Title
Visual Learning with Weak Supervision

Author
CICEK, BAYRAM SAFA

Publication Date
2021

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Visual Learning with Weak Supervision

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical and Computer Engineering

by

Bayram Safa Cicek

2021

ABSTRACT OF THE DISSERTATION

Visual Learning with Weak Supervision

by

Bayram Safa Cicek
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2021
Professor Stefano Soatto, Chair

We focus on two broad learning setups: The first one is the classic semi-supervised learning (SSL), wherein few labeled samples and many unlabeled samples are drawn from the same distribution. Second is a more challenging setting called unsupervised domain adaptation (UDA), where labeled and unlabeled samples are drawn from slightly different distributions. This setting is more practical as labeled data can be drawn from synthetic data, which can be produced abundantly using graph engines with the expense of domain shift from the real data.

Our contributions are multi-faceted. For the SSL setup, we show that the training speed in the supervised setting correlates strongly with the percentage of correct labels. Since the speed of the training is a measurable quantity at the training time unlike the accuracy of the estimates, we propose to use it as an inference criterion. We additionally show that robustness to small perturbations in input space and weight space do not imply each other and they both improve generalization performance. Then, we propose a method that combines an input-space smoothing with a weight-space smoothing.

In the UDA setup, we begin by proposing a method that trains a shared embedding to align the distributions of the learned features conditioned on the classes. To have more interpretable models, we also explore the use of generative modeling where we generate images in the unlabeled target domain in a manner that allows independent control of class and nuisance variability. We

also study UDA for dense prediction tasks like semantic segmentation where the manual annotation is more costly. Lastly, we examine monocular depth prediction tasks where the goal is to infer dense depth maps from images. Obtaining the 3D scene from a single image is a degenerate task as there are infinitely many 3D scenes compatible with the given image. So, we also leverage LiDAR measurements, which brings the additional challenges of processing sparse inputs. We propose to use only sparse depth as input, not images, so the method is not affected by the covariate shift and we use the image to refine the predicted depth map.

Finally, we study the effect of adversarial perturbations on the networks trained in the unsupervised fashion particularly for the task of monocular depth prediction. We explore the ability of small, imperceptible additive perturbations to selectively alter the perceived geometry of the scene. Our work helps to understand the corner cases and failure modes to develop more robust representations. This, in turn, will improve interpretability (or, rather, reduce nonsensical behavior).

The dissertation of Bayram Safa Cicek is approved.

Lieven Vandenberghe

Paulo Tabuada

Guy Van den Broeck

Stefano Soatto, Committee Chair

University of California, Los Angeles

2021

To my parents and my sister.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I would like to thank my advisor Professor Stefano Soatto, without his support I would not be able to complete this thesis. He has been patient with my struggles during my graduate studies and encouraged me to find my way when I felt lost several times. I joined the group with very limited background on any subject of computer science including computer vision and machine learning and learned almost anything I know on these subjects in UCLA Vision Lab. Stefano, knowing these, accepted me into his group and gave me a lifetime opportunity. He has been generous in sharing his knowledge, experience, and time. He taught me how a researcher should pick a problem and solve it. I learned from him that just like a composer, a researcher should learn everything there is to know on the literature and then should be able to forget what he knows before coming up with a solution. I hope I could partially follow his suggestions.

I would like to thank Alhussein Fawzi for introducing me the semi-supervised learning problem. He has been like a brother to me and selflessly supported me when writing my first paper. Without his help, this thesis would not be possible.

I would like to thank Pratik Chaudhari for guiding me in first years of graduate studies and sharing his knowledge in various problems from motion planning to machine learning. My thoughts on these subjects and many others are shaped by Pratik's wisdom.

I would like to thank Nikos Karianakis for being very supportive in the early years of my Ph.D. I met with him when he was teaching a machine learning class that I was taking as a student. He was nice enough to recommend me to Stefano and helped me in joining to UCLA Vision Lab.

I was lucky enough to meet and collaborate with Alex Wong in the last years of my Ph.D. I want to thank Alex for introducing me the depth-completion and monocular-depth prediction problems. Other than these 3D geometry problems, I learned a lot of good coding practices from Alex. He is a very helpful to people around him and I am grateful for getting to know him.

I would like to thank Xiaohan Fei for being a good friend and making noise in the lab. After Xiaohan graduated, we lost the last talkative person in the lab and I even missed his metallic keyboard noise.

During these years, I interned in high-quality research labs. One of these was Adobe Research where I collaborated with Ning Xu, Zhaowen Wang, and Hailin Jin. With them, I published my first work on generative models.

I also had a chance to work in Waymo with Alper Ayvaci, Vasiliy Karasev, Feiyu Chen, Justin Zheng. Thanks to them, I improved my collaborative coding skills.

I also would like to thank Ahmed Alaa, Muhammed Veli for being a brother and a friend.

I was lucky enough to meet with many other smart and nice people during my years at UCLA Vision Lab. I would like to acknowledge Alessandro Achille, Peng Zhao, Jingming Dong, Konstantine Tsotsos, Virginia Estellers, Brian Taylor, Simon Korman, Xinzhu Bei, Shay Deutsch, Aditya Golatkar, Tong He, Stephanie Tsuei, Alexandre Tiard, Yanchao Yang, Albert Zhao, Antonio Loquercio, Matteo Terzi. I thank them for their support and fruitful discussions.

Most importantly, I would like to thank my parents and sister for their endless and continuous support and prayers from the other side of the ocean. I dedicate this thesis to them.

VITA

2011–2015	Bachelor in Electrical Engineering, Bilkent University, Ankara, Turkey.
2015–2017	Master of Science in Electrical Engineering, University of California, Los Angeles.
2015–Present	Research Assistant, University of California, Los Angeles

PUBLICATIONS

S. Cicek, A. Fawzi, and S. Soatto. Saas: Speed as a supervisor for semi-supervised learning. In the Proceedings of European Conference on Computer Vision (ECCV). 2018.

S. Cicek, and S. Soatto. Input and Weight Space Smoothing for Semi-supervised Learning. In the Proceedings of IEEE International Conference on Computer Vision Workshops (ICCV). 2019.

S. Cicek, and S. Soatto. Unsupervised domain adaptation via regularized conditional alignment. In the Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2019.

S. Cicek, N. Xu, Z. Wang, H. Jin, S. Soatto. Generative Feature Disentangling for Unsupervised Domain Adaptation. In the Proceedings of European Conference on Computer Vision Workshops (ECCV). 2020.

S. Cicek, N. Xu, Z. Wang, H. Jin, S. Soatto. Spatial Class Distribution Shift in Unsupervised Domain Adaptation: Local Alignment Comes to Rescue. In the Proceedings of Asian Conference on Computer Vision (ACCV). 2020.

S. Cicek, A. Nakhaei, S. Soatto, K. Fujimura. MARL-PPS: Multi-agent Reinforcement Learning with Periodic Parameter Sharing. In International Conference On Autonomous Agents and Multi-Agent Systems (AAMAS). 2019.

A. Wong, **S. Cicek**, and S. Soatto. Learning Topology from Synthetic Data for Unsupervised Depth Completion. IEEE Robotics and Automation Letters (RAL). 2021.

A. Wong, **S. Cicek**, and S. Soatto. Targeted Adversarial Perturbations for Monocular Depth Prediction. In the Proceedings of Conference on Neural Information Processing Systems (NeurIPS). 2020.

CHAPTER 1

Introduction

For many engineering applications ranging from manipulation, navigation (e.g. autonomous driving), medical image analysis, entertainment (AR/VR) to security (e.g. surveillance), perception is the key building block where given the visual sensory, goal is to understand the environment by extracting and processing photometric, semantic, geometric and dynamic information as a representation most suitable for the task.

For the transportation (e.g. autonomous driving) example, the task can be defined as moving from source to destination without collision in the shortest possible time following the traffic rules.

Following the traffic rules requires one to process photometric information as the driver should be able to discriminate green, yellow and red colors in a traffic light. This is a trivial task given the visual sensory data. However, this color discrimination should be performed only when there is a traffic light in the environment and it should be performed only on the portion of the image projected from the traffic light in the scene. So, the perception system of the driver should be able to *classify* the traffic light in the environment that it is interacting with. Since only a small portion of the image would capture the traffic light, it also needs to *localize* the traffic light in the image space. This simple example alone shows that the driver should be able to semantically classify and localize objects so that it can first detect the traffic light and stop if the red light is on. Even better is to classify each pixel in an image (*segmentation*) to get finer level semantic map.

Of course, the use of *object detection* for the autonomous driving application is not limited to traffic light detection or to detection of other traffic signs. The most vital skill we expect from a driver is to not collide to another living being. So, being able to semantically discriminate humans from other objects in the scene is an essential skill. But, semantic information is not enough for

this motion planning task as the depth of the objects in the scene (i.e. how far they are from the ego-vehicle) are also needed. The driver should be able to localize objects in the 3D environment, to be able to slow down when it is getting closer to an object or more importantly to a pedestrian. If we are given a single camera as a sensor, depth of the scene has to be estimated with monocular frameworks (*monocular depth prediction*), which is a degenerate problem [MSK12]. If multiple views of the same scene are available, one can apply traditional methods to estimate the scene geometry under certain assumptions (e.g. Lambertian surface). Luckily, visual sensory is not limited to passive sensors like cameras and active sensors like LiDARs are commonly used. However, depth measurements from such devices are quite sparse – density of LiDAR measurements is typically $\approx 5\%$ of the entire image [WFT20] and they are concentrated on the lower half of the image space. So, the *depth completion* needs to be performed by leveraging the dense information in RGB images by applying sensor fusion techniques on multi-sensor data.

In the last decade, state of the art for these computer vision tasks have been improved significantly and for some of them, even the human performance is suppressed [RDS15, HZR15a]. [TVD20],[TPL20],[CZP18],[YLS19] achieved great results in Imagenet [DDS09] image classification task, COCO [LMB14] object detection task, CityScapes [COR16] semantic segmentation task and NYUD-V2 [SHK12] monocular depth prediction task respectively. However all these best performing models are trained in supervised fashion where an appropriate distance metric between network predictions and ground truths are minimized. For this, annotations for the variable to be estimated are provided during training in different forms such as class label per image, class label per pixel or depth corresponding to each pixel (range map).

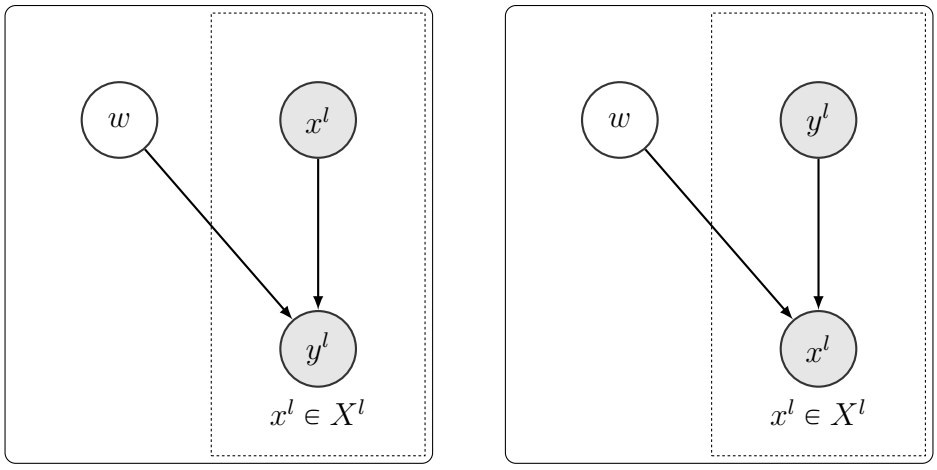
However, this approach is not scalable as annotations for these problems are often not available for all image datasets or when they are available, they are very expensive to acquire. For Imagenet [DDS09] with 3.2 million images in total, 10 Amazon Mechanical Turk (AMT) workers are asked to vote on each image. For COCO [LMB14] detection benchmark, 2.5 million instances in 328K images are labeled by AMT workers in three stages for category annotation, instance localization and segmentation annotation where total annotation takes around 60,000 worker hours. For segmentation benchmark CityScapes [COR16], annotation and quality control per a single

image requires more than 1.5 hours on average. [GLU12] provides semi-dense $\approx 50\%$ depth ground truth where correspondences between the laser and the camera images are manually selected for LiDAR-to-Camera calibration. Moreover, after projecting point clouds into image, ambiguous image regions such as windows and fences are manually removed. Even then, ground truth for a single frame is the result of aggregating LiDAR measurements corresponding to many adjacent frames. The process of ground truth generation is especially challenging in the presence of moving objects since they cannot be easily recovered from laser scanner data alone due to the rolling shutter of the LiDAR [MHG18]. So, first the static background of the scene is recovered by removing all dynamic objects and later, the dynamic objects are inserted relying on available CAD models [MHG18]. Moreover, non-rigidly moving objects like pedestrians and erroneous regions in the laser scans are manually masked.

So, supervised learning is not scalable. Hence, we forgo these manually annotated ground-truth annotations as supervision and instead we aim to leverage unannotated real data. Completely unsupervised learning for these tasks, especially for semantic ones is almost an impossible task. For this end, *semi-supervised learning (SSL)* is a more realistic setting where the algorithms have access to few labeled samples and many unlabeled samples drawn from the same training distribution. SSL algorithms can be *transductive* or *inductive* where latter ones can do inference in unseen test data as well. For, broader applicability of the proposed methods, we will be focusing on inductive algorithms.

Even though SSL is a key step towards unsupervised learning, there are cases where we do not have any annotation for the real data or at least some class attributes within the real data. However, we can still exploit the virtually infinite amount of synthetic data where ground truth comes for free such as SYNTHIA [RSM16], GTA5 [RVR16b] for semantic tasks and Scene Flow [MIH16], VKITTI [GWC16] for geometry tasks. This setting where algorithms have access to labeled virtual data and unlabeled real data at training time is called *unsupervised domain adaptation (UDA)*. The labeled virtual data and unlabeled real data are also called *source* and *target* domains respectively as the goal is to perform well on the unseen test samples drawn from the real data distribution.

Unfortunately, even the best graph engines still cannot perfectly model the real world. This



(a) Discriminative supervised (b) Generative supervised

Figure 1.1: Graph representations of supervised models: discriminative (left) and generative (right).

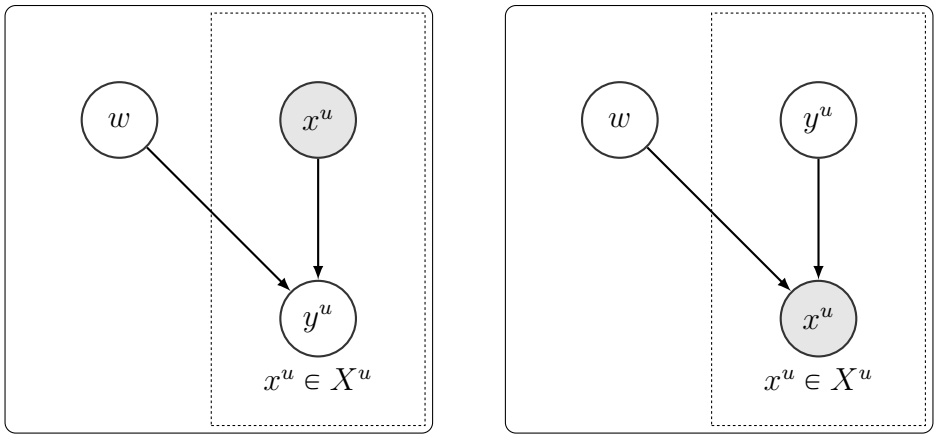
creates a domain gap in between input distributions, making it hard for a model trained on synthetic data to generalize well to real data. UDA methods aim to have small risk in the target domain by being invariant to domain dependent variations which are *nuisances*[1] for the task.

Sufficiency for the observed data can be achieved by applying identity transformation to the raw data. However, since the observed data is finite and does not necessarily capture all modes of the real distribution that it is drawn from, we do apply some transformations to the raw data with the hope of generalizing well to unseen samples of the same distribution. A common principle for choosing a transformation is to process data so that we keep only the portion of the data that is relevant to the task, and discount the complexity in the data due to the nuisances [Soa13]. Since nuisance is defined for a task, the transformations leading to nuisance-invariant representations should be designed depending on the task in hand.

In this thesis, we will propose such methods to tackle vision problems such as image classification, segmentation, monocular depth prediction and depth completion under SSL and UDA settings.

Current benchmarks for these tasks split the data into training and test sets which are disjoint sets sampled from the same distribution. However, worst-case empirical analysis of these networks

[1]A variable affecting observed data (e.g. the image formation process) but not of the task (e.g. variable to be estimated) is a nuisance [Soa13].



(a) Discriminative unsupervised (b) Generative unsupervised

Figure 1.2: Graph representations of unsupervised models: discriminative (left) and generative (right).

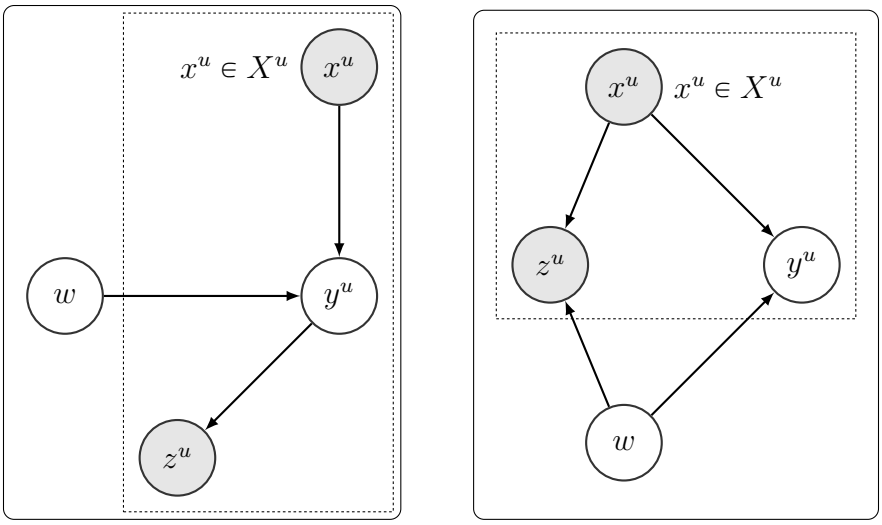
showed that they are vulnerable to even very small, imperceptible, additive noises [MFF16]. We will also examine the robustness of these networks more in depth.

1.1 When can unlabeled data help? – A Need for Regularization

In SSL, one is given some labeled and some unlabeled data to train (infer the parameters of) a classifier. This is an important problem in vision where annotations are costly but unlabeled data are aplenty. For a discriminative model to exploit unlabeled data, there has to be some prior on the model parameters or on the unknown labels [CSZ09].

1.1.1 Probabilistic Dependency of Unlabeled Data and Model Parameters.

Consider graph representations of supervised learning settings in Fig. 1.1. Here, shaded circles denote fully observed variables. The dotted rectangles denote the replication of variables for each sample in the set. Particularly, Fig. 1.1a shows that $\{y^l\}$ are independent and identically distributed, conditional on $\{x^l\}$ and model parameters w. Since data x^l and model parameter w is connected via observed child y^l, they are probabilistically dependent (i.e. d-dependent) [KF09]. For generative settings, model parameters and data are d-dependent whether y is observed (See Fig. 1.1b) or not



(a) (b)

Figure 1.3: Graph representations of discriminative unsupervised models augmented with different types of regularizations.

(See Fig. 1.2b) as parameters and data are directly connected.

In the discriminative approach, an unlabeled data does not influence the posterior for model parameters (See Fig. 1.2a). This is because the unlabeled points and the parameters become d-separated (independent from one another) when the label y is unobserved.

To restore the dependence, we need to augment the model. This can be done by introducing an additional variable z that is a child of y^u and it is observed (See Fig. 1.3a). This breaks the d-separation of observed data x and model parameters and allows probabilistic dependence to flow between these variables even though y^u is unobserved. To restore the dependence, we can also define observed variable z to be child of x^u and w (See Fig. 1.3b).

So, the question naturally arises once we define an observable variable z as a function of the model parameter and data, are we done solving unsupervised learning? Answer is negative as it is possible to restore the probabilistic dependency without solving the task. Consider a trivial example where we define z^u to be the output of our network $f(x^u; w)$ whose weights are randomly initialized. The network output $z^u = f(x^u; w)$ is fully observed, but it does not give us any criterion on how to choose the model parameters w. Or, in the next section, we will describe the max-margin

boundary where the idea is to keep $z_{01} = ||f(x_0; w) - f(x_1; w)||/||x_0 - x_1||$ small for any data pair (x_0, x_1). Clearly, z_{01} is an observable variable, but one can also choose to maximize it, which would converge to a completely different model.

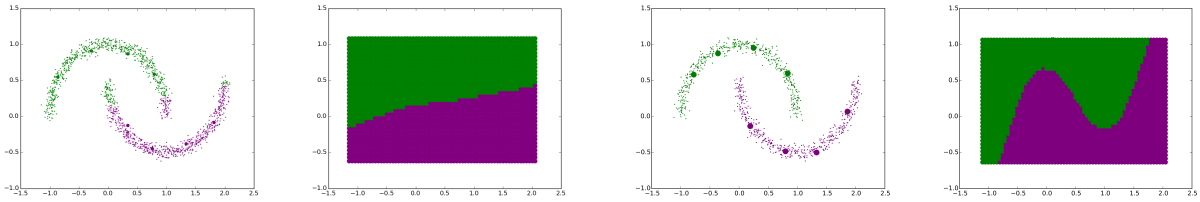
Or, we can consider the reconstruction task where y^u is equal to x^u; and thus y^u is also observed and probabilistic dependency is trivially satisfied. In other words, there is no unsupervised setting for the reconstruction task.

Therefore, the probabilistic dependency is only a necessary condition for leveraging the unlabeled data which can be trivial to achieve for certain tasks and for sufficiency, we need a task-specific regularization for a solvable task.

Furthermore, the use of generative modelling is not a solution to resolve the dependency problem. Since we are interested in discriminative tasks, we need to have a discriminative model even though it can be augmented with an auxiliary generative one $g(x^u; w_g)$. So, even though auxiliary generative model parameters and data are d-dependent by definition, there is no dependency between discriminative model $f(x^u; w_f)$ parameters and the data unless we augment the discriminative model with other regularizers. Or, one can use the Bayes theorem to get $P(y|x)$, but that would require one to correctly predict $P(x|y)$ for which d-dependency between data and parameters is not enough. We will see an example usage of generative modeling for discriminative UDA tasks in Chapter 5.

1.1.2 Common Regularizers

Topology of the scene can be estimated without ground-truth supervision exploiting the principles of epipolar geometry given sufficient parallax and visually discriminative Lambertian regions that are stationary in the environment, and are visible from the camera. So, traditional solutions holding under such conditions are applied in the form of regularization as a photometric (reprojection) consistency of predicted disparity, edge-aware local smoothness loss etc. However, for semantic vision problems like image classification, segmentation there is no such theoretically well-established unsupervised methods. Next, we will describe a few regularizers proposed for some of these problems that have shown to be successful by several empirical studies.



(a) Trained with labeled data alone.

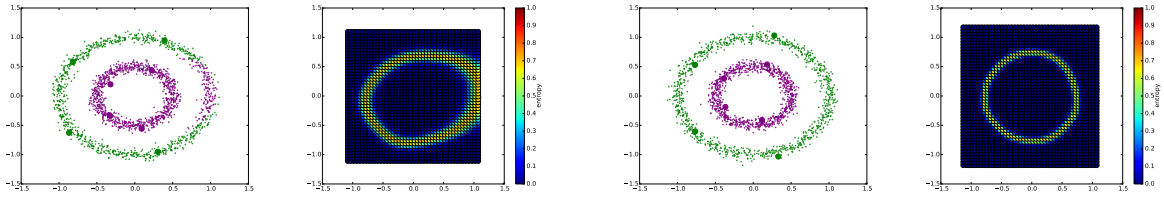
(b) Regularized with adversarial-training on unlabeled data.

Figure 1.4: Classification results and corresponding decision boundaries for the network trained with labeled data only (a) and the network regularized with unlabeled data using adversarial-training (b).

Cluster assumption posits that inputs with the same class are in the same cluster under an appropriate metric. It takes many forms: max-margin, low-density separation, input smoothness. For large-dimensional inputs like images, this can be achieved by adversarial training where network is forced to be robust to small, additive perturbations. Virtual adversarial training (VAT) of [MMK18] leveraged this idea but estimated second-order approximation of adversarial perturbations for computational efficiency.

In Fig. 1.4, we illustrate this idea for network trained only with few labeled samples (Fig. 1.4a) and regularized with many unlabeled samples using adversarial training (Fig. 1.4b). For ease of visualization, we choose 2-dimensional toy example where goal is to cluster two moons. As can be seen in the left panels, there are a few labeled samples (denoted with large circles) and many unlabeled samples (denoted with dots). In each sub-figure, resulting predictions on the training samples (colored as green or purple) and decision boundaries are given in the left and right panels respectively. As can be seen in Fig. 1.4a, when network is trained only with a few labeled samples, decision boundary intersects the unlabeled data and moons are not clustered correctly. In Fig. 1.4b, unlabeled samples are also exploited by forcing network output to be robust to perturbations on them resulting in max-margin decision boundary and accurate clustering.

Minimizing the entropy of predictions have a similar effect of forcing decision boundaries to be in the low-density region [GB05, KPG10]. The result of entropy minimization is demonstrated on a toy circle dataset in Fig. 1.5. Each circle contains 1000 unlabeled samples (denoted with dots)



(a) Trained with labeled data alone.

(b) Regularized with entropy-minimization on unlabeled data.

Figure 1.5: Classification results and corresponding entropy maps for the network trained with labeled data only (a) and the network regularized with unlabeled data using entropy-minimization (b).

and 4 labeled samples (denoted with larger circles). In each sub-figure, classification results on unlabeled samples and entropy of the classifiers are given in the left and right panels respectively. For network trained using labeled data alone (Fig. 1.5a), entropy of the classifier is high on the unlabeled samples (right panel) and clustering is wrong (left panel). However, when the entropy on the unlabeled samples is minimized (Fig. 1.5b), we get almost zero entropy everywhere except in the middle of the two circles (right panel) which results in perfect clustering of the unlabeled dots (left panel).

Even though these experiments demonstrate the potential of the unsupervised learning, for more complex datasets such as image datasets, input dimension is significantly higher than two. In such cases, simply enforcing max-margin boundary is not enough to ensure good generalization as there are many possible degenerate solutions satisfying the max-margin decision boundary. Thus, more task-specific regularizations are needed. In this thesis, we will propose new regularizers for various semantic and geometric vision tasks.

In unsupervised domain adaptation (UDA) on the other hand, we want to train a model to operate on input data from both the source and target domains, despite absence of annotated data for the latter. For instance, one may have a synthetic dataset, where annotation comes for free, but wish for the resulting model to work well on real data, where manual annotation is scarce or absent.

In UDA, common regularizers rely on aligning the marginal distributions of learned features for

the source and the target domains. These methods learn the parameters of a deep neural network using adversarial (min-max) criteria. The idea is to simultaneously recognize the class (output) as well as the domain (*e.g.*, "real vs. synthetic") by training the classifier to work as well as possible on both while encoder is fooling the discriminator for the latter. In a sense, the classifier becomes agnostic to the domain. This can be understood as aligning the marginal distribution of the inputs from the two domains. Unfortunately, this does not guarantee successful transfer, for it is possible that the source (say synthetic images) be perfectly aligned with the target (say natural images), and yet a natural image of a cat map to a synthetic image of a dog. It would be desirable, therefore, for the adaptation to align the outputs, along with the inputs. This prompted other methods to align, instead of the marginal distributions, the joint or conditional distribution of domain and class. This creates two problems: First, the target class labels are unknown; second, since there is a shared representation of the inputs, aligning the joint distributions may cause them to collapse thus losing the discriminative power of the model. For this, we will offer ideas to jointly align class and domain labels while leveraging the regularizers tailored to task.

1.2 Organization of Thesis

In Chapter 2, we introduce the vanilla semi-supervised learning (SSL) setting where few labeled samples and many unlabeled samples are drawn from the same distribution. We propose a novel regularization to be used in SSL settings and an algorithm called SaaS to optimize it. First, we show that the training speed in the supervised setting correlates strongly with the percentage of correct labels. Since, the speed of the training is a measurable quantity at the training time unlike the accuracy of the estimates, we propose to use it as an inference criterion for the unknown labels, without attempting to infer the model parameters at first.

In Chapter 3, we extend the work in SSL setting by acting on both the input (data) space, and the weight (parameter) space. First, we show that robustness to perturbation in input space and weight space do not imply each other. Then, we propose a method to perform such smoothing, which combines known input-space smoothing with a novel weight-space smoothing, based on a

min-max (adversarial) optimization. The resulting Adversarial Block Coordinate Descent (ABCD) algorithm performs gradient ascent with a small learning rate for a random subset of the weights, and standard gradient descent on the remaining weights in the same mini-batch.

In Chapter 4, we switch to a more challenging but potentially more practical problem setting called unsupervised domain adaptation (UDA) where labeled and unlabeled samples are drawn from slightly different distributions. This setting is more practical as labeled data can be drawn from synthetic data which can be produced abundantly using graph engines with the expense of domain shift from the real data. We propose a method for UDA that trains a shared embedding to align the joint distributions of inputs (domain) and outputs (classes), making any classifier agnostic to the domain. Joint alignment ensures that not only the marginal distributions of the domains are aligned, but the labels as well. We propose a novel objective function that encourages the class-conditional distributions to have disjoint support in feature space. We further exploit adversarial regularization to improve the performance of the classifier on the domain for which no annotated data is available.

In Chapter 5, we explore the use of generative modeling for the same UDA problem to have more interpretable models. We generate images in the unlabeled target domain in a manner that allows independent control of class (content) and nuisance variability (style). The proposed method differs from existing generative UDA models in that we explicitly disentangle the content and nuisance features at different layers of the generator network. We demonstrate the effectiveness of (pseudo)-conditional generation by showing that it improves upon baseline methods.

Up to Chapter 6, we focus on the classification tasks. But, unsupervised learning is more critical for dense prediction tasks like semantic segmentation where manual annotation is more costly as it requires each pixel of an image to be annotated instead of having one scalar annotation for the whole image. In Chapter 6, we propose a method for semantic segmentation in the UDA setting. We particularly examine the domain gap between spatial-class distributions and propose to align the local distributions of the segmentation predictions.

In Chapter 7, we switch to depth prediction tasks and we present a method for inferring dense depth maps from images and sparse depth measurements by leveraging synthetic data to learn the association of sparse point clouds with dense natural shapes, and using the image as evidence to

validate the predicted depth map. Our learned prior for natural shapes uses only sparse depth as input, not images, so the method is not affected by the covariate shift when attempting to transfer learned models from synthetic data to real ones. This allows us to use abundant synthetic data with ground truth to learn the most difficult component of the reconstruction process, which is topology estimation, and use the image to refine the prediction based on photometric evidence.

For all the methods proposed so far, we evaluated methods in terms of their accuracy on a held out test set. However, in many security-critical applications, worst-case risk analysis is essential. Especially given the vulnerability of deep networks on small additive perturbations, understanding these vulnerabilities and possible solutions is key to improve our models. In Chapter 8, we study the effect of adversarial perturbations on the task of monocular depth prediction. Specifically, we explore the ability of small, imperceptible additive perturbations to selectively alter the perceived geometry of the scene. We show that such perturbations can not only globally re-scale the predicted distances from the camera, but also alter the prediction to match a different target scene. We also show that, when given semantic or instance information, perturbations can fool the network to alter the depth of specific categories or instances in the scene, and even remove them while preserving the rest of the scene. To understand the effect of targeted perturbations, we conduct experiments on state-of-the-art monocular depth prediction methods. Our experiments reveal vulnerabilities in monocular depth prediction networks, and shed light on the biases and context learned by them.

Finally in Chapter 9, we will discuss the limitations of the proposed methods and possible future directions to conclude the thesis.

CHAPTER 2

Saas: Speed as a Supervisor for Semi-supervised Learning

2.1 Introduction

In semi-supervised learning, we are given N^l labeled samples $\{x_i^l\}_{i=1}^{N^l}$ with corresponding labels y^l and N^u unlabeled samples, $\{x_i^u\}_{i=1}^{N^u}$. The entire training dataset is X with cardinality $N = N^l + N^u$.

The key idea of our approach is to *use speed of convergence as an inference criterion for the value of the unknown labels* for semi-supervised learning (SSL). Fig. 2.1 explicitly shows the relation between label corruption and training speed. In the next paragraphs, we discuss our contribution in relation to the vast and growing literature on SSL.

2.2 Related Work

Cluster assumption posits that inputs with the same class are in the same cluster under an appropriate metric. In general, it could be framed as $\int ||\nabla_x f_w(x)||^2 d\mu_x$ being small where μ_x is the probability distribution over some manifold. VAT [MMK15, MMK17] is a recent application of this idea to deep networks, realized by adding a regularization term to minimize the difference between the network outputs for clean and adversarial noise-added-inputs. This method is similar to the adversarial training of [GSS14], the main difference being that it does not require label information, and thus can be applied to SSL.

Ensemble Methods include teacher-student models, that use a combination of estimates (or weights) of classifiers trained under stochastic transformations. Although we train only one network, our method resembles the teachers-student models. However, we randomly start a student model at

each outer epoch. In [LA16] the prediction of the network over the training epochs are averaged, whereby in each epoch a different augmentation is applied. [TV17] minimize the consistency cost, which is the distance between two network outputs. Hence, the student network minimizes classification and consistency costs with labeled data and only consistency with the unlabeled data. The weights of the teacher model are the running average of the weights of the student network.

Self-training is an iterative process where confident labels from previous iterations are used as ground truth. In [BM98], disjoint subsets of features of labeled samples are used to produce different hypotheses on randomly selected subsets of the unlabeled data. Labeled data are extended with the most confident estimates on this subset. We maintain an estimate of the posterior probability of each label and only force a point estimate in the refinement (second) phase of the algorithm.

Encoding priors. In image classification one can enforce invariance of labels to some transformations. This is achieved by minimizing the difference between network outputs under different transformations. In [SJT16b], transformations are affine (translation, rotation, flipping, stretching and shearing). Although they achieve good results, their improvement on baseline supervised performance (using only labeled data) is marginal. *E.g.*, in CIFAR-10 supervised error is 13.6% while the semi-supervised error is 11.29%. Similarly, [SVL92] suggests minimizing the norm of directional derivatives of the network with respect to small transformations. We also employ augmentations as most SSL works on image classification.

Generative models used to be the standard for SSL, but the high dimensionality of problems in vision presents a challenge. Adversarial methods like GANs have been recently applied, whereby an additional $K + 1$-th (fake) class is used. The loss function is designed to force the discriminator output to be small for the fake class for the unlabeled samples while making it high for the generated samples. [SGZ16] suggested a regularizer for the generator, called feature matching (FM), whereby the generator tries to match the first-order statistics of the generated sample features to those of the real data.

Graph based methods. [YCS16] assumes that an affinity matrix of size $N \times N$ is given, which has information independent from the one in the features of the data. In the loss function, they have a term penalizing different labels assigned to similar samples based on this similarity

matrix. [NWH11, SZY16] finds a sparse clustering using the ℓ_1-norm. Recent graph-based methods [HMC17, GTB18, KW16, WRM12] exploit deep networks for function approximation in a manner that can be used for SSL.

Within this rich and multi-faceted context, our approach provides one more element to consider: The fact that the speed of convergence when optimizing with respect to the probability of unknown labels is highly dependent on their correctness, even when starting from a random initial condition. This frees us from having to jointly optimized the parameters and the posterior on the labels, which would blow up the dimensionality, and allows us to focus sequentially on first estimating the unknown label distribution – irrespective of the model parameters/weight – and then retrieve the weights using the maximum a-posteriori estimate of the labels.

Our method can be combined with other ideas recently introduced in SSL, including using adversarial examples. We do not do so in our experiments, to isolate the contribution of our algorithm. Nevertheless, just the method alone, with some data augmentation but without sophisticated tricks, achieves promising performance.

In Sec. 2.4, we give the training details for the experiments. In Sec. 2.5, we test the SaaS algorithm on SSL benchmarks, and next, we formalize our method in greater detail.

2.3 Derivation of the model

We *estimate the posterior distribution of the unknown labels* $P_i^u := P(Y_i^u | X_i^u = x_i^u)$. The outer loop of the algorithm updates the estimates of the posterior P_i^u, while the inner loop optimizes over the weights (for the fixed estimate of the posterior) to estimate the loss decrease over the time interval. It is important to note that we are *not attempting to infer the weights* (but only the posterior distribution of the unknown labels), which are resampled at each (outer) iteration.[1] By design, the weights do not converge, yet empirically we observe that the posterior distribution of the unknown labels does. We then use the maximum a-posteriori estimate of the labels $\hat{y}_i^u = \arg\max_i P_i^u$ to infer

[1] We have tested both drawing the weights from a Gaussian distribution or resetting them to their initial value, which yields similar results.

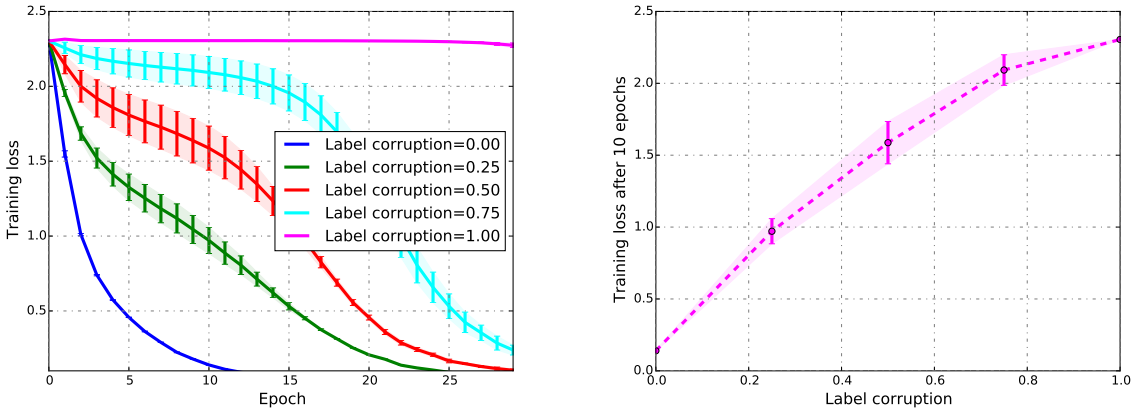


Figure 2.1: Supervision quality affects learning speed. During training, the loss decreases rapidly when most labels provided are correct, and slows down significantly as the percentage of correct labels decreases. The left plot shows the loss when training a Resnet18 on CIFAR10 for different percentages of corrupted labels. The error bars show mean and standard deviation over 3 runs with random initial weights. The right panel shows the loss as a function of the percentage of incorrect labels for a unit of time corresponding to ten epochs. All the results use a fixed learning rate of 0.1, with no data augmentation or weight decay.

a point-estimate of the weights $\hat{w}$ in a standard supervised training session. $N = N^u + N^l$ is the total number of samples and η are suitable learning rates for a batch size $|B|$ in SGD.

To measure learning speed, we use a small number of epochs as the unit of time and compute the decrease of the loss in that interval when following a standard optimization procedure (stochastic gradient descent or Langevin dynamics). The main idea of our SSL algorithm is then to optimize the labels of unlabeled data (or more precisely, the posterior distribution of the unlabeled data) to maximize the loss decrease.

The resulting SaaS algorithm is composed of an *outer loop*, which updates the distribution of unknown labels, and an *inner loop* which simulates the optimization procedure over a small number of epochs.

The proposed algorithm is unusual, as the posterior distribution of the unknown labels is initially inferred *independently of the model parameters* (weights), rather than along with them as customary in SSL.

Formulation. We represent a deep neural network with parameters (weights) w, trained for classification into one of K classes, as a function $f_w(x) \in \mathbb{R}^K$ where x is the input (test) datum, and the k-th component of the output approximates the posterior probability $f_{w_t}(x_i)[k] \simeq P(y_i = k|x_i)$. Stochastic gradient descent (SGD) performs incremental updates of the unknown parameters with each iteration t computing the loss by summing on a random subset of the training set called "mini-batch" B_t. The number of iterations needed to sample all the dataset is called an epoch. We represent SGD as an operator $G(\cdot) : w_t \to w_{t+1}$, which maps the current estimate of the weights to the next one. Note that G depends on the given (true) labels and the hypothesized ones for the unlabeled data.

To quantify *learning speed* we use the cumulative loss in a fixed time (epoch) interval: For a given training set $\{x, y\}$, it is the aggregated loss during T optimization steps, *i.e.*, the area under the learning curve

$$L_T = \frac{1}{T} \sum_{t=1}^{T} \frac{1}{|B_t|} \sum_{i=1}^{|B_t|} \ell(x_i, y_i; w_t) \tag{2.1}$$

where $|B_t|$ denotes the cardinality of the mini-batch, composed of samples

$(x_i, y_i) \sim P(x, y)$; ℓ denotes the classification loss corresponding to weights w_t. Computing the loss above over all the data points requires the labels being known. Alternatively, it can be interpreted as a loss for a joint hypothesis for the weights w_t *and* the label y_i. We use the cross-entropy loss, which is the sampled version of $H_{P,Q}(y|x) = \mathbb{E}_{P(x)}\mathbb{E}_{P(y|x)} - \log Q(y|x)$ where $Q(y = k|x) = f_w(x)[k]$ is the kth coordinate of the output of the network.

The true joint distribution $P(x)P(y|x) = P(x, y)$ is not known, but the dataset is sampled from it. In particular, if $y_i = k$ is the true label for x_i, we have $P(y_i|x_i) = \delta(y_i - k)$ where δ is Dirac's Delta. Otherwise, we represent it as an unknown K-dimensional probability vector P_i^u with k-th component $P_i^u[k] = P(y_i = k|x_i),\ k = 1, \dots, K$, to be inferred along with the unknown weights w. We can write the sum $\sum_{k=1}^{K} P_i[k]P_j[k]$ as an inner product between the probability vectors $\langle P_i, P_j \rangle = P_i^T P_j$, so that the *cumulative loss* can be written as

$$L_T(P^u) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{|B_t^u|}\sum_{i=1}^{|B_t^u|} \underbrace{-\langle \log f_{w_t}(x_i^u), P_i^u \rangle}_{\ell(x_i^u, P_i^u; w_t)} \tag{2.2}$$

where B_t^u is mini-batch of unlabeled samples at iteration t. Note that cross-entropy depends on the *posterior distribution* of the unknown labels, P_i^u, rather than their sample value y_i^u. The loss depends on the posterior for the entire unlabeled set, which we indicate as an $N^u \times K$ matrix P^u, and the entire set of weights $w = \{w_1, \dots, w_T\}$.

We also add as an explicit regularizer the entropy of the network outputs for the unlabeled samples: $-\mathbb{E}_Q \log Q(y^u|x^u)$, as common in SSL [GB05], which we approximate with the unlabeled samples as

$$H_Q(w) = \sum_{i=1}^{N^u} \underbrace{-\langle f_w(x_i^u), \log f_w(x_i^u) \rangle}_{q(x_i^u; w)} \tag{2.3}$$

We further incorporate data augmentation by averaging over group transformations $g(x) \in \mathbb{G}$, such as translation and horizontal flipping, sampled uniformly $g_i \sim U(\mathbb{G})$. Let us define the following shorthand notations: $\ell(B_t^u, P^u; w_{t-1}) = \frac{1}{|B_t^u|}\sum_{i=1}^{|B_t^u|} \ell(g_i(x_i^u), P_i^u; w_{t-1})$ and $q(B_t^u; w_{t-1}) = \frac{1}{|B_t^u|}\sum_{i=1}^{|B_t^u|} q(g_i(x_i^u); w_{t-1})$. Similarly for labeled set, $\ell(B_t^l, P^l; w_{t-1}) = \frac{1}{|B_t^l|}\sum_{i=1}^{|B_t^l|} \ell(g_i(x_i^l), P_i^l; w_{t-1})$ where $P_i^l = \delta_{i,k}$ is the Kronecker Delta with k the true label associated to x_i^l. The overall learning

can be framed as the following optimization

$$
\begin{aligned}
P^u = \arg\min_{P^u} \frac{1}{T} \sum_{t=1}^{T} \ell(B_t^u, P^u; w_{t-1}) \qquad (2.4)\\
\text{subject to } w_{t-\frac{1}{2}} = w_{t-1} - \eta_w \nabla_{w_{t-1}} \left(\ell(B_t^u, P^u; w_{t-1}) + \beta q(B_t^u; w_{t-1})\right)\\
w_t = w_{t-\frac{1}{2}} - \eta_w \nabla_{w_{t-\frac{1}{2}}} \ell(B_t^l, P^l; w_{t-\frac{1}{2}}) \; \forall \; t = 1 \dots T\\
P^u \in \mathcal{S}
\end{aligned}
$$

where the last constraint imposes that the rows of P^u be in the probability simplex of $\mathbb{R}^K$. The objective of the above optimization is to find the posterior of the unlabeled data that leads to the fastest learning curve, when using stochastic gradient descent to train the weights w on both labeled and unlabeled data. The update of the weights is specifically decomposed into two steps: the first step is an update equation for the weights with *unlabeled* samples and posterior P^u, while the second step updates the weights using the labeled samples and ground truth labels. We stress that the latter update is crucial in order to fit the weights to the available training data, and hence prevents from learning trivial solutions of P^u that lead to a fast convergence rate, but does not fit the data properly. We also note that the entropy term is only minimized for unlabeled samples. We set $\beta = 1$ in all the experiments.

It is customary to regularize the labels in SSL using entropy or a proxy [MMK17, DYY17, Spr15], including mutual exclusivity [SJT16a, XZF17]. [SSG12] uses mutual exclusivity adaptively by not forcing it in the early epochs for similar categories. All these losses force decision boundaries to be in the low-density region, a desired property under cluster assumptions. [KPG10, Spr15] also maximize the entropy of the marginal label distribution to balance the classes. Together with entropy minimization, balancing classes is equivalent to maximizing the mutual information between estimates and the data if the label prior is uniform. However, we did not apply this loss to not restrict ourselves to balanced datasets or to the settings where we have prior knowledge on the label distributions.

2.3.1 Implementation

To solve the optimization problem in Eqn. 2.4, we perform gradient descent over the unknowns P^u, where P^u is the unknown-label posterior initialized randomly. Starting from w_0 sampled from a Gaussian distribution, the inner loop performs a few epochs of SGD to measure learning speed (cumulative loss) L_T *while keeping the label posterior fixed*. The outer loop then applies a gradient step to update the unknown-label posterior P^u. After each update, the weights are either reset to w_0 or resampled from the Gaussian. At the beginning of each outer epoch, label estimates $P^u \in \mathbb{R}^{N^u \times K}$ are projected with operation $\Pi(P^u)$ to the closest point on the probability simplex of dimension $N^u \times K$.

After the label posterior converges (the weights never do, by design, in the first phase), we select the maximum a-posteriori estimate $\hat{y}_i^u = \arg\max_i P_i^u$ and proceed with training as if fully supervised in the second phase. We call the resulting algorithm SaaS and described in Alg. 1.

It should be noted that the computation of the gradient $\nabla_{P^u} \ell(B_t^u, P^u; w_t)$ is not straightforward, as w_t is, in general, a (complex) function of P^u. In the computation of the gradient, we omit here the dependence of w_t on P^u, and use the approximation $\nabla_{P_i^u} \ell(w_t, x_i^u, P_i^u) \approx -\log f_{w_t}(x_i^u)$. This approximation is exact whenever each data point is visited once (i.e., $T = 1$ epoch); as T is chosen to be relatively small here, we assume that this approximation holds.

It is important to note that, with the SaaS algorithm, we are not attempting to solve the optimization problem: $\min_{w,P^u} \sum_{i=1}^N \ell(x_i, P_i^u; w)$. This problem has many trivial solutions, as observed by [ZBH16], as deep neural networks can easily fit random labels when trained long enough. Thus, for many posteriors P^u, there are weights w achieving zero loss on this objective. One of many such trivial solutions is setting the label posterior P^u to the outputs of the network trained only with the labeled samples. This would result in the same test performance as that of a supervised baseline and does not utilize the unlabeled samples at all. On the other hand, SaaS uses the cumulative loss up to a *fixed*, small iteration T as an inference criterion for label posterior P^u.

Finally, instead of projecting P^u onto the probability simplex $\mathcal{S}$, we have found that the projection onto a slightly modified set $\mathcal{S}_\alpha = \{x \in \mathbb{R}^K : \sum_i x_i = 1, x_i \geqslant \alpha\}$ (with $\alpha \geqslant 0$ chosen to

Algorithm 1 SaaS Algorithm

$P^u \sim N(0, I)$

Select learning rates η for the weights η_w and label posteriors η_{P^u}

Phase I: Estimate P^u

while P^u has not stabilized **do**

$P^u = \Pi(P^u)$ (project posterior onto the probability simplex)

$w_1 \sim N(0, I)$

$\Delta P^u = 0$

// Run SGD for T steps (on the weights) to estimate loss decrease

for $t = 1 : T$ **do**

$w_{t-\frac{1}{2}} = w_{t-1} - \eta_w \nabla_{w_{t-1}} \left(\ell(B_t^u, P^u; w_{t-1}) + \beta q(B_t^u; w_{t-1}) \right)$

$w_t = w_{t-\frac{1}{2}} - \eta_w \nabla_{w_{t-\frac{1}{2}}} \ell(B_t^l, P^l; w_{t-\frac{1}{2}})$

$\Delta P^u = \Delta P^u + \nabla_{P^u} \ell(B_t^u, P^u; w_t)$

// Update the posterior distribution

$P^u = P^u - \eta_{P^u} \Delta P^u$

Phase II: Estimate the weights.

$\hat{y}_i^u = \arg\max_i P_i^u \; \forall i = 1, \cdots, N^u$

$w_1 \sim N(0, I)$

while w has not stabilized **do**

$w_{t-\frac{1}{2}} = w_{t-1} - \eta_w \nabla_{w_{t-1}} \frac{1}{|B_t^u|} \sum_{i=1}^{|B_t^u|} \ell(x_i^u, \hat{y}_i^u; w_{t-1})$

$w_t = w_{t-\frac{1}{2}} - \eta_w \nabla_{w_{t-\frac{1}{2}}} \frac{1}{|B_t^l|} \sum_{i=1}^{|B_t^l|} \ell(x_i^l, y_i^l; w_{t-\frac{1}{2}})$

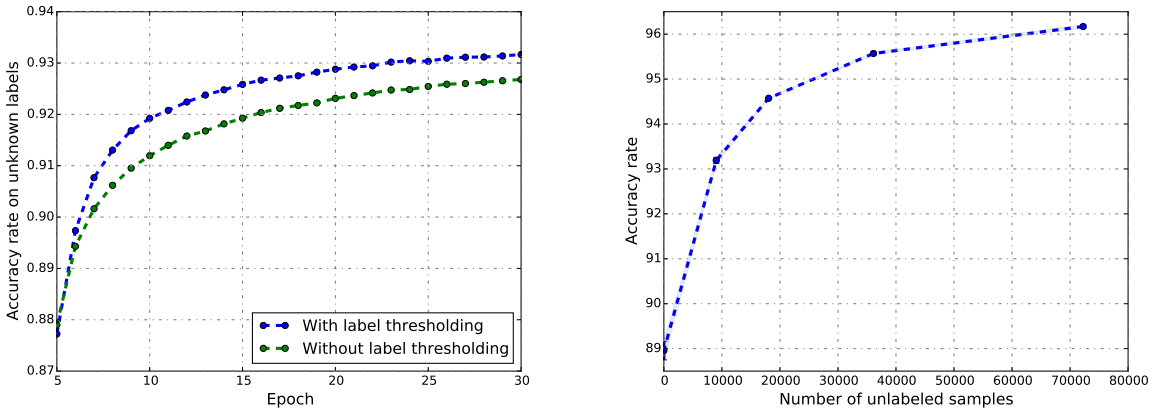


Figure 2.2: (Left) Effect of label thresholding. We project P^u to the closest probability simplex with minimum probability for a class being 0.05. The plot is given for unlabeled accuracy as label thresholding used in the first phase of the SaaS. The plot is given from epoch 5 to epoch 30. (Right) Test accuracy vs. the number of unlabeled samples. Accuracy on the test data versus the number of unlabeled samples for the SVHN dataset using ResNet18. As the number of unlabeled samples increases, performance improves significantly, as expected in a semi-supervised learning scheme. Results are averaged over three random labeled sets, but error bars are not visible because deviations are smaller than the line-width.

be small) lead to better optimization results for P^u. This is in line with recent work in supervised classification, where this technique is used in order to improve the accuracy of deep neural networks [PTC17, SVI16]. Fig. 2.2 (left) illustrates the effect of this approach for SaaS, and shows a clear improvement in SVHN dataset.

2.4 Training details

As pointed out by [ZBH16], deep networks can easily (over)fit random labels. We set T small enough (40 epochs for CIFAR10 and 5 epochs for SVHN) so that simulated weights cannot fit randomly initialized posterior estimates in the early epochs. We use ResNet18 [HZR16b] as our architecture and vanilla SGD with momentum 0.9 as an optimizer. We perform random affine transformations as data augmentations both in SVHN and CIFAR10. We additionally use the horizontal flip and color jitter in CIFAR10. Learning rates for w and P^u are chosen as $\eta_w = 0.01$ and $\eta_{P^u} = 1$ respectively. We keep these rates fixed when learning P^u. We fixed the number of outer epochs as well for the first phase of the algorithm by setting it to 75 for SVHN and 135 for CIFAR10. For the second phase of SaaS (supervised part), training is not limited to a small epoch T. Instead, the learning rate is initialized as 0.1 and halved after 50 epochs unless validation accuracy is increasing. We stop when the learning rate reaches 0.001.

Datasets. SVHN [NWC11] consists of images of house numbers. We use $73,257$ samples for training, rather than the entire $600,000$ images; $26,032$ images are separated for evaluation. CIFAR-10 [KH09] has $60,000$ images, of which $50,000$ are used for training and $10,000$ for testing. We choose labeled samples randomly. We also choose them to be uniform over the classes as it is done in previous works [MMK17].

Network. *conv-large* model used in some of the experiments is given in Table 2.1. Batch-norm is used after each convolutional layer. The slopes of leaky-ReLU functions are set to be 0.1.

Preprocessing. When we use *conv-large* instead of ResNet18 for the CIFAR-10 dataset, we have employed ZCA preprocessing. Before applying ZCA, we standardize the dataset. ZCA transformation is $X_{\text{ZCA}} = XU(S^{-0.5} + \epsilon I)U^T$ where S and U are eigenvalue and eigenvector

3×3 convolution, 128 lReLU
3×3 convolution, 128 lReLU
3×3 convolution, 128 lReLU
2×2 max-pool, stride 2, dropout with probability 0.5
3×3 convolution, 256 lReLU
3×3 convolution, 256 lReLU
3×3 convolution, 256 lReLU
2×2 max-pool, stride 2, dropout with probability 0.5
3×3 convolution, 512 lReLU
1×1 convolution, 256 lReLU
1×1 convolution, 128 lReLU
Global average pooling, $6 \rightarrow 1$
Fully connected layer: $128 \rightarrow 10$
Softmax

Table 2.1: The network *conv-large* [TV17, MMK17] used to get results in Table 2.4. The results with this network reported, in addition to ResNet18, to have a fair comparison with [TV17, MMK17]. Slope of each leaky RELU (lReLU) layer is 0.1.

matrices of covariance of data matrix X respectively. We have observed that the value of ϵ was important for our performance and set it to be 500.

Augmentations. For our results with ResNet18, we have used shear, zoom, rotate, translation. For CIFAR-10, we additionally used horizontal flip and color jitter. For our results with *conv-large*, we have only used translation and horizontal flip. The latter is not used for SVHN.

2.5 Empirical evaluation

We test the SaaS algorithm against the state-of-the-art in the most common benchmarks, described next.

As a baseline, the same network is trained on the same datasets but using only the labeled subset (i.e. 4K samples for CIFAR10 and 1K samples for SVHN). When training the (supervised) baseline, we employ the same learning parameters, architecture, and augmentations as Phase II of SaaS. As expected, SaaS substantially improves baseline results, which is indicative that unlabeled data being effectively exploited by the algorithm (See Table 2.2).

In Table 2.3, we compare SaaS with state-of-the-art SSL methods on standard SSL benchmarks. In CIFAR-10, algorithms are trained with 4,000 labeled and 46,000 unlabeled samples. In SVHN, they are trained with 1,000 labeled and 72,257 unlabeled samples. The means and deviations of the test errors are reported by averaging over three random labeled sets. The state-of-the-art methods we compare include input smoothing algorithms [MMK17], ensembling models [TV17, LA16], generative models [SGZ16] and models employing problem-specific prior [SJT16b]. SaaS is comparable to state-of-the-art methods. Specifically, SaaS achieves the best performance in SVHN and the second-best result in CIFAR10 after VAT. Considering that VAT does input smoothing by adversarial training, our performance can be improved by combining with it.

An SSL algorithm is expected to be more accurate when the number of unlabeled data increases. As it can be seen in Fig. 2.2 (right), we consistently get better results with more unlabeled samples.

We motivated SaaS as a method finding labels for which training decrease in a fixed small number of epochs (e.g. 10) is the maximum. To verify that our algorithm actually does what is

Dataset	CIFAR10-4k	SVHN-1k
Error rate by supervised baseline on test data	17.64 ± 0.58	11.04 ± 0.50
Error rate by SaaS on unlabeled data	12.81 ± 0.08	6.22 ± 0.02
Error rate by SaaS on test data	10.94 ± 0.07	3.82 ± 0.09

Table 2.2: Baseline error rates. Error rates on the benchmark test set for the baseline system trained with 4K labeled samples on CIFAR10 and 1K labeled samples on SVHN. Error rate on unknown labels. SaaS performance on the *unlabeled* set. Error rate on test data. SaaS performance on the *test* set. Results are averaged over three random labeled sets. As it can be seen, results of SaaS on test data are significantly better than that of baseline supervised algorithm.

Method-Dataset	CIFAR10-4k	SVHN-1k
VAT+EntMin [MMK17]	**10.55**	3.86
Stochastic Transformation [SJT16b]	11.29	NR
Temporal Ensemble [LA16]	12.16	4.42
GAN+FM [SGZ16]	15.59	5.88
Mean Teacher [TV17]	12.31	3.95
SaaS	10.94 ± 0.07	**3.82 ± 0.09**

Table 2.3: Comparison with the state-of-the-art. Error rates on the test set are given for CIFAR10 and SVHN. NR stands for "not reported." CIFAR10 is trained using 4K labels, SVHN using 1K. Results are averaged over three random labeled sets. Despite its simplicity, SaaS performs at the state-of-the-art. It could be combined with adversarial examples (VAT) but here we report the naked results to highlight the role of speed as a proxy for learning in a semi-supervised setting while maintaining a simple learning scheme.

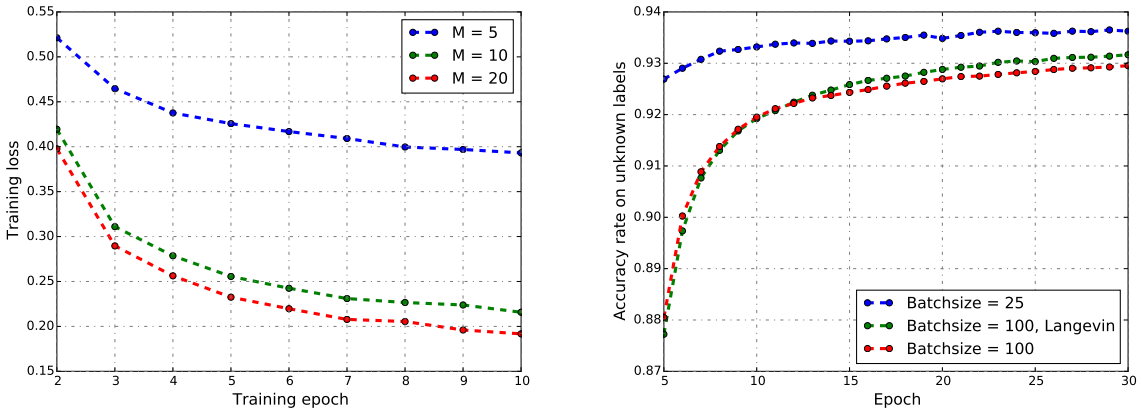


Figure 2.3: (Left) SaaS finds pseudo-labels on which training is faster. Training loss for a network trained with given P^u, as estimated by the first phase of our algorithm with different numbers of outer epochs M. M is the number of the epoch at which outer iteration stopped in the SaaS. Label hypotheses generated by our algorithm lead to faster training as the iteration count increases. Losses are plotted starting with epoch 2. This plot verifies that our algorithm finds labels on which training is faster. (Right) Effect of Langevin. Performance improves with a smaller batch size, albeit at a significant computational cost: the algorithm is about three times slower for $|B| = 25$ (plots shown for SVHN). We, therefore, choose $|B| = 100$ and add zero-mean Gaussian noise to the weight updates for comparable results (Langevin). The results converge to those of $|B| = 25$ when we train longer. The plot is given for unlabeled accuracy because Langevin is only used in the first phase of SaaS. The plot is given from epoch 5 to epoch 30.

intended to do, we train networks on the pseudo-labels generated by SaaS. One can see in Fig. 2.3 (left) that as SaaS iterates more (i.e. as the number of updates for P^u increases), resulting pseudo-labels leads to larger training loss decrease (faster training) in the early epochs. This experiment verifies that SaaS gives pseudo-labels on which training would be faster.

The results reported in Table 2.2 are with ResNet18 and affine augmentations. Our method uses augmentation, but for direct comparison with some of the previous works, we also report results with the convolutional network *conv-large* and translational augmentations as used in [MMK17, TV17] in Table 2.4. Additionally, horizontal flipping is used in CIFAR10. Moreover, as a pre-processing, we centered images relative to the Mahalanobis metric (known as ZCA) as in [MMK17, TV17].

Dataset	CIFAR10-4k	SVHN-1k
Error rate by supervised baseline on test data	17.88 ± 0.19	12.72 ± 1.13
Error rate by SaaS on unlabeled data	14.26 ± 0.30	7.26 ± 0.19
Error rate by SaaS on test data	13.22 ± 0.31	4.77 ± 0.27

Table 2.4: For direct comparison, we implement SaaS with the *conv-large* architecture of [MMK17] and the same augmentation scheme. Baseline performance (supervised) is also shown. SaaS improves both on the unlabeled set and test set. Results are averaged over three random labeled sets.

Small batch-size and Langevin dynamics. Finally, we discuss a method we use to reduce the training time for SaaS. We achieve better performance with a smaller batch size $|B| = 25$ for both labeled and unlabeled data. When $|B| = 100$, generalization performance degrades as expected [KMN16]. Unfortunately, small batch-size slows down training, so we use $|B| = 100$ for both labeled and unlabeled data and add zero-mean Gaussian noise to the weight updates, a process known as stochastic gradient Langevin dynamics (SGLD) [WT11, RRT17, CCS16], with variance $10^{-5}\eta_w$ for all the datasets. Comparison of small and large batches without noise and large batches with noise can be seen in Fig. 2.3 (right). With this, the first phase of the algorithm (getting the estimates of unknown labels) takes about 1 day for SVHN and 4 days for CIFAR10 using GeForce GTX 1080 when we use ResNet18.

Failure case. As we have shown, our algorithm performs well even with few augmentations (e.g. only translation). However, when we do not use any augmentation at all, our method does not perform as well. With no augmentation and ResNet18, our algorithm suffers a very large drop in performance, achieving error rates of 40.19 ± 3.89 for SVHN and 64.05 ± 1.79 for CIFAR10 on unlabeled data. We next explain this substantial change in the performance.

Fig. 2.1 suggests that there is a strong correlation between label accuracy and training speed. However, note that this plot is an average over different realizations of labels and initial weights. This does not suggest that for every realization of random labels, this correlation would hold.

A simple example is having constant labeling on all the samples for which training would be immediate. In this case, most of the labels would be incorrect for a balanced dataset meaning that the correlation between training speed and label accuracy does not hold for every single realization. Hence, we need to have a way of eliminating the degenerate solutions. While the first constraint we put to eliminate these solutions is to impose a small training loss on labeled examples, this might not be enough in many semi-supervised settings. Data augmentation further puts constraints on the desired unknown label posterior: labels of images have to remain constant with respect to image transformations. This constraint hence guides the algorithm to the desired label posterior and leads to significant performance gains on SaaS.

2.6 Discussion

The key idea of our approach to SSL is to leverage training speed as a proxy to measure the quality of putative labels as they are iteratively refined in a differentiable manner.

That speed of convergence relates to generalization is implicit in the work of [HRS16], who derive an upper bound on generalization error as a function of the sum of step sizes, suggesting that faster training correlates with better generalization.

Another way of understanding our method is via shooting algorithms used to solve boundary value problems (BVP). In a BVP with second-order dynamics, a trajectory is found by simulating it with a guess of the initial state; then, the initial state is refined iteratively such that the target error would be minimized. In our problem, dynamics are given by SGD. Assuming that we use SGD without momentum, we have a first-order differential equation. The first boundary condition is the initialization of weights and the second boundary condition is a small cumulative loss. The latter one is used to refine P^u which is a parameter of the dynamics instead of the initial state.

CHAPTER 3

Input and Weight Space Smoothing for Semi-supervised Learning

3.1 Introduction

Both input-space regularization, or "smoothing" ([Joa99, GSS14, MMK17, CS19b]) and weight-space smoothing ([HS97, CCS16]) have been shown to improve both supervised (SL) and semi-supervised learning (SSL). The first question we address is whether the two are, in some sense, equivalent. Although the two couple linearly in deep networks, the composition of non-linearities complicates the analysis beyond analytical feasibility.

To answer the question, we conduct experiments that show that, for nonlinear and/or over-parametrized classifiers, input, and weight smoothing are not only not equivalent, but they are complementary, suggesting that applying both may be beneficial. The second question we address, therefore, is whether this can be done efficiently to yield performance improvements relative to methods that only address one of the two.

To this end, we propose a new algorithm for weight smoothing called *Adversarial Block Coordinate Descent (ABCD)*, which we combine with a standard input-smoothing algorithm (VAT), and test the result on SSL benchmarks on the CIFAR10 and SVHN datasets. ABCD combined with VAT achieves state-of-the-art performance with minimal data augmentation (translation and reflection), without complex architectures (e.g. ResNet [HZR16b]).

In the next subsection, we review the related work. In the following three subsections, we describe input, weight smoothing, and we show them to not be equivalent. In the next section, we describe the proposed algorithm ABCD, and in the following one, we put it to the test on visual

SSL benchmarks. While in this section our method is motivated heuristically, there are theoretical groundings for performing joint regularization in weight- and input-space, which we discuss in Sec. 3.7.1. There, we show that the two are not equivalent, and in fact are complementary, one affecting the minimality of the resulting representation, the other insensitivity to nuisance variability.

3.2 Related Work

Next, we discuss our contribution in the context of related and recent work in SSL.

Input smoothing in SSL. In addition to works referenced in the earlier chapters, some graph-based methods ([SRG14, YCS16, ZGL03]) penalize having different labels for "similar" input pairs. For instance, given that s_{ij} is the measure of similarity for input samples x_i and x_j, they minimize the energy $s_{ij}(f(x_i; w) - f(x_j; w))^2$ performing label propagation under the constraint of fitting to labeled data. This forces the discriminant to change little in response to different inputs with large s_{ij}.

Weight smoothing in SSL. Teacher-student methods [LA16, TV17] average over many predictions or weights in a way that the teacher network can attract student networks towards itself. A similar algorithm is suggested by [ZCL15] for parallel computing under communication constraint where each replica is attracted to the reference system. [BBC16] studies such algorithms for models with discrete variables and they argue that they find robust local minima. Thus, one can relate the success of teacher-student models of state-of-the-art deep SSL algorithms for their ability to converge to robust weights. Recent work of [PPS17] also combines VAT with Virtual Adversarial Dropout (VAdD) and, like us, improve upon the VAT baseline as a result. VAdD finds a zero mask of dropout adversarially at each update rather than trying to be robust against adversarial "directions". VadD is not motivated with the flat-minima [HS97] and there is no experiment supporting whether it is able to find wide valleys. In this work, we establish that input smoothing and weight smoothing are complementary, and applying both is useful for SSL tasks. We achieve state-of-the-art results by combining VAT of [MMK17] with the proposed algorithm ABCD. We conduct many experiments to verify that ABCD-trained networks are indeed robust against adversarial perturbations in the

weight space.

Effect of noise on generalization. Adding random noise to gradients is known to improve the generalization [WT11]. [JKA17] analyzes the effect of the inherent noise due to mini-batch usage in the properties of point converged by SGD. They conclude that for larger noise, the network favors wider minima under the assumption that the noise is isotropic. ABCD differs from these works by adding adversarial noise to weights instead of random noise.

3.3 Preliminaries

3.3.1 Input smoothing

We call a classifier "input smooth" when its predictions are robust to small perturbations in the input space. So, input smoothing can be enforced with the following optimization problem:

$$\begin{aligned} &\min_{w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i + \Delta x_i; w)) \\ &\text{subject to } \Delta x_i = \underset{||\Delta x_i|| < \epsilon_x}{\arg\max}\, \ell(f(x_i; w), f(x_i + \Delta x_i; w)) \\ &\forall x_i \in X \end{aligned} \tag{3.1}$$

where $\ell(\cdot)$ can be cross-entropy, Kullbach-Liebler (KL) divergence or the mean-square error. $f(x; w) \in \mathbb{R}^K$ is the network output with weights w and K is the number of classes. This problem can be solved along with minimizing the objective function designed for the task, for instance the cross-entropy loss for classification. In other words, a desirable classifier should not change its predictions for any local perturbation within a ball of small radius ϵ_x for any input x_i. This idea is also known as max-margin or low-density assumptions in the SSL literature, championed by TSVM [Joa99]. Although the perturbations to which we seek insensitivity are unstructured, in imaging data the largest perturbations are often due to structured nuisance variability (*e.g.*. changes in illumination, vantage point, or visibility).

A popular way of attacking this min-max problem is through the use of adversarial examples. The underlying idea is to add a (regularization) term to the loss function, that penalizes the difference

between network outputs for clean samples, and samples with added adversarial noise. Adversarial training [GSS14] applies this idea to supervised learning where they change the problem to being robust against noise by moving predictions away from the ground truth labels:

$$\min_{w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i + \Delta x_i; w))$$
$$\text{subject to } \Delta x_i = \arg\max_{||\Delta x_i|| < \epsilon_x} \ell(P(y_i|x_i), f(x_i + \Delta x_i; w))$$
$$\forall x_i \in X \tag{3.2}$$

For this supervised setting, ground truth labels $P(y|x)$ can be used in calculating the adversarial noise Δx. Instead of finding the exact Δx for each input x, [GSS14] calculates the first-order approximation of adversarial perturbation leading to maximum change in the classifier predictions $f(x; w)$:

$$\Delta x \approx \epsilon_x \frac{g}{||g||_2}$$
$$\text{subject to } g = \nabla_x \ell(P(y|x), f(x; w)) \tag{3.3}$$

where $P(y|x)$ is the ground truth label for sample x. A natural extension of this idea to SSL is introduced by [MMK15, MMK17]. Since SSL algorithms do not have access to ground truth labels $P(y|x)$, their adversarial noise attempts to maximize $\ell(f(x; w), f(x+\Delta x; w))$. But, in the SSL case, the first-order approximation is not useful, because the first derivative of $\ell(f(x; w), f(x + \Delta x; w))$ is always zero at $\Delta x = 0$. Hence, [MMK15, MMK17] make a second-order approximation for Δx and propose the following approximation to the adversarial noise for each input x:

$$\Delta x \approx \epsilon_x \frac{g}{||g||_2}$$
$$\text{subject to } g = \nabla_{\Delta x} \ell(f(x; w), f(x + \Delta x; w))\Big|_{\Delta x = \xi d} \tag{3.4}$$

where $d \sim N(0, 1)$. Therefore, the regularization loss of [MMK15, MMK17] is

$$\ell_{VAT}(x; w) := \ell(f(x; w), f(x + \epsilon_x \frac{g}{||g||_2}; w))$$
$$\text{subject to } g = \nabla_{\Delta x} \ell(f(x; w), f(x + \Delta x; w))\Big|_{\Delta x = \xi d} \tag{3.5}$$

for one input sample x. We will minimize this regularizer as a way of doing input smoothing in our final SSL algorithm.

3.3.2 Weight smoothing

Just like input smoothing, weight smoothing can be formulated as a min-max problem. More explicitly,

$$\min_{w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i; w + \Delta w))$$
$$\text{subject to } \Delta w = \arg\max_{||\Delta w|| < \epsilon_w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i; w + \Delta w)) \tag{3.6}$$

Unlike input smoothing, the parameters of both minimization and maximization *are the same*, namely the weights of the network, w. It is important to note that the maximum is taken within a ball of small radius ϵ_w. This appears counter-intuitive at first.

Just like input smoothing can be associated with insensitivity to nuisance variability in the data (hence bias the representation towards invariance), weight smoothing can be associated with generalization, as it has been observed empirically that so-called "flat-minima" [HS97, JKA17] correspond to solutions that tend to yield better generalization. Recent methods [CCS16] try to bias solutions towards such flat minima, including using a conservative penalty [LZC14]:

$$w_t = \arg\min_{w} \ell(P(y|x); f(x; w)) + \gamma ||w - w_{t-1}||_2^2. \tag{3.7}$$

We follow a related, but different, approach: We explicitly find adversarial directions *with respect to random subsets of the weights,* then force the network to be robust against these directed perturbation using the remaining weights.

This problem of optimizing for the worst case is also studied in the robust optimization literature [BNT10]. Given "all" the possible adversarial Δw values maximizing the loss, they suggest an optimization framework for finding descent directions with second-order cone programming (SOCP) which would guarantee an optimal solution for a convex objective. Since finding all possible adversarial perturbations is not feasible, they find the ones around a ball with gradient ascent and solve SOCP for local solutions iteratively, which is related to our method.

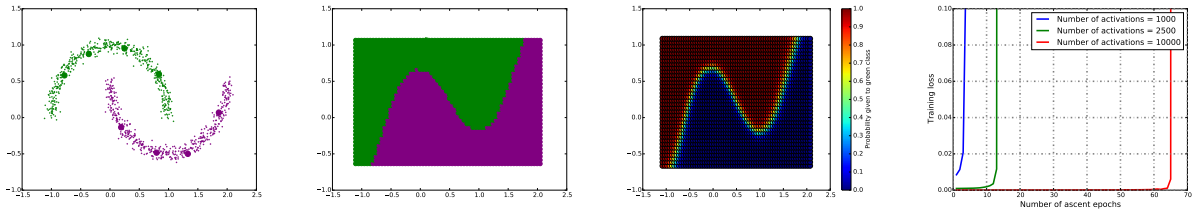


Figure 3.1: **Over-parametrized networks are more robust to adversarial noise in the weight space even when they have the same decision boundary (i.e. the same input smoothness).** Three MLP networks with different number of hidden units trained with VAT on the half-moon dataset (first panel) have the same decision boundary (second panel). Moreover, the network response (probability given to one of the classes) is almost identical for each of the three networks (third panel). Therefore, the robustness of the networks to input perturbation is not just the same in the "error" sense but also the same in the "loss" sense. However, the larger the network, the more robust it is to adversarial perturbations in weight space. The fourth panel shows the training loss versus the number of gradient ascent directions for varying sized networks. As can be seen, having a visible increase in the loss takes more ascent steps for larger networks. This experiment illustrates that networks with the same robustness to input perturbations may have completely different sensitivity to perturbations in the weight space.

3.3.3 Input smoothing and weight smoothing do not imply each other

For a linear classifier, the Hessian of the mean-square error (MSE) loss is the data covariance matrix. The local geometry of the loss landscape around the solution where the weights converged does not depend on the classifier structure, nor on its parametrization. The number of zero eigenvalues is determined by the dimensionality of the input data matrix alone. However, it can be easily shown that this is not necessarily the case for nonlinear classifiers. For instance, [SEG17] shows that the number of zero eigenvalues of the approximate Hessian of the loss has a lower bound of $|w| - N$, where N is the number of training samples and $|w|$ is the number of weights. This simple Hessian argument suggests that the robustness of a network to weight perturbations depends on factors other than its robustness to input perturbations, like the number of parameters in the network, for nonlinear classifiers. Even though they also have empirical evidence for this claim, the spectrum of the Hessian alone is not necessarily a good measure of flatness [DPB17]. So, we construct a counterexample to verify that input smoothness does not necessarily imply weight smoothness.

We use a half-moon toy dataset whereby there are 4 labeled (larger circles) and 1000 unlabeled samples from each cluster (first panel of Fig. 3.1). We run the VAT algorithm to find the max-margin decision boundary (second panel of Fig. 3.1). We repeat this experiment for 3 multi-layer perceptron (MLP) networks each having the same structure [1], except for a different number of weights. The decision boundary is the same, as can be seen in the second panel of Fig. 3.1, where they show as one as they overlap perfectly. This implies that for any perturbation in the input space, the increase in the error would be the same for each of these classifiers. To show that this also the case for the loss, we also provide the network response and again they are indistinguishable (third panel of Fig. 3.1). Hence, input smoothness is the same for each of the three MLPs.

After training three MLP networks with VAT, we apply gradient ascent on the converged weights with a small learning rate to evaluate the robustness of the networks with different number of weights. As can be seen in the right panel of Fig. 3.1, it takes more ascent updates for large

[1] By "the same structure" we mean the same number and type of layers with different number of filters. So networks have the same depth, but different widths.

networks to diverge.[2] Thus, over-parametrized networks are more robust to perturbations in weight space. This experiment shows that input smoothing does not necessarily imply weight smoothing. That is, there are factors other than input smoothness determining the geometry of the loss around the converged weight. Another observation is that losses diverge suddenly, implying that it takes several iterations to increase the loss, during which time the landscape is almost constant before the loss increases sharply.[3] The ascent learning rate chosen in the experiment is 0.005. During the ascent, SGD without momentum and weight decay is used. MLPs are in the form $2 \to FC(n) \to FC(n) \to FC(n) \to FC(n) \to 2$ where $FC(n)$ is fully connected layer followed by a RELU and n is number of hidden units. We report results for $n = 1000$, $n = 2500$ and $n = 10000$.

Note that we choose to make our experiments in two-dimensional inputs to visualize the decision boundary easily and verify that they are the same for different classifiers; not because these results are restricted to small dimensional inputs. These results align with those of [FB16] where they suggest that the amount of uphill climbing for connecting arbitrary weight pairs is correlated to the size of the network. For more discussion on the effect of over-parametrization on the loss landscape of DNNs see [SS16, SGA14, BSG18, SC16].

[2]Number of small ascent steps it takes for loss to diverge is not the only measure for the flatness of the converged minima. In Sec. 3.5.2, the same experiment is repeated with different measures. For instance, by plotting the change in the loss for various values of one ascent step norm.

[3]This plot implies that for very large networks and small dimensional inputs, reaching a considerably high loss level set may require many ascent iterations. This might seem counter-intuitive when we consider a typical Hessian histogram of a deep network weights as there are usually few large positive eigenvalues. But, it is important to note that the MLP networks we use in this experiment have 4 hidden layers and the largest of them has 10000 units per layer. So, the size of the network we use for this experiment is much larger than those for which Hessian histograms can be calculated. Unfortunately, calculating the Hessian of such a large network is very expensive computationally. Similarly, it can take many descent iterations to reach a low-level loss if the data is too complex for the model. For instance, when training with random labels [CFS18a], it takes many epochs to reduce the loss level even though the loss goes to zero eventually [ZBH16].

3.3.4 Joint input and weight smoothing

Once established that input smoothness and weight smoothness are not equivalent, it is natural to try enforcing both. So, the additional regularization we want to have can be framed as follows:[4]

$$\begin{aligned}
&\min_{w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i; w + \Delta w)) + \\
&\qquad \ell(f(x_i; w), f(x_i + \Delta x_i; w)) \\
&\text{subject to } \Delta w = \arg\max_{||\Delta w|| < \epsilon_w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i; w + \Delta w)) \\
&\qquad \Delta x_i = \arg\max_{||\Delta x_i|| < \epsilon_x} \ell(f(x_i; w), f(x_i + \Delta x_i; w)) \\
&\qquad \forall x_i \in X
\end{aligned} \tag{3.8}$$

So, the regularization term we want to minimize for one input x is

$$\begin{aligned}
L(x; w) = \ell(&f(x; w), \\
&f(x; w + \arg\max_{||\Delta w|| < \epsilon_w} \sum_{x_i \in X} \ell(f(x_i; w), f(x_i; w + \Delta w)))) \\
&+ \ell(f(x; w), f(x + \arg\max_{||\Delta x|| < \epsilon_x} \ell(f(x; w), f(x + \Delta x; w)); w))
\end{aligned} \tag{3.9}$$

Then, the overall problem we want to solve in the supervised learning setting becomes

$$\min_{w} \sum_{x} \ell_{CE}(x; w) + \lambda L(x; w) \tag{3.10}$$

where

$$\ell_{CE}(x; w) = -\langle P(y|x), \log f(x; w) \rangle \tag{3.11}$$

is the cross entropy loss calculated for label estimates $f(x; w)$ and ground truth labels $P(y|x)$.

Calculating the exact Δx for each input is not trivial. Instead, we will use VAT in Eqn. 3.4 to make our classifier robust against input space perturbations. For achieving weight space smoothness, we will use the proposed algorithm described next.

[4]Note that here we take each smoothness term separately. I.e. adversarial perturbations in weight space Δw are calculated for the original images x; not for perturbed images $x + \Delta x$. Even though it is possible to formulate the problem such that perturbations in the weight space (Δw) are functions of perturbations in the input space (Δx), the problem would be even more complicated in that case.

3.4 Adversarial Block Coordinate Descent (ABCD)

Algorithm 2 Adversarial Block Coordinate Descent (ABCD)

Input: Minibatch set B_t, loss function $\ell(\cdot)$, initial weights w_0.

Hyper-parameters: Ascent and descent learning rates η_A and η_D. Number of inner iterations L.

Outputs: Final weights w_L.

for $l = 1 : L$ **do**

  Γ_i sample from $\{0, -1\}$ for all $i \in \{1, \ldots, |w_0|\}$.

  $\Gamma_i^a = \Gamma_i$ for all $i \in \{1, \ldots, |w_0|\}$.

  $\Gamma_i^d = \Gamma_i + 1$ for all $i \in \{1, \ldots, |w_0|\}$.

  // Run stochastic gradient *ascent* with a *small* learning rate η_A

  $w_{l-\frac{1}{2}} = w_{l-1} - \eta_A \Gamma^a \odot \nabla_{w_{l-1}} \left(\frac{1}{|B_t|} \sum_{i=1}^{|B_t|} \ell(x_i; w_{l-1}) \right)$

  // Run stochastic gradient *descent* with a learning rate $\eta_D \gg \eta_A$

  $w_l = w_{l-\frac{1}{2}} - \eta_D \Gamma^d \odot \nabla_{w_{l-\frac{1}{2}}} \left(\frac{1}{|B_t|} \sum_{i=1}^{|B_t|} \ell(x_i; w_{l-\frac{1}{2}}) \right)$

Since the parameters of both minimization and maximization are the weights of the network for the min-max problem in Eqn. 3.6, we use a subset of the weights w for finding adversarial directions in weight space and the rest to impose robust to such additive adversarial perturbations in weight space. For this, at each update, we randomly choose half of the weights and take a gradient *ascent* step on them with a *small* learning rate. Then, on the same batch, we apply gradient descent for the remaining weights with an ordinary (larger) learning rate. We call this algorithm Adversarial Block Coordinate Descent (ABCD), for pseudo-code, see Alg. 2. ABCD can be considered as an extension of Dropout and Dropconnect [HSK12, SHK14, WZZ13] and coordinate descent [Wri15]. However, unlike Dropout and Dropconnect, we explicitly regularize our network to be robust against "adversarial directions" in the weight space; instead of just being robust to zeroed out weights or activations.

ABCD can be used for SSL in place of vanilla SGD. We use ABCD for SSL by minimizing the empirical cross entropy for labeled data and the entropy of empirical estimates for the unlabeled

data with ABCD. That is,

$$\ell_E(x; w) = -\langle f(x; w), \log f(x; w) \rangle \tag{3.12}$$

for unlabeled data which is a well-known regularizer in the SSL literature [DYY17, GB05, KPG10, Spr15].

Algorithm 3 SSL algorithm using ABCD as optimizer; VAT and entropy as regularizers. $\ell_{CE}(x; w)$, $\ell_E(x; w)$, $\ell_{VAT}(x; w)$ are as defined in Eqn. 3.11, Eqn. 3.12, Eqn. 3.5 respectively.

for $t = 1 : T$ **do**
  // Run ABCD on cross entropy for weight smoothing:
  Sample B_t^l // labeled samples
  $w_{t'} = ABCD(B_t^l, \ell_{CE}(x; w), w_{t-1})$
  // Run ABCD on entropy for weight smoothing:
  Sample B_t^u // unlabeled samples
  $w_{t'} = ABCD(B_t^u, \ell_E(x; w), w_{t'})$
  // Run SGD on VAT loss for input smoothing:
  $w_t = SGD(B_t^u, \ell_{VAT}(x; w), w_{t'})$

The randomness in ABCD is due to mini-batch optimization that we have to use for computational reasons and the randomness in the mask Γ selection. If we ignore these, it would be easier to see the loss minimized by ABCD. The loss minimized by descent in ABCD is the "worst-case" of the nominal loss landscape. By worst case, we mean at each point in the weight space, the loss ABCD minimizes is the maximum that can be reached from that point with one small ascent step. In other words, ABCD minimizes the maximum loss around a small ball at each point.

Finding the minimum norm adversarial perturbation defined in Eqn. 3.6 is not possible with one ascent step as this problem is highly non-convex. Therefore, we take $L > 1$ steps for each mini-batch. Taking multiple ascent steps does not guarantee to find the minimum-norm adversarial perturbation, but empirically we find that it gives better results. Moreover, we do not want the last update for any of the weight parameters to be ascent. So, we do not apply ABCD in the last few epochs.

Also, note that the coordinate descent mask is essential for ABCD to work as intended. Assume that true adversarial perturbation Δw in Eqn. 3.6 is given. Then, for minimizing the objective in Eqn. 3.6 locally is all we need to do is apply the original SGD update. This is because the first-order approximation of Δw is just the negative direction of the gradient given by the SGD. So, without block coordinate descent masks, the result of adversarial training in weight space would only be the change of effective learning rate. That is why we have dropout-like masks where half of the weights are trained to be robust when others add adversarial noise. Moreover, instead of a first-order method, using a second-order approximation like VAT in finding Δw for half of the weights and minimizing the objective in Eqn. 3.6 with the rest is possible. But, we choose to calculate Δw with the first-order gradients for faster training. This is done by ascending in cross-entropy and entropy for labeled and unlabeled cases respectively.

We report the performance of ABCD combined with VAT to see the effect of applying both input and weight smoothing. The pseudo-code using VAT as loss function and ABCD as an optimizer for SSL task is given in Alg. 3. For labeled data, ABCD is used as an optimizer to minimize cross-entropy to achieve weight smoothing. We do not minimize $\ell_{VAT}(x; w)$ for labeled data as proposed in the corresponding paper. For unlabeled data, ABCD is used to minimize entropy $\ell_E(x; w)$ and SGD is used to minimize $\ell_{VAT}(x; w)$ for weight and input smoothing respectively. We set $\eta_A = 10^{-5}$ for all of our experiments. η_D can be chosen as usual with a high initial value from $\{0.1, 0.01\}$ and is decreased during training. In all the SSL experiments, we only use the network called *conv-large* from [MMK17, TV17]. Only translation and horizontal flipping are used as data augmentations to allow a fair comparison with some of the previous SSL algorithms. Horizontal flipping is only used in CIFAR10. Cosine learning schedule of [LH16] is used for annealing the learning rate and momentum. Initial learning rates for SVHN and CIFAR10 are 0.1 and 0.04 respectively. We decrease the learning rate up to the final learning rate of $\eta_D = 10^{-4}$.

3.5 Evaluation

3.5.1 Performance of ABCD in SSL Benchmarks

In Table 3.1 and Table 3.2, we compare the performance of ABCD with state-of-the-art SSL algorithms. We report performance of the proposed algorithm on SVHN [NWC11] and CIFAR10 [KH09] in SSL setting. SVHN consists of 32×32 images of house numbers. We use $73,257$ samples for training, rather than the entire $600,000$ images; $26,032$ images are separated for evaluation. CIFAR10 has $60,000$ 32×32 images, of which $50,000$ are used for training and $10,000$ for testing. We choose labeled samples randomly. We also choose them to be uniform over the classes as it is done in previous works [MMK17]. In CIFAR10, $4,000$ and in SVHN $1,000$ of training samples are labeled. Except [SJT16b], all the methods use modest augmentations (translation and horizontal flipping) and do not exploit recent deep learning models like ResNet [HZR16a]. We report ABCD with entropy minimization alone and combined with VAT. In CIFAR10, when we run ABCD only with entropy minimization, it yields better performance than previous ensemble methods [LA16, TV17]. Combining ABCD with VAT improves the scores in both datasets verifying that they are complementary.

In our implementation of VAT [MMK17], we set ϵ_x in Eqn. 3.4 to be 128 for CIFAR10 and 0.25 for SVHN. Even though we search ϵ_x in a fine grid, we could not get to the performance reported in their paper for CIFAR10; possibly because we use a different optimizer (SGD instead of ADAM), different whitening and different learning rate scheme. The performance of our VAT implementation is given in Table 3.1 and Table 3.2. Our implementation (13.01%) of VAT without entropy minimization is about 1.5 percent worse than what is reported in [MMK17] (11.36%) for CIFAR10. But, we still use the numbers reported in [MMK17] for comparison purposes in Table 3.1 and Table 3.2.

3.5.2 Robustness of ABCD to weight perturbations

First, we report Hessian histograms to see the curvature of the point converged by ABCD. We calculate the histograms of Hessian spectra for random weights, SGD-trained weights, and ABCD-

SSL Method	Test error rate (%)
VAT+EntMin [MMK17]	10.55
Stochastic Transformation [SJT16b]	11.29
Temporal Ensemble [LA16]	12.16
GAN+FM [SGZ16]	15.59
Mean Teacher [TV17]	12.31
VAdD [PPS17]	**9.22**
EntMin (*)	15.29 ± 0.23
VAT without EntMin (*)	13.01 ± 0.14
VAT+EntMin (*)	11.63 ± 0.12
ABCD+EntMin	11.96 ± 0.06
ABCD+EntMin+VAT	9.28 ± 0.21

Table 3.1: Comparison with the state-of-the-art on CIFAR10 SSL task. Error rates on the test set are given for CIFAR10. CIFAR10 is trained using 4,000 labeled and 46,000 unlabeled samples. Results are averaged over three random labeled sets. We report the performance of ABCD alone and combined with VAT. ABCD+EntMin+VAT refers to the algorithm in Alg. 3 where ABCD is used as an optimizer; entropy and VAT are used as regularizers in the loss function. ABCD+EntMin uses only entropy for unlabeled data to report performance of ABCD without VAT. SSL baselines. Baseline algorithms are EntMin, VAT and VAT+EntMin. EntMin minimizes the entropy of estimates for unlabeled data with standard SGD. Similarly, VAT minimizes ℓ_{VAT} from Eqn. 3.5 and VAT+EntMin minimizes both on unlabeled data. Note that (*) means our own implementation.

SSL Method	Test error rate (%)
VAT+EntMin [MMK17]	3.86
Stochastic Transformation [SJT16b]	NR
Temporal Ensemble [LA16]	4.42
GAN+FM [SGZ16]	5.88
Mean Teacher [TV17]	3.95
VAdD [PPS17]	3.55
EntMin (*)	5.68 ± 0.03
VAT without EntMin (*)	5.36 ± 0.22
VAT+EntMin (*)	4.01 ± 0.07
ABCD+EntMin	4.52 ± 0.15
ABCD+EntMin+VAT	$\mathbf{3.53 \pm 0.24}$

Table 3.2: Comparison with the state-of-the-art on SVHN SSL task. Same as Table 3.1 except results are given for SVHN. NR stands for "not reported." SVHN is trained using 1,000 labeled and 72,257 unlabeled samples. Proposed algorithm achieves the state-of-the-art performance on SVHN SSL task.

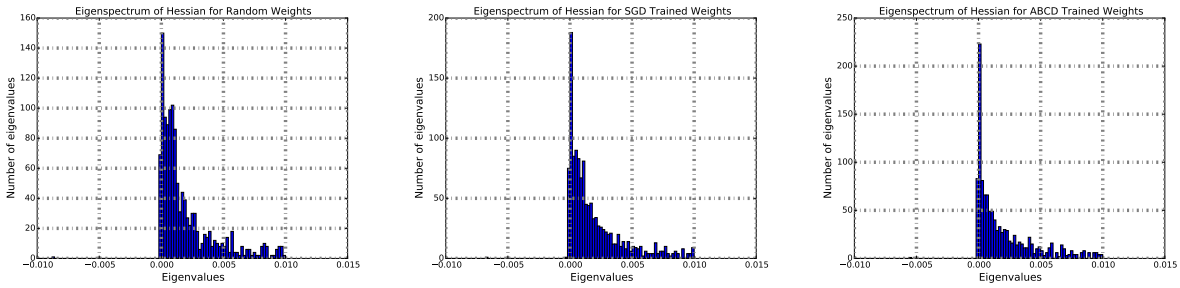


Figure 3.2: **Hessian spectra** of 1 hidden layer convolutional network with 1476 parameters for the loss calculated in MNIST. Left panel is for random weights, middle panel is for SGD trained weights and the right panel is for ABCD trained weights. The number of almost-zero eigenvalues is maximum for ABCD-trained networks. Histograms are cropped for eigenvalues in $[-0.01, 0.01]$.

trained weights in Fig. 3.2. We use the automatic differentiation package of PyTorch [PGC17] for calculating Hessians. The number of eigenvalues smaller than 10^{-4} for random weights, SGD trained weights and ABCD weights are $185, 226, 262$ respectively. So, the number of almost-zero eigenvalues for the ABCD-trained weight network is larger than that of SGD-trained weights. This implies that ABCD converges to wider minima than SGD. A small convolutional network with one hidden layer is trained on the MNIST dataset for this experiment due to computational constraints.

To evaluate the robustness of ABCD-trained weights to adversarial weight space perturbations, we report the number of ascent updates necessary for the loss to diverge. In Fig. 3.3 (left), the plot of training loss v.s. the number of ascent updates is given for SGD and ABCD-trained weights. As can be seen, the number of ascent updates needed for the loss to diverge for ABCD is more than for SGD.

Another way of comparing the local geometrical properties of the weights is by visualizing the loss landscape around the converged weights. As the deep networks we use have very high dimension, several visualization tricks have been suggested to visualize the loss landscapes in 1- or 2-dimensional subsets of the weight space [LXT17, GVS14, KMN16]. We employ the technique suggested in [GVS14] in Fig. 3.3 (right): we plot the loss on the curve $\alpha * w_{SGD} + (1-\alpha) * w_{ABCD}$ where $\alpha \in [-0.2, 1.2]$. w_{SGD} and w_{ABCD} are the weights converged when training with SGD and ABCD respectively. As can be seen, the loss does not increase around ABCD trained weights. To

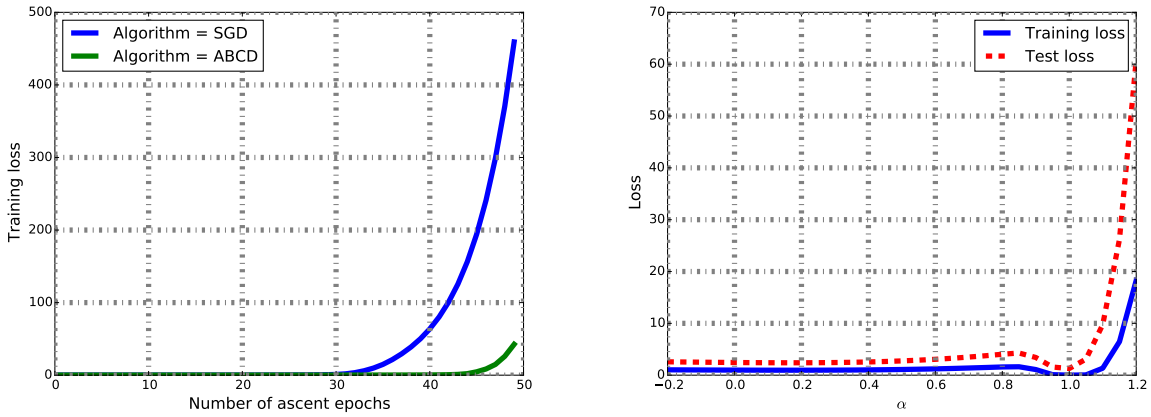


Figure 3.3: **Robustness of ABCD vs SGD trained networks to ascent updates**. Starting from the ABCD and SGD trained weights, we apply gradient ascent with the learning rate 0.001 on the cross-entropy loss. This plot shows the increase in the training loss versus the number of ascent steps. Both weights diverge quickly, but ABCD trained network diverges later than that of SGD verifying the robustness of ABCD to weight space perturbations. **1D visualization of loss landscapes of SGD and ABCD trained weights**. Training and test losses over the curve $\alpha * w_{SGD} + (1-\alpha) * w_{ABCD}$ are given as suggested by [GVS14] where $\alpha \in [-0.2, 1.2]$. $\alpha = 0$ and $\alpha = 1$ correspond to the weights of ABCD and SGD trained networks respectively. This method compares the flatness of two methods in one direction only and in that direction, ABCD seems much more robust to weight perturbations.

be consistent with Hessian experiments, these experiments are also conducted on MNIST dataset with the same network.

In the toy example of Fig. 3.1, we compared the robustness of different sized networks to weight perturbations. Now, we introduce ABCD, we can compare VAT-trained networks with VAT+ABCD trained networks in the same setting. As can be seen in Fig. 3.4, VAT+ABCD trained network diverges later than VAT-trained network for two different sizes of networks. We do not plot ABCD because ABCD alone is not enough to converge to 0 training loss. This is because perfect classification in the SSL setting of half-moon clusters requires a regularizer encouraging input smoothing. Since ABCD and VAT trained networks are not in the same loss level, weight smoothness comparison between them is not straightforward. Also, note that the experiment in

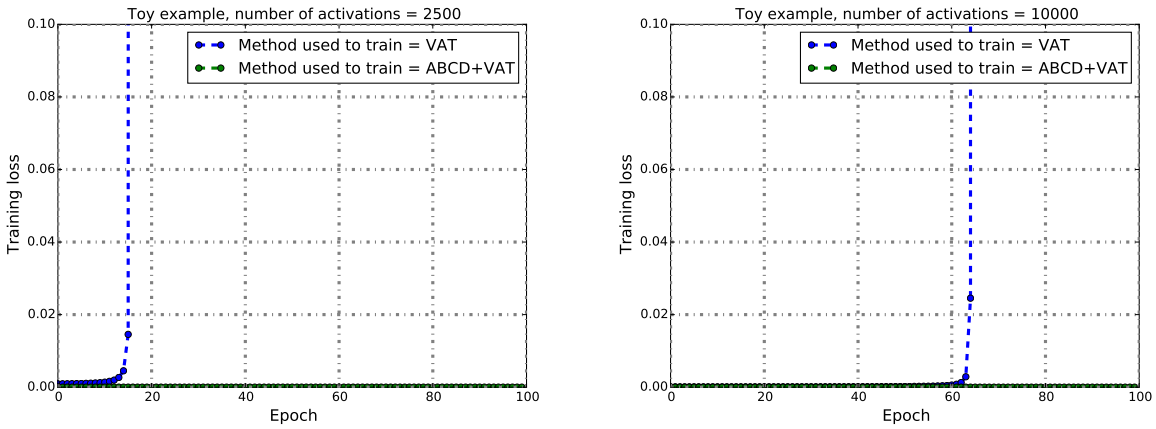


Figure 3.4: **ABCD vs VAT+ABCD-trained networks for the toy example in Fig. 3.1:** VAT+ABCD trained network is much more robust to weight perturbations than VAT-trained network for all sizes of networks. Here, we only report results for two networks of the same type: one with 2500 filters (left), another with 10000 filters (right).

Fig. 3.1 is in the SSL setting where there are 4 labeled samples from each class. Since there is no way for SGD to exploit the unlabeled data, we do not compare against SGD.

In Fig. 3.1 and Fig. 3.3-left, we measure the robustness by comparing the losses for the increasing number of small ascent steps. In Fig. 3.5-left, we repeat the experiment in Fig. 3.1 with a different criterion for measuring the robustness of algorithms to weight perturbations. This time, we plot the loss after one ascent step for various ascent learning rates. When we compare losses for different learning rates of one ascent step, over-parametrized networks are again more robust. Similarly, in Fig. 3.5-right, we repeat Fig. 3.3-left with this criterion and again ABCD trained network is more robust to weight perturbations than SGD. So, whether we take the number of ascent steps or the length of one ascent step to diverge as a measure of flatness, ABCD-trained networks systematically find wider solutions.

3.6 Training Details

In the CIFAR10 task, we apply ZCA. Before applying ZCA, we standardize the dataset. ZCA transformation is $D_{\text{ZCA}} = DU(S^{-0.5} + \epsilon I)U^T$ where S and U are eigenvalue and eigenvector

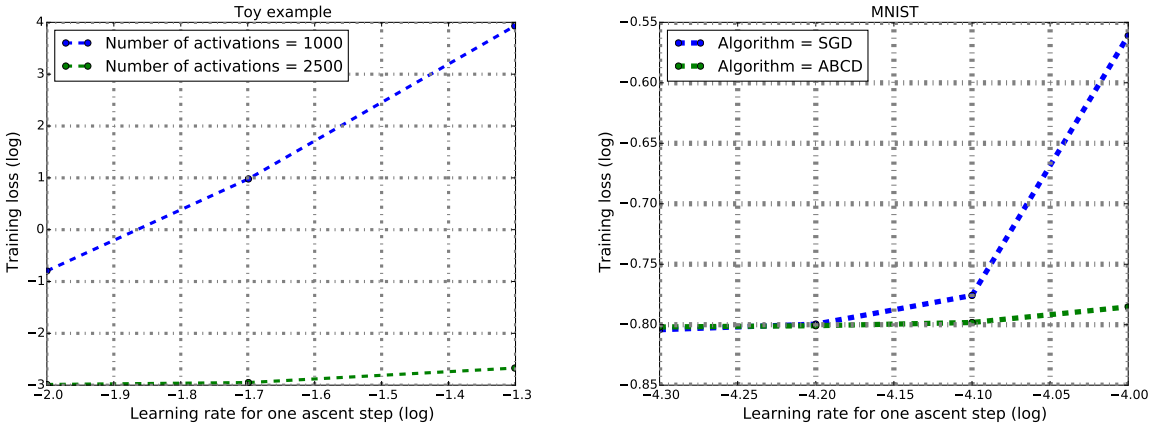


Figure 3.5: **Different robustness criteria for the experiments in Fig. 3.1 and Fig. 3.3-left.** Here, we plot loss vs. learning rate of one ascent step. Over-parameterized networks are again more robust to perturbations in the toy example (left). ABCD trained network in Fig. 3.3-left is also more robust than SGD trained network (right).

matrices of covariance of data matrix D respectively. We set ϵ to 500.

As data augmentation, we only use translation and horizontal flipping as in [MMK17, TV17]. Horizontal flipping is not used for SVHN. Translation size is uniformly randomly sampled from $\{1, 2\}$. As network, *conv-large* in Table 2.1 from [MMK17, TV17] is used in all the SSL experiments unless otherwise stated. There is a batch-norm layer after each convolutional layer.

We use batchsize of 128 for unlabeled data and 32 for labeled data in SSL experiments. For test data and supervised experiments, we use batchsize of 100. We train CIFAR10 and SVHN for 500 and 300 epochs respectively. In all the experiments, SGD is used as an optimizer with weight decay of 10^{-4}.

In ABCD, we try different ratios of ascent and descent directions, we observe that 0.5 is the best and set it to 0.5 in all the experiments. So, we apply gradient ascent with random half of weights and gradient descent for the rest. We set the number of ascent and descent iterations L in ABCD to be 2 for CIFAR10 and SVHN SSL experiments when we combine ABCD with VAT. It is possible to get better results with larger L values with the cost of more training time, but we choose a small number for relatively faster training. We do not apply ABCD in the last few epochs to not have ascent as the last iteration in any of the weights. Actually, we realize that applying ABCD in the

Fully connected layer: $2 \rightarrow$ n, n ReLU
Fully connected layer: n $\rightarrow$ n, n ReLU
Fully connected layer: n $\rightarrow$ n, n ReLU
Fully connected layer: n $\rightarrow$ 2
Softmax

Table 3.3: The network used in the toy example. $n \in \{1000, 2500, 10000\}$ is the number of filters at each layer.

first epochs are much more crucial than applying it later. Hence, we choose not to apply ABCD in the last 50 epochs. When applying ABCD without VAT, we set the number of ascent and descent iterations L to be 5 and 2 for CIFAR10 and SVHN respectively.

The network used in the toy example (See Fig. 3.1) is given in Table 3.3. [FFF18] observes that robustness of the networks to adversarial perturbations in input space increases as the depth of the network increases. But, robustness barely changes for different number of filters suggesting that the width of the network does not contribute to the robustness of the network to the input space perturbations. Thus, we choose to vary the number of filters (width of the network) instead of the depth of the network to get the same input smoothness across different sized networks for the toy example.

For Hessian experiments, we use a small, 1-layer network due to computational constraints. The network used is $Conv(1, 5) \rightarrow MP(2) \rightarrow FC(144, 10)$ where $Conv(1, 5)$ is the convolution layer with 1 channel and kernel size 5. $MP(2)$ is a max-pooling layer of size 2.

3.7 Discussion and Conclusion

3.7.1 Theoretical Grounding

Our algorithm yields a function (discriminant) $f(x;w)$ that maps a (test) datum x onto a class label y. More precisely, f maps x onto a vector $f(x;w) \in \mathbb{R}^K$ with K the number of labels, which can be evaluated for each class hypothesis $y = k \in \{1, \dots, K\}$ to yield a positive real number $f(x;w)[k]$ where $[k]$ denotes the k-th component. Ideally, such a discriminant would be the posterior probability $f(x;w)[k] = P(y = k|x)$, which is the (Bayesian) optimal discriminant. If we could compute and minimize the *expected* cross-entropy loss $\ell_{CE}(x;w)$ within a universally approximating family of functions $f(\cdot;w)$, we would obtain just that. Unfortunately, we only have a finite sample of the true distribution $D \doteq \{(x,y) \sim p(x,y)\}_{i=1}^N$, from which we can only compute and minimize the *empirical* cross entropy $\ell_{CE}^D(x;w)$. Nevertheless, we can construct a proxy loss that captures the properties of the optimal discriminant even with a finite sample, as we discuss next.

The posterior $P(y|x)$ is a function of the test datum that is *minimal, sufficient and invariant* [SC14]. It is sufficient (informationally equivalent to the data) for the task of classification; it is invariant to nuisance variability in the data; and it is minimal in the sense of having smallest (information) complexity, which relates to generalization [AS17]. On the other hand, the learned discriminant $f(\cdot;w)$ is a function of the dataset D that does not know anything about the test datum x. If, however, we could construct f to be any minimal, sufficient, and invariant function of the *training set* D, constructed with a deep neural network, then the Emergence Theory of [AS17] guarantees that the resulting discriminant is also a minimal sufficient invariant of the test datum. The key question then is: *What is the loss function that is computable from the training set that would yield a minimal, sufficient invariant?*

Sufficiency is captured by empirical cross-entropy: Any function that minimizes it (possibly by overfitting) is a sufficient representation of the training set.

Invariance is captured by minimizing the sensitivity to nuisance variability in the data. This can be done by considering the (worst-case) perturbations that do not modify the class (hence they are,

by definition, nuisance factors). This is precisely the criterion we have used to *define* input-space smoothing.

Minimality is measured *not* by the dimension of the weights, but by their *information content*. While in the Emergence theory this is measured in the sense of Shannon [AS17], measuring information in the sense of Fisher yields, under suitable approximation, the second-order criterion. As we verify with our Hessian histograms, the proposed method finds flat minima in this sense.

Thus, although derived heuristically, the loss function we construct by imposing both input and space smoothing, in addition to the small empirical loss, captures precisely the three defining properties of the optimal (Bayesian) discriminant, grounded in recent theoretical work [AS17]. Of course, the approximation of the optimal discriminant is only as good as the given data, so there is no (finite-sample) guarantee on the approximation of the posterior, but at least the loss function we adopt better captures the properties of the posterior than the raw empirical loss, which affords no invariance and no minimality.

3.7.2 Conclusion

We establish with a toy example that input and weight smoothing are complementary. In SSL benchmarks, combining input and weight smoothing algorithms resulted in a performance better than applying either algorithms alone. This further verifies that smoothing in input and weight spaces are complementary for high dimensional data as well. As a result, the proposed algorithm achieves competitive results in semi-supervised learning benchmarks. Furthermore, we conduct extensive experiments proving the robustness of the proposed algorithm to the adversarial perturbations in weight space.

CHAPTER 4

Unsupervised Domain Adaptation via Regularized Conditional Alignment

4.1 Introduction

In unsupervised domain adaptation (UDA), one is given annotated images from a *source* domain (e.g., hand-written digits), and images from a *target* domain (e.g., house numbers) with no annotations. The goal is to leverage annotated samples in the source domain to solve a task in the target domain, for example, to classify images into, numbers from 0 to 9.

As mentioned in Chapter 1, marginal alignment [GL14] does not guarantee a successful transfer, for it is possible that the source (say synthetic images) be perfectly aligned with the target (say natural images), and yet a natural image of a cat map to a synthetic image of a dog. To address these problems, we propose a method to perform the alignment of the joint distribution (Sec. 4.3.2). We employ ideas from semi-supervised learning (SSL) to improve generalization performance (Sec. 4.3.3). We propose an optimization scheme that uses a two-folded label space. The resulting method performs at the state of the art (Sec. 4.4). Finally, we analyze the proposed objective function in the supervised setting and prove that the optimal solution conditionally aligns the distributions while keeping them discriminative (Sec. 4.5). Next, we discuss our contribution in relation to the vast and growing literature on UDA (Sec. 4.2).

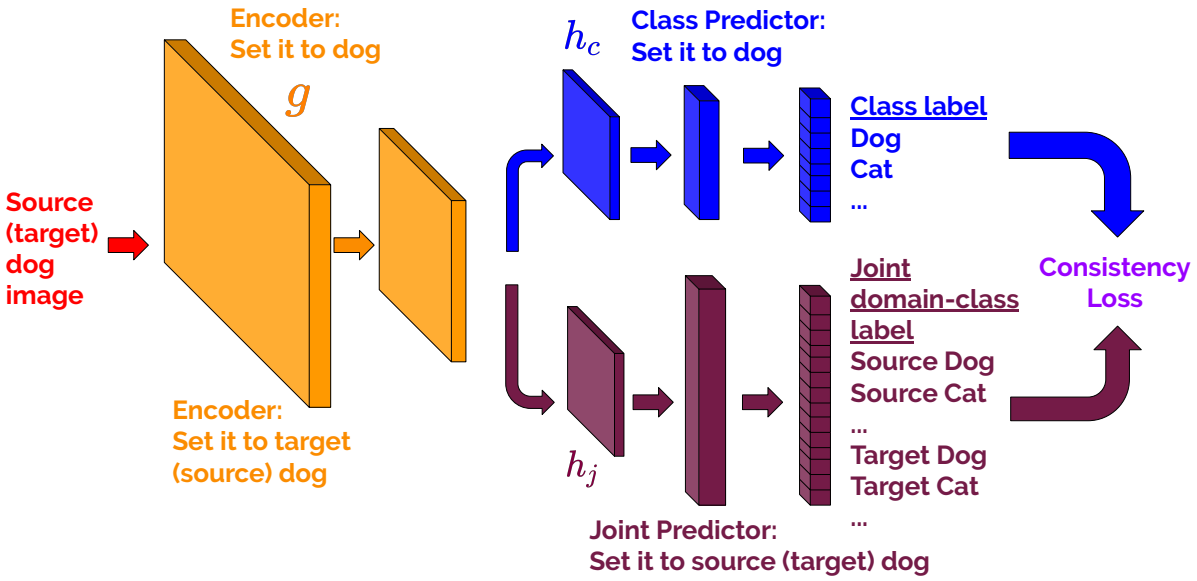


Figure 4.1: The network structure of the proposed approach. We propose to learn a joint distribution $P(d, y)$ over domain label d and class label y by a joint predictor (purple). The encoder (orange) is trained to confuse this joint predictor by matching the features corresponding to the same category samples of both domains. Since labels for the target data is not known, predictions of the class predictor (blue) on the target data is used with the help of consistency loss. Unlabeled data is further exploited with input smoothing algorithm VAT [MMK17] from the SSL literature.

4.2 Related Work

In this section, we will summarize the most relevant works from the UDA literature. For more in-depth coverage of the literature see the recent survey of [WD18] on deep domain adaptation for various vision tasks. Many of the domain adaptation works can be categorized into two: (1) the ones learning a shared feature space (symmetric-feature based) and the ones transferring features of one domain to another (asymmetric-feature based).

Shared feature (symmetric-feature based). Feature transferability drops in the higher layers of a network and there may not exist an optimal classifier for both the source and the target data. Hence many works use two separate classifiers for the source and the target domains while the encoder parameters are shared. In these works, the source classifier is trained with the labeled source data and the target classifier is regularized by minimizing a distance metric between the source classifier using all the data.

One common such metric is the (Maximum Mean Discrepancy) MMD which is a measurement of the divergence between two probability distributions from their samples by computing the distance of mean embeddings: $||\frac{1}{N^s}\sum_{i=1}^{N^s} g(x_i^s) - \frac{1}{N^t}\sum_{i=1}^{N^t} g(x_i^t)||$. DDC of [THZ14] applies MMD to the last layer while Deep Adaptation Network (DAN) of [LCW15] applies to the last 3 FC layers. CoGAN of [LT16] shares early layer parameters of the generator and later layer parameters of the discriminators instead of minimizing the MMD. [LZW16] models target classifier predictions as the sum of source classifier predictions and a learned residual function. Central Moment Discrepancy (CMD) of [ZGL17] extends MMD by matching higher moment statistics of the source and the target features.

Adversarial domain adaption methods described in the early sections [GL14] are another way of learning a shared feature space without needing separate classifiers for the source and the target data. DANN [GL14] proposed a shared encoder and two discriminator branches for the domain and class predictions. This makes *marginal* feature distributions similar for the domain classifier. Upcoming works [SBN18, KSW18] applied the same idea but instead of multiplying the gradient with a negative value, they optimize the discriminator and generator losses in an alternating fashion.

[SQZ17] suggested replacing the domain discrepancy loss with the Wasserstein distance to tackle gradient vanishing problems. The work of [LCW18] resembles ours where they also condition the domain alignment loss to labels. Unlike us, their domain discriminator takes the outer product of the features and the class predictions as input. Similarly, [CCC17] applies conditional domain alignment using K different class-conditioned binary predictors instead of one predictor with $2K$-way adversarial loss. Our approach allows to not only align the conditional push-forward distributions, but also encourage them to be disjoint. If our sole goal was to align the conditional distributions, a constant encoder function would be a trivial solution. Furthermore, these methods do not exploit SSL regularizers like VAT.

Mapping representations (asymmetric-feature based). These methods apply a transformation from the source domain to the target domain or vice-versa [BSD17]. Adaptive Batch Normalization (Ad-aBN) of [LWS16] proposed to map domain representations with first-order statistics. Before inference time, they pass all target samples through the network to learn the mean and the variance for each activation and apply these learned statistics to normalize the test instances. [SFS16] proposed Correlation Alignment (CORAL). They match the second-order statistics of the source data to the target by recoloring whitened source data with target statistics.

Exploiting unlabeled data with SSL regularizers. Given the features of the source and the target domains are aligned, standard SSL methods can be applied. [FMF18] employed the Mean Teacher [TV17] for UDA where the consistency loss on the target data between student and teacher networks is minimized. Even with extra tricks like confidence-thresholding and some data augmentation, the accuracy they achieved for MNIST→SVHN was 34%. This shows that, especially when domain discrepancy is high, SSL regularizers are not sufficient without first reducing the discrepancy.

Formalization

We are given N^s labeled source samples $x^s \in X^s$ with corresponding labels $y^s \in Y^s$ and N^t unlabeled target samples, $x^t \in X^t$. The entire training dataset X has cardinality $N = N^s + N^t$. Labeled source data and unlabeled target data are drawn from two different distributions (domain

shift): $(x^s, y^s) \sim P^s$, $(x^t, y^t) \sim P^t$ where their discrepancy, measured by Kullbach-Liebler's (KL) divergence, is $KL(P^s||P^t) > 0$ (covariate shift). Both distributions are defined on $X \times Y$ where $Y = \{1, ..., K\}$. Marginal distributions are defined on X and samples are drawn from them as $x^s \sim P_x^s$, $x^t \sim P_x^t$. Given finite samples $\{(x_i^s, y_i^s)\}_{i=1}^{N^s} := \{(x_1^s, y_1^s), (x_2^s, y_2^s), ..., (x_{N^s}^s, y_{N^s}^s)\}$ from P^s and $\{(x_i^t)\}_{i=1}^{N^t} := \{x_1^t, x_2^t, ..., x_{N^t}^t\}$ from P_x^t, the goal is to learn a classifier $f : X \to Y$ with a small risk in the target domain. This risk can be measured with cross-entropy:

$$\min_f E_{(x,y)\sim P^t} \ell_{CE}(f(x); y) \tag{4.1}$$

where

$$\ell_{CE}(f(x); y) := -\langle y, \log f(x) \rangle \tag{4.2}$$

is the cross-entropy loss calculated for one-hot, ground-truth labels $y \in \{0, 1\}^K$ and label estimates $f(x) \in \mathbb{R}^K$, which is the output of a deep neural network with input x, and K is the number of classes.

4.3 Proposed Method

In this section, we show how to formalize the criterion for aligning both inputs and outputs, despite the latter being unknown for the target classes in the absence of supervision.

Alignment of the marginal distributions can be done using Domain Adversarial Neural Networks (DANN) [GL14], that add to the standard classification loss for the source data a binary classification loss for the domain: Source vs. target. If all goes well, the class predictor classifies the *source* data correctly, and the binary-domain predictor is unable to tell the difference between the source and the target data. Therefore, the class predictor might also classify the target data correctly. Unfortunately, this is not guaranteed as there can be a misalignment of the output spaces that cause some class in the source map to a different class in the target, *e.g.*, a natural *cat* to a synthetic *dog*.

The *key idea* of our approach is to impose *not* a binary adversarial loss on the domain alignment, but a $2K$-way adversarial loss as if we had $2K$ possible classes: The first K are the known *source classes*, and the second K are the unknown *target classes*. We call the result a *joint domain-class*

predictor or *joint predictor* in short since it learns a distribution over domain and class variables. The encoder will try to fool the predictor by minimizing the classification loss between a dog sample in the source and a sample in the target domain whose predicted label in the aligned domain is also *dog*.

During training, the probabilities assigned to the first K labels of the joint predictor are very small for the target samples, and they eventually converge to zero. Therefore, we need a separate mechanism to provide pseudo-labels to the target samples to be aligned by the joint predictor. For this, we train another predictor, that we call *class predictor* outputting only class labels. The class predictor is trained on both the source data using ground-truth labels and the target data using semi-supervised learning (SSL) regularizers.

Both the joint predictor and the class predictor can be used for inference. However, we find that the class predictor performs slightly better. We conjecture this is because joint predictor is trained on a harder task of the domain and class prediction while only the latter one is needed at inference time.

We consider UDA as a two-fold problem. The first step deals with domain shift by aligning distributions in feature space. Given a successful alignment, one can use a source-only trained model for inference. But, once the domains are matched, it is possible to further improve generalization by acting on the label space. Ideas from SSL can help to that end [MMK17].

The overall architecture of the model is described in Fig. 4.1.

4.3.1 Network structure

We denote the shared encoder with g, the class predictor with h_c, the joint predictor with h_j and the overall networks as $f_c = h_c \circ g$ and $f_j = h_j \circ g$. Then, the class-predictor output for an input x can be written as,

$$f_c(x) = h_c(g(x)) \in \mathbb{R}^K. \tag{4.3}$$

Similarly, the joint-predictor output can be written as,

$$f_j(x) = h_j(g(x)) \in \mathbb{R}^{2K}. \tag{4.4}$$

4.3.2 Loss functions

The class predictor is the main component of the network which is used for inference. Its marginal features are aligned by the loss provided by the joint predictor. The class predictor is trained with the labeled source samples using the cross-entropy loss. This source classification loss can be written as,

$$L_{sc}(f_c) = E_{(x,y)\sim P^s} \ell_{CE}(f_c(x), y). \tag{4.5}$$

Both the encoder (g) and the class predictor (h_c) are updated while minimizing this loss.

We also update the joint predictor with the same classification loss for the labeled source samples. This time, only the joint-predictor (h_j) is updated. The joint-source classification loss is

$$L_{jsc}(h_j) = E_{(x,y)\sim P^s} \ell_{CE}(h_j(g(x)), [y, \mathbf{0}]) \tag{4.6}$$

where $\mathbf{0}$ is the zero vector of size K, chosen to make the last K joint probabilities zero for the source samples.

Similarly, the joint predictor is trained with target samples. As ground-truth labels for the target samples are not given, label estimates from the class predictors are used as pseudo-labels. The joint target classification loss is

$$L_{jtc}(h_j) = E_{x\sim P_x^t} \ell_{CE}(h_j(g(x)), [\mathbf{0}, \hat{y}]) \tag{4.7}$$

where $\hat{y} = e_k$ and $k = \arg\max_k f_c(x)[k] = \arg\max_k h_c(g(x))[k]$, e_k is the identity of size K whose kth element is 1.[1] Here, we assume that the source-only model achieves reasonable performance on the target domain (e.g. better than a chance). For experiments where the source-only trained model has poor performance initially, we apply this loss after the class predictor is trained for some time. Since the joint predictor is trained with the estimates of the class predictor on the target data, it can also be interpreted as a *student* of the class predictor.

The goal of introducing a joint predictor was to align label-conditioned feature distributions. For this, encoders are trained to fool the joint predictor as in [GL14]. Here, we apply conditional

[1] We use the notation $x[k]$ for indexing the value at the kth index of the vector x.

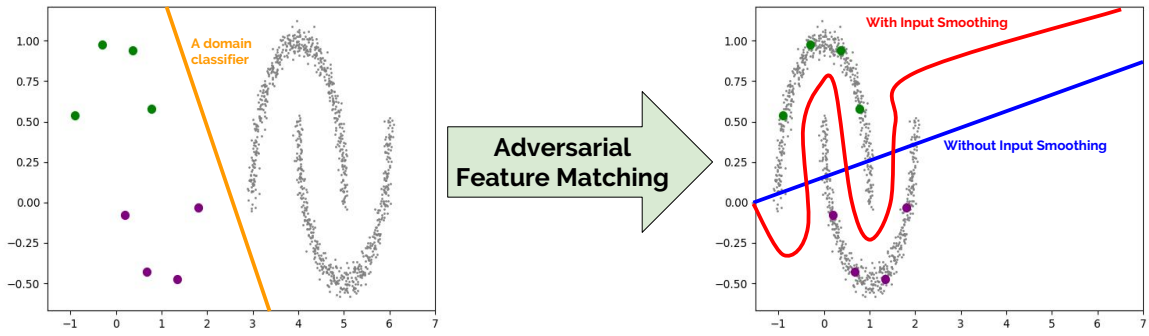


Figure 4.2: Left. In the UDA setting, there exist domain classifiers (e.g. orange line segment) being able to distinguish the source samples (green and purple dots) from the target samples (gray dots). Conditional feature matching is applied until there is no such classifier in the finite-capacity classifier space. As a result, the label-conditioned feature distributions of the source and the target data are matched. Right. Once the features are matched, exploiting unlabeled data using SSL regularizers like VAT [MMK17] becomes trivial. Only using labeled samples (green and purple dots) gives a poor decision boundary (blue line segment). When input adversarial training is applied using unlabeled samples (gray dots), the desired decision boundary is achieved (red curve). Best viewed in color.

fooling. The joint source alignment loss is

$$L_{jsa}(g) = E_{(x,y)\sim P^s} \ell_{CE}(h_j(g(x)), [\mathbf{0}, y]). \tag{4.8}$$

The encoder is trained to fool by changing the joint label from $[y, \mathbf{0}]$ to $[\mathbf{0}, y]$. Similarly, the joint-target alignment loss is defined by changing the pseudo-labels from $[\mathbf{0}, \hat{y}]$ to $[\hat{y}, \mathbf{0}]$,

$$L_{jta}(g) = E_{x\sim P_x^t} \ell_{CE}(h_j(g(x)), [\hat{y}, \mathbf{0}]). \tag{4.9}$$

The last two losses are minimized only by the encoder g.

4.3.3 Exploiting unlabeled data with SSL regularizers

Once features of the source and the target domains are matched, our formulation of UDA turns into a semi-supervised learning problem. In a way, adversarial domain adaptation deals with the large

domain shift between source and target datasets while adversarial input smoothing removes the shift in predictions within a small neighborhood of a domain (See Fig. 4.2).

For a discriminative model to exploit unlabeled data, there has to be some prior on the model parameters or on the unknown labels [CSZ09]. Applying entropy minimization for the predictions on the unlabeled data is a well-known regularizer in the SSL literature [GB05, KPG10, CFS18b]. This regularization forces decision boundaries to be in the low-density region, a desired property under the cluster assumption [CSZ09]. Our class predictor is trained to minimize this target entropy loss,

$$L_{te}(f_c) = E_{x\sim P_x^t}\ell_E(h_c(g(x))) \tag{4.10}$$

where $\ell_E(f(x)) := -\langle f(x), \log f(x)\rangle$. Since the joint predictor is already trained on the low-entropy estimates of the class predictor, it is enough to apply it to the class predictor. Minimizing entropy satisfies the cluster assumption only for Lipschitz classifiers [GB05]. The Lipschitz condition can be realized by applying adversarial training as suggested by [MMK15, MMK17]. VAT [MMK17] makes a second-order approximation for adversarial input perturbations Δx and proposes the following approximation to the adversarial noise for each input x:

$$\begin{aligned} \Delta x &\approx \epsilon_x \frac{r}{||r||_2} \\ \text{subject to } r &= \nabla_{\Delta x}\ell_{CE}(f(x), f(x+\Delta x))\Big|_{\Delta x=\xi d} \end{aligned} \tag{4.11}$$

where $d \sim N(0,1)$. Therefore, the regularization loss of [MMK15, MMK17] is

$$\begin{aligned} \ell_{VAT}(f(x)) &:= \ell_{CE}(f(x), f(x+\epsilon_x \frac{r}{||r||_2})) \\ \text{subject to } r &= \nabla_{\Delta x}\ell_{CE}(f(x), f(x+\Delta x))\Big|_{\Delta x=\xi d} \end{aligned} \tag{4.12}$$

for one input sample x. We will apply this regularizer both on the source and the target training data as in [SBN18, KSW18]. So, the source and target losses are given as follows:

$$L_{svat}(f_c) = E_{(x,y)\sim P^s}\ell_{VAT}(f_c(x)) \tag{4.13}$$

and

$$L_{tvat}(f_c) = E_{x\sim P_x^t}\ell_{VAT}(f_c(x)). \tag{4.14}$$

SSL regularizations can be applied in a later stage once feature matching is achieved [SBN18]. But, we find that in most tasks, applying SSL regularizers from the beginning of the training also works well.

We combine the objective functions introduced in this section and the previous section. The overall adversarial loss functions for the source and the target samples can be written as follows,

$$L_{adv}(g) = \lambda_{jsa} L_{jsa}(g) + \lambda_{jta} L_{jta}(g). \tag{4.15}$$

The remaining objective functions are

$$L(g, h_j, h_c) = L_s(g, h_j, h_c) + \lambda_t L_t(g, h_j, h_c) \tag{4.16}$$

where

$$L_s(g, h_j, h_c) = L_{sc}(f_c) + \lambda_{svat} L_{svat}(f_c) + \lambda_{jsc} L_{jsc}(h_j) \tag{4.17}$$

$$L_t(g, h_j, h_c) = L_{te}(f_c) + \lambda_{tvat} L_{tvat}(f_c) + \lambda_{jtc} L_{jtc}(h_j). \tag{4.18}$$

The proposed method minimizes Eqn. 4.15 and Eqn. 4.16 in an alternating fashion.

4.3.4 Connection to domain adaptation theory

The work of [BBC10] provides an upper bound on the target risk: $\ell_t(h, y) = E_{(x,y)\sim P^t}[|h(x) - y|]$ where h is the classifier. One component in the upper bound is a divergence term between two domain distributions. In UDA, we are interested in the difference of the measures between subsets of two domains on which a hypothesis in the finite-capacity hypothesis space $\mathcal{H}$ can commit errors. Instead of employing traditional metrics (e.g. the total variation distance), they use the $\mathcal{H}$-divergence. Given a domain X with P and Q probability distributions over X, and $\mathcal{H}$ a hypothesis class on X, the $\mathcal{H}$-divergence is

$$d_{\mathcal{H}\Delta\mathcal{H}}(P, Q) := 2 \sup_{h,h' \in \mathcal{H}} |Pr_{x\sim P}(h(x) \neq h'(x)) - Pr_{x\sim Q}(h(x) \neq h'(x))|. \tag{4.19}$$

Now, we can recall the main Theorem of [BBC10]. *Let $\mathcal{H}$ be an hypothesis space of VC dimension d. If X^s, X^t are unlabeled samples of size m' each, drawn from P_x^s and P_x^t respectively,*

Dataset	# of training samples	# of test samples	# of classes	Resolution	Channels
MNIST [LBB98]	$60,000$	$10,000$	10	28×28	Mono
SVHN [NWC11]	$73,257$	$26,032$	10	32×32	RGB
CIFAR10 [KH09]	$50,000$	$10,000$	10	32×32	RGB
STL [CNL11]	$5,000$	$8,000$	10	96×96	RGB
SYN-DIGITS [GL14]	$479,400$	$9,553$	10	32×32	RGB

Table 4.1: Specs of the datasets used in the experiments.

then for any $\delta \in (0,1)$, with probability at least $1-\delta$ (over the choice of the samples), for every $h \in \mathcal{H}$:

$$\ell_t(h) \leqslant \ell_s(h) + \frac{1}{2}\hat{d}_{H\Delta H}(X^s, X^t) + 4\sqrt{\frac{2d\log(2m') + \log(\frac{2}{\delta})}{m'}} + \lambda \tag{4.20}$$

where $\lambda = \ell_s(h^*) + \ell_t(h^*)$, $h^* = \arg\min_{h\in\mathcal{H}} \ell_s(h) + \ell_t(h)$ and $\hat{d}_{H\Delta H}(X^s, X^t)$ is empirical $\mathcal{H}$ divergence. In words, the target risk is upper bounded by the source risk, empirical $\mathcal{H}$-divergence and combined risk of ideal joint hypothesis λ.

If there is no classifier which can discriminate source samples from target samples then the empirical $\mathcal{H}$-divergence is zero from Lemma 2 of [BBC10]. DANN of [GL14] minimizes $\hat{d}_{H\Delta H}(X^s, X^t)$ by matching the marginal distributions (i.e. by aligning marginal push-forwards $g\#P^s_x$ and $g\#P^t_x$). But, if the joint push-forward distributions ($g\#P^s$ and $g\#P^t$) are not matched accurately, there may not be a classifier in the hypothesis space with low risk in both domains. Hence, λ has to be large for any hypothesis space $\mathcal{H}$.

Our proposed method tackles this problem, by making sure that the label-conditioned push-forwards are aligned disjointly. With disjoint alignment, we mean that no two samples with different labels can be assigned to the same feature point. Moreover, the third term in the upper bound decreases with the number of samples drawn from both domains. This number can increase with data augmentation. VAT has the same effect of augmenting the data with adversarially perturbed images where the small perturbations are nuisances for the task.

4.4 Empirical Evaluation

4.4.1 Datasets

We evaluate the proposed method on the standard digit and object image classification benchmarks in UDA. Namely, CIFAR → STL, STL → CIFAR, MNIST → SVHN, SVHN → MNIST, SYN-DIGITS → SVHN and MNIST → MNIST-M. The first three settings are the most challenging ones where state-of-the-art (SOA) methods accuracies are still below 90%. Our method achieves SOA accuracy in all these tasks.

CIFAR ↔ STL. Similar to CIFAR, STL images are acquired from labeled examples on ImageNet. However, images are 96×96 instead of the 32×32 images in CIFAR. All images are converted to 32×32 RGB in pretraining. We down-sampled images by local averaging. Note that we only used the labeled part of STL in all the experiments. CIFAR and STL both have 10 classes, 9 of which are common for both datasets. Like previous works [KSW18, FMF18] we removed the non-overlapping classes (class frog and class monkey) reducing the problem into a 9-class prediction. See Table 4.1 for the specs of the datasets.

MNIST ↔ SVHN. We convert MNIST images to RGB images by repeating the gray image for each color channel and we resize them to 32×32 by padding zeros. Following previous works [KSW18, FMF18], we used Instance Normalization (IN) for MNIST ↔ SVHN, which is introduced by [UVL17] for image style transfer. We preprocess images both at training and test time with IN.

SYN-DIGITS → SVHN. SYN-DIGITS [GL14] is a dataset of synthetic digits generated from Windows fonts by varying position, orientation, and background. In each image, one, two, or three digits exist. The degrees of variation were chosen to match SVHN.

MNIST → MNIST-M. MNIST-M [GL14] is a difference-blend of MNIST over patches randomly extracted from color photos from BSDS500 [AMF11]. I.e. $I^{out}_{ijk} = |I^1_{ijk} - I^2_{ijk}|$, where i, j are the coordinates of a pixel and k is a channel index. MNIST-M images are RGB and 28 by 28. MNIST images are replicated for each channel during preprocessing.

4.4.2 Implementation details

No data augmentation is used in any of the experiments to allow for a fair comparison with SOA methods [KSW18, SBN18]. Again, to allow a fair comparison with the previous works [FMF18, SBN18], we have not used sophisticated architectures like ResNet [HZR16a]. We report the inference performance of the class-predictor.

Batchsize of 64 is used for both the source and the target samples during training. Batchsize of 100 is used at inference time. The networks used in the experiments are given in Table 4.2 and Table 4.3. Instance norm is only used in MNIST $\leftrightarrow$ SVHN experiments. In all experiments, networks are trained for $60,000$ iterations. This is less than $80,000 + 80,000 = 160,000$ iterations that SOA methods VADA+DIRT-T and Co-DA+DIRT-T are trained for. Weight decay of 10^{-4} is used. In CIFAR $\leftrightarrow$ STL and SYN-DIGITS $\rightarrow$ SVHN, as an optimizer we use SGD with the initial learning rate of 0.1. Learning rate is decreased to 0.01 at iteration $40,000$. The momentum of SGD is 0.9. In MNIST $\leftrightarrow$ SVHN and MNIST $\rightarrow$ MNIST-M, Adam optimizer with the fixed learning rate 0.001 is used. Momentum is chosen to be 0.5.

We feed source and training samples into two different mini-batches at each iteration of training. As we are using the same batch layers for both source and target datasets, mean and variance learned – to be used at inference time – are the running average over both source and target data statistics.

Office UDA experiments (Amazon→Webcam, Webcam→DSLR, DSLR→Webcam) were used as the standard benchmark in early UDA works [LZW16, SQZ17, LWS16, THS17]. However, recent SOA methods [SUH17, KSW18, SBN18, FMF18] did not report on these datasets, as labels are noisy [BSD17]. Moreover, this is a small dataset with $4,652$ images from 31 classes necessitating the use of Imagenet-pretrained networks. Hence, we also choose not to report experiments on this dataset.

4.4.3 Results

We report the performance of the proposed method in Table 4.4. In all the experiments, the proposed method achieves the best or the second-best results after Co-DA [KSW18]. Especially in the

3×3 convolution, 64
3×3 convolution, 64
3×3 convolution, 64
2×2 max-pool, dropout
3×3 convolution, 64
3×3 convolution, 64
3×3 convolution, 64
2×2 max-pool, dropout

(a) Encoder

3×3 convolution, 64
1×1 convolution, 64
1×1 convolution, 64
Average pooling, $6 \rightarrow 1$
FC: $64 \rightarrow$ K
Softmax

(b) Class predictor

3×3 convolution, 64
1×1 convolution, 64
1×1 convolution, 64
Average pooling, $6 \rightarrow 1$
FC: $64 \rightarrow 2$K
Softmax

(c) Joint predictor

Table 4.2: The network used in the tasks involving MNIST dataset (i.e. MNIST $\leftrightarrow$ SVHN and MNIST $\rightarrow$ MNIST-M), *small-net* from [SBN18, KSW18]. Each convolutional layer is followed by a batch-norm and leaky RELU (lReLU). FC stands for fully connected layer.

3×3 convolution, 128
3×3 convolution, 128
3×3 convolution, 128
2×2 max-pool, dropout
3×3 convolution, 256
3×3 convolution, 256
3×3 convolution, 256
2×2 max-pool, dropout
3×3 convolution, 512
1×1 convolution, 256
1×1 convolution, 128
Average pooling, $6 \rightarrow 1$

(a) Encoder

Fully connected: $128 \rightarrow$ K
Softmax

(b) Class predictor

Fully connected: $128 \rightarrow 2$K
Softmax

(c) Joint predictor

Table 4.3: The network used in the STL $\leftrightarrow$ CIFAR, SYN-DIGITS $\rightarrow$ SVHN, *conv-large* from [FMF18]. Each convolutional layer is followed by a batch-norm and leaky RELU (lReLU). Slope of each leaky RELU (lReLU) layer is 0.1.

Source dataset	MNIST	SVHN	CIFAR	STL	SYN-DIGITS	MNIST
Target dataset	SVHN	MNIST	STL	CIFAR	SVHN	MNIST-M
[GL14] DANN*	60.6	68.3	78.1	62.7	90.1	94.6
[GKZ16] DRCN	40.05	82.0	66.37	58.86	NR	NR
[SSS16] kNN-Ad	40.3	78.8	NR	NR	NR	86.7
[SUH17] ATT	52.8	86.2	NR	NR	92.9	94.2
[FMF18] Π-model**	33.87	93.33	77.53	71.65	96.01	NR
[SBN18] VADA	47.5	97.9	80.0	73.5	94.8	97.7
[SBN18] DIRT-T	54.5	99.4	NR	75.3	96.1	98.9
[SBN18] VADA + IN	73.3	94.5	78.3	71.4	94.9	95.7
[SBN18] DIRT-T +IN	76.5	99.4	NR	73.3	96.2	98.7
[KSW18] Co-DA	81.7	99.0	81.4	76.4	96.4	99.0
[KSW18] Co-DA + DIRT-T	88.0	**99.4**	NR	77.6	**96.4**	99.1
Ours	**89.19**	99.33	**81.65**	**77.76**	96.22	**99.47**
Source-only (baseline)	44.21	70.58	79.41	65.44	85.83	70.28
Target-only	94.82	99.28	77.02	92.04	96.56	99.87

Table 4.4: Comparison to SOA UDA algorithms on the UDA image classification tasks. Accuracies on the target test data are reported. Algorithms are trained on entire labeled source training data and unlabeled target training data. NR stands for not reported. * DANN results are implementation of [SBN18] with instance normalized input. ** Results of [FMF18] with minimal augmentations are reported. The proposed method achieves the best or second highest score after Co-DA. The proposed method can be combined with Co-DA, but we report the naked results to illustrate the effectiveness of the idea.

most challenging tasks, for which SOA accuracies are below 90%, our method outperforms all the previous methods. Numbers reported in the corresponding papers are used except DANN for which reported scores from [SBN18] are used.

Works we compare to include [GKZ16] which proposed Deep Reconstruction Classification network (DRCN). The cross-entropy loss on the source data and reconstruction loss on the target data are minimized. [SBN18] applied the SSL method VAT to UDA which they call VADA (Virtual Adversarial Domain Adaptation). After training with domain-adversarial loss of [GL14] and VAT, they further fine-tune only on the target data with the entropy and VAT objectives. [KSW18] suggested having two hypotheses in a way that they learn diverse feature embeddings while class predictions are encouraged to be consistent. They build this method on VADA of [SBN18]. The proposed method can be further improved by combining it with Co-DA even though we ignore it to highlight the effectiveness of the clean method. Compared to Co-DA, our method has the memory and computational time advantage of not training multiple encoders. [SUH17] introduced ATT where two networks are trained on the source data and predictions of the networks are used as pseudo labels on the target data. Another network is trained on the target data with pseudo labels. A pseudo label is assigned if two networks agree and at least one of them is confident.

Source-only models are also reported as baselines. These models are trained without exploiting the target training data in the standard supervised learning setting using the same learning procedure (e.g. network, number of iterations, etc.) as UDA methods. Since CIFAR has a large labeled set (45000 after removing samples of class frog), CIFAR $\rightarrow$ STL has a high accuracy even without exploiting the unlabeled data. Still, the proposed method outperforms the source-only baseline by 2.24%. The target-only models are trained only on the target domain with class labels revealed. The target-only performance is considered as the empirical upper bound in some papers, but it is not necessarily the case, as seen in the CIFAR $\rightarrow$ STL setting where the target data is scarce; thus the target-only model is even worse than the source-only model.

The advantage of the proposed method is more apparent in the converse direction STL $\rightarrow$ CIFAR where the accuracy increases from 65.44% to 77.76%. STL contains a very small (4500 after removing samples of class monkey) labeled training set. That is why DIRT-T which fine-tunes

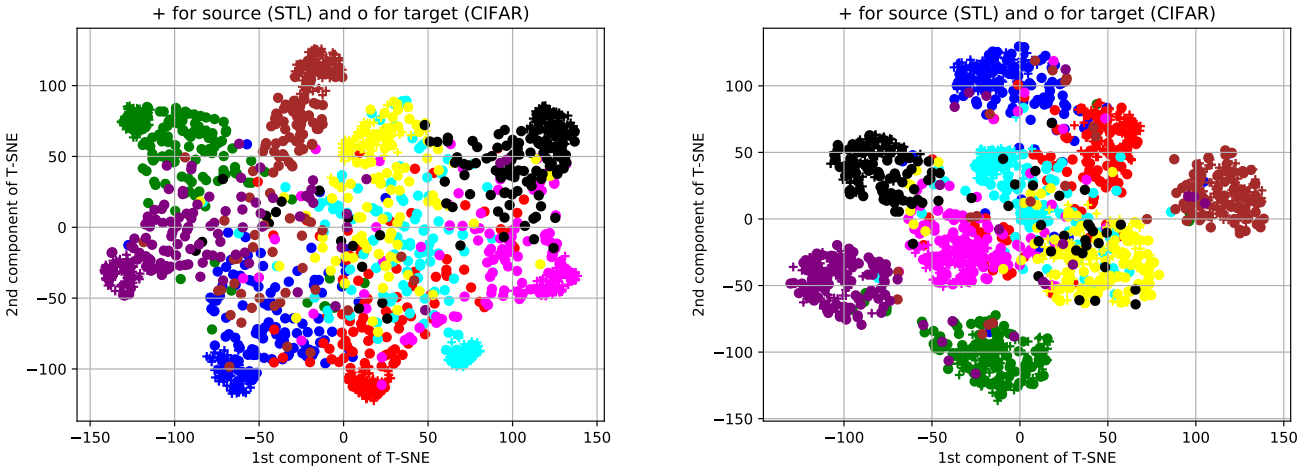


Figure 4.3: t-SNE plots for STL → CIFAR. t-SNE plots of the source-only trained (top panel) and the proposed method model (bottom panel). Encoder outputs are projected to two-dimensional space with t-SNE. Samples corresponding to the same class are visualized with the same color. The symbol "+" is used for the source samples and "o" is for the target samples. Best viewed in color.

on the target data, gave unreliable results for CIFAR → STL so they only report VADA results.

The source-only baseline has its lowest score in the MNIST → SVHN setting. This is a challenging task as MNIST is greyscale, in contrast to color digits in SVHN. Moreover, SVHN contains multiple digits within an image while MNIST pictures contain single, centered digits. SVHN → MNIST is a much simpler experimental setting where SOA accuracies are above 99%. We achieve SOA in MNIST → SVHN while being second best in SVHN → MNIST after Co-DA. Note that our accuracy in SVHN → MNIST is 99.33%. MNIST → MNIST-M and SYN-DIGITS → SVHN are other saturated tasks where our method beats SOA in the former one while being second best in the latter. At these levels of saturation of the dataset, top-rated performance is not as informative.

In MNIST → SVHN, our method (89.19%) is substantially better than VADA+IN (73.3%) which also uses input smoothing but with DANN (marginal alignment). Similarly, in STL → CIFAR, VADA achieves 73.5% while our method is SOA with 77.76% accuracy. This shows the effectiveness of our joint-alignment method.

To demonstrate the effectiveness of the proposed approach in aligning the samples of the same

class, we visualize the t-Distributed Stochastic Neighbor Embedding (t-SNE) [MH08] of the source-only baseline and the proposed approach in Fig. 4.3. t-SNE is performed on the encoder output for 1000 randomly drawn samples from both source and target domains for STL $\rightarrow$ CIFAR setting. As one can see, samples of the same classes are better aligned for the proposed approach compared to the source-only method.

4.5 Analysis

The main result of our analysis is that the objective introduced in Sec. 4.3.2 is minimized only for matching conditional push-forwards given the optimal joint predictor (Theorem 1). For that, we first find the optimal joint predictor in Proposition 1. We operate under the supervised setting, assuming the target labels are revealed. So, we replace $\hat{y}$ in the objective functions with ground-truth labels y for the target samples. Proofs follow similar steps to Proposition 1 and Theorem 1 in [GPM14].

Proposition 1. *The optimal joint predictor h_j minimizing $L_{jsc}(h_j) + L_{jtc}(h_j)$ given in the Eqn. 4.6, Eqn. 4.7 for any feature z with non-zero measure either on $g\#P_x^s(z)$ or $g\#P_x^t(z)$ is*[2]

$$h_j(z)[i] = \frac{g\#P^s(z, y = e_i)}{g\#P_x^s(z) + g\#P_x^t(z)}$$

$$h_j(z)[i + K] = \frac{g\#P^t(z, y = e_i)}{g\#P_x^s(z) + g\#P_x^t(z)} \text{ for } i \in \{1, ..., K\}.$$

Theorem 1. *The objective $L_{jsa}(g) + L_{jta}(g)$ given in the Eqn. 4.8, Eqn. 4.9 is minimized for the given optimal joint predictor if and only if $g\#P^s(z|y = e_k) = g\#P^t(z|y = e_k)$ and $g\#P^s(z|y = e_k) > 0 \Rightarrow g\#P^s(z|y = e_i) = 0$ for $i \neq k$ for any $y = e_k$ and z.*

Theorem 1 states that no two samples with different labels can be assigned to the same feature point for the encoder to minimize its loss given the optimal joint predictor. Moreover, the measure assigned to each feature is the same for the source and the target push-forward distributions to maximally fool the optimal joint predictor.

[2] *We used $P(z)$ both to denote the distribution of a random variable and the corresponding density function, but the meaning should be clear from the context.*

This result indicates that the global minimum of the proposed objective function is achieved when conditional feature distributions are aligned. But, this analysis does not necessarily give a guarantee that the converged solution is optimal in practice as we do not have access to the target labels in UDA. But, we demonstrated empirically in Fig. 4.3 that with reasonably good pseudo-labels provided by a separate class predictor, the objective gives better alignment than the source-only model. The second issue is that finding the optimal predictor or generator with finite samples may not be possible, as optimal solutions are derived as functions of true measures instead of the network parameters trained on finite samples. Lastly, the joint predictor is not trained until convergence; instead, a gradient step is taken in an alternating fashion for computational efficiency. So, the predictor is also not necessarily optimal in practice. Even though there are still gaps to be filled between this theory and practice, this analysis shows us that the proposed objective function is doing a sensible job given pseudo-labels for the target data are reasonably good.

4.5.1 Proofs

First, we prove a simple lemma that will be handy in the proof of Proposition 1.

Lemma 1. $\theta^* = \arg\min_\theta \sum_{i=1}^K -\alpha[i]\log(\theta[i])$ *s.t.* $1 \geqslant \theta[i] \geqslant 0$, $\sum_{i=1}^K \theta[i] = 1$, $\alpha[i] > 0$, *for all* i. *Then,* $\theta^*[k] = \frac{\alpha[k]}{\sum_{i=1}^K \alpha[i]}$ *for any* k.

Proof. Let us write the Lagrangian form excluding inequality constraints, $L(\theta, \lambda) = \sum_{i=1}^K -\alpha[i]\log(\theta[i]) + \lambda(\sum_{i=1}^K \theta[i] - 1)$. $\nabla_{\theta[i]} L(\theta, \lambda) = -\frac{\alpha[i]}{\theta[i]} + \lambda = 0$ and $\theta[i] = \frac{\alpha[i]}{\lambda}$ for all i. $-\log$ is convex hence sum of them also convex and the stationary point is global minima. Then, the dual form becomes $g(\lambda) = \sum_{i=1}^K -\alpha[i]\log(\frac{\alpha[i]}{\lambda}) + \lambda(\sum_{i=1}^K \frac{\alpha[i]}{\lambda} - 1)$ then $\nabla_\lambda g(\lambda) = 0$ when $\lambda = \sum_{i=1}^K \alpha[i]$ and $\theta[k] = \frac{\alpha[k]}{\sum_{i=1}^K \alpha[i]}$. Note that the constraint $1 \geqslant \theta[i] \geqslant 0$ does not constrain the solution space as $\theta[i]$ has to be non-negative for $\log(\theta[i])$ to be defined and $\sum_{i=1}^K \theta[i] = 1$ enforces $1 \geqslant \theta[i]$. □

Proposition 1. *The optimal joint predictor* h_j *minimizing* $L_{jsc}(h_j) + L_{jtc}(h_j)$ *given in the Eqn. 4.6, Eqn. 4.7 for any feature* z *with non-zero measure either on* $g\#P_x^s(z)$ *or* $g\#P_x^t(z)$ *is*

$$h_j(z)[i] = \frac{g\#P^s(z, y = e_i)}{g\#P_x^s(z) + g\#P_x^t(z)} \text{ and } h_j(z)[i+K] = \frac{g\#P^t(z, y = e_i)}{g\#P_x^s(z) + g\#P_x^t(z)} \text{ for } i \in \{1, ..., K\}.$$

Proof.

$$L_{jsc}(h_j) + L_{jtc}(h_j) = E_{(x,y)\sim P^s}\ell_{CE}(h_j(g(x)), [y,\, \mathbf{0}]) + E_{(x,y)\sim P^t}\ell_{CE}(h_j(g(x)), [\mathbf{0},\, y]) \quad (4.21)$$

$$= \int_{(x,y)\sim P^s} P^s_x(x)\ell_{CE}(h_j(g(x)), [y,\, \mathbf{0}])dx$$

$$+ \int_{(x,y)\sim P^t} P^t_x(x)\ell_{CE}(h_j(g(x)), [\mathbf{0},\, y])dx \quad (4.22)$$

$$= \int_{z\sim g\# P^s_x} \int_{(x,y)\sim P^s s.t. z=g(x)} P^s_x(x)\ell_{CE}(h_j(z), [y,\, \mathbf{0}])dxdz$$

$$+ \int_{z\sim g\# P^t_x} \int_{(x,y)\sim P^t s.t. z=g(x)} P^t_x(x)\ell_{CE}(h_j(z), [\mathbf{0},\, y])dxdz \quad (4.23)$$

$$= \int_{z\sim g\# P^s_x} \int_{(x,y)\sim P^s s.t. z=g(x)} P^s_x(x)\langle -\log h_j(z), [y,\, \mathbf{0}]\rangle dxdz$$

$$+ \int_{z\sim g\# P^t_x} \int_{(x,y)\sim P^t s.t. z=g(x)} P^t_x(x)\langle -\log h_j(z), [\mathbf{0},\, y]\rangle dxdz \quad (4.24)$$

$$= \int_{z\sim g\# P^s_x} \langle -\log h_j(z), [\int_{(x,y)\sim P^s s.t. z=g(x)} P^s_x(x) y dx,\, \mathbf{0}]\rangle dz$$

$$+ \int_{z\sim g\# P^t_x} \langle -\log h_j(z), [\mathbf{0},\, \int_{(x,y)\sim P^t s.t. z=g(x)} P^t_x(x) y dx]\rangle dz \quad (4.25)$$

$$= \int_{z\sim g\# P^s_x} \sum_{i=1}^{K} -\log h_j(z)[i]\, g\# P^s(z, y=e_i)dz$$

$$+ \int_{z\sim g\# P^t_x} \sum_{i=1}^{K} -\log h_j(z)[i+K]\, g\# P^t(z, y=e_i)dz. \quad (4.26)$$

From Lemma 1, $h_j(z)[i] = \frac{g\# P^s(z,y=e_i)}{Z}$ and $h_j(z)[i+K] = \frac{g\# P^t(z,y=e_i)}{Z}$ for $i \in \{1, ..., K\}$ where $Z = \sum_{i=1}^{K}(g\# P^s(z, y=e_i) + g\# P^t(z, y=e_i)) = g\# P^s_x(z) + g\# P^t_x(z)$ for any z. Note that, we integrated losses over x only; instead of (x, y) as we are assuming that the ground-truth labeling function is deterministic. Moreover, in Lemma 1, we assumed $\alpha[i] > 0$ while here $g\# P^s(z, y = e_i)$ and $g\# P^t(z, y = e_i)$ might be zero for some i. But since we are taking $\alpha[i]\log(\theta[i])$ as zero whenever $\alpha[i] = 0$ for any value of $\theta[i]$, the result does not change. □

The following Lemma will be used in the proof of Theorem 1.

Lemma 2. $\min_{P,Q} L(P, Q) = E_{x\sim P} - \log \frac{Q(x)}{P(x)+Q(x)} + E_{x\sim Q} - \log \frac{P(x)}{P(x)+Q(x)}$ *is achieved only if* $P(x) = Q(x)$ *for all* x.

Proof.

$$L(P,Q) = \int_x -P(x)\log(\frac{Q(x)}{P(x)+Q(x)}) - Q(x)\log(\frac{P(x)}{P(x)+Q(x)})dx \tag{4.27}$$

$$= \int_x P(x)\log(\frac{P(x)+Q(x)}{Q(x)}) + Q(x)\log(\frac{P(x)+Q(x)}{P(x)})dx \tag{4.28}$$

$$= \int_x P(x)\log(1+\frac{P(x)}{Q(x)}) + Q(x)\log(1+\frac{Q(x)}{P(x)})dx \tag{4.29}$$

$$= \int_x \log(1+\frac{P(x)}{Q(x)})(1+\frac{P(x)}{Q(x)})Q(x) - \log(1+\frac{P(x)}{Q(x)})Q(x)$$
$$+ Q(x)\log(1+\frac{Q(x)}{P(x)})dx \tag{4.30}$$

$$= \int_x \Big(\log(1+\frac{P(x)}{Q(x)})(1+\frac{P(x)}{Q(x)}) - \log(1+\frac{P(x)}{Q(x)}) + \log(1+\frac{Q(x)}{P(x)})\Big)Q(x)dx \tag{4.31}$$

$$= \int_x \big(\log(1+\frac{P(x)}{Q(x)})(1+\frac{P(x)}{Q(x)}) + \log(\frac{Q(x)}{P(x)})\big)Q(x)dx \tag{4.32}$$

$$= \log(4) - \int_x \log(2)\frac{P(x)+Q(x)}{Q(x)}Q(x)dx$$
$$+ \int_x \big(\log(1+\frac{P(x)}{Q(x)})(1+\frac{P(x)}{Q(x)}) + \log(\frac{Q(x)}{P(x)})\big)Q(x)dx \tag{4.33}$$

$$= \log(4)$$
$$+ \int_x \big(\log(1+\frac{P(x)}{Q(x)})(1+\frac{P(x)}{Q(x)}) - \log(\frac{P(x)}{Q(x)}) - \log(2)(1+\frac{P(x)}{Q(x)})\big)Q(x)dx. \tag{4.34}$$

Let $\phi(\beta) := \log(1+\beta)(1+\beta) - \log(\beta) - \log(2)(1+\beta)$. Then, $\nabla_\beta\phi(\beta) = 1+\log(1+\beta) - \frac{1}{\beta} - \log(2)$ and $\nabla_\beta\nabla_\beta\phi(\beta) = \frac{1}{1+\beta} + \frac{1}{\beta^2} > 0$ for $\beta > 0$. Hence $\phi(\beta)$ is convex and we can apply Jensen,

$$L(P,Q) = \log(4) + \int_x \phi(\frac{P(x)}{Q(x)})Q(x)dx \geqslant \log(4) + \phi(\int_x \frac{P(x)}{Q(x)}Q(x)dx)$$
$$= \log(4) + \phi(1) = \log(4). \tag{4.35}$$

Since ϕ is strictly convex, equality is satisfied only for constant argument i.e. when $\frac{P(x)}{Q(x)} = 1$ which is also the global minima of $\phi(\beta)$ as $\nabla_\beta\phi(1) = 0$. □

Theorem 1. *The objective $L_{jsa}(g) + L_{jta}(g)$ given in the Eq. 8,9 is minimized for the given optimal joint predictor if and only if $g\#P^s(z|y = e_k) = g\#P^t(z|y = e_k)$ and $g\#P^s(z|y = e_k) > 0 \Rightarrow g\#P^s(z|y = e_i) = 0$ for $i \neq k$ for any $y = e_k$ and z.*

Proof. The objective for the encoder is,

$$E_{(x,y)\sim P^s}\ell_{CE}((h_j(g(x)),[\mathbf{0},\,y]) + E_{(x,y)\sim P^t}\ell_{CE}((h_j(g(x)),[y,\,\mathbf{0}]) \tag{4.36}$$

For samples with label e_k we want to minimize,

$$E_{(x,y)\sim P^s(x,y=e_k)}\ell_{CE}((h_j(g(x)),[\mathbf{0},\,y]) + E_{(x,y)\sim P^t(x,y=e_k)}\ell_{CE}((h_j(g(x)),[y,\,\mathbf{0}]) \tag{4.37}$$

$$= E_{(x,y)\sim P^s(x,y=e_k)} - \langle[\mathbf{0},\,y],\log(h_j(g(x))\rangle + E_{(x,y)\sim P^t(x,y=e_k)} - \langle[y,\,\mathbf{0}],\log(h_j(g(x))\rangle \tag{4.38}$$

$$= -E_{(x,y)\sim P^s(x,y=e_k)}\log(h_j(g(x))[k+K] - E_{(x,y)\sim P^t(x,y=e_k)}\log(h_j(g(x))[k] \tag{4.39}$$

Given the classifier h_j is optimal, the above them becomes

$$\begin{aligned}
&-\int_{z\sim g\#P^s(z,y=e_k)} g\#P^s(z,y=e_k)\log\frac{g\#P^t(z,y=e_k)}{\sum_{i=1}^K(g\#P^s(z,y=e_i)+g\#P^t(z,y=e_i)}dz\\
&-\int_{z\sim g\#P^t(z,y=e_k)} g\#P^t(z,y=e_k)\log\frac{g\#P^s(z,y=e_k)}{\sum_{i=1}^K(g\#P^s(z,y=e_i)+g\#P^t(z,y=e_i))}dz \qquad (4.40)\\
&=\int_{z\sim g\#P^s(z,y=e_k)} g\#P^s(z,y=e_k)\Big(-\log\frac{g\#P^t(z,y=e_k)}{g\#P^s(z,y=e_k)+g\#P^t(z,y=e_k)}\\
&+\log\frac{\sum_{i=1}^K(g\#P^s(z,y=e_i)+g\#P^t(z,y=e_i))}{g\#P^s(z,y=e_k)+g\#P^t(z,y=e_k)}\Big)dz\\
&+\int_{z\sim g\#P^t(z,y=e_k)} g\#P^t(z,y=e_k)\Big(-\log\frac{g\#P^s(z,y=e_k)}{g\#P^s(z,y=e_k)+g\#P^t(z,y=e_k)}\\
&+\log\frac{\sum_{i=1}^K(g\#P^s(z,y=e_i)+g\#P^t(z,y=e_i))}{g\#P^s(z,y=e_k)+g\#P^t(z,y=e_k)}\Big)dz \qquad (4.41)
\end{aligned}$$

Let us write first and second terms in each integration separately:

$$\begin{aligned}
&L_1(g\#P^s,g\#P^t)\\
&=-\int_{z\sim g\#P^s(z,y=e_k)} g\#P^s(z,y=e_k)\log\frac{g\#P^t(z,y=e_k)}{g\#P^s(z,y=e_k)+g\#P^t(z,y=e_k)}dz\\
&-\int_{z\sim g\#P^t(z,y=e_k)} g\#P^t(z,y=e_k)\log\frac{g\#P^s(z,y=e_k)}{g\#P^s(z,y=e_k)+g\#P^t(z,y=e_k)}dz \qquad (4.42)
\end{aligned}$$

$$\begin{aligned}
&L_2(g\#P^s,g\#P^t)\\
&=\int_{z\sim g\#P^s(z,y=e_k)} g\#P^s(z,y=e_k)\log\frac{\sum_{i=1}^K(g\#P^s(z,y=e_i)+g\#P^t(z,y=e_i))}{g\#P^s(z,y=e_k)+g\#P^t(z,y=e_k)}dz
\end{aligned}$$

$$+\int_{z\sim g\#P^t(z,y=e_k)} g\#P^t(z,y=e_k)\log\frac{\sum_{i=1}^{K}(g\#P^s(z,y=e_i)+g\#P^t(z,y=e_i))}{g\#P^s(z,y=e_k)+g\#P^t(z,y=e_k)}dz \quad (4.43)$$

If there is a solution which is global minima of both L_1,L_2 then it is also the global minima of the overall term L_1+L_2. L_2 has its minimum at $\sum_{i=1 s.t. i\neq k}^{K}(g\#P^s(z,y=e_i)+g\#P^t(z,y=e_i))=0$ whenever $g\#P^s(z,y=e_k)+g\#P^t(z,y=e_k)>0$. From Lemma 2, $L_1(g\#P^s,g\#P^t)$ achieves its minimum only when $g\#P^s(z,y=e_k)=g\#P^t(z,y=e_k)$ for any z. Intersection of two minimas gives the desired solution.

□

4.6 Conclusion

We introduce a joint predictor that learns a distribution over class and domain labels. The encoder is trained to fool this predictor within the same-class samples of each domain. We also employed recent tools from SSL to improve the generalization. The proposed idea achieved state-of-the-art accuracy in most challenging image classification tasks for which accuracies are still below 90%.

CHAPTER 5

Disentangled Image Generation for Unsupervised Domain Adaptation

5.1 Introduction

We develop an architecture that embodies the ability to generate images with controlled class and domain labels. That is, the model can receive as input a domain (e.g. SVHN) and class label (e.g. 5), along with a random vector, and produce as output diverse images of the given class in the given domain for different realizations of the random vector (e.g. different digit fives in SVHN).

While generative models have been used for UDA [LT16, HW18, WH18, BSD17, RCT18], none has the ability to independently control both the domain and class of the generated images. UFDN [LLY18] is the closest method, where the domain variable is controlled independently but the label is implicit in the input image. Like other existing methods, UFDN needs an encoder to process the input image, and there is nothing to explicitly enforce that the class is preserved in the output. Moreover, the learning criterion is not just adversarial, but conflicting, as the reconstruction loss aims to keep as much information specific to the input image as possible, including the domain, whereas the adversarial loss aims to remove it from the encoder. The resulting generator could synthesize images that change class, as there is no explicit enforcement of class-constancy. Such a generative model therefore yields no insight as to whether the model really captures the class or simply transforms the low-level image statistics. Our generative model does not require an image as input, just class and domain labels, and a random vector. It does not rely on reconstruction losses [LLY18, HTP17, LT16] nor is it based on just image translation [RCT18]. We can test the model to see if, given just labels, it can produce images with the same domain and class. No existing method

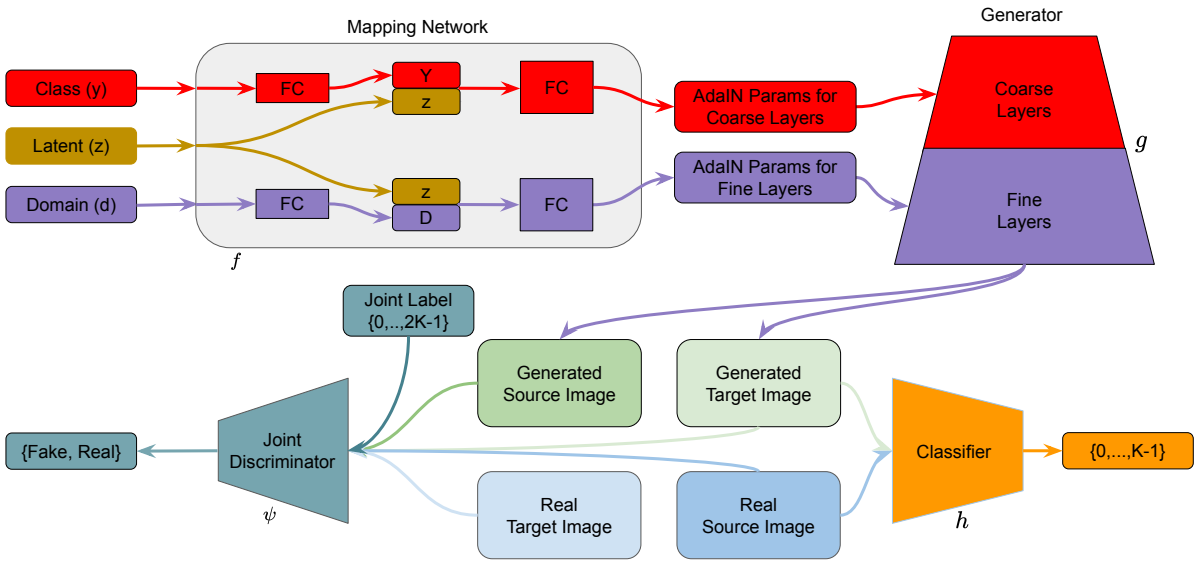


Figure 5.1: Detailed overview of the proposed approach. StyleGAN based generator is used to generate images in the UDA setting. The generator (g) is conditioned on both the domain (d) and the class (y) label. The class label (y) is used only to tweak the coarse layer parameters (the red blocks) while the domain label is used only to tweak the fine layer parameters (the purple blocks). The joint discriminator (the green block) tries to distinguish generated images from the reals ones for guiding the generator training. The joint discriminator has $2K$ dimensional output from which the only one is selected conditioned on the joint label. For instance, given that $K = 10$, the source (target) 9 has the joint label of $j{=}9(19)$, and jth index of the joint discriminator output is used to tell whether the image is fake or real. For the unlabeled target samples, another classifier (h, the orange block) is used to provide the pseudo-labels for calculating the loss the joint discriminator is minimizing (See Eqn. 5.3). The classifier (h) is trained on labeled images from both source and target (generated) domains.

has this capability. Of course, during UDA training we do not have ground-truth labels for the target domain, so we feed imputed (noisy) ones to the generative model, that incorporates several regularization mechanisms, both implicit and explicit, as we will see in Sec. 5.3. Hence, our model can approximate the distribution, $P(x|y, d)$ of the image x given the class label y and domain label $d \in \{source,\ target\}$ using a generator. Then, we can sample target images to train a classifier as we have complete control over the class labels. In the UDA setting, we regularize the model by not feeding the domain label in the early layers of the generator so it is forced to learn a common representation of each class for different domains.

Key Idea: To test the hypothesis that a generative model that allows control of a factor, say the class label, independent of another, say the nuisance variability in the images, improves UDA we leverage recent progress in the generative modeling for style transfer [KLA19], where the separation of style and content corresponds to class and nuisances in UDA. If the domain label and the class label can be disentangled, one can easily perform image generation in the unlabeled target domain while respecting the input class label simply by changing the domain label from source to target. As an architectural regularization, we encoded the class label information in the early stages of the decoder ("coarse layers"), while domain variations are fed to the late stages ("fine layers"). As a loss regularization, we proposed the pseudo-conditional GAN loss described in Sec. 5.3.2 on top of the standard UDA losses described in Sec. 5.3.3.

5.2 Related Work

Several generative methods have been proposed for UDA. CyCADA proposed by [HTP17] utilized the cyclic constraints. COGAN [LT16] used weight sharing to learn a joint distribution of images from the source and the target domains. NAM [HW18] proposed to use reconstruction losses for translation instead of adversarial and cycle losses. In GAGL [WH18], the generator network is used to guide classification boundaries towards the low-density regions. PixelDA [BSD17] and SBADA-GAN [RCT18] minimized the classification loss on the translated images where they feed the source image to the generator for getting the corresponding target sample. Finally, UFDN [LLY18] feeds

Figure 5.2: Conditional image generation results of the proposed method on the UDA digit benchmarks. The upper and lower half of the panels are generated images for the labeled source and **unlabeled** target domain respectively. The class label input y is $0, \cdots, 9$ from the first row to the last row. Each row shows the generated images for the same domain label and the same class label with different realizations of the latent vector z. The generator can generate a diverse set of samples for each class in the target domain which is utilized by the classifier. The results are given for SYNDIGITS → SVHN (left), SVHN → MNIST (middle) and MNIST → SVHN (right). In SVHN → MNIST, most digits are correctly generated. MNIST → SVHN is a more challenging task where the network is especially confused between the digits "6" and "8."

the domain label to the generator along with the inferred latent vector for manipulating the domain of generated images. Unlike these methods, we control the generation with three independent factors: class-label, domain-label and a random latent vector to generate diverse images for each domain- and class-label. This way, we do not condition the generation on an input image nor a latent vector learned from it. Hence, we can freely sample an image from the desired class-label on the target domain without introducing the bias of a particular source sample.

Translation methods [LFY17, LLK18, LBK17, LYF18, LHM19] are related to UDA in that the goal is to preserve the "content" of an image while its "style" is changed. One line of work builds on Instance Normalization (IN), and [HB17, MJG18, HLB18] extend it to Adaptive Instance Normalization (Ada-IN). They modify the style by modulating the hidden feature layers. The proposed method also performs image translation using similar architectures but its novel design is aimed at performing classification tasks in the UDA setting unlike these methods.

Contributions: We show it is possible to progressively learn shared representations of classes with differentiated domain characteristics in an unsupervised fashion. Since the class labels in the target-domain are unknown, predictions from a classifier are used as pseudo-labels for the discriminator loss. Absent regularization, such practice would be self-referential. With regularization, both implicit in the training process and choice of architecture, and explicit as described, we show that we can (i) effectively generate images in the target domain whose class labels are controlled; (ii) separate domain-dependent (nuisance) variability from content (class) variability, and finally (iii) perform inference in unseen target samples using the classifier trained to provide the pseudo-labels, thereby having adapted the classifier to the covariate shift due to the domain change, enabling unsupervised domain adaptation. To the best of our knowledge, this is the only generative model performing at or near the state-of-the-art in UDA classification benchmarks (Table 5.1) while outperforming the previous state-of-the-art (SOA) on MSDA (Table 5.3).

5.3 Proposed Method

5.3.1 Preliminaries

We aim to learn a classifier h from the input domain X to the output domain Y, $h : X \rightarrow Y$ with a small risk in the target domain, as measured by cross-entropy: $\min_h \mathbb{E}_{(x,y)\sim P^t} \ell_{CE}(h(x); y)$ where $\ell_{CE}(h(x); y) := -\langle y, \log h(x)\rangle$ is calculated using the one-hot ground-truth vector $y \in \{0, 1\}^K$ and class label estimates $h(x) \in \mathbb{R}^K$ from the output of a deep neural network with input x, and K is the number of classes. P^t is the distribution of the target domain. We do not have ground-truth labels in the target domain. So we leverage generative models described next.

StyleGAN follows [KAL17] in aligning coarse-layer outputs with the blurred real images in the early iterations to stabilize the training of the GAN. We do the same not for stability, but for learning a shared representation for both domains when generating these low-resolution images. We achieve this by not feeding the domain label d to the coarse layers of the decoder. Hence, the GAN loss aligns generated low-resolution images to both the real source and the real target samples with the corresponding class label. In a way, we learn a shared, low-resolution representation of two different domains for the given class label.

In StyleGAN, a latent factor $z \sim N(0, I)$ is fed to fully connected layers to learn Adaptive Instance Normalization (AdaIN) parameters for a generative adversarial network (GAN) decoder. No latent vector is used in the input of the decoder; instead, a constant latent input is updated with the same minimax loss as the rest of the decoder. The decoder has several AdaIN layers whose parameters are used to modify the image generated. Each AdaIN layer applies an affine transformation to a feature at layer i to get the transformed feature: $AdaIN(x_i, z) = \sigma_i \frac{x_i - \mu(x_i)}{\sigma(x_i)} + \mu_i$. That is, feature x_i is first standardized by applying Instance Normalization (IN) and then the learned σ_i and μ_i are applied to set the mean and dispersion parameter; $\sigma = [\sigma_1, \cdots, \sigma_L]$ and $\mu = [\mu_1, \cdots, \mu_L]$ which are learned from a fully connected network f (see the Mapping Network in Fig. 5.1): $w = [\sigma, \mu] = [w_1, \cdots, w_L] = f(z)$ and L is number of AdaIN layers. We refer to "coarse-layer" AdaIN parameters as those controlling lower resolutions in the output, e.g., $[\sigma_1, \cdots, \sigma_{L/2}]$ and $[\mu_1, \cdots, \mu_{L/2}]$. "Fine-layer" AdaIN parameters refer to the ones controlling

higher resolutions of the output, e.g., $[\sigma_{L/2+1}, \cdots, \sigma_L]$ and $[\mu_{L/2+1}, \cdots, \mu_L]$.

Given the class label $y = e_k \in \mathbb{R}^{1\times K}$, $k \sim [K]$, $[K] := \{0, ..., K-1\}$, the learned content vector is $w_c = f([Y, z])$ where $Y = yW_y$ and $[Y, z] \in \mathbb{R}^{1\times 2N}$.[1] The parameter $W_y \in \mathbb{R}^{K\times N}$ is introduced for matching the dimension of the latent $z \in \mathbb{R}^{1\times N}$. w_c can also be referred as coarse style as in [KLA19]. See the red blocks in Fig. 5.1.[2]

Similarly, we get the style vector that we feed to fine-layers from the domain label $d = e_m$, $m \sim [M]$: $w_s = f([D, z])$ where $D = dW_d$, $W_d \in \mathbb{R}^{M\times N}$ and $[Y, z] \in \mathbb{R}^{1\times 2N}$. In standard UDA benchmarks, the number of domains is $M = 2$ whereas for MSDA experiments $M = 5$. w_s is also referred as fine style [KLA19]. See the purple blocks in Fig. 5.1.

Now, we can write the generator g as a function of the content vector $w_c(z, k)$ and the style vector $w_s(z, m)$: $g(w_c, w_s)$; or, more conveniently as a function of the latent vector z, the class label $k \in [K]$ and the domain label $m \in [M]$: $g(z, k, m)$. In other words, we control the generation with three independent factors: class-label, domain-label and a random latent vector to generate diverse images for each domain- and class-label.

Note that in [KLA19], same AdaIN parameters are broadcasted to different layers. I.e. $w_1 = \cdots = w_L = f(z)$, so the coarse and the fine style parameters are the same for the given latent vector z. Since we are learning coarse and fine layer parameters from different inputs (y and d respectively), we do not broadcast the same parameters for all the layers. So, in our setting, $[w_1, \cdots, w_{L/2}] = w_c = f([Y, z])$ and $[w_{L/2+1}, \cdots, w_L] = w_s = f([D, z])$. [3]

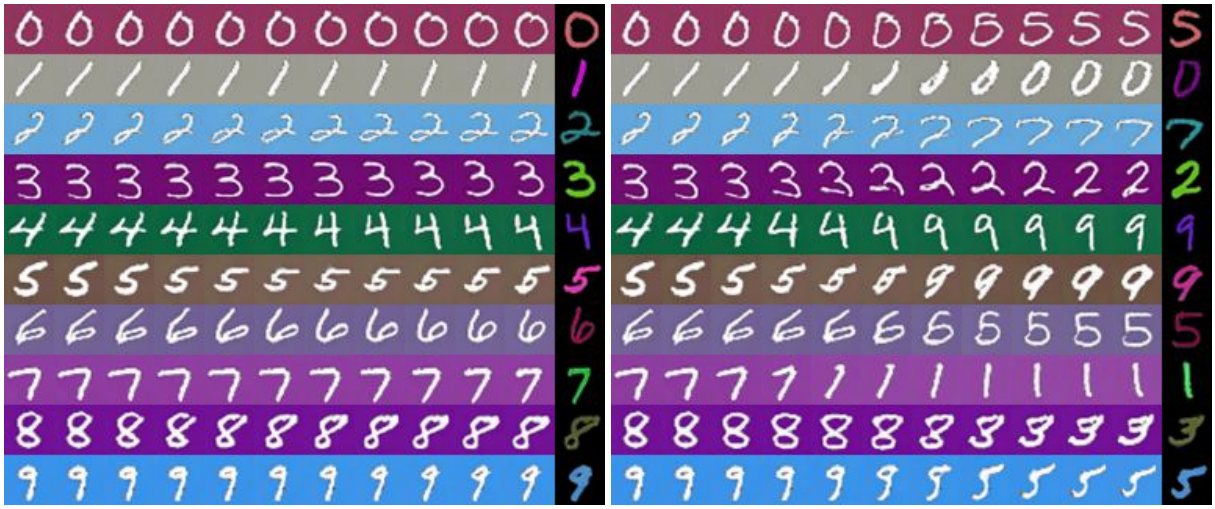
5.3.2 Pseudo-conditional GAN loss

To generate images with controlled domain and class labels, we construct a conditional GAN loss where the label space (for conditioning the joint discriminator) is the Cartesian product of the

[1] e_k is the identity of size K whose kth element is 1 and 0 elsewhere.

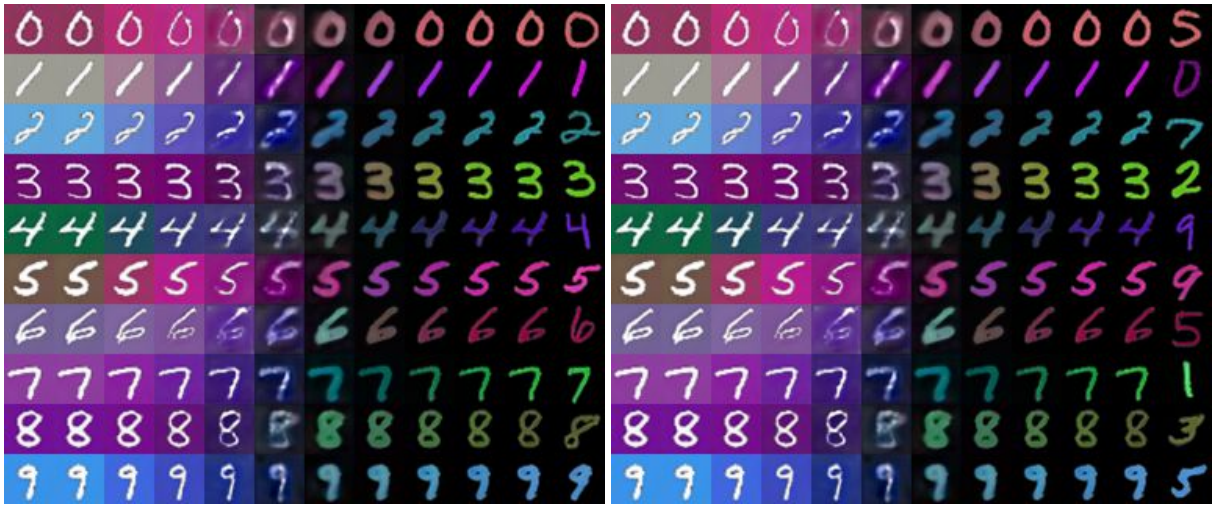
[2] Note that in [KLA19], both the coarse and fine layer parameters are referred as "style" parameters whereas in our context the former one is used to learn the class labels hence better term for it is "content".

[3] Even though it is possible to broadcast the learned parameters within the coarse or within the fine layers, we did not see any benefit in doing it, so we choose to relax this constraint.



MNIST-CB → MNIST-CD MNIST-CD → MNIST-CB

Figure 5.3: **Interpolation of the coarse layer parameters**. Images in the left and the rightmost columns of each panel are generated images from the source and the target domains respectively. Images in between them are generated by fixing the latent vector z that goes to fine layers while interpolating the coarse latent vector z from the source image to the target image. As a result, domain factors (background/digit colors) is the same as the source image. Other factors (e.g. rotation, boldness) change to match the target image. When the class labels are different for the source and the target images, the class labels also change to match the target image. These results verify that the coarse-layer parameters control the class label while not affecting the domain-dependent variations.



MNIST-CB → MNIST-CD MNIST-CD → MNIST-CB

Figure 5.4: **Interpolation of the fine layer parameters**. Images in the left and the right most columns of each panel are generated images from the source and the target domains respectively. This time, domain factors (background/digit colors) interpolate to match the target image. Other factors (e.g. rotation, boldness) are the same as the source image. When the class labels are different for the source and the target images, the class labels are kept the same as the source image. These results verify that the fine-layer parameters control the domain of generated images while not affecting the class label.

domain label set ($[M] := \{0, \cdots, M-1\}$) and class label set ($[K] := \{0, \cdots, K-1\}$), $[K] \times [M]$. For simplicity, we denote this as a set of scalars, $[MK] = \{0, \cdots, MK-1\}$. We define the "joint label" $j = mK + k \in [MK]$ for the given class label k and domain label m. The joint discriminator (the green block in Fig. 5.1) tries to distinguish fake and real images with the label $j \in [MK]$. The challenge in the UDA setting is that we do not have access to class labels k (thus joint label j) for the target samples. We will be relying on the pseudo-labels[4],

$$k(x^t) = \arg\max_k h(x^t)[k]. \tag{5.1}$$

We will first describe our pseudo-conditional GAN loss. Next, we will describe several losses that we use to train the classifier h.

For simplicity, we will assume the number of domains is $M = 2$ where $m = 0$ is for the source and $m = 1$ is for the target domain.[5] Non-saturating GAN loss [MGN18] is used as a conditional GAN loss where the generator g solves the following optimization problem

$$\min_g \mathbb{E}_{P(z,k,m)} \phi(-\psi(g(z,k,m))[mK+k]) \tag{5.2}$$

where $P(z,k,m) = N(z-0,I)\frac{I(k<K)}{K}\frac{I(m<M)}{M}$, I is indicator function and $\phi(x) = \text{softplus}(x) = \log(\exp(x)+1)$.

The joint discriminator ψ competes with the generator g by solving the following optimization,

$$\begin{aligned}\min_\psi \mathbb{E}_{P(z,k,m)} \phi(\psi(g(z,k,m))[mK+k]) + \mathbb{E}_{(x^s,e_k)\sim P^s} \phi(-\psi(x^s)[k]) \\ + \mathbb{E}_{x^t \sim P_x^t} \phi(-\psi(x^t)[K + k(x^t)])\end{aligned} \tag{5.3}$$

where K is added to $k(x^t)$ from Eqn. 5.1 as the last K entries of the discriminator output are devoted to the target samples.

Given that the target-domain images are generated whose class labels are controlled, we can minimize a classification loss on these generated images to train the classifier h. Then, the **t**arget **c**lassification loss is defined as,

$$L_{tc}(h,g) = \mathbb{E}_{P(z,k)} \ell_{CE}(h(g(z,k,1)), e_k). \tag{5.4}$$

[4]We use the notation $x[k]$ for indexing the value at the kth index of the vector x.

[5]The proposed method can also be applied to more than 2 domains.

Note that in the most general form, the classification loss on the generated target images is function of both the generator g and the classifier h: $L_{tc}(h, g)$. But, in the experiments, updating the generator parameters with this loss did not consistently improve the performance. So, we choose not to update the generator parameters with this loss.

For the source-domain samples, we do not need to use generated images for minimizing a classification loss as the class labels of the reals images are known. Then, the **s**ource **c**lassification loss is

$$L_{sc}(h) = \mathbb{E}_{(x,y)\sim P^s} \ell_{CE}(h(x), y). \tag{5.5}$$

5.3.3 Other UDA losses

To regularize the learning process further, we also apply the *explicit* regularization like entropy minimization as defined in Sec. 4.3.2. Minimizing entropy satisfies the cluster assumption only for Lipschitz classifiers [GB05]. We realize the Lipschitz condition by applying virtual adversarial training (VAT) [MMK15, MMK17] as described in Sec. 4.3.2.

We also find domain adversarial loss in the hidden layers $\theta(x)$ of the classifier h useful. The loss to minimize for aligning the hidden layer distributions of two domains with the help of a small (e.g. 2 FC layers) domain discriminator $\omega(x) \in \mathbb{R}^2$ is:

$$L_{dann}(\omega, \theta) = \mathbb{E}_{x\sim P_x^s} \ell_{CE}(\omega(\theta(x)), [1, 0]) + \mathbb{E}_{x\sim P_x^t} \ell_{CE}(\omega(\theta(x)), [0, 1]) \tag{5.6}$$

which is optimized in a mini-max fashion: $\min_\omega \max_\theta L_{dann}(\omega, \theta)$.

5.4 Empirical Evaluation

We first illustrate our method's ability to generate images whose domain and class labels can be controlled independently in Sec. 5.4.1. We then assess its benefits in the UDA benchmarks in Sec. 5.4.2 and Sec. 5.4.3.

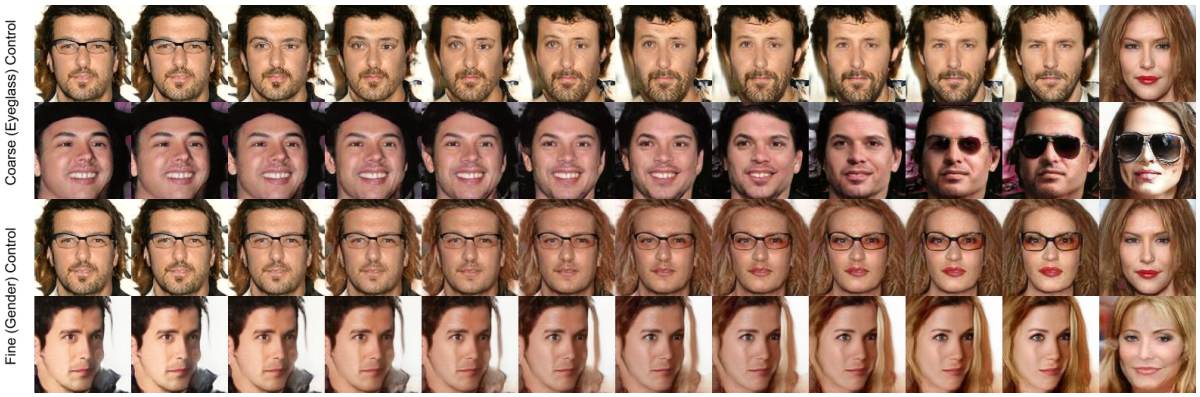


Figure 5.5: Same as Fig. 5.3-5.4 except this time for the CelebA dataset. The domain is defined by gender while the class is by eyeglasses. **Top.** An eyeglass is removed from the faces for the 1st row while it is added in the 2nd row. However, gender does not change in any of the rows. These results verify that the coarse-layer parameters control the class (eyeglass) of the generated image while not affecting the domain (gender). Factors other than gender and the eyeglass are not explicitly controlled by our training. So, the network chooses to learn most of them in the coarse layers. For instance, the pose is changing to match the target image in the last row. **Bottom.** Gender changes in all of the rows to match the target gender. Eyeglass is not removed even though the target image does not have eyeglasses in the first row. These results verify that the fine-layer parameters control the domain (gender) of the generated image while not affecting the class (eyeglass).

Figure 5.6: The first two rows are generated images at the intermediate layer of the decoder for the source and the target domains. 3rd and 4th rows are the corresponding generated images at the output of the decoder for the source and the target domains. Naturally, generated images in the intermediate layers are the replica of each other even for different domain labels (given the same latent z and the class label y). The generated source and the target images differ in the fine layers. The left panel is for SVHN $\rightarrow$ MNIST and the right panel is for SYN-DIGITS $\rightarrow$ SVHN.

5.4.1 Qualitative evaluation

Fig. 5.2 shows generated samples for both domains, and different class labels. Each row shows generated images for different realizations of the latent vector z. The upper half is for the images generated in the source domain, with the target domain in the lower half. Since the target class labels are not revealed during training, it is inevitable for the generator g to confuse some of the class-labels e.g. "6" and "8" are confused in MNIST $\rightarrow$ SVHN.

The MNIST-CB $\leftrightarrow$ MNIST-CD transfer [GWB18] can be used to assess the level of disentanglement of domain and class information: The domains are defined by the color of the digits and background, whereas the classes are the digits. Original 256×256 samples were downsized to 32×32 in pre-processing. We achieve 99.17 and 99.24 accuracy in MNIST-CB $\rightarrow$ MNIST-CD and MNIST-CD $\rightarrow$ MNIST-CB respectively; we did not add these results to Table 5.1 since no other SOA method reports them.

In Fig. 5.3, we first generate images of the source (left column) and the target domain (right column). Then, we fix the latent vector feeding the fine layers and linearly interpolate the coarse latent vector until it matches that of the target image. As can be seen in the left panel, interpolating coarse latent vector does not change the color of digits and background, as these layers do not encode domain information. However, pose, thickness, and other nuisance or "style" factors are

linearly interpolated to match the target image. In the right panel, we show the same experiment except that the target image has a different class label than the source image. The class label is also interpolated to match the target image as it is controlled in the coarse layers. Similarly, in Fig. 5.4, we fix the coarse latent vector and linearly interpolate the fine latent vector until it is matched to the target image. This time, both the color of the digit and the color of the background changes to match the target image as fine layers control the domain information. However, other variations like pose and thickness do not change with the fine layer interpolation. Moreover, as can be seen in the right panels, class labels do not change with the fine layer interpolation.

To show our method's ability to disentangle class and domain in the generation of more complex images, we repeat the above experiments with the CelebA [LLW15] dataset. Here, we split the dataset into two domains where men faces are labeled as source images and women faces are unlabeled target images.[6] We choose the presence of eyeglasses as the attribute (class) of interest. In Fig. 5.5, we show the interpolation results. Unlike what vanilla StyleGAN would do, gender here is controlled in fine-layers with our explicit regularization (bottom panel). Note that the network chooses to control pose in the coarse layers, which is natural as the pose is not explicitly regularized in neither the class (y) nor the domain (d). Eyeglass changes in the very first column for the 1st, and on the last columns for the 2nd row of Fig. 5.5. Even though the focus of this work is UDA, our method's ability to disentangle domain (gender) and class (eyeglass) in CelebA dataset – by learning these attributes at the different layers of the decoder – can be utilized in applications like image editing. For such applications, control of other variations (e.g. pose, identity) is essential hence labeled data [MVJ11] might be needed.

In Fig. 5.6, we show generated low-resolution images (1st and 2nd rows) and the corresponding full-resolution source (3rd row) and target images (4th row). As can be seen, a "shared" representation is learned, capturing the class label y for any given domain label input d. Domain differences become apparent only in the higher-resolution images. Since low-resolution aligning is only applied

[6]Even though women faces are unlabeled (i.e. we do not reveal whether there is en eyeglass on it or not), this does not pose an extra challenge for this particular task, because a binary eyeglass classifier trained only on men faces performs at 99.4% accuracy on women faces. The main purpose of these experiments is to show that we can control different attributes by changing the parameters going into different layers of the decoder.

Source dataset	MNIST	SVHN	CIFAR	STL	SYN-DIGITS	MNIST
Target dataset	SVHN	MNIST	STL	CIFAR	SVHN	MNIST-M
[GL14] DANN*	60.6	68.3	78.1	62.7	90.1	94.6
[GKZ16] DRCN	40.05	82.0	66.37	58.86	NR	NR
[SUH17] ATT	52.8	86.2	NR	NR	92.9	94.2
[FMF18] Π-model**	33.87	93.33	**77.53**	71.65	96.01	NR
[SBN18] DIRT-T +IN	76.5	**99.4**	NR	73.3	**96.2**	98.7
[LLY18] UFDN	NR	95.01	NR	NR	NR	NR
[LLY18] UFDN***	19.02	89.72	42.47	46.59	91.78	97.72
[LLY18] UFDN-aug***	22.23	93.77	42.15	46.78	91.84	97.66
[BSD17] PixelDA	NR	NR	NR	NR	NR	98.2
[RCT18] SBADA-GAN	61.1	76.1	NR	NR	NR	**99.4**
[WH18] GAGL	74.6	96.7	77.0	61.5	93.1	94.9
Ours	**84.84**	**98.78**	**77.35**	**74.83**	**95.3**	99.07
Source-only (baseline)	36.34	70.4	71.40	49.12	86.99	42.91

Table 5.1: **Comparison to SOA UDA algorithms on UDA image classification tasks.** Accuracies on the target test data are reported. Algorithms are trained on the entire labeled source training data and unlabeled target training data. NR stands for not reported. * DANN results are implementation of [SBN18] with instance normalized input. ** Results of [FMF18] with minimal augmentations are reported. UFDN, PixelDA, SBADA-GAN and GAGL are the other generative models. *** reports the performance of UFDN when we run the official implementation in different tasks. In UFDN***, we turn off data augmentation to have a fair comparison to other methods. In UFDN-aug***, augmentations are applied to follow the official implementation. The overall SOA methods are shown with bold and the SOA among the generative models are shown with bold and underline.

at the beginning of the training, generated low-resolution images do not have to be semantically meaningful. But, as we observe in Fig. 5.6, the network learns domain agnostic digit patters in the coarse layers with the help of the proposed regularization.

5.4.2 Quantitative assessment

We use the digit and object image classification benchmarks most common in UDA: CIFAR → STL, STL → CIFAR, MNIST → SVHN, SVHN → MNIST, SYN-DIGITS → SVHN and MNIST → MNIST-M. No data augmentation is used in any of the experiments, to ensure fair comparison against competing methods [FMF18, SBN18]; also, we have not used sophisticated architectures like ResNet [HZR16a]. In Table 5.1, we report the performance obtained when using source samples only. For each task, our method substantially improves upon the source-only baselines. The baseline score is lowest for MNIST → SVHN. This is a challenging task as MNIST images are greyscale, in contrast to color digits in SVHN. Also, SVHN images contain multiple digits while MNIST pictures contain a single digit.

We compare the proposed method to other generative models. The proposed method outperforms the previous generative models in all the tasks except MNIST → MNIST-M for which our method already performs close to perfect (99.07%). In MNIST → SVHN, our method achieves 84.84% which makes the relative error reduction 40.31% compared to the SOA generative method GAGL.

We compare against another generative model UFDN which is similar in controlling the domain label but differs in how do they encode the class information (implicitly, by encoding an image). UFDN only reports on SVHN → MNIST where the relative error reduction is 75.55% in SVHN → MNIST. For a more complete comparison, we run the official implementation in all the tasks. In the original code, augmentations were enabled, which we reported as UFDN-aug***. For a fair comparison, we also reported the results without augmentations (UFDN***). The relative error reductions against UFDN are 81.28%, 88.13%, 60.63%, 52.87%, 42.82%, 59.21% proving the effectiveness of the proposed independent class-label control.

We compare our results to discriminative methods as well: DIRT-T [SBN18] applied VAT to UDA after training with the domain-adversarial loss of [GL14] and VAT, further fine-tuning only on

Source dataset	MNIST	SVHN	CIFAR	STL	SYN-DIGITS	MNIST
Target dataset	SVHN	MNIST	STL	CIFAR	SVHN	MNIST-M
Source-only (baseline)	36.34	70.4	71.40	49.12	86.99	42.91
Without SSL and DANN regularizers	65.44	92.13	73.94	62.13	93.47	98.42
Without the proposed loss	56.62	78.47	74.48	55.13	91.85	98.20
Ours	84.84	98.78	77.35	74.83	95.3	99.07

Table 5.2: Ablations on loss functions. Accuracies on the target test data are reported. Algorithms are trained on the entire labeled source training data and unlabeled target training data. The second row (without SSL and DANN regularizers) shows the performance of the proposed algorithm when the SSL regularizers are not used. Even without these regularizers, the proposed loss (classification on the generated images) improves upon the source-only baselines in all 6 tasks with relative error reductions: 45.71%, 73.41%, 8.88%, 25.57%, 49.81%, 97.23%. The third row (without the proposed loss) shows the performance when using all the SSL regularizers but without the classification loss on the generated images described in Sec. 5.3.2. The overall method (fourth row) achieves higher scores than this stronger baseline.

the target data with the entropy and VAT objectives. ATT [SUH17] uses two networks trained on the source data and predictions of the networks are used as pseudo labels on the target data. Note that the best-performing methods for UDA do not involve a generative model, consistent with the challenge of using a generative model for unsupervised discriminative tasks. SVHN → MNIST, MNIST → MNIST-M and SYN-DIGITS → SVHN are the saturated tasks where the proposed method either surpasses DIRT-T or performs near it. The advantage of the proposed method compared to DIRT-T is most apparent in MNIST → SVHN where the relative error reduction is 35.49%.

In Table 5.2, we report the performance of the proposed algorithm when the SSL and DANN regularizers are not used. Even without the regularizers, the proposed method improves upon the source-only baselines in all 6 tasks with relative error reductions: 45.71%, 73.41%, 8.88%, 25.57%, 49.81%, 97.23%. We also report the results with SSL and DANN regularizer but without

Target dataset	SVHN	SYN-DIGITS	MNIST	USPS	MNIST-M
[XCZ18] DCTN	77.5	NR	NR	NR	70.9
[PBX18] M^{3}SDA	81.32	89.58	98.58	96.14	72.82
Ours	**90.71**	**98.91**	**99.65**	**97.20**	**98.45**
Source-only (baseline)	82.50	94.60	99.18	89.48	65.54
Relative error reduction to SOA	50.27%	89.54%	75.35%	27.46%	94.3%

Table 5.3: Comparison to state-of-the-art UDA algorithms on multi-source domain adaptation (MSDA) task "Digit-Five". Accuracy on the target test data are reported. Algorithms are trained on the 4 labeled source sets and 1 unlabeled target set. NR stands for not reported. The proposed method outperforms previous methods. Note that our baseline scores are better than the reported numbers in the earlier MSDA works. We found out that this is because of the classifier network they used. The classifier used in the earlier MSDA works has 3 FC layers whose dimensions are too large for digit tasks resulting in over-fitting. Our classifier is a standard, light-weight network used in earlier UDA works [FMF18].

the proposed classification loss on the generated images described in Eqn. 5.4. Overall results improve upon these stronger baselines as well.

5.4.3 Results on Multi-source Domain Adaptation

The "Digit-Five" dataset proposed by [XCZ18, PBX18] combines 5 digit datasets: MNIST, MNIST-M, SVHN, SYN-DIGITS, USPS. For each experiment, one of the five domains is chosen as the target domain in a leave-one-out setting. Following [XCZ18, PBX18], we sample 25000 images for training and 9000 images for testing from each dataset. For USPS, all of the images are used i.e. 7291 for training and 2007 for testing.

The proposed method can be applied to any number of domains with the modest expense of increased output dimension for the joint discriminator. We simply increase d dimension to the

MNIST-M SVHN SYN-DIGITS

Figure 5.7: Conditional image generation results of the proposed method on multi-source domain adaptation benchmarks. The first 8 rows and the last 2 rows of the panels are generated images for the labeled source domains and **unlabeled** target domain respectively. The class label input y is $0, \cdots, 9$ from the first row to the last row. Each row shows the generated images for the same domain label and the same class label with different realizations of the latent vector z. The generator can generate a diverse set of samples for each class in the target domain which is utilized by the classifier. The results are given for the following target domains: MNIST-M, SVHN, SYN-DIGITS.

number of domains: i.e., we increase the domain label dimension from 2 to 5 for DIGIT-Five [XCZ18, PBX18] experiments where the first $M-1$ entries of d are used for source domains and the last entry is for the target domain. So, say the domain label is $d = [0, 1, 0, 0, 0]$ then the generator produces fake images of the second source domain.

In Table 5.3, we report the performance of the proposed method along with the source-only baseline. Here the source-only baseline is only trained with the labeled source samples using the same network and training procedure as the actual method.[7] We outperform all the previous methods in this task. We outperform the SOA (M^{3}SDA) in all 5 tasks with error reduction rates of 50.27%, 89.54%, 75.35%, 27.46% and 94.3%.

[7]Note that, for SVHN, SYN-DIGITS, MNIST target settings, our baseline scores are higher than what was reported in the previous works. We reached out to the authors for a possible explanation of this. Apparently, the classifier network they used has a higher capacity than the one we used. Their network ends with 3 FC layers with dimensions: 8192 → 3072, 3072 → 2048, 2048 → K. Our network from [FMF18] has $4,430,474$ parameters for $K = 10$ while theirs has $31,795,146$. When we run our source-only baselines with their network, we get results closer to what was reported in [PBX18]. We observed that using their network causes overfitting in our experiments for such small datasets. So, we choose to use the smaller network that we used for all the other tasks.

Fig. 5.7 shows generated samples for different multi-source domain experiments. Each row shows generated images for different realizations of the latent vector z for the same domain and the class label. Rows $0-7$ are the generated images for 4 source domains while rows $8-9$ are for the target domain. When optimizing for the pseudo-conditional GAN loss, we sample the same number of source and target samples. This results in fewer samples per domain for source domains compared to the target domain. This choice has been made as the classification loss on target domain generations are minimized by the classifier h. As a result, the quality of generated images is better for the target domain. Artifacts can be observed especially for some of the generated SVHN source images in the left (rows 6-7) and the right (rows 0-1) panels of Fig. 5.7. This problem could be tackled by sampling more for the domains which are harder to generate. But, the focus of this work is to generate high-quality target images to be used in the training of the classifier h.

5.4.4 Implementation Details

The classifier network (h) and the domain discriminator network (ω) used in the experiments are given in Table 5.4. The domain discriminator (ω) is used to align the output of the last max-pool layer of the classifier (h). For all the other networks; namely the mapping (f), the decoder (g), and the joint discriminator (ψ), we followed the implementation of [KLA19]. Instance norm is only used in MNIST $\leftrightarrow$ SVHN experiments. No weight decay or batch norm is used. In all the experiments, Adam optimizer with the fixed learning rate of 0.001 is used. In all the experiments, training is stopped after the parameters are updated with 15M samples. Batchsize of 128 is used for both the source and the target samples during training and reduced to 64 after 1.8M samples. The classification loss on the generated images started to be minimized after 2M samples.

5.5 Style-Mixing in Supervised Setting

Here, we repeat the results of the style-mixing experiments from [KLA19] in our multi-domain setting where *new* images are generated by mixing the *learned* styles of *two other generated images*. Experiments, conducted in the *supervised* setting, verify the *implicit* regularization of the network

3×3 convolution, 128 lReLU
3×3 convolution, 128 lReLU
3×3 convolution, 128 lReLU
2×2 max-pool, stride 2
dropout with probability 0.5
3×3 convolution, 256 lReLU
3×3 convolution, 256 lReLU
3×3 convolution, 256 lReLU
2×2 max-pool, stride 2
dropout with probability 0.5
3×3 convolution, 512 lReLU
1×1 convolution, 256 lReLU
1×1 convolution, 128 lReLU
Global average pooling, $6 \rightarrow 1$
Fully connected layer: $128 \rightarrow$ K
Softmax

(a) Classifier (h)

Fully connected layer: $\lvert\theta\rvert \rightarrow 100$
lReLU
Fully connected layer: $100 \rightarrow 2$
Softmax

(b) Domain discriminator (ω)

Table 5.4: The classifier network (h) and the domain discriminator network (ω) used in the experiments.

structure.

We train a StyleGAN [KLA19] architecture in a multi-domain setting, with the generative process conditioned on the ground-truth class labels. When we mix the learned styles of different generated images, they look realistic [KLA19], and the mixed styles follow certain interesting patterns (See Fig. 5.8): The domain of a generated image is determined by the domain of the image from which we take the fine style while the class label of a generated image is determined by the class label of the image from which we take the coarse style. Even though this experiment is conducted in the supervised setting, there is no explicit incentive for the network to disentangle the domain and class variability. The network is simply trained with a standard conditional GAN loss [MO14, OOS17] where *the condition is the class label only*. Since the domain label is not fed to the network – for this experiment only – there is no incentive for the network to differentiate, say, MNIST from SVHN images with the same class label. This shows that the structure proposed in the StyleGAN provides us with an implicit way of disentangling domains from classes which can then be used for domain-controlled image generation in the unsupervised setting. *Note that in [KLA19], and in our motivational experiment (Fig. 5.8), style-mixing experiments (generating images by mixing the learned parameters for different decoder layers) are conducted as post-mortem analysis whereas, in the proposed method, we explicitly impose what to control at each layer during training.*

The disentanglement of domain and class labels via StyleGAN is purely an empirical observation validated for UDA digit datasets both through style-mixing experiments and from the empirical success of the proposed method on UDA benchmarks. However, the same idea does not hold if we simply swap the domain and class set definitions i.e. if we consider different digits (e.g. 5,6) as different domains and SVHN-MNIST as different class labels. Because with the current definition of class and domain labels, the shift between domains is mostly textural (e.g. color differences). So, learning such variations in the fine layers are easier compared to learning digit patterns. Hence, the network can learn different domains using fine-layers only. We show more ablation studies on this in Sec. 5.6. That is what [KLA19] also observed where the network chooses to learn the color scheme and microstructure in the fine layers without explicit regularization. Therefore, our proposed method works better under the assumption that the difference between the source and

Target dataset	SVHN	SYN-DIGITS	MNIST	USPS	MNIST-M
Number of coarse layers is 1 out of 8	87.36	97.57	99.35	95.41	82.52
Number of coarse layers is 7 out of 8	87.80	95.85	99.42	96.01	84.16
Number of coarse layers is 4 out of 8	**90.71**	**98.91**	**99.65**	**97.20**	**98.45**

Table 5.5: Ablations on the number of coarse/fine layers. We report the performance when using different number of layers (1,4,7) in StyleGAN decoder as the coarse layer. For instance, if the number of coarse layers is 1: for the first layer, the class label is fed and for the remaining 7, the domain label is fed. We get the best results when half of the layers are used as coarse layers.

the target domains are limited to textural shift. Such a similarity assumption across domains is necessary for any UDA method.

5.6 Ablations

In Table 5.5, we report the performance of the network based on the number of layers we use for feeding class and domain information. We get the best performance when we use half of the layers for feeding the class label and the other half is used for the domain label. The ablation study is reported for each MSDA task.

We consider using style-mixing for UDA under different settings. A natural choice was to generate unlabeled target domain samples as additional $(K+1)$th label by augmenting the class label y to $K+1$ dimensions instead of having an additional domain label d. I.e. $\overline{y} \in \mathbb{R}^{K+1}$ is fed as an input to the generator instead of feeding class $y \in \mathbb{R}^K$ and domain $d \in \mathbb{R}^M$ separately from different layers. For instance, if $\overline{y} = e_K$ then the decoder is expected to generate target domain images of an arbitrary class label and if $\overline{y} = e_0$ then it is expected to generate source domain images with class label 0.

In this case, style-mixing did not give desirable results as seen in the left panel of Fig. 5.9 for SVHN $\rightarrow$ MNIST task. As a comparison, we give the results in the same UDA setting for

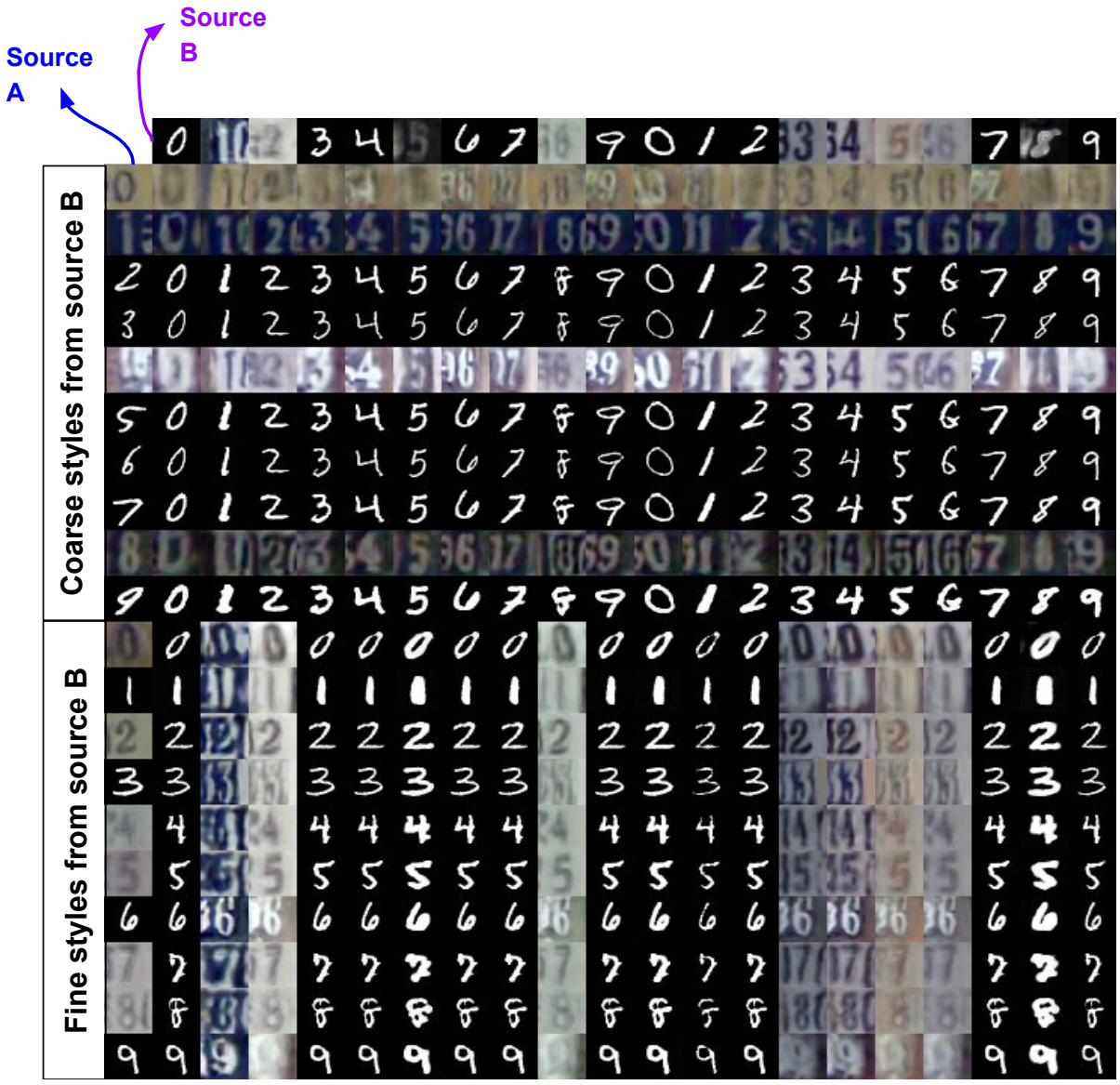
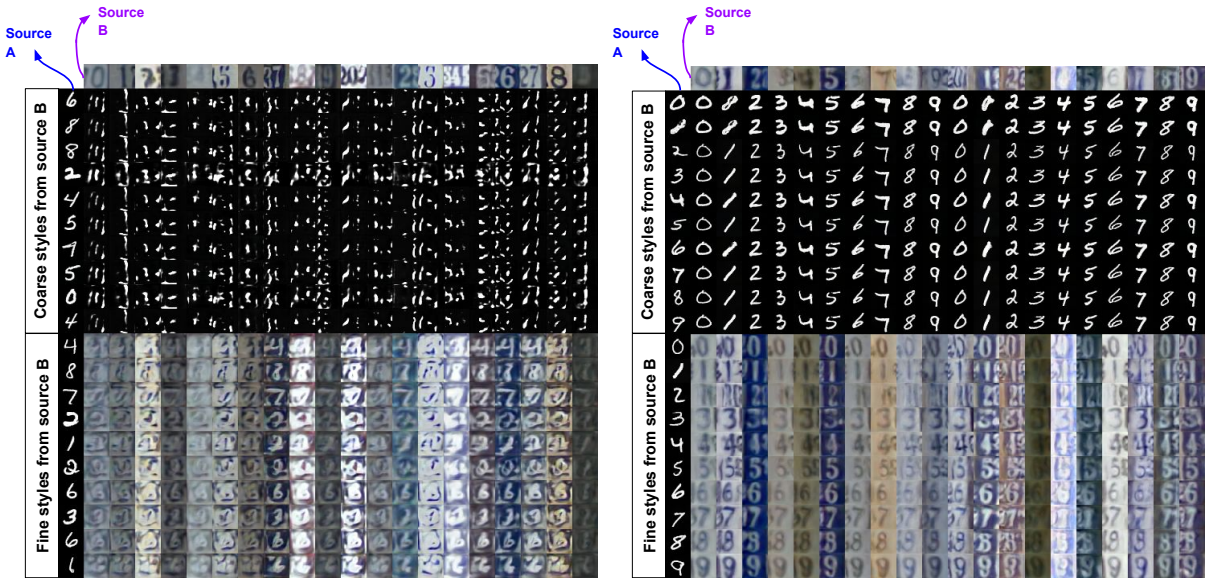


Figure 5.8: **Mixing learned styles.** Images in the first column (source A) and the first row (source B) are generated from their latent codes. The rest of the images are generated by taking half of the learned styles from source A and the rest from source B. We can observe a clear pattern in the generated images: The domain (class) of a generated image is determined by the domain (class) of the image from which we take the fine (coarse) style. Even though this motivational experiment is conducted in the supervised setting, the fact that no explicit regularization is used to have such disentanglement encouraged us to use StyleGAN architecture for UDA. In the UDA setting, we do not have enough constraint on unlabeled target samples so we explicitly encode class label information into coarse layers while domain label is fed to fine layers. *Note that this is the only supervised experimental result in this chapter.*



$\overline{y} \in \mathbb{R}^{K+1}$ ($K+1$ is for the target domain) $y \in \mathbb{R}^{K}$ and $d \in \mathbb{R}^{M}$ (proposed)

Figure 5.9: **Style-mixing in UDA setting.** Same as Fig. 5.8 except for this time SVHN → MNIST UDA setting. That is, the labels for the MNIST samples are not known. We compare the proposed method where the domain labels and the class labels are fed (right panel) with another way of generating unlabeled samples that we considered in our early trials (left panel). In this setting, we do not feed the class labels and the domain labels as two different inputs. Instead, the unlabeled samples are generated as if we had an additional (K+1)th class. It can be seen style-mixing gives realistic-looking images for the proposed method whereas it does not give any meaningful results for the other setting. Note that MNIST '8'" in the third row and first column of the right panel was supposed to be "1". The confusion is natural because the networks are trained in the unsupervised setting for these experiments.

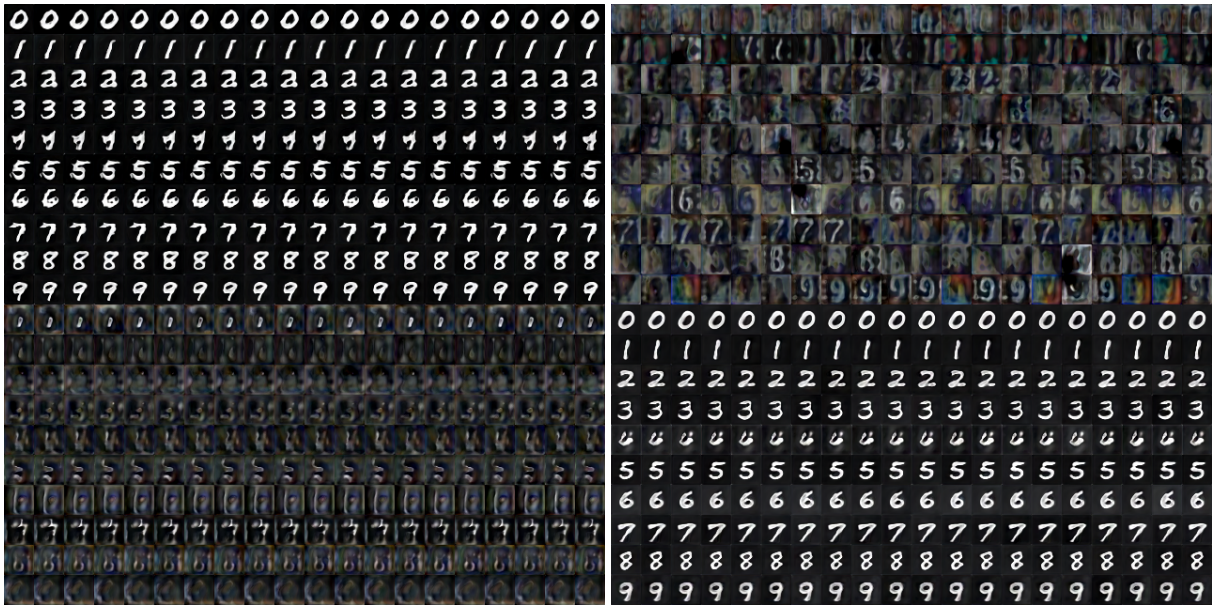


Figure 5.10: Image generation results when domain labels are fed to coarse layers and class labels are fed to fine layers (opposite of the proposed approach). The decoder fails to generate realistic SVHN images for both tasks MNIST → SVHN (left) and SVHN → MNIST (right), and we observe a mode collapse for generated MNIST images.

the proposed approach in the right panel of Fig. 5.9. Note that in the first column of the right panel, generated MNIST images are supposed to have labels from $0, \cdots, 9, 0, \cdots, 9$ and with one exception (first generated "1" looks like "8") generated images have the correct labels. Furthermore, few SVHN images generated by using the coarse latent vector of the MNIST "2" in the lower half of the right panel do not have label 2.

We cannot control the class label of the generated target images for the left panel as they are simply generated by feeding $\overline{y} = e_K$. So, the generated images in the first column of the left panel do not have any pattern in terms of their class labels. The expectation in this setting was to get MNIST looking images with the class label of the mixed SVHN image when we combine the coarse latent vector of an SVHN image and the fine latent vector of an MNIST image. But, mixed images do not give sensible results. This experiment validates our choice of explicit layer-wise control of class and domain information when using StyleGAN for UDA.

In Fig. 5.10, we further validate our choice of learning the domain variations in the fine-layers.

We swap the content and style vectors by feeding class labels to fine layers and domain labels to coarse layers. In this case, the network cannot even learn to generate realistic-looking SVHN images and there is a mode collapse for MNIST images. As discussed in Sec. 5.5, StyleGAN is better at learning textural differences in the fine layers. Hence, as expected, the network could not learn the shapes (digits) in the fine layers and textural differences (domain variations) in the coarse layers even when we explicitly regularize it.

5.7 Limitations: Dense Prediction Tasks

(a) Segmentation map. (b) Source image. (c) Target image.

Figure 5.11: **Disentangled image generation is an ill-defined approach for the segmentation task.** Segmentation map (a), the corresponding generated source (b), and the generated target images (c) are given. Source and target distributions are learned from GTA5 [RVR16a] and Cityscapes [COR16] datasets. The *generated* target image has palm trees even though in real Cityscapes (Germany) samples, there is no palm tree.

We focus on classification tasks and we propose an algorithm to approximate $\mathbb{P}(x|y,d)$ with the motivation of *independent* control of class label y and domain label d so that we can leverage the labeled samples of the source domain by switching domain label input d from *source* to *target*. In other words, we approximate $\{x|x = g(y, d = \text{target}, z)\}$ from $\{x|x = g(y, d = \text{source}, z)\}$ by learning a one-to-one mapping between the two sets. Necessary condition for having such a

mapping is independence of y and d.

However, for dense pixel prediction tasks like segmentation, independent control of segmentation map and domain label (source or target) is obscure as y and d are not independent in the first place.

Consider GTA5→Cityscapes task which is used to evaluate UDA segmentation works and consider an example segmentation map $y = \text{palm}$ from GTA5 (in Los Angeles) with palm trees. By just looking at this segmentation map with palm tree, you can tell that it is definitely from the source domain (GTA5, Los Angeles) as there is no palm tree in Cityscapes (Germany) i.e. $\mathbb{P}(d = \text{target}|y = \text{palm}) = 0$. Hence, $\mathbb{P}(y = \text{palm}, d = \text{target}) = \mathbb{P}(y = \text{palm})\mathbb{P}(d = \text{target}|y = \text{palm}) = 0 \neq \mathbb{P}(y = \text{palm})\mathbb{P}(d = \text{target})$.

We illustrate this idea in Fig. 5.11. StyleGAN is not designed for segmentation; instead, we trained state-of-the-art segmentation-conditioned image generation model GauGAN [PLW19] on GTA5 [RVR16a] and Cityscapes [COR16]. As can be seen in Fig. 5.11b and Fig. 5.11c, GauGAN manages to generate highly realistic source and target images which respect the input segmentation label given in Fig. 5.11a. As we enforce segmentation label consistency, generated Cityscapes image has also palm trees. Then, can we really claim that the generated image is actually from the target distribution (e.g. Cityscapes)?

Using these generated images, our segmentation network (DeepLabv2 [CPK17]) could not perform better than the baseline model. This matches with our expectations because in dense pixel prediction tasks (e.g. segmentation), class label (e.g. segmentation map) y and domain label d are *dependent*. Hence, independent generative control of class and domain labels are neither possible nor useful for these tasks.

5.8 Conclusion

We propose to explicitly disentangle the content and the domain features by encoding class labels into the coarse layers of the decoder and domain labels into the fine layers of the decoder. We have illustrated its use in unsupervised domain adaptation, where the "style" represents the domain variable, and the "content" the variable of interest. Beyond the improvements in the discriminative

task, the approach affords the flexibility of controlled sampling of intrinsic and nuisance variability. The original StyleGAN paper is feeding the same learned latent vector to all hidden layers of the generator for the task of image generation. Control of different generator layers with different parameters (class and domain in our case) is a novel idea even beyond the UDA literature and can be potentially explored for other, possibly supervised tasks.

One drawback of having a generative model is the training time is longer than a standard discriminative model. For the proposed method, the entire training takes about 40 hours using a single Nvidia Tesla V100. Moreover, the idea of independent control of class and domain variables is limited to classification tasks and cannot be trivially extended to dense pixel prediction tasks (e.g. semantic segmentation).

CHAPTER 6

Spatial Class Distribution Shift in Unsupervised Domain Adaptation: Local Alignment Comes to Rescue

6.1 Introduction

In this chapter, we focus on the UDA segmentation task where the goal is to estimate the segmentation map $y \in \{0,1\}^{K \times H \times W}$ for a given RGB image where K is the number of classes. For the source samples, we have access to ground-truth labels $y^s \in Y$ which are used to minimize cross-entropy loss[1],

$$L_{ce}(P^s; f) := \mathbb{E}_{(x^s,y^s)\sim P^s} \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} \ell_{CE}(f(x^s)_{ij}; y^s_{ij}) \tag{6.1}$$

where f is the segmentation network and P^s is the distribution of source domain samples and corresponding labels, $\ell_{CE}(f(x)_{ij}; y_{ij}) := -\langle y_{ij}, \log f(x)_{ij} \rangle$ is calculated using the one-hot ground-truth vector $y_{ij} \in \{0,1\}^K$ and class label estimates $f(x)_{ij} \in \mathbb{R}^K$ from the output of a deep neural network with input x. Next, we discuss the related work for leveraging unlabeled target data and our contribution in relation to this vast literature (Sec. 6.2). In the following section, we describe the proposed method (Sec. 6.3). Finally, we put it to the test on UDA segmentation benchmarks (Sec. 6.5).

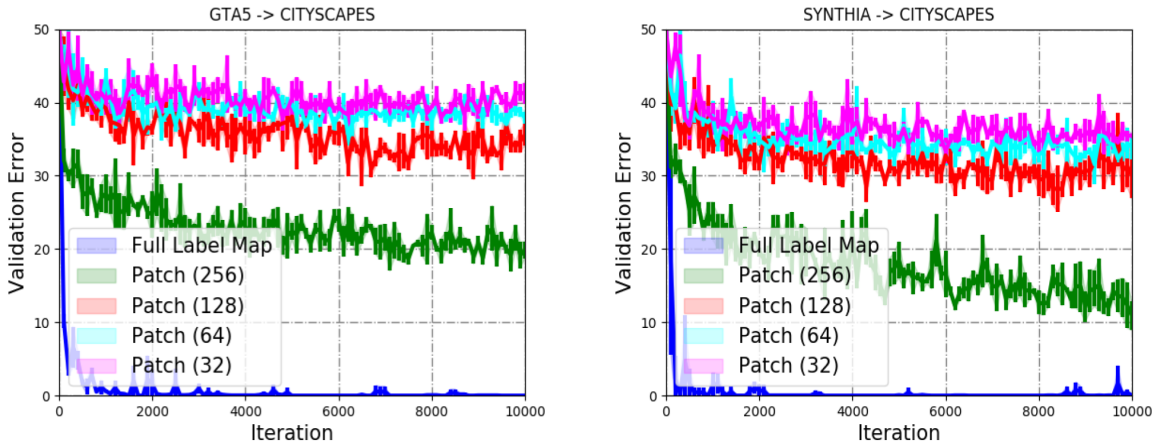


Figure 6.1: **Spatial-class distribution shift correlates with the receptive field on the segmentation maps.** Validation errors for a binary classifier trained to distinguish binary domain labels from segmentation maps are given for GTA5 → Cityscapes (left) and SYNTHIA → Cityscapes (right). If the domain gap between segmentation maps are large, then error decays faster. We repeat this experiment for different receptive fields. When the binary classifier is trained on the entire segmentation maps (blue curve), errors decay quickly, whereas, for smaller patch sizes, learning slows down. For patch sizes smaller than 128, predictions of the classifiers are close to luck (50%). Errors for SYNTHIA are slightly lower due to the larger spatial-class shift between SYNTHIA and Cityscapes. This experiment verifies that even when the spatial class distribution shift is large between the global segmentation maps, local segmentation maps are still almost indistinguishable.

Figure 6.2: **Samples from the datasets.** Spatial-class distributions vary from source to target domains for the UDA segmentation benchmarks; namely, SYNTHIA → Cityscapes and GTA5 → Cityscapes. SYNTHIA images are generated with random camera views, unlike Cityscapes which only have dashcam views. On the other hand, in GTA5, there are unrealistic scenarios e.g. ego-vehicle driving on the sidewalk.

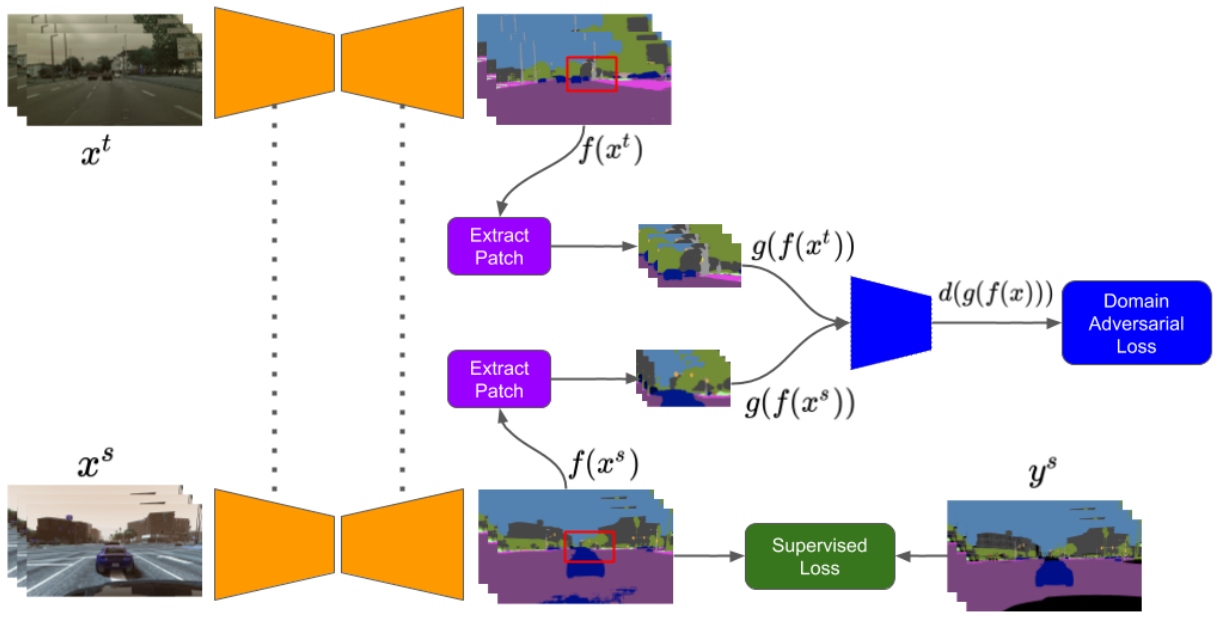


Figure 6.3: The network structure of the proposed approach. Our binary domain discriminator (blue) acts on the *random* patches of predictions ($g(f(x))$) and not on the global segmentation maps ($f(x)$).

6.2 Related Work

In the following, we present some of the previous works on the UDA segmentation tasks excluding the ones focusing on different tasks [CS19b, CS19a, CXW20a] for the sake of space. AdvEnt [VJB18] is the baseline method. It is observed that in the source-only training, entropy is mostly low for the source predictions and only high at the edges, while the entropy is mostly high on the predictions of the target images. Hence, they proposed to align the "weighted self-information" to minimize the entropy of target predictions while aligning them to source predictions. We improved this work in an orthogonal direction by performing a random-patch alignment. [HWY16, CCC17] proposed heuristics to have class-conditional alignments. [ZDG17] follows a curriculum learning approach: where they sequentially learn pseudo-labels for the entire image, superpixels, and finally the dense predictions.

[IZZ17] proposed to use a fully convolutional network (PatchGAN) for image translation which is later employed in several UDA works [THS18]. However, unlike PatchGAN that divides the image into a fixed collection of patches, we randomize the location of patches. Therefore, the distributions we align using the domain discriminator have supports defined by the values of these prediction patches. While the cardinality of the patch set exponentially increases with the size of the image for the proposed method, PatchGAN always uses a small subset of this set. Hence, the aligned distributions for the proposed method and PatchGAN are significantly different.

Work of [CLV18] looks similar to ours as they partition images into 3 by 3 regions before alignment. But actually, it has the exact opposite motivation and outcome as they only align the corresponding patches while we choose our partitions completely randomly. This saddle difference has great importance. Our motivation for aligning "random" patches is that the spatial-class distribution across domains can vary a lot whereas for them the motivation is to leverage the hypothesis that corresponding patches have similar spatial distributions. But, in reality, as camera views are random in SYNTHIA, corresponding patches should not be aligned to those of Cityscapes.

[1]We denote label and prediction corresponding to a pixel coordinates of (i, j) with $y_{ij} \in \{0, 1\}^K$ and $f(x)_{ij} \in \mathbb{R}^K$ respectively.

Method	Road	SW	Build	Wall*	Fence*	Pole*	TL	TS	Veg.	Sky	PR	Rider	Car	Bus	Motor	Bike	mIoU	mIoU-13
Source	14.9	11.4	58.7	1.9	0.0	24.1	1.2	6.0	68.8	76.0	54.3	7.1	34.2	15.0	0.8	0.0	23.4	26.8
MCD [SWU18]	84.8	43.6	79.0	3.9	0.2	29.1	7.2	5.5	83.8	83.1	51.0	11.7	79.9	27.2	6.2	0.0	37.3	43.5
Source	55.6	23.8	74.6	-	-	-	6.1	12.1	74.8	79.0	55.3	19.1	39.6	23.3	13.7	25.0	-	38.6
AdaptSegNet [THS18]	84.3	42.7	77.5	-	-	-	4.7	7.0	77.9	82.5	54.3	21.0	72.3	32.2	18.9	32.3	-	46.7
CLAN [LZG19]	81.3	37.0	80.1	-	-	-	16.1	13.7	78.2	81.5	53.4	21.2	73.0	32.9	22.6	30.7	-	47.8
MinEnt [VJB18]	73.5	29.2	77.1	7.7	0.2	27.0	7.1	11.4	76.7	82.1	57.2	21.3	69.4	29.2	12.9	27.9	38.1	44.2
AdvEnt [VJB18]	87.0	44.1	79.7	9.6	0.6	24.3	4.8	7.2	80.1	83.6	56.4	23.7	72.7	32.6	12.8	33.7	40.8	47.6
A+E [VJB18]	85.6	42.2	79.7	8.7	0.4	25.9	5.4	8.1	80.4	84.1	57.9	23.8	73.3	36.4	14.2	33.0	41.2	48.0
Source	64.3	21.3	73.1	2.4	1.1	31.4	7.0	27.7	63.1	67.6	42.2	19.9	73.1	15.3	10.5	38.9	34.9	40.3
CBST [ZYK18]	68.0	29.9	76.3	10.8	1.4	33.9	22.8	29.5	77.6	78.3	60.6	28.3	81.6	23.5	18.8	39.8	42.6	48.9
MRL2 [ZYL19]	63.4	27.1	76.4	14.2	1.4	35.2	23.6	29.4	78.5	77.8	61.4	29.5	82.2	22.8	18.9	42.3	42.8	48.7
MRENT [ZYL19]	69.6	32.6	75.8	12.2	1.8	35.3	23.3	29.5	77.7	78.9	60.0	28.5	81.5	25.9	19.6	41.8	43.4	49.6
MRKLD [ZYL19]	67.7	32.2	73.9	10.7	1.6	37.4	22.2	31.2	80.8	80.5	60.8	29.1	82.8	25.0	19.4	45.3	43.8	50.1
LRENT [ZYL19]	65.6	30.3	74.6	13.8	1.5	35.8	23.1	29.1	77.0	77.5	60.1	28.5	82.2	22.6	20.1	41.9	42.7	48.7
Ours	90.6	51.34	81.96	11.77	0.32	29.51	11.72	12.38	82.69	84.7	58.57	24.73	81.94	36.37	17.11	41.75	**44.84**	**51.99**

Table 6.1: Comparison to SOA on SYNTHIA → Cityscapes. All models are trained on the labeled source training data (SYNTHIA) and unlabeled target training data (Cityscapes) and performances on Cityscapes validation split are reported. A+E [VJB18] refers to the ensemble of two networks: one trained with the adversarial loss and the other is with entropy minimization. In the literature, two different mIoU scores are reported for this task: one is for 16 common classes between two domains (mIoU) and the other is for the 13 classes (mIoU-13) excluding *wall*, *fence*, *pole*. Our method outperforms all the previous methods in both metrics.

[ZYV18, ZYL19] updates network parameters using the pseudo-labels for which the network is confident. They incorporate spatial priors into the proposed CBST framework, leading to CBST with spatial priors (CBST-SP) by counting the class frequencies in the source domain, followed by smoothing with a Gaussian kernel to approximate the frequency of each class at a spatial location in the image space. Then, they simply modulate the network output with this spatial prior. Again, this idea contradicts the fact that the spatial distributions of segmentation maps differ across domains.

[LLD19] combined previous works of curriculum [ZDG17] and self training [ZYV18]. Instead of super-pixels, they use patches of sizes 4 and 8. The key idea is to alternatively update labels and network weights. They apply average pooling on the predictions and pseudo labels and minimize a classification loss between them. Our approach fundamentally differs from this as we align the predictions on the source and the target images whereas they align the predictions of the pre-trained network (providing pseudo-labels) and the main network both on the target images. Hence, unlike us, their algorithm may not align the local-prediction distributions across domains, which is the main motivation of this work.

The proposed method is orthogonal to most of the methods from this rich and multi-faceted context and can be improved by incorporating some of these ideas.

6.3 Proposed Method

As can be seen in Fig. 6.2, spatial-class distribution can greatly differ from source to target domains, due to scene structure, camera view changes, etc. This contradicts with the idea of aligning the segmentation network outputs globally via minimax losses.

Before describing the proposed method, we conducted a motivational experiment to verify and quantify this hypothesis. For this purpose, we train a binary domain classifier on the ground-truth segmentation maps to measure the identifiability of the domain label from segmentation maps. See Fig. 6.1 where the left panel is for GTA5-Cityscapes and the right one is for SYNTHIA-Cityscapes. When the task is to distinguish the global segmentation maps, classifiers can easily detect the domain (blue curves). As we decrease the receptive field on the segmentation maps, by cropping

smaller patch sizes, it is getting harder for the classifier to find the correct domain label. For very small patch sizes, the performance of the classifier is close to chance (50%). Details on the training is given in the Sec. 6.4.

This experiment quantifies and verifies two almost obvious claims: First, the global segmentation maps can have different distributions across domains (i.e. spatial-class distribution shift) hence one should not align the global segmentation predictions at the training time. Second, even if the spatial-class distribution is very large (e.g. SYNTHIA → Cityscapes), the local segmentation maps can have similar distributions across domains. Assuming this as a fact for any cross-domain task -which we only verify for UDA segmentation benchmarks-, one should align random patches of predictions at the training time, for network predictions on different domains to abide by this phenomenon.

6.3.1 Loss Functions

A natural choice for aligning the cropped prediction distributions for the source and the target domains is to optimize a domain adversarial loss on the extracted patches from the segmentation predictions:

$$L_{adv}(P_x^s, P_x^t; f, d) := \mathbb{E}_{x^s \sim P_x^s, x^t \sim P_x^t} \ell_{CE}\Big(\psi(x^s), [0,1]\Big) + \ell_{CE}\Big(\psi(x^t), [1,0]\Big) \tag{6.2}$$

where $\psi(x) := d(g(f(x))$, g randomly extracts a patch of size $i < H$ and $j < W$ from the segmentation prediction $f(x)$, and ℓ_{CE} is cross entropy loss. P_x^s, P_x^t are marginal distributions of the source and the target domains. $d : x \mapsto \mathbb{R}^2$ is binary domain discriminator (see Fig. 6.3).

As in the previous work of [VJB18], instead of applying domain adversarial loss on the segmentation maps $y \in \{0,1\}^{K \times H \times W}$ directly, we found aligning the "self-information maps", $\overline{y} = h(y) \in \mathbb{R}^{K \times H \times W}$ where $\overline{y}_{kij} = h(y_{kij}) := -y_{kij} \log y_{kij}$ more effective.[2] Hence, the final

[2]Note that this is not exactly entropy of the predictions as the y_{kij} terms are not summed over the class dimension k. But, the source sample predictions have low entropy as cross-entropy is minimized on them. Adversarial alignment results in aligned $\overline{y}_{kij}$ distributions and thus, results in low entropy for the target predictions as well. As in [VJB18], we found this weighted-scheme more effective because this adversarial loss promotes both the low entropy target predictions and aligned prediction maps across domains.

objective function becomes,

$$L_{advent}(P_x^s, P_x^t; f, d) := \mathbb{E}_{x^s \sim P_x^s, x^t \sim P_x^t} \ell_{CE}\Big(\overline{\psi}(x^s), [0,1]\Big) + \ell_{CE}\Big(\overline{\psi}(x^t), [1,0]\Big) \tag{6.3}$$

where $\overline{\psi}(x) := d(g(h(f(x))))$. Then, the overall optimization problem solved by the segmentation network f is,

$$\min_f \max_d L_{ce}(P^s; f) - \lambda L_{advent}(P_x^s, P_x^t; f, d). \tag{6.4}$$

Since there is no closed form solution, the objective function is optimized by the segmentation network f and the domain discriminator d in an alternating fashion using SGD.

6.4 Implementation Details

6.4.1 Datasets.

To evaluate the performance of the proposed method, we put it to test on the standard UDA segmentation benchmarks: GTA5 → Cityscapes and SYNTHIA → Cityscapes and compare against SOA and baseline methods.

GTA5 dataset consists of 24966 images with a resolution of 1914 × 1052 and collected from the video game based on the city of Los Angeles. The resolution of the images in the target set Cityscapes is 2048 × 1024. The number of target training images is 2975. Methods are tested on the 500 samples of the validation split of Cityscapes. For GTA5 → Cityscapes, 19 common classes are used. These are the same classes as the ones used in Cityscapes benchmark [COR16] where rare classes are excluded from the evaluation.

SYNTHIA [RSM16] is generated by rendering a virtual city created with the Unity development platform. RANDCITYSCAPES subset of SYNTHIA is used as the source training set. This subset consists of 9400 frames of the city taken from a virtual array of cameras moving randomly. We choose the 16 overlapping classes between SYNTHIA and Cityscapes following the earlier works [ZYK18, ZYV18, HWY16, ZDG17]. Classes that do not exist in this setting, compared to GTA5 setting are *terrain*, *truck*, *train*. There is another setting only considering 13 classes excluding the

Method	Road	SW	Build	Wall	Fence	Pole	TL	TS	Veg.	Terrain	Sky	PR	Rider	Car	Truck	Bus	Train	Motor	Bike	mIoU
Source	42.7	26.3	51.7	5.5	6.8	13.8	23.6	6.9	75.5	11.5	36.8	49.3	0.9	46.7	3.4	5.0	0.0	5.0	1.4	21.7
CyCADA [HTP17]	79.1	33.1	77.9	23.4	17.3	32.1	33.3	31.8	81.5	26.7	69.0	62.8	14.7	74.5	20.9	25.6	6.9	18.8	20.4	39.5
Source	36.4	14.2	67.4	16.4	12.0	20.1	8.7	0.7	69.8	13.3	56.9	37.0	0.4	53.6	10.6	3.2	0.2	0.9	0.0	22.2
MCD [SWU18]	90.3	31.0	78.5	19.7	17.3	28.6	30.9	16.1	83.7	30.0	69.1	58.5	19.6	81.5	23.8	30.0	5.7	25.7	14.3	39.7
Source	75.8	16.8	77.2	12.5	21.0	25.5	30.1	20.1	81.3	24.6	70.3	53.8	26.4	49.9	17.2	25.9	6.5	25.3	36.0	36.6
AdaptSegNet [THS18]	86.5	36.0	79.9	23.4	23.3	23.9	35.2	14.8	83.4	33.3	75.6	58.5	27.6	73.7	32.5	35.4	3.9	30.1	28.1	42.4
CLAN [LZG19]	87.0	27.1	79.6	27.3	23.3	28.3	35.5	24.2	83.6	27.4	74.2	58.6	28.0	76.2	33.1	36.7	6.7	31.9	31.4	43.2
MinEnt [VJB18]	84.4	18.7	80.6	23.8	23.2	28.4	36.9	23.4	83.2	25.2	79.4	59.0	29.9	78.5	33.7	29.6	1.7	29.9	33.6	42.3
MinEnt + ER [VJB18]	84.2	25.2	77.0	17.0	23.3	24.2	33.3	26.4	80.7	32.1	78.7	57.5	30.0	77.0	37.9	44.3	1.8	31.4	36.9	43.1
AdvEnt [VJB18]	89.9	36.5	81.6	29.2	25.2	28.5	32.3	22.4	83.9	34.0	77.1	57.4	27.9	83.7	29.4	39.1	1.5	28.4	23.3	43.8
A+E [VJB18]	89.4	33.1	81.0	26.6	26.8	27.2	33.5	24.7	83.9	36.7	78.8	58.7	30.5	84.8	38.5	44.5	1.7	31.6	32.4	45.5
Source	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	29.2
FCAN [ZQY18]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	46.6
Source	71.3	19.2	69.1	18.4	10.0	35.7	27.3	6.8	79.6	24.8	72.1	57.6	19.5	55.5	15.5	15.1	11.7	21.1	12.0	33.8
CBST [ZYK18]	91.8	53.5	80.5	32.7	21.0	34.0	28.9	20.4	83.9	34.2	80.9	53.1	24.0	82.7	30.3	35.9	16.0	25.9	42.8	45.9
MRL2 [ZYL19]	91.9	55.2	80.9	32.1	21.5	36.7	30.0	19.0	84.8	34.9	80.1	56.1	23.8	83.9	28.0	29.4	20.5	24.0	40.3	46.0
MRENT [ZYL19]	91.8	53.4	80.6	32.6	20.8	34.3	29.7	21.0	84.0	34.1	80.6	53.9	24.6	82.8	30.8	34.9	16.6	26.4	42.6	46.1
MRKLD [ZYL19]	91.0	55.4	80.0	33.7	21.4	37.3	32.9	24.5	85.0	34.1	80.8	57.7	24.6	84.1	27.8	30.1	26.9	26.0	42.3	**47.1**
LRENT [ZYL19]	91.8	53.5	80.5	32.7	21.0	34.0	29.0	20.3	83.9	34.2	80.9	53.1	23.9	82.7	30.2	35.6	16.3	25.9	42.8	45.9
Ours	89.72	32.54	82.19	31.27	25.12	30.11	38.0	24.64	84.68	41.36	76.59	60.25	29.2	86.27	39.05	51.43	1.37	29.07	39.73	46.98

Table 6.2: Comparison to SOA on GTA5 → Cityscapes. Same as Table 6.1 except for GTA5 → Cityscapes. Here, the results are reported on 19 common classes (mIoU). The proposed method outperforms 3 of 4 scores that previous SOA [ZYL19] reported and it is only 0.12% less than the best method (MRKLD) of [ZYL19] which selectively samples for hard classes. The proposed method can be combined with MRKLD, but we choose to report naked results to show the effectiveness of the proposed *random*-patch alignment.

classes *wall*, *fence* and *pole* [THS18, CCC17]. Mean scores corresponding to these 13 classes are reported as mIoU-13.

6.4.2 Training details.

We used ResNet101-based DeepLabv2 [CPK17] without CRF post-processing [KK13] to have a fair comparison with SOA methods [ZYK18, ZYL19]. We compare our method with the ones reporting in the same setting where they do not exploit more advanced segmentation networks like DeepLabv3+ [CPS17, CZP18], thus we exclude [ZZL19, LLD19] from the comparison. This is

Method	Road	SW	Build	Wall*	Fence*	Pole*	TL	TS	Veg.	Sky	PR	Rider	Car	Bus	Motor	Bike	mIoU	mIoU-13
Source-only	57.18	26.41	72.05	6.29	0.15	25.56	8.87	11.12	74.06	80.6	53.69	12.39	49.46	5.25	7.66	20.39	31.95	36.86
AP-CI	33.12	21.75	58.02	0.21	0.0	10.82	0.0	0.37	53.43	42.52	24.12	0.58	24.16	1.22	0.01	1.24	16.97	20.04
AGP-GI	83.42	38.16	77.33	4.34	0.17	25.5	6.2	6.82	77.81	83.97	55.29	18.76	79.13	40.55	15.86	31.62	40.31	47.3
ALP-GI (proposed)	90.6	51.34	81.96	11.77	0.32	29.51	11.72	12.38	82.69	84.7	58.57	24.73	81.94	36.37	17.11	41.75	44.84	51.99
Relative to source-only	33.42	24.93	9.91	5.48	0.17	3.95	2.85	1.26	8.63	4.1	4.88	12.34	32.48	31.12	9.45	21.36	12.89	15.13

Table 6.3: Ablations on SYNTHIA → Cityscapes. We compare the performance of the proposed method to the following baselines. (1) The source-only model is only trained on the labeled source examples minimizing the cross-entropy loss. (2) In AP-CI (Align Predictions of Cropped Images), *RGB images are cropped instead of prediction maps*. (3) AGP-GI (Align Global Predictions of Global Images) refers to minimizing the same adversarial loss (Eqn. 6.3) *on the global segmentation maps*. The proposed method, ALP-GI (Align Local Predictions of Global Images) outperforms all baselines with 15.13%, 31.59% and 4.69% (in mIoU) compared to Source-only, AP-CI (Align Predictions of Cropped Images) and AGP-GI (Align Global Predictions of Global Images) respectively. The last row shows the relative increase compared to the source-only baseline.

necessary to make fair comparison possible as DeepLabv3+ includes a decoder replacing bilinear interpolation with atrous convolutions. This results in better recovering of the object boundaries. Intersection Over Union (IoU) has been the standard evaluation metric for semantic segmentation task: $IoU = \frac{TP}{TP+FN+FP}$ where TP, FN and FP correspond to true positive, false negative and false positive respectively. Then, mean IoU (mIoU) is calculated by averaging IoU of all the classes. Pixel accuracy, $\frac{TP+TN}{TP+TN+FN+FP}$ also considers TN (pixels correctly identified as not belonging to the class). This is not a good metric if some classes are seen in a few pixels only. A trivial solution would be to never estimate such classes. So, we also do not report on this metric.

Batch size is set to be one due to memory constraints. Hence, batch norm parameters are updated with momentum but current batch statistics are not used during training. Image width and height are resized to (760, 1280) for SYNTHIA, (720, 1280) for GTA5 and (512, 1024) for Cityscapes during training. The number of training iterations is 150000. SGD and Adam are used for optimizing segmentation and discriminator networks respectively. Finally, the proposed method adds no computational cost to the baseline method and it takes approximately 25 hours to train with

Method	Road	SW	Build	Wall	Fence	Pole	TL	TS	Veg.	Terrain	Sky	PR	Rider	Car	Truck	Bus	Train	Motor	Bike	mIoU
Source-only	75.46	24.6	65.09	11.91	10.58	28.19	27.45	14.64	79.71	32.04	70.56	52.26	20.1	71.91	28.7	48.5	1.42	16.89	37.36	37.76
AP-CI	44.41	14.72	58.47	12.14	1.0	16.64	1.28	1.67	70.77	12.19	65.4	41.38	0.16	27.9	9.43	13.93	0.0	0.25	0.0	20.62
AGP-GI	86.62	10.48	81.79	27.41	17.29	25.19	29.36	14.85	84.27	34.7	78.16	57.3	28.3	83.65	31.93	35.52	0.15	22.59	21.05	40.56
ALP-GI (proposed)	89.72	32.54	82.19	31.27	25.12	30.11	38.0	24.64	84.68	41.36	76.59	60.25	29.2	86.27	39.05	51.43	1.37	29.07	39.73	46.98
Relative to source-only	14.26	7.94	17.1	19.36	14.54	1.92	10.55	10	4.97	9.32	6.03	7.99	9.1	14.36	10.35	2.93	-0.05	12.18	2.37	9.22

Table 6.4: Ablations on GTA5 → Cityscapes. Same as Table 6.3 except for GTA5 → Cityscapes. The proposed method, ALP-GI (Align Local Predictions of Global Images) outperforms all baselines with 9.22%, 26.36% and 6.42% (in mIoU) compared to Source-only, AP-CI (Align Predictions of Cropped Images) and AGP-GI (Align Global Predictions of Global Images) respectively. The last row shows the relative increase compared to the source-only baseline.

a single Nvidia Tesla V100.

6.4.3 Training details for the experiment in Fig. 6.1.

The implementation details for the motivational experiment are as follows. We report validation errors after each 100 training iterations. We randomly choose 500 samples from the source domains for validation and did not use them during training. Cityscapes have already the validation split of size 500 samples. In total, 1000 samples are used to calculate the validation errors. Errors are averaged over three runs. Deviations over different runs are shaded but in some regions, they are too small to be visible. Standard classifier, ResNet18 [HZR16a] is used as a binary classifier. Label maps are resized to (512, 1024) before and after cropping, to have the same size segmentation maps for both domains. SGD with momentum 0.9, weight decay 10^{-4}, and fixed learning rate of 10^{-3} is used.

6.5 Empirical Evaluation

6.5.1 Quantitative Evaluation

In Table 6.1-6.2, we compare the proposed method against SOA methods. The proposed method especially shines on SYNTHIA → Cityscapes as for this task, spatial-class distribution shift is larger (See Fig. 6.1-6.2).

Our method surpasses all the previous SOA methods in both metrics of SYNTHIA → Cityscapes. Previous SOA [ZYL19] applies mining on rarely predicted classes and manages to get relatively high scores even in the very challenging classes. For instance, our method could only achieve 1.37% in *train* class of GTA5 setting, while previous SOA [ZYL19] could perform 26.9%. Similarly, for *fence* class of SYNTHIA setting, we perform poorly. Domain shift for segmentation maps and RGB images of these classes is too large for achieving robust performance in these classes.

Other methods we compare are as follows. FCAN [ZQY18] which proposed to combine the image alignment and translation losses. FCAN does not report class-wise performance and only report on GTA5. [THS18] applied domain adversarial loss both at the hidden layers and the network outputs. [SWU18] encourages the consistency of different classifiers by having one encoder and two classifiers. Both classifiers are trained on the labeled source samples. The distance between predictions of two classifiers on the same target sample is minimized by the encoder and maximized by classifiers. [LZG19] leverages the consistency between two classifiers in a different way. If two classifiers agree on the prediction, they keep the adversarial loss weight for that prediction small. In both tasks, we outperform all these methods which are orthogonal to ours and it could be combined with them but here we report the naked results to highlight the role of *random*-patch alignment.

The closest apple-to-apple comparison to our method is with [VJB18] AdvEnt which applies the same loss without random cropping on the predictions. The proposed method improves baseline AdvEnt method [VJB18] from 47.6% to 51.99% in SYNTHIA → Cityscapes and 43.8% to 46.98% in GTA5 → Cityscapes. Moreover, A+E reported in [VJB18] is an ensemble of two networks trained with different losses, so it is not directly comparable to our method. Nonetheless, the relative accuracy improvements to their reported numbers are 3.15% for GTA5 and 8.12% for SYNTHIA.

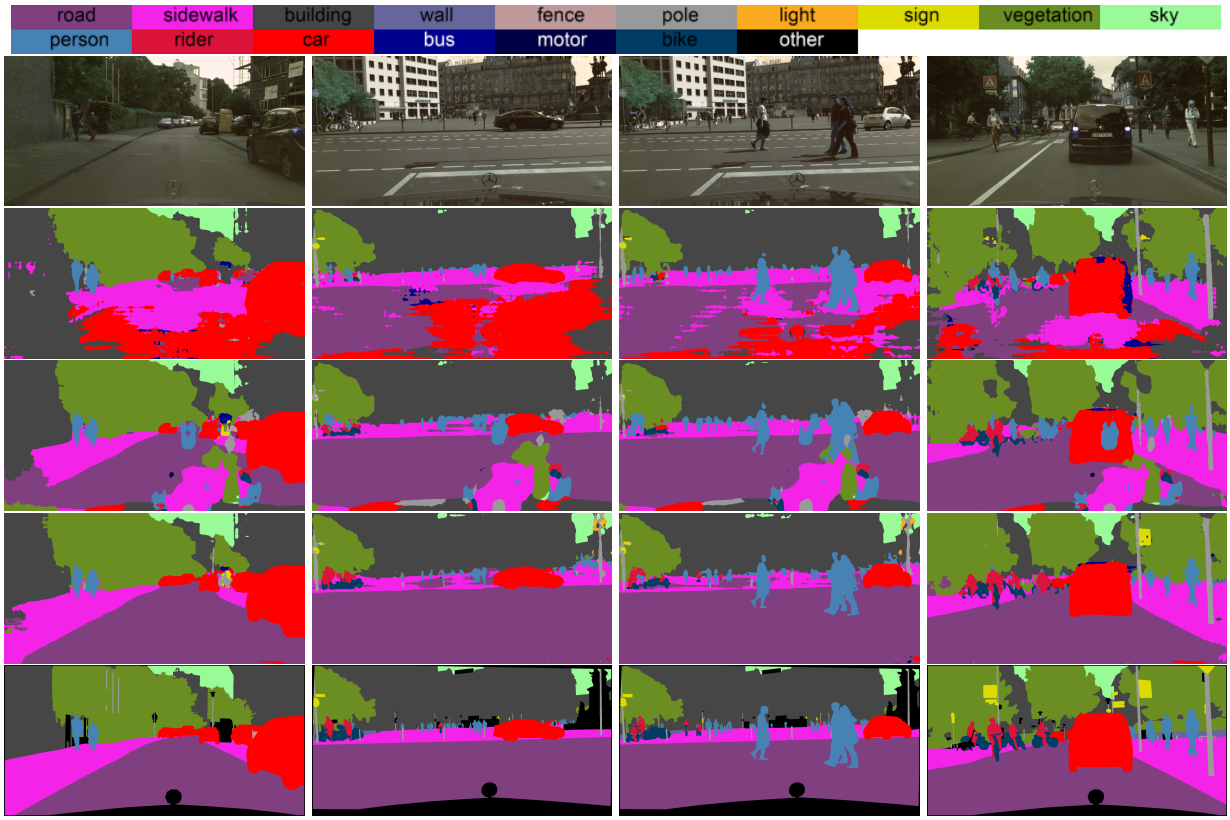


Figure 6.4: **Qualitative results for SYNTHIA → Cityscapes**. Visuals of Cityscapes test set predictions are presented along with the corresponding RGB images. From top to bottom: (1) RGB image, (2) source-only prediction, (3) global alignment prediction, (4) our prediction and (5) ground truth segmentation. The proposed method especially performs well on the more common classes like *road* or *sidewalks* whereas it misses some of the small and rare objects like *traffic signs*. More controlled ablations are discussed next.

In Table 6.3-6.4, we compare the proposed method Align Local Predictions of Global Images (ALP-GI) against the following baselines: (1) Source-only, (2) Align Predictions of Cropped Images (AP-CI) and (3) Align Global Predictions of Global Images (AGP-GI).

Source-only baselines are only trained on the labeled source samples minimizing the cross-entropy loss. Our method improves source-only baselines with 15.13% and 9.22% for SYNTHIA → Cityscapes and GTA5 → Cityscapes respectively. Since the network and other training details are the same for all the methods, the improvement verifies the effectiveness of the proposed loss in leveraging the unlabeled target samples. Similarly, the proposed method improves AGP-GI

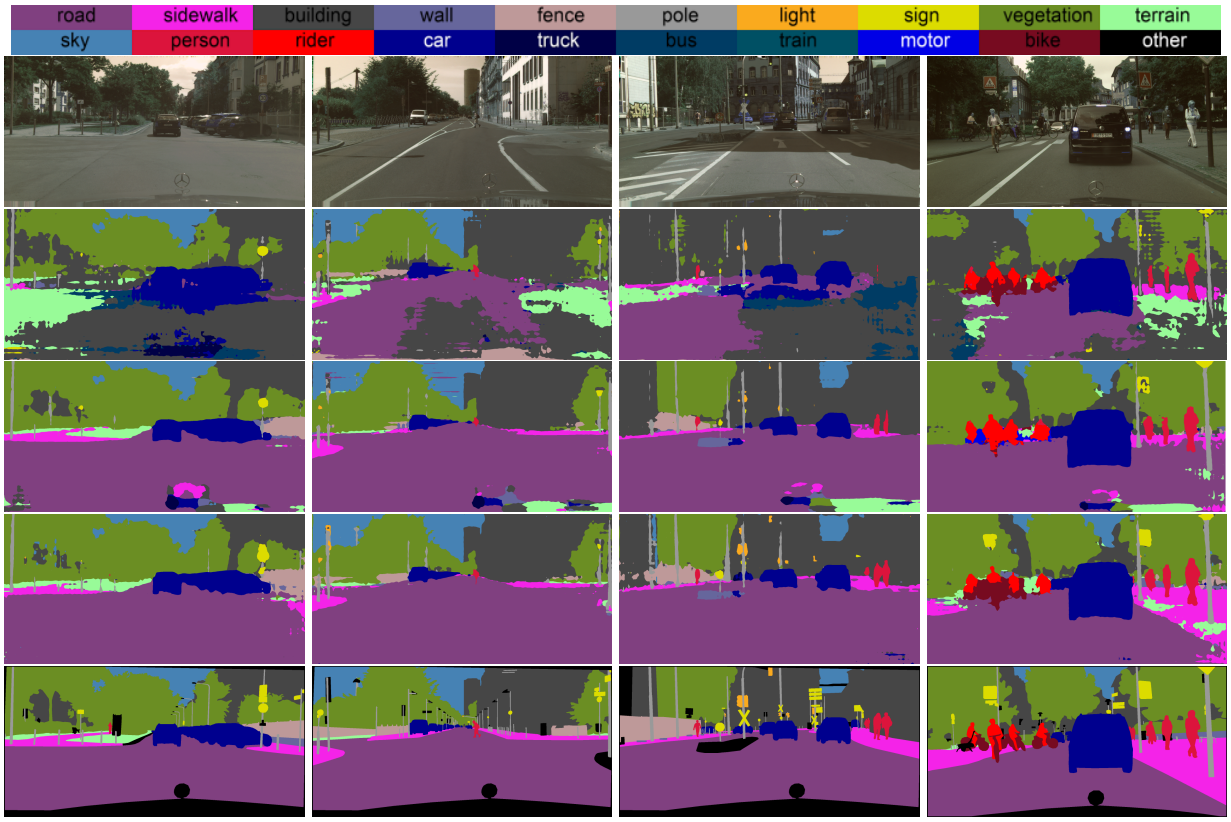


Figure 6.5: **Qualitative results for GTA5 → Cityscapes**. Same as Fig. 6.4 except for GTA5 → Cityscapes. Again, network can correctly capture objects belonging to classes *road*, *car* while missing tiny objects (e.g. *traffic light*, *traffic sign*) or classes with large domain gap (e.g. *fence*).

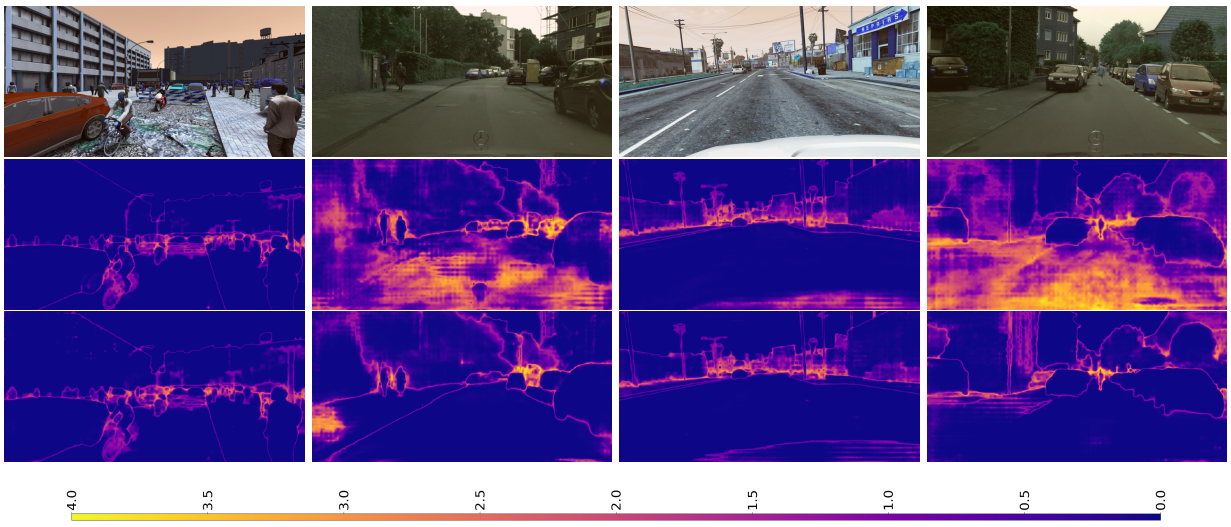


Figure 6.6: **Entropy of predictions**. The entropy of the predictions on the source samples (columns 1,3) and the target test samples (columns 2,4) are given. From top to bottom: RGB image, the entropy of the source-only trained model and the entropy of the proposed method. Left two columns are for models trained on SYNTHIA → Cityscapes and right two columns are for GTA5 → Cityscapes. Color transitions from purple to yellow as the value of entropy increases. Entropy values are low on the source images for both the source-only and the proposed models (column 1,3) except edges due to the cross-entropy loss. For the target test samples (column 2,4), the entropy of predictions are small for the proposed method thanks to the adversarial loss while source-only models have high uncertainty on the target images.

(Align Global Predictions of Global Images) baselines with 4.69% and 6.42% for SYNTHIA → Cityscapes and GTA5 → Cityscapes respectively. The improvement compared to AGP-GI verifies the significance of the *random*-patch alignment. Note that the results reported here for AGP-GI are slightly lower than those reported in [VJB18]. The difference from AdvEnt-only (ours is 47.3, theirs is 47.6) is due to implementation differences. Moreover, they report the best results by ensembling two different networks (A+E) whereas we report a single network's predictions for each method for having a controlled-experimental setting.

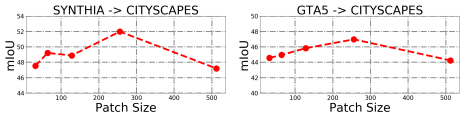
Another way to align the prediction patches is simply to feed the cropped images to the network. But, the problem with this approach is that the network cannot leverage the scene information (i.e. larger context) when inferring the semantic segmentation map for small crop sizes. That is

why in the proposed method, we minimize the cross-entropy loss on the entire images, and for the adversarial loss, *we randomly extract the patches of the predictions and not of the RGB images*. For completeness, we also evaluate this choice and report AP-CI (Align Predictions of Cropped Images) in Table 6.3,6.4 where we compare the proposed method to simply cropping the RGB images with the same crop sizes. This baseline gives terrible results for the patch size of 256 as such a small receptive field makes it hard for the network to correctly capture the scene. As a result, the performance of this baseline is even lower than the source-only baseline which does not leverage any target sample. The proposed method surpasses this baseline with 31.59% and 26.36% for SYNTHIA → Cityscapes and GTA5 → Cityscapes respectively.

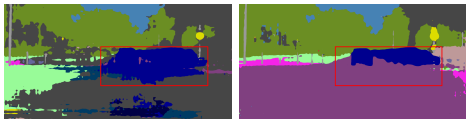
For SYNTHIA → Cityscapes, our method outperforms the source-only model for all the classes as can be seen in the last row Table 6.3. But, the improvements are especially significant for the classes *road*, *sidewalk*, *car*, *bus* where accuracies (IoU) increased from 57.18, 26.41, 49.46, 5.25 to 90.6, 51.34, 81.94, 36.37 respectively. These are more common classes and the shapes of these objects do not significantly differ from one domain to another. Hence, the proposed *random*-patch alignment method can leverage the object shapes learned from the source data. For GTA5 → Cityscapes, the proposed method improves the source-only baseline in all classes except *train*, which is a challenging class for this task as objects belonging to *train* class in GTA5 are far away from the ego-vehicle and they are hardly perceivable. The advantage of the proposed method is most apparent in the classes *building* and *wall* where IoU scores increased from 65.09 and 11.91 to 82.19 and 31.27 respectively.

6.5.2 Qualitative Evaluation

In Fig. 6.4 and 6.5, we present several qualitative results for SYNTHIA → Cityscapes and GTA5 → Cityscapes. In each figure, predictions of the source-only, global alignment, and the proposed methods along with corresponding RGB images and ground-truth segmentation maps are given. Black regions in the ground-truth maps belong to *other* class which are not evaluated at the test time. Thanks to the proposed random-patch alignment regularization, the network learns the shape of the objects like *car*, *traffic signs*, and corrects the mistakes of the global alignment method. Even



(a) Performance as a function of the patch size.



(b) The proposed method learns the shape of the objects from the labeled source domain.

Figure 6.7: **(a)** mIoU on the target test samples are given for models trained to align different patch size predictions. The left panel is for SYNTHIA → Cityscapes and the right one is for GTA5 → Cityscapes. Both models achieve the best results when aligning the predictions of size 256. For smaller and larger patch sizes, the performance of the model decays. **(b)** Blue regions denoted with the red rectangles are estimated as a *car*. Such a *car* shape does not exist in any of the source segmentation patches, hence unless we perform the proposed adversarial loss, a discriminator can easily tell apart domains only by looking at the segmentation maps. So, this prediction will be corrected with the proposed loss. On the other hand, we do not promote global segmentation map alignment unlike previous works as the global segmentation distributions are not necessarily the same across domains.

though the proposed method is quite successful in capturing the common classes *road*, *sidewalks* accurately, sometimes it can miss tiny and rare objects (e.g. belonging to class *fence*).

In Fig. 6.6, we give the entropy of predictions for the source-only baseline and the proposed method for both tasks along with the corresponding RGB images. As expected, entropy values of the predictions on the target test set (column 2,4) are less for the proposed method thanks to the adversarial loss. Entropies have high values mostly on the edges. For source images (column 1,3), both models have small uncertainty as both minimize cross-entropy loss on them.

In Fig. 6.7a, we plot the performance on the target test set as a function of patch size. We get the best results when aligning the prediction crops of size 256 for both tasks. Based on the experiment in Fig. 6.1, for this size of segmentation maps, a strong discriminator can have predictions that are better than luck. However, still, the discriminator cannot reduce validation error below 20% (green curve) unlike the global alignment case (blue curve) where the validation error quickly drops to 0%. Moreover, aligning the predictions with this crop size is sufficient to have aligned predictions for

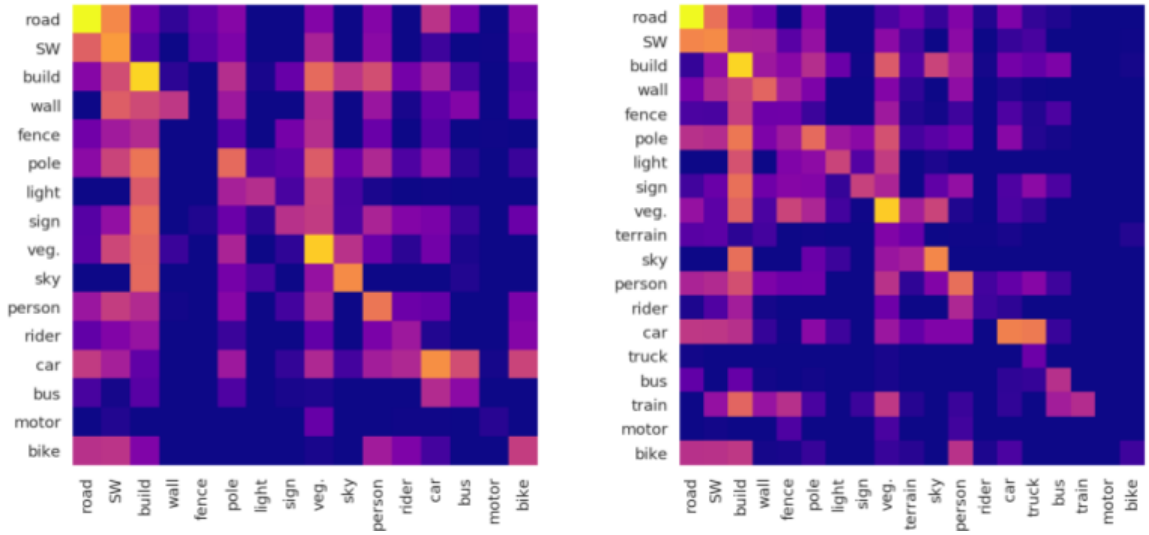


Figure 6.8: **Confusion matrices**. Log-scaled confusion matrices for SYNTHIA → Cityscapes (left) and GTA5 → Cityscapes (right) are given. Confusion matrices are calculated by averaging over all the target test set. The value of the matrix at row i and column j is equal to the number of observations that should be classified as i and predicted to be j. As the value increases, color changes from purple to yellow. Networks are confused between *sidewalk* and *road* classes in both tasks. The classes *building* and *vegetation* are attracting classes that networks tend to misclassify objects belonging to other classes as one of two.

smaller path sizes. In Fig. 6.7b, we present an illustrative example of how the proposed adversarial loss helps to learn the shapes of the objects from the labeled source domain and results in improved predictions compared to the source-only predictions.

6.5.3 Failure Cases

In Fig. 6.8, we give log-confusion matrices on the target test set predictions of the proposed method for both tasks. As can be observed, some classes are more likely to be confused (e.g. sidewalks and road). Furthermore, the false-positive ratio is high for some classes like building and vegetation (i.e. network is tempted to predict objects belonging to other classes as one of the two).

Our method performs poorly for some classes e.g. fence, train. As can be seen in Fig. 6.9,

Figure 6.9: **Samples from the hardest classes**. Sample RGB images and binary segmentation maps for the hardest classes are given. Left two columns are for the class *fence* (hardest class for SYNTHIA → Cityscapes) and the right two columns are for *train* (hardest class for GTA5 → Cityscapes).

domain shift for segmentation maps and RGB images of these classes is too large for achieving robust performance in these classes.

6.5.4 Results on Berkeley Deep Driving

In this section, we evaluate the proposed local alignment method on the Berkeley Deep Driving dataset [YXC18]. Berkeley Deep Driving dataset which consists of drive-cam images with resolution 1280×720. Frames are collected at 30 FPS and each 10th frame is annotated. We used the extended version of this dataset by [YXC18], BDD100K and not the earlier version [HWY16]. In this version, the number of samples is 7,000, 1,000 and 2,000 for training, validation and test splits respectively. We used the training split for training and the validation split for reporting the performance. We did not use the test split. Categories are compatible with Cityscapes and GTA5. The dataset covers diverse driving scenarios like different times of day (e.g. daytime, nighttime, dusk, dawn) and diverse weather conditions (e.g. sunny, rainy, snowy). The majority of data comes from New York, San Francisco, Berkeley unlike Cityscapes which covers various cities in Germany and neighboring countries making spatial layouts across datasets slightly different.

In Table 6.5, we report the performances of the proposed method and the baselines on four different UDA settings namely: Cityscapes → Berkeley, SYNTHIA → Berkeley, GTA5 → Berkeley, and Berkeley → Cityscapes. The source-only model is only trained on the labeled source examples

Cityscapes → Berkeley

Method	Road	SW	Build	Wall	Fence	Pole	TL	TS	Veg.	Terrain	Sky	PR	Rider	Car	Truck	Bus	Train	Motor	Bike	mIoU
Source-only	76.25	42.15	64.99	11.7	23.95	30.75	29.4	34.98	77.23	17.49	82.67	44.87	31.0	79.37	20.34	37.63	0.04	40.33	29.34	40.76
AGP-GI	87.77	45.89	71.79	14.63	28.27	31.73	35.59	38.5	78.29	27.24	84.37	46.7	31.89	82.36	28.59	32.48	0.0	32.9	26.17	43.43
Ours	90.79	50.06	72.57	13.08	29.53	37.45	36.46	43.01	82.81	33.33	85.93	55.23	39.26	85.28	28.12	40.37	0.0	41.8	33.35	**47.29**

SYNTHIA → Berkeley

Method	Road	SW	Build	Wall*	Fence*	Pole*	TL	TS	Veg.	Sky	PR	Rider	Car	Bus	Motor	Bike	mIoU	mIoU-13
Source-only	13.42	9.85	41.97	1.17	0.0	14.39	20.46	11.06	52.29	68.79	21.9	6.0	45.23	3.46	4.7	12.13	20.43	23.94
AGP-GI	3.82	7.67	45.34	1.52	0.06	17.99	16.99	8.19	58.21	64.6	16.38	5.91	61.86	6.06	9.64	22.86	21.69	25.19
Ours	29.07	11.15	57.82	1.56	0.0	27.44	30.67	14.22	65.64	78.48	32.85	16.88	67.85	20.34	24.73	33.69	**32.02**	**37.18**

GTA5 → Berkeley

Method	Road	SW	Build	Wall	Fence	Pole	TL	TS	Veg.	Terrain	Sky	PR	Rider	Car	Truck	Bus	Train	Motor	Bike	mIoU
Source-only	51.68	16.39	41.22	2.27	27.66	30.1	34.46	19.56	56.58	26.36	64.19	48.32	20.94	70.64	14.05	31.87	0.0	27.31	26.79	32.13
AGP-GI	81.16	14.42	66.86	9.13	28.33	32.22	36.63	26.89	68.01	24.1	83.27	49.74	28.48	77.67	19.09	17.91	0.0	34.05	33.87	38.52
Ours	81.73	25.97	69.45	12.69	32.12	34.83	40.02	29.75	70.43	30.58	83.62	56.23	28.57	80.64	27.9	52.3	0.0	36.08	32.04	**43.42**

Berkeley → Cityscapes

Method	Road	SW	Build	Wall	Fence	Pole	TL	TS	Veg.	Terrain	Sky	PR	Rider	Car	Truck	Bus	Train	Motor	Bike	mIoU
Source-only	93.93	58.07	84.52	27.29	34.38	36.07	36.13	45.43	85.95	46.81	85.05	59.2	32.36	88.27	50.99	52.98	0.03	26.74	47.37	52.19
AGP-GI	93.99	60.24	84.98	29.24	33.44	34.5	35.65	45.31	86.25	45.58	87.42	62.19	36.33	88.19	45.03	54.67	0.16	30.45	48.6	52.75
Ours	93.86	58.31	85.45	38.49	31.67	36.2	32.49	42.7	86.92	47.25	87.41	63.38	37.77	90.29	68.57	61.04	6.11	35.43	51.06	**55.5**

Table 6.5: All models are trained on the labeled source training data and unlabeled target training data and performances on the validation split of the target data are reported.

minimizing the cross-entropy loss. AGP-GI (Align Global Predictions of Global Images) refers to minimizing the same adversarial loss but on the global segmentation maps similar to [VJB18].

All the methods have higher scores on Berkeley → Cityscapes compared to Cityscapes → Berkeley. The transfer from Berkeley → Cityscapes is easier compared to Cityscapes → Berkeley as Berkeley covers more diverse scenes. Furthermore, mIoU scores for GTA5 → Cityscapes (46.98 %) and SYNTHIA → Cityscapes (51.99 %) are higher than GTA5 → Berkeley (43.43 %) and SYNTHIA → Berkeley (37.18 %). This was expected due to the larger diversity of the Berkeley dataset relative to Cityscapes. However, the proposed local alignment outperforms the baselines in all tasks. The proposed method surpasses the global-alignment baselines with 11.99 %, 4.9 %, 3.86 % and 2.75 % (mIoU) for SYNTHIA → Berkeley, GTA5 → Berkeley, Cityscapes → Berkeley and Berkeley → Cityscapes respectively. The proposed method especially shines on SYNTHIA → Berkeley where the spatial class distribution shift is the largest. All other datasets have dashcam views while SYNTHIA has random camera views.

[HWY16, ZYK18, ZQY18] reported on a single one of these four tasks. But, possibly they run on an earlier version of the dataset, so to not have an unfair comparison, we did not include them in the tables.

In Fig. 6.10, we present qualitative results for Cityscapes → Berkeley. Predictions of the source-only, global alignment, and the proposed methods along with the corresponding images and the ground-truth segmentation maps are given. Black regions in the ground-truth maps belong to *other* class which are not evaluated at the test time. Significant differences from the baseline predictions are highlighted with white rectangular boxes.

For Cityscapes → Berkeley, the source-only model trained on the Cityscapes over-fit the hood of the data collecting vehicle and sometimes produces erroneous segmentation for the lower part of the Berkeley images too (see Fig. 6.10). The proposed method especially performs well on the more common classes like *car* or *building* whereas it more often fails to detect small and rare objects like *traffic signs*. These classes are challenging for compared methods too. For more details, see [CXW20b].

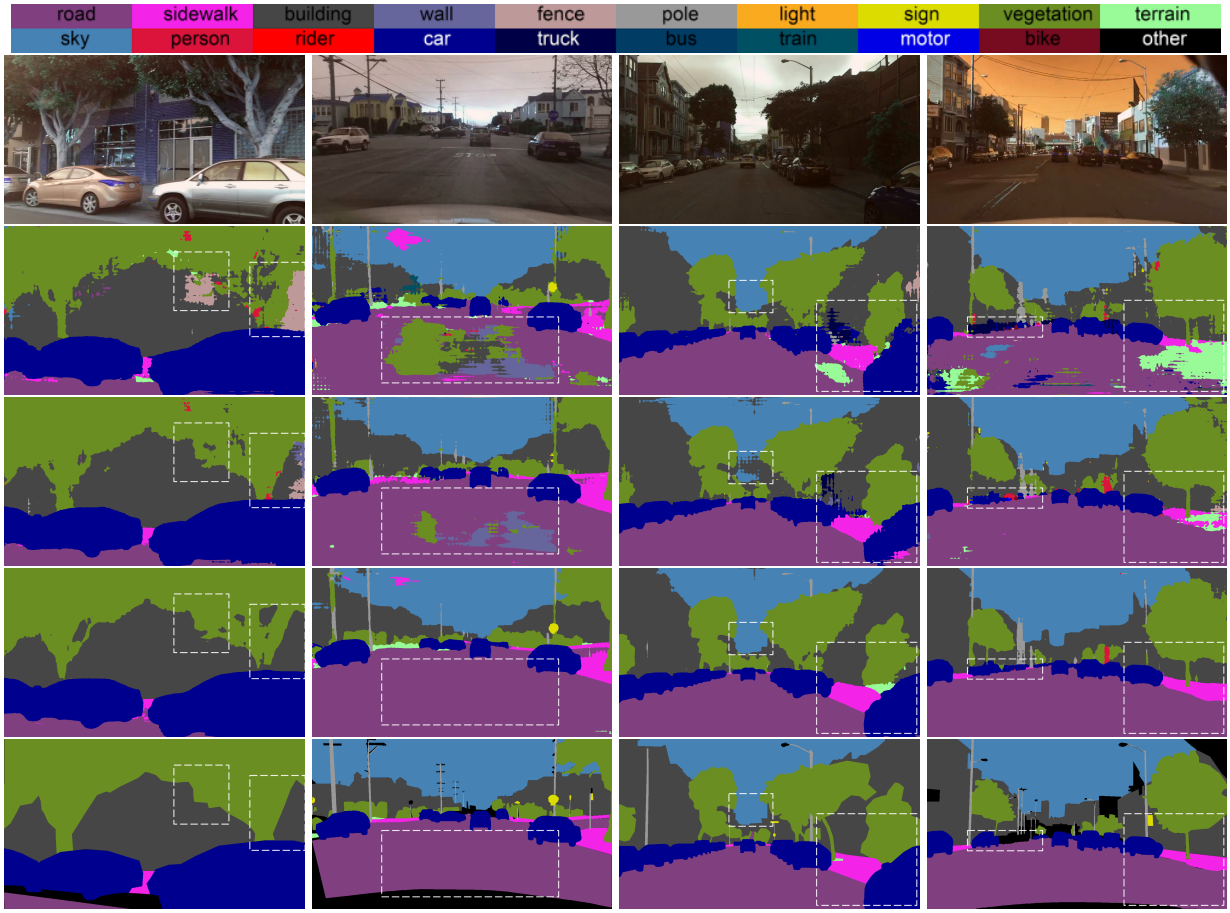


Figure 6.10: **Cityscapes → Berkeley**. From top to bottom: (1) Image, (2) source-only prediction, (3) global alignment prediction, (4) our prediction and (5) ground truth segmentation. Best viewed in color.

CHAPTER 7

Learning Topology from Synthetic Data for Unsupervised Depth Completion

7.1 Introduction

Images are "dense" in the sense of providing a color value (irradiance) at every pixel, but they contain only sparse information about the geometry of the scene, both because (i) the pre-image of a pixel is an arbitrarily large subset of the scene with no single depth value, and (ii) large portions of the image do not allow establishing unique correspondence due to occlusions or the aperture problem. We focus on the second problem (ii), aiming to use higher-level information to impute a depth value at every pixel even when correspondence is not defined (occlusion), or where it yields a continuum of possible depth values (aperture problem). Where the given images do not provide direct evidence on the geometry of the underlying scene, we have to use priors learned from *different* scenes: The fact that walls tend to be flat, surfaces piecewise smooth, objects mostly convex etc. can be evinced from data about scenes other than the one at hand.

The key challenge in this process is to determine the topology of the scene; that is, what point is "close" to which in the sense of being part of the same surface or object. Once that is determined, dense depth estimation is just a matter of piecewise smooth interpolation. Errors in the former cannot be compensated by however clever processing in the latter. So, we focus on *learning* scene topology from images, in such a way that can be used to complete the depth map where current images do not provide sufficient evidence.

In some cases, there may be independent mechanisms to associate measurements of light (irradiance at the pixels) to geometric properties of a scene, from tactile perception to controlled

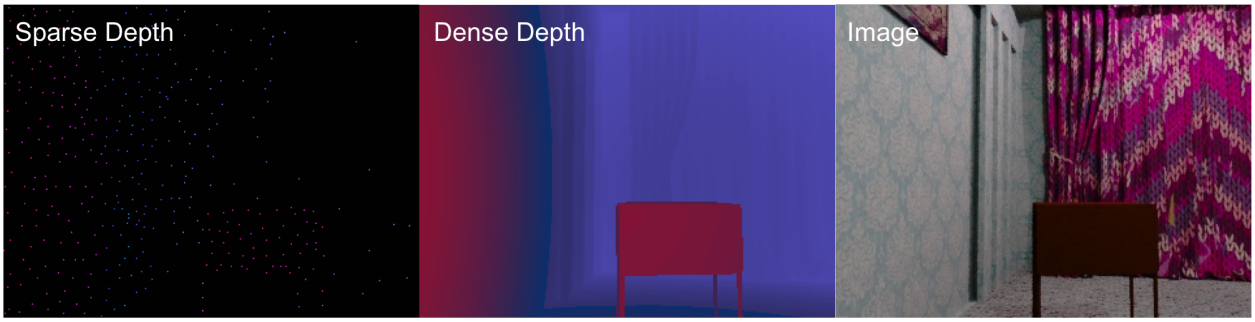


Figure 7.1: **Is it possible learn topology only from sparse points?** There exists an abundance of synthetic data (e.g. SceneNet [MHL16]) with ground-truth depth. We aim to infer the topology of objects from only sparse points (i.e. *without* images), so that we can leverage synthetic data without the need to adapt for the large domain gap between real and synthetic images.

illumination or other sensory devices. In a passive setting, absent any side information, it is difficult to obtain ground truth association, so synthetic data is a natural choice but for the covariate shift when transferring models to the real world. We bypass this challenge altogether by learning the association *not* from photometry to geometry (images to shapes), which requires a high-level understanding of the semantics of objects, but from sparse geometry (point cloud) to topology (connectivity and dense geometry), using abundant synthetic data, without having to face concerns about covariate shift and domain adaptation.

Recent approaches to depth completion that are amenable to utilize our model include [WFT20, YWS19]. [YWS19] utilizes both image and depth from synthetic data and is plagued by the domain gap when transferring to real data. [WFT20] uses a piecewise planar "scaffolding" of the scene to transfer the supervisory signal from sparse points to their neighbors. However, the piecewise planar assumption is too coarse and initial errors can have catastrophic consequences. We learn the topology of the objects only from sparse points (i.e. no RGB images). Of course, the learned topology is only a "guess," or prior, which needs to be reconciled with the images, but we posit that it is better than generic priors like smoothness or proximity, as it enables drawing from properties of the distribution of natural shapes. Accordingly, we first learn to predict an *approximate topology* from sparse points with a lightweight network that we call ScaffNet. In the second stage, our fusion network (FusionNet) leverages photometric evidence from real images to correct the prediction.

More specifically, our **contributions** include (i) repurposing Spatial Pyramid Pooling (SPP) [HZR15b] to densify the sparse depth measurements while balancing the trade-off between sparsity (pooling with small kernels) and levels of detail (large kernels); (ii) we train an SPP-augmented, light-weight network (ScaffNet) on synthetic data to learn connectivity and dense geometry from sparse inputs and demonstrate that the shapes learned can generalize well to datasets with different scene geometry (i.e. from synthetic scenes of randomly arranged household rooms in SceneNet [MHL16] to real scenes of laboratories, classrooms and gardens in VOID [WFT20]); (iii) we propose to learn additive and multiplicative residuals with FusionNet to alleviate the network from the burden of having to re-learn depth; (iv) we treat the topology learned by ScaffNet as a prior and design an adaptive loss function that selectively regularizes the predictions of FusionNet by conditioning on the fitness of the each model to data.

7.2 Related Work

Supervised depth completion methods learn a direct map from image and sparse depth to dense depth by minimizing the difference to ground-truth. [CWL18] cast depth completion as compressive sensing by learning a dictionary and [DVP18] morphological operators. Recent innovations include network operations [EFK18, HFC19] and architectures [CYL19, MCK19, USS17, YWS19] for processing sparse depth. [MCK19] handled sparse depth and images separately and fused them after a single convolution (early fusion), while [JCW18, YWS19] used two separate encoders (late fusion); [CYL19] used a 2D-3D fusion network. To propagate sparse depth through the network, [EFK18] used normalized convolutions with a binary map while [HFC19] performed joint concatenation and convolution to upsample the sparse depth. Additionally, [VND19] learned confidence maps and [XZS19, QCZ19, ZF18] exploited surface normals for guidance.

Training these methods requires per-pixel ground-truth, often unavailable or prohibitively expensive. We instead learn to infer topology from synthetic data with ground truth and abundant un-annotated real data.

Unsupervised depth completion assumes additional (stereo, temporally consecutive frames)

data available during training. Both stereo [SNM19, YWS19] and monocular [MCK19, WFT20] paradigms learn dense depth from an image and sparse depth measurements by minimizing the photometric error between the input image and its reconstruction from other views along with the difference between prediction and sparse depth input (sparse depth reconstruction). [MCK19] used Perspective-n-Point [LMF09] and RANSAC [FB81] to align consecutive frames. However, [MCK19] does not generalize well to indoor scenes with many textureless surfaces. [YWS19] also used synthetic data but require image, sparse and ground-truth depth. They do not address the domain gap between the additional dataset and the target dataset; hence, their learned prior may at times hurt performance. Unlike [YWS19], we seek to learn topology from sparse points from a synthetic dataset and do not require images, thus bypassing the domain gap. [MCK19, SNM19, YWS19] learn end-to-end without supervision, with sparse depth input. However, convolutions are ineffective in processing sparse input because most of the receptive fields are not activated in the early layers. Instead, we leverage spatial pyramid pooling [HZR15b] to increase the receptive field and "densify" the sparse input before feeding it into a topology estimation network. [WFT20] used a two-stage approach to first approximate the mesh and later fuse with image information. The "scaffolding" is prone to errors in regions that lack depth input or contain complex structures. Our approach is also two-staged, but we seek to learn topology from synthetic data. To alleviate the fusion network from having to re-learn the approximate geometry, we learn the residual from the image to refine the approximation using a fusion network. Lastly, we also regularize our predictions using the approximated topology conditioned on the fitness of the model to data.

Domain adaptation for depth prediction [NKP18, AB18, ZFG19] applied ordinary domain adaptation to single-image depth prediction. We do not attempt to reduce the domain gap between synthetic and real images, but leverage the sparse depth to learn dense topology from synthetic shapes. We use RGB images only as evidence to refine the estimates. [AAL19] leveraged synthetic data for in-painting depth. Our problem is more challenging as the sparse points cover $\approx$5% of the image space [USS17] and as few as 0.5% indoors [WFT20].

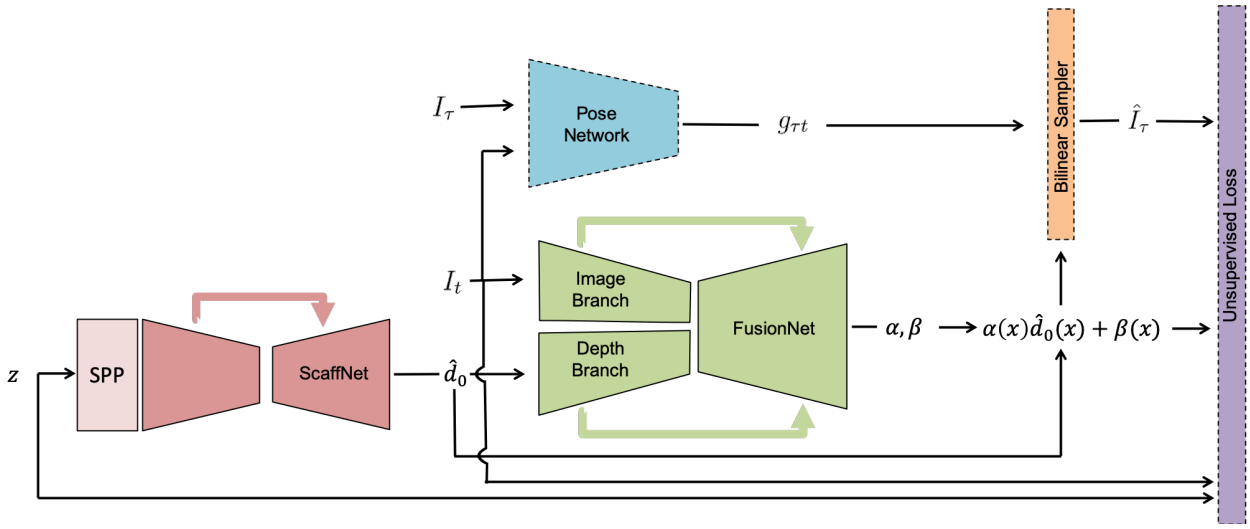


Figure 7.2: **Overview.** Sparse points (z) are first densified by SPP (see Fig. 7.3), and fed to ScaffNet (trained with synthetic data) to produce an approximate topology $\hat{d}_0$. FusionNet, trained with an unsupervised loss (Eqn. 7.3), refines $\hat{d}_0$ with α and β by fusing the $\hat{d}_0$ with the information from the image I_t to produce the final prediction $\hat{d}(x) = \alpha(x)\hat{d}_0(x) + \beta(x)$ for $x \in \Omega$. Only ScaffNet (red) and FusionNet (green) are required for inference.

7.3 Method Formulation

Our goal is to recover a 3D scene from a real RGB image $I_t : \Omega \subset \mathbb{R}^2 \mapsto \mathbb{R}^3_+$ and the associated set of sparse depth measurements $z : \Omega_z \subset \Omega \mapsto \mathbb{R}_+$, without access to ground-truth depth annotations. We follow the unsupervised monocular training paradigm [MCK19, WFT20] and assume there exists temporally adjacent frames, I_τ for $\tau \in T \doteq \{t-1, t+1\}$ denoting the previous and the next time stamp relative to I_t, available during training. Additionally, we assume there also exists a synthetic dataset, containing sparse depth $z_0 : \Omega_z \mapsto \mathbb{R}_+$ and associated ground-truth dense depth $d_0 : \Omega \mapsto \mathbb{R}_+$, available.

Our approach is separated into two stages: (i) we train a topology estimator $f_\omega(z_0)$ with a synthetic dataset $\mathcal{D}$ to approximate the scene $\hat{d}_0 := f_\omega(z_0)$. By exploiting the statistics of large synthetic datasets (where one can obtain ground-truth depth for free), our topology estimator learns to extract patterns of connectivity between sparse points to model scene structures too complex for hand-crafted priors (e.g. nearest-neighbor). However, as the topology is only informed by the sparse

points, one cannot hope to recover regions with very few (or no) points with high precision. This is where the image comes back into the picture; (ii) we refine the initial estimate $\hat{d}_0$ by incorporating information from the image belonging to the target (real) domain for which we *do not* have any ground-truth depth. We propose to learn a multiplicative scale $\alpha_\theta(I_t, z_0, \hat{d}_0)$ and an additive residual $\beta_\theta(I_t, z_0, \hat{d}_0)$ around $\hat{d}_0$ where $[\alpha_\theta(I_t, z_0, \hat{d}_0), \beta_\theta(I_t, z_0, \hat{d}_0)] = f_\theta(I_t, z_0, \hat{d}_0)$[1]. In other words, our network f_θ fuses the image information (I_t) with target sparse points (z_0) and initial estimate ($\hat{d}_0$) to produce the final dense depth $\hat{d}$ (see Fig. 7.2). Hence, the name FusionNet. By learning $\alpha(x)$ and $\beta(x)$ around $\hat{d}_0(x)$ for $x \in \Omega$, instead of directly mapping from the initial estimate and image to the final prediction, we alleviate the network from the burden of having to re-learn depth from scratch.

7.3.1 Learning Topology using Synthetic Data

Can we learn to infer the dense topology of the scene given *only* sparse points? This is an ill-posed problem as there exists infinitely many possible scenes compatible with the missing depth measurements. We propose to learn a topology estimator (ScaffNet) to produce dense depth from sparse depth measurements *without* the use of an RGB image. To accomplish this, we leverage synthetic datasets with accurate dense depth to capture the patterns of complex geometry, present in everyday objects, that hand-crafted priors (e.g. nearest neighbor, piece-wise smoothness [WFT20]) cannot.

For this, we train a small encoder-decoder network $f_\omega(\cdot)$, comprised of only ≈1.4M parameters, by minimizing the normalized $L1$ difference between the estimate $\hat{d}_0$ and the ground-truth dense depth d for each sample:

$$l_0 = \frac{1}{|\Omega|} \sum_{x \in \Omega} \left| \frac{f_\omega(z_0(x)) - d(x)}{d(x)} \right| \tag{7.1}$$

to learn a mapping from sparse points z_0 to dense topology $\hat{d}_0 = f_\omega(z_0(x))$. We note that it is impossible for $f_\omega(\cdot)$ to recover regions of the scene that lack sparse depth measurements. Hence, $\hat{d}_0$ serves as an initial estimate of the scene. Therefore, we propose to refine $\hat{d}_0$ using information from the image (see Sec. 7.3.2).

[1]More formally, α and β should be written as a function of position $x \in \Omega$, $\alpha_\theta(I_t(x), z_0(x), \hat{d}_0(x))$. For simplicity, we will refer to them as $\alpha(x)$ and $\beta(x)$ without the parameters and inputs.

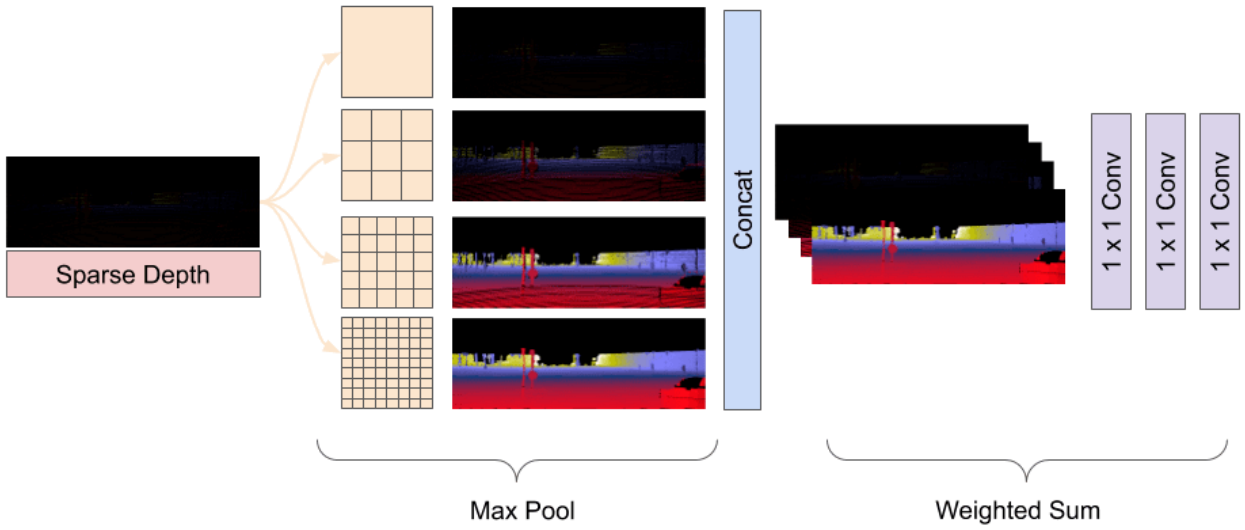


Figure 7.3: **Spatial Pyramid Pooling (SPP) for depth completion**. The SPP module balances details versus density for sparse depth inputs max-pooled at different scales. When pooled with small kernels, the details of the sparse inputs are preserved, but the subsequent layers will produce very few non-zero activations. When pooled with large kernels, inputs are densified, but details are lost. By weighting the output of the pooling with several stacked 1×1 convolutions, the network learns to optimize for this trade-off.

Spatial Pyramid Pooling (SPP). Since most of the input values for sparse depth measurements are zeroes, the activations of early convolutional layers tend to be zeroes as well. To process the sparse input, we augment our network ($f_\omega(\cdot)$) with an SPP module (see Fig. 7.3) where each layer in SPP is a max-pool of different kernel sizes. For max-pool layers with large kernel sizes, the sparse input is densified, leading to more neurons being activated in the subsequent layers. However, the finer details (e.g. small or thin objects close to the camera) are corrupted. For max-pool layers with small kernel sizes, details of the sparse input are preserved, but as a result, fewer neurons are activated. Therefore, we weight the output of the max-pool layers, with different kernel-sizes, using several stacked 1×1 convolutional layers such that the network can optimize for this trade-off. We note that SPP also increases the receptive field of the network. The output of SPP is fed into our encoder-decoder network and the weights belonging to the 1×1 convolutions are jointly optimized. We demonstrate the effectiveness of our SPP module in an ablation study in Table 7.4 of Sec. 7.6. We will also discuss differences between our variant of SPP and those employed in classification

[CC18] and stereo [HZR15b] (both operating on *already dense* inputs).

7.3.2 Learning to Refine (Bringing the Image Back)

ScaffNet, our topology estimator, $f_\omega(\cdot)$ learns to predict the coarse scene structure $\hat{d}_0$ from sparse points z_0 where available. However, regions where sparse points are unavailable (due to scan pattern or range of sensor), predictions of $f_\omega(\cdot)$ can be misleading. This is where we bring the image back. We want to learn a function $f_\theta(I_t, z, \hat{d}_0)$ that produces the scale $\alpha(x)$ and the residual $\beta(x)$ for every element in the image domain, $x \in \Omega$. $\alpha(x)$ and $\beta(x)$ refine each $\hat{d}_0(x)$ based on the image to produce the final depth prediction:

$$\hat{d}(x) = \alpha(x)\hat{d}_0(x) + \beta(x) \tag{7.2}$$

where α is encouraged to be close to 1 and β to be close to 0 (see Eqn. 7.9). To learn $\alpha(x)$ and $\beta(x)$, we leverage the geometric relations between the image I_t and its temporally adjacent frames I_τ, which provide a set of constraints on depth values, up to an unknown scale. These constraints are realized by (i) reconstructing I_t with I_τ to construct a photometric consistency loss (Eqn. 7.5). To ground the depth values to metric scale, we (ii) reconstruct the sparse depth measurements z (where available) with $\hat{d}$ (Eqn. 7.6). As depth completion is ill-posed, we assume (iii) generic (not informed by data) local smoothness and connectivity (Eqn. 7.7) on the surfaces populating the scene. Lastly, to leverage the prior learned from synthetic data (side information), we propose to (iv) *selectively* regularize $\hat{d}$ using $\hat{d}_0$ conditioned on the fitness of model to the data (Eqn. 7.9). Our unsupervised objective function is the linear combination of four terms:

$$\mathcal{L} = w_{ph}l_{ph} + w_{sz}l_{sz} + w_{sm}l_{sm} + w_{pz}l_{pz} \tag{7.3}$$

where l_{ph} denotes photometric consistency, l_{sz} sparse depth consistency, l_{sm} local smoothness and l_{pz} the topology prior. Each term is weighted by their associated w (see Sec. 7.5).

Photometric Consistency. We leverage epipolar constraints as a supervisory signal by reconstructing I_t from I_τ for $\tau \in T \doteq \{t-1, t+1\}$ to yield:

$$\hat{I}_\tau(x, \hat{d}) = I_\tau\big(\pi g_{\tau t} K^{-1} \bar{x} \hat{d}(x)\big) \tag{7.4}$$

where $\bar{x} = [x^\top \ 1]^\top$ are the homogeneous coordinates of $x \in \Omega$, $g_{\tau t} \in SE(3)$ is the relative pose of the camera from time t to τ, K denotes the camera intrinsics, and π refers to the perspective projection including the camera intrinsic matrix K.

To realize the geometric constraints as a loss, we minimize the average photometric reprojection error measured by a combination of $L1$ penalty and `SSIM` [WBS04], a perceptual metric, to penalize dissimilarities:

$$l_{ph} = \frac{1}{|\Omega|} \sum_{\tau \in T} \sum_{x \in \Omega} w_{co} |\hat{I}_\tau(x, \hat{d}) - I_t(x)| + w_{st}\big(1 - \texttt{SSIM}(\hat{I}_\tau(x, \hat{d}), I_t(x))\big) \tag{7.5}$$

w_{co} and w_{st} are weights for each term and will be discussed in Sec. 7.5. Note that we also jointly learn pose $g_{\tau t}$ as a by product of minimizing Eqn. 7.5.

Sparse Depth Consistency. Photometric reconstruction recovers the scene structure up to a scale; to ground the scene to *metric* scale, we minimize the $L1$ difference between our predictions $\hat{d}$ and the sparse depth measurements over the sparse depth domain (Ω_z):

$$l_{sz} = \frac{1}{|\Omega_z|} \sum_{x \in \Omega_z} |\hat{d}(x) - z(x)|. \tag{7.6}$$

Local Smoothness. While sparse depth measurements removes ambiguity, there still exists infinitely many scenes compatible with the image in regions *not* in the sparse depth domain ($\Omega \backslash \Omega_z$). Hence, we assume local smoothness and connectivity over $\hat{d}$ with an $L1$ penalty on the gradients in the $x-$ (∂_X) and $y-$ (∂_X) directions. To allow discontinuities along object boundaries, we weight each term using its respective image gradients, $\lambda_X = e^{-|\partial_X I_t(x)|}$ and $\lambda_Y = e^{-|\partial_Y I_t(x)|}$, where strong gradients produce lower weight:

$$l_{sm} = \frac{1}{|\Omega|} \sum_{x \in \Omega} \lambda_X(x) |\partial_X \hat{d}(x)| + \lambda_Y(x) |\partial_Y \hat{d}(x)|. \tag{7.7}$$

Topology Prior. Learning depth from motion is essentially a correspondence problem (2D search space over the image domain). Rather than exploring the entire solution or hypothesis space, we look to leverage the initial topology $\hat{d}_0$ predicted by $f_\omega(\cdot)$ as a prior to limit the scope or to bias our predictions towards the set of hypotheses compatible with what we have learned from a synthetic dataset, $\mathcal{D}$. However, the quality of $\hat{d}_0$ degrades in regions with very few or no sparse

Metric	Definition
MAE	$\frac{1}{\lvert\Omega\rvert}\sum_{x\in\Omega}\lvert\hat{d}(x)-d_{gt}(x)\rvert$
RMSE	$\left(\frac{1}{\lvert\Omega\rvert}\sum_{x\in\Omega}\lvert\hat{d}(x)-d_{gt}(x)\rvert^2\right)^{1/2}$
iMAE	$\frac{1}{\lvert\Omega\rvert}\sum_{x\in\Omega}\lvert 1/\hat{d}(x)-1/d_{gt}(x)\rvert$
iRMSE	$\left(\frac{1}{\lvert\Omega\rvert}\sum_{x\in\Omega}\lvert 1/\hat{d}(x)-1/d_{gt}(x)\rvert^2\right)^{1/2}$

Table 7.1: **Error metrics.** Evaluation metrics for KITTI and VOID. d_{gt} denotes the ground truth.

depth measurements. Hence, one should *selectively* regularize based on the compatibility between the prior $\hat{d}_0$ and the image I_t. To elaborate, we should bias our predictions $\hat{d}$ towards the prior $\hat{d}_0$ only if $\hat{d}_0$ is more compatible with the image than $\hat{d}$. Following this intuition, we present a simple, yet effective, per-pixel regularizer conditioned on the fitness of $\hat{d}_0$ and $\hat{d}$ to the image as measured by the discrepancies between their respective reconstructions (Eqn. 7.4), $\hat{I}_\tau(x,\hat{d})$ and $\hat{I}_\tau(x,\hat{d}_0)$, and the image $I_t(x)$.

To optimize for this trade-off, we construct $W(x)$, an indicator function that is 1 if the photometric discrepancy $\delta = I_t(x) - \hat{I}_\tau(x,\hat{d})$ is greater than that of $\delta_0 = I_t(x) - \hat{I}_\tau(x,\hat{d}_0)$ and 0 otherwise, for every $x \in \Omega$,

$$W(x) = \begin{cases} 1 & \text{if } \delta > \delta_0 \\ 0 & \text{otherwise.} \end{cases} \tag{7.8}$$

We then use $W(x)$ as a pixel-wise mask to selectively impose the topology prior $\hat{d}_0(x)$ on the predictions $\hat{d}(x)$:

$$l_{tp} = \frac{1}{\sum_{x\in\Omega} W(x)} \sum_{x\in\Omega} W(x)\lvert\hat{d}(x)-\hat{d}_0(x)\rvert. \tag{7.9}$$

Note that Eqn. 7.9 is flexible and encourages $\alpha(x)$ to be close to 1 and $\beta(x)$ to 0 *only* at locations where the initial topology prediction $\hat{d}_0$ is already a good fit for the image. The network is free to modify $\hat{d}_0$ where the topology is incorrect.

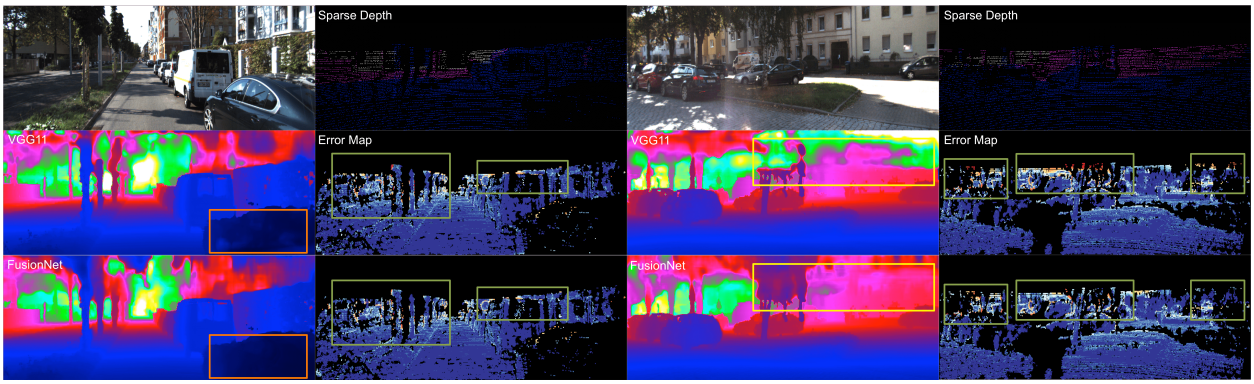


Figure 7.4: **KITTI depth completion testing set.** Visualizations are taken directly from KITTI online benchmark. Our method better captures smooth surfaces (car, highlighted in orange) and complex objects (trees, highlighted in yellow). We also perform better in recovering buildings (also in yellow). The overall improvement is apparent in the error map. Many red regions (high error) in [WFT20] are marked blue (low error) in our results.

7.4 Datasets

SceneNet [MHL16] consists of 5 million samples of size 320×240 rendered from indoor trajectories of randomly arranged rooms. Out of 17 splits provided, we only choose one due to computational restrictions. In a single split, there are 1000 subsequences where each subsequence includes 300 images of the same scene, recorded over a trajectory. To train our ScaffNet, which produces dense topology from sparse inputs, we constructed sparse points for each ground-truth dense depth sample by running Harris corner detector [HS88] over the RGB images. The resulting points are subsampled using k-means to produce the final 375 corners which correspond to 0.49% of all the pixels. The SceneNet sequence is used to train ScaffNet for indoor target dataset (VOID [WFT20]). Despite SceneNet (indoor, household rooms) having different scene structures from VOID (indoor and outdoor, laboratories and gardens), ScaffNet trained on SceneNet generalizes to VOID.

Virtual KITTI (VKITTI) [GWC16] consists of 35 photo-realistic synthetic videos (5 cloned from the original KITTI, coupled with 7 variations of each) for a total of $\approx$17,000 frames of a resolution of around 1242×375. Variations are weather conditions (e.g. rainy), lighting conditions, and camera angles. [GWC16] use the commercial computer graphics engine Unity to create virtual worlds that similar to scenes in KITTI. However, there is still a large domain gap between RGB

images of both domains. To bypass the domain gap in photometric variations, we only use the dense depth maps of VKITTI. To acquire the sparse points, we imitate the sparse depth measurement of KITTI (produced by a lidar) so that the marginal distributions of sparse points are close across domains. We use VKITTI to train the topology estimator for the outdoor target dataset (KITTI [USS17]).

KITTI. We evaluate our approach on the KITTI depth completion benchmark [USS17]. The dataset provides ≈80,000 raw image frames and associated sparse depth maps. The sparse depth maps are the raw output from the Velodyne lidar sensor, each with a density of ≈5%. The ground-truth depth map is created by accumulating the neighboring 11 raw lidar scans, with dense depth corresponding to the bottom 30% of the images. We use the official 1,000 samples for validation and test on 1,000 designated samples, which we submit to the KITTI online benchmark for evaluation.

VOID provides synchronized RGB image frames and sparse depth maps of $\approx 640 \times 480$ resolution of indoor (laboratories, classrooms) and outdoor (gardens) scenes. ≈ 1500 sparse depth points (covering $\approx 0.5\%$ of the image) are the set of features tracked by XIVO [FWS19], a VIO system. The ground-truth depth maps are dense and are acquired by the active stereo. The entire dataset contains 56 sequences with challenging motion. Of the 56 sequences, 48 sequences ($\approx 40,000$) are designated for training and 8 for testing. The testing set contains 800 frames. We follow the evaluation protocol of [WFT20] and cap the depths between 0.2 and 5 meters.

7.5 Implementation Details

Training ScaffNet on VKITTI [GWC16] took ≈12 hours (30 epochs) while training FusionNet on KITTI [USS17] requires ≈27 hours (30 epochs) on an Nvidia GTX 1080Ti. Training ScaffNet on SceneNet [MHL16] requires ≈6 hours (10 epochs). FusionNet also requires ≈6 hours (10 epochs) for VOID [WFT20]. End to end inference takes ≈20 ms per image. We used Adam [KB14] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ to optimize our network with a base learning rates of 1.5×10^{-4} for VKITTI and KITTI and 5×10^{-5} for SceneNet and VOID. We decrease the learning rate by half after 18 epochs for VKITTI and KITTI and 6 epochs for SceneNet and VOID, and again after 24

Method	# Param	MAE	RMSE	iMAE	iRMSE
Scaffolding [WFT20]	0	443.57	1990.68	1.72	6.43
Our ScaffNet	≈1.4M	318.41	1425.53	1.39	5.01
Ma [MCK19]	≈27.8M	358.92	1384.85	1.60	4.32
Yang [YWS19]	≈18.8M	347.17	1310.03	n/a	n/a
VGG8 [WFT20]	≈6.4M	308.81	1230.85	1.29	3.84
VGG11 [WFT20]	≈9.7M	305.06	1239.06	1.21	3.71
Our FusionNet	≈7.8M	**286.35**	**1182.81**	**1.18**	**3.55**

Table 7.2: **Results on the KITTI validation set.** Results of [MCK19, YWS19, WFT20] are directly taken from their papers. Our method (FusionNet) performs the best across all metrics for the KITTI validation set while using fewer parameters than the state of the art (VGG11 [WFT20]). Our topology estimator (ScaffNet) alone outperforms [MCK19, YWS19] on MAE and [MCK19] on iMAE ([YWS19] did not report their iMAE on the validation set). We note that ScaffNet does not use RGB images, is trained on synthetic data, and evaluated on real data.

epochs and 8 epochs, respectively. We train our network with a batch size of 8 using 768×320 crops for VKITTI and KITTI and 640×480 for SceneNet and VOID. To replicate our results on KITTI, we set the weights for each term in our loss function as: $w_{ph} = 1.00$, $w_{co} = 0.20$, $w_{st} = 0.40$, $w_{sz} = 0.10$, $w_{sm} = 0.01$ and $w_{tp} = 0.10$. For VOID, we increased w_{sz} to 1.00 and w_{sm} to 0.40. We only apply l_{tp} after 60K steps (for 271K total steps) for KITTI and 20K steps (for 51K total steps) for VOID to allow pose to stabilize. We perform horizontal shifts on as data augmentation on KITTI and VKITTI. We do not use any data augmentation for SceneNet and VOID.

7.5.1 Network Architecture

We present our network architectures for our topology estimator, and our depth-RGB image fusion network, ScaffNet and FusionNet, respectively. We design our network structures carefully to

Method	# Param	MAE	RMSE	iMAE	iRMSE
Schneider [SSP16]	n/a	605.47	2312.57	2.05	7.38
Ma [MCK19]	≈27.8M	350.32	1299.85	1.57	4.07
Yang [YWS19]	≈18.8M	343.46	1263.19	1.32	3.58
Shivakumar [SNM19]	n/a	429.93	1206.66	1.79	3.62
VGG8 [WFT20]	≈6.4M	304.57	1164.58	1.28	3.66
VGG11 [WFT20]	≈9.7M	299.41	1169.97	1.20	3.56
Our FusionNet	≈7.8M	**280.76**	**1121.93**	**1.15**	**3.30**

Table 7.3: **Results on the KITTI depth completion benchmark.** Results are directly taken from the KITTI online benchmark. Key comparisons: (i) [YWS19] uses both synthetic images and depth maps to train their model and is plagued by the domain gap between real and synthetic images. We bypass the need to adapt to the covariate shift by learning topology from only sparse depth. (ii) [WFT20] uses hand-crafted scaffolding and learns depth from scratch. Instead, we propose to exploit the distribution of natural shapes from synthetic datasets and learn multiplicative and additive residuals to alleviate the network from needed to re-learn depth. Our approach beats all competing methods across all metrics and achieves the state of the art on the unsupervised depth completion task.

make them light-weight so that they can be employed on standard embedded systems for real-time applications.

Our ScaffNet is an SPP module followed by an encoder and decoder, which all together consists of only ≈1.4M parameters. Given sparse depth measurements, our ScaffNet outputs an initial estimate of the topology, $\hat{d}_0$, which is refined by our FusionNet. Our FusionNet follows the late fusion paradigm with an image branch and a depth branch. The latent representation of both branches are concatenated and fed to decoder. The decoder produces multiplicative scales $\alpha(x)$ and additive residuals $\beta(x)$ to construct our final depth prediction $\hat{d}(x) = \alpha(x)\hat{d}_0(x) + \beta(x)$ for $x \in \Omega$.

Figure 7.5: **Qualitative ablation on the effect of SPP**. We show the benefits of leveraging SPP to increase receptive field and densify the sparse input. ScaffNet with SPP consistently outperforms the model without SPP in most regions of the error map. Using SPP, ScaffNet captures more details about the scene. For example, in the top panel, ScaffNet with SPP recovers the person while the model without SPP misses a portion of the person. In the bottom panel, we see that SPP helps retain more details about complex objects (e.g. plants and rails). Regions are highlighted in yellow for comparison.

Our FusionNet is extremely light-weight and only consists of ≈6.4M parameters. Together, our model has a total of ≈7.8M parameters, much fewer than the competing methods ≈9.7M [WFT20], ≈18.8M [YWS19], and ≈27.8M [MCK19] while outperforming all of them across all metrics.

7.5.1.1 ScaffNet

ScaffNet is an encoder-decoder network augmented with a Spatial Pyramid Pooling module. Our SPP module is comprised of several max pool layers followed by three 1×1 convolutional layers to weight the trade-off between small (fine details, but sparse) and large (coarse, but dense) kernel sizes. The output of our SPP module is fed into an encoder with five layers. The first layer uses a 5×5 kernel with 32 filters. The remaining four layers use 3×3 kernels with 64, 96, 128, and 196 filters, respectively. The latent representation is then fed into a decoder with skip connections. The decoder is separated into five modules, each corresponds to a resolution level. For each module, we perform de-convolution using 3×3 kernels with 128, 96, 64, 64, and 32 filters, respectively.

Method	MAE	RMSE	iMAE	iRMSE
ScaffNet w/o SPP	409.93	1776.42	1.72	6.40
ScaffNet w/ SPP	318.41	1425.53	1.39	5.01
FusionNet (β only)	301.08	1297.02	1.30	4.43
FusionNet (α only)	299.90	1304.77	1.32	4.34
FusionNet (direct map)	292.74	1213.68	1.20	3.59
FusionNet (no l_{tp})	297.39	1204.88	1.19	3.58
FusionNet	**286.35**	**1182.81**	**1.18**	**3.55**

Table 7.4: **Ablation study on the KITTI validation set.** Rows 1 and 2: we show a comparison between our SPP module (w/ SPP) and conventional convolutions (w/o SPP) for processing the sparse inputs. SPP improves performance across all metrics by large margins. Rows 3 to 5 and 7: we justify learning α and β for refining the initial estimate $\hat{d}_0$. α and β alone are unable to capture the scene and performs worse than direct mapping. When combined together α and β surpasses direct mapping on all metrics. Rows 6 and 7: the topology prior (Eqn. 7.9) allows us to leverage what we have learned from synthetic data in regions that are compatible with the image, providing a consistent performance boost.

The result is concatenated with the output of the corresponding resolution (skip connection) from the encoder. The output of which is convolved with another 3×3 and passed to the next decoder module.

7.5.1.2 FusionNet

Our FusionNet consists of two encoders (branches), one for processing the image and the other for depth. Both encoders contain five convolutional layers with the first layer using a 5×5 kernel. The remaining four layers for each branch use 3×3 kernels. For the image encoder, the convolutional layers consist of 48, 96, 192, 384, and 384 filters. For the depth encoder, the

layers consist of 16, 32, 64, 128, and 128 filters. The output latent representations of the encoders are concatenated together and fed into the decoder. The FusionNet decoder, unlike the ScaffNet decoder, contains four modules, where each of the modules is comprised of 3×3 de-convolution, followed by concatenation with skip connections, and a 3×3 convolution. The skip connections are the concatenation of the feature maps from the corresponding image and depth encoders. The de-convolution and convolution layers of each module use 256, 128, 64, 64 filters, respectively. The low-resolution output is produced by a 3×3 convolution with 2 filters, for α and β. The last layer is a nearest-neighbor upsampling layer to produce the full resolution α and β.

7.6 Experiments and Results

7.6.1 KITTI Depth Completion Benchmark

We evaluate our method on the *unsupervised* KITTI depth completion benchmark (outdoor driving scenarios). In Table 7.2, we compare ScaffNet, which uses *only* sparse depth and trained in synthetic data, to recent unsupervised methods [MCK19, YWS19, WFT20] that learn dense depth from *both* sparse depth and RGB images of the target (real) domain. We note that ScaffNet has never been trained on real data, yet, outperforms [MCK19, YWS19] by as much as 11.29% and 8.28%, respectively, in the MAE metric. Our findings verify that it is indeed possible to produce a decent topology estimate from only sparse inputs. More importantly, this shows that despite being trained on synthetic data, ScaffNet can bypass the domain gap and generalize to real scenarios. We note that the topology produced by ScaffNet is only an initial approximation. By fusing image information with the approximation, our FusionNet outperforms the top unsupervised depth completion methods on every metric.

Our approach also outperforms the state of the art, VGG11 [WFT20], across every metric while having a 19.6% parameter reduction on the KITTI depth completion benchmark testing set (Table 7.3. We attribute part of our success to ScaffNet, whose output serves as both an initialization as well as a prior for FusionNet. On its own, ScaffNet outperforms scaffolding [WFT20] by as much as 28.22% on the MAE metric (see Table 7.2). We note key comparisons: in contrast to [WFT20],

we learn α (multiplicative scale) and β (additive residual) around the initial approximation and hence alleviates FusionNet from having to re-learn depth, allowing us to reduce parameters while achieving better performance. Also, unlike [YWS19], our prior does not require an image, which is subject to the domain gap between real and synthetic data.

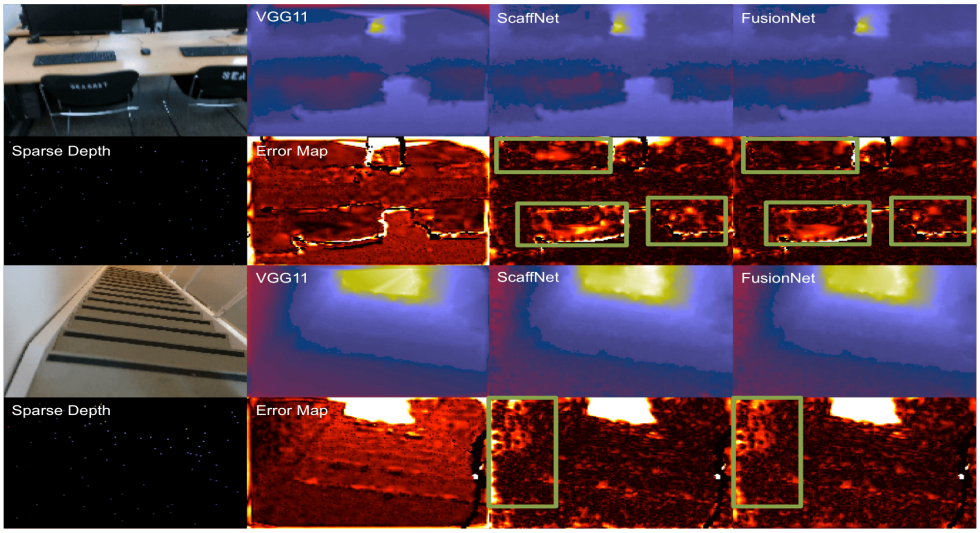


Figure 7.6: **VOID depth completion testing set.** Although ScaffNet only uses sparse points and is trained on synthetic data, it can still capture the topology of the scene. FusionNet approximated topology and refines the incorrect predictions (regions highlighted in green) with image information. Note: in the bottom two rows, the initial estimate is mostly correct; hence FusionNet does not modify those regions and instead leverages it as a prior (Eqn. 7.9).

To evaluate the contribution of SPP, we provide ablation study in Table 7.4. As seen in row 1 and 2, by augmenting our topology estimator with SPP, we gain a 22.32% error reduction on the MAE metric. This shows that the increase in the receptive field and weighted multi-scale densification of the sparse depth inputs are important elements in tackling the sparse depth completion problem. We illustrate the benefits of our SPP module in Fig. 7.5. The model with SPP consistently outperforms the one without and is able to retain more details about the scene with significantly lower errors.

To understand the architectural choice, we evaluated FusionNet using different output in rows 3 to 5 and 7 of Table 7.4. When learning α (scale) and β (residual) individually, FusionNet performs worse than directly mapping (learning from scratch) the initial approximation $\hat{d}_0$ and the image to

Method	MAE	RMSE	iMAE	iRMSE
Ma [MCK19]	198.76	260.67	88.07	114.96
Yang [YWS19]	151.86	222.36	74.59	112.36
VGG8 [WFT20]	98.45	169.17	57.22	115.33
VGG11 [WFT20]	85.05	169.79	48.92	104.02
Our ScaffNet	70.16	156.99	42.78	91.48
VGG11 + SLAM [WFT20]	73.14	146.40	42.55	93.16
Our FusionNet	**59.53**	**119.14**	**35.72**	**68.36**

Table 7.5: **Quantitative results on the VOID benchmark.** Results of [MCK19, YWS19, WFT20] are taken from [WFT20]. Because there are many textureless regions in indoor scenes, locally, the image does not inform the scene structure. Hence, a prior informed by data is even more important. Our method outperforms all competing methods on the VOID depth completion benchmark to achieve the state of the art.

dense depth. However, when combined together, α and β achieves the state of the art. We note the behavior of direct mapping during the early stages of training is learning to copy depth. This is because the network weights are initialized with Gaussian noises and thus the corresponding pixel in the input prediction has the highest weight on the output prediction [LLU16]. By learning α and β around $\hat{d}_0$, we relieve the network of this onerous task. In regards to why α and β alone cannot achieve the same performance, we believe it is related to their distribution. We observed that, when used separately, both α and β have long-tail distributions (to refine regions with no sparse depth) and large local disparities (e.g. object boundaries); whereas when used together, the range of their values are much smaller. We hypothesize that the sharp local changes make it harder to learn just α or β.

Finally, we assess the effect of our topology prior in rows 6 and 7. Row 6 (no l_{tp}) omits the topology prior from the objective function (Eqn. 7.3). We see an overall improvement by leveraging

what we have learned from synthetic datasets. This also shows that the topology obtained from synthetic data can generalize to real datasets. The proposed method (FusionNet) surpasses all baselines and variants to justify each architectural and loss component choice.

7.6.2 VOID Depth Completion Benchmark

We provide quantitative (Table 7.5) and qualitative (Fig. 7.6) evaluations of our approach on the indoor and outdoor VOID [WFT20] depth completion benchmark. Indoor scenes are composed of many textureless surfaces. Locally, they do not give any information about the scene structure; hence, learning a topology prior that is informed by data is even more important. The challenge here is that indoor scenes contain objects with more shape variations (from simple flat surfaces like walls to complex ones like chairs). Moreover, the density of the sparse points (tracked by VIO and SLAM systems) is $\approx$0.5% in contrast to outdoor driving scenarios (lidar), where density is $\approx$5% concentrated on the lower half of the image space. To demonstrate that ScaffNet can generalize even when the scene structures in the synthetic dataset are different from that of the real dataset, we trained our ScaffNet on SceneNet [MHL16] and evaluated on VOID – SceneNet consists of randomly arranged synthetic indoor household rooms while VOID contains real indoor and outdoor scenes of laboratories, classrooms and gardens. Despite having far fewer points, more complex geometry, and being trained on a synthetic dataset with different scene distribution, ScaffNet is still able to predict reasonable topology – in fact, it beats all of the competing methods that are trained on real data using both image and sparse depth. This is because ScaffNet does not learn the scenes themselves, but the shapes of natural objects populating them, which allows the network to exploit the abundance of synthetic data to learn patterns from sparse points to infer dense topology. Our FusionNet further incorporates the image information into the topology estimate to achieve the state of the art, beating VGG11 [WFT20] by as much as $\approx$30% on MAE. We also outperform their hybrid model VGG11+SLAM (which uses accurate SLAM pose instead of learning it from scratch) by $\approx$18.6% on MAE.

7.6.3 Comparison to Supervised Methods

We also compare our approach against the top-performing *supervised* methods in Table 7.6. Although we are the best performing method in the unsupervised setting, we note that the benchmark is still dominated by supervised methods. However, our approach shows promise as we surpass some supervised methods: [CWL18] across all metrics, [DVP18] on MAE, iMAE, and iRMSE metrics and [MCK19] on iMAE. We hope that our approach will lay the foundation for the push to close the gap between supervised and unsupervised learning frameworks for the depth completion task.

7.7 Ablation Study on the Effects of Spatial Pyramid Pooling

We begin this section by validating the claim regarding the sparsity in the feature maps produced by early convolutional layers without Spatial Pyramid Pooling. To demonstrate the effect of Spatial Pyramid Pooling, we show, in Fig. 7.7, a visualization of this phenomenon and how augmenting the network with a Spatial Pyramid Pooling module can produce much denser feature maps. In Sec. 7.7.2, we show (i) additional ablation studies on the performance benefits of Spatial Pyramid Pooling, (ii) discuss how Spatial Pyramid Pooling balances the trade-off between the level of density in the extracted features and the level of detail that they capture in Sec. 7.7.3, and (iii) how our variant of Spatial Pyramid Pooling is different from previous works.

7.7.1 Regarding its effect on feature maps

The challenge of sparse depth completion is precisely the sparsity. Because of the numerous "holes" (or zeroes) in the sparse input, the activations of the earlier convolutions layers tend to be zero as well. Therefore, much of the earlier layers are dedicated to "densifying" the feature maps. This is illustrated in the right column of Fig. 7.7. Our goal is to enable better learning by generating a denser representation. To this end, we proposed augmenting the topology estimator network with our Spatial Pyramid Pooling module, where multiple max pools of different scales are performed on the sparse input for an increase in the receptive field and input densification. In effect, the features

generated from Spatial Pyramid Pooling become much denser than those from simple convolutions. This, in turn, allows the activations of the subsequent convolutional layers in the encoder to be dense as well. We illustrate this phenomenon in Fig. 7.7 where we show that the feature maps of ScaffNet *without* the use of Spatial Pyramid Pooling is still sparse and resembles the input sparse depth; whereas, the feature maps of ScaffNet *with* Spatial Pyramid Pooling is much denser in comparison. By providing the encoder with dense outputs from Spatial Pyramid Pooling, we, not only, relieve the encoder from the onerous task of propagating the sparse signal spatially, but also provide a larger receptive field and context to the subsequent layers.

7.7.2 Regarding its effect on performance

To understand the benefits of using Spatial Pyramid Pooling, we show an ablation study on its impact on ScaffNet (when augmented with this module) and also its effect on later stages of inference (FusionNet) in both outdoors (KITTI, Table 7.7) and indoors (VOID, Table 7.8) scenarios. Overall, the performance benefits are apparent in Table 7.7 and 7.8, where both ScaffNet augmented with Spatial Pyramid Pooling and its associated FusionNet (marked with w/ SPP) outperform their variants without the module (marked with w/o SPP) by large margins across all metrics. Because ScaffNet consistently performs worse without Spatial Pyramid Pooling, FusionNet is given a lower quality initial topology; therefore, FusionNet like-wise performs worse – justifying the use of Spatial Pyramid Pooling in ScaffNet. This ablation study also shows that errors are propagated downstream. However, while errors do get introduced to FusionNet when using a ScaffNet without Spatial Pyramid Pooling, this is not a point of failure as FusionNet (w/o SPP) is still able to amend the mistakes and improve the reconstruction. We also note that our ScaffNet without Spatial Pyramid Pooling, in fact, still outperforms [MCK19, YWS19] in Table 7.8. We note it is possible to use ScaffNet to obtain dense topology from sparse depth to benefit existing supervised and unsupervised depth completion methods including, but not limited to [CYL19, MCK19, SNM19, QCZ19, USS17, WFT20, XZS19, YWS19, ZF18].

Figure 7.7: **A comparison of feature maps produced by ScaffNet with and without Spatial Pyramid Pooling (SPP)**. Hidden layer outputs (feature maps) after 5×5 convolutional layers with and without SPP. The feature maps produced by ScaffNet without SPP are sparse and resembles the input sparse depth, which illustrates our claim – neurons are not activated because of the missing sparse depth measurements. In contrast, when using SPP, ScaffNet produces a much denser representation.

7.7.3 Regarding the trade-off between detail and density

In Sec. 7.8, we examine the impact of various levels of input sparse depth density for both ScaffNet and FusionNet (Table 7.9 and Fig. 7.8).

We note that for the purpose of densification, one may just use a single max pool layer with a large kernel size. However, such a max pool layer would decimate the details of the sparse input. One may also choose to leverage heuristics such as nearest neighbor and local smoothness [WFT20] to interpolate depth between sparse points. However, when the nearest neighboring points are far away (both in 3D world or 2D image space), the plane interpolated between the points may contain incorrect values as the points may violate the local connectivity assumption. Hence, we use multiple

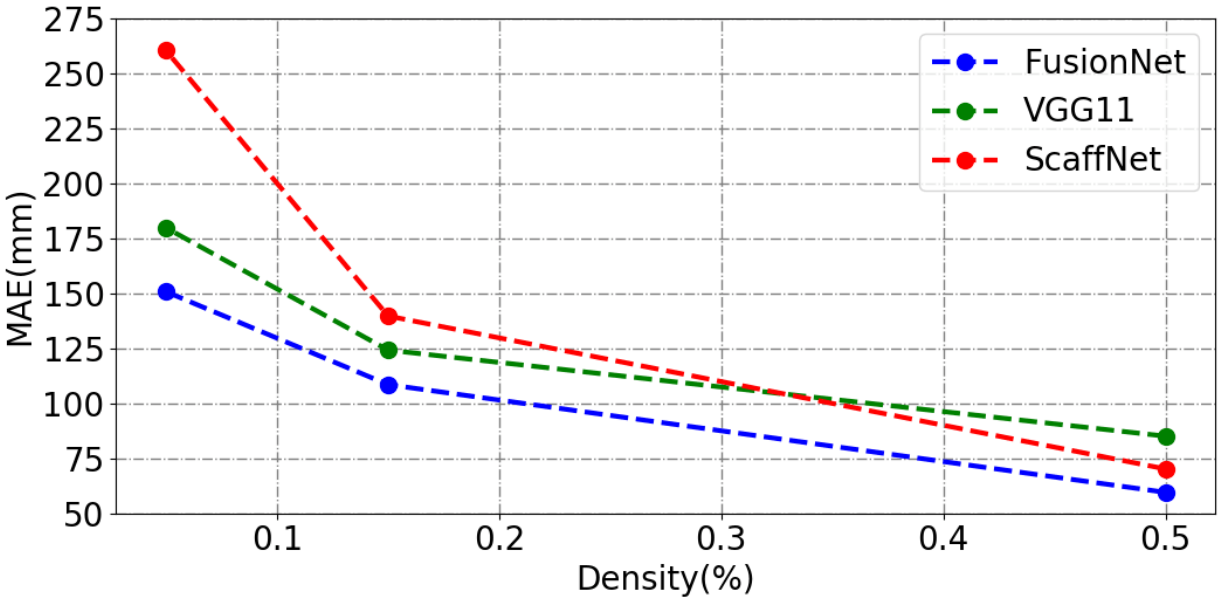


Figure 7.8: **Plot of MAE at different densities on the VOID depth completion benchmark.** Results of [WFT20] are taken directly from their paper. The three density levels examined are ≈0.5%, ≈0.15%, and ≈0.05%, which corresponds to ≈1500, ≈500, and ≈150 points, respectively. We show the trends of the MAE metric for ScaffNet, FusionNet and VGG11 [WFT20]. While ScaffNet beats VGG11 at the highest density (≈0.5%), with fewer points, ScaffNet performance degrades more quickly than FusionNet and VGG11 [WFT20]. This is because ScaffNet only takes sparse points as input (i.e without RGB image) and therefore cannot reliably infer the scene if there are very few or no sparse points. The improvement from ScaffNet to FusionNet is the effect of the errors amended by FusionNet.

kernel sizes for our max pool layers to capture local fine details (with small kernel sizes) and global denser structures (with large kernel sizes). To determine the trade-off between details and density, we leverage synthetic data to train three 1×1 convolutional layers to weight the output of the max pool layers. For network structure details, please see Sec. 7.5.1.

7.7.4 Differences from previous use cases

While variants of Spatial Pyramid Pooling have been employed in other problems such as classification and stereo matching, our use of Spatial Pyramid Pooling is unique. [HZR15b] introduced

Spatial Pyramid Pooling to ensure the same size feature maps are maintained when different size of inputs are fed through the network. Unlike us, [HZR15b] does not re-weight the features and directly feed max pooled results to fully connected layers. [CC18] used Spatial Pyramid (Average) Pooling with large kernels to create "region-level" features for increasing receptive field. In contrast to [CC18], we use max pooling to avoid loss of local detail from averaging large regions. Also unlike our use case in the depth completion problem, in classification and stereo, the inputs are *dense*; whereas, ours are *sparse*. Hence, we leverage the assumption that surfaces exhibit local smoothness and connectivity and perform max pooling with large kernels over local regions to get coarse local representations and max pooling with small kernels to retain detail. To balance the trade-off between detail and density, we re-weight the pooled features with 1×1 convolutions. This design not only allows us to obtain a denser representation, but also to obtain a larger receptive field (with similar motivation as [CC18]) – differentiating our variant of Spatial Pyramid Pooling from previous works.

7.8 Ablation Studies on Sparse Inputs with Various Density Levels

To understand the effect of sparse inputs of various density levels, we also show an quantitative study in Table 7.9 and Fig. 7.8. Fig. 7.8 compares the MAE metric between ScaffNet, FusionNet and VGG11 [WFT20] across three density levels, $\approx$0.5%, $\approx$0.15%, and $\approx$0.05%, which corresponds to $\approx$1500, $\approx$500, and $\approx$150 points, respectively. While ScaffNet beats VGG11 at the highest density ($\approx$0.5%), with fewer points, ScaffNet performance degrades at a much faster rate than FusionNet and VGG11 [WFT20]. This is because ScaffNet only takes sparse points as input (i.e without RGB image) and therefore cannot reliably estimate the topology when there are very few or no sparse points. Even though ScaffNet has comparable performance as many previous methods that use both sparse depth and image as input, this is precisely why FusionNet is critical in the success of our method – by performing cross-modal validation using the image and predicted topology to amend the incorrect predictions.

Table 7.9 shows that FusionNet is the best performing method across all metrics at every density

level. As expected, the performance of both ScaffNet and FusionNet degrade with lower density; however, we note that at $\approx$0.15% density, both ScaffNet and FusionNet are still comparable with the performance of methods at $\approx$0.5% density (see Table 7.8). This shows the effectiveness of our method even at low-density levels, which is a common scenario for indoor scenes (e.g. features of SLAM/VIO systems where points tracked must be visually discriminative and hence generally comprise of a small set). While our method does degrade with density (as do all known depth completion methods), we degrade more gracefully than [WFT20].

7.9 Discussion

To revisit the question, "is it *possible* to learn dense topology from sparse points?", we have demonstrated that it is, indeed, not only possible to learn the association of sparse points with dense natural shapes, but also using *only synthetic data*. While one may surmise that ScaffNet requires similar distributions of 3D scenes between synthetic and real datasets in order to generalize, we show the contrary; ScaffNet learns the shape of objects populating a scene rather than the scene itself as demonstrated in Sec. 7.6.2. This is especially important in the indoor settings where not only do the scene layouts vary a lot, but also consist of many textureless surfaces (for which one *needs* a prior on the shapes in the scene).

Method	MAE	RMSE	iMAE	iRMSE
Chodosh [CWL18]	439.48	1325.37	3.19	59.39
Dimitrievski [DVP18]	310.49	1045.45	1.57	3.84
Our FusionNet	*280.76*	*1121.93*	*1.15*	*3.30*
Ma [MCK19]	249.95	814.73	1.21	2.80
Qui [QCZ19]	226.50	758.38	1.15	2.56
Xu [XZS19]	235.73	785.57	1.07	2.52
Chen [CYL19]	221.19	752.88	1.14	2.34
Van Gansbeke [VND19]	215.02	772.87	0.93	2.19
Yang [YWS19]	**203.96**	832.94	**0.85**	2.10
Cheng [CWG19]	209.28	**743.69**	0.90	**2.07**

Table 7.6: **Supervised KITTI Depth Completion Benchmark.** *Quantitative results on the **supervised** KITTI depth completion benchmark*. All results are taken from the online benchmark [USS17]. Methods are ordered based on all metrics rather than just RMSE (ordering of benchmark). We note [MCK19, YWS19] compete in both unsupervised and supervised benchmarks. We compare our *unsupervised* method (italized, row 3 in the table) against supervised methods on the KITTI depth completion benchmark. We also note that while most supervised methods still do better, our approach surpasses some *supervised* methods: [CWL18] across all metrics, [DVP18] on MAE, iMAE, and iRMSE metrics and [MCK19] on iMAE. This demonstrates the potential of our method in closing the gap between supervised and unsupervised methods.

Method	MAE	RMSE	iMAE	iRMSE
Scaffolding [WFT20]	443.57	1990.68	1.72	6.43
Our ScaffNet w/o SPP	409.93	1776.42	1.72	6.40
Our ScaffNet w/ SPP	318.41	1425.53	1.39	5.01
Ma [MCK19]	358.92	1384.85	1.60	4.32
Yang [YWS19]	347.17	1310.03	n/a	n/a
VGG8 [WFT20]	308.81	1230.85	1.29	3.84
VGG11 [WFT20]	305.06	1239.06	1.21	3.71
Our FusionNet w/o SPP	306.54	1219.92	1.24	3.65
Our FusionNet w/ SPP	**286.35**	**1182.81**	**1.18**	**3.55**

Table 7.7: **Ablation study on the effect of Spatial Pyramid Pooling on KITTI validation set**. Results of [MCK19, YWS19, WFT20] are directly taken from their papers. Our ScaffNet w/o SPP omits the Spatial Pyramid Pooling (SPP) module. Our FusionNet w/o SPP uses ScaffNet w/o SPP as the initial topology estimator. Results of our ScaffNet with SPP consistently impoves its variant without the module. Similarly, because ScaffNet with SPP provides FusionNet with more accurately estimated topology, FusionNet like-wise improves.

Method	MAE	RMSE	iMAE	iRMSE
Ma [MCK19]	198.76	260.67	88.07	114.96
Yang [YWS19]	151.86	222.36	74.59	112.36
Our ScaffNet w/o SPP	100.75	242.27	71.32	191.60
VGG8 [WFT20]	98.45	169.17	57.22	115.33
VGG11 [WFT20]	85.05	169.79	48.92	104.02
Our FusionNet w/o SPP	77.62	140.36	51.58	91.84
Our ScaffNet w/ SPP	70.16	156.99	42.78	91.48
VGG11 + SLAM [WFT20]	73.14	146.40	42.55	93.16
Our FusionNet w/ SPP	**59.53**	**119.14**	**35.72**	**68.36**

Table 7.8: **Ablation study on the effect of Spatial Pyramid Pooling on VOID depth completion benchmark using ≈1500 points (≈0.5% density)**. Results of [MCK19, YWS19, WFT20] are taken from [WFT20]. Results of w/o SPP consistently performs worse than our model with SPP (marked with w/ SPP). We note that while errors do propagate from ScaffNet to FusionNet, FusionNet is able to amend them as see in the entry "Our FusionNet w/o SPP". We also note that our ScaffNet w/o SPP still outperforms [MCK19, YWS19].

	≈0.15% density			
Method	MAE	RMSE	iMAE	iRMSE
Our ScaffNet	139.60	308.47	71.50	151.49
VGG11 [WFT20]	124.11	217.43	66.95	121.23
Our FusionNet	**108.44**	**195.82**	**57.52**	**103.33**
	≈0.05% density			
Method	MAE	RMSE	iMAE	iRMSE
Our ScaffNet	260.13	531.69	115.21	218.27
VGG11 [WFT20]	179.66	281.09	95.27	151.66
Our FusionNet	**150.65**	**255.08**	**80.79**	**133.33**

Table 7.9: **Ablation study on various densities of sparse inputs.** Results of [WFT20] are taken from their papers. We examined two other density levels, ≈0.15%, and ≈0.05%, in addition to ≈0.5% density shown in the official benchmark (see Table 7.8), which corresponds to ≈500, and ≈150 points, respectively. The performance of ScaffNet and FusionNet decreased (as expected) proportional to the density, but ScaffNet decreases at a faster rate with fewer points, especially at 0.05% density. This is because ScaffNet needs to infer topology from only ≈150 points without the help of an image. However, with an input of ≈0.15% density, our approach still produces reasonable results, with numbers similar to that of [MCK19, YWS19, WFT20] using ≈0.5% input density (see Table 7.8).

CHAPTER 8

Targeted Adversarial Perturbations for Monocular Depth Prediction

Figure 8.1: **Altering the predicted scene with adversarial perturbations.** Top to bottom: input image; adversarial perturbations with upper norm of 2×10^{-2}; predicted scene visualized as disparity. Left to right: original image and predicted scene; overall scene altered to be 10% closer; all vehicles altered to be 10% closer; vehicle in the center of the road is removed by perturbations.

8.1 Introduction

Consider the image shown in the top-left of Fig. 8.1, captured from a moving car. The corresponding depth of the scene, inferred by a deep neural network and visualized as disparity, is shown underneath. Can adding a small perturbation cause the perceived vehicle in front of us to disappear? Indeed, this is shown on the rightmost panel of the same figure: The perturbed image, shown on the top-right, is indistinguishable from the original. Yet, the perturbation, amplified and shown in the center row, causes the depth map to be altered in a way that makes the car in front of us disappear.

Adversarial perturbations are small signals that, when added to images, are imperceptible yet

can cause the output of a deep neural network to change catastrophically [SZS13]. We know that they can fool a network to mistake a tree for a peacock [MFF16]. But, as autonomous vehicles are increasingly employing learned perception modules, mistaking a stop sign for a speed limit [EEF18] or causing obstacles to disappear is not just an interesting academic exercise. We explore the possibility that small perturbations can alter not just the class label associated to an image, but the inferred depth map, for instance to make the entire scene appear closer or farther, or portions of the scene, like specific objects, to become invisible or be perceived as being elsewhere in the scene.

When semantic segmentation is available, perturbations can target a specific category in the predicted scene. Some categories (e.g. traffic lights, humans) are harder to attack than others (e.g. roads, nature). When instance segmentation is available, perturbations can manipulate individual objects, for instance make a car disappear or move it to another location. We call these phenomena collectively as *stereopagnosia*, as the solid geometric analogue of prosopagnosia [DDV82].

Stereopagnosia sheds light on the role of context in the representation of geometry with deep networks. When attacking a specific category or instance, while most of the perturbations are localized, some are distributed throughout the scene, far from the object of interest. Even when the target effect is localized (e.g., make a car disappear), the perturbations are non-local, indicating that the network exploits a non-local context, which represents a vulnerability. Could one perturb regions in the image, for instance displaying billboards, thus making cars seemingly disappear?

We note that, although the adversarial perturbations we consider are not universal, that is, they are tailored to a specific scene and its corresponding image, they are somewhat robust. Blurring the image after applying the perturbations reduces, but does not eliminate, stereopagnosia. To understand the generalizability of adversarial perturbations, we examine the transferability of the perturbations between two monocular depth prediction models with different architectures and losses.

8.2 Related Work

Adversarial perturbations have been studied extensively for classification (Sec. 8.2.1). We focus on regression, where there exists some initial work. However, we study the *targeted* case where the entire scene, a particular object class, or even an instance is manipulated by the choice of perturbation.

8.2.1 Adversarial Perturbations

The early works [SZS13, GSS14] show the existence of small, imperceptible additive noises that can alter the predictions of the classification networks. Since then many more advanced attacks [MFF16] have been proposed. [MFF17] showed the existence of universal perturbations i.e. constant additive perturbations to degrade the accuracy over the entire dataset.

Despite the exponentially growing literature on adversarial attacks for the classification task, there only have been a few works extending the analysis of adversarial perturbations to dense-pixel prediction tasks. [XWZ17] studied adversarial perturbations for detection and segmentation. [HCB17] demonstrated targeted universal attacks for semantic segmentation. [MGB18] examined universal perturbations in a data-free setting for segmentation and depth prediction to alter predictions in arbitrary directions. Unlike them, we study *targeted* attacks where the network is fooled to predict a specific target.

Our goal is to analyze the robustness of the monocular depth prediction networks to different targeted attacks to explore possible explanations of what is learned by these models. With a similar motivation, [HZO19] identified the smallest set of image pixels from which the network can estimate a depth map with small error. Unlike them, we analyze the monocular depth networks by studying their robustness against *targeted adversarial* attacks.

8.2.2 Monocular Depth Prediction

[EPF14, EF15, LRB16] trained deep networks with ground-truth annotations to predict depth from a single image. However, high-quality depth maps are often unavailable and, when available, are

expensive to acquire. Recently, supervisory trends shifted to unsupervised (self-supervised) learning, which relies on stereo-pairs or video sequences during training, and provides supervision in the form of image reconstruction. While depth from video-based methods is up to an *unknown* scale, stereo-based methods can predict depth in *metric* scale because the pose (baseline) between the cameras is known.

To learn depth from stereo-pairs, [GBC16] predicted disparity by reconstructing one image from its stereo-counterpart. Monodepth [GMB17] predicted both left and right disparities from a single image and laid the foundation for [PTM18, PAG19, WHS19]. To learn depth from videos, [MWA18, ZBS17] also jointly learned pose between temporally adjacent frames to enable image reconstruction by reprojection. [WMZ18, YWS18] leveraged visual odometry, [FWS18] used gravity, [CPM19, LYW18] considered motion segmentation, and [YS18] jointly learned depth, pose and optical flow. Monodepth2 [GMF19] explored both stereo and video-based methods and proposed a reprojection loss to discard potential occlusions.

To study the effect of adversarial perturbations, we examine the robustness of Monodepth2, the state-of-the-art, and its predecessor, Monodepth. While Monodepth2 proposed both stereo and video-based models, we choose their stereo model because the predicted depth is in *metric* scale, which enables us to study perturbations to alter the scale of the scene without changing its topology.

In Sec. 8.3, we discuss our method. We show perturbations for altering entire predictions to a target scene in Sec. 8.4 and localized attacks on specific categories and object instances in Sec. 8.5. We discuss the transferabilty of such perturbations in Sec. 8.6 and their robustness against defenses in Sec. 8.7. In Sec. 8.8, existence of the successful adversarial attacks for indoor scenes (NYU-V2) is shown for state-of-the-art indoor monocular depth prediction model. In Sec. 8.9, we examine how predictions behave when linear operations are applied to perturbations (sum of two perturbations and linear scaling of a perturbation).

8.3 Finding Targeted Adversarial Perturbations

Given a pretrained depth prediction network, $f_d : \mathbb{R}^{H\times W\times 3} \rightarrow \mathbb{R}_+^{H\times W}$, $f_d : x \mapsto d(x)$ our goal is to find a small additive perturbation $v(x) \in \mathbb{R}^{H\times W\times 3}$, as a function of the input image x, which can change its prediction to a target depth $f_d(x + v(x)) = d^t(x) \neq d(x)$ with some norm constraint $||v(x)||_\infty \leqslant \xi$ and high probability $\mathbb{P}(f_d(x + v(x)) = d^t(x)) \geqslant 1 - \delta$.

We begin by examining Dense Adversarial Generation (DAG) proposed by [XWZ17] for finding adversarial perturbations for the semantic segmentation task. The perturbations from DAG can be formulated as the sum of a gradient ascent term (that pushes the predictions away from those of the original image) and a gradient descent term (that pulls predictions towards the target predictions). In the case of semantic segmentation, this formulation works well because the gradient ascent term suppresses the probability for the original predictions, which naturally increases the probability of the target predictions (zero-sum) driven by the gradient descent term. However, such is not the case for regression tasks, which require the network to predict a real-valued scalar (as opposed to probability mass) for a targeted scene. Hence, the gradient ascent term maximizes the difference between the original and predicted depth, which results in DAG "overshooting" the target depth.

Instead, we use a simple objective function, similar to [HCB17], but we modify it for the regression task by minimizing the normalized difference between predicted and target depth,

$$\ell(x, v(x), d^t(x), f_d) := \frac{||f_d(x + v(x)) - d^t(x)||_1}{d^t(x)}. \tag{8.1}$$

We minimize this objective function with respect to an image x by following an iterative optimization procedure as detailed in Alg. 4. The $\texttt{CLIP}\big(v_n(x), -\xi, \xi\big)$ operation clamps any value of $v(x)$ larger than ξ to ξ and any value smaller than $-\xi$ to $-\xi$. For all the experiments, $\xi \in \{2 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}, 2 \times 10^{-2}\}$.

8.3.1 Implementation Details

We evaluate adversarial targeted attacks on KITTI semantic split [AMM18]. This is a dataset of 200 outdoor scenes, captured by car-mounted stereo cameras and a LIDAR sensor, with ground-truth

Algorithm 4 Proposed method to calculate targeted adversarial perturbations for a regression task.

Parameters: Learning rate η, noise upper norm ξ.

Inputs: Image x, target depth map $d^t(x)$, pretrained depth network f_d.

Outputs: Perturbation $v_N(x)$.

Init: $v_0(x) = 0$.

for $n = 0 : N - 1$ **do**

$v_n(x) = \texttt{CLIP}\big(v_n(x), -\xi, \xi\big)$

Calculate $\ell(x, v_n(x), d^t(x), f_d)$ as defined in Eqn. 8.1.

$v_{n+1}(x) = v_n(x) - \eta \nabla \ell(x, v_n(x), d^t(x), f_d)$

$v_N(x) = \texttt{CLIP}\big(v_N(x), -\xi, \xi\big)$

semantic segmentation and instance labels. The semantic and instance labels in this split enables our experiments in Sec. 8.5 for targeting specific categories or instances in a scene.

The depth models (Monodepth, Monodepth2) are trained on the KITTI dataset [GLU12] using Eigen split [EF15]. The Eigen split contains 32 out of the total 61 scenes, and is comprised of 23,488 stereo pairs with an average size of 1242×375. Images are resized to 640×192 as a preprocessing step and perturbations are computed with 500 steps of SGD. Entire optimization for each frame takes $\approx$ 12s (Monodepth2 takes 22ms for each forward pass and 11s $\approx 500 \times$ 22ms in total) using a GeForce GTX 1080.

For all the experiments, we use absolute relative error (ARE), computed with respect to the target depth $d^t(x)$, as our evaluation metric:

$$\text{ARE} = \frac{||f_d(x + v(x)) - d^t(x)||_1}{d^t(x)}. \tag{8.2}$$

8.3.2 Monodepth and Monodepth2

We study the effects of adversarial perturbations on the state-of-the-art monocular depth prediction method, Monodepth2 [GMF19] and its predecessor Monodepth [GMB17]. The two models utilize different network architectures and are trained with different loss functions. In this section, we provide details on the two methods.

Regarding *Monodepth*: Monodepth uses a ResNet50 encoder architecture as its backbone and a standard decoder with skip connections. Monodepth predicts both left and right disparities from a single image (assuming it is the left image of a stereo-pair) and uses image reconstruction as supervision. Additionally, it is trained with a standard local smoothness term weighted by image gradients and a left-right disparity consistency term as its regularizers.

Regarding *Monodepth2*: Monodepth2, unlike Monodepth, uses ResNet18 encoder (pretrained on ImageNet) as its backbone network architecture. Rather than simply minimizing an image reconstruction loss, Monodepth2 leverages a heuristic to discount occluded pixels and also uses a criterion to discount static frames. Similar to Monodepth, Monodepth2 also minimizes a local smoothness regularizer weighted by image gradients.

8.4 Attacking the Entire Scene

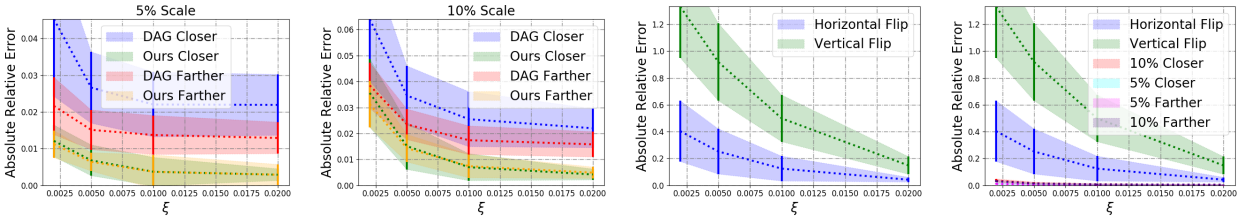
Given a depth network f_d, our goal is to find adversarial perturbations to alter the predictions to a target scene $d^t(x)$ for an image x. For this, we examine three settings (i) scaling the entire scene by a factor, (ii) symmetrically flipping the scene, and (iii) altering the scene to a preset scene.

8.4.1 Scaling the Scene

For autonomous navigation, misjudging an obstacle to be farther away than it is could prove disastrous. Hence, to alter the distances in the predicted scene without changing its topology or structure, we examine perturbations that will scale the scene (bringing the scene closer to or farther away from the camera) by a factor of $1 + \alpha$. The target scene is defined as:

$$d^t(x) = \texttt{scale}(f_d(x)) = (1 + \alpha) f_d(x) \tag{8.3}$$

for $\alpha \in \{-0.10, -0.05, +0.05, +0.10\}$ or -10%, -5% (closer), $+5\%$, $+10\%$ (farther), respectively. Column two of Fig. 8.1 shows the scene scaled 10% closer to the camera by applying visually imperceptible perturbations with $\xi = 2 \times 10^{-2}$. On average, scaling the scene by -10%, -5%, $+5\%$, $+10\%$ with $\xi = 2 \times 10^{-2}$ require an $\|v(x)\|_1$ of 0.0160, 0.0124, 0.0126, and 0.0161, respectively.



(a) (b) (c) (d)

Figure 8.2: **ARE with various upper norm ξ for scaling and flipping Monodepth2 predictions.** (a) and (b) Comparisons between DAG and the proposed method for scaling the scene by $\pm 5\%$ and $\pm 10\%$. (c) Results for horizontally and vertically flipping the predictions. (d) comparison between scaling and flipping tasks. Vertically flipping proves to be the most challenging.

We note that scaling the scene by $\pm 5\%$ requires less perturbations than $\pm 10\%$ and the magnitude required for both directions is approximately symmetric. Also, perturbations are typically located along the object boundaries with concentrations on the road.

In Fig. 8.2-(a, b), we compare our approach with DAG, (Sec. 8.3) re-purposed for depth prediction task. While both are bounded by the same upper norm, DAG consistently produces results with higher error and generally with a higher standard deviation. As seen in Fig. 8.2-(a, b), even with $\xi = 2 \times 10^{-3}$, we are able to find perturbations that can scale the scene to be $\approx 1\%$ from $\pm 5\%$ and $\approx 3\%$ from $\pm 10\%$. With $\xi = 2 \times 10^{-2}$, we are able to fully reach all four targets with less than $\approx 0.5\%$ error.

In Fig. 8.2-(a, b), we showed that it is possible, even for small norms such as $\xi = 2 \times 10^{-3}$ to scale the scene to be 5% or 10% closer or farther with small error. We also demonstrate that it is possible to scale the scene by larger amounts (up to 30% closer or farther). Fig. 8.3 shows that with $\xi = 2 \times 10^{-3}$, it is only able to scale the up to 15% with reasonable error; whereas perturbations with $\xi = 5 \times 10^{-3}$ can achieve this up to 20% closer or farther. However, using larger norms (1×10^{-2} and $\xi = 2 \times 10^{-2}$), one can scale the scene up to 30% with small errors (less than 5% ARE for $\xi = 1 \times 10^{-2}$ and $\approx 1\%$ ARE for $\xi = 2 \times 10^{-2}$). We note that $\xi = 2 \times 10^{-2}$, is still able to obtain less than 2% ARE when scaling the scene by 45%; however, standard deviation starts to grow larger as the scaling increases.

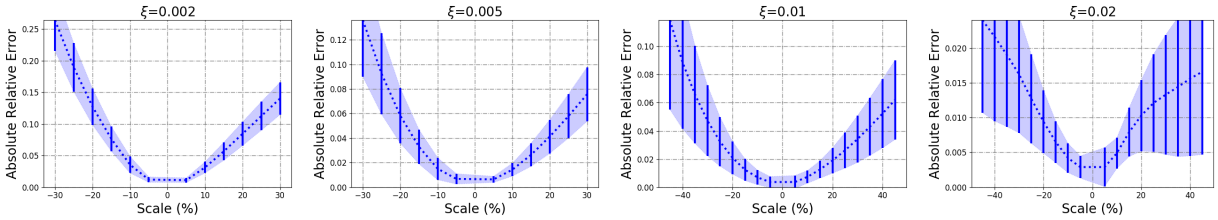


Figure 8.3: **ARE with various upper norm ξ for scaling Monodepth2 predictions.** This time error is plotted for large scale ratios 15%, 20%, 25% and 30% (up to 45% for larger ξ), for scaling both closer and farther, showing the limitations of each norm for the scaling task. While ARE is still relatively small for larger norms, standard deviation grows larger – meaning the perturbations can no longer scale the scene consistency with low error.

As we can see, for smaller norms of 2×10^{-3} and 5×10^{-3}, the perturbations are limited to scaling the scene by $\approx 15\%$ and $\approx 20\%$, respectively. Scaling factors higher than such increases the ARE by $\approx 4\%$ for every 5% increase in scaling factor, signaling the limit for these norms. For larger norms of 1×10^{-2} and 2×10^{-2}, the perturbations can afford to scale the scene by a much larger factor. For $\xi = 1 \times 10^{-2}$, perturbations can scale the scene by as much as 30% closer and farther less than 5% error. Whereas, for $\xi = 1 \times 10^{-2}$, perturbations can scale the scene up to $\pm 45\%$ with less than 2% ARE.

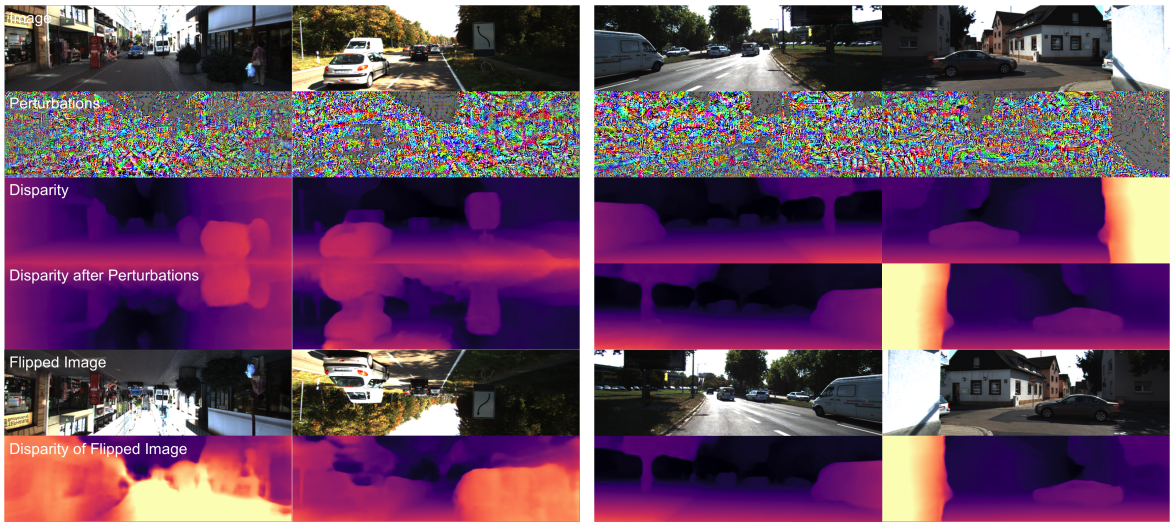
While for smaller scales (e.g. ± 5, ± 10) the ARE and amount of noise required is approximately the same, suggesting similar difficulty levels. As we plot the errors for larger scales in Fig. 8.3, scaling the scene farther generally yields lower error than scaling the scene closer.

8.4.2 Symmetrically Flipping the Scene

We now examine the problem setting where the target scene still retains the same structures given by the image, however, they are mirrored across the y- (horizontal flip) or x-axis (vertical flip).

For the *horizontal flip* scenario, we denote the target depth as $d^t(x) = \texttt{fliph}(f_d(x))$ where the `fliph` operator horizontally flips the predicted depth map $f_d(x)$ across the y-axis.

Fig. 8.4-(b) shows that the perturbations can fool the network into predicting horizontally flipped scenes. For scenes with different structures on either side, $v(x)$ fools the network into *creating*



(a) (b)

Figure 8.4: (a) **Examples of success (left) and failure (right) cases for vertical flip**. For the failure case, the car and road still remain on the bottom of the predictions. This is likely because the network is biased to predict closer structures on the bottom half of the image and farther ones on the top half (last two rows). (b) **Examples of horizontal flip.** Here, we observe the noise required to create and remove surfaces. Surprisingly, removing the white wall (right) requires very little perturbations.

and removing surfaces. We note that the amount of noise required to horizontally flip the scene is much more than that of scaling the scene (i.e. for $\xi = 2 \times 10^{-2}$, $\|v(x)\|_1 = 0.161$ for scaling $+10\%$ and 0.326 for flipping horizontally), which illustrates the difficulty in altering the scene structures. Interestingly, the amount of noise required to remove the white wall (Fig. 8.4-(b)) is significantly less than the rest.

For the *vertical flip* scenario, we denote the target depth as $d^t(x) = \texttt{flipv}(f_d(x))$ where `flipv` operator vertically flips the predicted depth map $f_d(x)$ across the x-axis.

As seen in Fig. 8.4-(a), perturbations cannot fully flip the predictions vertically. Even on successful attempts (left), there are still artifacts in the output. For failure cases (right), portions of the cars still remain on the bottom half of the predictions. This experiment reveals the potential *biases* learned by the network. To verify this, we feed vertically flipped images to the network. As seen in the last two rows of Fig. 8.4-(a), the network still assigns closer depth values to the bottom

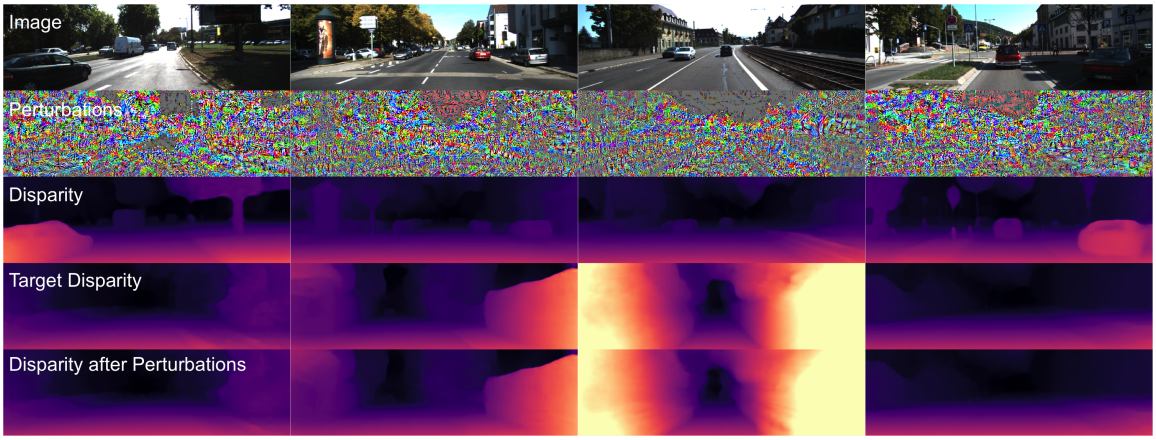


Figure 8.5: **Altering the predicted scene to a preset scene.** Adversarial perturbations can remove a car and replace a road sign with trees (leftmost), add walls (column two), and vegetation (column three) to open streets, and transform an urban environment with vehicles to an open road (rightmost). half of the image (now sky) and farther depth values to the top half (now road and cars).

In Fig. 8.2-(c, d), we plot the ARE achieved by the proposed method for different target depth maps: horizontal flip, vertical flip and different scales. Both flipping tasks are much harder than the scaling tasks. Particularly, fooling the network to produce vertically flipped predictions is the most challenging task as the error is $\approx 15\%$, even with $\xi = 2 \times 10^{-2}$.

8.4.3 Altering Predictions to Fit a Preset Scene

We now examine perturbations for altering the predicted scene $d(x_1) = f_d(x_1)$ to an entirely different pre-selected one $d^t(x_1) = f_d(x_2)$ obtained from images sampled from the same training distribution $x_1, x_2 \sim \mathbb{P}(x)$: $d(x_1) = f_d(x_1) \neq d^t(x_1) = f_d(x_2)$.

Fig. 8.5 shows that cars can be removed and road signs can be replaced with trees (leftmost), walls (column two) and vegetation (column three) can be added to the scene, and an urban street with vehicles can be transformed to an open road (rightmost). While perturbations are visually imperceptible, we note that $\|v(x)\|_1 = 0.362$, which is $\approx 2\times$ the amount required for horizontal flip. However, the existence of such perturbations demonstrates just how vulnerable depth prediction networks can be.

Additionally, this experiment also confirms the biases learned by the network discussed in

Sec. 8.4.2. While perturbations can alter the scene to a preset one with *structures not present* in the image, we have difficulties finding perturbations that can vertically flip the predicted scene.

8.5 Localized Attacks on the Scene

Given semantic and instance segmentation [AMM18], we now examine adversarial perturbations to target localized regions in the scene. Our goal is to fool the network into (i) predicting depths that are closer or farther by a factor of $1 + \alpha$ for all objects belonging to a semantic category, (ii) removing specific instances from the scene, and (iii) moving specific instances to different regions of the scene, all the while keeping the rest of the scene unchanged.

8.5.1 Category Conditioned Scaling

Unlike Sec. 8.4.1, we want to alter a subset of the scene, partitioned by semantic segmentation, such that predictions belonging to an object category (e.g. vehicle, nature, human) are brought closer to or farther from the camera by a factor of $1 + \alpha$ for $\alpha \in \{-0.10, -0.05, +0.05, +0.10\}$.

We assume a binary category mask $M \in \{0, 1\}^{H \times W}$ derived from a semantic segmentation where all pixels belonging to a category are marked with 1 and 0 otherwise. We denote the target depth as

$$d^t(x) = (\mathbf{1} - M) \circ f_d(x) + (1 + \alpha) M \circ f_d(x) \tag{8.4}$$

where $\mathbf{1}$ is an $H \times W$ matrix of 1s. Column three of Fig. 8.1 illustrates this problem setting where the perturbations fool Monodepth2 into predicting all vehicles to be 10% closer to the camera. Fig. 8.6 shows a comparative study between different categories. Unlike Sec. 8.4.1, it is more difficult to alter a specific portion of the scene without affecting the rest. Moreover, each category exhibits a different level of robustness to adversarial noise. *Some categories are harder to attack than others*, e.g. traffic signs and human categories ($\approx 3\%$ error for $\alpha = \pm 5\%$ and $\approx 6\%$ for $\alpha = \pm 10\%$) are harder to alter than vehicle and nature ($\approx 0.5\%$ error for $\alpha = \pm 5\%$ and $\approx 1\%$ for $\alpha = \pm 10\%$).

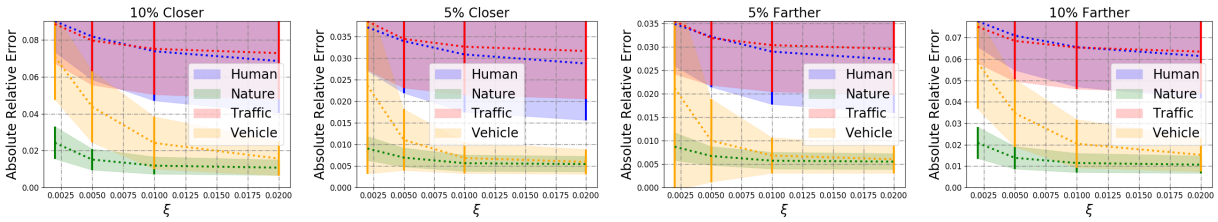


Figure 8.6: **ARE for scaling different categories closer and farther.** It is easier to fool the network to predict vehicle and nature categories closer and farther than is to fool human and traffic categories.

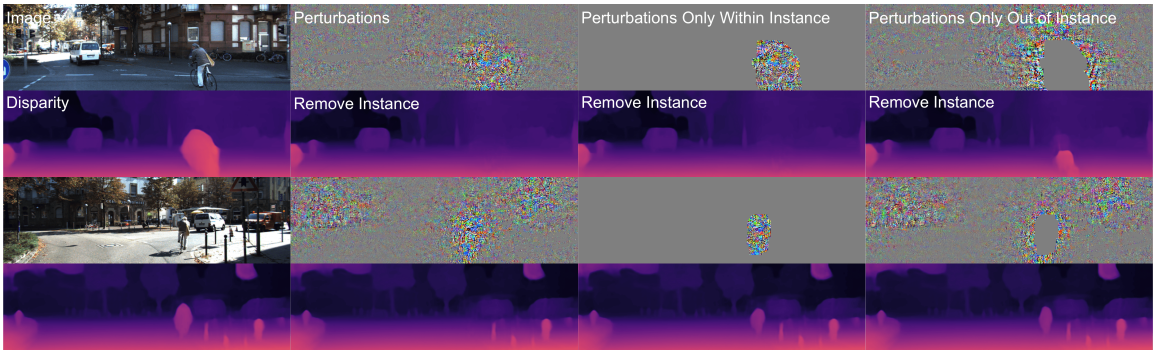


Figure 8.7: **Selectively removing instances of human (bikers and pedestrians).** Removing a localized target requires attacking non-local contextual information. Moreover, one can attack an instance without perturbing it at all. We demonstrate this by constraining the perturbation to be either completely within the instance mask or completely out of the mask.

8.5.2 Instance Conditioned Removing

We now consider the case where instance labels are available for removing a specific instance (e.g. car, pedestrian) from the scene. By examining this scenario, we hope to shed light on the possibility that a depth prediction network can "miss" a human or car, which may cause incorrect rendering in augmented reality or an accident in the autonomous navigation scenario.

Similar to Sec. 8.5.1, we assume a binary mask M, but in this case, of specific instance(s) in the scene, e.g. all pixels belonging to a specific pedestrian are marked with 1 and 0 otherwise. To obtain $d^t(x)$, we first remove the depth values in $f_d(x)$ belonging to M by multiplying $f_d(x)$ by $\mathbf{1} - M$. Then, we use the depth values $f_d(x)$ on the contour of M to linearly interpolate the depth

in the missing region:

$$d^t(x) = (\mathbf{1} - M) \circ f_d(x) + M \circ d^t_M(x) \tag{8.5}$$

where $d^t_M(x) := \texttt{interp}(\texttt{contour}(f_d(x), M))$. Fig. 8.7 shows examples of pedestrian and biker removal in the driving scenario where perturbations completely remove the targeted instance. With this attack, the road ahead becomes clear, which makes the agent susceptible to causing an accident.

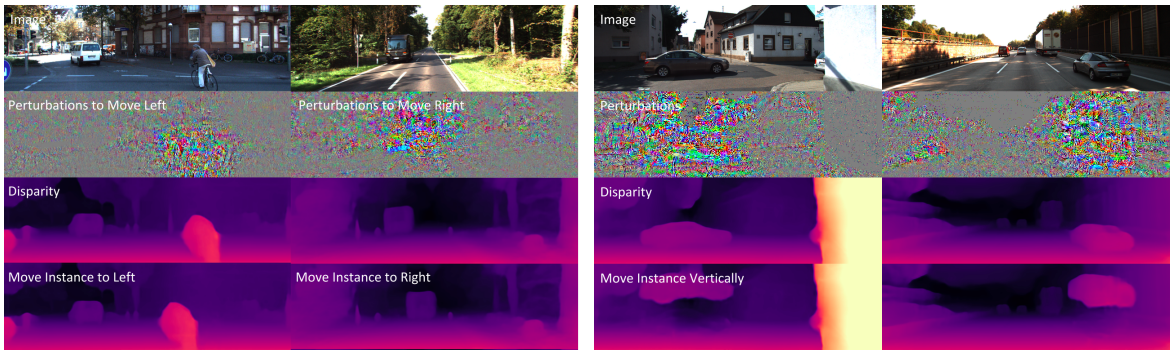
Even though perturbations are concentrated on the targeted instance region, non-zero perturbations can be observed in the surrounding regions. While the target effect is localized (e.g., make a pedestrian disappear), the perturbation is non-local, implying that the network exploits non-local context, which presents a vulnerability to attacks against a target instance by perturbing other parts of the image.

8.5.3 Instance Conditioned Removing with Spatial Constraints on the Perturbations

Motivated by our results in Sec. 8.5.2, we extend the instance conditioned removal task to more constrained scenarios where the perturbations have to either exist (i) completely within the target instance mask M or (ii) completely outside of it.

For perturbations within the targeted instance mask M, we constrain $v(x)$ to satisfy $||M \circ v(x)||_\infty \leqslant \xi$ and $||(\mathbf{1} - M) \circ v(x)||_\infty = 0$. When constrained within M, perturbations can only remove *some instances* successfully (e.g. biker is completely removed in row two, column three of Fig. 8.7). In other cases, the perturbations can only remove the outer part of the instance, leaving parts of the instance in the scene (row four). This shows that depth prediction networks leverage global context; without attacking the contextual information located outside of M (e.g. without perturbing the entire image as in column two of Fig. 8.7), it is not always possible to completely remove the target instance.

Second, we want to answer the question posed in Sec. 8.1. Can perturbations remove an object by attacking anywhere (e.g. a billboard), *but the object* (e.g. a car)? In this more challenging case, the perturbations are constrained to be outside of the instance mask: $||(\mathbf{1} - M) \circ v(x)||_\infty \leqslant \xi$ and $||M \circ v(x)||_\infty = 0$. Column four of Fig. 8.7 shows that even though there are no "direct attacks



(a) (b)

Figure 8.8: (a) **Moving horizontally.** Selected instances is moved by $\approx 8\%$ in the left and right directions while rest of the scene is preserved. (b) **Flying cars.** A vehicle instance is moved $\approx 42\%$ upward while rest of the scene is preserved. Noise is concentrated around the targeted instance.

on the object" (perturbations in the masked region), the perturbations can *still remove parts of the target instance*. While some of the target instance still remains, our experiment demonstrates that depth prediction networks are indeed susceptible to *attacks against a target instance that does not require perturbing the instance at all*.

8.5.4 Instance Conditioned Translation

In this case study, we examine perturbations for moving an instance (e.g. vehicle, pedestrian) horizontally or vertically in the image space. As Sec. 8.5.2 and 8.5.3 have demonstrated the ability to remove localized objects from the scene, we now show that it is possible for perturbations to move such objects to different locations in the scene (removing the instance and creating it elsewhere) while keeping the rest of the scene unchanged.

Fig. 8.8-(a) shows that perturbations can fool a network to move the target instance by $\approx 8\%$ across the image in the left and right directions. When moved left, the biker (left column) is now in front of our vehicle. When moved right, the truck (right column) is in the wrong lane and looks to be on-coming traffic. Moreover, Fig. 8.8-(b) shows that perturbations can move select instances by $\approx 42\%$ in the upward direction, creating the illusion that there are "flying cars" in the scene.

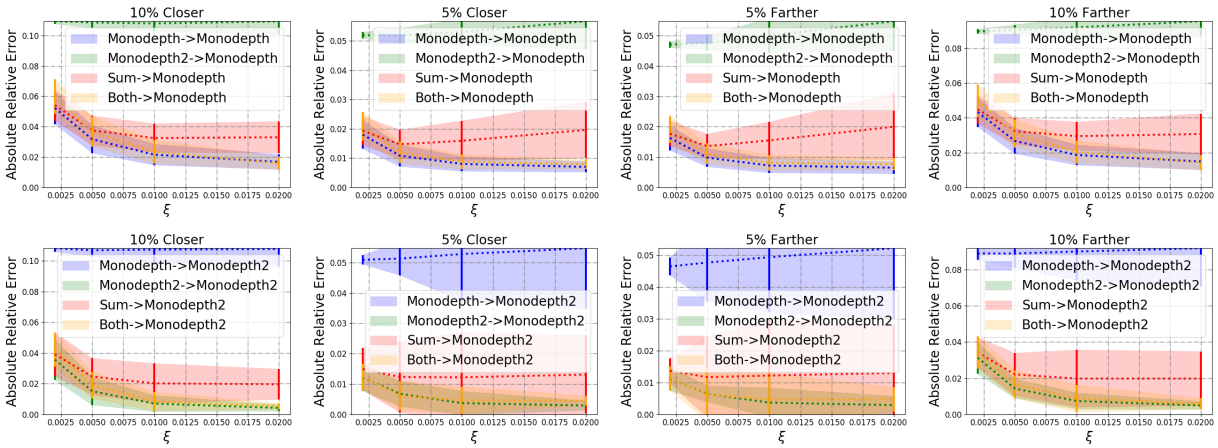


Figure 8.9: **Transferability across models.** Perturbations are (i) optimized for Monodepth and Monodepth2 separately, (ii) optimized for both together and (iii) summed over perturbations calculated for Monodepth and Monodepth2 separately. Each is tested on Monodepth and Monodepth2.

8.6 Transferability Across Different Models

Transferability is important for black-box scenarios, a practical setting where the attacker does not have access to the target model or its training data. To examine transferability, we test our perturbations crafted for Monodepth2 [GMF19] to fool its predecessor Monodepth [GMB17] (different architecture and loss function) in Monodepth2→Monodepth, and vice versa in Monodepth→Monodepth2, for the scene scaling task. (Sec. 8.4.1).

To this end, we also optimized perturbations for Monodepth to scale the entire scene. Overall, the perturbations optimized for one model does not transfer to another and, interestingly, transferability decays with increasing norm (Fig. 8.9), which may be due to perturbations overfitting to the model. We summed the perturbations for Monodepth and Monodepth2 ("Sum" in Fig. 8.9) and found that their summation can *affect both models* with reduced effects as the upper norm increases. For $\xi = 2 \times 10^{-3}$, the potency is nearly unaffected, meaning, for small norms, their summation can attack both models equally well. Lastly, by optimizing for both models ("Both" in Fig. 8.9), the *same perturbation* can fool both as if it was optimized for the models individually, with performance indistinguishable from Monodepth→Monodepth and Monodepth2→Monodepth2 *across all norms*. This shows that both models share a space that is vulnerable to adversarial attacks. Hence, crafting

perturbations for an array of potential models may be an avenue towards achieving absolute transferability across models.

8.7 Robustness of the Targeted Attacks Against Defense Mechanisms

In this section, we will examine the robustness of the perturbations against common defense mechanisms: (i) Gaussian blurring and (ii) adversarial training.

8.7.1 Defense through Gaussian Blurring

In Fig. 8.10, we show the effect of Gaussian blurring as a simple defense mechanism on our targeted attacks by blurring the image with additive perturbations. Even though Gaussian blur does reduce the effectiveness of the perturbations (increased ARE over all scales), the resulting scene is still only $\approx 3\%$ away from a target depth that is 10% closer or farther than the original predictions for $\xi = 2 \times 10^{-2}$. This is the performance that the method achieves for the case of $\xi = 2 \times 10^{-3}$ without blurring. In other words, the effect of the blurring can be suppressed simply by increasing the upper norm of the noise by $10\times$ for scaling the scene by $\pm 10\%$.

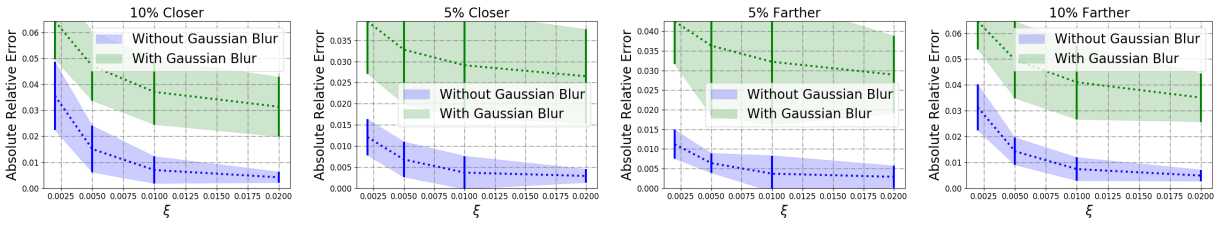


Figure 8.10: **Gaussian blur.** Absolute relative error (ARE) achieved by adversarial perturbations of different norms (ξ) for different scales. For each scale, we plot the ARE with and without Gaussian blur. Even though absolute relative error increases with the Gaussian blur, the proposed method can still find a small norm noise to alter the scene.

8.7.2 Defense through Adversarial Training

To examine the robustness of adversarial perturbations to adversarial training, we crafted adversarial perturbations for scaling the scene by a factor of $1 + \alpha$ where $\alpha \in \{-0.10, -0.05, +0.05, +0.10\}$ for the KITTI Eigen split [EF15] (consisting of 22600 stereo pairs). We trained Monodepth2 [GMF19] by minimizing the normalized discrepancy between the predicted depth of a perturbed image ($f_d(x + v(x))$) and its prediction for the original image ($f_d(x)$).

$$\ell(x, v(x), f_d) = \frac{\|f_d(x) - f_d(x + v(x))\|_1}{f_d(x)} \tag{8.6}$$

Fig. 8.11 shows the effect of the perturbations on Monodepth2 after adversarial training. While training does reduce the influence of adversarial perturbations on the scene scaling task, it does not make the network invariant to the adversarial perturbations. With perturbations of $\xi = 2 \times 10^{-3}$, the predict scene is $\approx 7\%$ from the target scene that is 10% closer than or farther from the original and $\approx 3\%$ from the target scene that is 5% closer or farther. For $\xi = 2 \times 10^{-2}$, perturbations can still fool the network to be predict a target scene scaled by $\pm 10\%$ with $\approx 5\%$ absolute relative error and $\approx 2\%$ error for fooling the network to predict a target scene that is scaled by $\pm 10\%$.

To compare the two defense mechanisms, we refer to Fig. 8.12. For smaller norms, e.g. $\xi = 2 \times 10^{-3}$, we observe a similar performance in using Gaussian blur (Sec. 8.7.1) and adversarial training as defenses against adversarial perturbations; whereas, adversarial training is clearly better for larger ξ. This may be due to Gaussian blur's ability to destroy the perturbation for small norms and, hence, able to mitigate the effect of the perturbations. However, for larger norms, the blurring does not corrupt the perturbations enough and therefore does not reduce the effect of perturbations by as much.

8.8 Adversarial Attacks for Indoor Scenes

To show the applicability of the adversarial method on indoor scenes, we examine the adversarial perturbations for Kinect Dataset NYU Depth V2 (NYU-V2) [SHK12]. We tested the effectiveness of adversarial perturbations on Virtual Normal Loss (VNL) [YLS19] which is the state-of-the-art

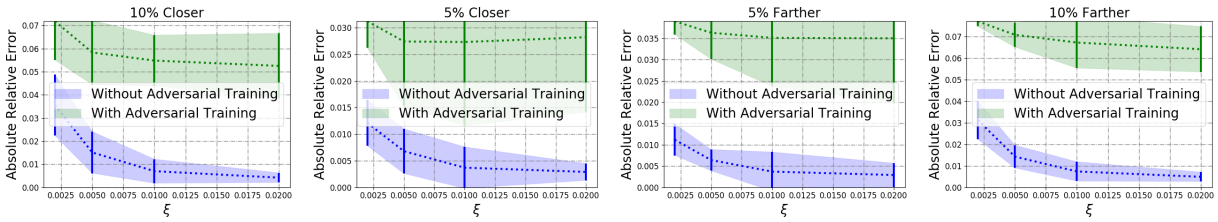


Figure 8.11: **Adversarial training.** Absolute relative error (ARE) achieved by adversarial perturbations of different norms (ξ) for different scales. For each scale, we plot the ARE with and without adversarial training. Even though absolute relative error increases with the adversarial training, the perturbations can still affect the predicted scene with small norm noise.

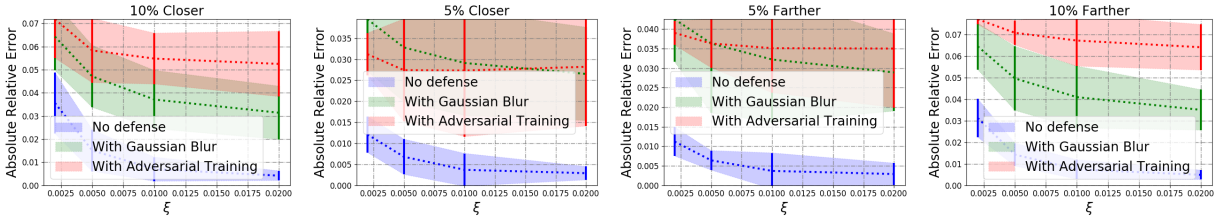


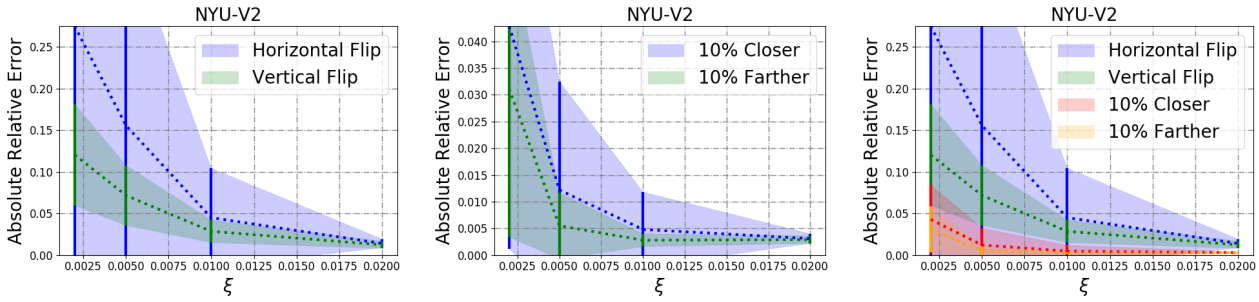
Figure 8.12: **Adversarial training vs Gaussian blur.** Absolute relative error (ARE) achieved by adversarial perturbations of different norms (ξ) for different scales. For each scale, we plot the ARE without any defense, with Gaussian blur and with adversarial training. Both Gaussian blur and adversarial training makes the depth prediction network more robust to perturbations. Performances of the two defense mechanisms are comparable for small norms (ξ), but adversarial training is more effective on larger norms.

monocular depth prediction method for NYU-V2, trained in the supervised setting.

8.8.1 Implementation Details

NYU-V2 consists of 1449 RGBD images gathered from a wide range of buildings, comprising 464 different indoor scenes across 26 scene classes. The images were hand-selected from 435,103 video frames, to ensure diversity. 1449 labeled samples are split into 795 training and 654 test images.

In the method proposed by [YLS19], a 3D point cloud is reconstructed from the estimated depth map. Then, three non-colinear points are randomly sampled with large distances to form a virtual plane. The deviation between ground truth and prediction for the direction of the normal



(a) (b) (c)

Figure 8.13: **Indoor quantitative results.** ARE with various upper norm ξ for scaling and flipping VNL predictions. (a) Results for horizontally and vertically flipping the predictions. (b) Results for scaling the scene by $\pm 10\%$. (c) comparison between scaling and flipping tasks.

vector corresponding to the plane is penalized. The pre-trained ResNeXt-101 [XGD17] model on ImageNet [DDS09] is used as the backbone architecture. During training, images are cropped to the size 384×384 for NYU-V2. We use the same image resolution for our experiments. The training set is randomly sampled from 29K images of the raw unlabeled training set.

The time it takes to forward an image with this method is ≈ 0.15 seconds (≈ 7 times more than Monodepth2). Due to computational limitations, we choose the first 20 images out of 654 images of the test split for our experiments. We run SGD for 1000 steps. The learning rate is kept at 10.0 for the entire optimization.

8.8.2 Scaling and Symmetrically Flipping the Scene

In Fig. 8.13, we compare the performance for different target depth maps: scaling the scene by $\pm 10\%$, horizontal and vertical flipping. Unlike outdoor case (KITTI), in the indoor (NYU-V2) horizontal flipping is a harder task than vertical flipping. Achieving a vertically flipped scene was expected to be simpler as layouts of indoor scenes are more diverse. Hence, the depth network does not overfit to a particular layout type e.g. the one in which there are large depth values only at the top of the image. Horizontal flipping being relatively harder for indoor scenes can be explained by the large divergence between the depth distributions of the original predictions and the target depths. The reason is that for the indoor scenes, most scenes are not symmetric in the horizontal direction,

unlike the outdoor driving scenario where the left and right parts of the scene from the ego-view are usually symmetric.

Since [YLS19] normalizes images with the deviation of the dataset which approximately scales the image by 5, it also effectively scales the noise with the same deviation. But, since the relative norm is still the same, we use the same norm values $\xi \in \{2 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}, 2 \times 10^{-2}\}$ when plotting the ARE in Fig. 8.13.

In Fig. 8.14, we present qualitative results for NYU-V2 for $\xi = 2 \times 10^{-2}$. Small, white borders around RGB images exist in the raw dataset. For all the tasks, including vertical flip, adversarial perturbations manage to fool the model to predict the target depth with small errors. For horizontally flipped target depth (a), predictions have more artifacts than vertical flipped depth (b) and scaled depths (c,d).

8.9 Linear Operations

To better understand how predictions of the depth network changes within a ball of small radius, we examine the effect of linear operations on perturbations. Specifically, we visualize the predictions for the scaled perturbations and for the perturbations which we get after summing two perturbations calculated for two different target depth maps.

In Fig. 8.15, we take the perturbation $v(x)$, which we calculated to horizontally flip the prediction for the given image, and we visualize the prediction of the network for $x + \gamma v(x)$ where $\gamma \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$. As can be seen, between $\gamma = 0$ and $\gamma = 1$, the scene is smoothly flipped. This implies that the adversarial perturbations can be used to control the depth prediction in a disentangled way. In other words, one causal factor (e.g. horizontal orientation) of the prediction can be independently controlled by tweaking γ only, keeping everything else the same.

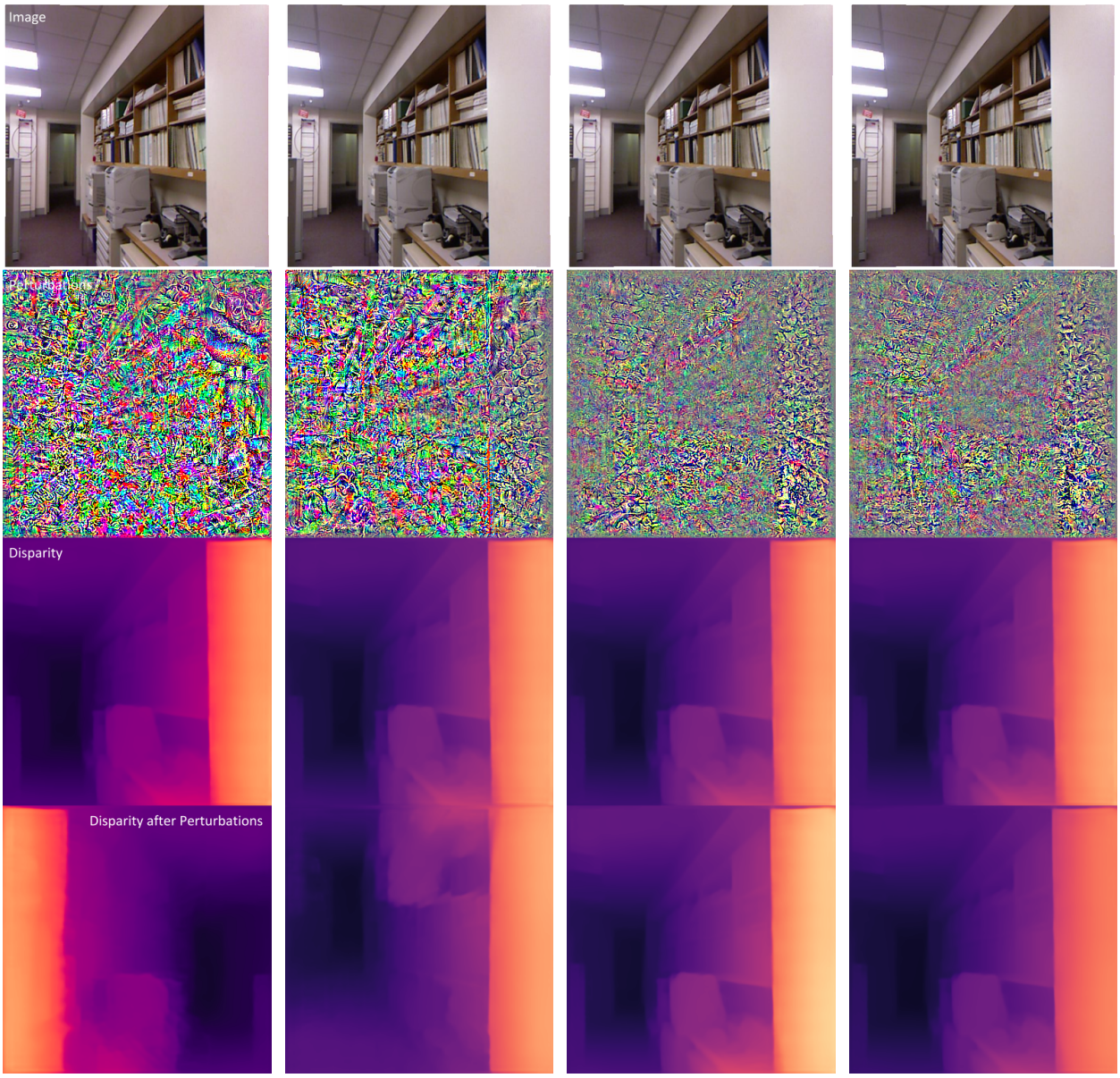
As observed before, noise is small for the white regions. See the third column, where there is a gray rectangle in the noise corresponding to the white region of the truck. We speculate the reason behind this phenomenon as the white color being on the border of the support of RGB images. But, the noise is still large for black regions which are at the other extreme of the support

(see perturbations corresponding to black vehicles). So, we left further understanding of this phenomenon as future work.

In Fig. 8.16, we take $v_1(x)$ and $v_2(x)$ which are optimized to scale the scene to 10% closer and 10% farther. Then, visualize the summed perturbation, $v(x) = v_1(x) + v_2(x)$ and the prediction for $x + v(x)$. As can be seen, two noises cancel each other: $||v_1(x)|| \approx ||v_2(x)|| \gg ||v_1(x) + v_2(x)||$. Furthermore, the prediction for the image perturbed with the summed noise is close to the original prediction: $f_d(x) \approx f_d(x + v_1(x) + v_2(x))$. This shows that two perturbations with inverse functionalities can neutralize their effects when applied together. For more details, see [WCS20].

8.10 Conclusion

Depth prediction networks are indeed vulnerable to adversarial perturbations. Not only can such perturbations alter the perception of the scene, but can also affect specific instances, making the network behave unpredictably, which can be catastrophic in applications that involve interaction with physical space. These perturbations also shed light on the network's dependency on the non-local context for local predictions, making non-local targeted attacks possible. While we have exposed vulnerabilities, we hope that our findings on the network's biases, the effect of context, and robustness of the perturbations can help design more secure and interpretable models that are not susceptible to such attacks.



(a) (b) (c) (d)

Figure 8.14: **Indoor qualitative results.** From left to right: (a) horizontal flip, (b) vertical flip, (c) scale 10% closer and (d) scale 10% farther. From top to bottom: RGB, noise, original disparity, disparity prediction for the perturbed image.

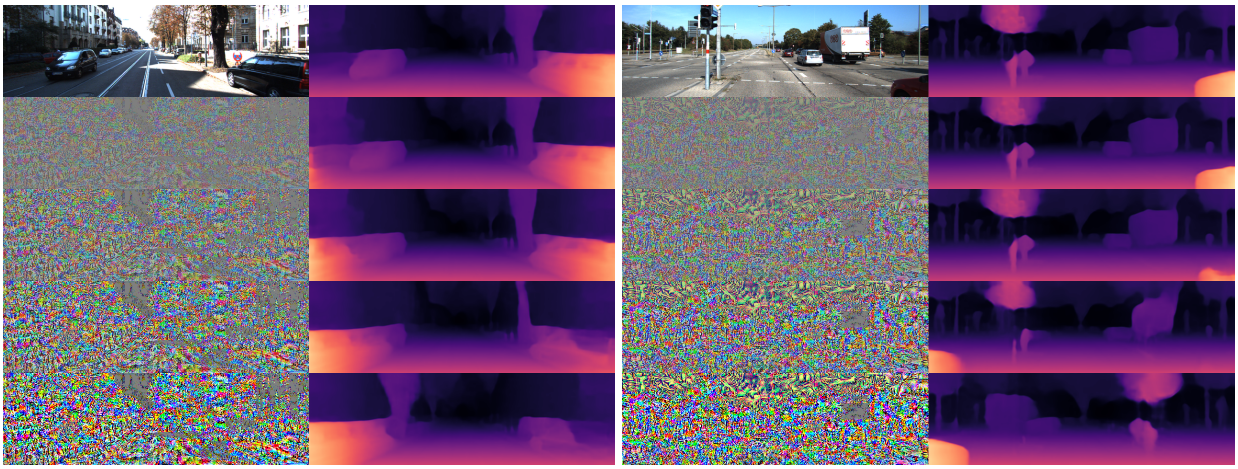


Figure 8.15: Disparity for $x + \gamma v(x)$ where γ is $0.0, 0.25, 0.5, 0.75, 1.0$ from top to bottom. $v(x)$ is calculated for $d^t = \texttt{fliph}(f_d(x))$. So, the top is the original disparity map while bottom most is the flipped one. In between, portions of the scene are flipped smoothly.

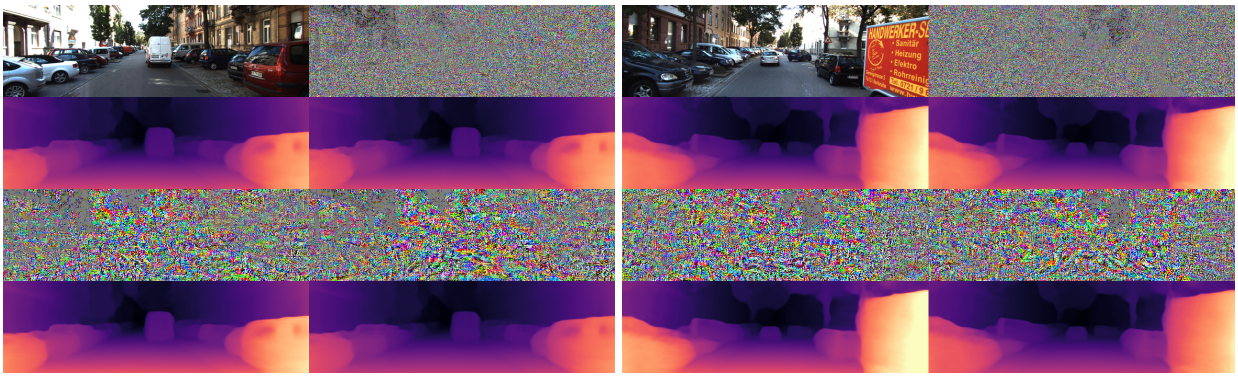


Figure 8.16: (1st row) left to right: RGB, sum of noises. (2nd row) left to right: original disparity, disparity when two noises $v_1(x) + v_2(x)$ are added to the image. (3rd row) left to right: noise for 10% closer, noise for 10% farther. (4th row): Disparity predictions for the images perturbed with the noises in the 3rd row. When added, two noises cancel each other's effect: the scale for $f_d(x + v_1(x) + v_2(x))$ is close to the original one $f_d(x)$.

CHAPTER 9

Discussion

In this thesis, we detailed several approaches for developing visioning systems for an agent to *understand* the scene that it is interacting with. For this, we examined image classification, segmentation, and depth prediction tasks in different unsupervised learning settings (SSL, UDA) to develop labeled-data efficient algorithms.

In Chapter 2, we leveraged the training speed as a proxy to measure the quality of putative labels. We demonstrated that compared to "random" labels, training on "accurate" labels is faster in expectation. Of course, this does not imply that any label set which would lead to fast training is accurate. An obvious example is predicting the same "constant" label for all samples. For instance, predicting all image labels to be "dog" on which over-fitting would be immediate. So, we need to have more regularizers to further constrain the solution space which we achieved by using regularizers described in Sec. 2.3. Moreover, SaaS involves two nested loops: the first one is to evaluate how fast the training is on the current pseudo-labels and the second loop is to update those pseudo-labels. So, overall optimization is slow for large-scale datasets. In Chapter 3, we extend our work on SSL by acting on both input and weight spaces where we merge the literature branched off into two different groups. We propose an adversarial training algorithm for SSL tasks to be robust against adversarial perturbations in both input and weight spaces where we combine the known input smoothing algorithm with a novel weight smoothing algorithm. However, both SaaS and ABCD require labeled samples for each class in the real dataset, which is not always feasible.

Conscious of this drawback, in Chapter 4, we switched to a more challenging UDA setting where we do not have any labeled samples from the target domain. We proposed a novel method for UDA with the motivation of conditionally aligning the features. While the quality of these

UDA classifiers can be measured by standard classification metrics, these yield no insight on how the classifier manages nuisance variability due to the domain (covariate) shift. Is the classifier agnostic, or invariant, to domain changes? Where is such invariance achieved in the model? Does the response to covariate shift generalize across multiple domains, or do we need to train UDA classifiers for every pair of source and target domain? Does the model "separate" in some sense class and domain representations? In Chapter 5, we devise a more interpretable model, where we can pinpoint the effects of changes in the domain and class of the input image within the model.

All the aforementioned methods are limited to the image classification task where only one scalar attribute per image is estimated. However, we want to be able to extract much more information given an image. Even though we achieved promising performance on the classification benchmarks with the previously proposed conditional domain alignment methods, these results heavily rely on the pseudo-labels. The pseudo-labels are generated by networks regularized with SSL losses which are needed to be tuned with care. This is why a method working well for the classification task does not perform well for the segmentation task. Moreover, these two tasks are inherently different. For instance, for the classification task, rotation, orientation, translations are all nuisances for the task whereas for the segmentation, we want to preserve the spatial information as much as possible. So, in Chapter 6, we proposed a simple yet, effective solution to the spatial-class distribution shift problem for the image segmentation task. We proved its effectiveness by performing at the state-of-the-art in UDA segmentation benchmarks. We believe there are still many open rooms to encode known priors about the physical world into our learning systems so that they do not have to re-learn the known regularities of our visual world.

In Chapter 7, we switched to the depth estimation problem which is a key task for autonomy. We first learn to predict an approximate topology from sparse points with a lightweight network that we call ScaffNet. However, the topology estimated by ScaffNet is only a "guess" and therefore must be reconciled with the image via FusionNet (e.g. regions with very few or no sparse points, where estimates from ScaffNet are less reliable). Hence, by leveraging the topology as a prior and learning the residual over it, we allow FusionNet the freedom to amend the scene as needed. However, we did not consider the case where the shapes in synthetic data are not representative of those in the

real data. Our findings demonstrate the benefits of leveraging synthetic data for learning topology from sparse points and motivates further exploration to incorporate the virtually unlimited amount of synthetic data into multi-sensor fusion pipelines for the 3D reconstruction task.

In Chapter 8, we demonstrated the vulnerability of unsupervised depth prediction networks to small additive perturbations following the works showing the same vulnerability for the classification and the segmentation models. Adversarial perturbations highlight limitations and failure modes of deep networks. They have captured the collective imagination by conjuring scenarios where AI goes awry at the tune of imperceptible changes. Some popular media and press has gone insofar as suggesting them as proof that AI cannot be trusted. While monocular depth prediction networks are indeed vulnerable to these attacks, we want to assure that these perturbations cannot cause harm in a practical transportation application. Optimizing for these perturbations is computationally expensive and hence it is infeasible to craft these perturbations in real-time. Additionally, they also do not transfer; so, we see little negative implications for real-world applications. However, the fact that they exist implies that there is room for improvement in the way that we learn representations, especially for unsupervised learning schemes. Hence, we see the existence of adversaries as an opportunity. Studying their effects on deep networks is also the first step to render a system robust to such a vulnerability.

REFERENCES

[AAL19] Amir Atapour-Abarghouei, Samet Akcay, Grégoire Payen de La Garanderie, and Toby P Breckon. "Generative adversarial framework for depth filling via Wasserstein metric, cosine transform and domain transfer." *Pattern Recognition*, **91**:232–244, 2019.

[AB18] Amir Atapour-Abarghouei and Toby P Breckon. "Real-time monocular depth estimation using synthetic data with domain adaptation via image style transfer." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2800–2810, 2018.

[AMF11] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. "Contour detection and hierarchical image segmentation." *IEEE transactions on pattern analysis and machine intelligence*, **33**(5):898–916, 2011.

[AMM18] Hassan Alhaija, Siva Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. "Augmented Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes." *International Journal of Computer Vision (IJCV)*, 2018.

[AS17] Alessandro Achille and Stefano Soatto. "On the emergence of invariance and disentangling in deep representations." *arXiv preprint arXiv:1706.01350*, 2017.

[BBC10] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. "A theory of learning from different domains." *Machine learning*, **79**(1-2):151–175, 2010.

[BBC16] Carlo Baldassi, Christian Borgs, Jennifer T Chayes, Alessandro Ingrosso, Carlo Lucibello, Luca Saglietti, and Riccardo Zecchina. "Unreasonable effectiveness of learning neural networks: From accessible states and robust ensembles to basic algorithmic schemes." *Proceedings of the National Academy of Sciences*, **113**(48):E7655–E7662, 2016.

[BM98] Avrim Blum and Tom Mitchell. "Combining labeled and unlabeled data with co-training." In *Proceedings of the eleventh annual conference on Computational learning theory*, pp. 92–100. ACM, 1998.

[BNT10] Dimitris Bertsimas, Omid Nohadani, and Kwong Meng Teo. "Robust optimization for unconstrained simulation-based problems." *Operations research*, **58**(1):161–178, 2010.

[BSD17] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. "Unsupervised pixel-level domain adaptation with generative adversarial networks." In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, p. 7, 2017.

[BSG18] M Baity-Jesi, L Sagun, M Geiger, S Spigler, G Ben Arous, C Cammarota, Y LeCun, M Wyart, and G Biroli. "Comparing Dynamics: Deep Neural Networks versus Glassy Systems." *arXiv preprint arXiv:1803.06969*, 2018.

[CC18] Jia-Ren Chang and Yong-Sheng Chen. "Pyramid stereo matching network." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5410–5418, 2018.

[CCC17] Yi-Hsin Chen, Wei-Yu Chen, Yu-Ting Chen, Bo-Cheng Tsai, Yu-Chiang Frank Wang, and Min Sun. "No more discrimination: Cross city adaptation of road scene segmenters." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1992–2001, 2017.

[CCS16] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, and Yann LeCun. "Entropy-sgd: Biasing gradient descent into wide valleys." *arXiv preprint arXiv:1611.01838*, 2016.

[CFS18a] Safa Cicek, Alhussein Fawzi, and Stefano Soatto. "SaaS: Speed as a Supervisor for Semi-supervised Learning." In *The European Conference on Computer Vision (ECCV)*, September 2018.

[CFS18b] Safa Cicek, Alhussein Fawzi, and Stefano Soatto. "SaaS: Speed as a Supervisor for Semi-supervised Learning." In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 149–163, 2018.

[CLV18] Yuhua Chen, Wen Li, and Luc Van Gool. "Road: Reality oriented adaptation for semantic segmentation of urban scenes." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7892–7901, 2018.

[CNL11] Adam Coates, Andrew Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning." In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, 2011.

[COR16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The cityscapes dataset for semantic urban scene understanding." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016.

[CPK17] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs." *IEEE transactions on pattern analysis and machine intelligence*, **40**(4):834–848, 2017.

[CPM19] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. "Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos." In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 8001–8008, 2019.

[CPS17] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. "Rethinking atrous convolution for semantic image segmentation." *arXiv preprint arXiv:1706.05587*, 2017.

[CS19a] Safa Cicek and Stefano Soatto. "Input and Weight Space Smoothing for Semi-supervised Learning." In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[CS19b] Safa Cicek and Stefano Soatto. "Unsupervised domain adaptation via regularized conditional alignment." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1416–1425, 2019.

[CSZ09] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]." *IEEE Transactions on Neural Networks*, **20**(3):542–542, 2009.

[CWG19] Xinjing Cheng, Peng Wang, Chenye Guan, and Ruigang Yang. "CSPN++: Learning Context and Resource Aware Convolutional Spatial Propagation Networks for Depth Completion." *arXiv preprint arXiv:1911.05377*, 2019.

[CWL18] Nathaniel Chodosh, Chaoyang Wang, and Simon Lucey. "Deep Convolutional Compressed Sensing for LiDAR Depth Completion." In *Asian Conference on Computer Vision (ACCV)*, 2018.

[CXW20a] Safa Cicek, Ning Xu, Zhaowen Wang, Hailin Jin, and Stefano Soatto. "Disentangled Image Generation for Unsupervised Domain Adaptation." In *European Conference on Computer Vision*, pp. 662–665. Springer, 2020.

[CXW20b] Safa Cicek, Ning Xu, Zhaowen Wang, Hailin Jin, and Stefano Soatto. "Spatial Class Distribution Shift in Unsupervised Domain Adaptation: Local Alignment Comes to Rescue." In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[CYL19] Yun Chen, Bin Yang, Ming Liang, and Raquel Urtasun. "Learning Joint 2D-3D Representations for Depth Completion." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 10023–10032, 2019.

[CZP18] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-decoder with atrous separable convolution for semantic image segmentation." In *Proceedings of the European conference on computer vision (ECCV)*, pp. 801–818, 2018.

[DDS09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

[DDV82] Antonio R Damasio, Hanna Damasio, and Gary W Van Hoesen. "Prosopagnosia: anatomic basis and behavioral mechanisms." *Neurology*, **32**(4):331–331, 1982.

[DPB17] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. "Sharp minima can generalize for deep nets." *arXiv preprint arXiv:1703.04933*, 2017.

[DVP18] Martin Dimitrievski, Peter Veelaert, and Wilfried Philips. "Learning morphological operators for depth completion." In *Advanced Concepts for Intelligent Vision Systems*, 2018.

[DYY17] Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Ruslan R Salakhutdinov. "Good semi-supervised learning that requires a bad gan." In *Advances in Neural Information Processing Systems*, pp. 6513–6523, 2017.

[EEF18] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. "Robust physical-world attacks on deep learning visual classification." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1625–1634, 2018.

[EF15] David Eigen and Rob Fergus. "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2650–2658, 2015.

[EFK18] Abdelrahman Eldesokey, Michael Felsberg, and Fahad Shahbaz Khan. "Propagating confidences through cnns for sparse data regression." In *Proceedings of British Machine Vision Conference (BMVC)*, 2018.

[EPF14] David Eigen, Christian Puhrsch, and Rob Fergus. "Depth map prediction from a single image using a multi-scale deep network." In *Advances in neural information processing systems*, pp. 2366–2374, 2014.

[FB81] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography." *Communications of the ACM*, **24**(6), 1981.

[FB16] C Daniel Freeman and Joan Bruna. "Topology and geometry of half-rectified network optimization." *arXiv preprint arXiv:1611.01540*, 2016.

[FFF18] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. "Analysis of classifiers' robustness to adversarial perturbations." *Machine Learning*, **107**(3):481–508, 2018.

[FMF18] Geoff French, Michal Mackiewicz, and Mark Fisher. "Self-ensembling for visual domain adaptation." 2018.

[FWS18] Xiaohan Fei, Alex Wong, and Stefano Soatto. "Geo-Supervised Visual Depth Prediction." *arXiv preprint arXiv:1807.11130*, 2018.

[FWS19] Xiaohan Fei, Alex Wong, and Stefano Soatto. "Geo-supervised visual depth prediction." *IEEE Robotics and Automation Letters*, **4**(2):1661–1668, 2019.

[GB05] Yves Grandvalet and Yoshua Bengio. "Semi-supervised learning by entropy minimization." In *Advances in neural information processing systems*, pp. 529–536, 2005.

[GBC16] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. "Unsupervised cnn for single view depth estimation: Geometry to the rescue." In *European Conference on Computer Vision*, pp. 740–756. Springer, 2016.

[GKZ16] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. "Deep reconstruction-classification networks for unsupervised domain adaptation." In *European Conference on Computer Vision*, pp. 597–613. Springer, 2016.

[GL14] Yaroslav Ganin and Victor Lempitsky. "Unsupervised domain adaptation by backpropagation." *arXiv preprint arXiv:1409.7495*, 2014.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? the kitti vision benchmark suite." In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361. IEEE, 2012.

[GMB17] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. "Unsupervised monocular depth estimation with left-right consistency." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 270–279, 2017.

[GMF19] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. "Digging into self-supervised monocular depth estimation." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3828–3838, 2019.

[GPM14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572*, 2014.

[GTB18] Alexander Gaunt, Danny Tarlow, Marc Brockschmidt, Raquel Urtasun, Renjie Liao, and Richard Zemel. "Graph Partition Neural Networks for Semi-Supervised Classification." 2018.

[GVS14] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. "Qualitatively characterizing neural network optimization problems." *arXiv preprint arXiv:1412.6544*, 2014.

[GWB18] Abel Gonzalez-Garcia, Joost van de Weijer, and Yoshua Bengio. "Image-to-image translation for cross-domain disentanglement." In *Advances in Neural Information Processing Systems*, pp. 1287–1298, 2018.

[GWC16] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. "Virtual worlds as proxy for multi-object tracking analysis." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4340–4349, 2016.

[HB17] Xun Huang and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1501–1510, 2017.

[HCB17] Jan Hendrik Metzen, Mummadi Chaithanya Kumar, Thomas Brox, and Volker Fischer. "Universal adversarial perturbations against semantic image segmentation." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2755–2764, 2017.

[HFC19] Zixuan Huang, Junming Fan, Shenggan Cheng, Shuai Yi, Xiaogang Wang, and Hongsheng Li. "Hms-net: Hierarchical multi-scale sparsity-invariant network for sparse depth completion." *IEEE Transactions on Image Processing*, 2019.

[HLB18] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. "Multimodal unsupervised image-to-image translation." In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 172–189, 2018.

[HMC17] Philip Haeusser, Alexander Mordvintsev, and Daniel Cremers. "Learning by association-a versatile semi-supervised training method for neural networks." In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[HRS16] Moritz Hardt, Ben Recht, and Yoram Singer. "Train faster, generalize better: Stability of stochastic gradient descent." In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1225–1234, 2016.

[HS88] Christopher G Harris, Mike Stephens, et al. "A combined corner and edge detector." In *Alvey vision conference*, volume 15, pp. 10–5244. Citeseer, 1988.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Flat minima." *Neural Computation*, **9**(1):1–42, 1997.

[HSK12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580*, 2012.

[HTP17] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A Efros, and Trevor Darrell. "Cycada: Cycle-consistent adversarial domain adaptation." *arXiv preprint arXiv:1711.03213*, 2017.

[HW18] Yedid Hoshen and Lior Wolf. "Nam: Non-adversarial unsupervised domain mapping." In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 436–451, 2018.

[HWY16] Judy Hoffman, Dequan Wang, Fisher Yu, and Trevor Darrell. "Fcns in the wild: Pixel-level adversarial and constraint-based adaptation." *arXiv preprint arXiv:1612.02649*, 2016.

[HZO19] Junjie Hu, Yan Zhang, and Takayuki Okatani. "Visualization of convolutional neural networks for monocular depth estimation." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3869–3878, 2019.

[HZR15a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[HZR15b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Spatial pyramid pooling in deep convolutional networks for visual recognition." *IEEE transactions on pattern analysis and machine intelligence*, **37**(9):1904–1916, 2015.

[HZR16a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[HZR16b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Identity mappings in deep residual networks." In *European conference on computer vision*, pp. 630–645. Springer, 2016.

[IZZ17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. "Image-to-image translation with conditional adversarial networks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[JCW18] Maximilian Jaritz, Raoul de Charette, Emilie Wirbel, Xavier Perrotton, and Fawzi Nashashibi. "Sparse and Dense Data with CNNs: Depth Completion and Semantic Segmentation." In *International Conference on 3D Vision (3DV)*, 2018.

[JKA17] Stanislaw Jastrzebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. "Three Factors Influencing Minima in SGD." *arXiv preprint arXiv:1711.04623*, 2017.

[Joa99] Thorsten Joachims. "Transductive inference for text classification using support vector machines." In *ICML*, volume 99, pp. 200–209, 1999.

[KAL17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. "Progressive growing of gans for improved quality, stability, and variation." *arXiv preprint arXiv:1710.10196*, 2017.

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*, 2014.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[KH09] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images." 2009.

[KK13] Philipp Krähenbühl and Vladlen Koltun. "Parameter learning and convergent inference for dense random fields." In *International Conference on Machine Learning*, pp. 513–521, 2013.

[KLA19] Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410, 2019.

[KMN16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On large-batch training for deep learning: Generalization gap and sharp minima." *arXiv preprint arXiv:1609.04836*, 2016.

[KPG10] Andreas Krause, Pietro Perona, and Ryan G Gomes. "Discriminative clustering by regularized information maximization." In *Advances in neural information processing systems*, pp. 775–783, 2010.

[KSW18] Abhishek Kumar, Prasanna Sattigeri, Kahini Wadhawan, Leonid Karlinsky, Rogerio Feris, Bill Freeman, and Gregory Wornell. "Co-regularized Alignment for Unsupervised Domain Adaptation." In *Advances in Neural Information Processing Systems*, pp. 9366–9377, 2018.

[KW16] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907*, 2016.

[LA16] Samuli Laine and Timo Aila. "Temporal Ensembling for Semi-Supervised Learning." *arXiv preprint arXiv:1610.02242*, 2016.

[LBB98] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, **86**(11):2278–2324, 1998.

[LBK17] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. "Unsupervised image-to-image translation networks." In *Advances in neural information processing systems*, pp. 700–708, 2017.

[LCW15] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I Jordan. "Learning transferable features with deep adaptation networks." *arXiv preprint arXiv:1502.02791*, 2015.

[LCW18] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. "Conditional adversarial domain adaptation." In *Advances in Neural Information Processing Systems*, pp. 1647–1657, 2018.

[LFY17] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. "Universal style transfer via feature transforms." In *Advances in neural information processing systems*, pp. 386–396, 2017.

[LH16] Ilya Loshchilov and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts." *arXiv preprint arXiv:1608.03983*, 2016.

[LHM19] Ming-Yu Liu, Xun Huang, Arun Mallya, Tero Karras, Timo Aila, Jaakko Lehtinen, and Jan Kautz. "Few-shot unsupervised image-to-image translation." *arXiv preprint arXiv:1905.01723*, 2019.

[LLD19] Qing Lian, Fengmao Lv, Lixin Duan, and Boqing Gong. "Constructing Self-motivated Pyramid Curriculums for Cross-Domain Semantic Segmentation: A Non-Adversarial Approach." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 6758–6767, 2019.

[LLK18] Xueting Li, Sifei Liu, Jan Kautz, and Ming-Hsuan Yang. "Learning linear transformations for fast arbitrary style transfer." *arXiv preprint arXiv:1808.04537*, 2018.

[LLU16] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. "Understanding the effective receptive field in deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 4898–4906, 2016.

[LLW15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. "Deep learning face attributes in the wild." In *Proceedings of the IEEE international conference on computer vision*, pp. 3730–3738, 2015.

[LLY18] Alexander H Liu, Yen-Cheng Liu, Yu-Ying Yeh, and Yu-Chiang Frank Wang. "A unified feature disentangler for multi-domain image translation and manipulation." In *Advances in Neural Information Processing Systems*, pp. 2590–2599, 2018.

[LMB14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context." In *European conference on computer vision*, pp. 740–755. Springer, 2014.

[LMF09] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. "Epnp: An accurate o (n) solution to the pnp problem." *International journal of computer vision*, **81**(2):155, 2009.

[LRB16] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. "Deeper depth prediction with fully convolutional residual networks." In *2016 Fourth international conference on 3D vision (3DV)*, pp. 239–248. IEEE, 2016.

[LT16] Ming-Yu Liu and Oncel Tuzel. "Coupled generative adversarial networks." In *Advances in neural information processing systems*, pp. 469–477, 2016.

[LWS16] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. "Revisiting batch normalization for practical domain adaptation." *arXiv preprint arXiv:1603.04779*, 2016.

[LXT17] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. "Visualizing the Loss Landscape of Neural Nets." *arXiv preprint arXiv:1712.09913*, 2017.

[LYF18] Yen-Cheng Liu, Yu-Ying Yeh, Tzu-Chien Fu, Sheng-De Wang, Wei-Chen Chiu, and Yu-Chiang Frank Wang. "Detach and adapt: Learning cross-domain disentangled deep representation." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8867–8876, 2018.

[LYW18] Chenxu Luo, Zhenheng Yang, Peng Wang, Yang Wang, Wei Xu, Ram Nevatia, and Alan Yuille. "Every pixel counts++: Joint learning of geometry and motion with 3d holistic understanding." *arXiv preprint arXiv:1810.06125*, 2018.

[LZC14] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. "Efficient mini-batch training for stochastic optimization." In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 661–670. ACM, 2014.

[LZG19] Yawei Luo, Liang Zheng, Tao Guan, Junqing Yu, and Yi Yang. "Taking a closer look at domain shift: Category-level adversaries for semantics consistent domain adaptation." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2507–2516, 2019.

[LZW16] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. "Unsupervised domain adaptation with residual transfer networks." In *Advances in Neural Information Processing Systems*, pp. 136–144, 2016.

[MCK19] Fangchang Ma, Guilherme Venturelli Cavalheiro, and Sertac Karaman. "Self-supervised Sparse-to-Dense: Self-supervised Depth Completion from LiDAR and Monocular Camera." In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

[MFF16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. "Deepfool: a simple and accurate method to fool deep neural networks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, 2016.

[MFF17] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. "Universal adversarial perturbations." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1765–1773, 2017.

[MGB18] Konda Reddy Mopuri, Aditya Ganeshan, and R Venkatesh Babu. "Generalizable data-free objective for crafting universal adversarial perturbations." *IEEE transactions on pattern analysis and machine intelligence*, **41**(10):2452–2465, 2018.

[MGN18] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. "Which training methods for GANs do actually converge?" *arXiv preprint arXiv:1801.04406*, 2018.

[MH08] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." *Journal of machine learning research*, **9**(Nov):2579–2605, 2008.

[MHG18] Moritz Menze, Christian Heipke, and Andreas Geiger. "Object Scene Flow." *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 2018.

[MHL16] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J Davison. "SceneNet RGB-D: 5M photorealistic images of synthetic indoor trajectories with ground truth." *arXiv preprint arXiv:1612.05079*, 2016.

[MIH16] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4040–4048, 2016.

[MJG18] Liqian Ma, Xu Jia, Stamatios Georgoulis, Tinne Tuytelaars, and Luc Van Gool. "Exemplar guided unsupervised image-to-image translation with semantic consistency." *arXiv preprint arXiv:1805.11145*, 2018.

[MMK15] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. "Distributional smoothing with virtual adversarial training." *arXiv preprint arXiv:1507.00677*, 2015.

[MMK17] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. "Virtual Adversarial Training: a Regularization Method for Supervised and Semi-supervised Learning." *arXiv preprint arXiv:1704.03976*, 2017.

[MMK18] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. "Virtual adversarial training: a regularization method for supervised and semi-supervised learning." *IEEE transactions on pattern analysis and machine intelligence*, **41**(8):1979–1993, 2018.

[MO14] Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets." *arXiv preprint arXiv:1411.1784*, 2014.

[MSK12] Yi Ma, Stefano Soatto, Jana Kosecka, and S Shankar Sastry. *An invitation to 3-d vision: from images to geometric models*, volume 26. Springer Science & Business Media, 2012.

[MVJ11] Jesús P Mena-Chalco, Luiz Velho, and RM Cesar Junior. "3D human face reconstruction using principal components spaces." In *Proceedings of WTD SIBGRAPI Conference on Graphics, Patterns and Images*, pp. 1–6, 2011.

[MWA18] Reza Mahjourian, Martin Wicke, and Anelia Angelova. "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5667–5675, 2018.

[NKP18] Jogendra Nath Kundu, Phani Krishna Uppala, Anuj Pahuja, and R Venkatesh Babu. "Adadepth: Unsupervised content congruent adaptation for depth estimation." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2656–2665, 2018.

[NWC11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. "Reading digits in natural images with unsupervised feature learning." In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, p. 5, 2011.

[NWH11] Feiping Nie, Hua Wang, Heng Huang, and Chris Ding. "Unsupervised and semi-supervised learning via 1-norm graph." In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2268–2273. IEEE, 2011.

[OOS17] Augustus Odena, Christopher Olah, and Jonathon Shlens. "Conditional image synthesis with auxiliary classifier gans." In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2642–2651. JMLR. org, 2017.

[PAG19] Sudeep Pillai, Rareş Ambruş, and Adrien Gaidon. "Superdepth: Self-supervised, super-resolved monocular depth estimation." In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9250–9256. IEEE, 2019.

[PBX18] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. "Moment Matching for Multi-Source Domain Adaptation." *arXiv preprint arXiv:1812.01754*, 2018.

[PGC17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. "Automatic differentiation in PyTorch." 2017.

[PLW19] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. "Semantic image synthesis with spatially-adaptive normalization." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2337–2346, 2019.

[PPS17] Sungrae Park, Jun-Keon Park, Su-Jin Shin, and Il-Chul Moon. "Adversarial Dropout for Supervised and Semi-supervised Learning." *arXiv preprint arXiv:1707.03631*, 2017.

[PTC17] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. "Regularizing neural networks by penalizing confident output distributions." *arXiv preprint arXiv:1701.06548*, 2017.

[PTM18] Matteo Poggi, Fabio Tosi, and Stefano Mattoccia. "Learning monocular depth estimation with unsupervised trinocular assumptions." In *2018 International Conference on 3D Vision (3DV)*, pp. 324–333. IEEE, 2018.

[QCZ19] Jiaxiong Qiu, Zhaopeng Cui, Yinda Zhang, Xingdi Zhang, Shuaicheng Liu, Bing Zeng, and Marc Pollefeys. "Deeplidar: Deep surface normal guided depth prediction for outdoor scene from sparse lidar data and single color image." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3313–3322, 2019.

[RCT18] Paolo Russo, Fabio M Carlucci, Tatiana Tommasi, and Barbara Caputo. "From source to target and back: symmetric bi-directional adaptive gan." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8099–8108, 2018.

[RDS15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge." *International journal of computer vision*, **115**(3):211–252, 2015.

[RRT17] Maxim Raginsky, Alexander Rakhlin, and Matus Telgarsky. "Non-convex learning via Stochastic Gradient Langevin Dynamics: a nonasymptotic analysis." In *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, pp. 1674–1703, 2017.

[RSM16] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3234–3243, 2016.

[RVR16a] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. "Playing for Data: Ground Truth from Computer Games." In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pp. 102–118. Springer International Publishing, 2016.

[RVR16b] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. "Playing for data: Ground truth from computer games." In *European Conference on Computer Vision*, pp. 102–118. Springer, 2016.

[SBN18] Rui Shu, Hung H Bui, Hirokazu Narui, and Stefano Ermon. "A DIRT-T Approach to Unsupervised Domain Adaptation." *arXiv preprint arXiv:1802.08735*, 2018.

[SC14] Stefano Soatto and Alessandro Chiuso. "Visual representations: Defining properties and deep approximations." *arXiv preprint arXiv:1411.7676*, 2014.

[SC16] Daniel Soudry and Yair Carmon. "No bad local minima: Data independent training error guarantees for multilayer neural networks." *arXiv preprint arXiv:1605.08361*, 2016.

[SEG17] Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. "Empirical Analysis of the Hessian of Over-Parametrized Neural Networks." *arXiv preprint arXiv:1706.04454*, 2017.

[SFS16] Baochen Sun, Jiashi Feng, and Kate Saenko. "Return of frustratingly easy domain adaptation." In *AAAI*, volume 6, p. 8, 2016.

[SGA14] Levent Sagun, V Ugur Guney, Gerard Ben Arous, and Yann LeCun. "Explorations on high dimensional landscapes." *arXiv preprint arXiv:1412.6615*, 2014.

[SGZ16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved techniques for training gans." In *Advances in Neural Information Processing Systems*, pp. 2234–2242, 2016.

[SHK12] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. "Indoor segmentation and support inference from rgbd images." In *European conference on computer vision*, pp. 746–760. Springer, 2012.

[SHK14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research*, **15**(1):1929–1958, 2014.

[SJT16a] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. "Mutual exclusivity loss for semi-supervised deep learning." In *Image Processing (ICIP), 2016 IEEE International Conference on*, pp. 1908–1912. IEEE, 2016.

[SJT16b] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. "Regularization with stochastic transformations and perturbations for deep semi-supervised learning." In *Advances in Neural Information Processing Systems*, pp. 1163–1171, 2016.

[SNM19] Shreyas S Shivakumar, Ty Nguyen, Ian D Miller, Steven W Chen, Vijay Kumar, and Camillo J Taylor. "Dfusenet: Deep fusion of rgb and sparse depth information for image guided dense depth completion." In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 13–20. IEEE, 2019.

[Soa13] Stefano Soatto. "Actionable information in vision." In *Machine learning for computer vision*, pp. 17–48. Springer, 2013.

[Spr15] Jost Tobias Springenberg. "Unsupervised and semi-supervised learning with categorical generative adversarial networks." *arXiv preprint arXiv:1511.06390*, 2015.

[SQZ17] Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. "Adversarial representation learning for domain adaptation." *arXiv preprint arXiv:1707.01217*, 2017.

[SRG14] Justin Solomon, Raif Rustamov, Leonidas Guibas, and Adrian Butscher. "Wasserstein propagation for semi-supervised learning." In *International Conference on Machine Learning*, pp. 306–314, 2014.

[SS16] Itay Safran and Ohad Shamir. "On the quality of the initial basin in overspecified neural networks." In *International Conference on Machine Learning*, pp. 774–782, 2016.

[SSG12] Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. "Constrained semi-supervised learning using attributes and comparative attributes." In *European Conference on Computer Vision*, pp. 369–383. Springer, 2012.

[SSP16] Nick Schneider, Lukas Schneider, Peter Pinggera, Uwe Franke, Marc Pollefeys, and Christoph Stiller. "Semantically Guided Depth Upsampling." In *German Conference on Pattern Recognition*. Springer, 2016.

[SSS16] Ozan Sener, Hyun Oh Song, Ashutosh Saxena, and Silvio Savarese. "Learning transferrable representations for unsupervised domain adaptation." In *Advances in Neural Information Processing Systems*, pp. 2110–2118, 2016.

[SUH17] Kuniaki Saito, Yoshitaka Ushiku, and Tatsuya Harada. "Asymmetric tri-training for unsupervised domain adaptation." *arXiv preprint arXiv:1702.08400*, 2017.

[SVI16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

[SVL92] Patrice Simard, Bernard Victorri, Yann LeCun, and John Denker. "Tangent prop-a formalism for specifying selected invariances in an adaptive network." In *Advances in neural information processing systems*, pp. 895–903, 1992.

[SWU18] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. "Maximum classifier discrepancy for unsupervised domain adaptation." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3723–3732, 2018.

[SZS13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks." *arXiv preprint arXiv:1312.6199*, 2013.

[SZY16] Hang Su, Jun Zhu, Zhaozheng Yin, Yinpeng Dong, and Bo Zhang. "Efficient and Robust Semi-supervised Learning Over a Sparse-Regularized Graph." In *European Conference on Computer Vision*, pp. 583–598. Springer, 2016.

[THS17] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. "Adversarial discriminative domain adaptation." In *Computer Vision and Pattern Recognition (CVPR)*, volume 1, p. 4, 2017.

[THS18] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. "Learning to adapt structured output space for semantic segmentation." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7472–7481, 2018.

[THZ14] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. "Deep domain confusion: Maximizing for domain invariance." *arXiv preprint arXiv:1412.3474*, 2014.

[TPL20] Mingxing Tan, Ruoming Pang, and Quoc V Le. "Efficientdet: Scalable and efficient object detection." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10781–10790, 2020.

[TV17] Antti Tarvainen and Harri Valpola. "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results." In *Advances in neural information processing systems*, pp. 1195–1204, 2017.

[TVD20] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. "Fixing the train-test resolution discrepancy: FixEfficientNet." *arXiv preprint arXiv:2003.08237*, 2020.

[USS17] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. "Sparsity invariant cnns." In *2017 International Conference on 3D Vision (3DV)*, pp. 11–20. IEEE, 2017.

[UVL17] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, p. 6, 2017.

[VJB18] Tuan-Hung Vu, Himalaya Jain, Maxime Bucher, Mathieu Cord, and Patrick Pérez. "ADVENT: Adversarial Entropy Minimization for Domain Adaptation in Semantic Segmentation." *arXiv preprint arXiv:1811.12833*, 2018.

[VND19] Wouter Van Gansbeke, Davy Neven, Bert De Brabandere, and Luc Van Gool. "Sparse and noisy lidar completion with rgb guidance and uncertainty." In *2019 16th International Conference on Machine Vision Applications (MVA)*, pp. 1–6. IEEE, 2019.

[WBS04] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. "Image quality assessment: from error visibility to structural similarity." *IEEE transactions on image processing*, **13**(4):600–612, 2004.

[WCS20] Alex Wong, Safa Cicek, and Stefano Soatto. "Targeted Adversarial Perturbations for Monocular Depth Prediction." In *Advances in neural information processing systems*, 2020.

[WD18] Mei Wang and Weihong Deng. "Deep Visual Domain Adaptation: A Survey." *Neurocomputing*, 2018.

[WFT20] Alex Wong, Xiaohan Fei, Stephanie Tsuei, and Stefano Soatto. "Unsupervised Depth Completion from Visual Inertial Odometry." *IEEE Robotics and Automation Letters*, 2020.

[WH18] Kai-Ya Wei and Chiou-Ting Hsu. "Generative Adversarial Guided Learning for Domain Adaptation." In *BMVC*, p. 100, 2018.

[WHS19] Alex Wong, Byung-Woo Hong, and Stefano Soatto. "Bilateral Cyclic Constraint and Adaptive Regularization for Unsupervised Monocular Depth Prediction." *arXiv preprint arXiv:1903.07309*, 2019.

[WMZ18] Chaoyang Wang, José Miguel Buenaposada, Rui Zhu, and Simon Lucey. "Learning depth from monocular videos using direct methods." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2022–2030, 2018.

[Wri15] Stephen J Wright. "Coordinate descent algorithms." *Mathematical Programming*, **151**(1):3–34, 2015.

[WRM12] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. "Deep learning via semi-supervised embedding." In *Neural Networks: Tricks of the Trade*, pp. 639–655. Springer, 2012.

[WT11] Max Welling and Yee W Teh. "Bayesian learning via stochastic gradient Langevin dynamics." In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 681–688, 2011.

[WZZ13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. "Regularization of neural networks using dropconnect." In *International conference on machine learning*, pp. 1058–1066, 2013.

[XCZ18] Ruijia Xu, Ziliang Chen, Wangmeng Zuo, Junjie Yan, and Liang Lin. "Deep cocktail network: Multi-source unsupervised domain adaptation with category shift." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3964–3973, 2018.

[XGD17] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. "Aggregated residual transformations for deep neural networks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

[XWZ17] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. "Adversarial examples for semantic segmentation and object detection." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1369–1378, 2017.

[XZF17] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. "A semantic loss function for deep learning with symbolic knowledge." *arXiv preprint arXiv:1711.11157*, 2017.

[XZS19] Yan Xu, Xinge Zhu, Jianping Shi, Guofeng Zhang, Hujun Bao, and Hongsheng Li. "Depth Completion from Sparse LiDAR Data with Depth-Normal Constraints." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2811–2820, 2019.

[YCS16] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. "Revisiting Semi-Supervised Learning with Graph Embeddings." In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 40–48, 2016.

[YLS19] Wei Yin, Yifan Liu, Chunhua Shen, and Youliang Yan. "Enforcing geometric constraints of virtual normal for depth prediction." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5684–5693, 2019.

[YS18] Zhichao Yin and Jianping Shi. "Geonet: Unsupervised learning of dense depth, optical flow and camera pose." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1983–1992, 2018.

[YWS18] Nan Yang, Rui Wang, Jörg Stückler, and Daniel Cremers. "Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry." In *European Conference on Computer Vision*, pp. 835–852. Springer, 2018.

[YWS19] Yanchao Yang, Alex Wong, and Stefano Soatto. "Dense Depth Posterior (DDP) from Single Image and Sparse Range." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[YXC18] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. "Bdd100k: A diverse driving video database with scalable annotation tooling." *arXiv preprint arXiv:1805.04687*, 2018.

[ZBH16] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. "Understanding deep learning requires rethinking generalization." *arXiv preprint arXiv:1611.03530*, 2016.

[ZBS17] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. "Unsupervised learning of depth and ego-motion from video." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1851–1858, 2017.

[ZCL15] Sixin Zhang, Anna E Choromanska, and Yann LeCun. "Deep learning with elastic averaging SGD." In *Advances in Neural Information Processing Systems*, pp. 685–693, 2015.

[ZDG17] Yang Zhang, Philip David, and Boqing Gong. "Curriculum domain adaptation for semantic segmentation of urban scenes." In *The IEEE International Conference on Computer Vision (ICCV)*, volume 2, p. 6, 2017.

[ZF18] Yinda Zhang and Thomas Funkhouser. "Deep depth completion of a single rgb-d image." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 175–185, 2018.

[ZFG19] Shanshan Zhao, Huan Fu, Mingming Gong, and Dacheng Tao. "Geometry-Aware Symmetric Domain Adaptation for Monocular Depth Estimation." *arXiv preprint arXiv:1904.01870*, 2019.

[ZGL03] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. "Semi-supervised learning using gaussian fields and harmonic functions." In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pp. 912–919, 2003.

[ZGL17] Werner Zellinger, Thomas Grubinger, Edwin Lughofer, Thomas Natschläger, and Susanne Saminger-Platz. "Central moment discrepancy (cmd) for domain-invariant representation learning." *arXiv preprint arXiv:1702.08811*, 2017.

[ZQY18] Yiheng Zhang, Zhaofan Qiu, Ting Yao, Dong Liu, and Tao Mei. "Fully convolutional adaptation networks for semantic segmentation." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6810–6818, 2018.

[ZYK18] Yang Zou, Zhiding Yu, BVK Kumar, and Jinsong Wang. "Domain adaptation for semantic segmentation via class-balanced self-training." *arXiv preprint arXiv:1810.07911*, 2018.

[ZYL19] Yang Zou, Zhiding Yu, Xiaofeng Liu, BVK Kumar, and Jinsong Wang. "Confidence regularized self-training." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5982–5991, 2019.

[ZYV18] Yang Zou, Zhiding Yu, BVK Vijaya Kumar, and Jinsong Wang. "Unsupervised domain adaptation for semantic segmentation via class-balanced self-training." In *Proceedings of the European conference on computer vision (ECCV)*, pp. 289–305, 2018.

[ZZL19] Qiming Zhang, Jing Zhang, Wei Liu, and Dacheng Tao. "Category Anchor-Guided Unsupervised Domain Adaptation for Semantic Segmentation." In *Advances in Neural Information Processing Systems*, pp. 433–443, 2019.