement, and also assess their usefulness in real world settings, we perform the following cloning tasks:

1. Text *Cloning speech directly from text:* For cloning speech directly from text, we first synthesize speech for the given text using a single speaker TTS model: Tacotron 2 + WaveGlow trained on the LJ Speech [218] dataset. We then derive the pitch contour of the synthetic speech using the Yin algorithm [208] and scale the pitch contour linearly to have the same mean pitch as that of the *target speaker samples*. For deriving rhythm, we use our proposed synthesis model as a forced aligner between the text and Tacotron2-synthesized speech. We use the *target speaker samples* for obtaining the GST embedding for both our proposed model and the baseline Tacotron2 + GST model.

2. Imitation - *Reconstruct a sample from the target speaker:* In this setup, we use a text and audio pair of the target speaker (not contained in the *target speaker samples*), and try to reconstruct the audio from its factorized representation using our synthesis model. All of the

style conditioning variables - pitch, rhythm and GST embedding are derived from the speech sample we are trying to imitate. The imitation task is a toy experiment that allows quantitative evaluation of style similarity metrics between the synthesized speech and style reference.

3. Style Transfer - *Transfer the pitch and rhythm of speech from a different expressive speaker:* The goal of this task is to transfer the pitch and rhythm from some expressive speech to the cloned speech for the target speaker. For this task, we use examples from the single speaker Blizzard 2013 dataset [219] as style references. This dataset contains expressive audio book readings from a single speaker with high variation in emotion and pitch. For our proposed model, we use this *style reference audio* to extract the pitch and rhythm. Similar to the Text task, we scale the pitch contour to have the same mean as that of the *target speaker samples*. In-order to retain speaker-specific latent style aspects, we use *target speaker samples* to extract the GST embedding. For the Tacotron2 + GST model, which does not have explicit pitch conditioning, we use the *style reference audio* for obtaining the GST embedding and the rhythm.

For the above-described cloning tasks, we evaluate three aspects of the cloned speech: i) speaker similarity to the target speaker, ii) style similarity to the reference style and iii) speech naturalness. We encourage the readers to listen to our audio examples referenced in the footnote of the first page to contextualize the following results.

Speaker Classification Accuracy: We train a speaker classifier on the VCTK dataset to classify a given utterance as one of the 108 speakers. The speaker classifier is a two layer neural network with 256 hidden units that takes as input the speaker encoding obtained through our pre-trained speaker encoder network. Similar to [205], our speaker classifier achieves 100% accuracy on a hold out set containing 200 examples from the VCTK dataset. However, since our classification model and training dataset for the synthesizer are not the same as [205] (1,151 speakers in ours vs. 2,481 speakers in [205]), we do not make direct comparisons with their work.

We conduct our speaker classification evaluations on all 108 speakers of the VCTK dataset. We clone 25 speech samples per speaker for each cloning task. Figure 6.2 (left) shows the speaker classification accuracy curves for all cloning tasks and techniques with respect to

the number of target speaker samples. Our results are consistent with the following findings of [205]—Model adaptation significantly outperforms the zero-shot voice cloning technique since it allows the model to adjust to the speaker characteristics of the new speaker. More target speaker samples helps improve speaker classification accuracy, although in the zero-shot scenario we do not observe much improvement after 10 target speaker samples.

For zero-shot voice cloning, both Tacotron2-GST and our proposed model achieve similar speaker classification accuracy for *Text* and *Style Transfer* cloning tasks. The accuracy of our proposed model is slightly higher for the imitation task as compared to other tasks for both model adaptation and zero-shot voice cloning. This implies that conditioning on the actual pitch contour of the target speaker improves speaker specific characteristics of the cloned speech. While linear scaling of a reference style pitch contour works well, our findings motivate future research on predicting speaker-specific pitch contours from text and speaker encodings.

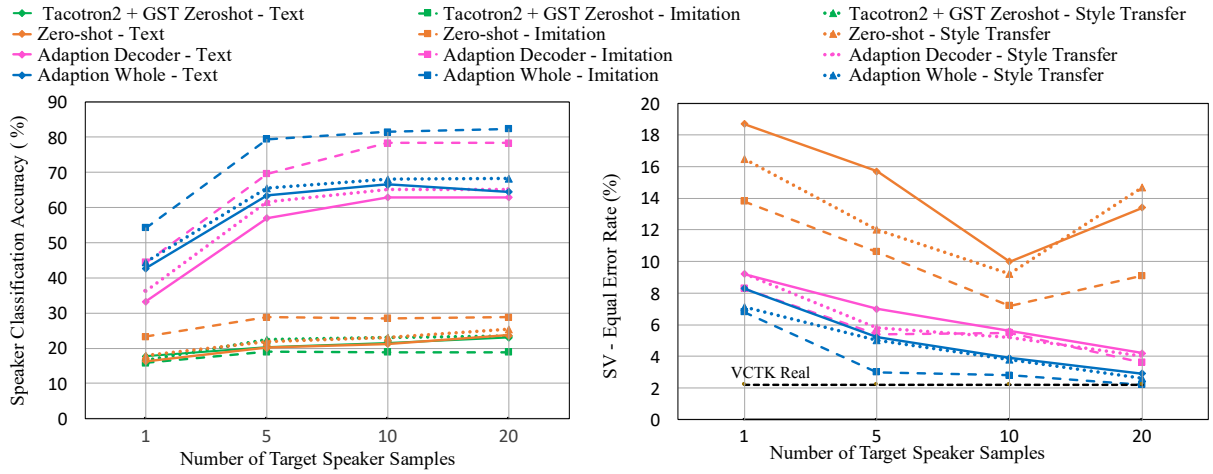


Figure 6.2. Speaker similarity evaluation of each cloning technique for different voice cloning tasks in terms of Speaker Classification Accuracy and Speaker Verification Equal Error Rate (SV-EER).

Speaker verification Equal Error Rate (SV-EER): SV-EER is another objective metric used to evaluate speaker similarity between the cloned audio and the ground-truth reference audio. We use a speaker verification system that scores the speaker similarity between two utterances based on the cosine similarity of the encodings obtained using the speaker encoder described

in Section 6.1.1. Equal Error Rate (EER) is the point when the false acceptance rate and false rejection rate of the speaker verification system are equal.

We perform speaker verification evaluations on randomly selected 20 speakers in the VCTK dataset. We enroll 5 speech samples per speaker in the speaker verification system and synthesize 50 speech samples per speaker for each cloning task. EERs are estimated by pairing each sample of the same speaker with another sample from a different speaker. Figure 6.2 shows the plots of SV-EER for different cloning techniques and tasks using our proposed model and also the those estimated using real data. Our observations on the SV-EER metric are similar to those on the accuracy metric. Model adaptation outperforms zero-shot cloning techniques and with more than 10 cloning samples achieves similar EER as the real data.

Table 6.1. Style similarity evaluations for the imitation and style transfer tasks. We use three objective error metrics (lower values are better). For the style transfer task we present the mean opinion scores on style similarity (Style-MOS) with 95% confidence interval.

	Imitation			*Style Transfer*
Approach	GPE	VDE	FFE	Style-MOS
Tacotron2 + GST - Zero-shot	20.37%	26.39%	29.47%	2.69 ± 0.11
Proposed Model - Zero-shot	3.72%	10.65%	11.74%	3.15 ± 0.11
Proposed Model - Adaptation Whole	2.97%	12.58%	13.60%	3.40 ± 0.10
Proposed Model - Adaptation Decoder	2.39%	11.60%	12.51%	3.29 ± 0.10

Style Similarity: In order to evaluate the similarity between the style of synthesized and reference audio, we perform quantitative evaluation on the Imitation task. We use the following error metrics: Gross Pitch Error (GPE) [220], Voicing Decision Error (VDE) [220] and F0 Frame Error (FFE) [221]. Results are presented in Table 6.1 in which we compare the error values for different approaches when using 10 target speaker samples for cloning. We synthesize 25 speech samples per speaker for all speakers in the VCTK dataset to estimate the reported error values. Our proposed models significantly outperform the Tacotron 2 + GST baseline, clearly indicating the importance of pitch contour conditioning for accurate style transfer.

We also conduct a crowd-sourced listening test on Amazon Mechanical Turk (AMT) for

the style transfer task in which we ask the listeners to rate the style similarity between the ground truth style reference and synthesized audio on a 5 point scale. For each cloning technique (using 10 target speaker samples), we synthesize 25 audio samples per speaker for 20 speakers in the VCTK dataset leading to 500 evaluations of each technique. We present the style similarity Mean Opinion Scores (Style-MOS) in Table 6.1. It can be seen that our proposed models significantly outperform the Tacotron 2 + GST model. Model adaptation techniques perform slightly better than zero-shot method suggesting that fine-tuning improves the model predictions for an unseen speaker encoding.

Naturalness: To assess speech naturalness, we conducted a crowd-sourced listening test on AMT and asked listeners to rate each audio utterance on a 5-point naturalness scale to collect Mean Opinion Scores (MOS). Similar to the above mentioned user study, we use 10 target speaker samples for each cloning technique. All evaluations are conducted on randomly selected 20 VCTK speakers with 25 audio samples synthesized per speaker. Each sample is rated independently by a single listener leading to 500 evaluations for each technique per cloning task. We report the MOS of Real data and audio synthesized using different cloning techniques in Table 6.2. Our proposed model significantly outperforms the baseline Tacotron2 + GST model for both zero-shot and model adaptation techniques. This suggests that pitch contour conditioning in a multi-speaker setting helps improve speech naturalness in addition to providing higher style similarity. It can be seen that the naturalness is even further improved with model adaptation techniques since it allows the generative model to adjust for the unseen speaker encodings.

6.2 Voice Conversion Using Iterative Self-Refinement

In this section, I describe a zero-shot voice conversion framework we developed using speech representations trained with self-supervised learning. The key idea behind our voice conversion system is disentangling speech into features describing the linguistic content and speaker characteristics. Synthesizing speech from these disentangled features allows voice

Table 6.2. Mean Opinion Score (MOS) for speech naturalness with 95% confidence intervals.

	Cloning Task		
Approach	Text	Imitation	Style Transfer
Real data VCTK		4.11 ± 0.08	
Real data Blizzard		4.07 ± 0.08	
Tacotron2 + GST - Zero-shot	2.67 ± 0.10	2.51 ± 0.10	3.02 ± 0.09
Proposed Model - Zero-shot	3.56 ± 0.09	3.54 ± 0.10	3.53 ± 0.10
Proposed Model - Adaptation Whole	3.75 ± 0.09	3.71 ± 0.09	3.60 ± 0.09
Proposed Model - Adaptation Decoder	3.61 ± 0.09	3.57 ± 0.09	3.45 ± 0.09

conversion by swapping the speaker embedding of a given utterance with a target speaker. The goal of this work is to develop *universal language-independent models*, that do not rely on text and, as a result, do not require transcribed speech files for training. This characteristic makes them especially suitable for synthesizing low-resource languages, where there is a scarcity of audio training data and often a lack of parallel text transcripts.

To derive disentangled speech representations in a text-free manner, recent methods [222, 223, 224, 225, 203] have proposed to obtain speaker information from a speaker verification model and linguistic content information from the output of models trained using self-supervised learning (SSL) [226, 227]. While the representations obtained from the SSL models are highly correlated with phonetic information, they also contain speaker information [228, 225, 229]. To remove speaker information from the SSL model outputs, some techniques utilize an information bottleneck approach such as quantization [223, 222, 230]. Alternatively, several researchers have proposed training strategies that employ an information perturbation technique to eliminate speaker information without quantization [231, 203, 232, 228]. Notably, for training synthesizers, NANSY [203] and NANSY++ [232] propose to heuristically perturb the voice of a given utterance with hand-engineered data augmentations, before deriving the content embedding from the SSL model. To reconstruct the original audio accurately, the synthesizer is forced to derive the speaker characteristics from the speaker embedding since the speaker information in the content embedding is perturbed. While such techniques are effective, heuristic voice perturbation

algorithms based on pitch randomization and formant shifting represent a very limited set of transformations. We hypothesize that such training strategies can be improved by utilizing neural network-generated augmentations.

In our work, I propose a learning framework to automatically generate diverse data transformations during training and enable controllable speech synthesis from imperfectly disentangled but uncompressed speech representations. First, we develop a feature extraction methodology that not only derives the content and speaker embeddings but also prosodic information such as speaking rate and pitch modulation. Next, to train a controllable synthesizer, we propose a training strategy that utilizes the synthesis model itself to create challenging voice-converted transformations of a given speech utterance. At any given training iteration, the current state of the synthesis model is used to transform the input content embedding and the model is updated to minimize the reconstruction error of the original utterance.

All the components in our framework are trained in a text-free manner requiring only audio data. Once trained, our framework can be used for tasks such as zero-shot voice conversion, audio reconstruction with pitch and duration modulation as well as multilingual voice conversion across languages outside of the training set. On metrics evaluating speaker similarity, intelligibility and naturalness of synthesized speech we demonstrate that our model outperforms previously proposed zero-shot voice conversion methods for both seen and unseen speakers.

6.2.1 Related Work

Voice conversion is the task of modifying an utterance of a source speaker to match the vocal qualities of a target speaker. Traditionally, voice conversion models were trained as a speech-to-speech translation system on a parallel dataset containing multiple speakers saying the same utterance [233, 234]. More recently, voice conversion systems have been developed by training neural synthesizers to reconstruct speech from disentangled representations describing content and speaker characteristics [202, 201]. For example, [235, 236] have utilized pre-trained automatic speech recognition (ASR) and speaker verification (SV) models to disentangle

content and speaker information respectively. The predicted text or phonetic posteriogram (PPG) obtained from the ASR model is taken as the content representation. However, such voice conversion systems have limitations: 1) Training such systems requires transcribed speech data and the synthesis is limited to the language the model is trained on. 2) Text and PPG do not capture all linguistic features such as accent, expressions, emotions or speaker-independent style resulting in neutral-sounding synthesized speech.

To derive linguistic content in a text-free manner, some prior works have utilized SSL based models. However, as noted by prior work [223, 225], SSL model outputs do not necessarily separate speaker and content information. One line of research [223, 222, 230] aiming to disentangle the speaker and content representations, proposes an information bottleneck approach to quantize SSL model outputs thereby limiting the information to only capture the content or pseudo-text of the audio. However, the loss of information during such a quantization approach leads to sub-optimal reconstruction quality.

Addressing the limitations of information bottleneck approaches, researchers have proposed training strategies based on heuristic transformations. For example, in ContentVec [231] and ACE-VC [228], while training the SSL-based feature extractor model, the audio is transformed using pitch-shift transformation and the SSL model is encouraged to output similar representations for the original and transformed audio. Alternatively, in NANSY [203], the transformations are applied while training the synthesizer, i.e. the synthesizer is tasked to reconstruct the original audio from the content representation of audio perturbed using transforms such as formant-shift, pitch-randomization and randomized frequency shaping. Although these heuristic transformations serve as a reasonable proxy for voice conversion methods, we hypothesize such methods can be greatly improved by utilizing the voice conversion system itself to generate more diverse input transformations.

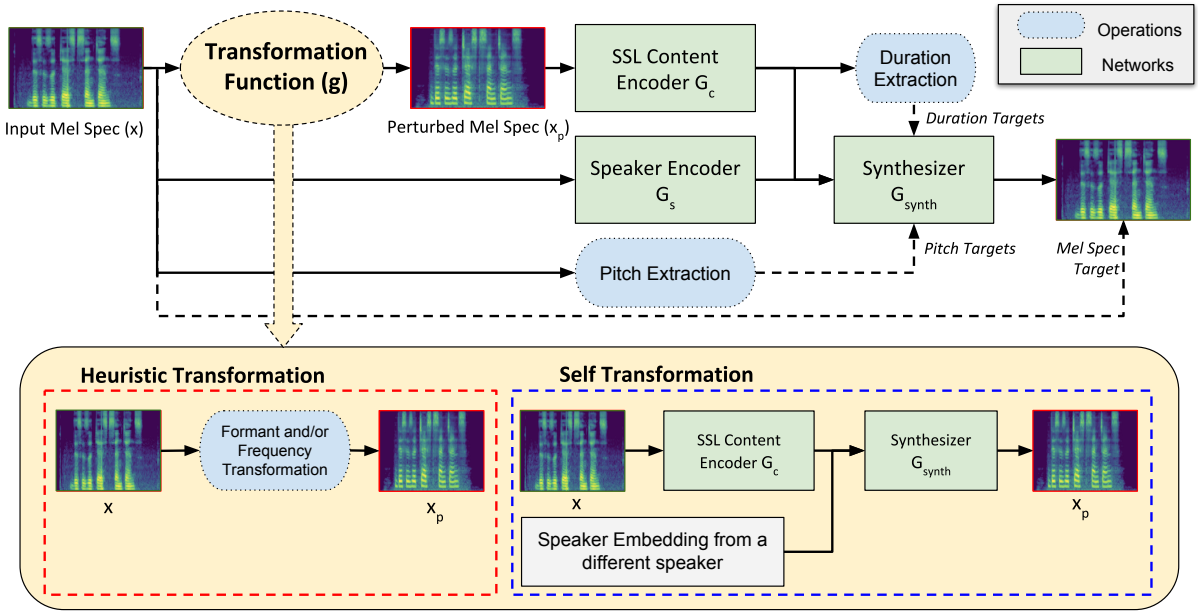


Figure 6.3. Voice Conversion Approach Overview: The synthesis model is trained to reconstruct the mel-spectrogram from SSL-based content representation of a transformed audio (heuristic or self-transformed) and speaker embedding of the original audio.

6.2.2 Voice Conversion Approach

Our framework consists of two main components: 1) A *feature extractor* that derives content (linguistic features), speaker and style representations from a given speech utterance. 2) A *synthesizer* that reconstructs the audio from the derived representations. To allow controllable synthesis from imperfectly disentangled representations, we propose a training strategy that challenges the model to reconstruct the audio from self-generated perturbations of the content representation. Specifically, we train the model to reconstruct the audio from the content representation of a heuristically modified or self transformed audio, while preserving the speaker and style representations. The content and speaker encoder networks remain fixed during synthesis model training.

Feature Extraction

Content Embedding: We define content as a temporal feature that encodes the linguistic information of a given speech utterance. We use the output of the Conformer-SSL [227] model (G_c) as

the content representation of speech (z). The Conformer-SSL model is a convolution-augmented transformer architecture that is trained to reconstruct the masked areas of the mel-spectrogram on 56k hours of English speech data, using contrastive and masked language modelling (MLM) losses. The representations derived from SSL-based speech encoder models have been shown to have a high correlation with corresponding phonetic information [226]. Given a speech utterance as a sequence of mel-spectrogram frames $x = x_1 \dots x_T$, the Conformer-SSL model outputs a temporally downsampled sequence of feature vectors $z = G_c(x) = z_1 \dots z_{T'}$. In our setup, the SSL model temporally downsamples the mel-spectrogram by a factor of 4 and the output at each time-step z_t is a 256 dimensional vector corresponding to a contextualized representation of roughly 46 milliseconds of audio.

Speaking Rate or Duration: Speaking rate determines how long the speaker vocalizes each phoneme of a given utterance. Since the speaking rate can vary greatly across different speakers and accents, accurate modelling of speaking rate during synthesis is important to closely mimic a target speaker. We propose a technique to derive the speaking rate or duration information in a text-free manner from the content representation. Since SSL representations have a high correlation with phonemes [226, 227], we conjecture that if a phoneme is emphasized in an utterance, the consecutive content vectors at the corresponding timesteps will have high similarity. Therefore, we propose Algorithm 5 to process the content representation $z = z_1 \dots z_{T'}$ into a duration-augmented content representation $z' = z'_1 \dots z'_{T'}$ and $d' = d'_1 \dots d'_{T'}$. We group together consecutive content vectors with cosine similarity higher than a threshold τ, and set the target duration for the averaged vector as the number of grouped vectors times the duration of a single vector.

Speaker Embedding: The speaker embeddings in our setup are derived from the TitaNet [237] model (G_s). TitaNet is based on a 1-D depthwise separable convolution architecture with Squeeze and Excitation layers that provide global context. The TitaNet speaker verification model is trained using additive angular margin loss [238] on 3373 hours of speech from multiple datasets that span 16681 speakers. The model is designed to be parameter-efficient and achieves state-of-

Algorithm 5. Deriving duration-augmented content by grouping similar consecutive vectors

$z' \leftarrow [z_1]$ ▷ Initialize z' with the vector from the first time-step in z
$d' \leftarrow [\delta]$ ▷ d'_t represents duration of z'_t. δ represents duration of of each z_t (i.e 46 ms)
$num\_grouped \leftarrow 1$ ▷ number of similar vectors grouped at the last processed time-step
for $t \leftarrow 2$ **to** T' **do**
  if $CosineSimilarity(z_t, z'[-1]) > \tau$ **then** ▷ Group z_t with the running group
    $z'[-1] \leftarrow (z_t + num\_grouped * z'[-1])/(num\_grouped + 1)$ ▷ Update average
    $d'[-1] \leftarrow \delta * (num\_grouped + 1)$
    $num\_grouped \leftarrow num\_grouped + 1$
  else ▷ Insert z_t in a new group
    $z'.append(z_t)$
    $d'.append(\delta)$
    $num\_grouped \leftarrow 1$
return z', d'

the-art results on the VoxCeleb-1 speaker verification benchmark with an EER of 0.68%. The output from the speaker verification model is a 256 dimensional speaker embedding $s = G_s(x)$.

Pitch Contour: The pitch contour p is derived from the fundamental frequency f_0 contour of the speech signal that represents the prosodic modulations over time. The raw values in the fundamental frequency contour (derived from PYin algorithm [208]) are speaker-dependent, therefore f_0 is not strictly disentangled from the speaker information. To ensure that the pitch contour only encodes the prosodic changes and not the speaker identity, we normalize f_0 using the mean (f_{mean}) and standard deviation (f_{std}) of all pitch contours of the given speaker. That is, $p = (f_0 - f_{mean})/f_{std}$.

Synthesizer

The task of the synthesizer is to first reconstruct the ground-truth mel-spectrogram from the extracted speech representations and then vocode the mel-spectrogram into a listenable audio waveform. For vocoding, we use a HiFiGAN [239] vocoder, which is trained separately on spectrogram and waveform pairs of real audio from a multi-speaker dataset.

Our mel-spectrogram synthesizer G_{synth} is composed of two feed-forward transformers F_e and F_d and intermediate modules to predict the duration and pitch similar to [240] but operates

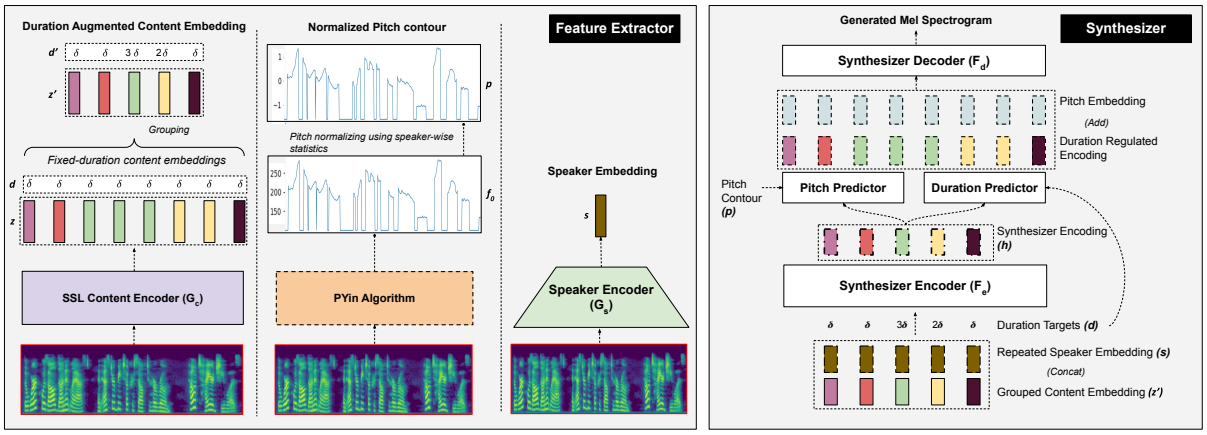


Figure 6.4. (a) The feature extractor derives the duration augmented content information from an SSL model, pitch information using PYin algorithm and speaker embedding from a speaker verification model. (b) The synthesizer reconstructs the mel-spectrogram from the derived features.

on the grouped content representation $z' = z'_1 \dots z'_{T'}$ instead of text. The speaker embedding s is repeated across all time-steps and concatenated with each z'_t to be fed as input to the first feed-forward transformer F_e. The hidden representation from F_e is then used to predict the duration and pitch, that is: $h = F_e(z', s)$; $\hat{y_d} = DurationPredictor(h)$, $\hat{y_p} = PitchPredictor(h)$. The pitch contour is projected and averaged over each time-step of the hidden representation h and added to h to get $k = h + PitchEmbedding(p)$. Finally, k is discretely upsampled as per the ground-truth duration d' and fed as input to the second transformer F_d to get the predicted mel-spectrogram $\hat{y} = F_d(DurationRegulation(k, d'))$

Our model is trained to optimize three losses — mel-reconstruction error, pitch prediction error and duration prediction error such that

$$L_{synth} = \|\hat{y} - y\|_2^2 + \lambda_1 \|\hat{y_p} - p\|_2^2 + \lambda_2 \|\hat{y_d} - d'\|_2^2 \tag{6.4}$$

During inference, we can use either the predicted pitch and duration, in which case the prosody is derived from both the content and speaker embeddings; or we can mimic the prosody and speaking rate of the source utterance by using ground-truth duration and pitch information.

6.2.3 Synthesizer Training: Iterative Refinement using Self Transforms

While the mel-spectrogram can be accurately reconstructed from a synthesizer trained using the objective given by Equation 6.4, during inference, we cannot effectively modify the voice of a given utterance. This is because the content representation z' is not strictly disentangled from the speaker information. To address this challenge, past works [203, 232], have proposed an information perturbation based training strategy as follows: Instead of feeding the content embedding of the original audio as the input, the audio is perturbed to synthetically modify the speaker characteristics using formant-shifting, pitch-randomization and randomized frequency shaping transforms to obtain $x_p = g_{heuristic}(x)$. Next, the content embedding is derived from the perturbed audio $z' = G_c(x_p)$, while the speaker embedding is still derived from the original audio $s = G_s(x)$. The network is then tasked to reconstruct the original audio from z' and s. While heuristically perturbed content representations play a crucial role in enhancing the synthesizer model's attention towards the speaker embedding, they are limited in terms of the range of transformations they can introduce. Heuristic transformations represent only a subset of the potential natural variations that can occur during voice conversion.

To expand on the heuristic set of transforms, we propose to utilize the synthesizer model itself to generate a voice-converted variation of a given utterance x. That is, given a synthesizer model G^i_{synth} trained until training iteration i, we obtain a self transformed audio for iteration $i+1$ as:

$$x_p = g_{self}(x) = G^i_{synth}((G_c(x), s') \tag{6.5}$$

where $G_c(x)$ is the content embedding of the original audio x and s' is the speaker embedding obtained from an utterance x' of a different randomly selected speaker, that is, $s' = G_s(x')$. The content embedding input for the training step $i+1$ is then derived as $z' = G_c(x_p)$.

Self transformations not only provide a more diverse set of transformations but also

present an increasingly challenging reconstruction task for the synthesizer, as its voice conversion capabilities improve with each training iteration. Figure 6.3 demonstrates the proposed self transformation training strategy. In our experiments, we begin self transformations after $100k$ mini-batch iterations of training with heuristically modified audio. Thereafter, we continue to use self transformations to obtain x_p.

6.2.4 Experiments on Voice Conversion

Dataset and Training

The Conformer-SSL model used as the content encoder is pretrained on $56k$ hours of unlabelled English speech from the LibriLight [241] corpus sampled at 16 KHz. We finetune the Conformer-SSL model (using self-supervision with contrastive and MLM loss) on the *train-clean-360* subset of LibriTTS [213] dataset with audio sampled at $22050Hz$ to make the model compatible with the mel-spectrogram representation of the synthesizer. For both the content encoder and synthesizer, we use 80 bands for mel spectrogram with the FFT, window, and hop size set to 1024, 1024, and 256 respectively. We finetune the Conformer-SSL on this revised spectrogram representation for 50 epochs with a batch size of 32 using the AdamW optimizer with a fixed learning rate of $5e-5$ and $\beta_1 = 0.9, \beta_2 = 0.99$. Finetuning takes around 50 hours on a single NVIDIA A600 GPU.

For our primary experiments, the mel-spectrogram synthesizer and the HifiGAN vocoder are also trained on the *train-clean-360* subset of the LibriTTS dataset which contains 360 hours of speech from 904 speakers. We train three variants of the mel-spectrogram synthesizer:

1. **Synth (NoTransform)** is trained to simply reconstruct the mel-spectrogram from the embeddings of the given utterance without any information perturbation procedure.

2. **Synth (Heuristic)** is trained to reconstruct the mel-spectrogram from the content embedding of the heuristically perturbed utterance and the speaker embedding of the original utterance. We employ two transforms g_1, g_2 proposed in [203]. g_1 perturbs formant, pitch,

and frequency response and g_2 perturbs formant and frequency response while preserving pitch.

3. **Synth (SelfTransform)** is first trained in the same way as Synth-Heuristic for the first $100k$ mini batch iterations. Thereafter, we use the g_{self} transformation procedure given by Equation 6.5.

All three variants of the synthesizer are optimized using AdamW optimizer [242] with a fixed learning rate of $1e-4$ and $\beta_1 = 0.8, \beta_2 = 0.99$ for 500 epochs with a batch size of 32. The threshold τ for duration extraction is set as 0.925. The loss coefficients for the duration and pitch loss are set as $\lambda_1 = \lambda_2 = 0.1$. Training time for Synth (SelfTransform) model is around 5 days on 4 NVIDIA A600 GPUs. The HifiGAN vocoder is trained on the mel-spectrogram and waveform pairs of the real audio utterances and the same vocoder is used across all three synthesizers.

Evaluation Metrics

Quantitatively, we evaluate the synthesized audio on the following aspects:

1. **Intelligibility (CER):** We compute the Character Error Rate (CER) between the ASR transcriptions of the original source and the generated audio. We use the pre-trained Quartznet [243] ASR models for the respective language of the given utterance.

2. **Speaker Similarity (SV-EER):** To evaluate speaker similarity to our target speaker, we compute the speaker embeddings of synthesized and real utterances using a separate pre-trained speaker verification model [244]. Then we pair the synthesized and real utterances to create an equal number of positive and negative pairs for each target speaker to compute the Equal Error Rate (SV-EER).

3. **Naturalness (MOS):** We perform a human study on Amazon Mechanical Turk, where human judges rate the naturalness of each utterance on a 1 to 5 scale with 0.5 point increments. Each utterance is rated by 4 independent listeners and each listener can rate

multiple utterances. For 200 synthesized utterances from each technique, this procedure results in a total of 800 evaluations of each technique.

4. **Prosodic Similarity (GPE):** To evaluate prosodic similarity for the reconstruction task (Section 6.2.4), we compute the error between the fundamental frequency contours of the original and synthesized audio. Specifically, we use the Gross Pitch Error (GPE) [221] to evaluate prosodic similarity.

Reconstruction Results

First, we evaluate how effectively our setup can reconstruct audio from the extracted representations for unseen utterances and speakers. Our synthesizers can operate in two modes during inference — *1) Guided:* In this scenario, we use ground truth pitch and duration information derived from the source utterance. *2) Predictive:* In this case, we use the predicted pitch and duration for synthesis.

We conduct the reconstruction test on two unseen datasets — 1) We choose 200 utterances from the VCTK [245] dataset (English) with 20 random utterances from each of the 10 speakers (5 random male and 5 random female speakers); 2) To evaluate performance on unseen languages, we choose 200 utterances from the CSS10 [246] dataset with 20 random utterances from each of the 10 unseen languages. The CSS10 dataset has a single speaker per language. For both of these evaluations, we use the synthesizer models trained on the same dataset, i.e. *train-clean-360* subset of LibriTTS (English). The synthesized speech is evaluated on the intelligibility, speaker similarity and prosodic similarity metrics. As indicated by the results in Table 6.3, all three synthesizers can effectively reconstruct the speech signal from the derived representation. Since the model is trained in a text-free manner, we also see a promising generalization to unseen languages. For unseen languages, our synthesizers produce more intelligible speech in the guided mode, where the duration information of the source utterance is kept intact. In the reconstruction mode, since the speaker and content embeddings are derived from the same utterance, both Synth (NoTransform) and Synth (Heuristic) models achieve competitive speaker similarity to the target

speaker. However, for controllable synthesis tasks such as voice conversion, we demonstrate that Synth (SelfTransforms) outperforms these models.

Table 6.3. Reconstruction evaluation: The resynthesized speech from different synthesizers is evaluated for intelligibility (CER), speaker similarity (SV-EER) and prosodic similarity (GPE). Lower values are desirable for all three metrics.

		Guided			*Predictive*		
Dataset	Technique	SV-EER	CER	GPE	SV-EER	CER	GPE
	Real Data	3.1%	-	-	3.1%	-	-
VCTK	Synth (NoTransform)	4.6%	3.5%	8.0%	4.7%	4.9%	22.0%
(English)	Synth (Heuristic)	4.3%	2.9%	8.8%	4.5%	4.1%	21.1%
Seen Language	Synth (SelfTransform)	4.2%	2.2%	8.9%	4.1%	3.9%	21.0%
	Real Data	2.3%	-	-	2.3%	-	-
CSS10	Synth (NoTransform)	5.5%	25.5%	11.7%	4.9%	29.8%	15.9%
(Multilingual)	Synth (Heuristic)	5.3%	26.1%	11.6%	5.5%	30.2%	16.1%
Unseen Language	Synth (SelfTransform)	4.1%	25.2%	10.8%	4.8%	29.2%	16.8%

Voice Conversion Results

To convert the voice of a given source utterance to a target speaker, we derive the content embedding from the source utterance and estimate the speaker embedding from the target speaker's audio and feed both as input to the synthesizer. We consider two voice conversion scenarios — for a seen speaker to another seen speaker from the training data (Many-to-Many) and from an unseen speaker to another unseen speaker outside of training data (Any-to-Any). For seen speakers, we use the holdout utterances of the *train-clean-360* subset of LibriTTS dataset, and for unseen speakers, we use the VCTK dataset. For each scenario, we randomly select 20 target speakers (10 male and 10 female). Next, we select 10 source utterances, each one from 10 alternate speakers. This results in a total of 200 voice conversion trials in each scenario.

For our primary evaluation, we use 10 seconds of speech from each target speaker to derive the speaker embedding. We split the 10 second target-speaker utterance into 2 second segments and estimate the speaker embedding as the mean speaker embedding across the segments. We also evaluate the speaker-similarity performance for different amounts of target

Table 6.4. Comparison of different voice-conversion techniques. Lower values for SV-EER and CER are desirable for higher speaker similarity and intelligibility respectively. Higher MOS (reported with 95% confidence interval) indicates more natural-sounding speech.

	Many-to-Many			*Any-to-Any*		
Technique	SV-EER	CER	MOS	SV-EER	CER	MOS
Real Data	2.9%	-	4.03 ± 0.09	3.1%	-	4.08 ± 0.09
AutoVC [202]	23.5%	21.2%	2.75 ± 0.11	38.3%	34.2%	2.46 ± 0.11
AdaIN-VC [201]	18.2%	29.2%	2.64 ± 0.11	27.5%	30.3%	2.82 ± 0.12
MediumVC [230]	10.2%	31.5%	3.01 ± 0.12	23.2%	36.2%	2.95 ± 0.11
FragmentVC [224]	15.9%	27.2%	3.10 ± 0.11	24.8%	38.5%	3.11 ± 0.12
S3PRL-VC [225]	13.7%	9.8%	3.20 ± 0.11	22.8%	9.8%	3.14 ± 0.12
YourTTS [204]	9.5%	6.1%	3.52 ± 0.10	12.3%	7.9%	3.58 ± 0.10
ACE-VC [228]	5.3%	3.7%	3.58 ± 0.10	9.2%	8.2%	3.68 ± 0.09
Synth (NoTransform)	19.1%	2.6%	3.55 ± 0.12	25.2%	3.8%	3.51 ± 0.11
Synth (Heuristic)	4.4%	2.3%	3.69 ± 0.12	10.5%	3.1%	3.65 ± 0.12
Synth (SelfTransform)	**3.0%**	**2.2%**	$\mathbf{3.72 \pm 0.11}$	**4.3%**	**3.1%**	$\mathbf{3.75 \pm 0.11}$

speaker data and present the results in Figure 6.5.

The synthesized speech is evaluated on three aspects: speaker similarity, intelligibility and naturalness. We compare our synthesis model against several prior voice conversion methods listed in Table 6.4. While NANSY [232] is not open-sourced, our Synth (Heuristic) baseline model closely follows the training strategy proposed in NANSY, incorporating more recent neural architectures for the synthesizer and feature extractors. As shown by the results, the Synth (SelfTransform) model outperforms the Synth (NoTransform) and Synth (Heuristic) models on the speaker similarity metrics. The improvement is even more notable for *Any-to-Any* voice conversion task. On all three metrics, our proposed technique outperforms previously proposed voice conversion models. In Figure 6.5, we show TSNE plots of the speaker embeddings of generated and real audio.

Cross-lingual Voice Conversion: In this setup, we consider three scenarios — 1) **S2U:** Source utterance from a seen language speaker (English VCTK) and target speaker from an unseen language (CSS10). 2) **U2S:** Source utterance from an unseen language (CSS10) and

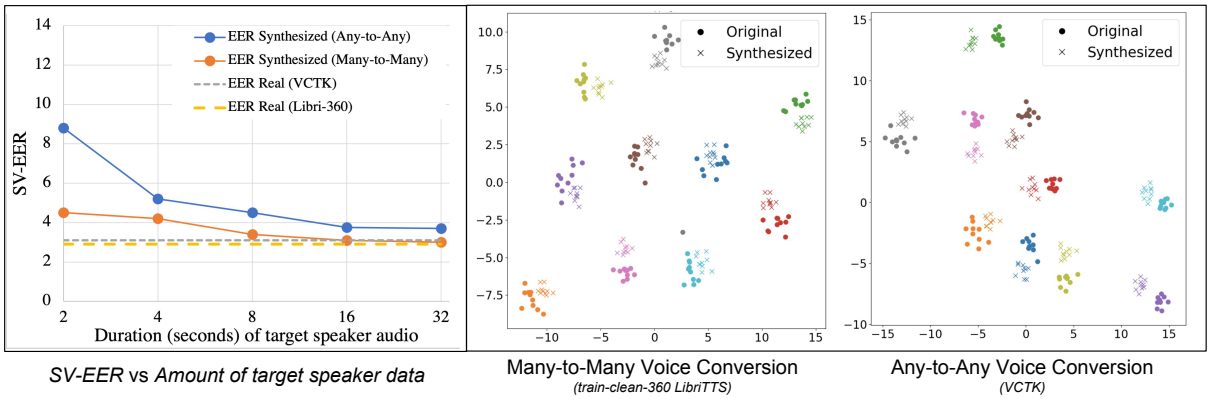


Figure 6.5. Left: SV-EER of voice-converted speech generated by Synth (SelfTransform) using different amounts of target speaker data. **Right:** TSNE visualization of speaker embeddings of generated (using Synth (SelfTransform)) and ground-truth audio. Each color represents a different speaker.

target speaker from the source language (English VCTK). 3) **U2U:** Source utterance from an unseen language (CSS10) and target speaker from another unseen language (CSS10).

We present the results in Table 6.5. While the Synth (SelfTransform) model generates speech with high speaker-similarity in all three scenarios, the generated speech is more intelligible when the source utterance is in English. This is not surprising since the synthesizer is trained on only English speech (LibriTTS).

Table 6.5. Results on cross-lingual voice conversion task in three scenarios considering different languages for source utterance and target speaker. Lower SV-EER is desirable for higher speaker similarity and lower CER is desirable for more intelligible speech.

	S2U		U2S		U2U	
Technique	SV-EER	CER	SV-EER	CER	SV-EER	CER
Real Data	2.3%	-	3.1%	-	3.1%	-
Synth (NoTransform)	31.2%	3.8%	28.2%	29.7%	39.1%	30.7%
Synth (Heuristic)	15.3%	3.1%	9.0%	28.5%	22.1%	29.5%
Synth (SelfTransform)	8.5%	3.0%	5.4%	27.5%	15.1%	29.1%

6.3 Conclusion

In this chapter, I described two speech synthesis frameworks that tackle the problem of expressive speech synthesis for new speakers. The first framework is a voice-cloning method that performs text-to-speech synthesis with explicit control over speaker and style aspects. By utilizing both latent and heuristically derived style information, the model is able to learn a wide-range style control for unseen speakers while being trained on a mostly style-neutral dataset. The second framework is a voice conversion model that proposes a training strategy to perform controllable speech synthesis from imperfectly disentangled speech representations. The synthesis model of the voice conversion framework allows speaker-adaptive duration and pitch control for more natural-sounding speech achieving state-of-the-art results on various voice conversion metrics. Both of the above frameworks enable high-quality speech synthesis that can accompany AI-generated visuals for various generative media applications.

6.4 Acknowledgements

Chapter 6 contains material found in the following two papers. (1) *Expressive Neural Voice Cloning*. 2021. Neekhara, Paarth; Hussain, Shehzeen; Dubnov, Shlomo; Koushanfar, Farinaz; McAuley, Julian. Asian Conference on Machine Learning 2021. (2) *Controllable Speech Synthesis with Iterative Refinement using Self Transformations*. 2023. Neekhara, Paarth; Hussain, Shehzeen; Ranjan, Rishabh; Dubnov, Shlomo; Koushanfar, Farinaz; McAuley, Julian. Currently under review for publication. The dissertation author and Paarth Neekhara made equal contributions to this work.

Part III

Robust and Efficient Media Authentication and Recognition

Chapter 7

Deepfake Detection and Their Vulnerability to Adversarial Attacks

Media synthesis and editing techniques were traditionally based on computer graphics and signal processing algorithms that required manual editing to generate high-quality audio-visual content. Deep Learning techniques have transformed the field of multimedia synthesis in two major ways: 1) Deep neural networks (DNNs) have improved the quality of the generated content producing more realistic video and speech. 2) DNN based synthesizers are often trained in an end-to-end manner, minimizing the need for domain-specific hand-engineered workflows and resulting in streamlined media synthesis pipelines.

Deepfakes are a new genre of synthetic videos, in which a subject's face is modified into a target face in order to simulate the target subject in a certain context and create convincingly realistic footage of events that never occurred. Neural Face swapping techniques [247, 248, 249] can easily swap the face in a source videos with a target face to create fake visuals. When combined with realistic voice conversion techniques described in Chapter 6, these methods can generate convincing fake videos.

The intent of generating such videos can be harmless as they can be used for advertisements, movies and entertainment purposes. However, such synthesis techniques can also be used maliciously to spread misinformation, harass individuals or defame famous personalities [250]. These videos are now an emerging threat, especially within the realms of politics and misin-

formation. Deepfakes have been used to create fake news aggravating political and religious tensions, with the aim to influence results in election campaigns [251, 252, 253]. Such extensive spread of fake videos through social media platforms has raised significant concerns worldwide, particularly hampering the credibility of digital media. Recent research has found evidence that widespread misinformation not only misleads individuals and reduces public trust on digital media but also leads to increased cynicism within democratic societies [254].

To guard against the misuse of Deepfakes, several countermeasures have been proposed to identify media forgery [255]. In this chapter, I first describe the recently proposed state-of-the-art methods to detect Deepfakes. These methods typically use a visual DNN-based classification system that is trained in a supervised manner on a curated dataset of real and fake videos. Deepfake detection is typically modelled as a per-frame classification problem. Additionally, the best-performing models employ a face-tracking method following which the cropped face from a frame is passed on to a Convolutional Neural Network (CNN) classifier for classification as real or fake [256, 257, 5, 258]. Some of the recent Deepfake detection methods also use models that operate on a sequence of frames as opposed to a single frame to exploit temporal dependencies in videos [259].

While such CNN based Deepfake detectors achieve promising results in accurately detecting manipulated videos, my research exposes major vulnerabilities in such detectors. Later in the chapter, I describe my work on *Adversarial Deepfakes* that examines the vulnerabilities of Deepfake detection systems adversarial examples. An adversarial example is an intentionally perturbed input that can fool a victim classification model [8]. We quantitatively assess the vulnerability of Deepfake detectors to adversarial examples in different threat scenarios. Assuming a complete access (white-box) threat scenario, we find that it is trivial to bypass a Deepfake detector with an imperceptible adversarial modification to a given video. However, in a practical threat scenario, the attacker may not have knowledge of the victim detection model and parameters. To this end, we assume a more challenging threat scenario in which the attacker can only query a victim model to get the detection scores for a video frame. Even in this attack scenario, we find

that it is possible to bypass the detector with a slightly higher amount of adversarial perturbation. Additionally, to ensure the adversarial videos remain effective even after video compression, we incorporate expectation over input transforms [151] while training the adversarial perturbation to craft robust adversarial videos. While the above attacks can effectively bypass Deepfake detectors, they can be easily thwarted by the service provider. Detection models and parameters can be kept private to prevent the white-box attack and query access can be limited to prevent the black-box attack. Adversarial examples pose a practical threat to Deepfake detection if they are transferable across different detection methods. That is, if adversarial videos designed to fool some open source Deepfake detection method can also reliably fool other unseen CNN-based detection methods, this would pose a real security threat to deploying CNN-based detectors in production. We experimentally demonstrate that it is possible to design highly transferable adversarial examples by ensuring robustness to input-transformation functions while training the perturbation. Finally, we design more accessible adversarial attacks by creating transferable universal adversarial perturbations that can be universally added across all frames of all videos to reliably fool a number of Deepfake detection methods.

7.1 Deepfake Detection Datasets

Deepfake detection methods rely on the availability of high-quality deepfake detection datasets, which are crucial for training and evaluating deepfake detection models. Several Deepfake detection datasets have been developed in recent years, each with its own unique characteristics and properties. One of the most widely used Deepfake detection datasets is the FaceForensics++ dataset. This dataset contains Deepfake and real videos captured using a variety of cameras and settings. The dataset includes four types of Deepfakes:

- **FaceSwap (FS):** FaceSwap [249] is a classical computer graphics-based approach for face replacement in videos. In this method, sparse facial landmarks are detected to extract the face region in an image. These landmarks are then used to fit a 3D template model which

is back-projected onto the target image by minimizing the distance between the projected shape and localized landmarks. Finally, the rendered model is blended with the image and color correction is applied.

- **Face2Face (F2F):** Face2Face [247] is a facial reenactment system that transfers the expressions of a person in a source video to another person in a target video, while maintaining the identity of the target person. In this method, faces are compressed into a low-dimensional expression space, where expressions can be easily transferred from the source to the target.

- **DeepFakes (DF):** While the term 'Deepfake' has commonly been used in mainstream media as a blanket term for deep-learning based face replacement, it is also the name of a specific manipulation [260] method that was spread via online forums. In the learning phase, two auto-encoders with a shared encoder are trained to reconstruct the images of source and target face. To create a fake image, the encoded source image is passed as input to the target image decoder.

- **NeuralTextures (NT):** NeuralTextures [248] is a Generative Adversarial Network (GAN) based facial reenactment technique. In this method, a generative model is trained to learn the neural texture of a target person using original video data. The GAN objective is a combination of an adversarial and photometric reconstruction loss.

FaceForensics++ has been widely used in research, with several deepfake detection models trained on this dataset. Aside from the FaceForensics++ dataset, another prominent collection of Deepfake videos was released by Facebook, Inc in 2019. To the best of our knowledge, this recently developed DeepFake Detection Challenge (DFDC) dataset [261] is the largest collection of real and Deepfake videos, consisting of over one million training clips of face swaps produced with a variety of methods. For synthesizing the fake videos in the DFDC dataset, 8 different video manipulation techniques were used, many of which are CNN-based

techniques. These methods include the traditional Deepfake auto-encoder architecture, a non-learned morphable mask face swap algorithm, and several Generative Adversarial Networks (GAN) techniques like Neural Talking Heads [262], FSGAN [263] and StyleGAN [264]. In conjunction with the dataset, a corresponding competition[1] was launched in which competitors were encouraged to submit models trained for Deepfake detection on the training set. These models were then ranked on a hidden, held-out test set, and the winning competitors released their architectures and training strategies publicly.

7.2 Deepfake Detectors

Traditionally, multimedia forensics investigated the authenticity of images [265, 266, 267] using hand-engineered features and/or a-priori knowledge of the statistical and physical properties of natural photographs. However, video synthesis methods can be trained to bypass hand-engineered detectors by modifying their training objective. We direct readers to [268, 269] for an overview of counter-forensic attacks to bypass traditional (non-deep learning based) methods of detecting forgeries in multimedia content.

More recent works have employed CNN-based approaches that decompose videos into frames to automatically extract salient and discriminative visual features pertinent to Deepfakes. Some efforts have focused on segmenting the entire input image to detect facial tampering resulting from face swapping [270], face morphing [271] and splicing attacks [272, 273]. Other works [274, 275, 256, 276, 6, 277] have focused on detecting face manipulation artifacts resulting from Deepfake generation methods. The authors of [275] reported that eye blinking is not well reproduced in fake videos, and therefore proposed a temporal approach using a CNN + Recurrent Neural Network (RNN) based model to detect a lack of eye blinking when exposing Deepfakes. Similarly, [278] used the inconsistency in head pose to detect fake videos. However, this form of detection can be circumvented by purposely incorporating images with closed eyes and a variety of head poses in training [279, 280].

[1] https://www.kaggle.com/c/deepfake-detection-challenge

7.2.1 Per-frame Deepfake Detectors

The Deepfake detectors proposed in [6, 256, 261] model Deepfake detection as a per-frame binary classification problem. The authors of [6] demonstrated that XceptionNet can outperform several alternative classifiers in detecting forgeries in both uncompressed and compressed videos, and identifying forged regions in them. Since the task is to specifically detect facial manipulation, these models incorporate domain knowledge by using a face tracking method [247] to track the face in the video. The face is then cropped from the original frame and fed as input to a classification model to be labelled as real or fake. Experimentally, the authors of [6] demonstrate that incorporation of domain knowledge helps improve classification accuracy as opposed to using the entire image as input to the classifier. The best performing classifiers amongst others studied by [6] were both CNN based models: XceptionNet [257] and MesoNet [256]. Figure 7.1 demonstrates the detection pipeline of these per-frame Deepfake classifiers.

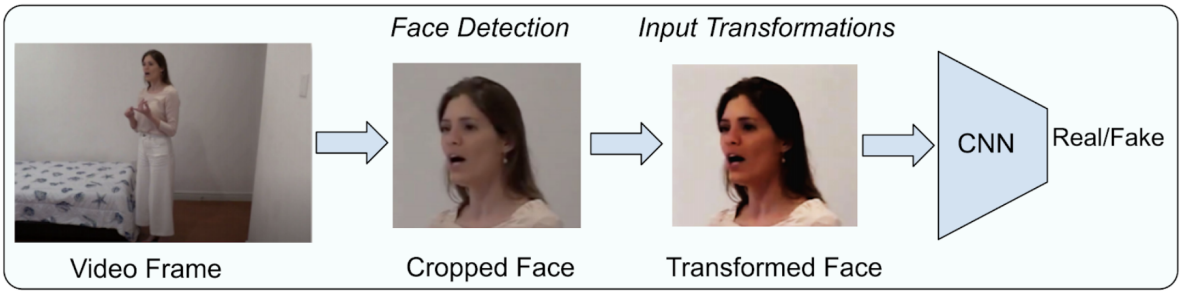


Figure 7.1. Per-frame Deepfake Classification Models typically follow a two-step pipeline: Face detection followed by binary classification.

7.2.2 Sequence-based Deepfake Classifiers

Some detectors have also focused on exploiting temporal dependencies for detecting Deepfake videos. Such detectors work on sequence of frames as opposed to a single frame using a CNN + RNN model or a 3D CNN model. One such model based on a 3D EfficientNet [127] architecture, was used by the third place winner [259] of DFDC challenge [261] in addition

to a per-frame classification model. While intuitively, exploiting temporal dependencies using sequence models should improve a detector's ability to detect manipulated videos, the insights from the results of the DFDC challenge [261] show that the best performing models operate on a frame level. In fact, the winning team [5] of the DFDC challenge explicitly noted that other ideas besides frame-by-frame detection did not improve their performance on the public leaderboard. The first two winning submissions were both CNN based per-frame classification models similar to the ones described above.

7.2.3 Understanding Deepfake detectors

To gain insight into the decision-making logic of Deepfake detectors, we obtain the gradient of the score of the predicted class with respect to the input image and plot the magnitude of these gradients as a heat-map. Back-propagating gradients naively does not result in very interpretable visualizations. This is because it is more important to consider pixels which activate a neuron and do not suppress it (suppression is indicated by negative gradients). Therefore, we use guided back-propagation which defines custom gradient estimates for activation functions like ReLU and suppresses negative gradients during the backward pass. We then standardize the gradient obtained with respect to the input and overlay the heat-map on the frame to visualize the areas of an image that trigger the network's output. Figure 7.2 shows some examples of the saliency maps obtained while analyzing two different detectors on Deepfake videos.

Our initial observations on these saliency maps suggest that different CNN based detection methods attend to similar aspects of the input frame for predicting the label. These aspects include the edges of the face, the eyes, lips, teeth etc. These similarities across different detection methods indicate that adversarially modifying such aspects of the image could potentially fool multiple detection methods. We validate this hypothesis in our work by studying the transferability of adversarial examples (Section 7.5.2) across different detection methods and proposing techniques (Section 7.3.3) that improve the transferability.

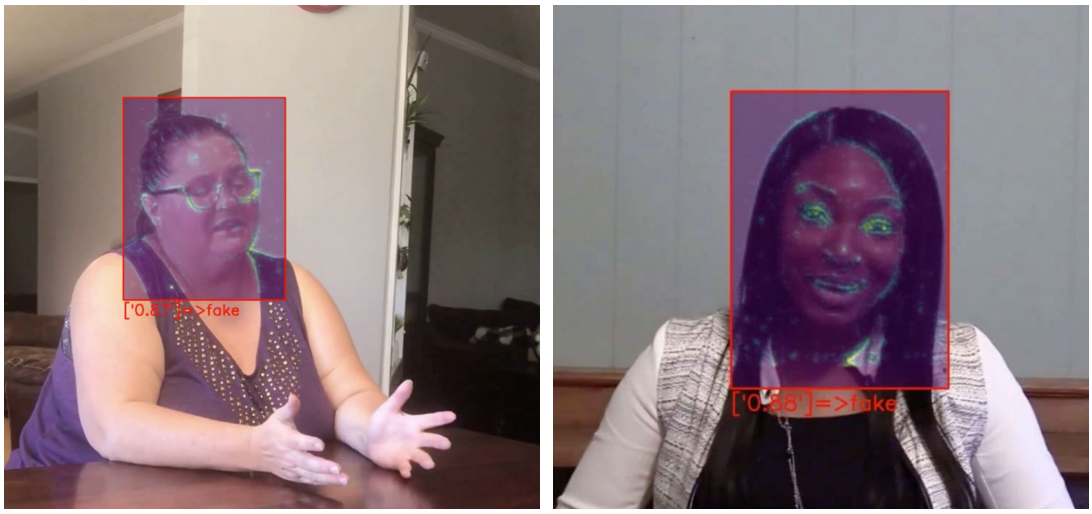


Figure 7.2. Gradient saliency maps obtained on Deepfake videos using guided backpropogation on a CNN-based detector [5]. The highlighted areas indicate the image regions that strongly influence the detector's predictions.

7.3 Adversarial attacks on Deepfake detectors

In this section, I discuss the threat models for Deepfake detectors in various attack settings assuming different attacker capabilities. First, we mathematically define the threat model and attack goal (Section 7.3.1). Next, we propose a white-box attack to achieve the attack goal in a scenario when the attacker has complete access to the victim model architecture and parameters (Section 7.3.2). In our experiments, we find that while the simple white-box attack works well on uncompressed videos, the attack success rate drops significantly on compressed videos. Another challenge in the simple white-box attack is the limited transferability of the attack to unseen models. We tackle these two challenges using our robust and transferable attack which poses a real world threat — the adversarial videos are more robust to compression and can also fool unseen detectors to a significant extent thereby posing a real-world threat (Section 7.3.3). Next we propose query based black-box attacks which do not require access to any surrogate model but only require query access to the model scores (Section 7.3.4, 7.3.5). Finally, we propose a highly accessible attack using universal adversarial perturbations — we find that it is possible to craft a single input-agnostic perturbation that can be added across all frames of any given video

to cause classification to the target label by many seen and unseen detectors. Once crafted, this perturbation can be easily shared amongst adversaries thereby posing a very practical challenge to Deepfake detection (Section 7.3.6).

7.3.1 Threat Model

Given a video (*Real* or *Fake*), our task is to adversarially modify the video such that the label predicted by a victim Deepfake detection method is incorrect. That is, we want to modify the videos such that the *Fake* videos are classified as *Real* and vice-versa. Misclassifying a *Fake* video as *Real* can be used by the adversary to propagate false information. Misclassifying a *Real* video as *Fake* can be used by the adversary to cover up an event that did actually happen.

Distortion Metric

To ensure imperceptibility of the adversarial modification, the L_p norm is a widely used distance metric for measuring the distortion between the adversarial and original inputs. The authors of [11] recommend constraining the maximum distortion of any individual pixel by a given threshold ε, i.e., constraining the perturbation using an L_∞ metric. Additionally, *Fast Gradient Sign Method* (FGSM) [11] based attacks, which are optimized for the L_∞ metric, are more time-efficient than attacks which optimize for L_2 or L_0 metrics. Since each video can be composed of thousands of individual frames, time-efficiency becomes an important consideration to ensure the proposed attack can be reliably used in practice. Therefore, in this work, we use the L_∞ distortion metric for constraining our adversarial perturbation and optimize for it using gradient sign based methods.

Notation

We follow the notation previously used in [13, 281]; we define F to be the full neural network (classifier) including the softmax function, $Z(x) = z$ to be the output of all layers except

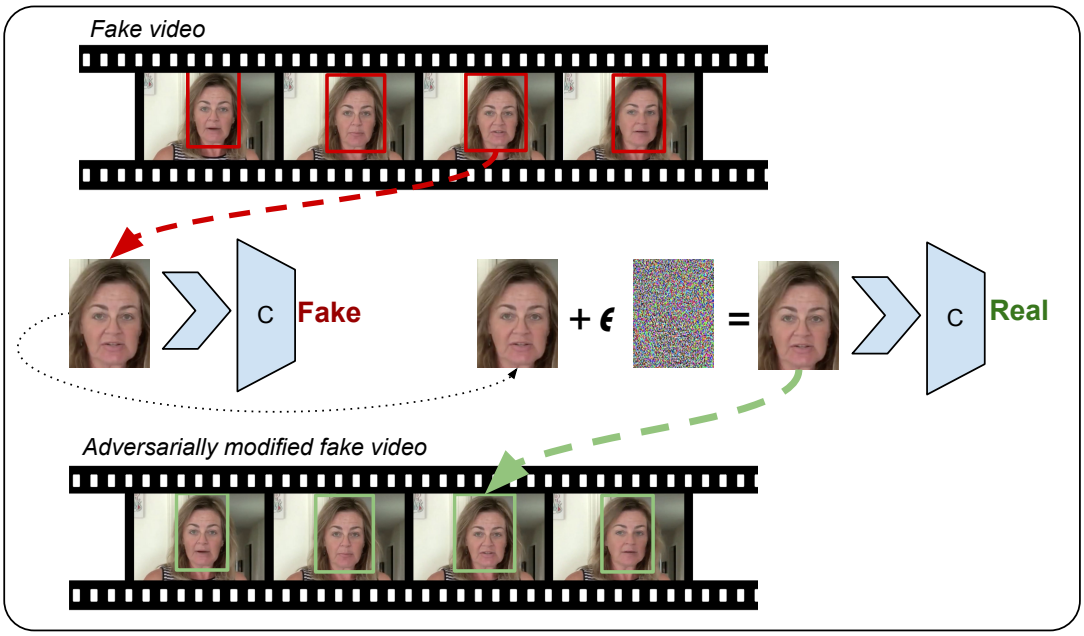


Figure 7.3. An overview of our attack pipeline to generate Adversarial Deepfakes. We generate an adversarial example for each frame in the given fake video and combine them together to create an adversarially modified fake video.

the softmax (that is z are the logits). That is:

$$F(x) = softmax(Z(x)) = y$$

The classifier assigns the label $C(x) = \arg\max_i(F(x)_i)$ to input frame x.

Problem Formulation

Mathematically, for each video frame x, we aim to find an adversarial frame x_{adv} such that:

$$C(x_{adv}) = y \text{ and } ||x_{adv} - x_0||_\infty < \varepsilon$$

where y is the target label. In our case the target label is *Real* for *Fake* videos and *Fake* for *Real* videos. In the upcoming sections, we study this attack goal in various attacker knowledge settings and constraints.

Attack Pipeline

An overview of the process of generating adversarial fake videos is depicted in Figure 7.3. For any given frame, we craft an adversarial example for the cropped face, such that after going through some image transformations (normalization and resizing), it gets classified as *Real* by the classifier. The adversarial face is then placed in the bounding box of face-crop in the original frame, and the process is repeated for all frames of the video to create an adversarially modified fake video. In the following sections, we consider our attack pipeline under various settings and goals. Note that, the proposed attacks can also be applied on detectors that operate on entire frames as opposed to face-crops. We choose face-crop based victim models because they have been shown to outperform detectors that operate on entire frames for detecting facial-forgeries.

7.3.2 Simple White-box attack

In this setting, we assume that the attacker has complete access to the detection model, including the face extraction pipeline and the architecture and parameters of the classification model. To construct adversarial examples using the attack pipeline described above, we use the iterative gradient sign method [83] to optimize the following objective:

$$\begin{aligned} &\text{Minimize } loss(x') \text{ where} \\ &loss(x') = max(Z(x')_o - Z(x')_y, 0) \end{aligned} \tag{7.1}$$

Here, $Z(x)_y$ is the final score for target label y and $Z(x)_o$ is the score of the original label o before the softmax operation in the classifier C. The loss function we use is recommended by [13] because it is empirically found to generate less distorted adversarial samples and is robust against defensive distillation. We use the iterative gradient sign method to optimize the above loss function while constraining the magnitude of the perturbation as follows:

$$x_i = x_{i-1} - \text{clip}_\varepsilon(\alpha \cdot \text{sign}(\nabla loss(x_{i-1}))) \tag{7.2}$$

We continue gradient descent iterations until success or until a given number of maximum iterations, whichever occurs earlier. We solve the optimization problem for each frame of the given video and combine all the adversarial frames together to generate the adversarial video. In our experiments, we demonstrate that we are able to successfully fool all the detection methods studied in our work in the white-box attack setting using the above attack. However, the transferability of adversarial examples generated using this attack across different methods is limited. In the next section we propose techniques to overcome this challenge.

7.3.3 Robust and Transferable attack

Videos uploaded to social networks and other media sharing websites are usually compressed. Standard operations like compression and resizing are known to remove adversarial perturbations from an image [282, 283, 284]. To ensure that the adversarial videos remain effective even after compression, it is important to ensure robustness to input-transformation functions while training the perturbation.

Also, past works [285, 286, 287, 10, 288, 289] have studied that adversarial inputs can transfer across different models. That is, an adversarial input that was designed to fool a particular victim model can possibly fool other models that were trained for the same task. This is because different models learn similar decision boundaries and therefore have similar vulnerabilities. However, for Deepfake detectors, the goal of making transferable adversarial videos is more challenging due to multiple steps involved in the Deepfake detection pipeline and the differences in these steps across various methods.

- Different face detection methods result in different face-crops.
- Different data-augmentation procedures during training result in different levels of robustness to adversarial examples.
- Different input pre-processing pipelines, such as image resizing, cropping and channel normalization parameters vary across different detection methods.

Therefore ensuring robustness to input transformation functions not only helps create adversarial videos that are robust to compression, but can also potentially result in adversarial videos that are transferable across different detection methods. We use the expectation over transforms [151] attack to craft robust and transferable adversarial examples. Given a distribution of input transformations T, input image x, and target class y, our objective is as follows:

$$x_{adv} = argmax_x \mathbb{E}_{t\sim T}[F(t(x))_y] \text{ s.t. } ||x - x_0||_\infty < \varepsilon$$

That is, we want to maximize the expected probability of target class y over the distribution of input transforms T. To solve the above problem, we update the loss function given in Equation 7.1 to be an expectation over input transforms T as follows:

$$loss(x) = \mathbb{E}_{t\sim T}[max(Z(t(x))_o - Z(t(x))_y, 0)]$$

Following the law of large numbers, we estimate the above loss functions for n samples as:

$$loss(x) = \frac{1}{n}\sum_{t_i\sim T}[max(Z(t_i(x))_o - Z(t_i(x))_y, 0)] \tag{7.3}$$

Since the above loss function is a sum of differentiable functions, it is tractable to compute the gradient of the loss w.r.t. to the input x. We minimize this loss using the iterative gradient sign method given by Equation 7.2. We iterate until a given number of maximum iterations or until the attack is successful under the sampled set of transformation functions, whichever happens first.

Next we describe the class of input transformation functions we consider for the distribution T:

- **Gaussian Blur:** Convolution of the original image with a Gaussian kernel k. This transform is given by $t(x) = k * x$ where $*$ is the convolution operator.

- **Gaussian Noise Addition:** Addition of Gaussian noise sampled from $\Theta \sim \mathcal{N}(0, \sigma)$ to the input image. This transform is given by $t(x) = x + \Theta$

- **Translation:** We pad the image on all four sides by zeros and shift the pixels horizontally and vertically by a given amount. Let t_x be the transform in the x axis and t_y be the transform in the y axis, then $t(x) = x'_{H,W,C}$ s.t. $x'[i, j, c] = x[i + t_x, j + t_y, c]$

- **Downsizing and Upsizing:** The image is first downsized by a factor r and then up-sampled by the same factor using bilinear re-sampling.

The details of the hyper-parameter search distribution used for these transforms can be found in the Section 7.5.1.

7.3.4 Query based Black-box Attack

In the black-box setting, we consider the more challenging threat model in which the adversary does not have access to the classification network architecture and parameters. We assume that the attacker has knowledge of the detection pipeline structure and the face tracking model. However, the attacker can solely query the classification model as a black-box function to obtain the probabilities of the frame being *Real* or *Fake*. Hence there is a need to estimate the gradient of the loss function by querying the model and observing the change in output for different inputs, since we cannot backpropagate through the network.

We base our algorithm for efficiently estimating the gradient from queries on the Natural Evolutionary Strategies (NES) approach of [290, 291]. Since we do not have access to the pre-softmax outputs Z, we aim to maximize the class probability $F(x)_y$ of the target class y. Rather than maximizing the objective function directly, NES maximizes the expected value of the function under a search distribution $\pi(\theta|x)$. That is, our objective is:

$$\textit{Maximize: } \mathbb{E}_{\pi(\theta|x)}[F(\theta)_y]$$

This allows efficient gradient estimation in fewer queries as compared to finite-difference methods. From [290], we know the gradient of expectation can be derived as follows:

$$\nabla_x \mathbb{E}_{\pi(\theta|x)}\left[F(\theta)_y\right] = \mathbb{E}_{\pi(\theta|x)}\left[F(\theta)_y \nabla_x \log\left(\pi(\theta|x)\right)\right]$$

Similar to [291, 290], we choose a search distribution $\pi(\theta|x)$ of random Gaussian noise around the current image x. That is, $\theta = x + \sigma\delta$ where $\delta \sim \mathcal{N}(0,I)$. Estimating the gradient with a population of n samples yields the following variance reduced gradient estimate:

$$\nabla\mathbb{E}[F(\theta)] \approx \frac{1}{\sigma n}\sum_{i=1}^{n} \delta_i F(\theta + \sigma\delta_i)_y$$

We use antithetic sampling to generate δ_i similar to [292, 291]. That is, instead of generating n values $\delta \sim \mathcal{N}(0,I)$, we sample Gaussian noise for $i \in \{1,\ldots,\frac{n}{2}\}$ and set $\delta_j = -\delta_{n-j+1}$ for $j \in \{(\frac{n}{2}+1),\ldots,n\}$. This optimization has been empirically shown to improve the performance of NES. Algorthim 6 details our implementation of estimating gradients using NES. The transformation distribution T in the algorithm just contains an identity function i.e., $T = \{I(x)\}$ for the black-box attack described in this section.

After estimating the gradient, we move the input in the direction of this gradient using iterative gradient sign updates to increase the probability of the target class:

$$x_i = x_{i-1} + \text{clip}_\varepsilon(\alpha \cdot \text{sign}(\nabla F(x_{i-1})_y)) \tag{7.4}$$

7.3.5 Query based Robust Black-box Attack

To ensure robustness of adversarial videos to compression, we incorporate the Expectation over Transforms (Section 7.3.3) method in the black-box setting for constructing adversarial videos.

To craft adversarial examples that are robust under a given set of input transformations T, we maximize the expected value of the function under a search distribution $\pi(\theta|x)$ and our distribution of input transforms T. That is, our objective is to maximize:

$$\mathbb{E}_{t\sim T}[\mathbb{E}_{\pi(\theta|x)}[F(t(\theta))_y]]$$

Following the derivation in the previous section, the gradient of the above expectation can be estimated using a population of size n by iterative sampling of t_i and δ_i:

$$\nabla\mathbb{E}[F(\theta)] \approx \frac{1}{\sigma n}\sum_{i=1,t_i\sim T}^{n}\delta_i F(t_i(\theta+\sigma\delta_i))_y$$

Algorithm 6. NES Gradient Estimate

Input: Classifier $F(x)$, target class y, image x
Output: Estimate of $\nabla_x F(x)_y$
Parameters: Search variance σ, number of samples n, image dimensionality N
$g \leftarrow 0_n$
for $i = 1$ to n **do**
    $t_i \sim T$
    $u_i \leftarrow \mathcal{N}(0_N, I_{N\cdot N})$
    $g \leftarrow g + F(t_i(x+\sigma\cdot u_i))_y \cdot u_i$
    $g \leftarrow g - F(t_i(x-\sigma\cdot u_i))_y \cdot u_i$
return $\frac{1}{2n\sigma}g$

We use the same class of transformation functions listed in Section 7.3.3 for the distribution T. Algorithm 6 details our implementation for estimating gradients for crafting robust adversarial examples. We follow the same update rule given by Equation 7.4 to generate adversarial frames. We iterate until a given a number of maximum iterations or until the attack is successful under the sampled set of transformation functions.

7.3.6 Universal attack

While the transferability of adversarial perturbations poses a practical threat to Deepfake detectors in production, creating an adversarial video requires significant technical expertise in adversarial machine learning — the attacker needs to solve an optimization problem for each frame of the video to fool the detector.

To ease the process of fooling Deepfake detectors, we aim to design more accessible adversarial attacks that can be easily shared amongst attackers. Past works [12, 293, 2] have shown the existence of universal adversarial perturbations that can fool classification models in various input domains. We aim to find a single universal adversarial perturbation which when added across all frames of any video, will cause the victim Deepfake Detector to classify the video to a target label.

That is, we aim to find a targeted universal perturbation δ such that:

$$\begin{aligned} &C(x+\delta) = y \quad s.t \quad ||\delta||_\infty < \varepsilon \\ &\text{for "most" } x \text{ in our dataset} \end{aligned} \tag{7.5}$$

where y is the target class. We train separate perturbations for Real and Fake target labels. In order to ensure robustness to differences across detection methods, we incorporate the transformation functions described in Section 7.3.3. We train the universal adversarial perturbation on a dataset of videos that are labelled opposite from our target label. On this dataset of videos, we aim to maximize the log-likelihood of predicting our target label y. Additionally to ensure the imperceptibility of the adversarial perturbation we penalize the L_2 distortion of the perturbation by adding a regularization term in our objective. Thus, our final objective to train a universal perturbation for a target label y is as follows:

$$\begin{aligned} &\textit{Minimize} \sum_{x \text{ in } D} \mathbb{E}_{t\sim T}[L(F(t(x+\delta)), y)] + c||\delta||_2 \\ &\textit{such that} \quad ||\delta||_\infty < \varepsilon \end{aligned} \tag{7.6}$$

Here, L is the cross-entropy loss between the predictions and our target label, c is a hyper-parameter to control the regularization loss and x is an input frame of a video from our dataset D. Similar to Equation 7.3, we estimate the above expectation using n samples as follows:

$$\mathbb{E}_{t\sim T}[L(F(t(x+\delta)),y)] = \frac{1}{n}\sum_{t_i\sim T}[L(F(t_i(x+\delta)),y)] \tag{7.7}$$

To ensure the constraint $||\delta||_\infty < \varepsilon$, we express δ as follows:

$$\delta = \varepsilon \cdot tanh(p)$$

where p is a trainable unconstrained parameter having the same dimensions as δ. We fix the size of the perturbation vector p to be $3 \times 256 \times 256$ in our experiments, but resize the perturbation using bilinear interpolation to match the size of our input x. We iteratively optimize the objective given by Equation 7.6 using gradient descent. In our experiments, we find that targeting certain Deepfake detectors not only results in input-agnostic universal perturbations but also model-agnostic universal perturbations.

7.4 Experimental Setup

We perform adversarial attacks on the FaceForensics++ [6] and the DFDC datasets [261] and choose the best performing models on these datasets as the victim models. We first craft adversarial videos for the FaceForensics++ dataset and target the XceptionNet and MesoNet models which are the best reported architectures reported in the paper [6] introducing this dataset (Section 7.5.1). We use these two models as a test-bed to study the robustness of our attacks to video compression and demonstrate the using our robust attack helps significantly improve attack performance on compressed videos. Next we conduct the transferable and universal attack experiments on the DFDC dataset. We choose the models from top three winning entries in the DFDC Kaggle competition as the victim models for these experiments (Section 7.5.2, 7.5.3).

Finally, we evaluate our attacks on a sequence based 3D CNN model to demonstrate that adversarial examples are a threat to not only frame by frame detectors but also sequence based models (Section 7.5.4).

7.4.1 Dataset and Models

On the FaceForensics++ dataset, XceptionNet [257] and MesoNet [256] CNN classifiers have been reported to achieve the best performance in the paper introducing the dataset [6]. For these two models, we perform our attack on the test set of the FaceForensics++ Dataset [6], consisting of manipulated videos from the four methods described in Section 7.1. We construct adversarially modified fake videos on the FaceForensics++ test set, which contains 70 videos (total 29,764 frames) from each of the four manipulation techniques. For simplicity, our experiments are performed on high quality (HQ) videos, which apply a light compression on raw videos. The accuracy of the detector models for detecting facially manipulated videos on this test set is reported in Table 7.1.

Table 7.1. Accuracy of Deepfake detectors on the FaceForensics++ HQ Dataset as reported in [6]. The results are for the entire high-quality compressed test set of Deepfakes.

	DF	**F2F**	**FS**	**NT**
XceptionNet [6] Acc %	97.49	97.69	96.79	92.19
MesoNet [6] Acc %	89.55	88.6	81.24	76.62

For the DFDC dataset, we choose the top three winners of the challenge, which was hosted by Facebook on the Kaggle website. The top two winning entries of the challenge rely solely on face detection models and per-frame CNN classifiers similar to the best performing models on the FaceForensics++ dataset. The third place winner of the challenge uses a combination of per-frame classifiers and a 3D CNN based sequence model. Table 7.2 lists the Deepfake detection methods studied in this work along with their respective CNN architectures used for classification and face detection. We use the DFDC dataset and these top three winning models as the test bed for evaluating the transferability of our attacks across different models. In our

Table 7.2. Different Deepfake detection systems studied in our work with their respective classification models, face detection models and detection AUC scores on the DFDC test set.

Model	*Team Name*	*Classifier*	*Face detection*	*AUC*
EN-B7 Selim [5]	Selim	EfficientNet B7 [127]	MTCNN [294]	0.717
XN WM [258]	Team WM	XceptionNet [257]	RetinaFace [295]	0.724
EN-B3 WM [258]	Team WM	EfficientNet B3 [127]	RetinaFace [295]	0.724
EN-B7 NLab [259]	NTech Lab	EfficientNet B7 [127]	DSFD [296]	0.717

transferability experiments we use the terms *victim model* and *test model* and define them as:

- *Victim model:* The detection model that the attack/adversarial perturbation is trained on, in the complete-knowledge (white-box) attack scenario.
- *Test model:* The model on which we evaluate the attack. This can be the same as the victim model (white-box) or an unseen detection model (black-box).

We craft adversarial videos for the first 100 Fake and 100 Real videos in the public DFDC validation set [261]. These videos contain a total of 30,300 frames. The videos are recorded in various lighting and background conditions and include people with different skin-tones.

7.4.2 Evaluation Metrics

Once the adversarial frames are generated, we combine them and save the adversarial videos in the following formats:

- *Uncompressed (Raw):* Video is stored as a sequence of uncompressed images.
- *Compressed (MJPEG):* Video is saved as a sequence of JPEG compressed frames.
- *Compressed (H.264):* Video is saved in the commonly used mp4 format that applies temporal compression across frames.

We conduct our primary evaluation on the *Raw* and *MJPEG*. We also study the effectiveness of our white box robust attack using different compression levels in the *H264* codec. We report the following metrics for evaluating our attacks:

- **Success Rate (SR)**: The percentage of frames in the adversarial videos that get classified to our target label. We report: **SR-U**- Attack success rate on uncompressed adversarial videos saved in Raw format; and **SR-C**- Attack success rate on compressed adversarial videos saved in MJPEG format.

- **Accuracy**: The percentage of frames in videos that get classified to their original label by the detector. We report **Acc-C**- accuracy of the detector on compressed adversarial videos.

- **Mean distortion (L_∞)**: The average L_∞ distortion between the adversarial and original frames. The pixel values are scaled in the range [0,1], so changing a pixel from full-on to full-off in a grayscale image would result in L_∞ distortion of 1 (not 255).

7.5 Results

7.5.1 Evaluation on FaceForensics++ dataset

Simple white-box attack

To craft adversarial examples in the white-box setting, in our attack pipeline, we implement differentiable image pre-processing (resizing and normalization) layers for the CNN. This allows us to backpropagate gradients all the way to the cropped face in-order to generate the adversarial image that can be placed back in the frame. We set the maximum number of iterations to 100, learning rate α to $1/255$ and max L_∞ constraint ε to $16/255$ for both our attack methods described in Sections 7.3.2 and 7.3.3.

Table 7.3 shows the results of the white-box attack (Section 7.3.2). We are able to generate adversarial videos with an average success rate of 99.85% for fooling XceptionNet and 98.15% for MesoNet when adversarial videos are saved in the Raw format. However, the attack

Table 7.3. Evaluation of various attacks on the two models XceptionNet and MesoNet on the FaceForensics++ dataset. We report the average L_∞ distortion between the adversarial and original frames and the attack success rate on uncompressed (SR-U) and compressed (SR-C) videos.

		XceptionNet				MesoNet			
Attack	Dataset	$\mathbf{L}_\infty$	SR - U	SR - C	Acc-C %	$\mathbf{L}_\infty$	SR - U	SR - C	Acc-C %
Simple White-box (Section 7.3.2)	DF	0.004	99.67	43.11	56.89	0.006	97.30	92.27	7.73
	F2F	0.004	99.85	52.50	47.50	0.007	98.94	96.30	4.70
	FS	0.004	100.00	43.13	56.87	0.009	97.12	86.10	13.90
	NT	0.004	99.89	95.10	4.90	0.007	99.22	96.20	3.80
	All	0.004	99.85	58.46	41.54	0.007	98.15	92.72	7.53
Robust and Transferable (Section 7.3.3)	DF	0.016	99.56	98.71	1.29	0.030	99.94	99.85	0.15
	F2F	0.013	100.00	99.00	1.00	0.020	99.71	99.67	0.33
	FS	0.013	100.00	95.33	4.67	0.026	99.02	98.50	1.50
	NT	0.011	100.00	99.89	0.11	0.025	99.99	99.98	0.02
	All	0.013	99.89	98.23	1.77	0.025	99.67	99.50	0.50
Query based Black-box (Section 7.3.4)	DF	0.055	89.72	55.64	44.36	0.062	96.05	93.33	6.67
	F2F	0.055	92.56	81.40	18.60	0.0627	84.08	77.68	22.32
	FS	0.045	96.77	23.50	76.50	0.0627	77.55	62.44	37.56
	NT	0.024	99.86	94.23	5.77	0.0627	85.98	79.25	20.75
	All	0.045	94.73	63.69	36.31	0.0626	85.92	78.18	21.83
Query based Robust Black-box (Section 7.3.5)	DF	0.060	88.47	79.18	20.82	0.047	96.19	93.80	93.80
	F2F	0.058	97.68	94.42	5.58	0.054	84.14	77.50	77.50
	FS	0.052	98.97	63.26	36.74	0.061	77.34	61.77	61.77
	NT	0.018	99.65	98.91	1.09	0.053	88.05	80.27	80.27
	All	0.047	96.19	83.94	16.06	0.053	86.43	78.33	78.33

average success rate drops to 58.46% for XceptionNet and 92.72% for MesoNet when MJPEG compression is used. This result is coherent with past works [282, 283, 284] that employ JPEG compression and image transformations to defend against adversarial examples.

Robust attack

For our robust white box attack, we sample 12 transformation functions from the distribution T for estimating the gradient in each iteration. This includes three functions from each of the four transformations listed in Section 7.3.3. Table 7.4 shows the search distribution for different hyper-parameters of the transformation functions.

Table 7.4. Search distribution of hyper-parameters of different transformations used for our Robust White box attack. During training, we sample three functions from each of the transforms to estimate the gradient of our expectation over transforms.

Transform	Hyper-parameter search distribution
Gaussian Blur	Kernel $k(d,d,\sigma)$, $d \sim \mathscr{U}[3,7]$, $\sigma \sim \mathscr{U}[5,10]$
Gaussian Noise	$\sigma \sim \mathscr{U}[0.01,0.02]$
Translation	$d_x \sim \mathscr{U}[-20,20]$, $d_y \sim \mathscr{U}[-20,20]$
Down-sizing & Up-sizing	Scaling factor $r \sim \mathscr{U}[2,5]$

Table 7.3 shows the results of our robust white-box attack. It can be seen that robust white-box is effective in both Raw and MJPEG formats. The average distortion between original and adversarial frames in the robust attack is higher as compared to the non-robust white-box attack. We achieve an average success rate (SR-C) of 98.07% and 99.83% for XceptionNet and MesoNet respectively in the compressed video format.

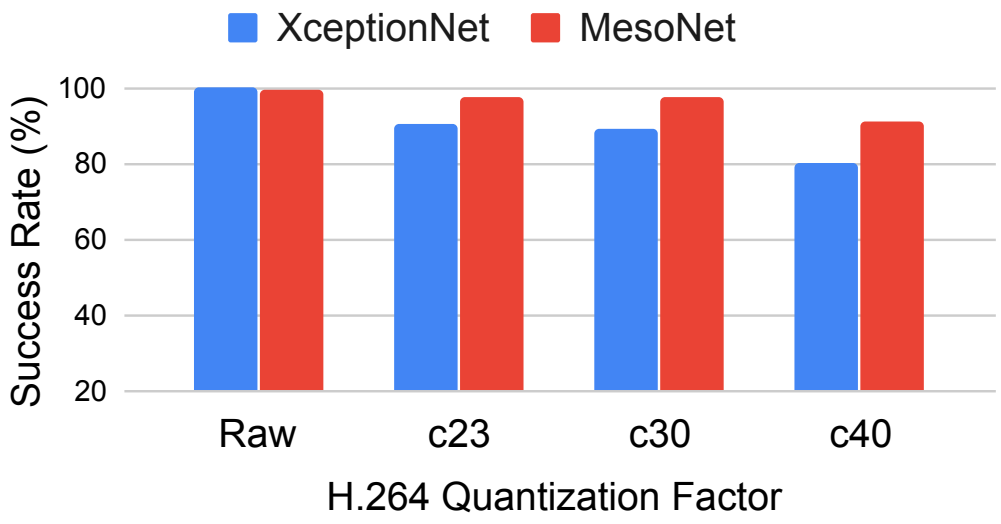


Figure 7.4. Attack success rate vs Quantization factor used for compression in H264 codec for robust white box attack.

We also study the effectiveness of our robust white box attack under different levels of compression in the H.264 format which is widely used for sharing videos over the internet. Figure 7.4 shows the average success rate of our attack across all datasets for different quantiza-

tion parameter c used for saving the video in H.264 format. The higher the quantization factor, the higher the compression level. In [6], fake videos are saved in HQ and LQ formats which use $c = 23$ and $c = 40$ respectively. It can be seen that even at very high compression levels ($c = 40$), our attack is able to achieve 80.39% and 90.50% attack success rates for XceptionNet and MesoNet respectively, without any additional hyper-parameter tuning for this experiment. Video examples of Adversarial Deepfakes can be found on the website linked in the footnote [2].

7.5.2 Transferability of adversarial attacks

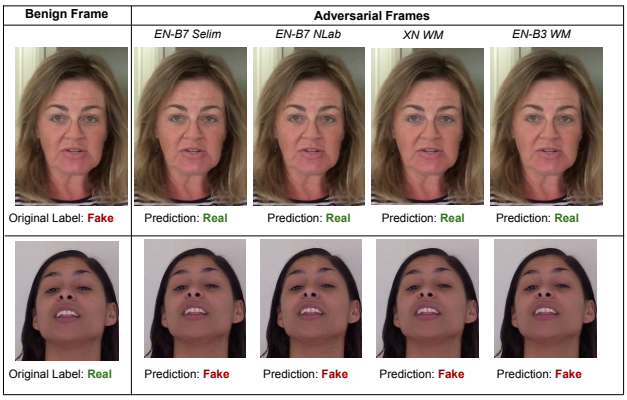
We evaluate the transferability of adversarial perturbations across different detectors trained on the DFDC dataset. We train adversarial videos targeting a given victim model and test the videos against different test models. For our simple whitebox attack, while we achieve 100% attack success rate for the same test model as the victim model, the attack success rate drops significantly on alternate models. EfficientNet-B7 by NTech Lab requires the highest amount of adversarial perturbation under the L_∞ metric as compared to other methods in this study. We find that perturbations trained to fool EfficientNet-B7 by Team NTech Lab result in the most transferable attacks as indicated by the higher success rates on other test models. This suggests that *EN-B7 NLab* is relatively more robust to adversarial perturbations in comparison to the other models used in this study (also indicated by higher L_∞ perturbation required to fool *EN-B7 NLab*).

To improve the transferability of adversarial examples across different methods, we perform our robust transfer attack described in Section 7.3.3 and evaluate the adversarial videos against unseen detection methods in a black-box setting. The hyper-parameters of the transformation functions used for the attack have been provided in Table 7.4. All other attack hyper-parameters are kept the same as our simple white-box attack.

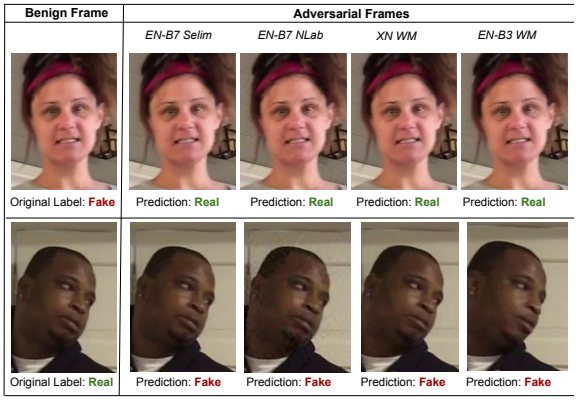
As indicated by the results in Table 7.5, we are able to significantly improve the transferability of adversarial perturbations across different detection methods as compared to our simple

[2] https://adversarialdeepfakes.github.io/

white-box attack. The adversarial perturbations are most transferable across models with the same architecture. For example, we are able to achieve high cross-transferability between *EN-B7 Selim* vs *EN-B7 NLab*. Similar to our observation in the previous section, attacking *EN-B7 NLab* results in the most transferable adversarial attacks - we are able to achieve at least 72% success rate across all other detection methods when attacking *EN-B7 NLab*. Sample images for these attacks are presented in Figure 7.5.



Simple White-box attack examples on DFDC detectors



Robust and Transferable attack examples on DFDC detectors

Figure 7.5. Randomly selected frames of adversarial videos from attacks on the DFDC detectors.

Table 7.5. Attack success rates (SR-U) of the *white-box* (Section 7.3.2) and *robust and transferable attacks* (Section 7.3.3) on different victim models and their transferability to seen and unseen detectors (test models).

			Test Model			
	Victim Model	L_∞	EN-B7 Selim	EN-B7 NLab	XN WM	EN-B3 WM
Simple White-box (Section 7.3.2)	EN-B7 Selim	0.007	100.0 %	59.5 %	57.0 %	38.5 %
	EN-B7 NLab	0.013	94.0 %	100.0 %	66.5 %	49.5 %
	XN WM	0.006	13.0 %	12.5 %	100.0 %	12.0 %
	EN-B3 WM	0.005	21.0 %	15.5 %	22.0 %	100.0 %
Robust and Transferable (Section 7.3.3)	EN-B7 Selim	0.010	100.0 %	89.0 %	72.5 %	62.0 %
	EN-B7 NLab	0.018	99.0 %	100.0 %	72.0 %	76.5 %
	XN WM	0.018	49.0 %	33.5 %	100.0 %	46.0 %
	EN-B3 WM	0.008	46.5 %	35.0 %	47.5 %	100.0 %

7.5.3 Universal attacks

To create more accessible attacks, we train a universal adversarial perturbation using the procedure described in Section 7.3.6. We set the L_2 regularization term $c = 0.01$ and use

the Adam optimizer with a learning rate of 0.001. For our initial experiments, we set the L_∞ threshold $\varepsilon = 40/255$ for all victim models. Since the goal of finding a single input-agnostic perturbation is more challenging than finding one perturbation per video frame, a higher amount of distortion is required for a successful attack as compared to the per-frame attacks described earlier. We train the universal perturbation on a dataset of 100 videos from the DFDC train set which are separate from our evaluation dataset. We train the perturbation using a batch size of 8 for $10,000$ iterations.

Table 7.6. Attack success rates (SR-U) of the universal attacks (Section 7.3.6) on different victim models and their transferability to unseen detectors (test models).

		Test Model			
Victim Model	L_∞	EN-B7 Selim	EN-B7 NLab	XN WM	EN-B3 WM
EN-B7 Selim	0.156	100.0%	94.5%	65.0%	69.0%
EN-B7 NLab	0.156	94.5%	100.0%	75.0%	81.5%
XN WM	0.156	77.5%	61.0%	100.0%	20.0%
EN-B3 WM	0.156	66.5%	50.5%	60.0%	100.0%

We target one victim model at a time and test the transferability of the universal perturbation on seen and unseen detectors. Table 7.6 presents the results of performing the universal attack on different victim models at $\varepsilon = 40/255 = 0.156$. We are able to achieve 100% attack success rate on the same test model as the victim model using a single perturbation across all frames and videos of the same label. Also, the universal perturbation is transferable to a significant extent across different models which poses an extremely practical threat to Deepfake detectors in production. Attacking *EN-B7 NLab* results in the most transferable perturbations where we are able to achieve at least a 75% success rate across all unseen detectors.

Visually, the universal perturbations at $\varepsilon = 0.156$ are more perceptible than our per-frame attacks discussed in the sections above. Figure 7.6 shows examples of universal adversarial perturbations trained on different Deepfake detectors and the resulting adversarial images obtained after adding the perturbation to the face-crop of the benign frame.

We perform an additional experiment to study the effectiveness of universal adversarial

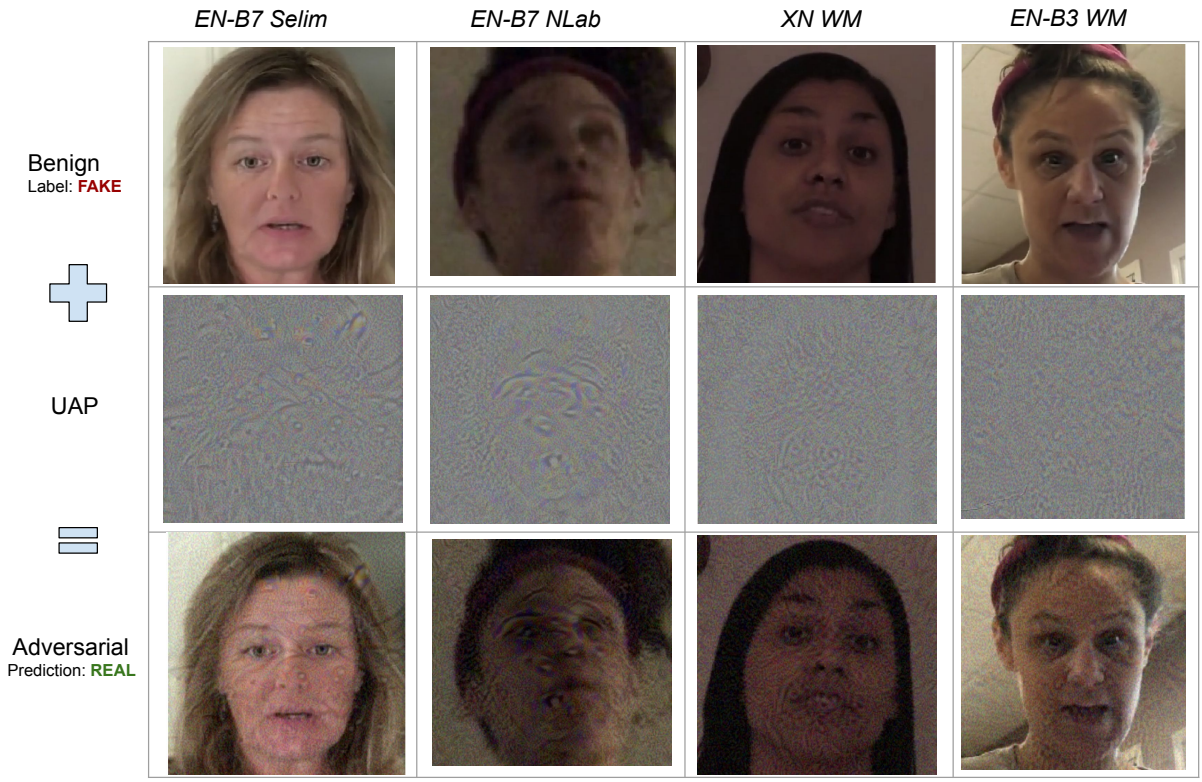


Figure 7.6. Visualization of universal adversarial perturbations trained on different Deepfake detection models at $\varepsilon = 0.156$.

perturbations at different magnitudes of added perturbations. We choose *EN-B7 NLab* as the victim model and perform our universal attack at different values of ε. The attack success rates across different models are shown in Figure 7.7. Figure 7.7 also shows what a perturbed image looks like at different values of ε. At $\varepsilon < 0.1$, the perturbation is fairly imperceptible but can still achieve high success rates on various test models.

7.5.4 Evaluation on Sequence Based Detector

We consider the 3D CNN based detector described in Section 7.2. The detector performs 3D convolution on a sequence of face-crops from 7 consecutive frames. We perform our attacks on the pre-trained model checkpoint (trained on the DFDC [261] train set) released by the NTech-Lab team [259]. We evaluate our attacks on the Deepfake videos from the DFDC public validation set which contains 200 Fake videos. We report the accuracy of the detector on the

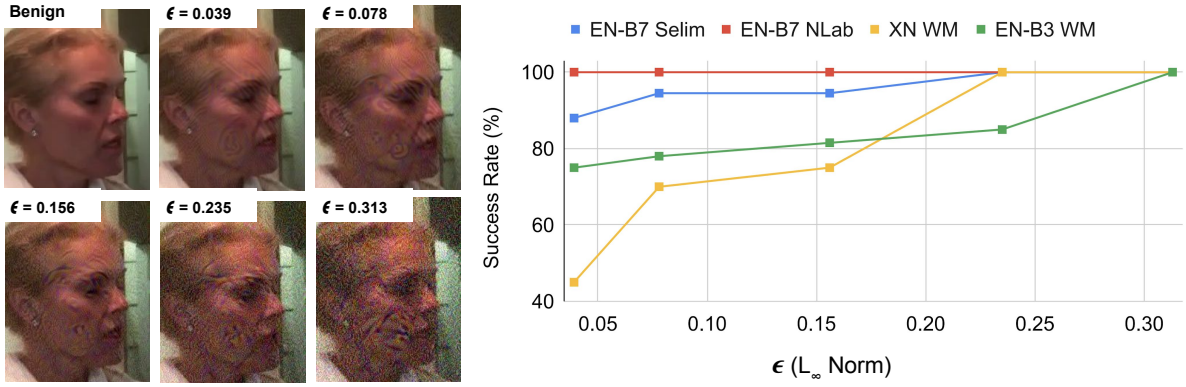


Figure 7.7. *Left:* Visualization of the perturbed images using different magnitudes (ϵ) of universal adversarial perturbations trained on *EN-B7 NLab*. *Right:* Attack success rates of the universal attacks (Section 7.3.6) on different victim models and their transferability to unseen detectors (test models).

Table 7.7. Evaluation of different attacks on a sequence based detector on the DFDC validation dataset. The first row indicates the performance of the classifier on benign (non adversarial) videos.

	3D CNN Sequence Model			
Attack Type	**L_∞**	**SR - U**	**SR - C**	**Acc. - C%**
None	-	-	-	91.74
Simple White-box (Section 7.3.2)	0.037	100.00	77.67	22.33
Robust and Transferable (Section 7.3.3)	0.059	100.00	100.00	0.00
Query based Black-box (Section 7.3.4)	0.061	87.99	24.43	75.57
Query based robust Black-box (Section 7.3.4)	0.062	88.21	51.02	48.98

7-frame sequences from this test set in the first row of Table 7.7.

Similar to our attacks on frame-by-frame detectors, in the white-box setting we back-propagate the loss through the entire model to obtain gradients with respect to the input frames for crafting the adversarial frames. While both white-box and robust white-box attacks achieve 100% success rate on uncompressed videos, the robust white-box attack performs significantly better on the compressed videos and is able to completely fool the detector. As compared to frame-by-frame detectors, a higher magnitude of perturbation is required to fool this sequence model in both the white-box attacks. In the black-box attack setting, while we achieve similar

attack success rates on uncompressed videos as the frame-by-frame detectors, the attack success rate drops after compression. The robust black-box attack helps improve robustness of adversarial perturbations to compression as observed by higher success rates on compressed videos (51.02% vs 24.43% SR-C).

7.6 Conclusion

In this chapter, I described the current best-performing Deepfake classifiers and studied their vulnerability to adversarial examples. We consider both per-frame and sequence-based Deepfake detection models and demonstrate that they can be bypassed under various attack settings and attacker capabilities. We first design an attack pipeline to bypass Deepfake detectors in a white-box attack setting and propose techniques to increase the robustness of such attacks to video compression codecs. Next, we demonstrate that adversarial videos crafted using our robust attacks can fool alternate models to a significant extent thereby posing a real-world threat in a black-box attack setting. Finally, we demonstrate the existence of universal adversarial perturbations which pose a more practical threat since they can be easily shared amongst attackers and applied to any video in real-time. In the upcoming chapter, I discuss a semi-fragile watermarking framework as a proactive media authentication method to overcome the limitations of Deepfake classifiers.

7.7 Acknowledgements

Chapter 7 contains material found in the following two papers. (1) *Adversarial Deepfakes: Evaluating Vulnerability of Deepfake Detectors to Adversarial Examples*. 2021. Hussain, Shehzeen; Neekhara, Paarth; Jere, Malhar; Koushanfar, Farinaz; McAuley, Julian. IEEE Winter Conference on Applications of Computer Vision 2021. (2) *Exposing Vulnerabilities of Deepfake Detection Systems with Robust Attacks*. 2022. Hussain, Shehzeen; Neekhara, Paarth; Dolhansky, Brian; Bitton, Joanna; Canton, Cristian; McAuley, Julian; Koushanfar, Farinaz. ACM Journal on

Digital Threats: Research and Practice, Vol 3, 2022. The dissertation author was the primary investigator and author of these papers.

Chapter 8

Media Authentication using DL based Proactive Watermarking

With the advancements in deep learning-based generative models, media authentication has become an important challenge. As discussed in the previous chapters, DNN-based generative models can easily manipulate images, videos and audio to create realistic multimedia content that can support fake news, spread misinformation and reduce trust in social media platforms [297]. Media authentication is crucial in ensuring the accuracy of news and maintaining public trust. Media authentication also plays a crucial role in law enforcement, where videos and images are often used as evidence.

Recent methods to detect Deepfakes rely on DNN-based classification systems [6, 298]. As described in Chapter 7, there are certain limitations in classification-based Deepfake detectors: 1) The current best-performing detectors for synthetic media can be easily bypassed by attackers using adversarial examples. 2) Classifiers trained in a supervised manner on existing media synthesis techniques cannot be reliably secure against unseen generation methods.

To address the above challenges of Deepfake detectors, we propose to proactively embed a secret verifiable message into images and videos at the time of their capture from a device to establish media authenticity. Conventional image watermarking systems use hand-engineered pipelines [299, 300, 301, 302] to embed information in the spatial or frequency domain of an image. However, the major limitations of traditional approaches lie in the higher visibility of

the embedded watermarks, limited message capacity, and low robustness to digital compression techniques like JPEG transforms. As a result, these traditional approaches are being quickly replaced with deep learning based techniques, which are achieving state-of-the-art results in image watermarking and steganography tasks.

Deep learning based watermarking techniques rely on encoder and decoder convolutional neural networks (CNNs). These networks are trained end-to-end for the task of embedding and retrieving a given message in an image. While deep learning techniques significantly outperform hand-engineered watermarking pipelines, the improvement comes at the cost of increased computational overhead and memory requirement of these models. The best-performing neural image watermarking encoders are parameterized by around half a million floating point parameters, which makes it challenging to deploy such systems on resource-constrained hardware such as FPGAs or handheld devices. Such techniques have only been implemented at a software level which results in significant latency between image capture and transmission. Our work aims to embed watermarks in images and videos in real-time as they are being captured. Embedding the watermarks at the hardware level can not only reduce the latency of the watermarking process but also enable media authentication and provenance by leveraging unique hardware signatures from PUFs [303] or secure enclaves as the watermarking data.

In this chapter, I describe FastStamp: a light-weight yet robust neural image watermarking framework [304] we developed to enable real-time watermarking on hardware platforms. Keeping resource constraints in mind, we develop a parameter efficient CNN-based watermarking model that can match and even outperform the success metrics of state-of-the-art neural image watermarking models. Our watermarking model leverages efficient neural blocks such as depthwise separable convolutions, spatial upsampling operations, and linear layers to reduce the memory requirement and inference latency without compromising the watermark retrieval accuracy, message capacity, and imperceptibility. We then design and verify an FPGA implementation of our watermarking encoder model to allow image watermarking directly on hardware. Our most optimized design achieves real-time image watermarking and only requires 3 milliseconds

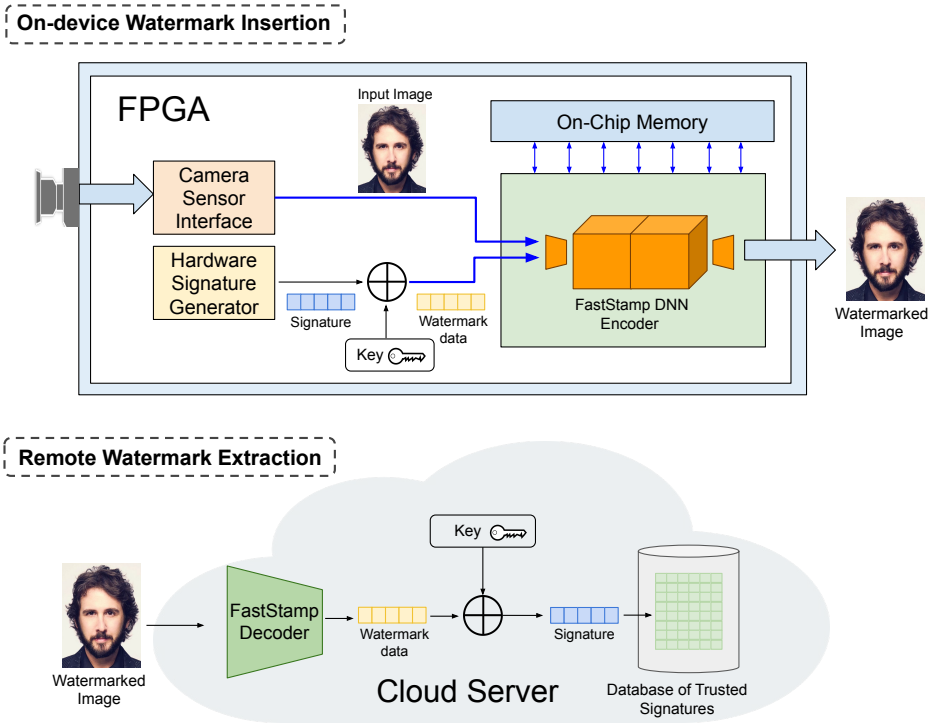


Figure 8.1. Schematic diagram of FastStamp watermarking pipeline. The pipeline is divided into two steps: watermark insertion using a DNN encoder on FPGA (top) and watermark extraction using a DNN decoder on a cloud server (bottom).

to watermark a given image while achieving the same results as the software implementations. Our framework is end-to-end and supports two types of watermarking schemes: *robust* and *semi-fragile*. FastStamp learns to be robust to a wide range of real-world digital image processing operations such as lighting, color adjustments, and compression techniques. Our semi-fragile watermarking scheme learns to be robust to above benign transformations while being fragile to media forgery and local tampering.

8.1 Background

8.1.1 Digital Watermarking

Digital watermarking techniques broadly seek to generate three different types of watermarks: fragile [305], robust ,[306, 307, 308] and semi-fragile [299, 309, 300]. Fragile and semi-fragile watermarks are primarily used to certify the integrity and authenticity of image

data. Fragile watermarks are used to achieve accurate authentication of digital media, where even a one-bit change to an image will lead it to fail the certification system. In contrast, *robust* watermarks aim to be recoverable under several image manipulations to allow media producers to assert ownership over their content even if the video is redistributed and modified. Semi-fragile watermarks combine the advantages of both robust and fragile watermarks and are mainly used for fuzzy authentication of digital images and identification of image tampering [309].

Prior work has proposed hand-engineered pipelines to embed semi-fragile watermark information in the spatial[310] and frequency domain of images and videos. Particularly, in the frequency domain, the watermark can be embedded by modifying the coefficients produced with transformations such as the discrete cosine transform (DCT) [311, 300] and discrete wavelet transform (DWT) [312, 302, 313]. For example, during 2D DWT an image is first decomposed into various frequency channels using a Haar filter. A scaled image is used as the watermark and inserted into mid frequency wavelet channel. Taking 2D inverse DWT of the altered wavelet decomposition produces the watermark embedded image. The major drawbacks of traditional approaches lie in higher visibility of the embedded watermarks, increased distortions in generated images, and low robustness to compression techniques like JPEG transforms as discussed in Section 8.4.2. Specifically, since digital images shared over social media are highly compressed and undergo various lighting and color adjustments, the watermarks generated by DWT and DCT algorithms break under benign real-world transformations and compression.

More recently, CNNs have been used to provide an end-to-end solution to the watermarking problem. They replace hand-crafted hiding procedures with neural network encoding [314, 315, 308, 316, 317, 318, 319, 320]. These techniques train end-to-end encoder CNNs to embed and decode watermarks, which have resulted in lower imperceptibility and more robust recovery of the watermark data. However, these models are memory intensive and much slower as compared to DCT based algorithms. In our work, we address the performance limitations of neural watermarking systems and propose a light-weight framework for both robust and semi-fragile image watermarking suitable for hardware acceleration platforms.

8.1.2 FPGA Accelerated Techniques

There have been several efforts to accelerate neural networks on FPGAs [179, 181, 173, 178]. Equipped with the necessary hardware for basic DNN operations, FPGAs are able to achieve high parallelism and utilize the properties of neural network computation to remove unnecessary logic. Some prior efforts have been made in FPGA acceleration of convolutional autoencoder architectures [321, 322, 323]. While these works implement many sub-blocks used in neural network computation, we find that they cannot be directly used for an image watermarking framework because efficient implementations of sub-blocks like 2D upsampling and separable convolutions with skip-connections are absent. Moreover, existing neural architectures for image watermarking are not optimized for hardware-software co-design.

Prior work [324, 325, 326, 327] has made significant efforts in accelerating traditional image watermarking schemes that hide secret information in the frequency domain of images and rely on DCT and DWT based algorithms. As discussed earlier, while such algorithms are vastly popular for hardware applications due to its simplicity and low computational overhead, the resulting watermarked image is often not robust to real-world image transformations. In our work, we develop the first FPGA accelerator platform for DNN based image watermarking and steganography that enables robustness to real-world digital image processing and compression while being selectively fragile to media tampering techniques.

8.1.3 Countering Media Forgery

With the widespread development of deep learning based image and video synthesis techniques [328, 329, 297, 249], it has become increasingly easier and faster to generate high-quality convincing fake images/videos such as Deepfakes. Such manipulated media can fuel misinformation, defame individuals and reduce trust in media. While considerable research effort has been made in designing CNN based deepfake detectors [298, 256], these techniques have been shown to hold major security vulnerabilities and can be bypassed by attackers [330].

To counter such threats, authors [331, 319, 317] have proposed semi-fragile watermarking as a solution to perform media authentication and distinguish deepfake media from real media by verifying a secret watermark embedded in the media. For both fragile and semi-fragile techniques, a watermark must be inserted when the image is captured, which makes these techniques dependent on both algorithmic and hardware implementation. If watermark information is embedded separately in images and videos after it is captured by a device, this method may fail in situations where tampering is carried out before inserting the signature or watermark. While the proposed solution to media authentication is to add a verifiable digital signature or watermark to an image/video using neural networks, prior works [319, 331, 317] do not actively implement the technology in resource constrained settings or camera hardware. Upon empirical study, we found that it is challenging to fit such off-the-shelf models [318, 308, 319] on FPGAs since they were developed without any attention to hardware-software co-design practices and range from 500 k to 2 million parameters for the encoder model. To this end, we design our own DNN based watermarking system that utilizes depthwise separable convolutions and a parameter efficient message upsampler to reduce computational overhead while preserving required bit recovery accuracy, capacity, and imperceptibility.

8.2 FastStamp Methodology

8.2.1 Training Framework

Our goal is to develop a learnable, end-to-end model for image steganography and watermarking such that the encoder model can embed a message as a visually invisible perturbation in the image, and the decoder network can extract the message from the watermarked image. We develop two variants of our training framework to generate *robust* and *semi-fragile* watermarks. A robust watermark is designed to be recoverable when real-world image transformations are applied. A semi-fragile watermark is designed to be robust to benign image transformations such as compression, minor color, and contrast adjustments but it should be unrecoverable when

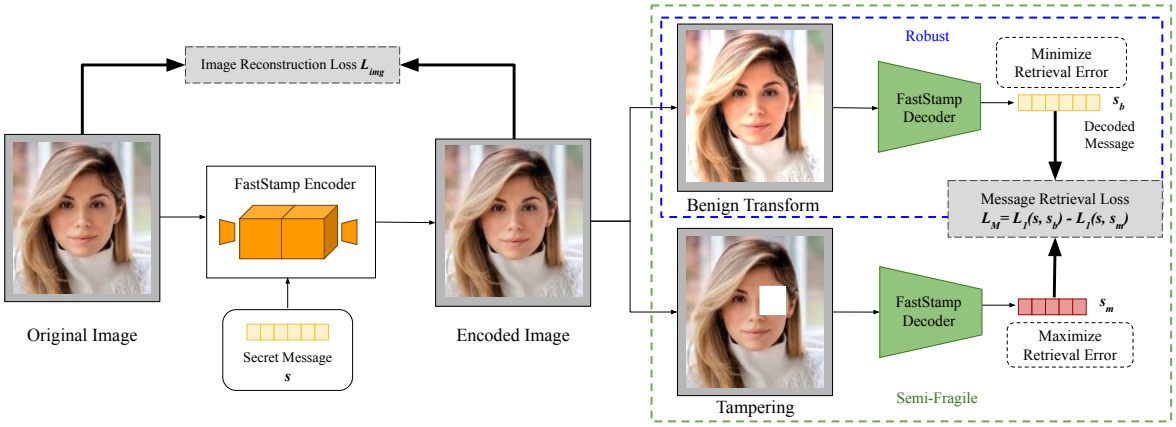


Figure 8.2. FastStamp Encoder-Decoder Training: In both robust and semi-fragile schemes, the encoder model encourages retrieval from the decoder model under benign transforms. In the semi-fragile scheme, it maximizes message retrieval error under tampered/malicious transforms.

malicious image transforms such as image tampering and face-swapping are applied. A semi-fragile watermark is designed to be robust to benign image transformations such as compression and minor color, and contrast adjustments but it should be unrecoverable when malicious image transforms such as image tampering and face-swapping are applied.

The encoder network E takes as input an image x and a bit string $s \in \{0,1\}^L$ of length L, and produces an encoded (watermarked) image x_w. That is, $x_w = E(x,s)$. Depending on our task of either *semi-fragile* or *robust* watermarking, the encoded image goes through the following operations:

1. ***Robust watermarking:*** In this setting, the image goes through a benign image transformation $g_b \sim G_b$ to produce $x_b = g_b(x_w)$. The benign image is then fed to the decoder network, which predicts the message $s_b = D(x_b)$. For optimizing secret retrieval during training, we use the L_1 distortion between the predicted and ground-truth bit strings. The decoder is encouraged to be robust to benign transformations by minimizing the message distortion $L_1(s,s_b)$. Therefore the secret retrieval error for an image $L_M(x)$ is obtained as follows:

$$L_M(x) = L_1(s,s_b) \tag{8.1}$$

2. ***Semi-fragile watermarking:*** In this setting, the watermarked image goes through two image transformation functions—one sampled from a set of benign transformations ($g_b \sim G_b$) and the other sampled from a set of malicious transformations ($g_m \sim G_m$) to produce a benign image $x_b = g_b(x_w)$ and a malicious image $x_m = g_m(x_w)$. The benign and malicious watermarked images are then fed to the decoder network, which predicts the messages $s_b = D(x_b)$ and $s_m = D(x_m)$ respectively. The decoder is encouraged to be robust to benign transformations by minimizing the message distortion $L_1(s, s_b)$; and fragile for malicious manipulations by maximizing the error $L_1(s, s_m)$. Therefore the secret retrieval error for an image $L_M(x)$ is obtained as follows:

$$L_M(x) = L_1(s, s_b) - L_1(s, s_m) \tag{8.2}$$

The watermarked image is encouraged to look visually similar to the original image by optimizing three image distortion metrics: L_1, L_2 and L_{pips} [332] distortions.

$$L_{\text{img}}(x, x_w) = L_1(x, x_w) + L_2(x, x_w) + c_p L_{\text{pips}}(x, x_w) \tag{8.3}$$

Therefore, the parameters α, β of the encoder and decoder network are trained using mini-batch gradient descent to optimize the following loss over a distribution of input messages and images:

$$\mathbb{E}_{x,s,g_b,g_m}[L_{\text{img}}(x, x_w) + c_M L_M(x)] \tag{8.4}$$

In the above equations, c_p, and c_M are scalar coefficients for the respective loss terms. We refer the readers to our supplementary material for the values we use for these coefficients and other implementation details.

8.2.2 Message encoding

The input of our encoder network is a bit string s of length L. This watermarking data includes a secret message or a hardware signature generated by trusted execution environments or PUFs. To further ensure message secrecy, we can encrypt the message using a stream cipher with a secret key that is shared between the encoding and decoding devices. In our work we embed messages of length 128 bits in an image of size 128×128, which allows embedding 2^{128} unique messages in each 128×128 image patch.

8.2.3 Model Architecture and Optimization

The encoder model takes as input an image x and a message bit-string s to produce a watermarked image x_w. The encoder model of a typical neural watermarking system follows a convolutional U-Net architecture comprising several downsampling and upsampling layers with skip-connections. In prior work [318, 319], the secret message bit-string is first projected using a learnable linear layer reshaped as a matrix to have the same height and width as the input image; and then attached as the fourth channel of the input image. The combined input and secret image then undergo the downsampling and upsampling operations of the U-Net to produce the watermarked image.

The above described encoder model architecture in prior work [318, 308, 319] has a large memory footprint and is unsuitable for deployment in resource-constrained settings. To reduce the model size without compromising on the watermarking performance, we propose the following architectural optimizations:

Secret Message Upsampler

The secret message upsampler projects the message string s to a matrix that gets attached as the fourth channel of the input image. A naïve implementation using a linear layer can result in a large memory footprint since the number of parameters is given by hwL, where h and w are the input image height and width, respectively, and L is the secret message length. An input

image size of 128×128 and a message length of 128, would result in more than two million parameters. To optimize the number of parameters, we perform the secret upsampling operation as follows:

1. Project the message s to a vector s_{proj} of size $h'w'$ using a linear layer.
2. Reshape s_{proj} to a matrix $s_{\text{proj}M}$ of dimensions (h', w')
3. Upsample $s_{\text{proj}M}$ to a matrix of size (h, w) using nearest-neighbour upsampling.

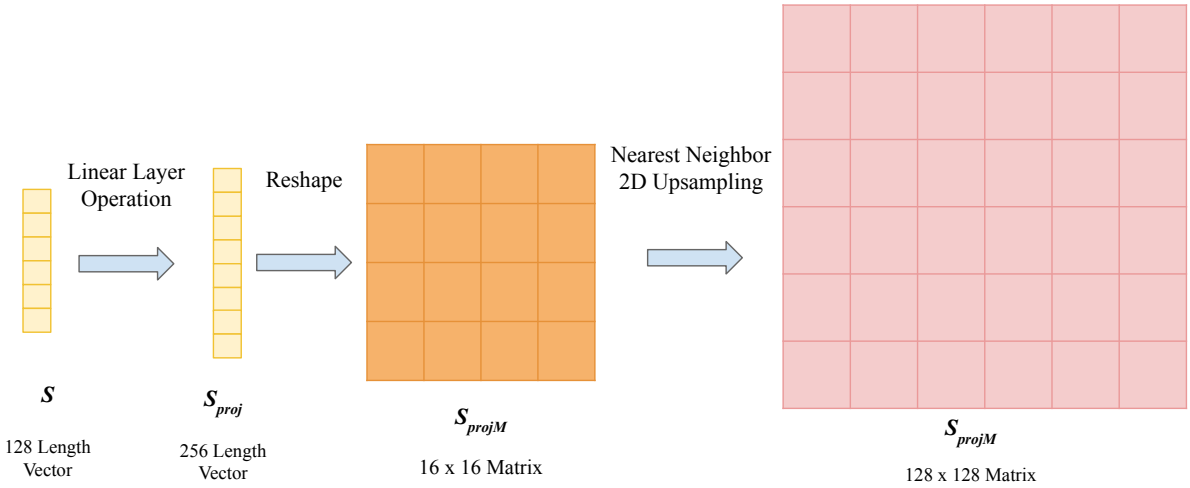


Figure 8.3. An example of optimized secret message upsampling using linear layer projection followed by nearest neighbor 2D upsampling.

The nearest-neighbour upsampling operation is parameter-free and computationally more efficient as compared to matrix-vector multiplication. Through our experiments, we find that using an (h', w') that are much smaller than (h, w) can achieve the same watermarking while being significantly efficient in both time and memory. For an image of size $(128, 128)$ and message length $L = 128$, we use $h' = w' = 16$ thus requiring only 32768 parameters. The upsampled secret gets attached as the fourth channel of the input image and undergoes the U-Net downsampling and upsampling operations described below.

U-Net Downsampling

The downsampling network in U-Net architecture typically comprises 5 to 8 convolutional blocks. Each block contains a strided convolutional layer, a batch normalization layer, and a non-linear activation like ReLU or leaky ReLU. The number of output channels of each convolutional layer increase with the depth of the network, doubling at each step until a maximum value is reached. To optimize this architecture, we first replace the convolutional layers with depthwise and point-wise separable convolutional layers [333]. Not only does this optimization reduce the number of parameters but also reduces the number of floating point operations required in each layer computation. Next, we optimize a number of output channels of each convolutional block. In our experiments, we perform a design-space exploration to find that we can substantially reduce the number of output channels in each layer without compromising on secret retrieval accuracy and imperceptibility. Our most optimized design uses 5 downsampling layers with 64 output channels in the final separable convolutional layer. Figure 8.4 details the network architecture and output tensor sizes after each downsampling step.

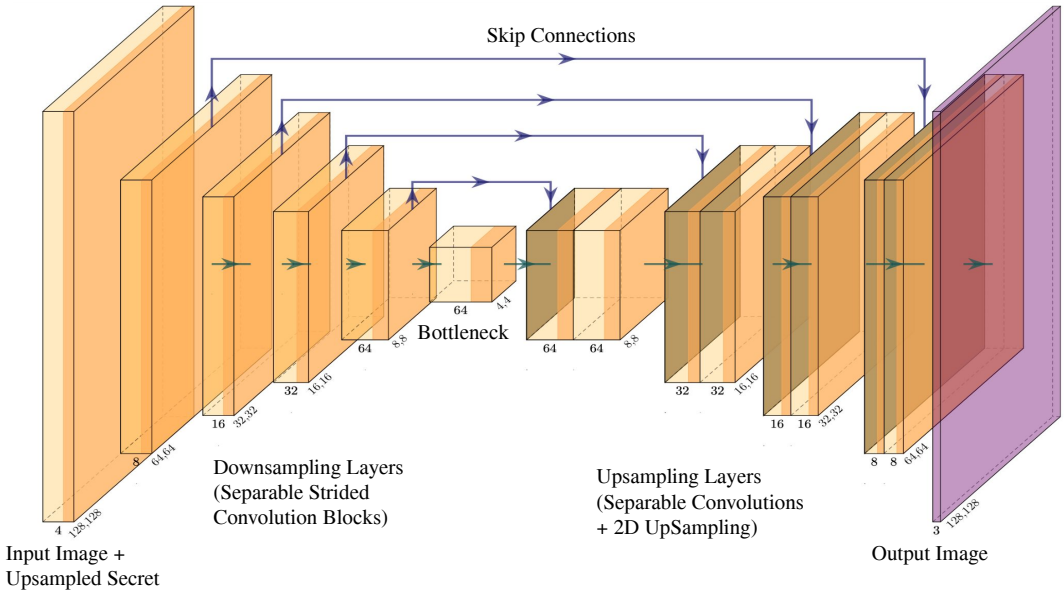


Figure 8.4. FastStamp Encoder Architecture. Our encoder network takes as input an image x and the output of the secret message upsampler $s_{\text{proj}M}$ and generates the watermarked image

U-Net Upsampling

The upsampling network in U-Net architecture follows a mirror image of the downsampling network. Instead of a regular convolutional layer, each upsampling block typically contains a transposed convolutional layer. However, transposed convolutional layers have been shown to introduce unwanted visual artifacts in the generated images [334] and we see the same effect empirically in our work. To remove such artifacts, we replace the transposed convolution layer with a separable convolution layer followed up by nearest neighbor 2D upsampling following the recommendations given by past work [334]. At each upsampling step, the output of the corresponding downsampling step is concatenated with the block input to provide skip-connections which are known to improve the performance of encoder-decoder models. The output of the last upsampling layer undergoes a tanh activation function to normalize output values between -1 and 1, which are then scaled between 0 and 1 to produce an RGB image. Figure 8.4 details our encoder network architecture.

FastStamp's Decoder follows a similar architecture as the encoder but contains 8 downsampling and 8 upsampling layers. After the upsampling U-net, the decoder network follows the inverse architecture of the secret message upsampler to predict the 128-bit message.

8.3 Accelerator Design

8.3.1 Design Overview

Figure 8.5 gives a high-level overview of our FPGA accelerator design. Each neural network layer is treated as a separate dataflow stage. To be low latency and high throughput, all weights and biases are stored on-chip. Complex activation functions are implemented via precomputed lookup tables. The design uses task-level pipelining (i.e., HLS dataflow) for each layer and streams the data between each dataflow stage using first-in-first-out buffers (FIFOs). As FIFOs can only be read once, to implement the skip connections, an additional dataflow stage is used to clone the skip connection data from its input FIFO into two other FIFOs so that it can

be read twice for its two datapaths.

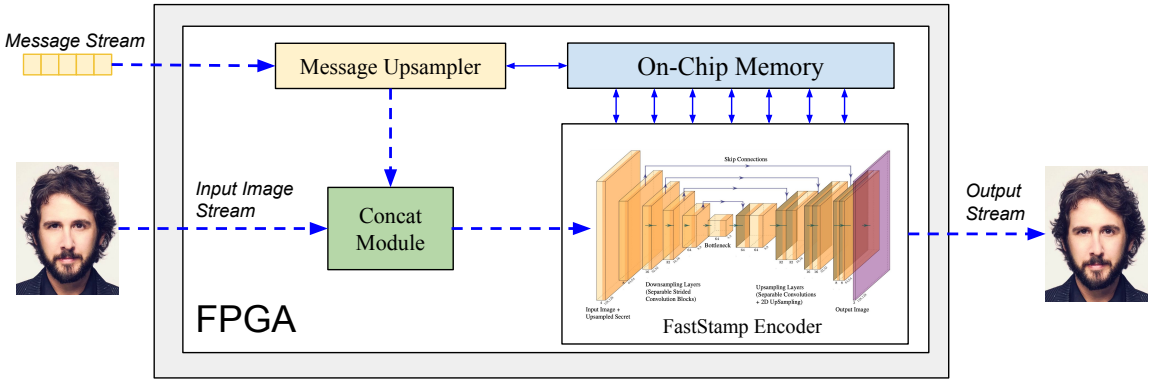


Figure 8.5. Design overview of FastStamp Accelerator Platform.

8.3.2 Implementation Details

To implement FastStamp on FPGA, we began with using the standard *hls4ml* [335, 336, 337] framework and added additional libraries to this to support the necessary functionalities such as depthwise separable convolutions, and nearest-neighbor 2D upsampling, concatenation layers (i.e., skip connections) which were previously not supported. We create custom sub-blocks and modify existing implementations with optimizations that better support our model. Our design is composed of the following 6 main architectural sub-blocks:

Linear Layer

The linear layer in the *Secret Message Upsampler* module is implemented using efficient matrix-vector multiplication based on product and sum tree. In order to optimize the design and make efficient use of the DSP blocks, we use a parallelized approach to convert layer computations into multiple MAC operations. Multiple rows of the weight matrix are processed simultaneously by dividing it into chunks. In each round, a chunk of the weights matrix is copied to one of the weight buffers while the other weight buffer is fed into the *dot product* modules together with a copy of the input vector. The iterations end when all rows of the weight matrix have been processed. Then each *dot-product* function partitions its input vectors into chunks and

concurrently executes MAC operations over the partitioned subsets. The accumulated results of the subsets are then added together within the *reduce_sum* function to compute the final output. The *reduce_sum* module performs a tree-based reduction algorithm which further reduces latency.

Nearest-neighbor 2D Upsampling

2D upsampling layers are used in the secret *Secret Message Upsampler* and the *U-Net Upsampling layers*. To upsample a matrix, we iterate through every element in the matrix and start building the new resized matrix. We use nearest neighbour as our interpolation method, as it is the least hardware intensive computation available for resizing purposes. We do not utilize pipelining in this sub-block, as it would cause a drastic increase in resource utilization for this resizing task while not having a substantial effect on latency. Due to these problems, pipelining makes this sub-block unscalable. Instead, loop unrolling is utilized to reduce loop iterations, giving us latency reductions with negligible impact on resource utilization.

Skip Connections

Concatenate layers are used in FastStamp to implement skip connections between upsampling and downsampling layers. These layers contributed towards some of the highest resource utilization in our design. To realize FastStamp on hardware, efficient support for concatenation of different sized inputs was implemented. We also distribute resource utilization to BRAM within these layers due to a large allocation of LUTs. Alongside this, we are able to minimize latency by pipelining the operations with complete unrolling in these layers.

Separable Convolution

Due to resource limitations, traditional convolution techniques in machine learning settings are often not feasible on hardware. Recent works [338, 339] have shown that utilizing separable convolutions significantly reduces the number of multiplications needed. Instead of traditional convolution, separable convolution layers are used in both U-Net downsampling and upsampling layers. A separable convolution comprises of a depthwise separable convolution

and point-wise convolution operation. The depthwise separable convolution is performed independently across the input channels and results in the same number of output channels. A point-wise convolution operation reduces to matrix-vector multiplication along the channel axis for each spatial cell of the input. We also use a streaming implementation of depthwise convolution utilizing pipelining with complete loop unrolling, array partitioning, and loop flattening to achieve minimal latency. We optimize standard point-wise convolution in a similar manner as our linear layer.

Batch-normalization

Batch-normalization [340] layers are used in U-Net downsampling and upsampling layers. These layers are used after the separable convolution layers to stabilize the network. Rather than using this optimization, pipelining, loop unrolling, and array partitioning are enforced to accelerate the normalization layer. This allows for the batch-normalization output to be used as an input to skip connections, a feature that cannot be done when fusing layers. Our optimizations achieve low latency and minimal resource utilization.

Non-linear activations

ReLU and tanh are the two non-linear activation functions are used in FastStamp. The ReLU function, which is used in the U-net downsampling and upsampling layers computes the following function for each input x, $y = \max(0,x)$. Due to the simplicity of the ReLU activation function, we use loop unrolling to optimize latency for this operation. The tanh activation function computes the following function for each input x, $y = (e^x - e^{-x})/(e^x + e^{-x})$. We use the default tanh implementation in *hls4ml* which is performed using a pre-computed lookup table to reduce latency.

Table 8.1. Capacity, imperceptibility, and BRA metrics of different watermarking systems for images of size $H \times W$. High BRA is desirable for benign transforms in both robust and semi-fragile systems. In semi-fragile systems, a low BRA is desirable for tampering transforms.

	Model Size	*Capacity*			*Imperceptibility*		*BRA (%) – Benign*			*BRA (%) – Tampering*
Method	# Params	H,W	L	BPP	PSNR	SSIM	None	JPG-75	Filtering	FaceSwap
DCT (Semi-Fragile) [300]	—	128	256	5.2×10^{-3}	22.49	0.871	99.81	56.65	94.62	85.51
HiDDeN (Robust) [308]	411 k	128	30	6.1×10^{-4}	27.57	0.934	97.06	72.71	94.52	—
StegaStamp (Robust) [318]	528 k	400	100	2.0×10^{-4}	29.39	0.925	99.92	99.91	99.84	—
FastStamp (Robust)	45 k	128	128	2.3×10^{-3}	30.65	0.942	100.00	99.84	99.78	—
FastStamp (Semi-Fragile)	45 k	128	128	2.3×10^{-3}	30.64	0.940	100.00	99.74	99.72	51.11

8.4 Experiments and Results

8.4.1 Dataset

We conduct experiments on the CelebA dataset [341] which is a large database of over 200,000 face images of 10,000 unique celebrities. We set aside 1000 images for testing the watermarking models and split the remaining data into 80% training and 20% validation. All FastStamp models are trained using images of size 128×128, which are obtained after center-cropping and resizing the CelebA images. We conduct experiments with message bit length $L = 128$.

8.4.2 Evaluation Metrics

For the evaluation of our watermarking techniques, we investigate the following metrics based on prior works [342, 326].

1. **Imperceptibility:** We compute **peak signal to noise ratio (PSNR)** and **structural similarity index (SSIM)** between the watermarked and original images. A higher value of both these metrics indicates a more imperceptible watermark.

2. **Capacity:** Capacity measures the amount of information that can be embedded in the image. We use **bits per pixel (BPP)** which is calculated as $L/(HWC)$ where L is the message length and H, W, C indicate the height, width, and channels of the image. Higher

BPP values indicate higher capacity.

3. **Bit-Recovery Accuracy (BRA):** BRA calculates the recovery accuracy of the bit string s. For robustness, we aim to have a high BRA when benign or intended transformations such as JPEG compression or color and contrast adjustments are applied. For semi-fragile watermarking systems, the goal is to have a low BRA when image tampering operations such as local tampering or FaceSwap are applied while maintaining robustness against benign transformations.

We compare our watermarking framework against three prior works on image watermarking: a DCT based semi-fragile watermarking system [301] and two robust neural image watermarking systems HiDDeN [308] and StegaStamp [318]. To evaluate the robustness and fragility of our watermarking systems, we perform the following transformations that are unseen during training:

1. **JPEG Compression:** Digital images are usually stored in a lossy format such as JPEG. We compress the watermarked images using JPEG-75 compression and measure the decoding BRA.

2. **Filtering:** We apply a set of real-world image filtering operations using the Pilgram library [343] that simulates photo editing filters that are common on social media. These include color, contrast, and lighting adjustments.

3. **Face Swapping:** For evaluating semi-fragile watermarking systems, we simulate image tampering by performing face swapping using the open source implementation of FaceSwap [249]. A low BRA is desirable against this transform to detect tampering.

8.4.3 Training and Architecture Optimization

Our encoder model follows the depthwise separable convolutional U-Net architecture as discussed in Section 8.2.3. To find the optimum architecture, we create different-sized versions

of the baseline U-Net architecture by reducing the number of channels in each layer by a factor of 2, 4 and 8. We find that reducing the number of channels by a factor of 4 does not compromise model performance while being significantly lighter than the base U-Net model.

We train two variants of this optimized design in the robust and semi-fragile settings using the training technique described in Section 8.2.1. In the robust setting, we simulate differentiable JPEG compression, Gaussian blur, color, and contrast adjustment as the benign transforms g_b during training. In the semi-fragile setting, in addition to the benign transforms, we simulate differentiable localized tampering as the malicious transform g_m during training. We train our models for 200 k mini-batch iteration with a fixed learning rate of 1.5×10^{-4} using Adam optimizer. Table 8.1 compares our optimized models *FastStamp (Robust)* and *FastStamp (Semi-Fragile)* against prior neural and DCT based image watermarking frameworks. As compared to prior neural image watermarking and steganography models, FastStamp is significantly smaller and achieves a similar BRA with slightly improved imperceptibility as compared to StegaStamp [318] and HiDDeN [308]. The improvement in imperceptible metrics is achieved by using nearest neighbor 2D upsampling in the U-Net architecture as opposed to transposed convolutions in prior work.

Table 8.2. Design-space exploration for FPGA implementation of FastStamp on Xilinx XCVU13P FPGA board. Our optimized 16-bit fixed point implementations fit within the available resources while maintaining the same correctness metrics as the 32-bit implementation.

	Resource Utilization (%)				*Performance*			*Correctness*		
Design *Available Resources*	BRAM 94 Mb	FF 3456K	LUT 1728K	DSP 12288	Clock Period (ns)	Latency (# Cycles)	Throughput (Hz)	BRA	PSNR	SSIM
FixedPoint-32	>100%	51%	>100%	>100%	—	—	—	100.0	30.67	0.942
FixedPoint-16	89%	18%	>100%	54%	—	—	—	100.0	30.64	0.941
FixedPoint-16-Optimized	59%	14%	72%	53%	5	596823	335	100.0	30.64	0.941

8.4.4 Design Space Exploration

We implement the individual submodules described in Section 8.3.2 using Vivado HLS for the Xilinx XCVU13P FPGA board. First, we perform a search over the bit-width of the fixed

point representation of the network weights and intermediate outputs. Our goal is to find the lowest bit-width that does not compromise on message recovery and imperceptibility. Figure 8.6 indicates the BRA and PSNR of different bit-width implementations of FastStamp. Based on this analysis, we use a 16-bit representation with 6 bits for the integer and 10 bits for the decimal representation. Next, we explore pipelining and loop unrolling options in our Vivado HLS

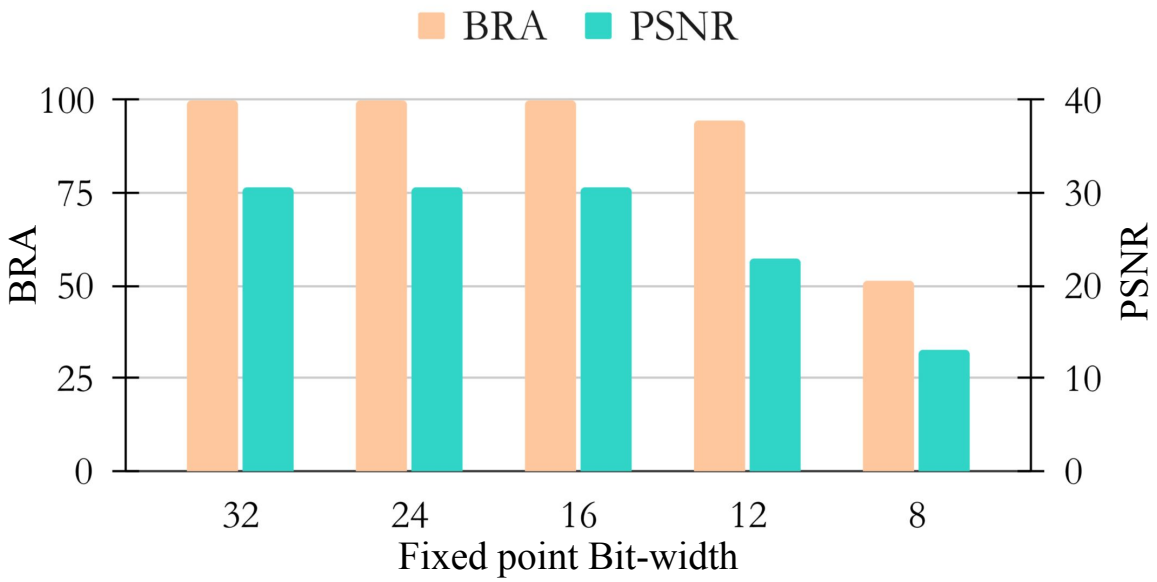


Figure 8.6. Watermarking success metrics for different fixed-point representations. A high value for both BRA and PSNR is desirable for accurate message recovery and imperceptibility.

implementation of various submodules. Complete loop unrolling in submodule implementations was an infeasible design choice for FastStamp since it exceeded the available resources on the device. We found that partial loop unrolling with factors of 8 and pipelining loops that were completely unrolled were the most effective optimizations for FastStamp.

While effective pipelining and loop unrolling resulted in a significant reduction in resource utilization, the LUT requirement of our design still exceeded the available resources on the device. Reducing the loop unrolling factor was an effective strategy to reduce LUT utilization but resulted in significant increases in encoding latency. To avoid latency increase, the next strategy we applied was distributing variables, such as model weights, that were initially all implemented as LUTs, to the BRAM on our board. We also force operations, such

as multiplication in the separable convolution into DSP blocks, which results in lower LUT utilization. We utilize per layer reuse factor to tune the inference latency versus utilization of FPGA resources and enable parallelization. This allows us to process multiple MAC operations at every unit of time. Using all of these optimizations, we are able to store our model entirely in the on-chip memory of the FPGA and perform low latency encoding while avoiding communication with off-chip memory. Table 8.2 lists the resource utilization, performance, and correctness of some of the design choices that led to our most optimized design, *FixedPoint-16-Optimized*. Figure 8.7 shows sample outputs of this optimized FPGA implementation and compares them with the GPU implementation in PyTorch.

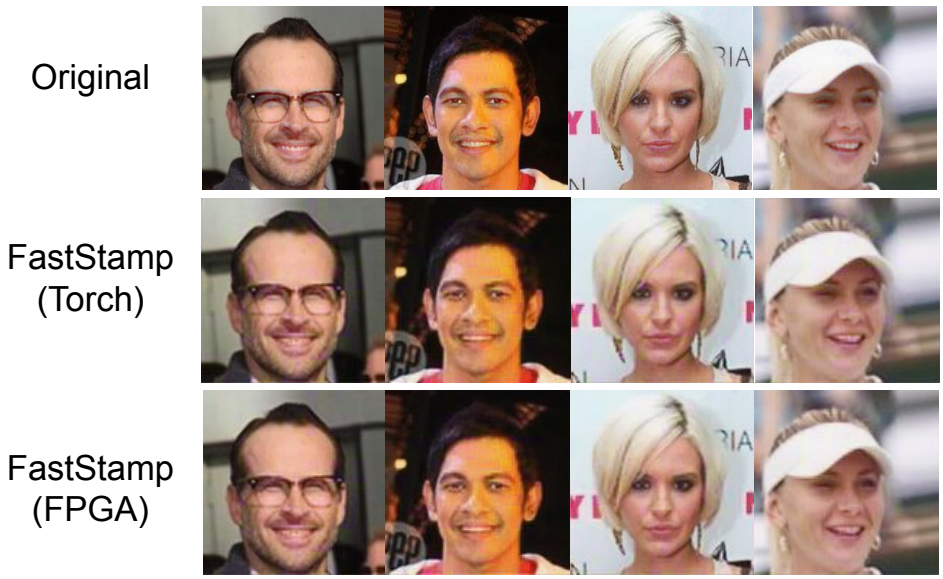


Figure 8.7. Sample image outputs of FastStamp optimized design and PyTorch implementation with the original image

8.4.5 Performance and Power Analysis

Table 8.3 compares the inference time and power requirement of our optimized FPGA implementation with the highly optimized CPU and GPU implementation of FastStamp and the open source implementations of prior neural watermarking systems. We benchmark the optimized PyTorch implementation of FastStamp on the Nvidia Tesla V100 GPU. The CPU

implementation is a NumPy inference program written by us and optimized fully. We measure the power consumption for the GPU benchmarks using the Nvidia power measurement tool (*Nvidia-smi*) running on *Linux* operating system, which is invoked during program execution. For our FPGA implementations, we synthesize our designs using Xilinx Vivado 2020.1. We then integrate the synthesized modules accompanied by the corresponding peripherals into a system-level schematic using the Vivado IP Integrator. The frequency is set to 200 MHz, and power consumption is estimated using the synthesis tool. Our FPGA implementation achieves $\sim$68$\times$ faster speed against prior work's GPU implementation and $\sim$10$\times$ faster speed against FastStamp's GPU implementation at a 3$\times$ lower power requirement.

Table 8.3. Power consumption and wall-clock time (in milliseconds) required to generate a single watermarked image per implementation.

Implementation	Time (ms)	Power (W)
StegaStamp GPU [318]	205	76
HiDDeN GPU [308]	234	65
FastStamp CPU	326	—
FastStamp GPU	30	59
FastStamp FPGA	3	19

8.5 Conclusion

In this chapter, I describe an efficient image watermarking model that can embed recoverable digital data into visual media. Our framework matches or even outperforms prior neural image watermarking and steganography models while utilizing significantly fewer parameters. By leveraging an efficient secret message upsampling module, depthwise separable convolutions, and 2-D upsampling, we were able to train a smaller model while preserving success metrics for watermarking tasks. Finally, we implement our encoder model on an FPGA to achieve 68$\times$ higher throughput as compared to prior GPU implementations. Our implementation allows watermark embedding directly at the hardware source, which not only secures the image capture

and transmission pipeline but also reduces latency in embedding the watermark. In the process of this implementation, we develop reconfigurable sub-modules which can accelerate convolutional downsampling and upsampling networks on hardware.

8.6 Acknowledgements

Chapter 8 is a reprint of the material as it appears in *FastStamp: Accelerating Neural Steganography and Digital Watermarking of Images on FPGAs*. IEEE/ACM International Conference on Computer-Aided Design, 2022. Hussain, Shehzeen; Sheybani, Nojan; Neekhara, Paarth; Zhang, Xinqiao; Duarte, Javier; Koushanfar, Farinaz. The dissertation author was the primary investigator and author of this paper.

Bibliography

[1] N. Carlini and D. Wagner, "Audio adversarial examples: Targeted attacks on speech-to-text," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018.

[2] P. Neekhara, S. Hussain, P. Pandey, S. Dubnov, J. McAuley, and F. Koushanfar, "Universal adversarial perturbations for speech recognition systems," in *Proc. Interspeech 2019*, 2019.

[3] Y. Qin, N. Carlini, G. Cottrell, I. Goodfellow, and C. Raffel, "Imperceptible, robust, and targeted adversarial examples for automatic speech recognition," in *International Conference on Machine Learning*, 2019.

[4] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio." in *SSW*, 2016, p. 125.

[5] S. Seferbekov, "https://github.com/selimsef/dfdc_deepfake-_challenge," 2020.

[6] A. Rossler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Niessner, "Faceforensics++: Learning to detect manipulated facial images," in *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, 2014.

[9] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016.

[10] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017.

[11] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *stat*, 2015.

[12] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[13] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (sp)*. IEEE, 2017, pp. 39–57.

[14] D. Song, K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, F. Tramèr, A. Prakash, and T. Kohno, "Physical adversarial examples for object detectors," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. USENIX Association, 2018.

[15] Y. Shi, S. Wang, and Y. Han, "Curls & whey: Boosting black-box adversarial attacks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[16] Y. Qin, N. Carlini, G. Cottrell, I. Goodfellow, and C. Raffel, "Imperceptible, robust, and targeted adversarial examples for automatic speech recognition," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 2019, pp. 5231–5240.

[17] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "Hotflip: White-box adversarial examples for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2018.

[18] Y. Belinkov and Y. Bisk, "Synthetic and natural noise both break neural machine translation," in *International Conference on Learning Representations*, 2018.

[19] P. Neekhara, S. Hussain, S. Dubnov, and F. Koushanfar, "Adversarial reprogramming of text classification neural networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, 2019.

[20] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, 2018.

[21] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.

[22] B. Ru, A. Cobb, A. Blaas, and Y. Gal, "Bayesopt adversarial attack," in *International Conference on Learning Representations*, 2019.

[23] W. Wu, Y. Su, X. Chen, S. Zhao, I. King, M. R. Lyu, and Y.-W. Tai, "Boosting the transferability of adversarial samples via attention," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1161–1170.

[24] Y. Lu, Y. Jia, J. Wang, B. Li, W. Chai, L. Carin, and S. Velipasalar, "Enhancing cross-task black-box transferability of adversarial examples with dispersion reduction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 940–949.

[25] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *Stat*, 2015.

[26] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *5th International Conference on Learning Representations, ICLR*, 2017.

[27] V. Chandrasekaran, C. Gao, B. Tang, K. Fawaz, S. Jha, and S. Banerjee, "Face-off: Adversarial face obfuscation," *Proceedings on Privacy Enhancing Technologies*, 2021.

[28] S. Shan, E. Wenger, J. Zhang, H. Li, H. Zheng, and B. Y. Zhao, "Fawkes: Protecting privacy against unauthorized deep learning models," in *29th USENIX Security Symposium*, 2020.

[29] "Fawkes press release," in *https://sandlab.cs.uchicago.edu/fawkes/press*.

[30] "The new york times," in *https://www.nytimes.com/2020/08/03/technology/fawkes-tool-protects-photos-from-facial-recognition.html*.

[31] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.

[32] S. Baluja and I. Fischer, "Learning to attack: Adversarial transformation networks," in *Proceedings of AAAI*, 2018.

[33] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *CVPR*, 2018.

[34] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*. Springer International Publishing, 2015.

[35] M. E. Kandel, Y. R. He, Y. J. Lee, T. H.-Y. Chen, K. M. Sullivan, O. Aydin, M. T. A. Saif, H. Kong, N. Sobh, and G. Popescu, "Phase imaging with computational specificity (pics) for measuring dry mass changes in sub-cellular compartments," *Nature communications*, 2020.

[36] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CVPR*, 2017.

[37] S. Hussain, T. Huster, C. Mesterharm, P. Neekhara, and F. Koushanfar, "Reface: Real-time adversarial attacks on face recognition systems," in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2023.

[38] S. Hussain, P. Neekhara, M. Jere, F. Koushanfar, and J. McAuley, "Adversarial deepfakes: Evaluating vulnerability of deepfake detectors to adversarial examples," in *WACV*, 2021.

[39] S. Hussain, P. Neekhara, B. Dolhansky, J. Bitton, C. Canton Ferrer, J. McAuley, and F. Koushanfar, "Exposing vulnerabilities of deepfake detection systems with robust attacks," *Digital Threats: Research and Practice (DTRAP)*, 2022.

[40] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[41] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "Sphereface: Deep hypersphere embedding for face recognition," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6738–6746, 2017.

[42] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[43] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, 2015.

[44] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019.

[45] S. Komkov and A. Petiushko, "Advhat: Real-world adversarial attack on arcface face id system," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021.

[46] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2016.

[47] ——, "A general framework for adversarial examples with objectives," *ACM Transactions on Privacy and Security (TOPS)*, 2019.

[48] A. Rajabi, R. B. Bobba, M. Rosulek, C. V. Wright, and W.-c. Feng, "On the (im) practicality of adversarial perturbation for image privacy," *Proceedings on Privacy Enhancing Technologies*, pp. 85–106, 2021.

[49] Y. Dong, H. Su, B. Wu, Z. Li, W. Liu, T. Zhang, and J. Zhu, "Efficient decision-based black-box adversarial attacks on face recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[50] A. Rozsa, M. Günther, and T. E. Boult, "Lots about attacking deep features," in *IEEE International Joint Conference on Biometrics (IJCB)*, 2017, pp. 168–176.

[51] Y. Xu, K. Raja, R. Ramachandra, and C. Busch, "Adversarial attacks on face recognition systems," in *Handbook of Digital Face Manipulation and Detection*. Springer, Cham, 2022.

[52] V. Cherepanova, M. Goldblum, H. Foley, S. Duan, J. P. Dickerson, G. Taylor, and T. Goldstein, "Lowkey: Leveraging adversarial attacks to protect social media users from facial recognition," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=hJmtwocEqzc

[53] J. Byun, H. Go, and C. Kim, "Geometrically adaptive dictionary attack on face-recognition," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022.

[54] A. Pinkus, "Approximation theory of the mlp model in neural networks," *Acta Numerica*, vol. 8, pp. 143 – 195, 1999.

[55] D. Abdelhafiz, S. Nabavi, R. Ammar, C. Yang, and J. Bi, "Residual deep learning system for mass segmentation and classification in mammography," *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, 2019.

[56] H. Li, D. Chen, B. Nailon, M. E. Davies, and D. Laurenson, "Improved breast mass segmentation in mammograms with conditional residual u-net," *ArXiv*, vol. abs/1808.08885, 2018.

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *ArXiv*, vol. abs/1603.05027, 2016.

[58] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI*, 2017.

[59] Q. Cao, L. Shen, W. Xie, O. Parkhi, and A. Zisserman, "Vggface2: A dataset for recognising faces across pose and age," *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, 2018.

[60] H.-W. Ng and S. Winkler, "A data-driven approach to cleaning large face datasets," in *IEEE international conference on image processing (ICIP)*. IEEE, 2014.

[61] "Umdfaces dataset," in *http://umdfaces.io/*.

[62] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[63] L. E. Baum and J. A. Eagon, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bull. Amer. Math. Soc.*, 1967.

[64] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *The annals of mathematical statistics*, 1970.

[65] A. Acero, l. Deng, T. Kristjansson, and J. Zhang, "Hmm adaptation using vector taylor series for noisy speech recognition," 2000.

[66] S. Ahadi and P. C. Woodland, "Combined bayesian and predictive techniques for rapid speaker adaptation of continuous density hidden markov models," *Computer speech & language*, 1997.

[67] L. Bahl, P. Brown, P. de Souza, and R. Mercer, "Maximum mutual information estimation of hidden markov model parameters for speech recognition," in *ICASSP'86. IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 1986.

[68] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[69] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, and G. Chen, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning, ICML*, 2016.

[70] A. van den Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. van den Driessche, E. Lockhart, L. Cobo, F. Stimberg, N. Casagrande, D. Grewe, S. Noury, S. Dieleman, E. Elsen, N. Kalchbrenner, H. Zen, A. Graves, H. King, T. Walters, D. Belov, and D. Hassabis, "Parallel WaveNet: Fast high-fidelity speech synthesis," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 3918–3926.

[71] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep speech: Scaling up end-to-end speech recognition," *CoRR*, vol. abs/1412.5567, 2014.

[72] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *CoRR*, vol. abs/1312.6199, 2013. [Online]. Available: http://arxiv.org/abs/1312.6199

[73] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1605.07277

[74] M. Alzantot, B. Balaji, and M. B. Srivastava, "Did you hear that? adversarial examples against automatic speech recognition," *CoRR*, vol. abs/1801.00554, 2018. [Online]. Available: http://arxiv.org/abs/1801.00554

[75] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou, "Hidden voice commands," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016.

[76] H. Yakura and J. Sakuma, "Robust audio adversarial example for a physical attack," *CoRR*, vol. abs/1810.11793, 2018. [Online]. Available: http://arxiv.org/abs/1810.11793

[77] D. Iter, J. Huang, and M. Jermann, "Generating adversarial examples for speech recognition," 2017.

[78] T. Vaidya, Y. Zhang, M. Sherr, and C. Shields, "Cocaine noodles: Exploiting the gap between human and machine speech recognition," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. Washington, D.C.: USENIX Association, 2015.

[79] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *ICML*. ACM, 2006.

[80] L. Schönherr, K. Kohls, S. Zeiler, T. Holz, and D. Kolossa, "Adversarial attacks against automatic speech recognition systems via psychoacoustic hiding," *arXiv preprint arXiv:1808.05665*, 2018.

[81] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE Trans. Pattern Anal. Mach. Intell.*, Jun. 2007.

[82] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.

[83] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018, pp. 99–112.

[84] "Project deepspeech," https://github.com/mozilla/DeepSpeech.

[85] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, CA, USA: California Technical Publishing, 1997.

[86] "Speech to text wavenet," https://github.com/buriburisuri/speech-to-text-wavenet.

[87] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. ACM, 2017.

[88] N. Papernot, P. D. McDaniel, A. Swami, and R. E. Harang, "Crafting adversarial input sequences for recurrent neural networks," *CoRR*, vol. abs/1604.08275, 2016. [Online]. Available: http://arxiv.org/abs/1604.08275

[89] W. Hu and Y. Tan, "Black-box attacks against rnn based malware detection algorithms," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[90] P. Yang, J. Chen, C. Hsieh, J. Wang, and M. I. Jordan, "Greedy attack and gumbel attack: Generating adversarial examples for discrete data," *CoRR*, vol. abs/1805.12316, 2018. [Online]. Available: http://arxiv.org/abs/1805.12316

[91] G. F. Elsayed, I. J. Goodfellow, and J. Sohl-Dickstein, "Adversarial reprogramming of neural networks," in *International Conference on Learning Representations, ICLR*, 2019.

[92] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[93] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax." *CoRR*, vol. abs/1611.01144, 2016. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1611.html#JangGP16

[94] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[95] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013.

[96] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: Transfer learning from unlabeled data," in *Proceedings of the Twenty-fourth International Conference on Machine Learning*, 2007.

[97] C. B. Do and A. Y. Ng, "Transfer learning for text classification," *Advances in neural information processing systems*, vol. 18, 2005.

[98] T. Semwal, G. Mathur, P. Yenigalla, and S. B. Nair, "A practitioners' guide to transfer learning for text classification using convolutional neural networks," *CoRR*, vol. abs/1801.06480, 2018. [Online]. Available: http://arxiv.org/abs/1801.06480

[99] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969033.2969173

[100] M. J. Kusner and J. M. Hernández-Lobato, "GANS for sequences of discrete elements with the gumbel-softmax distribution," *CoRR*, vol. abs/1611.04051, 2016.

[101] J. Gu, D. J. Im, and V. O. K. Li, "Neural machine translation with gumbel greedy decoding," *CoRR*, vol. abs/1706.07518, 2017.

[102] E. Gumbel, *Statistical theory of extreme values and some practical applications: a series of lectures*, ser. Applied mathematics series. U. S. Govt. Print. Office, 1954. [Online]. Available: https://books.google.com/books?id=SNpJAAAAMAAJ

[103] P. Bachman and D. Precup, "Data generation as sequential decision making," in *Advances in Neural Information Processing Systems*, 2015, pp. 3249–3257.

[104] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. C. Courville, and Y. Bengio, "An actor-critic algorithm for sequence prediction," *CoRR*, vol. abs/1607.07086, 2016.

[105] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient." *CoRR*, vol. abs/1609.05473, 2016.

[106] S. Robertson, "Classifying names with a character-level rnn - pytroch tutorial," https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html, 2017.

[107] X. Li and D. Roth, "Learning question classifiers," in *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, ser. COLING '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002.

[108] N. A. Abdulla, N. A. Ahmed, M. A. Shehab, and M. Al-Ayyoub, "Arabic sentiment analysis: Lexicon-based and corpus-based," in *2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, Dec 2013.

[109] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA, June 2011.

[110] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[111] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional lstm networks for improved phoneme classification and recognition," in *Proceedings of the 15th International Conference on Artificial Neural Networks: Formal Models and Their Applications - Volume Part II*, ser. ICANN'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 799–804.

[112] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014.

[113] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations, ICLR*, 2015.

[114] E. Kloberdanz, "Reprogramming of neural networks: A new and improved machine learning technique," *Masters Thesis*, 2020.

[115] Y.-Y. Tsai, P.-Y. Chen, and T.-Y. Ho, "Transfer learning without knowing: Reprogramming black-box machine learning models with scarce data and limited resources," in *ICML*, 2020.

[116] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2009.

[117] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[118] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.

[119] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "Fastspeech: Fast, robust and controllable text to speech," in *Neurips*, 2019.

[120] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations, ICLR*, 2021.

[121] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[122] J. Tu, M. Ren, S. Manivasagam, M. Liang, B. Yang, R. Du, F. Cheng, and R. Urtasun, "Physically realizable adversarial examples for lidar object detection," in *CVPR*, 2020.

[123] S. Hussain, P. Neekhara, S. Dubnov, J. McAuley, and F. Koushanfar, "Waveguard: Understanding and mitigating audio adversarial examples," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/hussain

[124] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015.

[125] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[126] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016.

[127] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, 2019, pp. 6105–6114.

[128] M. O. Noordewier, G. G. Towell, and J. W. Shavlik, "Training knowledge-based neural networks to recognize genes in dna sequences," in *NIPS*, 1990.

[129] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[130] D. Pokholok, C. Harbison, S. Levine, M. Cole, N. Hannett, T. Lee, G. Bell, K. Walker, P. Rolfe, E. Herbolsheimer, J. Zeitlinger, F. Lewitter, D. Gifford, and R. Young, "Genome-wide map of nucleosome acetylation and methylation in yeast," *Cell*, 2005.

[131] N. Ngoc Giang, V. Tran, D. Ngo, D. Phan, F. Lumbanraja, M. R. Faisal, B. Abapihi, M. Kubo, and K. Satou, "Dna sequence classification by convolutional neural network," *Journal of Biomedical Science and Engineering*, 2016.

[132] A. Matthews, J. Hron, M. Rowland, R. E. Turner, and Z. Ghahramani, "Gaussian process behaviour in wide deep neural networks," in *International Conference on Learning Representations, ICLR*, 2018.

[133] J. Lee, J. Sohl-dickstein, J. Pennington, R. Novak, S. Schoenholz, and Y. Bahri, "Deep neural networks as gaussian processes," in *International Conference on Learning Representations, ICLR*, 2018.

[134] L. R. Rabiner and R. W. Schafer, "Introduction to digital speech processing," *Foundations and Trends® in Signal Processing*, 2007.

[135] J. Shen, P. Nguyen, Y. Wu, Z. Chen, M. X. Chen, Y. Jia, A. Kannan, T. N. Sainath, and Y. Cao, "Lingvo: a modular and scalable framework for sequence-to-sequence modeling," *ArXiv*, vol. abs/1902.08295, 2019.

[136] X. Yuan, Y. Chen, Y. Zhao, Y. Long, X. Liu, K. Chen, S. Zhang, H. Huang, X. Wang, and C. A. Gunter, "Commandersong: A systematic approach for practical adversarial voice recognition," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018.

[137] Y. Chen, X. Yuan, J. Zhang, Y. Zhao, S. Zhang, K. Chen, and X. Wang, "Devil's whisper: A general approach for physical adversarial attacks against commercial black-box speech recognition devices," in *29th USENIX Security Symposium (USENIX Security 20)*. Boston, MA: USENIX Association, 2020.

[138] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[139] C. Guo, M. Rana, M. Cisse, and L. van der Maaten, "Countering adversarial images using input transformations," in *International Conference on Learning Representations, ICLR*, 2018.

[140] J. Lin, C. Gan, and S. Han, "Defensive quantization: When efficiency meets robustness," *Artificial Intelligence, Communication, Imaging, Navigation, Sensing Systems*, 2019.

[141] F. Khalid, H. Ali, H. Tariq, M. A. Hanif, S. Rehman, R. Ahmed, and M. Shafique, "Qusecnets: Quantization-based defense mechanism for securing deep neural network against adversarial attacks," in *25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2019.

[142] B. Liang, H. Li, M. Su, X. Li, W. Shi, and X. Wang, "Detecting adversarial image examples in deep neural networks with adaptive noise reduction," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[143] K. Rajaratnam, K. Shah, and J. Kalita, "Isolated and ensemble audio preprocessing methods for detecting adversarial examples against automatic speech recognition," in *Conference on Computational Linguistics and Speech Processing (ROCLING)*, 2018.

[144] Z. Yang, P. Y. Chen, B. Li, and D. Song, "Characterizing audio adversarial examples using temporal dependency," in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[145] Y. Qin, N. Frosst, S. Sabour, C. Raffel, G. Cottrell, and G. Hinton, "Detecting and diagnosing adversarial images with class-conditional capsule reconstructions," in *International Conference on Learning Representations*, 2020.

[146] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille, "Mitigating adversarial effects through randomization," in *International Conference on Learning Representations*, 2018.

[147] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," *arXiv preprint arXiv:1902.06705*, 2019.

[148] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017.

[149] F. Tramer, N. Carlini, W. Brendel, and A. Madry, "On adaptive attacks to adversarial example defenses," 2020.

[150] C. Herley and P. C. Van Oorschot, "Sok: Science, security and the elusive goal of security as a scientific pursuit," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017.

[151] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[152] H. Kwon, H. Yoon, and K.-W. Park, "Poster: Detecting audio adversarial example through audio modification," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019.

[153] J. Lu, T. Issaranon, and D. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[154] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[155] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, and R. Skerrv-Ryan, "Natural TTS synthesis by conditioning WaveNet on mel spectrogram predictions," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[156] P. Neekhara, C. Donahue, M. Puckette, S. Dubnov, and J. McAuley, "Expediting tts synthesis with adversarial vocoding," *Proc. Interspeech 2019*, 2019.

[157] C. Miao, S. Liang, M. Chen, J. Ma, S. Wang, and J. Xiao, "Flow-tts: A non-autoregressive network for text to speech based on flow," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020.

[158] Bhadragiri Jagan Mohan and Ramesh Babu N., "Speech recognition using mfcc and dtw," in *2014 International Conference on Advances in Electrical Engineering (ICAEE)*, 2014.

[159] M. Ravanelli, T. Parcollet, and Y. Bengio, "The pytorch-kaldi speech recognition toolkit," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019.

[160] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The Journal of the Acoustical Society of America*, 1937.

[161] J. Le Roux, H. Kameoka, N. Ono, and S. Sagayama, "Fast signal reconstruction from magnitude STFT spectrogram based on spectrogram consistency," in Proc. *International Conference on Digital Audio Effects*, 2010.

[162] D. W. Griffin, Jae, S. Lim, and S. Member, "Signal estimation from modified short-time Fourier transform," *IEEE Trans. Acoustics, Speech and Sig. Proc*, 1984.

[163] Y. He, *TensorFlow implementation of Griffin-Lim algorithm*, 2017. [Online]. Available: https://github.com/candlewill/Griffin_lim

[164] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, and R. Skerrv-Ryan, "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4779–4783.

[165] A. Gibiansky, S. Arik, G. Diamos, J. Miller, K. Peng, W. Ping, J. Raiman, and Y. Zhou, "Deep voice 2: Multi-speaker neural text-to-speech," in *Advances in neural information processing systems*, 2017, pp. 2962–2970.

[166] K. Qian, Y. Zhang, S. Chang, X. Yang, D. A. F. Florêncio, and M. Hasegawa-Johnson, "Speech enhancement using bayesian wavenet," in *INTERSPEECH*, 2017.

[167] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, and K. Kavukcuoglu, "Neural machine translation in linear time," *CoRR*, vol. abs/1610.10099, 2016.

[168] Z. Yang, Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick, "Improved variational autoencoders for text modeling using dilated convolutions," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3881–3890.

[169] R. J. Williams and D. Zipser, "Backpropagation," Y. Chauvin and D. E. Rumelhart, Eds. L. Erlbaum Associates Inc., 1995, ch. Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity, pp. 433–486.

[170] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *CoRR*, vol. abs/1803.01271, 2018.

[171] H. Tachibana, K. Uenoyama, and S. Aihara, "Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4784–4788.

[172] T. L. Paine, P. Khorrami, S. Chang, Y. Zhang, P. Ramachandran, M. A. Hasegawa-Johnson, and T. S. Huang, "Fast wavenet generation algorithm," *arXiv preprint arXiv:1611.09482*, 2016.

[173] M. Samragh, M. Ghasemzadeh, and F. Koushanfar, "Customizing neural networks for efficient fpga implementation," in *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*. IEEE, 2017, pp. 85–92.

[174] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 17.

[175] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung, "Fpga-based low-power speech recognition with recurrent neural networks," in *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2016, pp. 230–235.

[176] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. B. J. Dally, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. ACM, 2017, pp. 75–84.

[177] S. Li, C. Wu, H. Li, B. Li, Y. Wang, and Q. Qiu, "Fpga acceleration of recurrent neural network based language model," in *Proceedings of the 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '15. IEEE Computer Society, 2015, pp. 111–118.

[178] S. Hussain, M. Javaheripi, P. Neekhara, R. Kastner, and F. Koushanfar, "Fastwave: Accelerating autoregressive convolutional neural networks on fpga," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[179] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015.

[180] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, no. 11, 2015.

[181] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 16–25.

[182] C. Shea, A. Page, and T. Mohsenin, "Scalenet: A scalable low power accelerator for real-time embedded deep neural networks," in *Proceedings of Great Lakes Symposium on VLSI*. ACM, 2018, pp. 129–134.

[183] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "Fpga-based accelerator for long short-term memory recurrent neural networks," 2017.

[184] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-efficient cnn implementation on a deeply pipelined fpga cluster," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: Association for Computing Machinery, 2016.

[185] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded fpga platform for convolutional neural network," ser. FPGA '16. New York, NY, USA: Association for Computing Machinery, 2016.

[186] M. Samragh, M. Javaheripi, and F. Koushanfar, "Encodeep: Realizing bit-flexible encoding for deep neural networks," *ACM Trans. Embed. Comput. Syst.*, 2020.

[187] K. G., S. Z., J. Y., Y. Wang, and H. Y., "A survey of fpga based neural network accelerator," *ACM Transactions on Reconfigurable Technology and Systems*, 2017.

[188] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello, "Snowflake: An efficient hardware accelerator for convolutional neural networks," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.

[189] M. Carreras, G. Deriu, L. Raffo, L. Benini, and P. Meloni, "Optimizing temporal convolutional network inference on fpga-based accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, pp. 348–361, 2020.

[190] A. van den Oord, T. Walters, and T. Strohman, "Wavenet launches in the google assistant," 2017. [Online]. Available: https://deepmind.com/blog/wavenet-launches-google-assistant/

[191] A. Tamamori, T. Hayashi, K. Kobayashi, K. Takeda, and T. Toda, "Speaker-dependent wavenet vocoder," in *INTERSPEECH*, 2017.

[192] L.-J. Liu, Z.-H. Ling, Y. Jiang, M. Zhou, and L.-R. Dai, "Wavenet vocoder with limited training data for voice conversion," in *Proc. Interspeech*, 2018, pp. 1983–1987.

[193] G. Saon, G. Kurata, T. Sercu, K. Audhkhasi, S. Thomas, D. Dimitriadis, X. Cui, B. Ramabhadran, M. Picheny, L.-L. Lim, B. Roomi, and P. Hall, "English conversational telephone speech recognition by humans and machines," in *Proc. Interspeech 2017*, 2017, pp. 132–136.

[194] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[195] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[196] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1984–1992.

[197] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, and S. Song, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 26–35.

[198] L. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*. Prentice-Hall, Inc., 1993.

[199] D. Saito, K. Yamamoto, N. Minematsu, and K. Hirose, "One-to-many voice conversion based on tensor representation of speaker space," in *Interspeech*, 2011.

[200] S. H. Mohammadi and A. Kain, "An overview of voice conversion systems," in *Speech Communication*. Elsevier, 2017.

[201] J. Chou and H. Y. Lee, "One-shot voice conversion by separating speaker and content representations with instance normalization," *Interspeech*, 2019.

[202] K. Qian, Y. Zhang, S. Chang, X. Yang, and M. Hasegawa-Johnson, "Autovc: Zero-shot voice style transfer with only autoencoder loss," in *ICML*. PMLR, 2019.

[203] H. S. Choi, J. Lee, W. Kim, J. Lee, H. Heo, and K. Lee, "Neural analysis and synthesis: Reconstructing speech from self-supervised representations," *NeurIPS*, 2021.

[204] E. Casanova, J. Weber, C. Shulby, A. Junior, E. Gölge, and M. A. Ponti, "Yourtts: Towards zero-shot multi-speaker tts and zero-shot voice conversion for everyone," in *ICML*. PMLR, 2022.

[205] S. Arik, J. Chen, K. Peng, W. Ping, and Y. Zhou, "Neural voice cloning with a few samples," in *NeurIPS*, 2018. [Online]. Available: http://papers.nips.cc/paper/8206-neural-voice-cloning-with-a-few-samples.pdf

[206] Y. Jia, Y. Zhang, R. Weiss, Q. Wang, J. Shen, F. Ren, z. Chen, P. Nguyen, R. Pang, I. Lopez Moreno, and Y. Wu, "Transfer learning from speaker verification to multispeaker text-to-speech synthesis," in *NeurIPS*, 2018.

[207] Y. Wang, D. Stanton, Y. Zhang, R. Skerry-Ryan, E. Battenberg, J. Shor, Y. Xiao, F. Ren, Y. Jia, and R. A. Saurous, "Style tokens: Unsupervised style modeling, control and transfer in end-to-end speech synthesis," *arXiv:1803.09017*, 2018. [Online]. Available: https://arxiv.org/abs/1803.09017

[208] A. De Cheveigné and H. Kawahara, "Yin, a fundamental frequency estimator for speech and music," in *The Journal of the Acoustical Society of America*, 2002.

[209] L. Wan, Q. Wang, A. Papir, and I. L. Moreno, "Generalized end-to-end loss for speaker verification," *arXiv:1710.10467*, 2017.

[210] G. Louppe, *Resemblyzer - https://github.com/resemble-ai/Resemblyzer/*, 2019. [Online]. Available: https://github.com/resemble-ai/Resemblyzer/

[211] R. Valle, J. Li, R. Prenger, and B. Catanzaro, "Mellotron: Multispeaker expressive voice synthesis by conditioning on rhythm, pitch and global style tokens," *ICASSP*, 2020.

[212] R. Prenger, R. Valle, and B. Catanzaro, "WaveGlow: A flow-based generative network for speech synthesis," in *ICASSP*, 2018.

[213] H. Zen, V. Dang, R. Clark, Y. Zhang, R. J. Weiss, Y. Jia, Z. Chen, and Y. Wu, "LibriTTS: A Corpus Derived from LibriSpeech for Text-to-Speech," in *INTERSPEECH*, 2019. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2019-2441

[214] G. Louppe, "Master thesis : Automatic multispeaker voice cloning," 2019.

[215] A. Nagrani, J. S. Chung, W. Xie, and A. Zisserman, "Voxceleb: Large-scale speaker verification in the wild," *Computer Science and Language*, 2019.

[216] J. S. Chung, A. Nagrani, and A. Zisserman, "Voxceleb2: Deep speaker recognition," in *INTERSPEECH*, 2018.

[217] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *ICASSP*. IEEE, 2015.

[218] K. Ito, "The lj speech dataset," https://keithito.com/LJ-Speech-Dataset/, 2017.

[219] S. J. King and V. Karaiskos, "The blizzard challenge 2013," in *In Blizzard Challenge Workshop*, 2013.

[220] T. Nakatani, S. Amano, T. Irino, K. Ishizuka, and T. Kondo, "A method for fundamental frequency estimation and voicing decision: Application to infant utterances recorded in real acoustical environments," *Speech Communication*, 2008.

[221] W. Chu and A. Alwan, "Reducing f0 frame error of f0 tracking algorithms under noisy conditions with an unvoiced/voiced classification frontend," in *ICASSP*. IEEE, 2009.

[222] K. Lakhotia, E. Kharitonov, W. Hsu, Y. Adi, A. Polyak, B. Bolte, T. Nguyen, J. Copet, A. Baevski, and A. Mohamed, "On generative spoken language modeling from raw audio," *Transactions of the Association for Computational Linguistics*, 2021.

[223] A. Polyak, Y. Adi, J. Copet, E. Kharitonov, K. Lakhotia, W. Hsu, A. Mohamed, and E. Dupoux, "Speech resynthesis from discrete disentangled self-supervised representations," in *Interspeech*, 2021.

[224] Y. Lin, C. M. Chien, J. H. Lin, H. Lee, and L. S. Lee, "Fragmentvc: Any-to-any voice conversion by end-to-end extracting and fusing fine-grained voice fragments with attention," in *ICASSP*. IEEE, 2021.

[225] W. C. Huang, S. W. Yang, T. Hayashi, H. Y. Lee, S. Watanabe, and T. Toda, "S3prl-vc: Open-source voice conversion framework with self-supervised speech representations," in *ICASSP*. IEEE, 2022.

[226] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *NeurIPS*, 2020.

[227] A. Gulati, J. Qin, C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, and Y. Wu, "Conformer: Convolution-augmented transformer for speech recognition," *Interspeech*, 2020.

[228] S. Hussain, P. Neekhara, J. Huang, J. Li, and B. Ginsburg, "Ace-vc: Adaptive and controllable voice conversion using explicitly disentangled self-supervised speech representations," in *ICASSP*. IEEE, 2023.

[229] S. Hussain, V. Nguyen, S. Zhang, and E. Visser, "Multi-task voice activated framework using self-supervised learning," in *ICASSP*. IEEE, 2022.

[230] Y. Gu, Z. Zhang, X. Yi, and X. Zhao, "Mediumvc: Any-to-any voice conversion using synthetic specific-speaker speeches as intermedium features," *arXiv:2110.02500*, 2021.

[231] K. Qian, Y. Zhang, H. Gao, J. Ni, C. Lai, D. Cox, M. Hasegawa-Johnson, and S. Chang, "Contentvec: An improved self-supervised speech representation by disentangling speakers," in *International Conference on Machine Learning*. PMLR, 2022.

[232] H.-S. Choi, J. Yang, J. Lee, and H. Kim, "NANSY++: Unified voice synthesis with neural analysis and synthesis," in *International Conference on Learning Representations, ICLR*, 2023.

[233] L. Sun, S. Kang, K. Li, and H. Meng, "Voice conversion using deep bidirectional long short-term memory based recurrent neural networks," in *ICASSP*, 2015.

[234] L.-H. Chen, Z.-H. Ling, L.-J. Liu, and L.-R. Dai, "Voice conversion using deep neural networks with layer-wise generative training," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 12, pp. 1859–1872, 2014.

[235] L. Sun, K. Li, H. Wang, S. Kang, and H. Meng, "Phonetic posteriorgrams for many-to-one voice conversion without parallel data training," in *ICME*, 2016.

[236] X. Tian, J. Wang, H. Xu, E. S. Chng, and H. Li, "Average modeling approach to voice conversion with non-parallel data." in *Odyssey*, 2018.

[237] N. R. Koluguri, T. Park, and B. Ginsburg, "Titanet: Neural model for speaker representation with 1d depth-wise separable convolutions and global context," in *ICASSP*, 2022.

[238] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "Sphereface: Deep hypersphere embedding for face recognition," in *CVPR*. IEEE, 2017.

[239] J. Kong, J. Kim, and J. Bae, "Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis," *NeurIPS*, 2020.

[240] A. Łańcucki, "Fastpitch: Parallel text-to-speech with pitch prediction," in *ICASSP*. IEEE, 2021.

[241] J. Kahn, M. Rivière, W. Zheng, E. Kharitonov, Q. Xu, P. E. Mazaré, J. Karadayi, V. Liptchinsky, R. Collobert, C. Fuegen, T. Likhomanenko, G. Synnaeve, A. Joulin, A. Mohamed, and E. Dupoux, "Libri-light: A benchmark for asr with limited or no supervision," in *ICASSP*, 2020.

[242] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations, ICLR*, 2019.

[243] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang, "Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions," in *ICASSP*. IEEE, 2020.

[244] N. Koluguri, J. Li, V. Lavrukhin, and B. Ginsburg, "Speakernet: 1d depth-wise separable convolutional network for text-independent speaker recognition and verification," *arXiv:2010.12653*, 2020.

[245] J. Yamagishi, C. Veaux, and K. MacDonald, "CSTR VCTK Corpus: English multi-speaker corpus for CSTR voice cloning toolkit (version 0.92)," 2019.

[246] K. Park and T. Mulc, "Css10: A collection of single speaker speech datasets for 10 languages," *Interspeech*, 2019.

[247] J. Thies, M. Zollhofer, M. Stamminger, C. Theobalt, and M. Nießner, "Face2face: Real-time face capture and reenactment of rgb videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[248] J. Thies, M. Zollhöfer, and M. Nießner, "Deferred Neural Rendering: Image Synthesis using Neural Textures," *ACM Transactions on Graphics*, vol. 38, no. 66, pp. 1–12, 2019.

[249] FaceSwap, 2018.

[250] S. Suwajanakorn, S. Seitz, and I. Kemelmacher-Shlizerman, "Synthesizing Obama: Learning Lip Sync From Audio," *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1–13, 2017.

[251] B. Guo, Y. Ding, L. Yao, Y. Liang, and Z. Yu, "The future of false information detection on social media: New perspectives and trends," *ACM Comput. Surv.*, vol. 53, no. 4, 2020.

[252] Z. Jin, J. Cao, H. Guo, Y. Zhang, Y. Wang, and J. Luo, "Detection and analysis of 2016 us presidential election related rumors on twitter," in *SBP-BRiMS*, 2017.

[253] C. News, "*Doctored Nancy Pelosi video highlights threat of "deepfake" tech*," 2019. [Online]. Available: https://www.cbsnews.com/news/doctored-nancy-pelosi-video-highlights-threat-of-deepfake-tech-2019-05-25/

[254] C. Vaccari and A. Chadwick, "Deepfakes and disinformation: exploring the impact of synthetic political video on deception, uncertainty, and trust in news," *Social Media+ Society*, 2020.

[255] L. Verdoliva, "Media Forensics and DeepFakes: an Overview," *arXiv preprint arXiv:2001.06564*, 2020.

[256] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen, "Mesonet: a compact facial video forgery detection network," in *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, 2018.

[257] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[258] C. Hao, "https://github.com/cuihaoleo/kaggle-dfdc," 2020.

[259] A. Davletshin, "https://github.com/ntech-lab/deepfake-detection-challenge," 2020.

[260] DeepFakes, "https://github.com/deepfakes/faceswap," 2017.

[261] B. Dolhansky, R. Howes, B. Pflaum, N. Baram, and C. C. Ferrer, "The deepfake detection challenge (dfdc) dataset," *arXiv preprint arXiv:2006.07397*, 2020.

[262] E. Zakharov, A. Shysheya, E. Burkov, and V. Lempitsky, "Few-shot adversarial learning of realistic neural talking head models," in *International Conference on Computer Vision (ICCV)*, 2019, pp. 9459–9468.

[263] Y. Nirkin, Y. Keller, and T. Hassner, "FSGAN: Subject agnostic face swapping and reenactment," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 7184–7193.

[264] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4401–4410.

[265] W. Wang and H. Farid, "Exposing digital forgeries in interlaced and deinterlaced video," *IEEE Transactions on Information Forensics and Security*, vol. 2, no. 3, pp. 438–449, 2007.

[266] R. Bohme and M. Kirchner, "Digital image forensics: There is more to a picture than meets the eye, chapter counter-forensics: Attacking image forensics," 2013.

[267] H. Farid, *Photo Forensics*. The MIT Press, 2016.

[268] M. Barni, M. C. Stamm, and B. Tondi, "Adversarial multimedia forensics: Overview and challenges ahead," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 962–966.

[269] R. Böhme and M. Kirchner, "Counter-forensics: Attacking image forensics," in *Digital image forensics*. Springer, 2013, pp. 327–366.

[270] P. Zhou, X. Han, V. I. Morariu, and L. S. Davis, "Two-stream neural networks for tampered face detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2017, pp. 1831–1839.

[271] R. Raghavendra, K. B. Raja, S. Venkatesh, and C. Busch, "Transferable deep-cnn features for detecting digital and print-scanned morphed face images," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2017, pp. 1822–1830.

[272] J. H. Bappy, A. K. Roy-Chowdhury, J. Bunk, L. Nataraj, and B. Manjunath, "Exploiting spatial structure for localizing manipulated image regions," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 4970–4979.

[273] J. H. Bappy, C. Simons, L. Nataraj, B. Manjunath, and A. K. Roy-Chowdhury, "Hybrid lstm and encoder–decoder architecture for detection of image forgeries," *IEEE Transactions on Image Processing*, vol. 28, no. 7, pp. 3286–3300, 2019.

[274] L. Li, J. Bao, T. Zhang, H. Yang, D. Chen, F. Wen, and B. Guo, "Face x-ray for more general face forgery detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5001–5010.

[275] Y. Li, M.-C. Chang, and S. Lyu, "In ictu oculi: Exposing ai created fake videos by detecting eye blinking," in *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE, 2018, pp. 1–7.

[276] D. Güera and E. J. Delp, "Deepfake video detection using recurrent neural networks," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2018, pp. 1–6.

[277] E. Sabir, J. Cheng, A. Jaiswal, W. AbdAlmageed, I. Masi, and P. Natarajan, "Recurrent convolutional strategies for face manipulation detection in videos," *Interfaces (GUI)*, vol. 3, p. 1, 2019.

[278] X. Yang, Y. Li, and S. Lyu, "Exposing deep fakes using inconsistent head poses," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 8261–8265.

[279] K. Vougioukas, S. Petridis, and M. Pantic, "Realistic speech-driven facial animation with gans," *International Journal of Computer Vision*, pp. 1–16, 2019.

[280] A. Duarte, F. Roldan, M. Tubau, J. Escur, S. Pascual, A. Salvador, E. Mohedano, K. McGuinness, J. Torres, and X. Giro-i Nieto, "Wav2pix: speech-conditioned face generation using generative adversarial networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 3, 2019.

[281] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 582–597.

[282] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, "A study of the effect of jpg compression on adversarial images," *arXiv preprint arXiv:1608.00853*, 2016.

[283] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, L. Chen, M. E. Kounavis, and D. H. Chau, "Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression," *arXiv preprint arXiv:1705.02900*, 2017.

[284] C. Guo, M. Rana, M. Cisse, and L. Van Der Maaten, "Countering adversarial images using input transformations," *arXiv preprint arXiv:1711.00117*, 2017.

[285] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9185–9193.

[286] Y. Dong, T. Pang, H. Su, and J. Zhu, "Evading defenses to transferable adversarial examples by translation-invariant attacks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[287] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[288] C. Xie, Z. Zhang, J. Wang, Y. Zhou, Z. Ren, and A. Yuille, "Improving transferability of adversarial examples with input diversity," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2725–2734, 2019.

[289] W. Zhou, X. Hou, Y. Chen, M. Tang, X. Huang, X. Gan, and Y. Yang, "Transferable adversarial perturbations," in *ECCV*, 2018.

[290] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 949–980, Jan. 2014.

[291] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," in *International Conference on Machine Learning*, 2018, pp. 2137–2146.

[292] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," Sep. 2017. [Online]. Available: http://arxiv.org/abs/1703.03864

[293] M. Behjati, S.-M. Moosavi-Dezfooli, M. S. Baghshah, and P. Frossard, "Universal adversarial attacks on text classifiers," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 7345–7349.

[294] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[295] J. Deng, J. Guo, E. Ververas, I. Kotsia, and S. Zafeiriou, "RetinaFace: Single-shot multi-level face localisation in the wild," in *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[296] J. Li, Y. Wang, C. Wang, Y. Tai, J. Qian, J. Yang, C. Wang, J. Li, and F. Huang, "DSFD: dual shot face detector," in *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[297] Y. Mirsky and W. Lee, "The creation and detection of deepfakes: A survey," *ACM Comput. Surv.*, vol. 54, no. 1, jan 2021. [Online]. Available: https://doi.org/10.1145/3425780

[298] B. Dolhansky, J. Bitton, B. Pflaum, J. Lu, R. Howes, M. Wang, and C. C. Ferrer, "The DeepFake Detection Challenge (DFDC) dataset," *arXiv preprint arXiv:2006.07397*, 2020.

[299] E. T. Lin, C. I. Podilchuk, and E. J. Delp III, "Detection of image alterations using semifragile watermarks," in *Security and Watermarking of Multimedia Contents II*. International Society for Optics and Photonics, 2000.

[300] C. K. Ho and C.-T. Li, "Semi-fragile watermarking scheme for authentication of jpeg images," in *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.* IEEE, 2004.

[301] H. Yang, X. Sun, and G. Sun, "A semi-fragile watermarking algorithm using adaptive least significant bit substitution," *Information Technology Journal*, 2009.

[302] C. Li, A. Zhang, Z. Liu, L. Liao, and D. Huang, "Semi-fragile self-recoverable watermarking algorithm based on wavelet group quantization and double authentication," *Multimedia tools and applications*, 2015.

[303] C. Gu, N. Hanley, and M. O'neill, "Improved reliability of fpga-based puf identification generator design," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2017.

[304] S. Hussain, N. Sheybani, P. Neekhara, X. Zhang, J. Duarte, and F. Koushanfar, "Faststamp: Accelerating neural steganography and digital watermarking of images on fpgas," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022.

[305] F. Di Martino and S. Sessa, "Fragile watermarking tamper detection via bilinear fuzzy relation equations," *Journal of Ambient Intelligence and Humanized Computing*, 2019.

[306] I. J. Cox, J. Kilian, F. T. Leighton, and T. Shamoon, "Secure spread spectrum watermarking for multimedia," *IEEE transactions on image processing*, 1997.

[307] A. Shehab, M. Elhoseny, K. Muhammad, A. K. Sangaiah, P. Yang, H. Huang, and G. Hou, "Secure and robust fragile watermarking scheme for medical images," *IEEE Access*, 2018.

[308] J. Zhu, R. Kaplan, J. Johnson, and L. Fei-Fei, "Hidden: Hiding data with deep networks," in *ECCV*, 2018.

[309] X. Yu, C. Wang, and X. Zhou, "Review on semi-fragile watermarking algorithms for content authentication of digital images," *Future Internet*, 2017.

[310] J. Xiao and Y. Wang, "A semi-fragile watermarking tolerant of laplacian sharpening," in *2008 International Conference on Computer Science and Software Engineering*. IEEE, 2008.

[311] R. Preda and D. Vizireanu, "Watermarking-based image authentication robust to jpeg compression," *Electronics Letters*, 2015.

[312] R. Tay and J. Havlicek, "Image watermarking using wavelets," in *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.* IEEE, 2002.

[313] O. Benrhouma, H. Hermassi, and S. Belghith, "Tamper detection and self-recovery scheme by dwt watermarking," *Nonlinear Dynamics*, 2015.

[314] S. Baluja, "Hiding images in plain sight: Deep steganography," *NIPS*, 2017.

[315] J. Hayes and G. Danezis, "Generating steganographic images via adversarial training," in *International Conference on Neural Information Processing Systems*, 2017.

[316] R. Zhang, S. Dong, and J. Liu, "Invisible steganography via generative adversarial networks," *Multimedia tools and applications*, 2019.

[317] R. Wang, F. Juefei-Xu, M. Luo, Y. Liu, and L. Wang, "Faketagger: Robust safeguards against deepfake dissemination via provenance tracking," in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, p. 3546.

[318] M. Tancik, B. Mildenhall, and R. Ng, "Stegastamp: Invisible hyperlinks in physical photographs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[319] P. Neekhara, S. Hussain, X. Zhang, K. Huang, J. McAuley, and F. Koushanfar, "Facesigns: Semi-fragile neural watermarks for media authentication and countering deepfakes," 2022.

[320] X. Luo, R. Zhan, H. Chang, F. Yang, and P. Milanfar, "Distortion agnostic deep watermarking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[321] S. Liu, H. Fan, X. Niu, H.-c. Ng, Y. Chu, and W. Luk, "Optimizing cnn-based segmentation with deeply customized convolutional and deconvolutional architectures on fpga," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2018.

[322] W. Zhao, Z. Jia, X. Wei, and H. Wang, "An fpga implementation of a convolutional auto-encoder," *Applied Sciences*, 2018.

[323] E. Govorkova *et al.*, "Autoencoders on field-programmable gate arrays for real-time, unsupervised new physics detection at 40 mhz at the large hadron collider," *Nat. Mach. Intell.*, vol. 4, p. 154, 2022.

[324] S. Kiran, K. N. Sri, and J. Jaya, "Design and implementation of FPGA based invisible image watermarking encoder using wavelet transformation," in *2013 International Conference on Current Trends in Engineering and Technology (ICCTET)*. IEEE, 2013.

[325] S. Hazra, S. Ghosh, S. De, and H. Rahaman, "Fpga implementation of semi-fragile reversible watermarking by histogram bin shifting in real time," *Journal of Real-Time Image Processing*, 2018.

[326] M. A. Hajjaji, M. Gafsi, A. Ben Abdelali, and A. Mtibaa, "Fpga implementation of digital images watermarking system based on discrete haar wavelet transform," *Security and Communication Networks*, 2019.

[327] R. M. Khoshki, S. Oweis, S. Wang, G. Pappas, and S. Ganesan, "Fpga hardware based implementation of an image watermarking system," *Int. J. Adv. Res. Comput.*, 2014.

[328] M. Liu, Y. Ding, M. Xia, X. Liu, E. Ding, W. Zuo, and S. Wen, "Stgan: A unified selective transfer network for arbitrary image attribute editing," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[329] Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha, "Stargan v2: Diverse image synthesis for multiple domains," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[330] S. Hussain, P. Neekhara, M. Jere, F. Koushanfar, and J. McAuley, "Adversarial deepfakes: Evaluating vulnerability of deepfake detectors to adversarial examples," in *WACV*, 2021.

[331] A. Qureshi, D. Megías, and M. Kuribayashi, "Detecting deepfake videos using digital watermarking," in *2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2021.

[332] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *CVPR*, 2018.

[333] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017, p. 1800.

[334] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, 2016. [Online]. Available: http://distill.pub/2016/deconv-checkerboard

[335] J. Duarte *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *JINST*, vol. 13, no. 07, p. P07027, 2018.

[336] T. Aarrestad *et al.*, "Fast convolutional neural networks on FPGAs with hls4ml," *MLST*, vol. 2, no. 4, p. 045015, 2021.

[337] F. Fahim *et al.*, "hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices," in *tinyML Research Symposium 2021*, 3 2021.

[338] L. Bai, Y. Zhao, and X. Huang, "A cnn accelerator on fpga using depthwise separable convolution," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2018.

[339] B. Yoo, Y. Choi, and H. Choi, "Fast depthwise separable convolution for embedded systems," in *Neural Information Processing*, L. Cheng, A. C. S. Leung, and S. Ozawa, Eds. Cham: Springer International Publishing, 2018.

[340] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds. PMLR, 2015.

[341] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *ICCV*, December 2015.

[342] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *2010 20th international conference on pattern recognition*. IEEE, 2010.

[343] A. Kamakura, "pilgram https://github.com/akiomik/pilgram."