UCLA
UCLA Electronic Theses and Dissertations

Title

Architecture, Modeling, and Optimization of Photonic Neural Network Accelerators

Permalink

https://escholarship.org/uc/item/8pq9f0jz

Author

Li, Shurui

Publication Date

2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA

Los Angeles

Architecture, Modeling, and Optimization of Photonic Neural Network Accelerators

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical and Computer Engineering

by

Shurui Li

2024

© Copyright by
Shurui Li
2024

ABSTRACT OF THE DISSERTATION

Architecture, Modeling, and Optimization of Photonic Neural Network Accelerators

by

Shurui Li
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2024
Professor Puneet Gupta, Chair

As artificial intelligence (AI) continues to advance, so too must the computational architectures that support it. Traditional Complementary Metal-Oxide-Semiconductor (CMOS)-based accelerators, while having served as the backbone of computing for decades, are now encountering significant limitations due to the slowing of Moore's Law. Photonic neural network accelerators emerge as a promising alternative, offering high-speed, parallel optical computations that can significantly enhance performance and energy efficiency.

Despite their potential, photonic computing systems face significant challenges, particularly related to the conversion overhead from digital to analog signals and vice versa. The conversion overhead can drastically reduce the energy efficiency of photonic neural network accelerators. This dissertation addresses these challenges by introducing cross-layer optimizations and co-design strategies that span computing methods, circuits, architectures, and algorithms. Specifically, we focus on reducing the conversion overhead through innovative design and optimization techniques.

Part 1: We introduce an innovative use of free-space optical systems, specifically 4F systems, to accelerate CNNs. By leveraging Fourier optics, we reduce the complexity of

convolution operations from $O(N^2)$ to $O(N)$, a feat unachievable by traditional electronic systems. This part includes the design, construction, and optimization of free-space optical CNN accelerators, demonstrating significant performance improvements and energy efficiency through experimental evaluations on datasets such as MNIST and CIFAR-10.

Part 2: We delve into on-chip photonic neural network accelerators, presenting two pioneering architectures: PhotoFourier and ReFOCUS. PhotoFourier leverages the Joint Transform Correlator (JTC) approach to perform convolutions with reduced complexity and fewer photonic components. ReFOCUS builds upon this with innovative features like optical buffers and wavelength-division multiplexing (WDM), further improving energy and area efficiency. We demonstrate that these on-chip designs outperform contemporary photonic and CMOS accelerators in terms of throughput, power efficiency, and energy-delay product (EDP).

Part 3: We focus on algorithmic innovations and theoretical analysis to enhance the efficiency of photonic and analog computing systems. We propose a weight pool compression algorithm that reduces storage requirements and memory traffic, enabling efficient deployment of large neural networks. This compression algorithm also has a promising synergy with analog and photonic neural network accelerators, which could avoid or drastically reduce the conversion overhead of weights. Additionally, since the ADC power is heavily dependent on the bitwidth (ADC resolution), we develop a comprehensive analytical model for partial sum precision requirements, optimizing the trade-offs between accuracy and energy efficiency in analog neural network accelerators.

Experimental JTC challenges and findings: We also included a chapter dedicated to the challenges and non-idealities we observed in our JTC hardware prototype. We further provide sensitivity analysis for various non-linearity of the photodetectors. The goal of this section is to complement the architecture work with some experimental analysis and findings, and provide insights and directions for future work.

Through our contributions, we advance the field of photonic neural network accelera-

tors, providing new architectures, modeling techniques, and optimization strategies. Our findings demonstrate the potential of photonic technologies to achieve high-performance, energy-efficient AI computations, thereby addressing critical challenges in modern computing hardware and paving the way for future advancements in this rapidly evolving domain.

The dissertation of Shurui Li is approved.

Sudhakar Pamarti

Chee Wei Wong

Anthony J. Nowatzki

Puneet Gupta, Committee Chair

University of California, Los Angeles

2024

To my entire family, for their unwavering support and encouragement throughout this challenging yet rewarding journey.

TABLE OF CONTENTS

1 Introduction **1**

1.1 The Challenges of Traditional CMOS-Based Accelerators 1

1.2 Importance of accelerating CNN workloads 2

1.3 Photonic Neural Network Accelerators as a Promising Alternative 3

1.4 Challenges of Photonic Computation 4

1.5 Dissertation Outline 8

2 Acceleration of Convolutional Neural Networks with Programmable Free-space Optics **14**

2.1 Introduction 15

2.2 Baseline System 16

2.2.1 Methods 16

2.2.2 Results 19

2.3 Parallelized System 19

2.3.1 Methods 20

2.3.2 Results 21

2.4 Conclusion 22

3 Optimizations and Performance Scaling Analysis of Free-space Neural Network Accelerators **23**

3.1 Introduction 24

3.2 A Primer on 4F Optical Computing Systems 26

3.2.1 An Overview of Optical 4F Computing System 26
3.2.2 Introduction of 4F Computing System Hardware 27
3.2.3 Leveraging Massive Optical Parallelism Using Computation Tiling . . 31
3.2.4 The Positive-only Photodetection Challenge 33
3.3 Channel Tiling for Optical Compute Parallelism 35
3.3.1 Channel Tiling Operation 36
3.3.2 Mixed Tiling 38
3.3.3 Tiling Efficiency Analysis and Optimization 39
3.3.4 Hardware Requirement Analysis 40
3.3.5 Impact of Camera Bit-Depth and Sensing Noise 41
3.4 Evaluation and Results 42
3.4.1 Comparing Performance of Tiling Approaches 43
3.4.2 Impact of Tiling Approaches on Network Accuracy 44
3.4.3 Impact of Camera Limitations on Inference Accuracy 45
3.5 Discussion 47
3.5.1 Experimental Realization of Programmable 4F System 47
3.5.2 Optimal Filter Size and Architecture Search 48
3.6 Conclusion 49

4 PhotoFourier: A Photonic Joint Transform Correlator-based Neural Network Accelerator 55
4.1 Introduction 56
4.2 A Primer on the JTC system 58
4.2.1 Background of JTC 58

4.2.2 A JTC accelerator prototype 60
4.2.3 Issues faced by on-chip JTC accelerators 60
4.3 2D Convolution Computation on JTC 62
4.3.1 Row tiling 64
4.3.2 Partial row tiling 65
4.3.3 Row partitioning 66
4.3.4 Accuracy of row tiling/partitioning 66
4.4 PhotoFourier Compute Unit 67
4.4.1 Pipelining the PFCU 67
4.4.2 Optimizing PFCU for small filters 68
4.5 Architecture Design 69
4.5.1 Overall system architecture 69
4.5.2 Bottleneck analysis of baseline system 71
4.5.3 Temporal accumulation 72
4.5.4 Choice of parallelization scheme 75
4.5.5 Number of waveguides and PFCUs 77
4.5.6 Dataflow and reuse 79
4.6 Evaluation 81
4.6.1 System Setup 81
4.6.2 Effect of optimizations 83
4.6.3 Area 84
4.6.4 Power 85
4.6.5 Comparison with prior works 86

4.7 Related work 89
4.8 Discussion 90
4.8.1 PhotoForuier-base 90
4.8.2 Laser power adjustment for linear photodetector response 90
4.8.3 Performance sensitivity analysis on effective JTC bitwidth 92
4.9 Conclusion 94

5 ReFOCUS: Reusing Light for Efficient Fourier Optics-Based Photonic Neural Network Accelerator 95

5.1 Introduction 96
5.2 Background 99
5.2.1 Background of JTC 99
5.2.2 Computing 2D convolutions using 1D JTC 101
5.3 A case study for a typical JTC-based accelerator 103
5.4 ReFOCUS Compute Unit 104
5.4.1 Optical reuse 104
5.4.2 Lens sharing 111
5.5 ReFOCUS Architecture 114
5.5.1 Overall architecture 114
5.5.2 Memory hierarchy 115
5.5.3 Dataflow 116
5.5.4 Choice of design parameter 119
5.6 Evaluation 122
5.6.1 Power and area 123

5.6.2 Effect of optimizations 125
5.6.3 Comparison with prior work 127
5.7 Discussion 129
5.7.1 Instruction scheduling 129
5.7.2 Compensating system noise 129
5.7.3 DRAM, weight sharing, and weight DAC 130
5.7.4 Non-CNN tasks 131
5.7.5 Slow light 131
5.8 Related Work 132
5.9 Conclusion 132

6 Bit-serial Weight Pools: Compression and Arbitrary Precision Execution of Neural Networks 136
6.1 Introduction 136
6.2 Addressing Compression and Quantization Challenges for General Purpose Processors 138
6.3 Bit-Serial Weight Pool Methodology 140
6.3.1 Lookup Table Based Bit-serial Computation 145
6.3.2 Lookup Table Bitwidth and Weight Pool Storage 148
6.3.3 Activation Bitwidth and Weight Pool Network Runtime 149
6.4 Weight Pool Implementation: Overheads and Optimizations 149
6.4.1 Bit Unpacking Overheads and Optimized Dataflow 150
6.4.2 Memory Latency and Lookup Table Caching 152
6.4.3 Weight Pool Computation Reuse Through Precomputation 153

6.5 Evaluation 156
6.5.1 Experimental Setup 156
6.5.2 Compression Ratio 157
6.5.3 Accuracy Evaluation 158
6.5.4 Runtime Evaluation 160
6.5.5 Comparison with Binarized Networks 162
6.6 Related Work 162
6.6.1 Neural Network Weight Sharing 162
6.6.2 Lookup Table Based Vector Multiplication Acceleration 163
6.6.3 Software Based Convolution Acceleration for Sub-byte Precision . . . 163
6.7 Conclusion 164

7 Characterizing the Effect of Partial Sum Precision on Accuracy and Energy for Analog Neural Network Accelerators 165
7.1 Introduction 165
7.2 A quick 10.1145/3007787.3001140r on PIM-style analog neural network accelerators 167
7.3 Quantization error modeling 168
7.3.1 Case 1: saturation not allowed 169
7.3.2 Case 2: saturation allowed 171
7.3.3 Optimization of case 2 176
7.4 Evaluation 179
7.4.1 Case 1 179
7.4.2 Case 2 180

7.4.3 A PIM case study 180

7.5 Conclusion 184

8 Experimental Analysis of JTC 185

8.1 Non-idealities, Training, and Calibration of the Experimental JTC Setup . . 186

8.1.1 JTC Hardware Prototype 186

8.1.2 Custom training methodology 187

8.1.3 Custom Neural Network Design 187

8.1.4 Non-Ideal MRR Outputs 189

8.1.5 Phase difference of input signals 190

8.2 Sensitivity of Neural Network Accuracy to Optical System Non-Linearity . . 192

8.2.1 Non-Linear JTC 193

8.2.2 Evaluation Results 194

9 Conclusion and Directions for Future Work 198

9.1 Summary of contributions 198

9.1.1 Custom training flow and neural network model for optical neural network accelerators 198

9.1.2 Efficient architectures for on-chip photonic neural network accelerators 199

9.1.3 Weight pool compression algorithm 199

9.1.4 Experimental insights and real-world challenges 200

9.2 Directions for future works 200

9.2.1 Improving the simulation model of JTC hardware 200

9.2.2 A more advanced JTC prototype 201

9.2.3 Custom deep neural network for 4F/JTC based accelerators 201

9.2.4 Efficient compute-in-memory (CIM) using weight pool 201

References **203**

LIST OF FIGURES

2.1 (a): Schematic representation of a 4F system based on Digital Micromirror Devices (DMDs). (b): Experimental implementation of the amplitude-only Fourier filter based on a DMD 4F system. (c): Convolutional Neural Network structure for the CIFAR-10 dataset. The optical amplitude-only Fourier filter is used as convolution layer, with the subsequent layers realized electronically. (d) Flowchart of the training process. The physical model of the amplitude-only Fourier filter layer is used for training the entire convolutional neural network, obtaining the weights for the kernel to be loaded in the 2nd DMD of the convolution layer. Experimentally obtained results of the Amplitude Only Fourier filtering are fed to the FC layer for performing the final prediction on unseen data. 18

2.2 Inference accuracy of normal space domain convolution, our simulation model, hardware evaluation with and without fine-tuning. 20

2.3 Inference accuracy of different setups. 21

3.1 Illustration of optical 4F system. 26

3.2 Illustration of filter tiling method. (a): Padded input block and tiled filter block. (b): Convolution visualization of filter f1 with the input. (c): Effect of zero padding. The padding between filters ensures that the input is not overlapped with two filters at the same time. 32

3.3 Illustration of the tiling process of channel tiling, which effectively unrolls the channel dimension. Blue regions represent zero padding. (a): Input channel tiling. (b): Filter channel tiling. 50

3.4 Illustration of the convolution process, blue regions represent zero padding. (a): Tiled input and filter blocks. (b): Start point of the valid region. (c): Effect of padding, filters will not overlap with multiple input channels. (d): Example of the invalid region, filters are not convolved with their corresponding input channels. (e): Output format of channel tiling. 51

3.5 Illustration of the tiling process of mixed tiling. Blue regions represent zero padding. (a): Filter channel tiling, multiple filters' channels are tiled on SLM. (b): Input channel tiling, only a single input's channels are tilied on SLM, following the same order as filter channels. 52

3.6 Accuracy of FashionMNIST dataset using different setups. 53

3.7 Accuracy of CIFAR-10 dataset using different setups. 53

3.8 Experimental setup of the DMD-based programmable 4F system. 54

4.1 (a): The annotated layout diagram of a baseline on-chip JTC system. (b): The PCB photo of the fabricated prototype. 60

4.2 Simulated JTC output for a 256-element input (partitioned from a CIFAR-10 input) with tiled convolution kernels. 60

4.3 Visualization of row tiling with an example of 5×5 input, 3×3 kernel, and maximum 1D convolution size of 20. Different rows of the input are represented using different colors. (a): 2D input and kernel. (b): Tiled 1D input and kernel. Kernel rows are zero-padded to match the input row size. The last row of input is not tiled due to the limit of 1D convolution size. (c): Sliding window convolution process of normal 2D convolution to produce rows 1-3 of the output. (d): Sliding window convolution process of 1D convolution. For the first two rows, the tiled kernel rows are aligned with their corresponding input rows and produce valid results. Row 3 illustrates the case where the tiled kernel 'slides' outside the input, and generates invalid results (since input row 5 is not tiled). (e): Edge effect. For 2D convolution with zero-padding when the filter is sliding outside of the inputs, the part outside the input (c, f, j) will convolve with zero. However, for 1D convolution they will convolve with the next input row, producing different results compared to 2D convolution. (f): Output format of 1D convolution. Invalid results are marked with red color. 63

4.4 Visualization of pipelined PFCU. 68

4.5 High-level architecture diagram of PhotoFourier-CG. WS stands for weight SRAM, BUF stands for buffer and PD stands for photodetector. (a): CMOS processing tile assigned to one PFCU. It contains two parts, one part for filter weight data generation and the other part for output processing. (b): PhotoFourier-CG architecture with 8 PFCUs, using 2.5D integration. (c): Simplified layout diagram of PFCU. MRRs and photodetectors are grouped to reduce the width of PFCU. 72

4.6 Power contribution of different components of a 1-PFCU baseline system. 73

4.7 Accuracy of ResNet-s versus different temporal accumulation depth. fp_psum is the accuracy without ADC quantization. 75

4.8 Value of $\frac{IB}{N_{TA}} + CP$ with different IB, for three different number of PFCUs. . . 77

4.9 Visualization of the activation memory mapping and the execution process of input tile 2. IT: input tile, OT: output tile, IC: input channel, OC: output channel, F: filter. (a): Accessing input activation from memory. (b): Storing output activation to memory. 81

4.10 Geometric mean of FPS/W for PhotoFourier with different optimizations. Starting from the baseline, each column adds one optimization to the system, and includes all previous optimizations (columns on the left). 84

4.11 Area breakdown of PhotoFourier. Waveguide routing includes waveguides area and redundant area due to layout constraints. (a): CG version. PIC chiplet: $92.2mm^2$, SRAM: $5.85mm^2$, CMOS tile: $10.15mm^2$. (b): NG version. PFCU: $93.5mm^2$, SRAM: $5.3mm^2$, CMOS tile: $16.5mm^2$. 84

4.12 Power breakdown of the two PhotoFourier versions. (a): PhotoFourier-CG. (b): PhotoFourier-NG. 86

4.13 Inference performance of common CNNs on ImageNet dataset, compared to prior works. Missing bars indicate the data are not available from the original papers. (a) Inference throughput in terms of frames per second (FPS). (b): Inference efficiency in terms of frames per Joule (FPS/W). -nm means PhotoFourier versions without memory access power. To the best of our knowledge, Albireo does not report memory access power, so -nm versions are included for reference. (c): Energy-delay Product (EDP) in terms of $\frac{1}{EDP}$, inverse is used for better visualization, therefore larger is better. 87

4.14 Power breakdown of a baseline JTC (single JTC) with same power numbers for DAC, ADC, and MRR as PhotoFourier-base. 91

4.15 Power breakdown of PhotoFourier-base. 91

4.16 Power breakdown of a 6-bit baseline single JTC setup. 93

4.17 Power breakdown of 6-bit PhotoFourier-base. 93

4.18 Power breakdown of a 4-bit baseline single JTC setup. 94

4.19 Power breakdown of 4-bit PhotoFourier-base. 94

5.1 High-level diagram of a typical on-chip JTC system. 100

5.2 Illustration of how 2D convolution is computed using on-chip JTC system. IR, KR, and OR stand for input row, kernel row, and output row. The gray block represents zero-padding. 102

5.3 (a) Power breakdown of single JTC system and ReFOCUS-baseline. (b): Area breakdown of ReFOCUS-baseline, only photonic components are included. . . . 103

5.4 Schematic diagram of two versions of optical buffers used in ReFOCUS. (a): Feedback version. (b): Feedforward version. 108

5.5 Illustration of how WDM is implemented in ReFOCUS (not drawn to scale). Two wavelengths are modulated and encoded into a single waveguide through MRRs with different wavelengths, for both filters and inputs generation. Photodetectors and ADCs receive the sum of the convolution output of the two wavelengths. . . 112

5.6 High-level architecture diagram of ReFOCUS. CCU stands for CMOS compute unit. Each RFCU has two CCUs, one for input generation and the other for output processing. The diagram is not drawn to scale. 116

5.7 Dataflow used in ReFOCUS. An 8-RFCU system that implements feedforward optical buffers with 4-cycle delay lines and WDM with 2 wavelengths is assumed for this example. IC($a-b$) refers to input channel $a-b$, F(x)C($a-b$) refers to channel $a-b$ of filter x. λ_i refers to the ith wavelength in WDM. R means reused input signal through the optical buffer. Difference channel groups are marked with different colors. 119

5.8 (a): Power breakdown of ReFOCUS-FF. (b): Power breakdown of ReFOCUS-FB. The same legend is applied to both pie charts. 123

5.9 Area breakdown of ReFOCUS. The secondary pie chart shows the area breakdown of non-photonic components (CMOS, SRAM memory, and data buffers). 125

5.10 Relative FPS/W for ReFOCUS with different optimizations. Each column includes the optimizations that are reported on its left. Resnet-34 is used for this benchmark. OB stands for optical buffer while SB stands for SRAM buffer. . . 126

5.11 Two ReFOCUS versions compared to PhotoFourier, in terms of relative FPS, FPS/W, FPS/mm^2, PAP, and inverse of EDP. Benchmarked on 5 CNNs. 127

5.12 ReFOCUS compared with other accelerators. The logarithmic axis is used. (a): FPS. (b): FPS/W. 134

5.13 ReFOCUS compared with different digital accelerators on ResNet-50. RF stands for ReFOCUS. (a) FPS. (b) FPS/W. 135

6.1 High level flow of the proposed framework. Pretrained weights are clustered into weight vectors pool, any fine tuning and activation bitwidth selection are done offline. At inference time, the processor only stores the weight pool dot product results and indices to weight vectors used in the network. 141

6.2 Overall flow of generating a weight pool network from a pretrained network. . . 142

6.3 Visualization of the z-dimension weight grouping. This example shows a $8 \times 3 \times 3$ filter with weight vector size of 4. The weights are grouped in the channel dimension and same color represent weights in a single group. After the z-dimension grouping, 18 $4 \times 1 \times 1$ weight vectors (6 are shown in the figure) are generated for the given filter. 143

6.4 Accuracy of weight pool ResNet-14 with different setups, on the CIFAR-10 dataset. For a weight pool with 3×3 kernels, its setups are denoted by xy_n_(coeff), where n means the weight pool size (how many weight vectors in the weight pool) and coeff means the version with scaling coefficients. For the z-dimension weight pool, the setups are denoted by z_n_g8, where n is the weight pool size and g8 means the weight vector size (group size) is 8. The original accuracy is 92.26%. 144

6.5 Visualization of the bit decomposition step. (a): The original 8-element dot product between input and weight vectors. (b): The original dot product is transformed into matrix-vector multiplication followed by dot product after bit decomposition. I_{mn} means the nth bit (starting from LSB) of the mth element. The original input vector is decomposed into an 8×8 matrix with each element representing a single bit. Each column represents all the bits of an input value while each row represents a unique bit position of all input values. The weight vector is kept the same and is multiplied with all the bit positions of input. The result of the matrix-vector multiplication should be the dot product of input and weight vector at every input bit position. The result is then multiplied with the power-of-two vector which represents bit weights to generate the final dot product result. 147

6.6 Visualization of lookup table caching. Green blocks represent active lookup table regions corresponding to the input vectors that are shared across filters. Red blocks represent the inactive lookup table regions. Active regions are cached into SRAM before the filter loop and the function only accesses lookup table results from SRAM. 154

6.7 Relative speedup of just lookup table caching (orange) and precomputation + lookup table caching (green) against baseline implementation. Four 3×3 convolution layers with different number of filters are tested. The number of channel is set to be same as number of filters and the input size is 16×16. Weight pool size is 64. 156

6.8 Speedup against 8-bit bit-serial lookup implementation for different activation bitwidths. (a): results without precomputation. (b): results with precomputation. The input size is 16×16 and number of channels and filters are both 128. Weight pool size is 64. 160

7.1 Histogram of the distribution of a batch of multiplication outputs. 172

7.2 Normalized mean absolute clipping error versus number of summations obtained from different methods, for different α. Y-axis is the normalized clipping error. 178

7.3 Mean absolute clipping error generated with the proposed model and simulation, for different hardware dot product sizes. 179

7.4 Mean absolute clipping error with different model parameters generated using the analytical model and simulation. (a): α (relative ADC threshold). (b): Hardware dot product size. (c) ADC bitwidth. (d): Mean of the original normal distribution before truncation (multiplication outputs). 181

7.5 (a): Power of the memristor array vs. column sizes for fixed and adjusted ADC bitwidth. (b): Mean absolute quantization error vs. column size for the case with adjusted ADC bitwidth. 183

8.1 PCB photo and layout diagram of the JTC prototype. 186

8.2 CIFAR-10 simulation accuracy using the hardware simulation model, with and without modeling of non-ideal MRR behavior. 190

8.3 CIFAR-10 simulation accuracy using the hardware simulation model, with and without modeling of the path difference between JTC channels. 192

8.4 CIFAR-10 simulation accuracy using the hardware simulation model, for various power functions applied at the Fourier plane. 195

8.5 CIFAR-10 simulation accuracy using the hardware simulation model, for various photodetector saturation probabilities. 196

LIST OF TABLES

3.1 Table of common notations used in this paper. 31

3.2 Comparison of single convolution time (in seconds) between 4F SLM and CuDNN implementations, taking into account the effect of tiling and parallelism. 33

3.3 Comparison of the resolution requirements for different tiling schemes, assuming all cases are fully tiled. 41

3.4 Overall network inference time in seconds (per input) for different tiling schemes and network architectures. The results are for convolution layers only. Note for DeconvNet there's a convolution layer of 1x1 filters which is not suitable for 4F system acceleration and is not taken into account for the estimation. 44

3.5 Comparison of the accuracy of different datasets trained using VGG16 with different methods. For Input and filter tiling the absolute value function is applied to each individual channel's convolution result while for channel and mixed tiling the absolute value function is applied after channel summation and acts as the activation function. For pseudo-negative cases, positive weights are used and subtraction is implemented after camera detection, then ReLU is applied on subtracted results as the activation function. 45

4.1 Original accuracy of three CNNs and the accuracy drop of different neural network accelerators (%). T-1 and T-5 mean top-1 and top-5 accuracy. Ours stands for the proposed row tiling/partitioning method with 1D convolution. Accuracy drop is reported instead of raw accuracy because we have slightly different original accuracy than what is reported in [LLY19] and [ZLY20]. Top-1 accuracy is not reported in both prior works. 67

4.2 Table of notations used in the analysis and their meaning. 76

4.3 Maximum number of input waveguides per PFCU and the geometric mean of normalized FPS/W on 5 CNNs for PhotoFourier-CG and PhotoFourier-NG with different number of PFCUs, given a 100 mm^2 area budget. 79

4.4 Power of different components and high-level design parameters used for PhotoFourier-CG and PhotoFourier-NG 83

4.5 Dimensions of the photonic components used in area estimation 83

5.1 The length, area, and loss of a delay line with 0.1 ns delay (1 cycle in 10 GHz system). 110

5.2 Area and normalized area efficiency in terms of frames per second per mm^2 of a 16-RFCU system with different wavelengths. 114

5.3 Notations and definitions of common terms used in the analysis. 118

5.4 Number of RFCUs can be placed and relative FPS/W, FPS/mm^2, PAP for different delay line lengths in terms of cycles, for both ReFOCUS-FF and ReFOCUS-FB. The absolute values are shown for the baseline case where $M = 1$. 121

5.5 Relative laser power when compared to the system without optical buffer and the dynamic range of input signals for different R and α. 122

5.6 Power of active components and the area of photonic components used in ReFOCUS. 124

5.7 Potential reuse can be achieved by different optimizations. OB stands for optical buffer and TA stands for temporal accumulation. 126

6.1 Accuracy of z-dimension weight pool with different group size. The network is ResNet-14 and dataset is CIFAR-10. Original network accuracy is 92.26%. . . . 143

6.2 STM Nucleo family microcontrollers used for benchmarking. Both use ARM Cortex M3 for the core. 155

6.3 Total number of parameters (uncompressed), overall compression ratio (CR) and lookup table overhead of the selected networks. The lookup table overhead is the proportion of lookup table storage to the total network storage after compression. 157

6.4 Accuracy (%) of the z-dimension weight pool with different weight pool sizes on selected network-dataset combinations. Original means original network accuracy and 32/64/128 are the weight pool size. 158

6.5 Inference accuracy (%) of bit-serial lookup table implementation. No-LUT column shows accuracy that not using lookup table implementation. The activation bitwidth is 8 bit. 159

6.6 Inference accuracy (%) of weight pool networks with different activation bitwidths. Results in brackets are accuracy after retraining. The last column shows the minimum activation bitwidth with less than 1% accuracy drop. The lookup table bitwidth is set to 8 bit. 160

6.7 Full-network inference latency (in seconds) with different setups for both microcontrollers. CM. stands for CMSIS implementation, -8 means 8-bit activation precision while -m means minimum activation precision that has less than 1% accuracy drop that determined in Table 6.6. 32 and 64 are the weight pool size. / means the network cannot fit into flash memory. 161

7.1 Common symbols used in the analysis and their meanings. 169

7.2 Default value of model parameters. 180

7.3 Summary of component power in the baseline configuration. 182

8.1 Simulation accuracy on CIFAR-10 dataset of the JTC prototype using ideal square function versus the JTC prototype using the actual transfer function of photodetectors. 195

ACKNOWLEDGMENTS

This dissertation is the culmination of five years of hard work, representing not just my research but also the prime years of my youth. These formative years were wholeheartedly devoted to my PhD journey. I poured my energy, passion, and focus into this work, and the experience has profoundly shaped both my academic and personal life.

The road was filled with challenges—late nights coding, navigating complex simulations, last-second paper rush, and the relentless pursuit of new knowledge—but it was also a period of tremendous growth and moments of triumph. Looking back, I can say with confidence that this has been an incredibly rewarding journey, and I am deeply grateful for the decision to walk this path five years ago. However, none of this would have been possible without the incredible support, guidance, and encouragement of many individuals, to whom I owe my deepest gratitude.

First and foremost, I would like to express my deepest thanks to my advisor, Professor Puneet Gupta. Throughout my PhD, he has been an outstanding mentor, providing a constant source of fresh ideas and critical insights that helped guide my research. His ability to strike the right balance between encouragement and challenge has been pivotal in my development as a researcher. What I admire most about Professor Gupta is the mutual respect that characterized our discussions. Even when we disagreed or when my work needed improvement, he remained calm, patient, and supportive, creating an environment that fostered both intellectual growth and personal development. I am profoundly grateful for his mentorship, which has shaped not only this dissertation but also the researcher I have become.

I am also deeply indebted to Professor Chee Wei Wong from UCLA and Professor Volker Sorger from the University of Florida for their invaluable collaboration throughout my research. Their guidance and expertise have been instrumental to the success of many projects featured in this dissertation. Their intellectual contributions, coupled with their support and

encouragement, enriched my work in countless ways. I am extremely fortunate to have had the opportunity to collaborate with such distinguished scholars and to learn from them during the course of my PhD.

A special thanks goes to my collaborators and internship mentors. I had the privilege of working alongside brilliant researchers from Professor Wong's and Professor Sorger's groups, including Dr. Hangbo Yang, Dr. Mario Miscuglio, Dr. Zibo Hu, Dr. Nicola Peserico, and many others. Their insights, dedication, and collaboration were invaluable in shaping the direction of many projects. I would also like to express my gratitude to my internship managers at Qualcomm, Sameer Wadhwa and Michael Chen, for their mentorship during my two internships. Their leadership, practical guidance, and thoughtful feedback opened my eyes to the industry's perspective on my research, greatly enriching my understanding of the field. I am truly grateful for the opportunities to grow, both professionally and personally, under their guidance.

I am fortunate to have shared this journey with incredible labmates and friends, whose camaraderie made the long hours in the lab both productive and enjoyable. To Dr. Saptadeep Pal, Dr. Irina Alam, Dr. Tianmu Li, Dr. Wojciech Romaszkan, Alexander Graening, Yingheng Li, Rhesa Ramadhan, George Karfakis, Ravit Sharma, and Zhichao Chen, thank you for your companionship, stimulating discussions, and constant support. The friendly, collaborative spirit in our lab created a unique working environment that made even the toughest challenges more manageable, and I am deeply thankful to have worked alongside such a talented group of people. I would also like to express my gratitude to all my friends, whose companionship and shared laughter have brightened this journey and made the difficult moments more bearable.

On a more personal note, I owe a huge debt of gratitude to my two faithful fluffy friends, Nuomi and Panghu. Their comforting presence, soothing purrs, and occasional keyboard invasions provided much-needed joy during the long nights of work. I also want to honor the memory of my beloved dog, Hali, who was by my side through my teenage years back in

China and sadly returned to the stars during my PhD. Though he is no longer with me, his memory has been a source of comfort and strength throughout this journey.

Most importantly, I would like to express my deepest gratitude to my wife, Hongjing Sun, for her unwavering support, love, and companionship. We have shared this journey not only as fellow PhD students but as life partners, and her presence has been a constant source of strength and happiness. Despite the demands of our academic careers, our time together has been a cherished escape from the pressures of research. From enjoying delicious food to exploring theme parks and national parks, these moments of joy have allowed me to recharge and reconnect with life outside academia. Her belief in me, especially during moments of self-doubt, has been nothing short of invaluable. I feel incredibly fortunate to have had her by my side throughout this journey, and I cannot thank her enough for the endless love and encouragement she has given me.

I also want to express my heartfelt gratitude to my grandparents, who have been a source of strength and love throughout my life. My maternal grandfather, Enyi Jia, who was always my biggest supporter and believed in my potential unconditionally since I was a child. Though he passed away before I began this PhD journey, his belief in me has been a guiding force throughout my life, and I dedicate much of this achievement to his memory. To my maternal grandmother, Guiyun Chen, and my paternal grandparents, Zhenglun Li and Lanying Zhang, your love, care, and belief in me have been constant sources of comfort and motivation during this long and challenging journey. I am deeply grateful for the strong foundation you have provided, and for the values you instilled in me, which have carried me through these past five years.

Finally, I wish to offer my deepest thanks to my parents, Tiejun Li and Qun Jia, for their boundless love, support, and belief in me. Though we have often been separated by great distances, their unwavering encouragement and understanding have been my foundation throughout this journey. Their love has lifted me through the difficult times and shared in my moments of success, and their support has made my PhD life much less stressful. I am

especially grateful to them for guiding me when I was feeling lost while choosing my major for my bachelor's degree, helping me find the path that ultimately led to this field. Without their sacrifices and support, none of this would have been possible. For all they have done, I am forever grateful.

Copyrights, Re-use of Published Materials, and Collaborative Contributions

This dissertation contains significant material that has been previously published or is intended to be published in the future. Chapter 2 contains materials published in [MHL20] and [HLS22]. Chapter 3 contains materials published in [GL22]. Chapter 4 contains materials published in [LYW23a] and chapter 5 contains materials published in [LYW23b]. Chapter 6 contains materials published in [LG22].

Many projects of this dissertation are collaboration efforts, while some work are led by myself and some are led by my collaborators. Here I want to give credit and clarify contributions to each collaboration work. Chapter 2 contains two collaboration works led by Dr. Mario Miscuglio [MHL20] and Dr. Zibo Hu [HLS22] respectively, both from George Washington University. My contribution to these two projects is to design and train the custom neural networks for the actual hardware used in these projects. Although these works were not led by myself, they are included in this dissertation because they serve as the foundation for much of the subsequent work, and thus deserve introduction.

Chapter 3 is an architecture exploration of the work discussed in Chapter 2 and is led by myself. My main collaborators are Dr. Mario Miscuglio [MHL20] and Dr. Zibo Hu, credited for their experimental work discussed in Chapter 2.

Chapter 4 and chapter 5 are two architecture works about the on-chip version of the photonic neural network accelerator, both of which were led by myself. My main collaborators in these works were Dr. Hangbo Yang from UCLA and Dr. Nicola Peserico from the University of Florida. Their contributions included providing theoretical analysis, certain simulation results related to photonic components, and chip measurement results for our

prototype.

VITA

2019	M.Eng. (B.Eng. integrated), Electrical and Electronic Engineering, Imperial College London, UK
2019	UCLA ECE Department Fellowship
2020	PhD Intern, Qualcomm Inc.
2023	PhD Intern, Qualcomm Inc.
2023	UCLA Dissertation Year Fellowship
2024	Distinguished PhD Dissertation Award, UCLA ECE

PUBLICATIONS

Li, Shurui, Hangbo Yang, Chee Wei Wong, Volker J. Sorger, and Puneet Gupta. "ReFOCUS: Reusing Light for Efficient Fourier Optics-Based Photonic Neural Network Accelerator." In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 569-583. 2023.

Li, Shurui, Hangbo Yang, Chee Wei Wong, Volker J. Sorger, and Puneet Gupta. "Photo-Fourier: A Photonic Joint Transform Correlator-Based Neural Network Accelerator." In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 15-28. IEEE, 2023.

Li, Tianmu, **Shurui Li**, and Puneet Gupta. "Training Neural Networks for Execution on Approximate Hardware." *tinyML Research Symposium*, 2023.

Li, Shurui, and Puneet Gupta. "Bit-serial Weight Pools: Compression and Arbitrary Precision Execution of Neural Networks on Resource Constrained Processors." *Proceedings of Machine Learning and Systems 4* (2022): 238-250.

Li, Shurui, and Puneet Gupta. "4F optical neural network acceleration: an architecture perspective." In *AI and Optical Data Sciences III*, vol. 12019, pp. 77-84. SPIE, 2022.

Hu, Zibo, **Shurui Li**, Russell LT Schwartz, Maria Solyanik-Gorgone, Mario Miscuglio, Puneet Gupta, and Volker J. Sorger. "High-Throughput Multichannel Parallelized Diffraction Convolutional Neural Network Accelerator." *Laser & Photonics Reviews* (2022): 2200213.

Li, Shurui, Wojciech Romaszkan, Alexander Graening, and Puneet Gupta. "SWIS–Shared Weight bIt Sparsity for Efficient Neural Network Acceleration." *tinyML Research Symposium*, 2021.

Miscuglio, Mario, Zibo Hu, **Shurui Li**, Jonathan K. George, Roberto Capanna, Hamed Dalir, Philippe M. Bardet, Puneet Gupta, and Volker J. Sorger. "Massively parallel amplitude-only Fourier neural network." *Optica* 7, no. 12 (2020): 1812-1819.

CHAPTER 1

Introduction

1.1 The Challenges of Traditional CMOS-Based Accelerators

As the landscape of artificial intelligence continues to evolve, the architectures underpinning its computational mechanisms must also adapt. Central to the current challenges in AI hardware development is the reliance on Complementary Metal-Oxide-Semiconductor (CMOS) technology, which has been the bedrock of computing hardware for decades. While CMOS technology has historically met the demands of increasing computational needs through scaling according to Moore's Law, recent years have seen a significant slowdown in these advancements. As Moore's Law—predicting the doubling of transistors on a microchip approximately every two years—approaches its physical and economic limits, the semiconductor industry faces critical challenges in enhancing performance while managing power consumption and heat dissipation when scaling beyond advanced nodes.

Neural network models, for example Convolutional Neural Networks (CNNs), are becoming increasingly complex. This complexity manifests in larger models with billions of parameters, requiring trillions of operations for even a single forward pass through the network. Traditional CMOS-based accelerators, while benefiting from decades of refinement, are hitting a performance wall due to inherent limitations in serial processing capabilities, energy inefficiency at smaller node sizes, and the latency introduced by data movement between memory units and processing cores. The energy cost of arithmetic operations and data movement in these accelerators becomes prohibitively high as models scale, making

them less suitable for power-sensitive applications.

1.2 Importance of accelerating CNN workloads

Despite the recent surge in popularity of transformer-based models, particularly in natural language processing and some computer vision tasks, the acceleration of CNNs remains critically important for several reasons. First, CNNs continue to be widely used in many state-of-the-art computer vision models due to their effectiveness in handling spatial hierarchies and local contexts in images—capabilities that are intrinsic to the architecture of CNNs. These models are fundamental in applications ranging from autonomous vehicle systems to medical image analysis, where precision and reliability are paramount.

Second, the practical applications of CNNs often demand fast processing times, which is critical in real-time systems where decisions must be made rapidly and reliably. This requirement is particularly pronounced in fields such as autonomous driving and drone navigation, where the ability to process visual information swiftly can be the difference between safe operation and catastrophic failure. Additionally, CNNs are integral to real-time video analysis and augmented reality applications, where delays in processing can significantly degrade user experience and effectiveness. The need for speed in these applications makes the acceleration of CNNs not just a technical desire but a practical necessity.

Moreover, as these applications often operate within power-constrained environments, optimizing CNNs for both speed and energy efficiency becomes even more crucial. Efficient acceleration ensures that CNNs can be deployed more broadly, including in mobile and edge devices, where power and thermal constraints limit the feasibility of deploying larger, more power-intensive models. Thus, the acceleration of CNNs aligns with the broader goals of making AI more accessible and sustainable, particularly in a world where the proliferation of smart devices and IoT technologies relies increasingly on efficient and powerful computational capabilities at the edge.

Third, the architectural characteristics of CNNs make them more amenable to acceleration through hardware optimizations. Unlike transformers, which are often bottlenecked by memory access due to their extensive need for parameter interactions (resulting in what is known as the "memory wall"), CNNs are typically more compute-bound. This distinction means that CNNs suffer less from memory latency issues and stand to benefit more directly from improvements in computational speed. Accelerating the compute capabilities directly translates to performance gains for CNNs, making them particularly suitable candidates for photonic acceleration, which can exploit high-speed, parallel optical computations to enhance throughput and energy efficiency significantly. Therefore, while the landscape of machine learning models continues to evolve, the optimization and acceleration of CNNs remain a high priority, given their ongoing relevance and distinct computational characteristics.

1.3 Photonic Neural Network Accelerators as a Promising Alternative

Photonic neural network accelerators emerge as a compelling alternative to traditional CMOS-based systems[HSM22, STN21, DCC19, XJ23]. Photonics, the science of generating, controlling, and detecting photons, offers several intrinsic advantages that can alleviate the limitations faced by electronic systems. Key among these advantages are the high bandwidth and speed of light transmission, which allow for rapid data transfer with minimal latency. Unlike electronic systems, where data transfer speed is often a limiting factor, photonic systems can move data at the speed of light, potentially reducing the data transport delays that bottleneck large-scale neural computations.

Besides, photonic components that generate signals and perform computations generally can operate at very high frequencies, e.g., more than 10 GHz. Therefore, photonics computing systems have a speed advantage over CMOS-based computing systems.

Additionally, photonic systems can exploit the parallelism of light waves, allowing mul-

tiple data streams to be processed simultaneously through techniques such as wavelength-division multiplexing (WDM). This capability enables significant throughput enhancements for tasks like matrix multiplication, a fundamental operation in neural network computations, which can be performed in parallel across multiple wavelengths without interference.

Energy efficiency is another critical advantage of photonic technologies. Photons do not carry charge and do not generate heat through resistance as electrons do, which means that photonic devices can operate with much lower energy costs and minimal thermal output. This characteristic is particularly important as power consumption becomes a limiting factor in scaling up computational resources in data centers and other AI deployment scenarios.

Lastly, the integration of photonic components on a chip—using processes compatible with existing semiconductor manufacturing techniques—offers a pathway to creating compact, integrated systems that combine the advantages of optical processing with the scalability and versatility of silicon-based technology. The ability to integrate photonic components using established CMOS fabrication processes also suggests a convergence of electronic and photonic systems, potentially leading to hybrid devices that leverage the strengths of both technologies to address the demands of modern computing tasks.

Given these compelling attributes, exploring photonic technologies for neural network acceleration is not merely an academic exercise but a necessary evolution in computing hardware to keep pace with the advancing frontiers of artificial intelligence.

1.4 Challenges of Photonic Computation

Photonic computation holds immense potential, yet it faces significant challenges that impede its progress toward fulfilling that potential [LDY23, LZL21]. These challenges can be broadly categorized into two main areas: technology and devices, and system and performance.

On the technology and device side, challenges include the maturity of the technology, thermal sensitivity, and various device variations and noise. While this dissertation does not

directly address these issues, considerable ongoing research aims to enhance the maturity, stability, and predictability of photonic components and circuits.

At the system and architecture level, photonic computing platforms—and specifically photonic neural network accelerators—encounter several significant challenges. If not adequately addressed, these challenges can severely compromise the power efficiency and precision of these accelerators. A principal challenge is the conversion overhead. Due to the absence of effective photonic memory solutions, signals must be retrieved from memory or buffers and converted into analog signals via Digital-to-Analog Converters (DACs) before computation. Subsequently, these signals are converted back to digital format using Analog-to-Digital Converters (ADCs) for storage. Given that photonic components can operate at extremely high frequencies, such as 10 GHz, the DACs and ADCs also need to function at these speeds. This requirement can dominate system power consumption, significantly reducing the efficiency of photonic neural network accelerators and potentially making them less efficient than their state-of-the-art CMOS counterparts due to the power-intensive nature of high-speed, high-precision DACs and ADCs. Without strategic optimization across the entire stack, the energy efficiency of photonic neural network accelerators may fall behind.

Therefore, it is critical to minimize the overall conversion overhead to allow photonic neural network accelerators to achieve their promised efficiency. This dissertation is dedicated to enhancing the overall energy efficiency of these accelerators, primarily by reducing the conversion overhead through integrated co-design and co-optimization across computing methods, circuits, architecture, and algorithms. This dissertation is divided into three parts, each addressing the challenge from a different perspective.

The first part focuses on innovations in computing methods. While most current photonic neural network accelerators primarily perform multiplication and addition—the core operations in neural networks—using photonic components such as Mach-Zehnder Interferometers (MZIs) and micro-ring resonators (MRRs), these methods often do not substantially improve energy efficiency over state-of-the-art CMOS accelerators and may even result in

lower efficiency due to high conversion overheads.

In this dissertation, we tackle the problem in a fundamentally different way—by using Fourier optics to reduce the complexity of convolution operations. By employing a pair of Fourier lenses, a 4F (F stands for focal length) system can be built to reduce the complexity of convolution operations from $O(N^2)$ to $O(N)$. This complexity reduction leverages the well-known convolution theorem, which states that convolution in the spatial domain is equivalent to point-wise multiplication in the Fourier domain. While CMOS computing systems can also use the convolution theorem to reduce the complexity of convolution to $O(N \log(N))$ by using FFT to implement the Fourier transform, optical systems can achieve this more efficiently. The Fourier transform can be implemented passively through Fourier lenses, which consume no power. Consequently, the overall computation required is just the point-wise multiplication, which has $O(N)$ complexity.

This complexity reduction is the core idea of the photonic systems proposed in this dissertation, as it is a unique advantage applicable only to Fourier optics-based optical/photonic computing systems. Although focusing on 4F systems restricts the scope to accelerating CNNs, the efficiency benefits are significant. For an order N number of ADC/DAC conversions, an order N^2 number of computations can be performed, significantly reducing the number of conversions required per computation.

The work in the first part involves designing, building, verifying, and optimizing free-space optical 4F system-based neural network accelerators. While the free-space versions are bulky and less efficient compared to their on-chip counterparts, they serve as proof-of-concept work to verify the feasibility of using 4F systems to accelerate CNNs, paving the way for the eventual on-chip version, which is the focus of the third part of this dissertation.

The second part of this dissertation focuses on the architecture of on-chip photonic neural network accelerators. The proposed architectures are based on the Joint-Transform Correlator (JTC), a variant of the 4F system, and share the same complexity reduction advantage. Compared to the free-space variants, on-chip photonic neural network accelerators offer sub-

stantially better energy efficiency and broader use cases. However, even with complexity reduction, ADC and DAC overhead still dominates system power, necessitating further optimizations to make photonic neural network accelerators more energy-efficient than CMOS counterparts. In this part, various architectural and system-level designs and optimizations are proposed to further reduce DAC and ADC overhead, including signal broadcasting, temporal accumulation, wavelength-division multiplexing (WDM), optical buffering, and dataflow optimization. These optimizations further reduce both DAC and ADC power, enhancing the overall energy efficiency to surpass state-of-the-art CMOS neural network accelerators.

The third part of the dissertation focuses on the algorithm and software side to optimize the performance and efficiency of photonic computing systems and analog computing systems in general. We propose a codebook-based compression algorithm, termed weight pool, to compress neural networks and reduce storage requirements as well as memory traffic. This algorithm also synergizes well with photonic neural network accelerators and can be adopted to reduce DAC energy, which tends to be a major power contributor for JTC-based accelerators. With weight pool compression, all weight vectors are chosen from a small codebook that could be entirely hard-coded onto analog computing systems. This approach could potentially eliminate or significantly reduce the weight signals' conversion overhead, thereby improving the overall energy efficiency of photonic neural network accelerators.

Besides the compression algorithm, a theoretical study and modeling of partial sum precision requirements are also conducted in this part. Partial sum precision requirements are crucial in determining the optimal ADC bitwidth, as ADCs usually receive partial sums rather than full output activations due to the size and utilization limitations of photonic computing systems. Since ADC bitwidth has an exponential relationship with ADC power, and ADC power constitutes a large portion of overall system power, it is essential to choose ADCs with the smallest possible bitwidth for a given accuracy or output precision requirement.

After the three parts, a separate section is included to discuss and analyze the various challenges and non-idealities of real-world experimental JTCs such as our JTC prototype.

1.5 Dissertation Outline

This dissertation is split into three parts, with 9 chapters in total including the introduction and conclusion chapter. A brief overview of each chapter in this dissertation is included in this section.

Chapter 1: This is the introduction of this dissertation. This chapter introduces the background and motivation of this dissertation, as well as provides an overview of the dissertation flow and introduction of each chapter.

Chapter 2, Part 1:

This chapter presents the development and evaluation of an innovative platform utilizing free-space optics for accelerating convolutional neural networks (CNNs), addressing the critical challenges posed by the increasing size of state-of-the-art neural networks and the slowdown of Moore's law. Through experimental demonstrations, the study explores the use of amplitude-only electro-optical convolutions performed with high-speed reprogrammable digital micromirror devices (DMDs) in a 4F optical system. This approach significantly enhances throughput and accuracy in processing datasets such as MNIST and CIFAR-10.

The system employs a novel method of implementing Fourier transforms and convolutions optically, enabling real-time image processing capabilities that are markedly more energy-efficient than traditional CMOS technology. By leveraging the inherent parallelism and low energy consumption of free-space optics, this work showcases a scalable alternative for executing CNNs. System enhancements include input tiling and high-pass filtering to optimize throughput and system utilization, culminating in a demonstration of effective real-time deep learning tasks with improved performance metrics.

This summary provides a high-level overview focusing on the key achievements and technological innovations discussed in the chapter, which would set the stage for its detailed examination in the subsequent sections of your dissertation.

Chapter 3, Part 1:

This chapter explores the scalability and performance of 4F systems for CNN acceleration, focusing on a novel channel tiling method. This method accumulates convolution results inherently in the optical domain, enabling the use of filters with negative weights and significantly reducing computational overhead. By leveraging channel tiling, the proposed approach not only enhances the performance and accuracy of optical CNNs but also makes these systems more viable for high-speed, high-resolution neural network acceleration.

Chapter 4, Part 2:

This chapter introduces PhotoFourier, a pioneering on-chip photonic neural network accelerator designed to enhance the computational efficiency of Convolutional Neural Networks (CNNs) using the Joint Transform Correlator (JTC) approach. Unlike traditional digital accelerators, PhotoFourier leverages the low-latency and high-throughput capabilities of integrated photonics to perform convolutions, effectively addressing the computational demands and power inefficiencies associated with modern AI applications. The core innovation lies in its use of Fourier optics to simplify the convolution process, offering a significant reduction in the complexity and number of required photonic components compared to existing solutions.

The chapter outlines the architecture of PhotoFourier, detailing its unique ability to compute two-dimensional convolutions through one-dimensional on-chip lenses. This capability is facilitated by a novel algorithm proposed for approximating 2D convolutions using 1D operations, a method that proves critical for integrating the system on a chip. Additionally, the temporal accumulation technique introduced here enhances both the accuracy and energy efficiency of the system by reducing the frequency and power requirements of analog-to-digital conversions.

In-depth analyses are presented to determine optimal design parameters, such as dataflow strategies, parallelization schemes, and the appropriate number of waveguides and compute units. These considerations are crucial for maximizing the performance and efficiency of the JTC-based accelerator. Furthermore, the chapter compares PhotoFourier against contemporary photonic and electronic accelerators, demonstrating its superior performance in terms of throughput, power efficiency, and energy-delay product (EDP).

Finally, the chapter discusses the broader implications and remaining challenges for on-chip photonic accelerators, including issues like the scalability of optical components, manufacturing variations, and the overarching need for network architectures that are better suited for photonic execution. The insights garnered from the development and analysis of PhotoFourier not only advance the field of photonic neural network accelerators but also set a foundational stage for future research in this rapidly evolving domain. This work marks a significant step toward realizing efficient, high-performance computing architectures that leverage the unique properties of photonics for artificial intelligence applications.

Chapter 5, Part 2:

This chapter introduces ReFOCUS, a Fourier-optics on-chip photonic neural network accelerator designed to enhance energy and area efficiency through optical reuse. ReFOCUS is a follow-up work of PhotoFourier, and further improves energy efficiency through many innovative and careful optimizations.

ReFOCUS introduces optical buffers, enabling efficient reuse of light through optical delay lines, and optimizes dataflow and memory hierarchy. Two versions of optical buffer designs are proposed: feedback and feedforward, each with distinct advantages. Additionally, wavelength-division multiplexing (WDM) is employed to share on-chip lenses and improve area efficiency.

Evaluations demonstrate that ReFOCUS achieves 2× throughput, 2.2× energy efficiency, and 1.36× area efficiency compared to state-of-the-art photonic neural network accelerators.

It also outperforms MZI/MRR-based accelerators by over 25× in power efficiency. These substantial improvements highlight the potential of ReFOCUS for future high-performance computer vision applications, emphasizing its relevance and contributions to the field of photonic neural network accelerators.

Chapter 6, Part 3: With ReFOCUS architecture, the computing part becomes more efficient and the memory access energy starts to contribute more to the overall system power. The two largest power contributors in ReFOCUS are DAC energy and memory access energy, with more than 50% combined power consumption. In order to overcome these limitations, a novel compression algorithm called weight pool is proposed. A framework that combines network compression and acceleration using arbitrary sub-byte precision is further proposed, achieving significant reductions in both storage requirement and total memory traffic for almost any neural network. This compression method also has interesting synergy analog computing systems including the JTC-based neural network accelerators, and can be leveraged in such systems to reduce the DAC efficiency.

This chapter presents this compression framework, as well as a microcontroller-based implementation of this compression method, as an extra use case for this frame work.

The framework's first component involves compressing neural networks by sharing a pool of weight vectors across the entire network. This technique, referred to as weight pool networks, enables up to 8× compression compared to 8-bit networks with negligible accuracy loss. By storing indices of shared weight vectors instead of actual weights, the parameter storage requirements of the network are significantly reduced.

The second component is a bit-serial lookup-based software implementation that supports arbitrary precision execution of weight pool networks. This approach allows for runtime-bitwidth trade-offs, providing more than 2.8x speedup and 7.5x storage compression compared to 8-bit networks, with less than 1% accuracy drop. The bit-serial computation processes multiplications serially by looping through all bits of one operand, leveraging the value reuse properties of weight pools to achieve substantial runtime reduction.

Extensive evaluations of different network-dataset combinations, including ResNet and MobileNet, demonstrate the effectiveness of the proposed framework. The results show that the bit-serial weight pool networks can achieve significant compression and speedup, making it feasible to deploy large neural networks on small microcontrollers without compromising accuracy.

Chapter 7, Part 3:

This chapter delves into the critical issue of partial sum precision in analog neural network accelerators, focusing on its impact on accuracy and energy efficiency. Analog neural network accelerators, such as processing in memory (PIM) and on-chip photonics, have garnered significant interest due to their superior power efficiency. However, Analog-to-Digital Converters (ADCs) often emerge as the power bottleneck, making high-precision ADCs infeasible. This constraint necessitates the quantization of partial sums, which typically requires high precision and can significantly influence the overall accuracy and efficiency of the accelerators.

The chapter presents an analytical model of the quantization error associated with partial sum quantization in analog neural network accelerators. This model, to the best of our knowledge, is the first to consider both rounding and clipping errors, providing a comprehensive framework for exploring the trade-offs between accuracy and performance. The work begins by examining the role of partial sum precision in various analog accelerators, highlighting its significance in determining the system's energy efficiency and accuracy.

To validate the proposed models, extensive evaluations are conducted, comparing the analytical results with simulation data. The chapter also includes a case study on a PIM architecture, illustrating how the models can be used to optimize design parameters for enhanced power efficiency and reduced quantization error.

Chapter 8: This chapter discusses the challenges faced in experimental JTCs, such as various sources of non-ideality and how their impact could be mitigated. Non-linear JTC is

briefly introduced, and the impact of photodetector non-linearity on accuracy is evaluated and discussed.

Chapter 9: This chapter summarizes the key contributions of this dissertation and provides directions for future work.

CHAPTER 2

Acceleration of Convolutional Neural Networks with Programmable Free-space Optics

Neural networks play a key role in modern Artificial Intelligence (AI) technologies and are the core of many applications including computer vision, natural language processing, and smart healthcare. The size of state-of-the-art neural networks is constantly expanding to achieve even better capabilities, which increases the computation and memory requirement of the underlying hardware. Many efforts have been spent on designing domain-specific accelerators utilizing parallel architectures to accelerate neural network computation. However, because of the rapidly growing size of neural networks and the slowdown of Moore's law, the energy cost of computation and data movement makes traditional CMOS accelerators less energy efficient. Free-space optics is a promising alternative due to its massive inherent parallelism and efficient computation of Fouirer transforms and convolutions. In this work, we experimentally demonstrated a novel platform for real-time image processing using amplitude-only electro-optical convolutions with high-speed reprogrammable digital micromirror devices in a 4F optical system, achieving significant throughput and accuracy for CNNs on datasets like MNIST and CIFAR-10. We further enhanced system utilization and throughput through input tiling and high-pass filtering, showcasing the potential of free-space optics as an alternative to CMOS technology for deep learning tasks.

External collaborators:

Listed affiliations are at the time of collaboration:

- Dr. Mario Miscuglio, George Washington University
- Dr. Zibo Hu, George Washington University
- Prof. Volker J. Sorger, George Washington University

2.1 Introduction

Due to the slowdown of Moore's scaling, it is necessary to find alternative technologies to execute CNNs in a massively parallel and energy-efficient fashion other than CMOS. The intrinsic parallelism and simultaneous low energy consumption make free-space optics a particularly attractive candidate for deep-learning, computing and predominantly for real-time image classification and pattern recognition using CNNs. In this context, over the past 6 decades, optical filtering relying on spatial Fourier transform of images in the frequency domain, using converging lenses in 4F systems, has been used to extrapolate similarity (specific features) between images and signatures. Until the early seventies, the idea of realizing a real-time optical data processor was pushed to proof-of-concept demonstration, but no real-time applications were possible, due to the scarce technological advancement of the optical tools. Subsequently, the research becomes more realistic with several attempts to build optical processors [Lug74, WG66]. In recent years, due to the twilight of Moore's law and the advancement of neural networks, there has been an awakening interest in optical engines based on the 4F system for real-time deep-learning tasks. Novel miniaturized and reprogrammable optical components have ignited renewed interest in the development of optical engines as a convincing alternative to electronic implementations of convolutional neural networks. However, existing works [CSD18, CCS19, LRY18] that target neural networks either have no programmability (phase mask based) or operate extremely slow (< 100 Hz).

In this work, we experimentally demonstrated a novel platform able to perform amplitude-only (AO) electro-optical convolutions between large matrices or images using kHz-fast reprogrammable high-resolution digital micromirror devices (DMDs), based on 2 stages of Fourier Transform (FT), without using any interferometric scheme. Low-power laser light is actively patterned by commercial off-the-shelf electronically configured DMDs in both the object and Fourier plane of a 4F system, encoding information only in the amplitude of the wavefront. By individually controlling the 2 million programmable micromirrors, with a resolution depth of 8 bits and a speed of 1031 Hz (around 20 kHz with 1bit resolution), it is possible to achieve reprogrammable operations for real-time image processing with a maximum throughput of 4Petabyte/s, emulating on the same platform multiple convolutional layers of the deep CNN. We further demonstrated an optimized version of the baseline system that features input parallelization for improved system utilization and throughput.

2.2 Baseline System

2.2.1 Methods

2.2.1.1 System setup

The system architecture typology for realizing the Amplitude Only Fourier Filter (AO-FF) layer is one of the most straightforward for performing filtering in optics.13 The coherent optical image processor is based on a 4F system, in which there are four focal distances f separating the object from the image plane, intercalated by 2 Fourier transforming lenses (Figure 2.1 a). The engine is composed of 3 planes the input (object) plane, the processing (Fourier) plane, and the output (image) plane. The data to be processed comes from the input plane through electro-optic transduction. In our implementation, this transduction is achieved through a DMD. Collimated low-power laser light (633 nm HeNe) is expanded to uniformly interest the entire active area of the 1st DMD in the object plane, which, by

independently tilting each micromirror of its array according to a pre-loaded image, defines the input image (feature map). The DMD in the object plane is oriented with a 22.5-degree tilting angle with respect to the normal incidence and rotated 45 degrees in-plane. The light reflected from the DMD is Fourier transformed passing through the 1st Fourier lens at one focal length, *f*, apart from the 1st DMD in the object plane. The pattern in the 2nd DMD, with specular orientation with respect to the 1st one, acts as a spatial mask in the Fourier plane, opportunely selecting the spatial frequency components of the input image. The frequency-filtered image is inverse Fourier transformed into the real space by the 2nd Fourier lens and imaged by a high-speed camera (Figure 2.1 b).

On the system level, a computer loads the input image as well as the kernel (1920x1080, 8bit, 1000 Hz) stored in its memory to the DMDs by means of an HDMI cable or of high-speed FPGA connection, aiming to reduce the latency in providing the signals. Thus, the convolution is detected with a charge-coupled-device (CCD) camera (1000 Frames/s with 8bit resolution). (More details of the System I/O interfacing in the SOM).

2.2.1.2 Neural network design and training

In this work, we design a single-layer CNN, which includes one Fourier convolution layer followed by a pooling layer, a fully connected layer, and nonlinear thresholding. The convolutional layer has 16 Fourier filters with size of 32×32. We adopt Fourier domain training to fit our DMD-based hardware setup. All filters are real-valued filters initialized in the Fourier domain directly, and the entire convolution is computed using Fourier transforms during training. For actual inferences, the convolution layer is implemented using the optical hardware while the other layers (pooling layer and fully connected layer) are implemented in electronics. The neural network architecture is shown in Figure 2.1 c and the flow chart of the training process is shown in Figure 2.1 (d). To fully utilize the maximum update speed of the DMD, we restrict the filter values to be real and binary, therefore in the training, a custom binarization step is implemented.

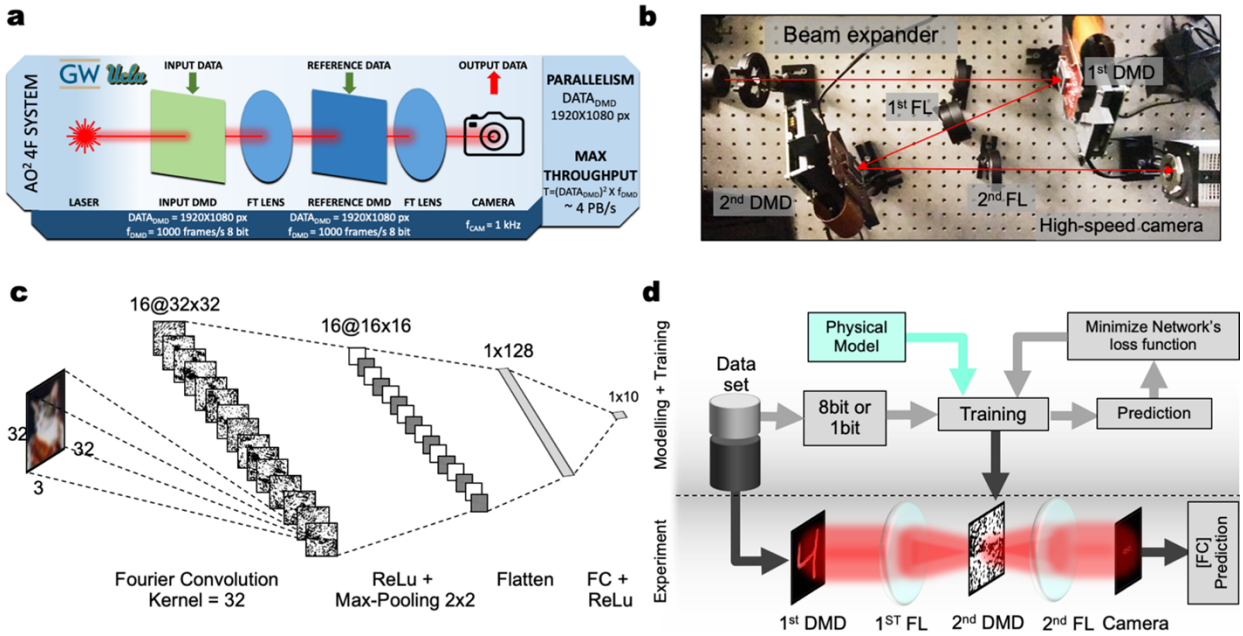


Figure 2.1: (a): Schematic representation of a 4F system based on Digital Micromirror Devices (DMDs). (b): Experimental implementation of the amplitude-only Fourier filter based on a DMD 4F system. (c): Convolutional Neural Network structure for the CIFAR-10 dataset. The optical amplitude-only Fourier filter is used as convolution layer, with the subsequent layers realized electronically. (d) Flowchart of the training process. The physical model of the amplitude-only Fourier filter layer is used for training the entire convolutional neural network, obtaining the weights for the kernel to be loaded in the 2nd DMD of the convolution layer. Experimentally obtained results of the Amplitude Only Fourier filtering are fed to the FC layer for performing the final prediction on unseen data.

Since the Fourier transform executed by the actual hardware is not ideal, if the neural network model is trained with the ideal Fourier transform, the learned filter weights might be invalid. Therefore, we develop a python-based simulation model of our hardware setup to simulate the behavior of the actual hardware and embed this simulation model into our training process. With our hardware-aware training process, the learned filters can be directly loaded onto the DMD for evaluation.

However, there is still a certain amount of discrepancies between the simulation model and the hardware outputs even though the simulation model can qualitatively simulate the hardware setup. This discrepancy can lead to undesired results if the fully connected layer's

weights trained using the simulation model are used for hardware inference. To address this issue, we implemented a fine-tuning process that uses the hardware convolution results to re-train the fully connected layer's weights. This approach proves to be useful and the tuned hardware result's accuracy shows a significant improvement compared with the one without fine-tuning.

2.2.2 Results

To evaluate the performance of our Fourier convolution system, we choose two widely used datasets, MNIST and CIFAR-10. All neural network training and evaluation are implemented using PyTorch. We compare 4 different setups, which are normal space domain convolution, our simulation model (theoretical accuracy), hardware evaluation with and without fine-tuning (experimental accuracy) respectively, and the results are shown in Figure 2.2. From the results, it is interesting that the theoretical accuracy of Fourier domain training with binary weights is almost the same as conventional space-domain convolution with full precision. This result demonstrates the capability and potential of Fourier domain training. Regarding hardware accuracy, there is a huge drop in accuracy for hardware models without fine-tuning compared to simulation results, but this difference can be compensated by the proposed fine-tuning. With fine-tuning, the hardware setup achieves the same accuracy as the simulation model for MNIST and is 8% lower for CIFAR-10.

2.3 Parallelized System

One issue of the baseline system setup is the low system utilization as only one input and filter are processed at the same time. To improve the system utilization, we adopt input tiling for parallel computation.

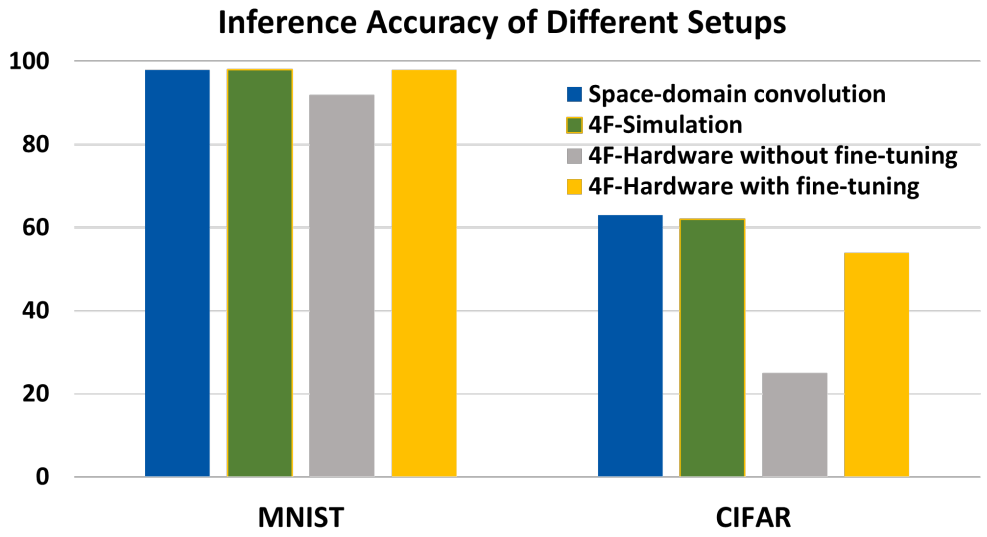


Figure 2.2: Inference accuracy of normal space domain convolution, our simulation model, hardware evaluation with and without fine-tuning.

2.3.1 Methods

We optimize the system to enhance the benefits provided by the intrinsic parallelism and fully exploit the nominal bandwidth. The input to the optical convolution layer has been parallelized so that the 5×5 matrix of CIFAR-10 or 7×7 matrix of MNIST/Google quickdraw images can be fed into the classifier. To avoid the crosstalk in the frequency domain, we spatially separate the input images with 30 pixels gap to restrict the crosstalk frequency from overlapping with the significant information carrying frequencies. The high-pass filter is applied to filter out these crosstalk frequencies and increase the misalignment tolerance. In training, this results in applying a mask on the learned Fourier kernel that sets the center 3×3 pixels to zero.

One can see a clear potential to further increase data throughput by expanding the parallel input beyond 7×7, and the parallel kernel beyond 2. However, limited by the size of the Fourier domain and diffraction limit, one would do so at the expense of resolution and classification accuracy. We analyze the physical limitations on kernel size by sweeping the resolution and estimating the corresponding classification accuracy in our simulation. We

find that the smaller the input matrix, the higher the kernel resolution, and the better the accuracy, as expected. However, this dependence becomes saturated at 2x kernel resolution in the simulation, and 1x resolution in the experiment with respect to the original size. For example, given the original size of an MNIST image being 28×28 pixels, the accuracy will be saturated at 56×56 kernel size in the simulation, and at 28×28 experimentally. This means that the Fourier transformed image size projected on DMD 2 is not a bottleneck of this D-CNN prototype. A larger input image provides for a more robust output; hence we implement $3\times$ input enlargement to mitigate the DMD 2 alignment difficulty yet giving up throughput.

2.3.2 Results

We benchmark the accuracy of different setups on three datasets (MNIST, CIFAR-10, and Google Quickdraw). The benchmarked setups consist of simulation results (without tiling), experimental results with and without tiling, and experimental results with tiling and with high-pass filter. The benchmark results are shown in Figure 2.3.

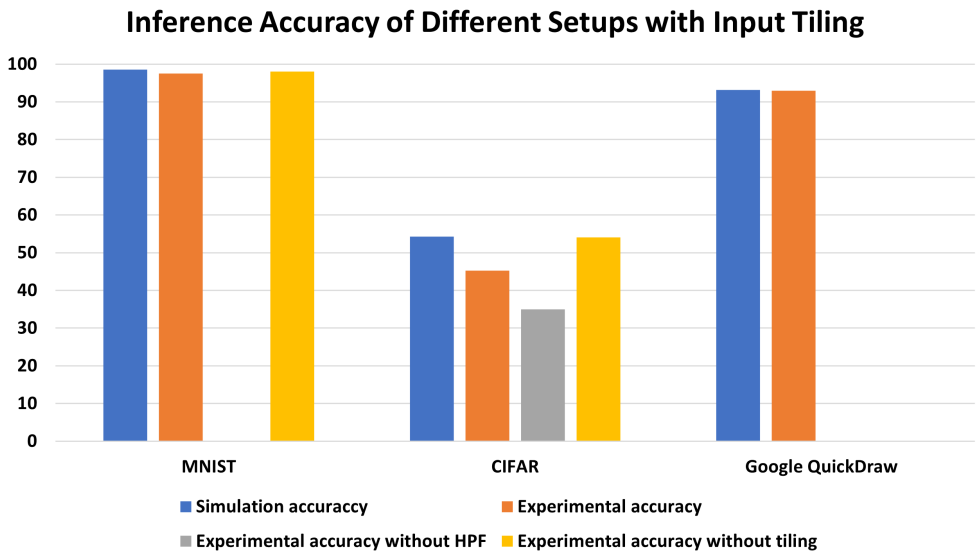


Figure 2.3: Inference accuracy of different setups.

From the CIFAR-10 results, it's clear that applying the high-pass filter can significantly improve the experimental accuracy for input tiling, though there is still s small accuracy gap compared to the experimental accuracy without tiling. The reason is that the high-pass filter cannot completely remove all the crosstalk, and it can remove some useful information as well. For easier datasets like MNIST and Google Quickdraw, the difference between experimental accuracy (with tiling) and simulation accuracy is almost negligible. In such cases, input tiling can improve the system throughput by at least 25× without accuracy drop.

2.4 Conclusion

In summary, we have demonstrated an amplitude-only electro-optic Fourier filter engine with high-speed programmability and throughput. As a proof-of-principle demonstration, we constructed an electro-optical convolutional engine for classifying handwritten digits (MNIST) and color images (CIFAR-10). We trained the network with a detailed physical model that describes the electro-optical system and its nonidealities. The proposed system achieves experimental accuracy of 98% and 54% for MNIST and CIFAR-10 respectively. We further propose input tiling with a high-pass filter to improve the overall system utilization and throughput without a significant accuracy drop. The proposed system demonstrates the feasibility and potential of accelerating convolution neural networks with programmable free-space 4F optics and paves the way for on-chip 4F accelerators.

CHAPTER 3

Optimizations and Performance Scaling Analysis of Free-space Neural Network Accelerators

Convolutional neural networks (CNNs) have achieved remarkable success in image classification and computer vision, but their deployment on traditional electronic hardware remains challenging due to the high computational complexity of convolution operations. Photonic and optical computing hardware, particularly free-space 4F systems, offer a promising alternative by leveraging the high parallelism and low latency of light-based computation. This paper explores the scalability and performance of 4F systems for CNN acceleration, proposing a novel channel tiling method that accumulates convolution results inherently in the optical domain. This approach enables the use of filters with negative weights, reduces computational overhead, and significantly improves network accuracy and throughput. Our analysis demonstrates that channel tiling can enhance the performance and accuracy of 4F systems, making them viable for high-speed, high-resolution neural network acceleration. The proposed method provides a practical solution to fully utilize the parallelism of 4F systems, addressing key challenges in optical CNN computation and bringing optical computing closer to practical deployment.

External collaborators:

Listed affiliations are at the time of collaboration:

- Dr. Mario Miscuglio, George Washington University

- Dr. Zibo Hu, George Washington University
- Prof. Volker J. Sorger, George Washington University

3.1 Introduction

Convolutional neural networks (CNNs) have proliferated in image classification and computer vision. Over the years, CNNs have become increasingly larger and deeper, making it harder to deploy such networks on traditional electronic machines due to convolution's high computation complexity. Although many efforts have been made on both algorithms and hardware to speed up the convolution process, inference of large CNNs, especially those with high-resolution inputs, is still computationally prohibitive. There is renewed and growing interest in optical/photonic computation hardware for CNN inference acceleration, due to their low compute latency (essentially time of flight of light) and the potential to support large parallelism [DCC19].

Current photonic CNN accelerators can be roughly classified into two main categories: (1) on-chip implementations using photonic devices including Mach-Zehnder Interferometers [BSS18] and micro-ring resonators [MAS18, BMM19, NDT19]; and (2) free-space 4F system, using spatial light modulators (SLM) and phase masks [CCS19, CSD18, WM19, MHL20].

On-chip photonic implementations usually have a high clock speed (in the GHz range), but the amount of parallelism is far less than the 4F system. The performance of on-chip photonic implementations is usually bounded by digital-to-analog converters' relatively low operation frequency, and they might not be efficient when dealing with high-resolution inputs. Scaling is another issue of on-chip photonic implementations since the photonic hardware scale is significantly slower than semiconductors.

In contrast, free-space 4F systems offer massive parallelism due to the high resolution of SLMs and phase masks, as well as efficient convolution computation using the well-known

4F theory, which states that convolution in the space domain is equivalent to point-wise multiplication in the Fourier domain. The 4F system can be implemented using two Fourier lenses, an input source, a device for multiplication and an output sensor. Fourier transform, point-wise multiplication and inverse Fourier transform in the 4F system is essentially constant time (the time of flight of light). The earliest optical correlators based on the 4F theory date back to the 1960s. Recently advancements in CNNs have renewed interest in 4F systems to speed up the 'expensive' convolution process. Fast SLMs and fast cameras still remain a challenge despite substantial improvements in optical 4F computing.

In this paper, we focus on improving the accuracy and performance of high-speed, high-resolution optical 4F computing systems for neural network acceleration by proposing a simple tiling method to accumulate the convolution results of all channels of a filter inherently in the optical domain, before the non-linearity applied by photodetectors/cameras. The main contributions of this work are summarized as follows.

- We provide scalability analysis and performance estimation of free-space 4F systems for CNN acceleration, which indicate that 4F systems have the potential to outperform GPUs with advanced hardware and proper algorithm and system configuration.

- We propose a channel tiling approach that allows filters to have negative weights, an implicit non-linear activation by the camera/photodetector, and optical domain accumulation of channels. We further combine channel tiling with input or filter tiling to fully leverage available optical parallelism and achieve *10-50X* utilization improvements compared with other approaches.

- Our results indicate that channel tiling dramatically improves network accuracy (*36 percent* points on CIFAR10 dataset running VGG16 network) compared to alternative tiling approaches.

- We show that channel tiling can be done without any changes to optical hardware itself unlike optical alternatives to preserve sign and it can have (at least) *2X* throughput

advantage over the recently proposed pseudo-negative approach.

- We show that channel tiling can reduce the photodetector/camera resolution requirements by orders of magnitude alleviating the main bottleneck of 4F computing systems.
- We further show that channel tiling is much more robust to the camera's quantization error and photo-detection noise (1 percent vs. *15 percent* accuracy drop compared to state of the art with 8-bit camera and 20dB SNR). This allows for channel tiling to support much faster and lower bit-depth cameras.

3.2 A Primer on 4F Optical Computing Systems

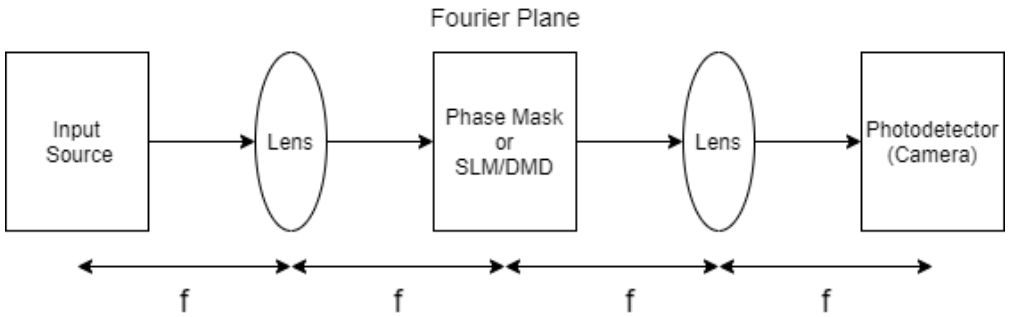


Figure 3.1: Illustration of optical 4F system.

3.2.1 An Overview of Optical 4F Computing System

A 4F optical system usually consists of 4 parts: an input source, two Fourier lenses, a filter light modulator, and an output sensor. Figure 3.1 shows the high-level system diagram of the 4F optical computing engine. The input source usually includes a laser emitter to generate coherent light and an SLM to encode the input by modifying the light intensity. Then the encoded light passes through the first Fourier lens to perform the Fourier transform. The Fourier-transformed light signal is projected onto the filter light modulator (Fourier plane), where it actively or passively modulates the light to conduct point-wise multiplication in

the Fourier domain. Afterward, the light signal that contains multiplication results passes through the second Fourier lens for inverse Fourier transform. Finally, it is captured by a high-speed camera and enters into the electronic domain.

Most published works [CSD18, CCS19] use phase masks as the filter light modulator to modulate the light in the Fourier plane passively. SLM and DMD (digital micromirror device, a special type of SLM) can also be used as filter light modulators to actively modulate the light, hence providing programmability. [MHL20] recently demonstrates the first optical 4F CNN implementation with real-time programmable filters, which uses high-speed DMD for both input and filter generation. For passive approaches, multiplication can be implemented with zero latency and power, but with no flexibility since the weights are fixed. In contrast, active approaches have more flexibility (weights can be modified) and can hence easily scale to large multi-layer networks, albeit at the cost of latency and power overheads.

3.2.2 Introduction of 4F Computing System Hardware

The field of optical neural networks including 4F-based systems is gaining interest and becoming increasingly active. However, it is still at its beginning stage with huge potential as well as many constraints and issues. Thus many works at this stage focus more on proof of concept rather than building a full, working system that outperforms state-of-art electronic systems. Even though these proof-of-concept works do not demonstrate better performance than electronics, optical computing systems still show promise to overtake electronic counterparts, especially with the help of ongoing development and evolution of advanced optic devices and materials. In this subsection, we will provide a brief introduction to the hardware that could be used in optical 4F systems and their performance estimation.

3.2.2.1 Spatial Light Modulator

Spatial light modulator (SLM) is a core component of the 4F system. It modulates the amplitude and/or phase of incoming light according to the programmed values and hence can be used for both input and filter generation. There are several types of light modulators that are widely available: liquid crystal spatial light modulators and micromirror arrays (MMA) which can be further classified into digital MMA (DMD) and analog MMA.

Liquid crystal SLM can modulate the phase and/or the intensity of light directly by adjusting the cell's refractive index. Though they can offer high resolution such as 4K [HOL] and can modulate both amplitude and phase of incoming light directly [ZW14], even high-speed SLMs operate below one kilohertz [Mea, AVR] (due to the nature of liquid crystal, adjusting the phase takes a long time). The low operating frequency makes liquid crystal SLM not suitable for CNN acceleration.

To overcome liquid crystal SLM's low operating frequency, ongoing research is trying to use alternative materials to replace liquid crystal, enabling fast operating SLM. For example, [PHS19] proposed a phase-only SLM architecture that uses microcavities with barium titanate to achieve GHz operation frequency with high-pixel resolution. If the concept can be materialized, 4F systems could potentially operate in the GHz regime and SLM will no longer be the system's bottleneck.

MMAs modulate light intensity at high speed by flipping their micromirrors to deflect input light [Sam94, Lee74]. Compared to SLM, the advantage of MMA is its high operating frequency, though it cannot modulate phase directly [MMC13, GBM14]. For *DMD*, each pixel contains a mirror and a memory unit, and the mirror flips according to the value stored in memory to let the light either pass or deflect away. Gray value modulation is implemented using a time-multiplexing technique similar to pulse width modulation, where the length of bit (duration where the mirror is on) is weighted by its corresponding power of two. The modulation result is measured by the averaging intensity over the entire multi-bit

duration. Current commercially available DMD resolution can scale up to 4K [Texb, Texa], with a nominal operating frequency of 20 to 30 KHz for binary mode [Texb].

Unlike DMDs which have a fixed micromirror tilt angle to pass or deflect the light, the tilt angle of *analog MMA's* micromirrors can be adjusted by voltage according to the desired intensity. The main idea behind such design is MMA could be treated as an optical grating, hence by adjusting the mirror tilting angle slightly, the intensity of zeroth diffraction order is also adjusted [SDD14, LDD01]. Therefore analog MMA naturally supports multi-bit mode without sacrificing operating frequency. Analog MMAs require digital to analog converters (DAC) to encode pixel values into voltages applied to MMA.[1] Analog MMAs are under active research and can achieve much higher switching speeds than commercial DMDs [SDD14, HDP08]. For instance, [HDP08] (11M pixels, 2.3MHz) and [SDD14] (2.2M pixels, 1MHz) have demonstrated high-speed, high-resolution MMAs.

Most spatial light modulators can either modulate the phase or the amplitude of incoming light, while only very few can modulate both simultaneously. However, to accurately compute convolution using 4F system, complex weight representation is desired, hence both amplitude and phase need to be modulated. Since the support of complex representation using either phase-only or amplitude-only SLM is also beneficial to many other use cases, extensive research has been conducted to solve this problem. [ZW14, WCT15] propose different methods to enable complex modulation using phase-only SLM while complex modulation using amplitude-only SLM can be addressed using the concept of Mach-Zehnder Interferometer [Mac92] or spatial encoding [GBM14, GMS18, UOO11]. Thus given either phase-only or amplitude-only SLMs, full complex modulation can be achieved using the cited techniques, hence enabling accurate convolution computation.

[1] 8bit+ resolution, MHz speed DACs are well behind the state of the art [OAN15, YZL17] and are not expected to be the bottlenecks for MMAs.

3.2.2.2 High-speed Camera

For high-speed 4F optical systems, the output sensing is usually the performance bottleneck due to the performance gap between high-speed SLM and high-speed camera. In such systems, the inputs are usually generated by high-resolution, high-speed SLM/DMD, while the multiplication is carried out using either another SLM or phase mask. As discussed, commercial SLMs can operate in 4K resolution at 20-30 KHz [Texa], with several research SLMs going to MHz/GHz regime. The operation frequency of such SLM is much higher than the state-of-the-art commercial 4K high-speed camera which operates at roughly 1 KHz [Phaa]. The main performance bottleneck of high-speed cameras is the readout time, which scales with the camera's resolution. The operating frequency of high-speed cameras can match or exceed SLM if the resolution goes down to around 1K, as there are high-speed framing cameras that can operate in the MHz or even GHz range [Spea, Speb]. But it also means the SLM's high resolution cannot be fully utilized since normally the camera and SLM should have the same resolution.

3.2.2.3 Overall System

Optical 4F system has many variations, and the overall system capability and performance depend on the system setup and devices used. In this paper we consider a programmable 4F system that uses SLMs for both input and filter modulation, and full complex modulation capability is assumed. For system performance, based on previous analysis, a 4K SLM operating at 2MHz is assumed for the system setup, which is realizable in the near future. For simplicity of performance analysis and comparison, we assume the camera can operate at the same frequency and resolution as SLMs used in the system thus the whole system can operate at 2 MHz. This frequency is not the upper bound of the 4F system since SLM could potentially operate in the GHz range, but in that case I/O might be the bottleneck of the system and complicates the assumption. Therefore GHz operating frequency is not assumed

in this paper.

3.2.3 Leveraging Massive Optical Parallelism Using Computation Tiling

Table 3.1: Table of common notations used in this paper.

Description	Notation
Input size	M × M
Filter size	N × N
SLM size	D × D
Total number of filters	N_k
Total number of input channels	N_c
Total number of inputs	N_i
Number of blocks can be tiled on SLM	T

The light modulators for free-space 4F systems usually have high resolution. Phase mask can theoretically be fabricated to any resolution and high-end commercial SLM devices have resolution up to 4K [Texa]. Take 4K SLM as an example, there are 16M pixels available to represent inputs or filter weights, which is much larger than almost any input/filter size used in CNNs. If only one input is convolved with one filter at a time, the SLMs (or phase masks) are severely under-utilized. Therefore a common approach to optimize the SLM utilization is tiling the inputs and/or filters across the SLM.

[CSD18] adopts filter tiling to fully utilize the phase mask. For standard filter tiling, the filters are tiled in the space domain and then the Fourier transform of the tiled filter is loaded onto SLM or phase mask to perform the point-wise multiplication. The tiled filters convolve with only one input, similar to broadcasting. Filters need to be zero-padded to $(M+N-1) \times (M+N-1)$ as shown in fig 3.2(a) to generate valid results. The filter blocks are then tiled over the SLM to form a single large block and convolve with zero-padded input. Fig 3.2(b)(c) illustrates the convolution process. The output captured by the camera contains the tiled convolution results of the input with all individual filters tiled on the filter SLM/phase mask, which need to be extracted in the digital domain. Input tiling can be

implemented in a similar way, but in this case inputs are tiled and convolved with a single filter. The tiled input can be directly loaded onto the input SLM, since the Fourier transform of the input is carried out by the Fourier lens.

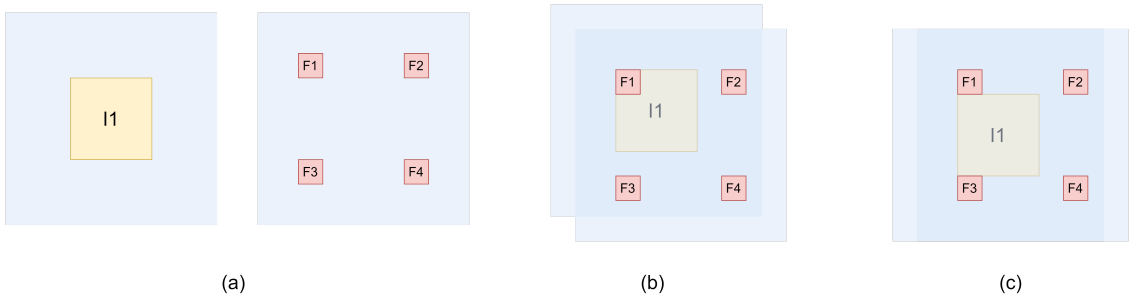


Figure 3.2: Illustration of filter tiling method. (a): Padded input block and tiled filter block. (b): Convolution visualization of filter f1 with the input. (c): Effect of zero padding. The padding between filters ensures that the input is not overlapped with two filters at the same time.

To demonstrate the 4F system's advantage of 'free Fourier transform and multiplication', we compare a 4F system's (4K SLMs and 4K camera) convolution performance against a GPU (Nvidia RTX-2080 Ti) implementation of convolution using the Nvidia CuDNN library and FP16 precision. The CuDNN library contains three types of convolution algorithms, GEMM-based algorithms, Winograd-based algorithms and FFT-based algorithms, and the optimal algorithm is selected for the given kernel/input size. Table 3.2 shows the comparison between the 4F system and CuDNN implementations for different input sizes in terms of single convolution time. Single convolution time is the average time to perform a single 2D convolution operation, calculated by

$$T_{single} = T_{total}/(N_i \times N_c \times N_k)$$

where T_{total} is the batch time for a single layer. For CuDNN implementation, batch size, number of kernels and channels are selected for optimal performance (full GPU utilization). For 4F implementation, a SLM-based system with 4K resolution and 2 MHz frequency is assumed, based on the analysis in section 3.2.2. Input tiling is applied to maximize the

Table 3.2: Comparison of single convolution time (in seconds) between 4F SLM and CuDNN implementations, taking into account the effect of tiling and parallelism.

		N=3		N=7		N=M	
Input size	N_c,N_k	CuDNN	4F	CuDNN	4F	CuDNN	4F
M=32	256,256	5.49-10	3.47e-11	2.22e-9	4.37e-11	8.21e-9	1.22e-10
M=64	256,256	2.02e-9	1.30e-10	5.47e-9	1.48e-10	4.25e-8	4.88e-10
M=128	128,128	9.12e-9	5.20e-10	1.92e-8	5.56e-10	9.56e-6	1.95e-9
M=256	128,128	3.62e-8	2.22e-9	7.41e-8	2.22e-9	3.24e-3	7.81e-9
M=512	64,64	1.93e-7	1.02e-8	6.83e-7	1.02e-8	2.29e-1	3.13e-8
M=1024	32,32	1.59e-6	5.56e-8	4.04e-6	5.56e-8	6.02e0	1.25e-7

utilization of the 4F system. Three filter sizes are evaluated, which are 3×3 (commonly used in modern CNNs), 7×7 (less common but still used in many networks) and $M \times M$ (same as input size, where 4F system's advantage is maximized). Based on table 3.2, for all input and filter sizes, a 2 MHz 4F system outperforms CuDNN implementation. The gap is larger when the filter size goes up, which is expected since the 4F system has $\mathcal{O}(1)$ complexity for convolution. For 4F implementation, the difference in convolution time for different filter sizes is trivial, only affected by the padding size (if tiling is applied), suggesting that the 4F system has more advantages on networks with large filters.

3.2.4 The Positive-only Photodetection Challenge

For any optic/photonic CNN hardware, including the 4F system, the convolution/computation output needs to be converted to the digital domain by photodetectors or cameras, which apply a square function to the results by measuring the intensity. Since the camera/photodetector readouts are all positive, the weights need to be all-positive to make the individual convolution result valid (can use a square root function to retrieve the original result). While this non-linearity could be potentially used as an activation function of CNN, it cannot be utilized on current 4F systems. Published 4F CNN hardware [CCS19, CSD18] cannot compute the convolution of input with a multi-channel filter in one pass hence they need to repeat

in a channel-by-channel manner and accumulate the partial sums in electronics (or can only process a fixed number of channels), which essentially puts the non-linearity before the channel summation and invalidates the convolution results. While positive weights might have a moderate impact on simple convolution networks with a small number of input channels for each layer, it will make the network barely functional for modern large networks with hundreds of channels for each layer. Clearly, just using positive weights is not a solution and better methods are required to make the 4F system functional. Existing works deal with this limitation either optically [TDZ17] or computationally [CSD18].

Detecting the phase information using dedicated optical hardware can remove the positive weight restriction. [TDZ17] uses balanced photodetectors with MZM (Mach-Zehnder modulator) to detect the intensity and phase for photonic neuromorphic networks, though this technique cannot be used in conventional cameras and may require precise alignment. [WM19] designs an optical ReLU module for the free-space 4F system, using SLMs and custom-built circuits. Polarization interferometry is used with a reference beam to detect the sign of each pixel and then feed the result back to SLM to let corresponding light pass through or deflect away. This method may be used to purely detect the sign information of the outputs detected by the camera for a conventional 4F system, at the cost of extra specialized hardware (at least double the number of photodetectors required and extra reference beam) and the challenge of integrating this method with conventional high-speed cameras (original design is based on SLMs).

[CSD18] introduced a 'Pseudo-negative' method that can address the positive weight restriction without additional hardware, at the cost of doubling the number of filters and consequent computation overheads. The main idea of this method is to use all positive weights for each filter to ensure valid results, but label half of the filters as positive and the other half as negative. After the sensor readout, the results of negative filters are subtracted from the results of positive filters to form the final convolution results. The advantage of this approach is the convolution results can be considered identical to normal convolution

due to the linearity of convolution (by using positive weights and taking square root after readout, the sensor non-linearity can be removed). However, the number of filters required is doubled, which means 2X weight memory and 0.5X throughput to replicate the original network.

3.3 Channel Tiling for Optical Compute Parallelism

Existing tiling approaches in optical computation overlook the fact that nearly all neural network layers in modern CNNs have multiple input channels giving another axis along which to parallelize the computation. To address the all-positive readout challenge, as well as eliminate the performance gap between SLM and high-speed cameras, we propose channel tiling (and mixed tiling) that can sum all channels inherently in the optical domain and with significantly lower output resolution requirements compared to other tiling methods. The channel tiling method tiles both input and filter channels and produces a single output, with convolution results for all channels accumulated (essentially implementing the multi-channel convolution optically).

All tiling approaches are based on the assumption that the 4F system has the ability to multiply the inputs with complex-valued weights in the Fourier domain, which can be achieved by the system introduced in 3.2.2. Thus, any real-valued filter can be transformed into the Fourier domain and multiplied with inputs without loss. The 4F system can then essentially be thought of as a black box convolution engine that can take inputs and filter each size up to its resolution. In the analysis we assume SLM with resolution $D \times D$ is used for implementing point-wise multiplication in the Fourier domain, but the method also generalizes to other devices/components.

Tiling is done in the space domain and based on cross-correlation, which is more commonly used in the field of deep learning. To apply on a 4F system that performs convolution, filters need to be flipped before doing Fourier Transform. There are two common convolution

modes ('same' and 'valid'), and they require slightly different tiling setups. For the 'same' mode where output has the same size as input, padding is essential during tiling to generate correct results. For 'valid' mode where the size of the output is $(M-N+1)\times(M-N+1)$ (see table 3.1), padding is not necessary since the correct result can be extracted by down-sampling the output. In this section, all analysis is based on the 'same' mode which includes zero padding, while the tiling approach still holds for the 'valid' mode except that padding is not required. For simplicity of analysis, actual SLM resolution is not taken into account for all tiling schemes except for mixed tiling. We validated all our analytical models of tiling (presented below) with computational experiments in a Python+Scipy+Numpy setup to confirm that tiling has no impact on the correctness of the convolution results.

3.3.1 Channel Tiling Operation

Consider the case where N_c input channels with size $M\times M$ convolve with N_c corresponding filters with size $N\times N$. The normal convolution process (same mode) can be formulated by

$$Y(i,j)=\sum_{a=0}^{N}\sum_{b=0}^{N}\sum_{c=0}^{N_c}X(a+i,b+j,c)\times F(a,b,c) \tag{3.1}$$

where X is zero padded input and F is the convolution filter. Channel tiling method tiles input and filter channels on the corresponding 2D plane thus the summation over channels in the above formula is removed. Like other schemes, the channels for both input and filter need to be zero-padded into blocks with size $(M+N-1)\times(M+N-1)$, to avoid overlap between single filter channels with multiple input channels. Then both the input channel blocks and filter channel blocks are tiled in the same order to form two large blocks X_T and F_T with size $\lceil\sqrt{N_c}\rceil(M+N-1)\times\lceil\sqrt{N_c}\rceil(M+N-1)$, as shown in fig 3.3. For simplicity, denote the size of tiled input and filter blocks as $M_t\times M_t$, where $M_t=\lceil\sqrt{N_c}\rceil(M+N-1)$.

The formula for each output activation can be formulated by

$$Y_T(i,j) = \sum_{a=0}^{M_t-1} \sum_{b=0}^{M_t-1} X_{TP}(a+i, b+j) \times F_T(a,b) \tag{3.2}$$

where X_{TP} is tiled input block X_T and circular padded to size $(2M_t - 1) \times (2M_t - 1)$, which is inherently generated by Fourier transform. F_T is the tiled filter block. The size of Y_T and F_T is $M_t \times M_t$.

When i, j are in range of $\frac{M_t-M}{2}, \frac{M_t-M}{2}$ to $\frac{M_t-M}{2} + M, \frac{M_t-M}{2} + M$, the convolution result in this region can be expressed as

$$Y_{Tvalid}(i,j) = \sum_{a=0}^{M-1} \sum_{b=0}^{M-1} X_T(a+i, b+j) \times F_T(a,b) \tag{3.3}$$

where X_T is the original tiled input block before circular padding. Since input and filter channels are tiled in the same order, each individual input channel on the tiled input block aligns with its corresponding filter on the tiled filter block in this region, as shown in Fig. 3.4 (b). The sum over channel dimension in equation 3.1 is effectively unrolled into the other two dimensions thus equation 3.3 has the same output as equation 3.1. Thus within this region, the convolution result Y_T is the standard multi-channel convolution result.

When i, j are outside of this region, as the case in Fig. 3.4 (c), the result is not valid, e.g., input channels are convolved wrong filter channels. Those invalid results make the $(M_t - 1)/2$ extra zero padding of tiled input and filter due to inherent circular padding of Fourier transform unnecessary since circular padding only corrupts the results of invalid region. Therefore in this scheme only the center $M \times M$ region of the whole $M_t \times M_t$ convolution result is equivalent to the multi-channel convolution result of all input channels and should be extracted as the final result.

The output of this tiling scheme Y_T is a single block with size $\lceil\sqrt{N_c}\rceil(M+N-1)\lceil\sqrt{N_c}\rceil(M+N-1)$ and only the center $M \times M$ region is valid. We can extract the valid region by di-

rectly selecting $y(i,j)$in range of $\frac{M_t-M}{2}$, $\frac{M_t-M}{2}$ to $\frac{M_t-M}{2}+M$, $\frac{M_t-M}{2}+M$. Fig 3.4(e) visualizes the output format. Since only the center $M \times M$ part is used as the output, the camera's resolution requirement is massively reduced to the size of a single input. This property can significantly reduce output bandwidth and improve camera readout time.

We propose this novel channel tiling scheme to carry out the channel summation inherently in the optical domain, so that it won't be affected by the camera's non-linearity. By doing so the space domain filter values no longer need to be positive only, thus the 4F system can be modeled using absolute value or square function as an activation function with unconstrained filters during training, which is not possible in other tiling schemes.

3.3.2 Mixed Tiling

Since free-space 4F systems can support high resolution up to 4K, a large number of blocks can be tiled for inputs with small sizes. For some neural network structures, SLMs/phase masks cannot be fully tiled hence causing under-utilization. Taking Alexnet [KSH12] as an example, for systems that support resolution higher than 1K, the system is far from fully utilized for any layer except the first one if channels or filters are tiled alone. To address this issue, we propose a mixed tiling scheme that combines channel tiling and filter tiling to improve the SLM utilization while still preserving the ability of channel tiling to carry out channel summation inherently in the optical domain.

A mixed tiling scheme is essentially applying two tiling schemes sequentially. The first step is applying channel tiling to tile channels of inputs and filters across a larger block. The tiled filter blocks are then further tiled across the filter SLM using the filter tiling method, but zero-padding of individual tiled filter blocks is not necessary. The output is the convolution result of a multi-channel input against multiple filters with all channel results accumulated. By doing so the SLM utilization can be vastly improved compared with the channel tiling scheme while channel summation capability is not affected.

To maximize tiling efficiency, in step one, channels should be tiled horizontally into rows to better utilize the SLM resolution. The size of padded individual channel blocks is $(M+N-1) \times (M+N-1)$, the maximum number of such blocks that can be tiled on a single SLM is denoted as T, which equals to $\lfloor \frac{D}{M+N-1} \rfloor^2$. The tiled block B has shape $D, (M+N-1) \times \lceil \frac{N_c}{\sqrt{T}} \rceil$. The condition for mixed tiling is $N_c < \frac{T}{2}$, where N_c is the total number of blocks that need to be tiled for the first tiling scheme.
If the above condition is true, B can be further tiled across all available SLM areas from top to bottom. Figure 3.5 (a) visualizes the filter plane where three filters and their channels are tiled and figure 3.5 (b) visualizes the input plane where one input's channels are tiled. Zero padding inside B blocks is not necessary, the padding within each single $(M+N-1) \times (M+N-1)$ block is enough to generate a correct result since only the invalid region (same as channel tiling) will be corrupted by overlapping of multiple filters.

The output format is a combination of the two applied tiling schemes. To extract the valid outputs, first split the raw output into results of individual filters and then extract the valid region inside each filter's result. Compared with other tiling schemes, the number of pixels required for detection is still low due to the combined channel tiling. However, since for mixed tiling the valid outputs are located at different regions, modification of the detection device is necessary to utilize the low output pixel count property in the mixed tiling case, such as a specialized controllable photodetector array. Without this, mixed tiling still delivers the required output bandwidth reduction from the camera but without a reduction in resolution requirements.

3.3.3 Tiling Efficiency Analysis and Optimization

The tiling efficiency, or how much the SLM is utilized, is determined by several factors including the number of inputs/channels/filters, their sizes, and whether padding is required or not.

For the case where only a single tiling scheme is used, the number of blocks that need to

be tiled $N_{i,c,k}$ should be larger than T. For best performance, $N_{i,c,f}$ should be multiples of T so that for each SLM update all pixels are fully utilized. The utilization rate is

$$U = \frac{M^2 \times N_{i,c,f}}{D^2 \times \lceil \frac{N_{i,c,f}}{T} \rceil}$$

The numerator in equation is the total number of pixels for all blocks that require tiling and the denominator is total number of pixels actually used, which is a multiple of D^2. When $N_{i,c,f}$ is small compared with T, the mixed tiling scheme should be adopted for optimal performance.

For the mixed tiling scheme, two tiling schemes are combined to improve SLM utilization. Again we use B to denote the tiled blocks for the first scheme, N_1 and N_2 to denote the number of blocks that need to be tiled for the first and second tiling scheme respectively (i.e., number of channels and filters). Since we assume SLMs are square-shaped, the number of individual blocks that can be tiled in one row or column is $\sqrt{T}$. Similarly, the utilization of mixed tiling is

$$U = \frac{M^2 \times N_1 \times N_2}{D^2 \times \lceil \frac{N_2}{T_B} \rceil}$$

T_B, the total number of tiled block B can be tiled across a single SLM, calculated by

$$T_B = \left\lfloor \frac{\sqrt{T}}{\lceil \frac{N_1}{\sqrt{T}} \rceil} \right\rfloor$$

3.3.4 Hardware Requirement Analysis

Table 3.3 shows the resolution requirements for different tiling schemes. Input and filter tiling are analogous to broadcasting where the untiled plane is broadcast to the tiled plane. For filter tiling (used in [CSD18] along with the pseudo-negative method), the input plane is not tiled therefore the input plane resolution is the same as a single input. For input tiling, the filter plane is not tiled, but it still requires full resolution since the filter plane is in the Fourier

domain and must have the same resolution as input to perform point-wise multiplication. Channel-tiling and mixed tiling require full resolution of input and filter plane, while having very low resolution requirements for output. Modern CNN models usually have input image resolution lower than 300×300 and are further downsampled in subsequent layers. Consider a 4K system setup with 300×300 input resolution, *channel tiling will reduce the output resolution requirement by 186 times* compared with input/filter tiling when fully tiled. Such a massive reduction in output resolution will eliminate the performance gap between the output sensing unit and input/filter SLM. This reduction holds even for mixed tiling whose output resolution requirement is N_c times less than input/filter tiling. N_c is the number of input channels in a layer and is usually between 32 to 512 depending on the exact network structure and layer.

Table 3.3: Comparison of the resolution requirements for different tiling schemes, assuming all cases are fully tiled.

Tiling schemes	Input res.	Filter res.	Output res.
None	M^2	M^2	M^2
Input tiling	D^2	D^2	D^2
Filter tiling [CSD18]	M^2	D^2	D^2
Channel tiling	D^2	D^2	M^2
Mixed tiling	D^2	D^2	$\frac{D^2}{N_c}$

3.3.5 Impact of Camera Bit-Depth and Sensing Noise

For the free-space 4F system, a high-speed camera is usually used as the output detection device and it adds two kinds of errors to the system, namely the quantization error and random sensing noise. Getting high bit-depth (i.e., precision), high resolution, high speed and low noise is a tough challenge for cameras and photodetectors. Most high-speed cameras with reasonable resolution are limited to 8 or 12-bit precision (e.g., see [Phab, Phaa]).

Due to the limited precision or bit-depth of cameras, all outputs need to be quantized to 8-bit (or 12-bit) fixed-point format. While conventional CNNs usually do not require high

precision for inference and the impact of activation quantization on accuracy may be small, the case is different for optical systems since the square function is applied to the activation during sensing which increases the dynamic range and leads to larger quantization error. The input and filter tiling methods quantize *each* channel as channel summation happens electronically after optical sensing (i.e., the partial sums themselves are quantized, not just the final activation value), thus the quantization errors are propagated during channel accumulation. In contrast, channel accumulation is carried out in the optical domain at full precision for the channel/mixed tiling and only the accumulated results are quantized to 8-bit or 12-bit, leading to a smaller overall quantization error.

Similarly, channel tiling is less susceptible to sensing noise in the camera. Sensing noise can be especially limiting for fast, high-resolution cameras needed for optical computing. Random sensing noise increases error in every channel in input/filter tiling unlike channel tiling. The error scales with the number of channels thus it impacts more for larger networks. Furthermore, camera SNR (Signal-to-noise ratio) scales with the photon flux (or number of photons captured by a pixel) [HK92a, HK92b]. Intuitively, if the physical size of the camera's sensor is fixed, then the higher resolution it supports (more pixels), the fewer photons each individual pixel will receive. As discussed, channel tiling (and mixed tiling) requires significantly less camera resolution compared to other tiling methods, which means it can have higher camera SNR than other methods.

Channel tiling inference accuracy is expected to suffer much less from camera quantization and noise. Results illustrating this benefit of channel tiling are discussed in Section 3.4.3.

3.4 Evaluation and Results

In this section, we evaluate and compare the proposed channel/mixed tiling approaches with other state-of-the-art 4F optics approaches, both in terms of performance and inference accuracy.

3.4.1 Comparing Performance of Tiling Approaches

To compare the tiling efficiency of different tiling methods, we compare them against an Nvidia RTX-2080 Ti GPU with FP16 precision using inference time on real networks. Inference time is the average time for a single input to pass through the whole network. For GPU, the batch size is set for full utilization while for the 4F system a 4K, 2 MHz SLM-based system is assumed. Only convolution layers are benchmarked. We pick VGG16 [SZ14] (with CIFAR, ImageNet and SpaceNet [VLB18]) and AlexNet [KSH12] (with ImageNet) as representative benchmarks. Besides these two widely used CNN architectures for image classification, we also include two other networks for image super-resolution and de-convolution. For image super-resolution, the SRCNN [DLH16] is used, which consists of two convolution layers with filter size 9×9 and 5×5 and the target image resolution is set to 512×512. The Deconv Net [XRL14] is used for image de-convolution and contains five convolution layers with large filter sizes ($121 \times 1, 1 \times 121, 16 \times 16, 1 \times 1$, and 8×8 respectively).

The 2MHz SLM system is faster than GPU in all cases and is as much as 61.7X faster for the SRCNN case, which is better suited to the 4F system given its larger kernel sizes. For most conventional neural network architectures with relatively small filter sizes (e.g., AlexNet and VGG), the speedup is around 20X. It is interesting to compare VGG16 performance on CIFAR10 vs. ImageNet. The smaller input size (32×32 vs. 227×227) of the CIFAR10 dataset leads to severe underutilization of the 4K SLM, especially in later layers of the network. This indicates that smaller (and therefore potentially cheaper, faster SLMs) may be a better design point for small input networks.

Since the main goal of this paper is to focus on bringing 4F optical computing closer to reality, the complete analysis of energy and power is out of the scope of this paper. The analysis of power and energy of various photonic and optical neural network implementations can be found in [DCC19].

Table 3.4: Overall network inference time in seconds (per input) for different tiling schemes and network architectures. The results are for convolution layers only. Note for DeconvNet there's a convolution layer of 1x1 filters which is not suitable for 4F system acceleration and is not taken into account for the estimation.

Network-dataset	GPU	Channel tiling	Mixed tiling	No Tiling	Best speedup
VGG16-CIFAR10	5.07e-5	1.98e-3	6e-6	8.17e-1	**8.45**
VGG16-ImageNet	1.41e-3	1.98e-3	6e-5	8.17e-1	**23.50**
AlexNet-ImageNet	1.31e-4	6.88e-4	7e-6	1.84e-1	**18.71**
VGG16-SpaceNet7	1.79e-2	2.27e-3	1.06e-3	8.17e-1	**16.89**
DeconvNet	3.76e-4	3.35e-4	8e-6	1.02e-2	**47.00**
SRCNN	1.48e-3	9.60e-5	2.4e-5	2.88e-4	**61.67**

3.4.2 Impact of Tiling Approaches on Network Accuracy

As discussed previously, different tiling schemes impose different restrictions on the network: input/filter tiling places a square function on each channel's convolution result before summation, while channel/mixed tiling restricts the activation function to be an absolute value function (taking a square root after camera readout). The effect of these restrictions on network-level accuracy is reported in table 3.5, evaluating three datasets trained with a VGG-16 like model. The camera is assumed to have unlimited precision (addressed in the next section). All-positive filters are not used in input/filter tiling methods as they effectively nullify the purpose of activation and make deep CNNs like VGG-16 extremely hard to train properly since there are (almost always) multiple input channels. Optical compensation approaches (e.g., the sign detection proposed in [WM19]) and pseudo-negative [CSD18], in the absence of any optical non-idealities, should give the same accuracy as standard convolution. For the proposed channel/mixed tiling, the network is trained with an absolute value function applied to convolution results. All the results reported in table 3.5 are trained from scratch with floating point precision.

The results clearly show that input or filter tiling is unacceptably inaccurate for anything but the simplest of classification tasks. Our proposed channel and mixed tiling approaches lose <3 percent accuracy compared to unconstrained electronic implementations or pseudo-negative approaches (which use twice as many filters). This small gap in accuracy can be bridged in the future by better optical non-linearities (an active area of research [WHM20, GMA18]), improved training methods (e.g. different regularization terms during training to incentivize positive filter weights) or combining with the pseudo-negative approach [CSD18], all are part of our ongoing work but not explored further in this paper.

Table 3.5: Comparison of the accuracy of different datasets trained using VGG16 with different methods. For Input and filter tiling the absolute value function is applied to each individual channel's convolution result while for channel and mixed tiling the absolute value function is applied after channel summation and acts as the activation function. For pseudo-negative cases, positive weights are used and subtraction is implemented after camera detection, then ReLU is applied on subtracted results as the activation function.

Method	Fashion MNIST	SVHN	CIFAR10
Input/Filter Tiling	75.4%	78.5%	55.6%
Channel/mixed Tiling	93.2%	95.1%	89.3%
Pseudo negative	93.6%	95.1%	91.6%

3.4.3 Impact of Camera Limitations on Inference Accuracy

The previous accuracy results are ideal cases and do not consider camera bit-precision and sensing noise. To simulate quantization error due to the camera's limited bit-precision, the square function is applied to the convolution results for simulating the intensity measurement and then the results are then quantized to scaled 8-bit and 12-bit format (256/4096 uniform intervals). The square root is taken on the quantized results to get the absolute value. We simulate the sensing noise using white Gaussian noise with average SNR at 15dB, 20dB and 30dB [HX17]. For both cases only partial sum and activations are quantized, while weights

as assumed to be floating point precision. [2]

Figure 3.6 and figure 3.7 show the Fashion MNIST and CIFAR-10 classification accuracy using VGG-16 for pseudo-negative with filter tiling and the proposed channel/mixed tiling, taking into account camera quantization error and different level of random noise. The results clearly show that channel/mixed tiling is far more robust to both quantization and sensing noise due to its error-free channel summation. The pseudo-negative (filter tiling) approach requires at least 12-bit camera precision as opposed to 8-bit for channel/mixed tiling to remain within *5 percent* accuracy drop from full precision. Interestingly, once the camera bit-depth is taken into account, channel tiling always has higher accuracy than a pseudo-negative approach despite being somewhat more restrictive.

For cases with random noise, the accuracy of the proposed channel/mixed tiling method is higher than the pseudo-negative method for almost all cases. 30dB or 20dB SNR is good enough for channel tiling accuracy to be within *5 percent* of the noiseless case while other tiling approaches need at least 30dB SNR. Moreover, the achievable SNR for channel/mixed tiling can be higher than filter tiling since it requires lower output resolution, making channel/mixed tiling more robust to sensing noise.

Altogether, our results indicate that the proposed channel tiling approach can reduce required camera precision by *33 percent* (8-bit vs. 12-bit) and improve noise tolerance by *10dB*. Though we do not explore it here, such relaxation can substantially improve the speed, energy, and cost of sensing in 4F computing systems.

[2] We do not use high SNR values as high resolution, high-speed cameras are likely to have higher photon noise [HX17].

3.5 Discussion

3.5.1 Experimental Realization of Programmable 4F System

One primary assumption in this paper is an SLM-based, programmable 4F system. The experimental verification is not the focus of this paper, and the realization of such system has been demonstrated by our recent work [MHL20]. In that work a prototype of a DMD-based high-speed programmable 4F system has been built with KHz operating frequency, and is able to get decent inference accuracy for a single layer CNN. Figure 3.8 shows the photo of the experimental setup of the programmable 4F system. Unlike passive 4F systems where filters are fixed, an active 4F system requires filter weights to feed into the DMD at high speed consistently, thus high-speed I/O and synchronization between SLM and camera become challenges to the system. An FPGA-based unified high-speed I/O interface is designed to manage the I/O and control of the whole 4F system. Besides, a training-compatible simulation model is designed to accurately model the 4F system during network training while fine-tuning is applied to compensate for various system non-idealities, addressing two other common issues of 4F systems.

This experimental realization of a programmable 4F system presents a way to scale the 4F system and make it target a broader aspect of applications. As discussed in section 3.2.2, the SLM modulation speed is continuously improving with new techniques and materials, and the trend is highly encouraging: although commercially available SLM only operates in KHz range [Texb, Texa], GHz SLM [PHS19] is with-in reach with advanced materials. The advancement of SLM technology makes the 4F computing system promising, as it scales better than conventional electronic computing systems.

3.5.2 Optimal Filter Size and Architecture Search

One major trend in CNN architecture is making the filter size (pixel size) as small as possible, 5×5 and larger filters are rarely seen in recent years. This trend is fully understandable, as almost all current CNNs need to be trained and implemented on electronic systems, whose computation time scales with filter size. Using a small filter size with a deeper structure will make the neural network a lot faster and still have a similar receptive field size. Following this trend, many researchers developed various successful CNN architectures with very small filter sizes and making a small filter size become the de-facto choice for CNN architectures.

However, it is a completely different story for 4F systems. Based on the complexity analysis and experiment results from table 3.2, increasing filter size has a trivial impact on convolution speed for 4F systems, which indicates that larger filters may be more suitable for 4F systems as its almost 'free'. Unfortunately, due to the dominance of electronics in the field of computer vision, the potential of large filters has not been thoroughly explored, due to this computational inefficiency. We argue that large filter sizes should be of interest in many use cases if the computation efficiency burden is removed. For example, the image resolution used in computer vision tasks is constantly growing: A smartphone photo can have 4K resolution and some other applications like medical images or satellite images have much higher resolution. If those high-resolution images are fed into a model designed for medium-resolution images, without proper scaling, a 3×3 filter will capture way less information than intended and may not be the optimal filter size anymore. Thus, the advancement of 4F systems enables a new CNN architecture search direction: a shallower network with large filters. In this region, 4F systems could operate significantly more efficiently than electronic systems, and thus be able to support architectures that are not possible using electronic systems due to performance reasons.

3.6 Conclusion

In this work, we provide scalability analysis and performance estimation of 4F optical systems on CNN acceleration, along with related issues. We then propose the channel tiling and mixed tiling methods for optical 4F systems to boost performance and accuracy, without extra hardware or computation. By utilizing the properties of convolution and 4F systems' high resolution, channel tiling, and mixed tiling make 4F systems able to accumulate all channel's convolution results in the optical domain, thus bypassing various constraints applied by output sensing. Compared to the recent pseudo-negative approach with filter tiling [CSD18], our method gives similar accuracy (<3 percent difference on three datasets), significantly better robustness to sensing quantization error (33 percent improvement in required sensing precision) and noise (10dB reduction in tolerable sensing noise), 0.5X total filters required, 10-50X+ throughput improvement and at least 3X reduction in required output camera resolution/bandwidth. The proposed channel tiling and mixed tiling methods provide a simple and practical way to fully utilize the massive parallelism inherent in 4F optical computing systems to accelerate CNNs and bring them closer to reality.

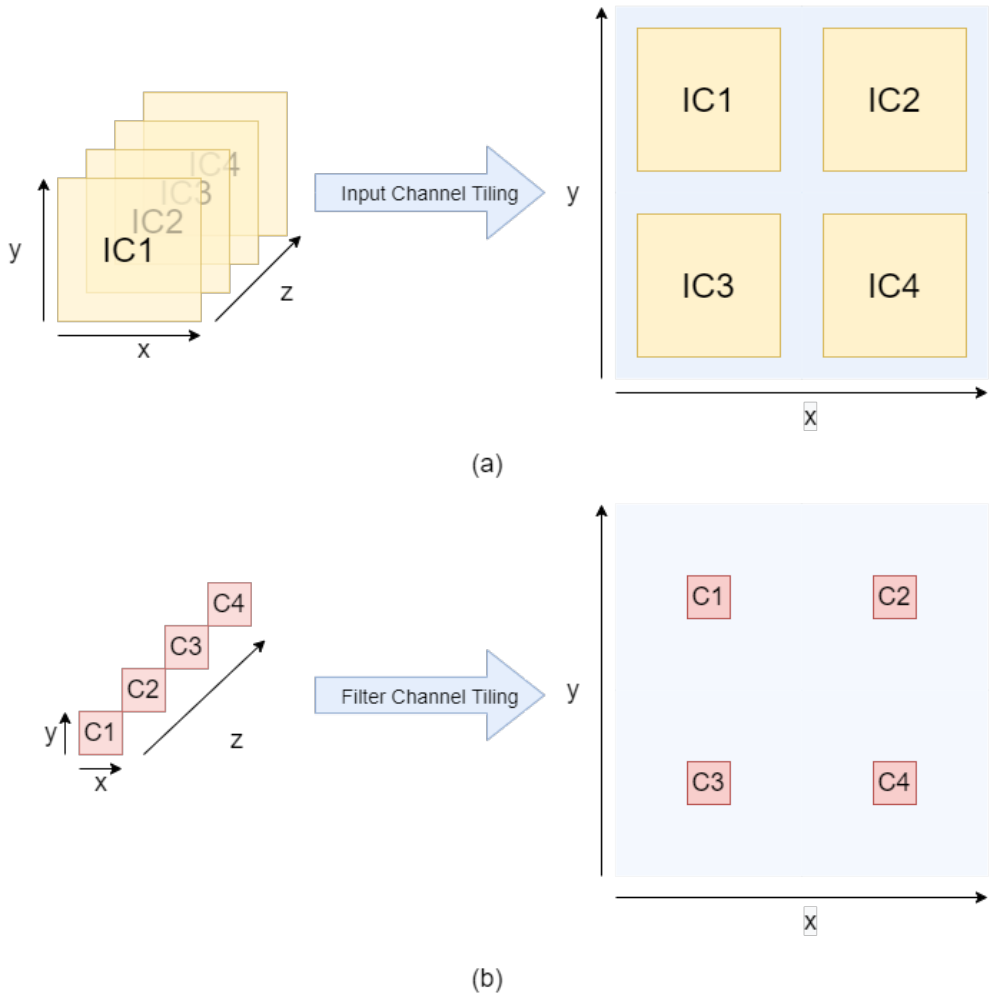


Figure 3.3: Illustration of the tiling process of channel tiling, which effectively unrolls the channel dimension. Blue regions represent zero padding. (a): Input channel tiling. (b): Filter channel tiling.

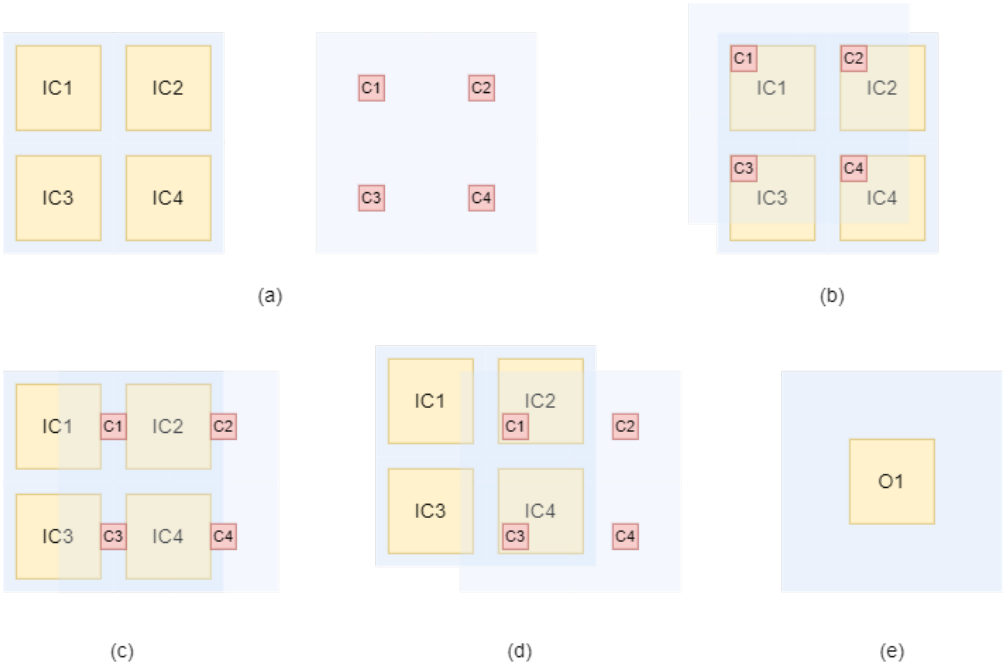


Figure 3.4: Illustration of the convolution process, blue regions represent zero padding. (a): Tiled input and filter blocks. (b): Start point of the valid region. (c): Effect of padding, filters will not overlap with multiple input channels. (d): Example of the invalid region, filters are not convolved with their corresponding input channels. (e): Output format of channel tiling.

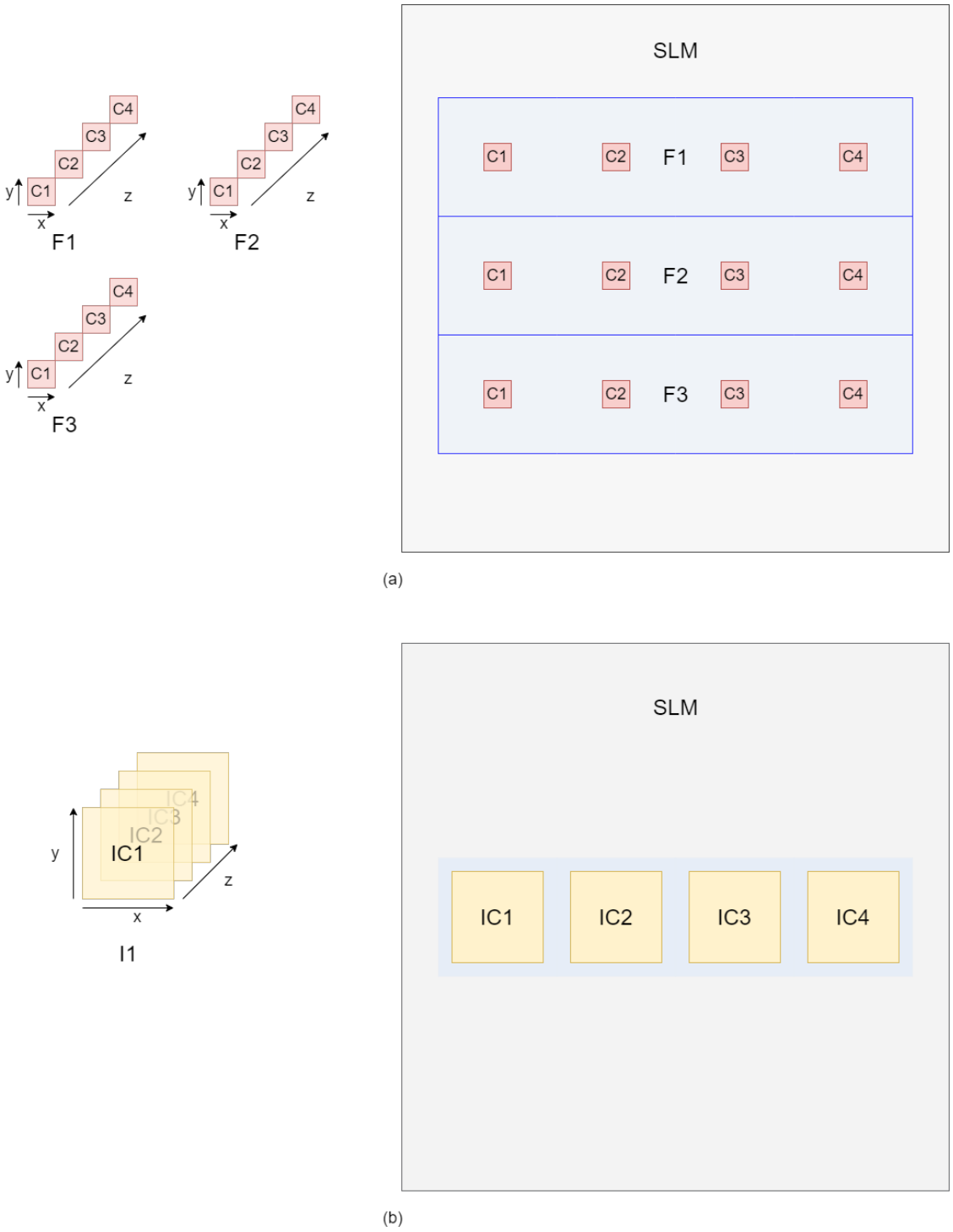


Figure 3.5: Illustration of the tiling process of mixed tiling. Blue regions represent zero padding. (a): Filter channel tiling, multiple filters' channels are tiled on SLM. (b): Input channel tiling, only a single input's channels are tilied on SLM, following the same order as filter channels.

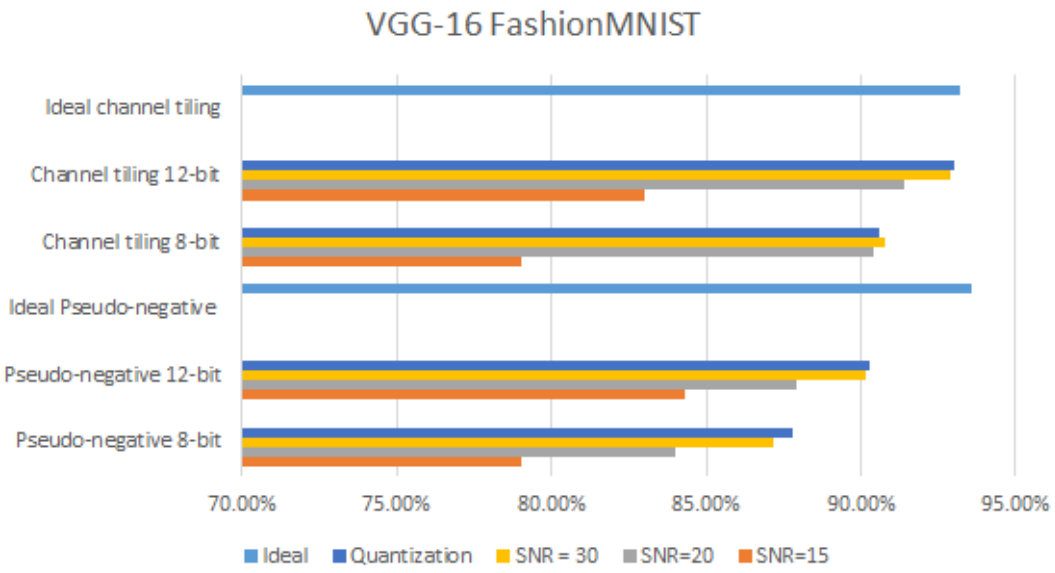


Figure 3.6: Accuracy of FashionMNIST dataset using different setups.

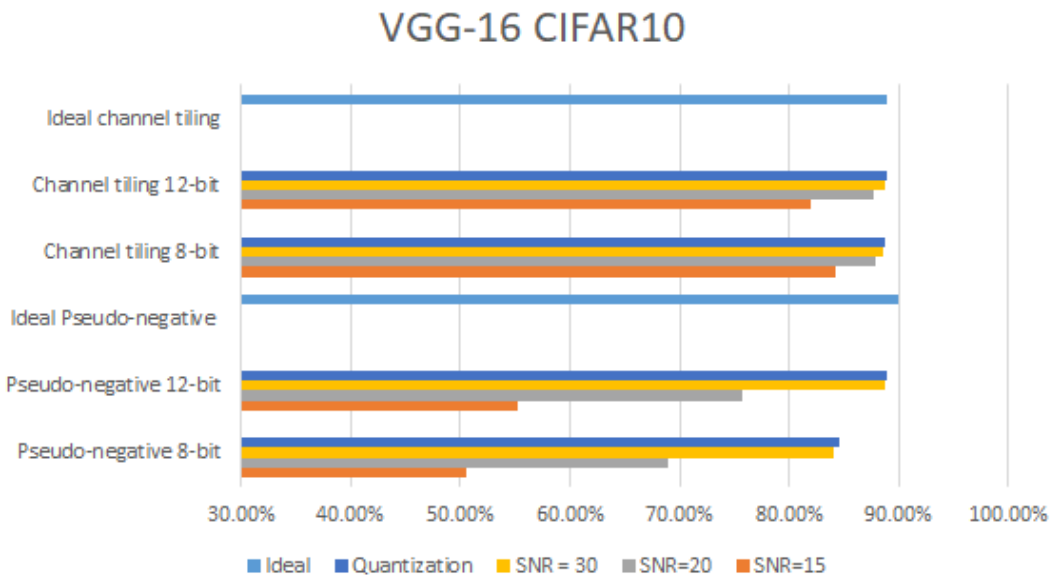


Figure 3.7: Accuracy of CIFAR-10 dataset using different setups.

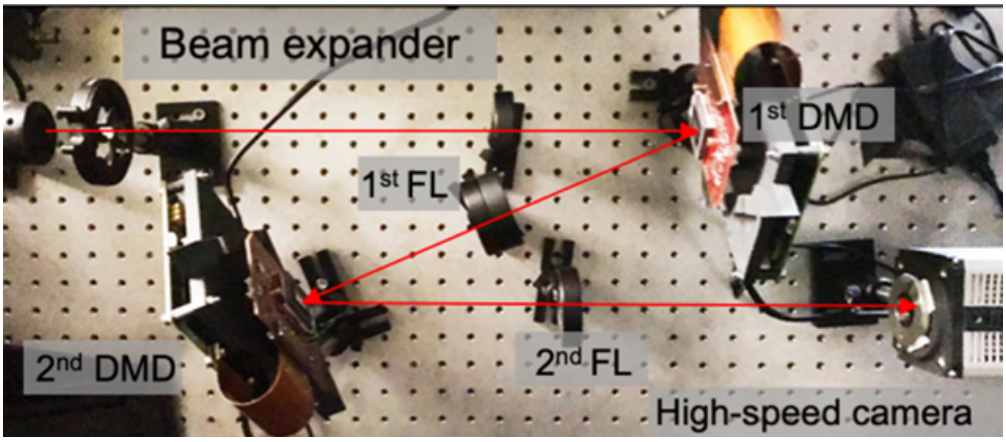


Figure 3.8: Experimental setup of the DMD-based programmable 4F system.

CHAPTER 4

PhotoFourier: A Photonic Joint Transform Correlator-based Neural Network Accelerator

The last few years have seen a lot of work to address the challenge of low-latency and high-throughput convolutional neural network inference. Integrated photonics has the potential to dramatically accelerate neural networks because of its low-latency nature. Combined with the concept of Joint Transform Correlator (JTC), the computationally expensive convolution functions can be computed instantaneously (time of flight of light) with almost no cost. This 'free' convolution computation provides the theoretical basis of the proposed PhotoFourier JTC-based CNN accelerator. PhotoFourier addresses a myriad of challenges posed by on-chip photonic computing in the Fourier domain including 1D lenses and high-cost optoelectronic conversions. The proposed PhotoFourier accelerator achieves more than 28× better energy-delay product compared to state-of-art photonic neural network accelerators.

External collaborators:

Listed affiliations are at the time of collaboration:

- Dr. Hangbo Yang, UCLA
- Dr. Nicola Peserico, George Washington University
- Prof. Volker J. Sorger, George Washington University

4.1 Introduction

Convolutional neural networks (CNNs) play a key role in modern Artificial Intelligence (AI) technologies and are the core of many computer vision applications including image classification [KSH12, SZ14, HZR16], object tracking [RDG16, GDD13], medical imaging [GWL18, GJN18], etc. Over the past decade, there have been many efforts to design domain-specific accelerators utilizing parallel architectures to accelerate the computation of neural networks in an energy-efficient way [CES16, CDS14, SZW18, ADJ17]. However, the rapidly growing size of modern CNNs and the slowdown of Moore's law have limited CMOS digital accelerators in terms of the energy cost of data movement and computation [CES16, LNM17]. Silicon photonics has emerged as a promising approach to deliver massive compute parallelism and high efficiency [SKB21, LLY19, SWK20]. Photonic components can easily operate above 10 GHz while still being relatively low-power [MLW17, SLM21], and photonic waveguides do not suffer from RC delay or energy losses [LSZ15, RMN20]. These features give photonics an unmatched advantage in low-latency and low-power computation.

Photonic neural network accelerators can be roughly classified into two main categories: Mach-Zehnder Interferometer (MZI) and micro-ring resonator (MRR) based dot product accelerators [SWK20, SKB21, BMM19, BSS18, MAS18, LLY19, ZLY20, GZF20, SMN21] and Fourier optics-based convolution accelerator [CSD18, MHL20, GL22]. Most MZI/MRR dot product accelerators resemble compute-in-memory analog accelerators [SNM16, AUO17, AHR18], but with high clock frequencies (5-10 GHz). The large number of large-sized MZIs and/or MRRs required can become a problem. On the other hand, Fourier optics-based designs typically utilize the convolution theorem to accelerate the convolution operation, which states that convolution in the space domain is equivalent to point-wise multiplication in the Fourier domain. Such systems, typically called 4F systems (total system length is 4 times the focal length of the lens), leverage time-of-flight (and passive, hence, zero energy) Fourier transform using Fourier lenses to reduce the complexity of convolution from $O(N^2)$

to just $O(N)$. A point-wise multiplication unit is required at the Fourier plane (after the Fourier transform) and the filter weights are directly loaded into the multiplication unit [MHL20]. Theoretically, compared to dot product accelerators, 4F systems can perform the same computation with significantly fewer optical components because of the complexity reduction. However, 4F systems require Fourier domain filters that are complex-valued, with sizes the same as inputs. This constraint makes 4F systems harder to implement as supporting complex multiplication is hard. Moreover, it makes 4F systems less efficient when executing conventional CNNs, which typically use 3×3 real-valued filters. All prior works on 4F-based CNN accelerator are prototypes using free-space optics [MHL20, CSD18], which are slow and bulky compared to on-chip photonics.

In this work, we propose using Joint Transform Correlator (JTC) to accelerate CNNs by reducing the computation complexity through Fourier optics, while addressing the issues faced by typical 4F systems. JTC is a variant of Fourier optics that computes the auto-convolution of two input signals using a pair of Fourier lenses. Just like 4F systems, JTC also takes the advantage of the 'free' Fourier transform but uses spatial filters instead of complex-valued Fourier filters. Therefore, JTC systems allow filters to be smaller than inputs and only need to support real-valued multiplication.

In this paper, we present PhotoFourier, a photonic CNN accelerator based on Joint Transform Correlator (JTC). The main contributions can be summarized as follows:

- We propose the row tiling/partitioning algorithm to implement 2D convolutions using 1D on-chip lenses.
- We develop a temporal accumulation approach to cut down Analog-to-Digital Converter (ADC) power by 16X and improve neural network accuracy significantly.
- To the best of our knowledge, this is the first work to propose the architecture design of an on-chip Fourier-optics based photonic neural network accelerator. PhotoFourier can achieve as much as 28× better energy-delay product compared to state-of-art photonic

neural network accelerators.

4.2 A Primer on the JTC system

4.2.1 Background of JTC

JTC has been widely used for many applications including optical encryption [NJ00, RBH09, VMP13], image filtering [TS98, Jav90], and object tracking [TFG90, LA04, LKK01] over the past two decades. Recently there has been a growing interest in optical and photonic neural networks, with some works trying to realize JTC-based optical neural networks. [GSY22, GLF11, HQS14, Col18] provides theoretical analysis and experimental demonstration of a free-space JTC system designed for low latency convolution operations while [YLM22] demonstrates the concept of a basic on-chip JTC-based photonic neural network.

In physics, an optical lens can achieve Fourier transform $\mathcal{F}[\tilde{E}(x, y, f)]$ [Goo05] on its back focal plane if an input image $\tilde{E}(x, y, f)$ illuminated by a coherent light (usually a laser) is at the front focal plane of the lens. $\tilde{E}$ is the amplitude of the light at the front focal plane, $\mathcal{F}$ is the symbol of the Fourier transform. Adopting the Fourier transform of the lens, [WG66] first made an optical JTC to generate the optical convolution with both phase and amplitude. Based on the traditional 2D optical JTC, a baseline 1D on-chip photonic JTC can be built with slight modifications. Figure 4.1 (a) depicts the layout of a baseline on-chip JTC system, which consists of five key components: (1) a 1D multi-channel input beam with a signal $s(x + x_s)$ and a kernel $k(x - x_k)$ (where x_s and x_k are offsets of s and k from the global origin in x direction, respectively) passes through (2) the first on-chip metasurface-base lens functioning as a traditional free-space lens, to achieve 1D Fourier transform $\mathcal{F}\left[s\left(x + x_s\right) + k\left(x - x_k\right)\right]$, (3) a *nonlinear function* component implemented using photodetectors (to transfer optical signals to electrical signals meanwhile achieve a *square function*) and electro-optic modulators (EOM) (to transfer electrical signals back to optical signals), (4) the second on-chip metasurface-base lens, and arrives at (5) photodetectors

recording the intensity pattern of the convolution computed by the JTC:

$$s(x + x_s + x_k) * k(-x) + s(-x) * k(x - x_s - x_k) + O(x) \tag{4.1}$$

, where $*$ means convolution, $O(x) = \mathcal{F}\left[|S(x)|^2 + |K(x)|^2\right]$. The first and second terms are the computed auto-convolution between the two inputs whereas the third term $O(x)$ is a non-convolution term. The convolution terms in Equation 5.1 can be shifted off the center non-convolution term $O(x)$ by adjusting the distance between two inputs, so that the convolution would not be affected by the non-convolution term $O(x)$. The photodetectors only need to detect one of the convolution terms to get the convolution result. To demonstrate this, we simulate the JTC output of a 256-element input which is a partitioned and tiled CIFAR-10 input, with a tiled convolution kernel (refer to Section 4.3 for tiling details), and the JTC output is shown in Figure 4.2. The simulated output clearly shows the three terms in the output are spatially separated with no overlap.

The non-linear function in JTC, applied in the frequency domain after the first lens, is essential to compute the convolution since without it the output will be the same as the input (Fourier transform followed by inverse Fourier transform). In the baseline system, the non-linear function is a square function achieved by photodetectors. One photodetector and one MRR are required for each waveguide. EOMs in this design are tunable MRRs[MLW17], which transfer electrical signals back to optical signals. MRRs that implement the square function can be directly controlled by the output of the photodetectors, without conversion between analog and digital domains. The distinction between on-chip JTC and conventional free-space JTC is that 2D lenses are replaced with 1D on-chip lenses, hence 1D convolutions are computed instead of 2D convolutions.

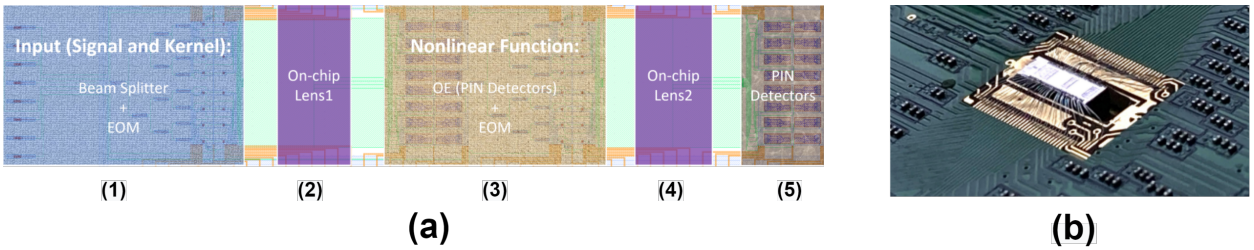


Figure 4.1: (a): The annotated layout diagram of a baseline on-chip JTC system. (b): The PCB photo of the fabricated prototype.

4.2.2 A JTC accelerator prototype

We have designed and fabricated a prototype of the baseline system, which is the first on-chip JTC system. Figure 4.1 (b) shows the fabricated JTC chip inside a PCB. The detailed experimental evaluation of the prototype system is out of the scope of this paper, as we focus on the architecture design and analysis of an *upscaled* system. Still, the prototype system demonstrates that on-chip JTC systems are suitable and realizable in terms of accelerating CNNs.

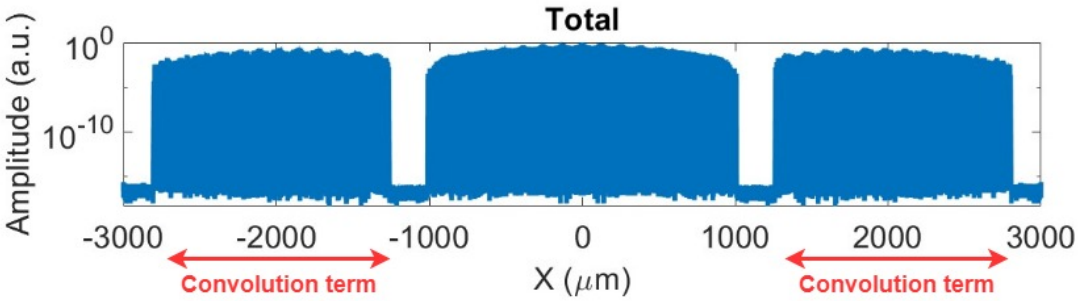


Figure 4.2: Simulated JTC output for a 256-element input (partitioned from a CIFAR-10 input) with tiled convolution kernels.

4.2.3 Issues faced by on-chip JTC accelerators

The advantage of reducing the complexity of convolution operation without adding weight bandwidth overhead makes JTC a potentially better candidate than other photonic systems for efficiently accelerating CNNs. However, there are still many challenges that need to be

addressed. Some issues are faced by photonic accelerators in general while others are specific to on-chip JTC accelerators.

4.2.3.1 1D lens

Being on-chip means the lenses can only be one-dimensional, hence only 1D Fourier transform is supported and results in 1D convolution. Most CNNs use 2D convolution to capture information on both x and y dimensions. Clearly, just using 1D convolution will lead to poor accuracy and make JTC systems not able to execute conventional CNNs. To overcome this challenge, we propose the row tiling method to approximate 2D convolutions with 1D convolutions accurately.

4.2.3.2 Component redundancy

The baseline JTC system described in Section 4.2.2 can be split into two identical parts. Each part contains a set of MRRs, Fourier lens, and photodetectors. When processing a convolution, both parts can not be utilized at the same time, resulting in a 50% utilization. Such inefficiency leads to potential optimizations including pipelining the system, which will be discussed in Section 4.4.

4.2.3.3 Overhead of the non-linear function implementation

A baseline JTC system uses MRRs to implement the required non-linear function, which results in undesired power and area overhead. In fact, the non-linear function could be implemented passively using optical non-linear materials, which can massively reduce the total number of active photonic components. Promising research results have been reported on optical non-linear materials [AGW21, NON15, ADB16] and JTC systems with non-linear materials [KKA94, VMP20, GSY22]. The reason for not using non-linear materials in the baseline JTC system is that such materials are not mature enough to be fabricated with

silicon photonics. However, in the near future, passive non-linear materials could be used to implement the non-linear function, making designs more power efficient.

4.2.3.4 O-E and E-O conversion overhead

Theoretically, the power efficiency of photonic accelerators should be an advantage over digital accelerators but the components required for O-E and E-O conversions (ADC, DAC, and modulators) are active and draw a large amount of power. If the architecture is not carefully designed to compensate for the conversion overhead, the overall power efficiency of photonic accelerators can easily be worse than CMOS accelerators. A key part of our architecture and dataflow design is to minimize the number of O-E and E-O conversions for optimal power efficiency.

4.2.3.5 Mismatch between the frequency of photonics and CMOS

One advantage of silicon photonics is that they can be clocked extremely fast. Optical components like MRRs can operate above 30 GHz. Most existing photonic neural network accelerators set clock frequency between 5 to 10 GHz. However, it is extremely challenging to design and fabricate CMOS components with a 10 GHz frequency. CMOS circuit is required to generate inputs, receive outputs, communicate with memory, and compute operations that the photonic accelerator is not able to compute. How to address the frequency mismatch between CMOS circuits and photonics is a challenge for realistic photonic neural network accelerator designs.

4.3 2D Convolution Computation on JTC

As discussed in Section 4.2, an on-chip JTC can compute the convolution between two inputs, but is limited to 1D, while most CNNs use 2D convolutions. To address this issue,

we propose a generic algorithm to compute 2D convolution using 1D convolution, which can be applied to any hardware that supports 1D convolution, including JTC systems.

The key idea of the proposed algorithm is row tiling (and partitioning), where the rows of 2D inputs and kernels are tiled to form 1D inputs and kernels for 1D convolution. The proposed algorithm can achieve identical results as 2D convolutions in 'valid' mode (without zero padding, output size smaller than input size), and can closely approximate 2D convolutions in 'same' mode (with zero padding, output size same as input size). In the rest analysis, we assume 2D convolution uses the 'same' mode, which is more common. Assuming a 2D input has size $S_i \times S_i$, a 2D kernel has size $S_k \times S_k$, and the maximum 1D convolution size supported N_{conv}. Depending on S_i, S_k, and N_{conv}, the algorithm is split into three variations.

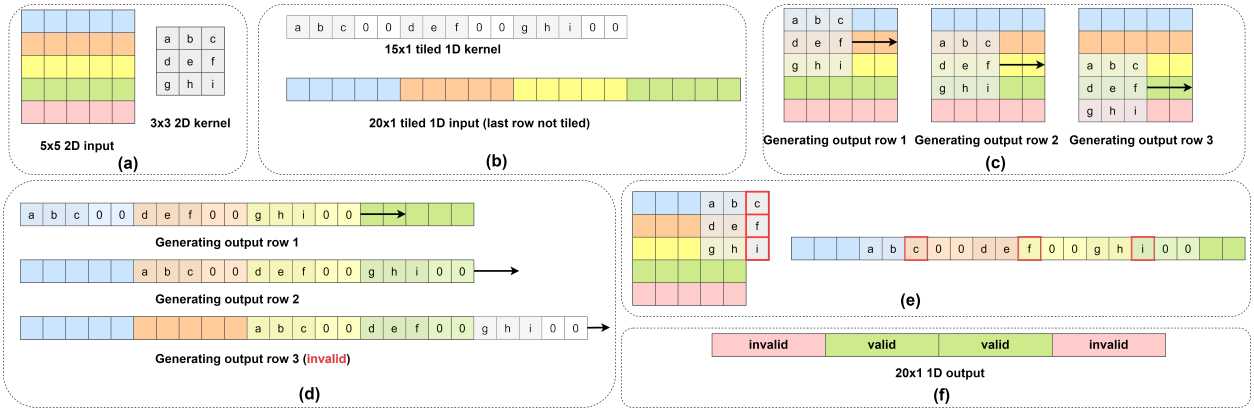


Figure 4.3: Visualization of row tiling with an example of 5×5 input, 3×3 kernel, and maximum 1D convolution size of 20. Different rows of the input are represented using different colors. (a): 2D input and kernel. (b): Tiled 1D input and kernel. Kernel rows are zero-padded to match the input row size. The last row of input is not tiled due to the limit of 1D convolution size. (c): Sliding window convolution process of normal 2D convolution to produce rows 1-3 of the output. (d): Sliding window convolution process of 1D convolution. For the first two rows, the tiled kernel rows are aligned with their corresponding input rows and produce valid results. Row 3 illustrates the case where the tiled kernel 'slides' outside the input, and generates invalid results (since input row 5 is not tiled). (e): Edge effect. For 2D convolution with zero-padding when the filter is sliding outside of the inputs, the part outside the input (c, f, j) will convolve with zero. However, for 1D convolution they will convolve with the next input row, producing different results compared to 2D convolution. (f): Output format of 1D convolution. Invalid results are marked with red color.

4.3.1 Row tiling

Row tiling can be applied when $N_{conv} > S_k \times S_i$, which is the most common case. The process is best explained with the visualizations of sliding window convolution, which are shown in Figure 4.3. The first step is to tile the rows of the 2D input and kernel. The number of input rows that can be tiled each time is $\left\lfloor \frac{N_{conv}}{s_i} \right\rfloor$, which depends on the maximum size of 1D convolution. All kernel rows are tiled, but they are separated by zero-padding with size $S_i - S_k$ to ensure input and kernel rows are aligned after tiling (Figure 4.3 (b)). Zeros are added to the end of tiled input and kernel rows, to make both of them have length N_{conv}. For conventional CNNs, the 2D convolution process can be visualized by sliding the kernel over the input, and in each step the overlapped regions between the kernel and input are multiplied and accumulated (dot product) to generate a single output value. This step is repeated until the kernel is convolved with the entire input (Figure 4.3 (c)). Similarly, for 1D convolution, the 1D kernel is sliding from left to right and the dot product is computed for the overlapped region. Since kernel rows and input rows are aligned after tiling, 1D convolutions essentially perform the same computation as 2D convolutions and generate the same results (first two rows of Figure 4.3 (d)). The outputs are valid 2D convolution results as long as the tiled kernel is fully inside the tiled input rows. However, when continuously sliding the 1D kernel as shown in the last row of Figure 4.3 (d), the dot product results are invalid, since filter row 3 (g, h, i) is not convolving with the correct input row (row 5 of original input cannot be tiled). For the example in Figure 4.3, a 20-element output is generated, but only the middle 10 elements are valid convolution results (two valid output rows). For the cases where the entire 2D input cannot be fully tiled, the tiling will be repeated until all the valid output rows are generated. The general formula for the number of valid output rows N_{or} that can be generated through row tiling in one convolution operation is

$$N_{or} = \left\lfloor \frac{N_{conv}}{s_i} \right\rfloor - S_k + 1$$

, and the total number of 1D convolution required is $\left\lceil \frac{S_i}{N_{or}} \right\rceil$. Therefore the computation efficiency (measured by the percentage of valid outputs) is higher when N_{conv} is large or S_i is small.

Edge effect: 2D convolution in 'same' mode pads input edges with zero. The output of the proposed row tiling algorithm will be different in the regions where a single kernel row overlaps with two input rows because row tiling does not pad inputs (Figure 4.3 (e)). The difference only happens at the edges of the original input rows and the impact is minimal, especially for small kernels. Zero-padding can be applied during tiling so that the proposed algorithm can generate identical results compared to 2D convolution. However, adding zero-padding will make the output size larger than the input, which leads to additional overheads caused by extracting the desired output. Since the impact of the edge effect is small (Section 4.3.4), zero-padding is not applied by default.

4.3.2 Partial row tiling

When $S_i <= N_{conv} < S_k \times S_i$, not enough input rows can be tiled to generate an entire row of 2D convolution output in one step. In this case, tiling can still be applied but multiple cycles are required to obtain the full results of one output row.

For example, when $N_{WA} = 2 \times S_A$, the computation of a single output row is split into two cycles and the results are accumulated after both cycles complete the execution. In cycle 1 the first two rows of the input and the kernel are tiled while in cycle 2 only the third row of input and kernel are processed (under-utilizing the convolution hardware). The number of cycles required to compute a full 2D convolution is $S_i \times \left\lceil \frac{S_k}{N_{ir}} \right\rceil$, where $N_{ir} = \left\lfloor \frac{N_{conv}}{S_i} \right\rfloor$ (the number of input rows can be tiled).

4.3.3 Row partitioning

When $N_{conv} < S_i$, a single row of input needs to be split into multiple partitions. The partitioning is similar to the case where $N_{conv} = S_i$ (dividing the 2D input into individual rows), except that each input row is further divided into partitions. The total number of cycles required to compute a full 2D output plane is $S_i \times S_k \times \left\lceil \frac{S_i}{N_{conv}} \right\rceil$. Row partitioning is typically only used for the first layer of CNNs with high-resolution inputs. In later layers, the size of inputs usually will be reduced through pooling.

4.3.4 Accuracy of row tiling/partitioning

We evaluate the accuracy of the proposed row tiling method with 1D convolution (theoretical accuracy of PhotoFourier) on three common CNNs using the ImageNet dataset, which are AlexNet [KSH12], VGG-16 [SZ14], and ResNet-18 [HZR16]. Since the proposed PhotoFourier accelerator implements 1D convolution, the row tiling/partitioning accuracy is the same as the theoretical accuracy of PhotoFourier.

Prior works on photonic accelerators that focus on system architectures either did not report any accuracy [SWK20, SKB21] (accelerate uncompressed neural networks) or just reported theoretical accuracy of their compression method [LLY19, ZLY20] (accelerate compressed neural networks). Therefore the theoretical accuracy is the only metric to compare the relative accuracy between different on-chip photonic accelerators, and we compare them whenever possible in this evaluation.

The evaluated accuracy results are shown in Table 4.1, original accuracy is the floating-point accuracy. We use the row tiling algorithm in this evaluation, but partial row tiling and row partitioning should achieve the same accuracy. In general, PhotoFourier with the proposed row tiling/partitioning method can achieve less than 1% drop in top-1 and top-5 accuracy for most cases and performs on par with or better than [LLY19] and [ZLY20]. The accuracy results for the row tiling method are inference only using weights trained with 2D

Table 4.1: Original accuracy of three CNNs and the accuracy drop of different neural network accelerators (%). T-1 and T-5 mean top-1 and top-5 accuracy. Ours stands for the proposed row tiling/partitioning method with 1D convolution. Accuracy drop is reported instead of raw accuracy because we have slightly different original accuracy than what is reported in [LLY19] and [ZLY20]. Top-1 accuracy is not reported in both prior works.

	Original		Ours		[LLY19]	[ZLY20]
	T-1	T-5	T-1	T-5	T-5	T-5
AlexNet	56.5	79.1	-0.7	-0.4	-0.8	N/A
VGG-16	73.4	91.5	-0.8	-0.4	N/A	N/A
ResNet-18	69.8	89.1	-1.3	-0.9	-0.6	-1.5

convolutions, and the accuracy drop could be eliminated with retraining.

4.4 PhotoFourier Compute Unit

We name the building block of the proposed PhotoFourier accelerator PhotoFourier Compute Unit (PFCU). Each PFCU is essentially an optimized version of the JTC system shown in Section 4.2.2.

4.4.1 Pipelining the PFCU

The baseline JTC system requires photodetectors and MRRs in the middle of the system to implement the square function in the Fourier domain, hence the system can be split into two identical parts each with a set of MRRs, Fourier lens, and photodetectors. The reaction time of photodetectors is usually the bottleneck and prevents the system from operating at higher frequencies. Figure 4.4 depicts the pipelined version of the JTC system. The pipelining is implemented by adding a sample and hold unit at the Fourier plane to buffer the output of the photodetectors. This two-stage pipelined PFCU, processing two convolutions at the same time, can double the throughput with a negligible increase in energy per convolution.

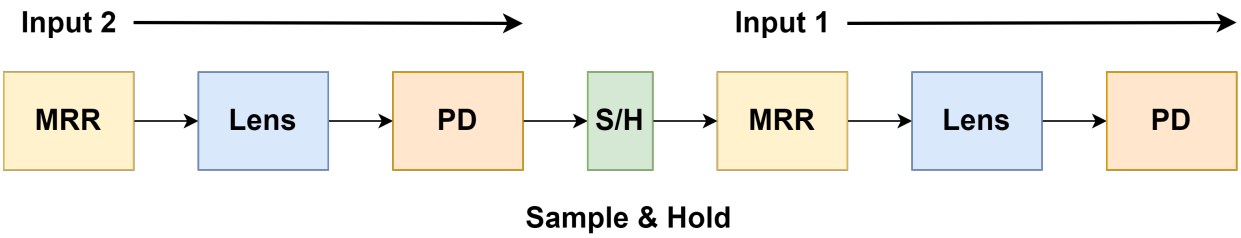


Figure 4.4: Visualization of pipelined PFCU.

4.4.2 Optimizing PFCU for small filters

JTC is originally designed to compute the convolution between two input signals of the same size. Therefore, the number of input waveguides is the same as the number of filter waveguides in the baseline JTC system However, for CNNs, the filter size is typically much smaller than the size of input activations. Each filter waveguide requires a DAC and an MRR to generate the corresponding weight value, and these devices are redundant if the waveguide is inactive, which means the waveguide never needs to generate non-zero values. To improve the area and power efficiency of the JTC system, DACs that correspond to inactive weight waveguides should be removed. Since the locations of active waveguides depend on the input activation size according to the proposed row tiling method, MRRs should not be removed so that every filter waveguide can be active if necessary. MRRs require far less area and power compared to DACs, and can be power gated to save power when inactive. In modern CNNs, the filter size is rarely larger than 5×5, therefore most of the waveguides are inactive. PFCU keeps 25 active waveguides with corresponding DACs for backward compatibility considerations. For the rare cases where the filter size is larger than 5×5, the inputs and filters can be partitioned to fit onto PFCUs (discussed in Section 4.3.2). The inactive waveguides act as zero-padding, they are still fabricated on the JTC, but they do not receive any inputs and consume zero energy.

4.5 Architecture Design

We will introduce the high-level architecture and configuration of PhotoFourier first. The optimizations and reasons behind the choice of design parameters will be covered later in this section (Section 4.5.2 to Section 4.5.6).

4.5.1 Overall system architecture

We architect two versions of PhotoFourier, PhotoFourier-CG (current generation) and PhotoFourier-NG (next generation). PhotoFourier-CG, as its name suggests, uses conservative estimations on area, power, and integration technology. Figure 4.5 shows the high-level architecture of PhotoFourier-CG. We architect PhotoFourier-CG as a two-chiplet system, with one CMOS chiplet and one photonic integrated circuits (PIC) chiplet. The PIC contains 8 PFCUs, each with 256 input waveguides, and is clocked at 10 GHz. PhotoFourier-CG by default operates at 8-bit precision. PhotoFourier-CG uses input broadcasting and OS dataflow, and implements 16-channel temporal accumulation to reduce the ADC and CMOS (except for the input generation circuit) frequency to 625 MHz. The input and weight DACs still operate at 10 GHz while SRAM operates at 625 MHz. Data buffers are used to communicate between two clock domains during input/weight generation. PhotoFourier also contains 8 CMOS tiles that are designed to handle the input and output of PFCUs, as shown in Figure 4.5 (a). The CMOS tile contains two sub-circuits, one is for input generation and one is for output processing. The input generation circuit has two clock domains, the slower clock is for weight memory access while the faster clock is used to control the DACs. The output processing circuit is used to read and accumulate photodetector outputs as well as to apply scaling/normalization and activation functions. Each CMOS tile has a 512 KB local weight SRAM while the entire PhotoFourier shares a 4 MB global activation SRAM. The activation memory size is set to be large enough to store the activations of common CNNs [SZ14, KSH12, HZR16] locally with ping-pong buffering (2 $\times$ maximum activation size),

such that the number of DRAM access is minimized and activations storing and loading can happen at the same time. Similarly, the weight SRAM size is set to store the weights of an entire layer of common CNNs [SZ14, KSH12, HZR16]. We have taken pseudo-negative processing into account when determining the size of weight SRAM, which will double the storage requirement (see Section 4.6.1). PhotoFourier-CG has an activation tile that is similar to the input generation circuit in Figure 4.5 (a), but is connected to activation SRAM and generates input activations that are shared among all PFCUs.

We choose to not assume CMOS and photonics can be fabricated on the same chip monolithically with *advanced* technology nodes in PhotoFourier-CG, unlike some other works that are based on such assumption [SWK20, SKB21, ZLY20, LLY19]. The reason is the current state-of-art commercial available technology for monolithic CMOS and photonics integration can only fabricate 45nm CMOS [RMN20], which is several technology nodes behind the state-of-art 5nm technology [YLC19].

PFCU layout optimization Figure 4.5 (c) shows a simplified layout diagram of the PFCU. Compared to the baseline JTC system in Figure 4.1 (a), the system is flipped after the first set of photodetectors which is in the middle of the system and signals travel towards the CMOS chip in the second part of the system. This folded layout is adopted to place the weight MRRs and the final photodetectors on the same side of the PFCU and close to the CMOS chiplet, which can reduce the length of the analog signals to/from the CMOS chiplet. In the 2-chiplet based system, ADCs and DACs are placed on the CMOS chiplet, hence analog signals need to travel between the chiplets. The loss of analog link due to IR drop will be troublesome if the link length is too long, therefore final photodetectors cannot be placed on the other end of the PIC.

Another layout optimization is the MRRs and PDs are grouped into rows with the size of 32 and stacked vertically to reduce the PFCU width (if oriented as in Figure 4.5 (c)). Placing 512 MRRs and PDs in one row horizontally will lead to more than 10 mm width of a

single PFCU, which makes the multi-PFCU layout impossible. Even with this optimization, each PFCU still has about 2.32 mm width due to a large number of waveguides and the folded layout. This width makes the layout and fabrication of 16-PFCU challenging as all PFCUs need to be placed close to the CMOS chiplet. Therefore, PhotoFourier-CG only uses 8 PFCUs to make the PIC width reasonable.

PhotoFourier-NG We also architect an advanced version of PhotoFourier, PhotoFourier-NG, which assumes next-generation technologies that are not mature enough currently, but will be available in the near future. On the architecture level, there are two main differences compared to PhotoFourier-CG: (1) PhotoFourier-NG assumes non-linear materials are used to implement the square function of JTC passively instead of photodetectors and MRRs; (2) PhotoFourier-NG assumes monolithic integration of CMOS and photonics with advanced technology node, which eliminates all layout constraints discussed in the previous paragraph. In this case, PFCUs no longer need to have a folded layout and can be placed in just one dimension. Therefore, PhotoFourier-NG uses 16 PFCUs instead of 8 to improve power efficiency. Besides these two differences, all other design parameters are the same for PhotoFourier-CG and PhotoFourier-NG.

4.5.2 Bottleneck analysis of baseline system

To optimize the system for power efficiency, it is important to understand the power bottleneck of a baseline system. The baseline system is configured as having 1 PFCU, 256 input activation waveguides, and clocked at 10 GHz. We evaluate the system on VGG-16 [SZ14] and profile the power contribution of different components. Figure 4.6 shows the power profiling results. ADCs and DACs dominate the system power and contribute more than 80% of the total system power. Therefore the architecture and dataflow should be designed to minimize the number of O-E and E-O conversions.

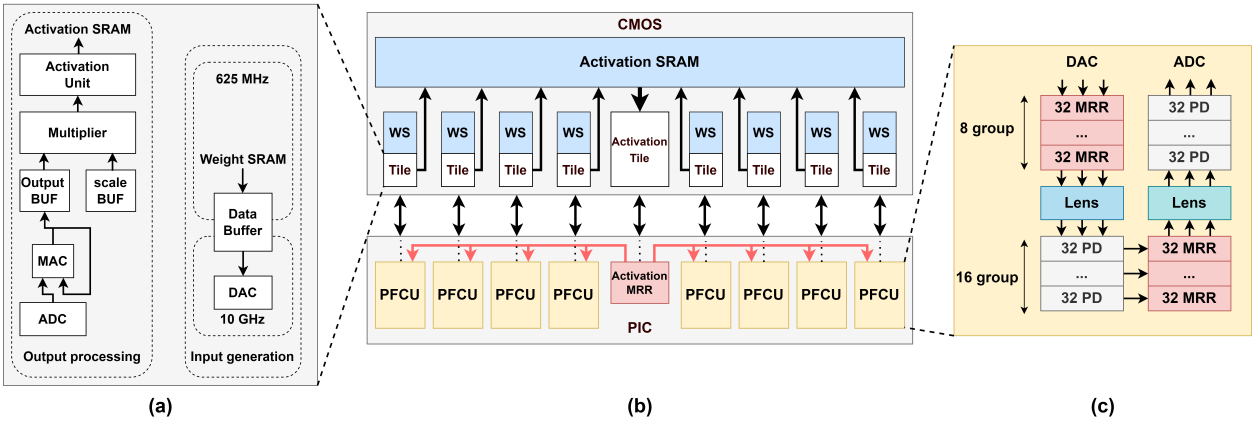


Figure 4.5: High-level architecture diagram of PhotoFourier-CG. WS stands for weight SRAM, BUF stands for buffer and PD stands for photodetector. (a): CMOS processing tile assigned to one PFCU. It contains two parts, one part for filter weight data generation and the other part for output processing. (b): PhotoFourier-CG architecture with 8 PFCUs, using 2.5D integration. (c): Simplified layout diagram of PFCU. MRRs and photodetectors are grouped to reduce the width of PFCU.

4.5.3 Temporal accumulation

From the results of Section 4.5.2, it's clear that O-E and E-O conversions are the bottlenecks of the baseline system. It is crucial to reduce the power consumption of DACs and ADCs to improve overall power efficiency. Thus, DACs and ADCs should be as low-frequency and low-precision as possible (lower frequency also makes the CMOS receiving circuit operate slower). However, lower precision usually leads to worse accuracy, especially for ADCs since they quantize partial sums which typically require higher precision than activations and weights.

To address these issues, we propose the temporal accumulation method, which is a key optimization of PhotoFourier. Temporal accumulation is a method to accumulate the convolution results temporally using photodetectors (before the O-E conversion), which can reduce the ADC and CMOS frequency and the output data bandwidth, as well as improve the accuracy for designs with 8-bit ADCs. Since the accumulation happens before the ADC readout which applies quantization, temporal accumulation can be considered as full-precision and

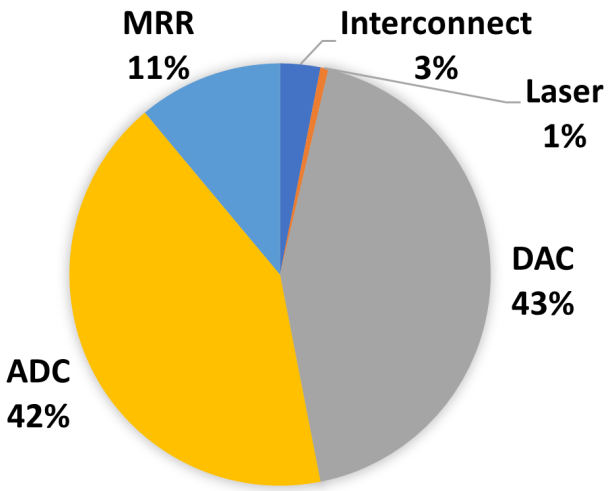


Figure 4.6: Power contribution of different components of a 1-PFCU baseline system.

can improve the accuracy. In other words, temporal accumulation allows 8-bit ADCs to be used without significant accuracy drop, which is otherwise impossible for certain cases. The accumulation happens at the photodetector and can be achieved through capacitors which accumulate the charges to be read at a later time.

In a typical CNN, the convolution results of different input channels need to be accumulated to compute the final output activation, therefore the dataflow needs to be organized in a way that the innermost loop is the input channel such that the output of the channels can be accumulated by the photodetector. There are both accuracy and performance considerations when choosing the number of channels that are accumulated by the photodetector (temporal accumulation depth). The accuracy results suggest that the temporal accumulation depth of 16 achieves the best accuracy and can restore the accuracy drop due to ADC quantization. Further increasing the temporal accumulation depth will not improve the accuracy, but will make photodetectors larger and slower (and harder to design). Also, for small CNNs, the number of input channels can be quite small and can result in under-utilization if the temporal accumulation depth is too large. Therefore, we set the temporal accumulation depth to 16 in PhotoFourier, which significantly improves overall accuracy, while still being flexible

and implementable. This leads to a 16× reduction in ADC frequency, CMOS frequency on the receiving end, and output bandwidth. Consequently, the power of ADC can be massively reduced while the CMOS circuit can operate below 1 GHz (except for the input/weight generation circuit). Temporal accumulation addresses two issues faced by photonic neural network accelerators with minimal hardware overhead, hence is the de-facto design choice of PhotoFourier and is prioritized in our dataflow and parallelization scheme analysis.

4.5.3.1 Temporal accumulation accuracy

To demonstrate temporal accumulation can improve accuracy, we generate the accuracy results of ResNet-s (a pruned version of ResNet-18 used in [BRT21]) on CIFAR-10 with different temporal accumulation depths. ResNet-s is selected since it is a compressed network and is more sensitive to quantization. The network is trained using the pseudo-negative approach described in Section 4.6.1. The model simulates the impact of photodetection, which includes applying square function to partial sums and adding sensing noise. The signal-to-noise (SNR) ratio is obtained by computing the average signal power at the photodetectors and compare to the noise power due to dark current. The results in Figure 4.7 suggest that temporal accumulation can significantly improve the accuracy for designs with 8-bit ADCs. The reason is that 8-bit precision is not enough for partial sums even though it is typically enough for inputs and weights. Since each ADC quantization incurs a large quantization error, having a greater temporal accumulation depth results in fewer partial sum quantization operations (temporal accumulation is full precision), and leads to smaller overall quantization error and better accuracy. The accuracy starts to converge towards the original accuracy when depth is 8 and the best accuracy is achieved when depth is 16, with less than 2% accuracy drop.

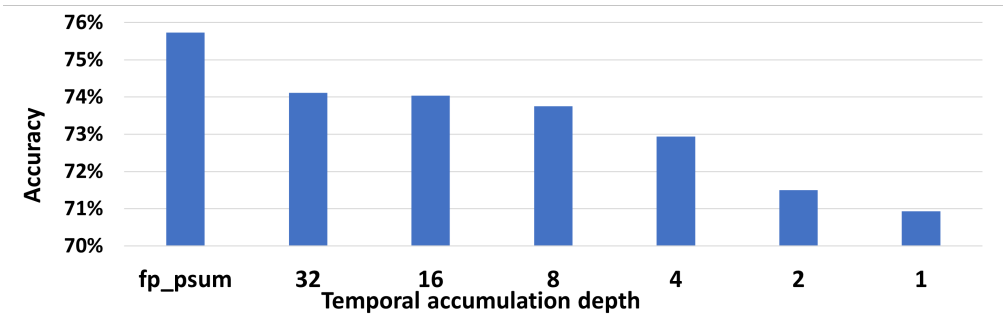


Figure 4.7: Accuracy of ResNet-s versus different temporal accumulation depth. fp_psum is the accuracy without ADC quantization.

4.5.4 Choice of parallelization scheme

We architect PhotoFourier as a system that consists of multiple PFCUs, hence the suitable parallelization scheme needs to be determined for optimal power efficiency. Given a set of available PFCUs, there are three parallelization schemes that can be considered, namely input broadcasting, weight broadcasting, and channel parallelization. Input broadcasting broadcasts the input activations to all PFCUs, and each PFCU computes a unique filter. In this scheme, the DACs and MRRs used to generate input activations can be shared among all PFCUs. Weight broadcasting broadcasts a single filter to all PFCUs, and each PFCU processes a unique convolution window or a full input activation (requires batched processing). Similarly, the DACs and MRRs required to generate filter weights can be shared. In channel parallelization, each PFCU processes one input channel, and the convolution outputs of all PFCUs are accumulated with a single set of photodetectors and ADCs. Channel parallelization scheme shares ADCs among PFCUs rather than DACs and MRRs. The three parallelization schemes can also be mix-and-matched for optimal results.

In our parallelization scheme analysis, we exclude the weight broadcasting scheme and only consider input broadcasting and channel parallelization. There are two reasons for this choice: (1) The number of DACs required to generate filter weights is significantly less than input activations, so the benefit of weight broadcasting is less than input broadcasting; (2) For many situations an entire input activation can be loaded on a single PFCU, thus

multi-batch processing is required for weight broadcasting, which is not always possible for inference tasks. Output stationary dataflow is used in this analysis, which is required for temporal accumulation.

Table 4.2: Table of notations used in the analysis and their meaning.

Notation	Definition
N_i	# input waveguides of each PFCU
N_w	# active weight waveguides of each PFCU
N_{PFCU}	# PFCUs available
IB	# PFCUs that inputs are broadcasted to
CP	# PFCUs that share ADCs
N_{TA}	# channels accumulated at photodetector
P_{DAC}	Power of DAC
P_{ADC}	Power of ADC

The optimal parallelization scheme can be formulated as a minimization problem of minimizing the sum of ADC and DAC power since they dominate the power consumption. Table 5.3 summarizes the notations used in the analysis. Given a fixed N_{PFCU}, the design parameter we want to find a solution is IB, and CP can be computed by $CP = \frac{N_{PFCU}}{IB}$. Assuming the power of ADC scales linearly with frequency, the sum of ADC and DAC power can be computed as:

$$P_{total} = P_{ADC} \times \frac{IB \times N_i}{N_{TA}} + P_{DAC} \times (CP \times N_i + N_{PFCU} \times N_w)$$

Since the power of ADC and DAC with the same frequency are similar, they can be removed from the minimization formulation. After some simplification, the minimization problem can be formulated as:

$$\textbf{Mimimize } \frac{IB}{N_{TA}} + CP$$

$$\textbf{Subject to: } IB \times CP = N_{PFCU}$$

This minimization problem can be solved by rewriting CP to $\frac{N_{PFCU}}{IB}$. The exact solution depends on hyperparameters N_{TA} and N_{PFCU}. By setting $N_{TA} = 16$, we can sweep IB to

find solutions for different N_{PFCU}.

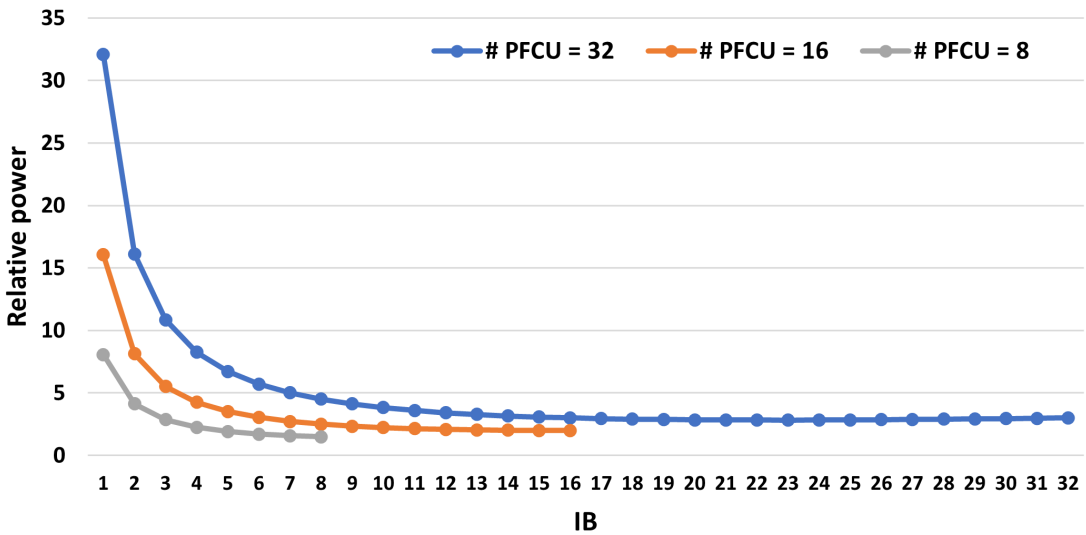


Figure 4.8: Value of $\frac{IB}{N_{TA}} + CP$ with different IB, for three different number of PFCUs.

Figure 4.8 plots the value of $\frac{IB}{N_{TA}} + CP$ for all IB values. For the cases where N_{PFCU} is 8 or 16, the system power minimizes when $IB = N_{PFCU}$. When $N_{PFCU} = 32$, the system power is the same when $IB = 16$ and $IB = 32$, and the minimum system power is achieved when $IB = 23$. However, the solution is not valid as the valid solutions for IB can only be $1, 2, 4, 8, 16, 32$ (due to the constraints of IB and CP). Therefore, both 16 and 32 are optimal solutions of IB when $N_{PFCU} = 32$. The result suggests that when N_{TA} is set to 16, and the number of PFCUs is less than or equal to 32, input broadcasting is always the best parallelization scheme.

4.5.5 Number of waveguides and PFCUs

The number of input waveguides per PFCU and the total number of PFCUs are two design parameters that need to be determined, and they are discussed together because of the trade-off between them. Given a fixed area budget, the more waveguides per PFCU, the fewer PFCUs can be placed. Assuming input broadcasting is used, increasing the number

of PFCUs improves the power efficiency by sharing the input activation with more filters, while increasing the number of waveguides per PFCU can also improve the power efficiency by effectively sharing a filter with more convolution windows (weight broadcasting within PFCU). However, PFCU can be under-utilized if the number of input waveguides is too large. Under-utilization typically happens when the system is executing later layers of a CNN, where the input activation size can be small (caused by pooling). For example, assuming the number of input waveguides is 512, then the PFCU will not be fully utilized if the input size is less than 23×23, which is common in the later layers of CNNs (e.g., ResNet-34 has 18 convolution layers with input size $\leq 14 \times 14$).

The optimal choice of the number of waveguides and the number of PFCUs is determined through benchmarks on real CNNs. We set the area budget of PhotoFourier-CG to 100 mm^2, which is around the upper limit of chip area due to the layout constraint discussed in Section 4.5.1. For consistency, we use the same area budget for PhotoFouirer-NG. We select 5 values for the number of PFCU and compute the maximum number of input waveguides per PFCU under the area budget for both PhotoFouirer versions. We then evaluate the selected configurations on 5 different CNNs, which are AlexNet [KSH12], VGG-16 [SZ14], ResNet-18 [HZR16], ResNet-32, and ResNet-50, and compute the geometric mean of the normalized FPS/W on these 5 CNNs. The results are listed in Table 4.3. PhotoFourier-CG achieves the best average FPS/W with 8 PFCUs and 270 waveguides per PFCU, while PhotoFourier-NG achieves the best average FPS/W with 16 PFCUs and 267 waveguides per PFCU. Therefore, for the given area budget, we architect PhotoFourier-CG to have 8 PFCUs with 256 input waveguides per PFCU and PhotoFouirer-NG to have 16 PFCUs with 256 input waveguides per PFCU.

Table 4.3: Maximum number of input waveguides per PFCU and the geometric mean of normalized FPS/W on 5 CNNs for PhotoFourier-CG and PhotoFourier-NG with different number of PFCUs, given a 100 mm^2 area budget.

	PhotoFourier-CG		PhotoFourier-NG	
# PFCU	# waveguides	avg. FPS/W	# waveguides	avg. FPS/W
4	412	0.70	576	0.55
8	270	**0.97**	395	0.75
16	172	0.89	267	**0.97**
32	105	0.72	177	0.82
64	61	0.74	114	0.81

4.5.6 Dataflow and reuse

4.5.6.1 Reuse analysis

Output stationery (OS) dataflow is used in PhotoFourier to implement temporal accumulation. In OS dataflow, every cycle processes a new channel of input activations and filters, so that the convolution results (partial sums) can be accumulated locally at the photodetectors. OS dataflow minimizes output data bandwidth at the cost of higher input/weight bandwidth. If input broadcasting is used without any data reuse scheme (e.g., OS dataflow), then the output bandwidth will be 8-16× higher than the input bandwidth, which may prevent the architecture to scale efficiently. OS dataflow addresses the imbalance between input and output bandwidth. By accumulating 16 channels at the photodetector, output bandwidth reduces by 16×. Although the input broadcasting + OS dataflow scheme does not lead to a direct reduction in weight bandwidth, the weight bandwidth requirement is much smaller than the input and output bandwidth for a single PFCU. There are two factors that contribute to the small weight bandwidth. (1): filters are much smaller than activations in general. (2): the local weight reuse over different convolution windows within each PFCU. This reuse is inherently implemented by JTC, which computes an entire convolution in one cycle so the weights are effectively shared among the inputs loaded onto the JTC. The weight reuse can be seen as weight broadcasting within the PFCU. When executing a

3×3 convolution layer, the total weight bandwidth can be 1.78× lower than the input and output bandwidth. Overall, PhotoFourier utilizes data reuse (sharing) on all three dimensions, which are input, weight, and partial sum. Input reuse is achieved by broadcasting the inputs to all PFCUs, weight reuse is achieved inherently by the JTC, and partial sum reuse is achieved by temporal accumulation at photodetectors.

4.5.6.2 Execution sequence

During the execution, PhotoFourier processes 8/16 filters in parallel and computes the convolution of one input channel tile per cycle. After the completion of one cycle, PhotoFourier processes the next input channel, until all input channels are processed. Since many convolution layers have more than 16 input channels, two-level accumulation is implemented. Input channels are divided into groups of 16 and partial sums are accumulated by the photodetector within each group, while CMOS accumulators are used to accumulate partial sums between groups. Once all input channels are processed, the accumulated results are sent to activation units to generate the output activation, which will then be stored in the activation memory. This process will be repeated until all the filters and input tiles are processed.

Figure 4.9 visualizes the activation memory mapping and the execution process of one input tile (tile 2). 16 channels of the input tile are loaded from the activation memory to the data buffer in parallel. This is possible since they are contiguous in memory. The data buffer connects the slower clock domain to the faster clock domain. The data buffer loads the 16 channels in one cycle of the slow clock domain and the input generation circuit operates 16X faster to generate one channel of the input tile in every cycle of the fast clock domain, which is then broadcasted to all PFCUs. Each PFCU processes a unique filter so 16 filters are processed in parallel, which produces the partial sums of 16 output channels. After all channels are computed and accumulated, the 16 output channels are stored in the activation memory.

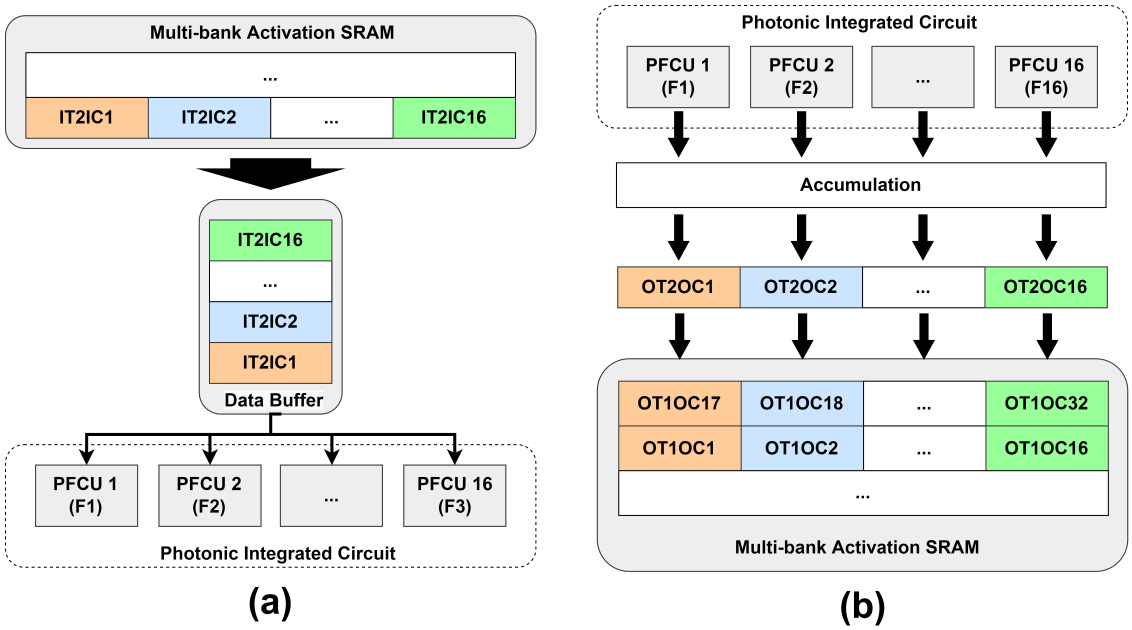


Figure 4.9: Visualization of the activation memory mapping and the execution process of input tile 2. IT: input tile, OT: output tile, IC: input channel, OC: output channel, F: filter. (a): Accessing input activation from memory. (b): Storing output activation to memory.

4.6 Evaluation

4.6.1 System Setup

We build a custom Python-based simulator to simulate the latency, power, area, and efficiency of PhotoFourier on actual CNN inferences. For PhotoFourier-CG, we use Cadence Genus with a commercial 14nm library to simulate the delay, power, and area of the CMOS circuit of PhotoFourier shown in Figure 4.5 (a). We use a commercial 14nm memory compiler to obtain the area, leakage power, and access energy of activation and weight SRAMs. For PhotoFourier-NG, we scale the technology node to 7nm, using the scaling equations proposed in [SB17] (for CMOS circuit, based on our 14nm results). We use PCACTI [SWL14] to model the 7nm FinFET SRAM. The simulation results of the CMOS circuit are embedded into the python simulator and combined with the simulation results of photonics to obtain the performance of the entire accelerator. Table 4.4 lists the power of different components that we

used in our simulator and the high-level design parameters, for both PhotoFourier-CG and PhotoFourier-NG. Since there are no references of ADC and DAC with the exact bitwidth, frequency, and technology node, we find 8-bit ADC and DAC with similar technology nodes (14nm and 16nm), but with higher frequency than required. We then linearly scale down the power of the cited ADC/DAC according to the frequency ratio to obtain the value we used in PhotoFourier-CG. We scale the power of ADC by 5.81× in PhotoFourier-NG, which is obtained using the Walden ADC Figure-of-merit (FOM) formula [Wal99] with the envelope line (best achievable FOM of published ADCs for different frequencies) constructed in [Mur]. We obtain the best FOM for 625 MHZ to estimate the optimal 8-bit ADC power from the FOM equation. We scale the DAC with the same number since DAC and ADC have similar scaling properties (SAR-ADCs are based on DACs). Table 4.5 lists the dimension of the optical components we used to compute the PFCU area, and we keep them the same for both PhotoFourier-CG and PhotoFourier-NG. The laser power is set to maintain larger than 20 dB SNR at photodetectors in most cases, estimated using the dark current of photodetectors and the system loss of PhotoFouirer. The simulator implements the proposed row tiling/partitioning algorithm when simulating CNN inferences, and uses a batch size of 1. As PhotoFourier is designed as a convolution accelerator, only convolution layers are accelerated and benchmarked. This will not affect the overall speedup on common CNNs[SZ14, HZR16] since more than 99% of total MAC operations are from convolution layers. We use PyTorch with custom convolution functions to generate all the accuracy results used in this paper.

Dealing with negative weights: PhotoFourier uses the pseudo-negative method [CSD18] to deal with negative weights, which can be troublesome for photonic accelerators to process. The pseudo-negative method breaks every filter into a pair of positive-value filters using the formula $x = p - n$, where x is the original weight and p, n are two positive-value filters. Pairs of filters are processed as normal in PFCUs and they are subtracted digitally in the CMOS circuit. The method makes photonic accelerators able to process negative weights but at the cost of 2× computation.

Table 4.4: Power of different components and high-level design parameters used for PhotoFourier-CG and PhotoFourier-NG

	PhotoFourier-CG	PhotoFourier-NG
Component power		
MRR	3.1mW[MLW17]	0.42mW[SLM21]
Laser	0.5mW per waveguide	0.5mW per waveguide
ADC @ 625 MHz	0.93mW[LHC22]	0.16mW
DAC @ 10 GHz	35.71mW [CMA20]	6.15mW
High-level design parameters		
# PFCUs	8	16
# input waveguides	256	256
# chiplets	2	1
technology node	14nm	7nm

Table 4.5: Dimensions of the photonic components used in area estimation

Component	Dimension
MRR[AIM]	15 um × 17 um
Optical splitter[ZYL13]	1.2 um × 2.2 um
Photodetector[AIM]	16 um × 120 um
Waveguide pitch[ZLZ19]	1.3 um
Laser [DJB13]	400 um × 300 um
On-chip lens	2 mm × 1 mm

4.6.2 Effect of optimizations

We first demonstrate the effect of proposed optimizations in terms of the geometric mean of FPS/W on the same five CNNs used in Table 4.3, and the results are shown in Figure 4.10. The baseline system is a single-PFCU system with 256 input channels, and we stick with the power number of PhotoFourier-CG in this evaluation to exclude the effect of technology scaling. We order the optimizations from PFCU-level optimization to architectural-level optimization. Small filter optimizations reduce the number of weight DACs per PFCU, PFCU parallelization shares 256 input DACs with 8 PFCUs, temporal accumulation reduces ADC frequency by 16×, and non-linear material (used in NG version only) removes the MRRs

used to compute the square function. The proposed optimizations significantly improve the power efficiency, and can be 15× better than the baseline system.

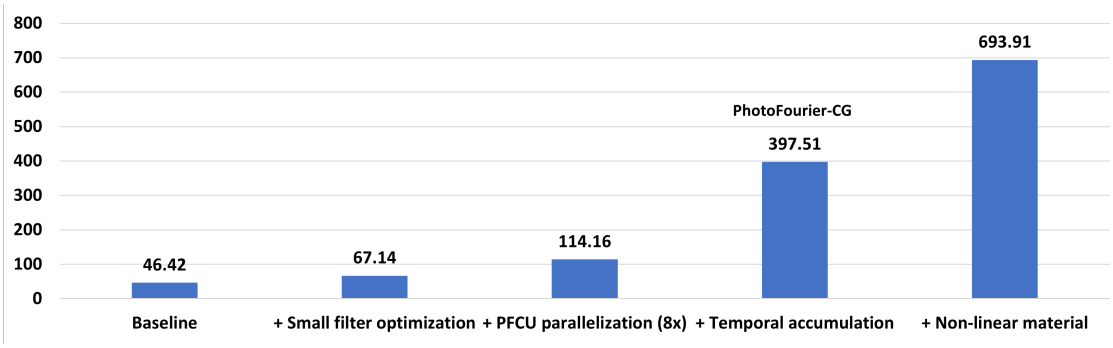


Figure 4.10: Geometric mean of FPS/W for PhotoFourier with different optimizations. Starting from the baseline, each column adds one optimization to the system, and includes all previous optimizations (columns on the left).

4.6.3 Area

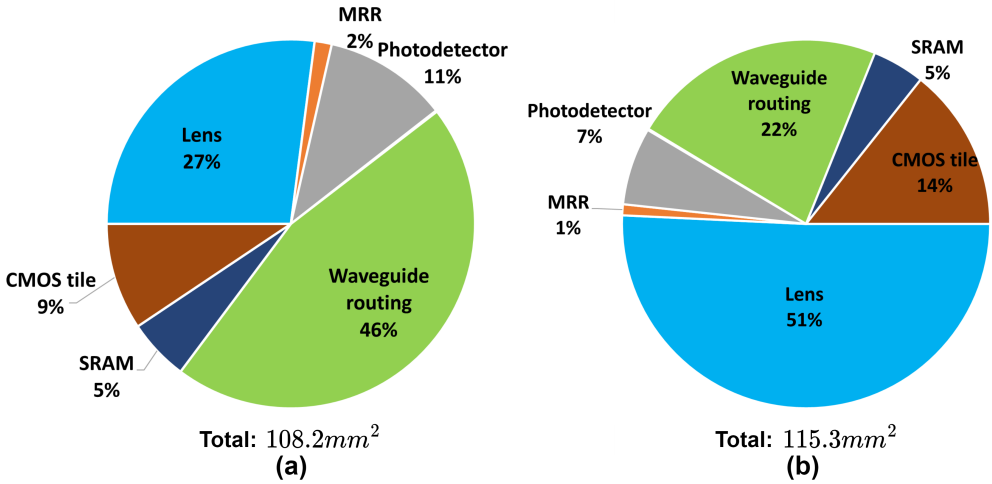


Figure 4.11: Area breakdown of PhotoFourier. Waveguide routing includes waveguides area and redundant area due to layout constraints. (a): CG version. PIC chiplet: 92.2mm^2, SRAM: 5.85mm^2, CMOS tile: 10.15mm^2. (b): NG version. PFCU: 93.5mm^2, SRAM: 5.3mm^2, CMOS tile: 16.5mm^2.

Figure 4.11 shows the total area and area breakdown for two PhotoFourier versions, with photonic components dominating the area for both versions. While having 2× PFCUs,

PhotoFourier-NG has roughly the same area as PhotoFourier-CG. The area reduction of individual PFCUs is the result of using non-linear materials to replace MRRs and photodetectors that implement the non-linear function. For PhotoFourier-CG, waveguide routing (including redundant space) uses nearly half of the chip area. Such low area utilization is caused by the layout constraint discussed in Section 4.5.1, which makes the layout less compact. In PhotoFourier-NG, the monolithic integration of CMOS and photonics, together with non-linear materials, greatly relaxes the layout constraint and makes the layout more compact. Since photodetector and MRR consume a very small portion of the total area in both versions, shrinking their sizes can barely improve area efficiency. Instead, more compact on-chip lenses should be studied to further reduce the footprint.

4.6.4 Power

We benchmark the two versions of PhotoFourier on the same 5 CNNs used in Section 4.5.5 to evaluate their performance. The average power of PhotoFourier CG and NG over the five networks are 26.0 W and 8.42 W respectively. Figure 4.12 shows the power breakdown of PhotoFourier-CG and PhotoFourier-NG. The power distribution of PhotoFourier-CG is somewhat evenly spread across MRR, DAC, and other components. The DAC and ADC no longer dominate the power consumption when compared to a baseline JTC system (Figure 4.6), as temporal accumulation and input broadcasting greatly reduce the power of ADCs and DACs. Temporal accumulation can reduce ADC power by more than 30× compared to 10 GHz ADCs [HC19], which makes ADC power significantly less than DAC power. For PhotoFourier-NG, the SRAM access power replaces MRR/DAC to become the largest contributor to the total system power. There are two reasons, one is the power of MRRs, DACs, and ADCs is further reduced in the NG version, and the reduction is larger than SRAM power reduction due to technology node scaling. Another reason is the SRAM access energy for PhotoFourier is on the higher end, as wide memory buses are required to keep up the bandwidth requirement of the 10 GHz photonic circuits, which increases the access

energy.

We make the following observations: Since data movement (memory accessing + interconnect) dominates the power of PhotoFourier-NG, the priority of further improving power efficiency with next-generation technologies is no longer optimizing O/E conversions. Reducing the data movement cost should be the main focus, which we will discuss more in Section 4.8.

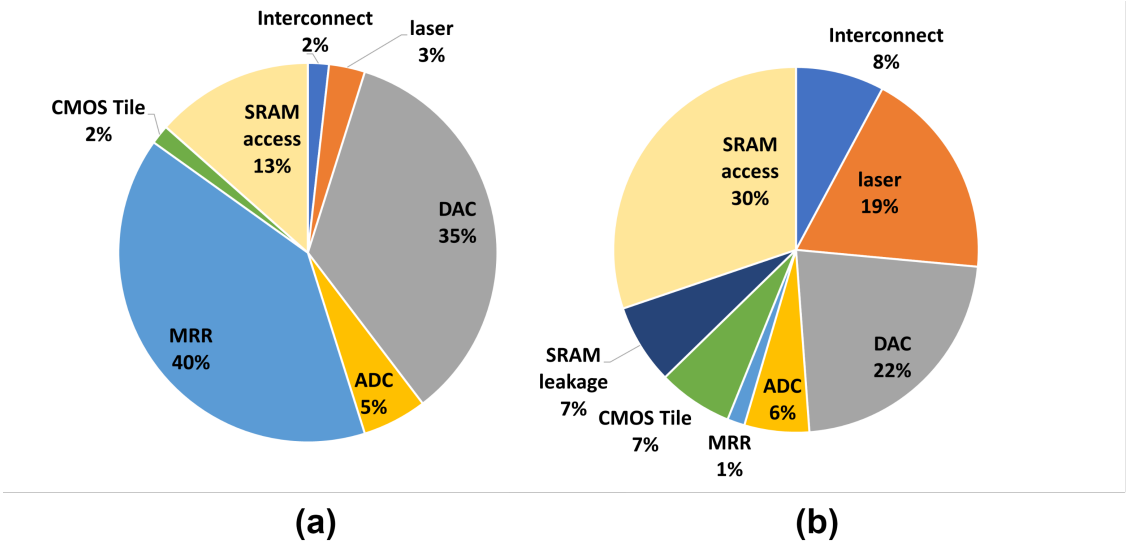


Figure 4.12: Power breakdown of the two PhotoFourier versions. (a): PhotoFourier-CG. (b): PhotoFourier-NG.

4.6.5 Comparison with prior works

We mainly compare the performance of PhotoFourier with the recently published Albireo accelerator which reports state-of-art power efficiency results for uncompressed CNNs. For reference purposes, we also compare with some other photonic neural network accelerators (DEAP-CNN [BMM19], Lightbulb [ZLY20], two versions of Holylight [LLY19]) and one digital accelerator (UNPU [LKK19]). Albireo is a CNN accelerator based on MZIs and MRRs. DEAP-CNN, Lightbulb, and Holylight are based on only microdisks/MRRs. Albireo and Holylight-m target 8-bit CNNs, DEAP-CNN targets 7-bit CNNs, while Holylight-a and

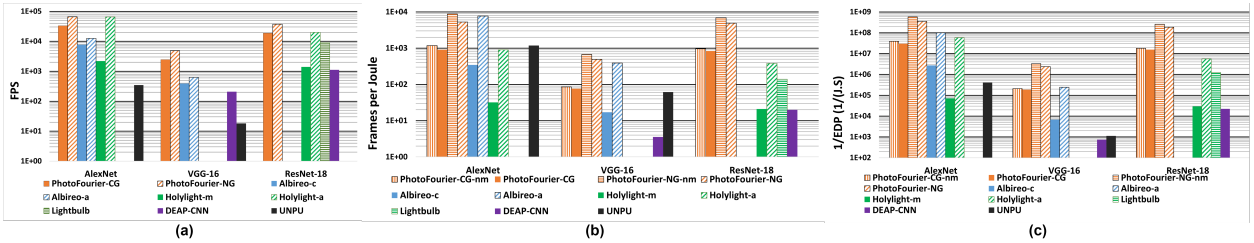


Figure 4.13: Inference performance of common CNNs on ImageNet dataset, compared to prior works. Missing bars indicate the data are not available from the original papers. (a) Inference throughput in terms of frames per second (FPS). (b): Inference efficiency in terms of frames per Joule (FPS/W). -nm means PhotoFourier versions without memory access power. To the best of our knowledge, Albireo does not report memory access power, so -nm versions are included for reference. (c): Energy-delay Product (EDP) in terms of $\frac{1}{EDP}$, inverse is used for better visualization, therefore larger is better.

Lightbulb target power-of-two quantized and binary CNNs respectively. Therefore, PhotoFourier is best compared to Albireo and Holylight-m since they can accelerate uncompressed 8-bit CNNs, which is not possible for Holylight-a and Lightbulb. We benchmark PhotoFourier on AlexNet, VGG-16, and ResNet-18 as each of the selected accelerators reports results on some of these networks. We compare PhotoFourier with both Albireo-c (conservative) and Albireo-a (aggressive), which is similar to PhotoFourier-CG and PhotoFourier-NG. Albireo has more aggressive assumptions for their advanced version since Albireo-a reduces the ADC and DAC power by 10× compared to Albireo-c, while we reduce them by 5.8× in the NG version based on FOM analysis. All results of other works are obtained directly from the original papers without modification except for DEAP-CNN (targets for small CNNs), for which we use a scaled version to generate results for our benchmarks. The results of Holylight and Lightbulb are estimated based on bar charts and are not precise. Albireo uses a 7nm library to model the CMOS components and UNPU uses the 65nm technology node.

Figure 4.13 (a) shows the throughput results in terms of frames per second (FPS). PhotoFourier-CG and PhotoFourier-NG have 5-10× higher throughput compared to Albireo-c and Albireo-a. Given that Albireo's chip area ($124.6mm^2$) is similar to PhotoFourier, PhotoFourier has 5-10× better area efficiency than Albireo. Holylight-a and Lightbulb

have higher throughput in general since they target quantized CNNs, but are still less than PhotoFourier-NG, except for AlexNet where PhotoFourier-NG is on par with Holylight-a. PhotoFourier is less efficient on AlexNet due to its first convolution layer with 11×11 filter and stride of 4. PhotoFourier is less efficient when processing strided convolutions as PhotoFourier handles them by computing with unit stride and then discarding unnecessary results, limited by the underlying JTC operation which only supports unit stride convolution.

Figure 4.13 (b) shows the FPS/W result, which measures the power efficiency. Compared to 8-bit accelerators and with *memory access power modeled*, PhotoFourier-CG achieves around 3-5× higher FPS/W than Albireo-c on benchmarked networks, and is 532× and 704× better than Holylight-m and DEAP-CNN respectively. Compared to Albireo-a, PhotoFourier-NG is slightly ahead for VGG-16, but is slightly behind for AlexNet because of the inefficiency of strided convolutions. PhotoFourier-NG achieves similar power efficiency compared to Albireo-a, despite modeling memory access power and using less aggressive ADC/DAC scaling. Even when compared to Holylight-a and Lightbulb which target heavily quantized CNNs, both PhotoFourier versions achieve better FPS/W, demonstrating superior power efficiency. While having low throughput, UNPU achieves decent power efficiency and is on par with PhotoFourier-CG (but behind PhotoFourier-NG). Figure 4.13 (c) visualizes the energy-delay product (EDP) in terms of $\frac{1}{EDP}$. Given PhotoFourier's high throughput and power efficiency, PhotoFourier-NG achieves the best EDP on all three networks. Even PhotoFourier-CG has better EDP than other accelerators in most cases, except for the less efficient AlexNet where PhotoFourier-CG falls behind Holylight-a, which targets heavily quantized networks.

We also compare PhotoFourier with another recent MRR-based photonic NN accelerator CrossLight [SMN21]. Since CrossLight does not evaluate using our selected networks, we evaluate PhotoFourier on their custom 4-layer CNN for the CIFAR-10 dataset. PhotoFourier-CG achieves more than 100× better energy per inference (4.76μJ vs 427μJ), despite having relatively low utilization on this network. Overall, PhotoFourier achieves state-of-art

throughput and efficiency results of photonic neural network accelerators. Compared to Albireo which also targets uncompressed CNNs, PhotoFourier-CG achieves up to 28× better EDP compared to Albireo-c and PhotoFouier-NG achieves up to 10× better EDP compared to Albireo-a. The better performance is contributed by the complexity reduction of Fourier optics, as well as the proposed optimizations. PhotoFourier requires fewer optical components to perform the same convolution operation which reduces the area and power of photonic components. Being compact means PhotoFourier can have more components than Albireo with a similar area budget, hence can benefit more from parallelism.

4.7 Related work

4F systems require spatial filters to be transformed into complex-valued Fourier filters before feeding into the system. This requires 4F systems to support complex multiplication, which is hard to implement as it requires both amplitude and phase modulation. Furthermore, 4F systems require filter sizes to match input activation sizes (for point-wise multiplication in the Fourier domain), thereby wasting substantial weight modulation bandwidth (conventional CNNs all have small filters). Unlike 4F systems, JTC treats filters the same way as inputs, where the Fourier lens computes the Fourier transform of filters. Thus JTC can use real-valued spatial filters with arbitrary size (with zero padding) and significantly improve overall efficiency compared to 4F systems. 4F CNN accelerators [CSD18, MHL20, GL22] are most closely related to PhotoFourier. However, they all target free-space 4F systems and face the common issues of 4F systems (discussed above), whereas PhotoFourier is faster, more flexible, and more efficient for accelerating conventional CNNs.

Most on-chip photonic neural network accelerators [SWK20, SKB21, BMM19, GZF20, BSS18, MAS18, LLY19, ZLY20, SHS17, SMN21] proposed so far are based on MZIs and MRRs. They are typically designed to accelerate dot products and some require IM2COL transformation to compute convolutions. They share similar architecture with other General

Matrix Multiplication (GEMM) based accelerators like systolic arrays or compute-in-memory arrays, but can operate faster and utilize wavelength-division multiplexing for extra parallelization. Unlike these approaches, PhotoFourier has a fundamentally different architecture by leveraging the complexity reduction of "free" Fourier transform of Fourier optics to deliver large throughput gains with fewer photonic components (which do not scale well with technology).

4.8 Discussion

4.8.1 PhotoForuier-base

The original PhotoFourier-CG uses many conservative estimations for component power in order to draw fairer comparisons with early works, while PhotoFourier-NG's estimation for ADC and DAC power is future-looking and lacks solid references. For the purpose of creating a solid (all the component power numbers have proper references) yet efficient baseline for future works to compare against, we introduce PhotoFourier-base, a slightly more realistic version of PhotoFourier-NG, which adopts the ADC and DAC used in PhotoFourier-CG, and keeps everything else the same.

Figure 4.14 and 4.15 visualize the power breakdown of the updated single JTC setup and PhotoFourier-base. The power breakdown results of PhotoFourier-base suggest that DACs and ADCs combined still dominate the overall system power after the optimization proposed in PhotoFourier and still have space for further improvement.

4.8.2 Laser power adjustment for linear photodetector response

Photodetectors, which are essentially made from photodiodes, do not always have a linear response to input light intensity. Only when the photodetector is operating in reverse-bias mode (linear region), the output is linearly related to the incident light intensity, which

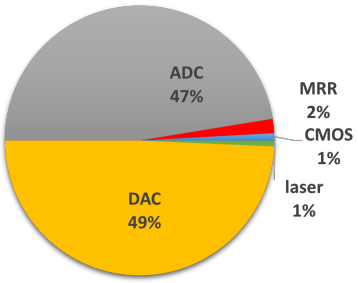


Figure 4.14: Power breakdown of a baseline JTC (single JTC) with same power numbers for DAC, ADC, and MRR as PhotoFourier-base.

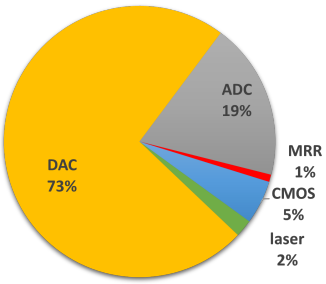


Figure 4.15: Power breakdown of PhotoFourier-base.

essentially achieves a square function to the input signal. When the light intensity is too low, the signal-to-noise ratio will drop significantly due to the dark current. Meanwhile, when the light intensity is too high, the photodetector can saturate, causing a near-clipping effect on the outputs. A default setup can make the photodetectors operate in their full range instead of the linear range, adding extra non-linearity to the system. This extra nonlinearity is not desired for classic JTC since photodetectors should implement precise square functions.

However, the laser power can be increased to force photodetectors' incident light intensity into the linear region to make the JTC output more precise, albeit saturation could occur if the dynamic range of photodetectors is not large enough. A rough analysis conducted in our ongoing work suggests that a 15% increase in laser power is sufficient to let the photodetectors operate in the linear region. The 15% increase in laser power will have an almost negligible impact on overall system efficiency since laser power only consumes a tiny fraction of total system power (2%) in PhotoFourier. Still, we updated the laser power in all results shown in this discussion section to reflect this increase in laser power.

It is also worth noting that extra nonlinearity, especially saturation, is not necessarily a bad thing for JTC. The extra nonlinearity essentially converts the system from classic JTC to nonlinear JTC, which can approximate convolution operations and can improve the overall output SNR. Our simulation results also suggest that saturation can indeed improve

the overall accuracy of our JTC prototype. This nonlinearity and its impact on accuracy is more thoroughly discussed in the final chapter.

4.8.3 Performance sensitivity analysis on effective JTC bitwidth

In PhotoFourier, the bitwidth of ADC and DAC are set to 8-bit, a commonly used bitwidth for fixed-point neural networks that achieve a good balance between accuracy and efficiency. However, due to all the noises and non-idealities of the JTC hardware, its effective number of bits (ENOB) might be less than 8 bits. In such cases, ADCs and DACs with lower bitwidths should be used instead of 8-bit ones, and this can lead to a significant reduction in ADC and DAC power since their power scales roughly quadratically with bitwidth. Given that ADCs and DACs dominate the system power for 8-bit case and hence become the main optimization target, does the argument still hold for ADCs and DACs with lower bitwidths?

This section aims to answer this question by modeling the overall performance and power breakdown for different ENOB for the system. Since the exact achievable ENOB depends on the exact hardware and cannot be pre-assumed, 2 common bitwidths are studied and evaluated in this section, which are 6 bits and 4 bits.

ADC and DAC scaling For these two cases, the ADC and DAC bitwidth will be changed to 6 and 4 bit respectively, hence their power numbers used for performance modeling should also be updated. Given it's hard to find citable sources for ADCs and DACs with exact bitwidth, frequency, and technology node requirements, we adopt a conservative exponential scaling for their power when reducing ADC and DAC bitwidths. More specifically, 6-bit ADCs and DACs have 4× less power than the 8-bit ones used in the original PhotoFourier, while 4-bit ADCs and DACs have 16× less power than the 8-bit ones.

Power breakdown results Figure 4.16, 4.17, 4.18, 4.19 shows the power breakdown of the baseline single JTC setup and PhotoFourier-base for 6-bit and 4-bit ADCs and DACs.

Figure 4.16 suggests that for 6-bit system precision, ADCs and DACs remain to dominate the system power consumption for the baseline JTC so that the optimizations introduced in PhotoFourier are still valid. For 4-bit case, the overall power contribution of ADCs and DACs reduces noticeably while the power of other components starts to matter more. Still, for the baseline JTC setup, ADC and DAC remain to be the largest power contributors and hence should still be the primary optimization target. For the 4-bit PhotoFourier-base, after the proposed optimization in this chapter, the DACs are no longer the largest power contributor, unlike The 8-bit and 6-bit versions. CMOS power starts to dominate and hence CMOS circuit and architecture optimization should be considered when further optimizing the system.

A conclusion can be drawn from this sensitivity analysis is that for a baseline JTC system, the conversion overhead (ADCs and DACs) should be the top optimization priority for system bitwidth of 4-bit and higher. For bitwidth below that, the top priority should be try to improve the ENOB of the optical system since a system with less than 4-bit precision might not be sufficient enough for neural network inference tasks. However, if in certain cases such low ENOB is acceptable, then a more through analysis should be performed to determine the priority when optimizing the system performance from architecture level.

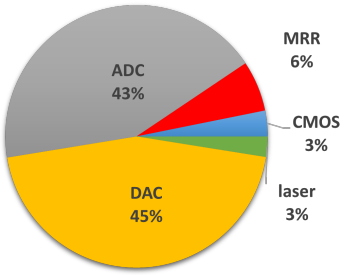


Figure 4.16: Power breakdown of a 6-bit baseline single JTC setup.

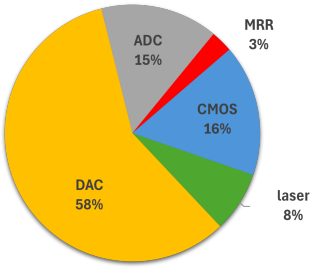


Figure 4.17: Power breakdown of 6-bit PhotoFourier-base.

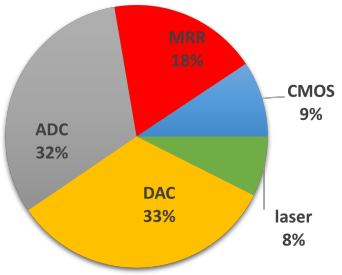


Figure 4.18: Power breakdown of a 4-bit baseline single JTC setup.

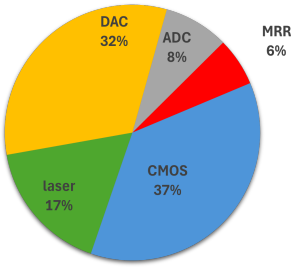


Figure 4.19: Power breakdown of 4-bit PhotoFourier-base.

4.9 Conclusion

In this paper, we present PhotoFourier, a JTC-based on-chip photonic neural network accelerator. We propose an algorithm to compute 2D convolutions with 1D convolutions that can be implemented using the 1D on-chip lenses. We also propose temporal accumulation to improve the accuracy and power efficiency of the system. Besides, we provide a detailed analysis of how to determine optimal design parameters for a JTC-based CNN accelerator including dataflow, parallelization scheme, and the number of waveguides and compute units. Compared to uncompressed photonic neural network accelerators, the EDP of PhotoFourier-CG is 28× better compared to Albireo-c, 532× better compared to Holylight-m and 704× better compared to DEAP-CNN. There are still many remaining challenges for on-chip photonic neural network accelerators, which include the relatively large size and limited flexibility of optical components, manufacturing variations of photonics, data movement cost, and neural network architectures and training methods for photonic execution. We plan to address these issues in our future work, along with a large-scale experimental demonstration of PhotoFourier.

CHAPTER 5

ReFOCUS: Reusing Light for Efficient Fourier Optics-Based Photonic Neural Network Accelerator

In recent years, there has been a significant focus on achieving low-latency and high-throughput convolutional neural network (CNN) inference. Integrated photonics offers the potential to substantially expedite neural networks due to its inherent low-latency properties. Recently, on-chip Fourier optics-based neural network accelerators have been demonstrated and achieved superior energy efficiency for CNN acceleration. By incorporating Fourier optics, computationally intensive convolution operations can be performed instantaneously through on-chip lenses at a significantly lower cost compared to other on-chip photonic neural network accelerators. This is thanks to the complexity reduction offered by the convolution theorem and the passive Fourier transforms computed by on-chip lenses. However, conversion overhead between optical and digital domains and memory access energy still hinder overall efficiency.

We introduce ReFOCUS, a Joint Transform Correlator (JTC) based on-chip neural network accelerator that efficiently reuses light through optical buffers. By incorporating optical delay lines, wavelength-division multiplexing, dataflow, and memory hierarchy optimization, ReFOCUS minimizes both conversion overhead and memory access energy. As a result, ReFOCUS achieves 2× throughput, 2.2× energy efficiency, and 1.36× area efficiency compared to state-of-the-art photonic neural network accelerators.

External collaborators:

Listed affiliations are at the time of collaboration:

- Dr. Hangbo Yang, UCLA
- Dr. Nicola Peserico, University of Florida
- Prof. Volker J. Sorger, University of Florida

5.1 Introduction

Convolutional neural networks (CNNs) [KSH12, HZR16, SZ14, RDG16, GDD13, TL19] have become indispensable in modern Artificial Intelligence (AI) applications, forming the basis of numerous computer vision tasks such as image classification, object detection, and autonomous driving. Although vision transformers [CRC20, TCD21, DBK20] are gaining popularity, CNNs still maintain an edge in terms of model compactness and the ability to achieve comparable accuracy to vision transformers with significantly fewer parameters [vis23]. Due to the complexity of convolution operations, executing them on general-purpose processors is not energy efficient. Therefore, researchers have focused on developing domain-specific accelerators employing parallel architectures for energy-efficient computation of neural networks [CES16, CDS14, SZW18, LKK19, PRM17]. However, the ever-increasing complexity of modern CNNs, the end of Dennard scaling, and the slowdown of Moore's law have imposed limitations on CMOS digital accelerators concerning energy consumption for data movement and computation. Silicon photonics emerging as a promising solution to this problem, which offers remarkable computational parallelism and efficiency.

Photonics components possess several unique advantages, including high frequency, relatively low power consumption, and no RC delay. These characteristics make photonics an unparalleled contender for low-latency and low-power computation. Generally, there are two

types of photonic neural network accelerators: free-space and on-chip versions. While free-space optical neural network accelerators [LRY18, MHL20, CSD18, CCS19, HLS, GL22] are often bulky and inflexible, on-chip photonics-based accelerators have gained significant interest due to their efficiency and flexibility. On-chip photonics can be further classified into two main categories. Most existing works compute dot products or vector-matrix multiplications using Mach-Zehnder Interferometers (MZI) and/or micro-ring resonators (MRR) [SWK20, SKB21, BMM19, LLY19, ZLY20, GZF20, SMN21, LLK22]. These MZI/MRR-based photonic neural network accelerators share similarities with compute-in-memory (CIM) analog accelerators but feature high clock frequencies (5-10 GHz) and the possibility of leveraging wavelength-division multiplexing (WDM) for extra parallelism. However, a major bottleneck of photonic and other analog neural network accelerators is the conversion cost between digital and analog domains, which can consume a significant amount of power. Unlike CIM accelerators which are typically designed to have tall columns to reduce the compute-to-conversion ratio, photonic neural network accelerators often have significantly smaller arrays because of relatively large photonic components and limitations of WDM. This results in lower compute-to-conversion ratios. Moreover , the conversion overhead between digital and optical domains often prevents photonic neural networks from delivering their theoretical advantage over CMOS electronics.

The second category focuses on computing the convolution directly. This can be achieved through a pair of Fourier lenses that compute the Fourier transform passively. Fourier optics-based designs capitalize on the convolution theorem, which states that convolution in the space domain is equivalent to point-wise multiplication in the Fourier domain. These systems, commonly referred to as 4F systems, utilize time-of-flight Fourier transform via Fourier lenses to reduce convolution complexity from $O(N^2)$ to $O(N)$. Compared to conventional MZI/MRR-based photonic neural network accelerators, 4F systems require significantly fewer optical components to perform the same amount of computations, thanks to the complexity reduction. This type of photonic neural network accelerator was typically built

as a free-space system, but recently, silicon photonics versions have been proposed, opening a new direction for designing efficient photonic neural network accelerators. [LYW23a] proposed a Joint Transform Correlator (JTC) based on-chip photonic neural network accelerator, which is a variant of the 4F system (still using Fourier optics), and achieved orders of magnitude better efficiency than previous state-of-the-art photonic neural network accelerators. JTC computes the auto-convolution of two input signals using a pair of Fourier lenses similar to 4F systems, but it uses spatial filters instead of complex-valued Fourier-domain filters. JTC addresses some limitations of conventional 4F systems, such as the support for complex filters and the large filter size (as Fourier-domain filters need to have the same size as inputs).

Although the JTC-based photonic neural network accelerator already demonstrates state-of-the-art efficiency, there is still substantial room for further optimizations. On one hand, the conversions between analog and digital domains still consume a large proportion of system power. On the other hand, as computation becomes even more efficient, memory access power becomes non-negligible. Both of these aspects could be optimized to further improve system efficiency.

In this work, we propose ReFOCUS, a JTC based on-chip photonic neural network accelerator that reuses light through optical buffers to minimize the conversion cost between optical and digital domains. With optical reuse and various optimizations, ReFOCUS is able to achieve significantly better energy efficiency compared to state-of-the-art photonic neural network accelerators. The main contributions can be summarized as follows:

- We propose optical reuse based on optical buffers constructed using optical delay lines, and incorporate corresponding dataflow and laser power optimization to significantly improve the power efficiency of the system.
- We adopt wavelength-division multiplexing (WDM) to improve the area efficiency by sharing on-chip lenses, which also reduces the area overhead of optical buffers.

- ReFOCUS can achieve 2× throughput, 2.2× energy efficiency and 1.36× area efficiency than previous state-of-the-art photonic neural network accelerator.

5.2 Background

5.2.1 Background of JTC

Over the past couple of decades, JTC has found applications in a variety of fields, such as image filtering [TS98, Jav90] and object tracking [TFG90, LA04]. Recently, JTC systems have been used for accelerating neural networks [GSY22, LYW23a, YLM22, PMY23]. Theoretical analysis and experimental demonstrations of low-latency convolution operations using JTC systems have been presented in [GSY22] and [YLM22, PMY23] respectively, while [LYW23a] proposed the architecture-level design and optimizations.

The math behind JTC operations has been adequately discussed and analyzed in previous literature, so we will not go into too much detail in this paper as the focus is on architecture design and optimization. Still, we will provide a brief introduction to JTC operations for easier understanding.

Optical lenses can perform a Fourier transform $\mathcal{F}[\tilde{E}(x, y, f)]$ on their back focal plane when an input image $\tilde{E}(x, y, f)$, illuminated by a coherent light source, is placed at the front focal plane [Goo05]. Utilizing the Fourier transform property of lenses, [WG66] introduced an optical JTC that generates optical convolution with both phase and amplitude components. A 1D on-chip photonic JTC can be derived from a traditional 2D optical JTC with minor modifications. There are five main components in a typical on-chip JTC system: (1) a 1D multi-channel input beam containing a signal $s(x + x_s)$ and a kernel $k(x - x_k)$ (with x_s and x_k representing the offsets of s and k from the origin in the x direction); (2) the first on-chip lens, which functions like a traditional free-space lens, to achieve the 1D Fourier transform $\mathcal{F}\left[s\left(x + x_s\right) + k\left(x - x_k\right)\right]$; (3) a *nonlinear function* unit (not the activation function of neural networks), realized using photodetectors and electro-optic modulators (EOM)

or non-linear materials to achieve a *square function* at the Fourier plane which is essential for JTC operation ; (4) the second on-chip lens, to transform the signal back to spatial domain; (5) photodetectors that detect the intensity pattern of the computed convolution by the JTC:

$$s(x + x_s + x_k) * k(-x) + s(-x) * k(x - x_s - x_k) + N(x) \tag{5.1}$$

, where $*$ denotes convolution. The first and second terms represent the computed auto-convolution between the two inputs. The third term $N(x)$, equals to $\mathcal{F}\left[|S(x)|^2 + |K(x)|^2\right]$, is a non-convolution term that can be spatially filtered out. Figure 5.1 illustrates the high-level diagram of a typical on-chip JTC system, which includes the five main components. Besides the 5 photonic components, DACs and ADCs are also required to convert the signals to and from the optical domain. The non-linear function in JTC, applied in the frequency domain after the first lens, is crucial for computing the convolution, as the output would be identical to the input without it (Fourier transform followed by inverse Fourier transform). The primary difference between on-chip JTC and conventional free-space JTC is the replacement of 2D lenses with 1D on-chip lenses, which results in the computation of 1D convolutions instead of 2D convolutions. This will be further discussed in Section 5.2.2. In this work, we assume the non-linear function is achieved through passive non-linear materials [MMH18, JT16, BCX15, ADB16], which is also used in the NG version of [LYW23a].

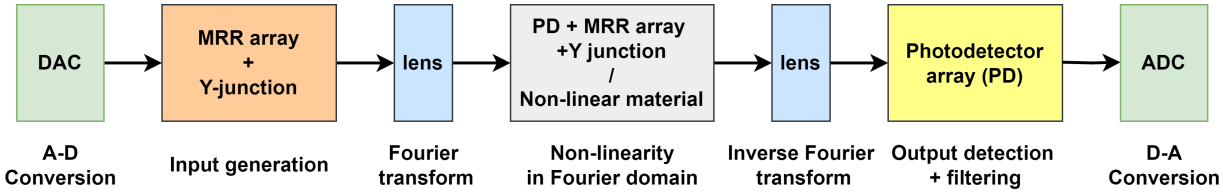


Figure 5.1: High-level diagram of a typical on-chip JTC system.

5.2.2 Computing 2D convolutions using 1D JTC

Unlike free-space 4F/JTC systems that naturally support 2D convolutions through the use of 2D Fourier lenses, their on-chip counterparts can only employ 1D on-chip metasurface-based lenses, and therefore, by default, only support 1D convolutions. To enable on-chip JTC systems to perform 2D convolutions, [LYW23a] proposed a generic algorithm for computing 2D convolutions using 1D convolutions, which is applicable to JTC systems. With this algorithm, 2D convolution can be computed using 1D convolution with no computation overhead for digital systems. For JTC-based systems, the supported 1D Fourier transform size needs to be large enough to avoid computation overhead. The core idea involves row tiling and partitioning, in which rows of 2D inputs and kernels are tiled with zero padding to form 1D inputs and kernels for 1D convolution. For $k \times k$ kernels, row tiling can be implemented if the JTC can accommodate at least K rows of inputs. This method can achieve identical results to conventional 2D convolutions when input rows are zero-padded with $k-1$ zeros per row and can closely approximate conventional 2D convolutions without zero-padding. While the 1D kernels needed to be zero-padded to the size of 1D input tiles, the zero-padding does not add overhead to JTC systems thanks to a unique property of JTC. For JTC, the actual convolution can be computed by the optical components passively, drawing almost no power. The computation cost comes from the input generation and output conversion part. For the zero-padding part, since all values are zero, the corresponding DACs and MRRs can be switched off so that no power will be consumed.

In cases where the JTC cannot hold k rows of inputs, 2D convolutions can still be computed by partially tiling or partitioning the input rows and taking multiple cycles to generate a single output row. The convolution results are identical to those obtained in the row tiling case but require more iterations. Since JTC can typically support a large number of input waveguides (256), and CNNs usually incorporate multiple pooling layers to reduce activation size, partial row-tiling or row-partitioning generally occurs only during the execution of the first layer, where activation sizes are large. Therefore, the overhead of

partial row-tiling and row-partitioning is negligible.

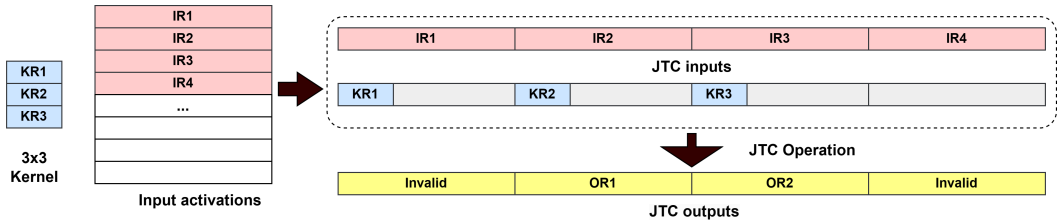


Figure 5.2: Illustration of how 2D convolution is computed using on-chip JTC system. IR, KR, and OR stand for input row, kernel row, and output row. The gray block represents zero-padding.

An example of performing 2D convolution with a 3×3 kernel using the on-chip JTC system is illustrated in Figure 5.2. In this example the input (activation) size is larger than the number of input waveguides in the JTC, therefore multiple iterations are required to compute the full convolution. The input is split into chunks and the rows in one chunk are tiled and loaded into the JTC. The kernel rows are padded to the same size as the input rows and also tiled and loaded into the JTC. The convolution between the tiled input rows and kernel rows completes in one cycle, and the output is received by the photodetectors and ADCs. Because of the circular padding nature of Fourier transform-based convolutions in JTC, only two output rows are valid in this example. Consequently, the invalid rows are discarded, constituting the primary source of computation overhead. The process is repeated multiple times to complete the convolution of the entire 2D input. The number of valid output rows is $R_i - k$ for $k \times k$ kernels, where R_i is number of input rows that can be tiled on the JTC. Therefore, the effective utilization is higher for larger JTCs and smaller input activations.

Comparing the amount of operations required for processing convolutions of digital systems (e.g., GPUs) and JTCs is non-trivial due to JTC's passive computation nature. However, if assuming the JTC's computational requirement is the number of input conversions needed, JTC with 256 input waveguides requires more than 5 times fewer computations than a GPU when computing a convolution between a 32×32 input and a 3×3 kernel. For JTC,

each pass can tile 8 rows and generate 6 valid outputs $(8-2)$, thereby requiring 6 JTC passes to compute the actual value. This leads to 1590 conversions in total $(6 \times (256+9))$ while GPU typically requires 9216 multiply-and-accumulate operations $(32^2 \times 3^2)$.

5.3 A case study for a typical JTC-based accelerator

In this section, we briefly introduce the baseline system of ReFOCUS, and analyze its bottlenecks while discussing how to further improve the efficiency of Fourier optics-based accelerators. We use a slightly modified version of PhotoFourier-NG (next-gen version) [LYW23a], the state-of-the-art Fourier-optics based photonic neural network accelerator, as our baseline system. The baseline system keeps the architecture of Photo Fourier-NG, which includes 16 JTCs in parallel, assumes monolithic integration of CMOS and photonics, and incorporates passive non-linear materials. The modification we made is to use citable sources for ADC and DAC power. The average power and the area of the baseline system are 15.7W and 116.3 mm^2 (90.7 mm^2 for photonic components) respectively.

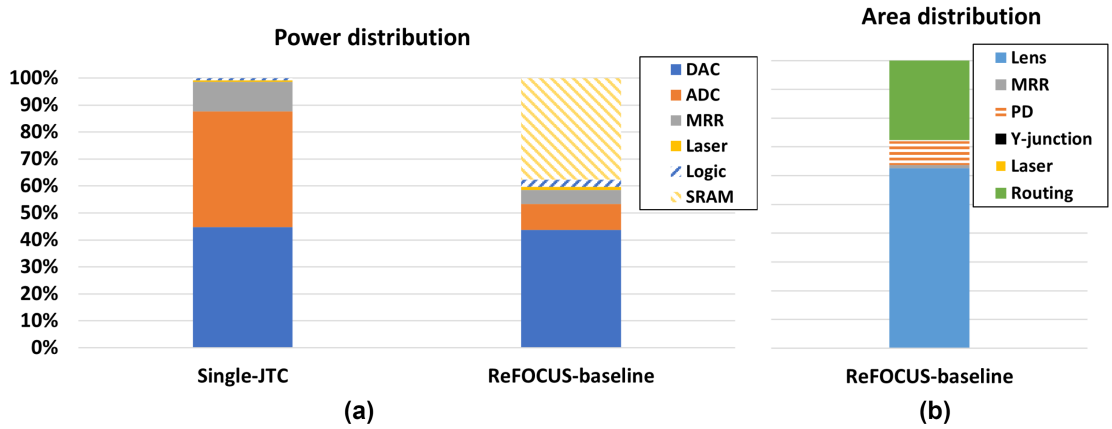


Figure 5.3: (a) Power breakdown of single JTC system and ReFOCUS-baseline. (b): Area breakdown of ReFOCUS-baseline, only photonic components are included.

The power breakdown comparison of a single JTC system (no optimizations) and our baseline system is illustrated in Figure 5.3 (a). It is evident that for the single JTC system,

the overall power consumption is dominated by ADCs and DACs (> 85%). ReFOCUS-baseline exhibits reduced ADC power due to the implementation of an optimization technique called temporal accumulation introduced in [LYW23a], which accumulates convolution results before ADC readout using photodetectors, resulting in a significant reduction of ADC power consumption. The DAC and SRAM access power constitute a large proportion of the total system power, highlighting the need for further optimization. By reducing their power consumption, the efficiency of the baseline system can be enhanced. Area-wise, as demonstrated in Figure 5.3 (b), the lens area dominates, consuming more than 50% of the total area. Therefore, reducing lens area is crucial for achieving better area efficiency.

In ReFOCUS, we propose optical reuse and WDM to mitigate the power consumption of both the DACs and the memory accesses, which are the two most dominating factors in total power consumption, thereby enhancing overall energy efficiency. These two optimizations, along with the architecture-level optimizations, will be discussed in detail in Section 5.4 and 5.5.

5.4 ReFOCUS Compute Unit

ReFOCUS comprises multiple compute units, which are named as ReFOCUS Compute Unit, or RFCU in short. Each RFCU is essentially a JTC system described in Section 5.2.1. The JTC configuration of the RFCU is kept the same as [LYW23a] unless related to optical reuse since this work focuses on optical reuse. Each RFCU has 256 input waveguides and 25 active weight waveguides (active means waveguides with DACs). On top of the baseline design, we introduce two main optimizations to improve energy efficiency and area efficiency.

5.4.1 Optical reuse

As discussed in Section 5.3, the ADC power can be reduced by temporal accumulation. However, this technique does not effectively reduce the DAC power, necessitating further

optimization. One approach to decrease DAC power is to reuse the optical signals generated by the DACs. Reusing can be easily achieved in digital electronics through data buffers, but this proves to be non-trivial in photonics/optics due to the absence of optical memory. Despite this, optical buffers can be achieved through the use of optical delay lines. Optical delay lines essentially consist of spiral waveguides that require light signals to travel a relatively long distance within the delay line, consequently causing a delay. The delay line length can be calculated by multiplying the speed of light by the target delay time. The waveguides are placed in a spiral shape to minimize the area, as depicted in Figure 5.4 (the red square). The light signal is split into two parts, and one part travels through the delay line to be reused at a later time, such that DACs do not need to be active when light is reused from the delay line, effectively reducing the average DAC power. To accomplish optical reuse, we propose two versions of optical buffer design based on optical delay lines, which have different use cases. In this work, both optical buffer designs will be adopted and evaluated, hence forming two versions of ReFOCUS - ReFOCUS-FB (feedback) and ReFOCUS-FF (feedforward).

5.4.1.1 Feedback optical buffer

The schematic diagram of the feedback version of the optical buffer design is depicted in Figure 5.4 (a), which comprises a delay line module, a switch MRR, and a Y-junction. The input signals generated by the DAC are divided into two parts by a Y-junction. One part is used for JTC computation, while the other is designated for reuse. The reuse signal passes through the optical delay line module and returns to be reused N cycles later, where N is determined by the delay line length. An MRR is required as a switch to control whether the feedback should be used for computation since, when a new input signal is generated by the input MRRs, the reuse signal should be blocked to avoid corruption of the final input. For instance, if a second Y-junction is employed to replace the switch MRR, the delayed optical will be added to the main signal that goes to the first Y-junction and the JTC even when the JTC is supposed to receive new input activations, causing data corruption. A switch

MRR can be turned off to block the feedback signal. When the switch MRR is turned on, the reuse signal will be coupled to the main waveguide connected to the Y-junction, and the input MRR should be turned off to avoid data corruption.

The advantage of this feedback approach is that, theoretically, the signals can be reused as many times as desired, which can maximize the reuse and significantly cut down the DAC power. However, one potential limitation of this design is that the signal power of the feedback signal will be lower with every iteration due to the Y-junction and the delay line loss. Define the power split ratio of Y-junction as α (percentage of input power directed to the JTC), and the delay line loss as l_d, the relationship between the signal power that goes into the JTC can be derived as:

$$X_i = (1 - l_d) \cdot (1 - \alpha) \cdot X_{i-1} \tag{5.2}$$

, where X_i is the signal power that goes into the JTC for the ith iteration. The overall signal loss for every reuse iteration l_t is hence $(1 - l_d) \cdot (1 - \alpha)$. The signal power of a particular iteration can then be calculated as:

$$X_i = ((1 - l_d) \cdot (1 - \alpha))^i \cdot X_0 \tag{5.3}$$

, where X_0 is the signal power of the initial input to the JTC.

Assuming the input activations are reused, typically different convolution filters will be processed each time the input is reused. That means different filters will see inputs with different magnitudes, which are supposed to be the same. Since the power reduction of the signals for each iteration is fixed and can be pre-determined, a hardware-aware scheduler can be designed to adjust the weights of the filters according to Equation 5.1, and the convolution outputs will be scaled back in the digital domain. In this case, the number of times the same signal can be reused is determined by the laser power overhead (average laser power will be higher to compensate for the loss due to delay lines), the dynamic range of photodetectors,

and ADCs. This will be further analyzed in Section 5.5.4.

5.4.1.2 Feedforward optical buffer

In addition to the software solutions, there is a hardware solution to address the issue of the reduction of power of the reused signals, at the cost of the amount of achievable reuse. This solution involves using a feedforward optical buffer, as depicted in Figure 5.4 (b). The difference between the feedback version is that the delayed signal is not connected back to the input of the Y-junction; instead, it goes directly to the JTC through a second Y-junction. The second Y-junction is used to connect the delayed signal back to the main waveguide. A switch MRR is not required in this design, as there are no signal loops - which means the delayed signal does not need to be blocked. In this design, the split ratio α of the Y-junction can be configured to make the signal power of the original signal and the delayed signal identical. The signal power that directly goes to the JTC (without delay) is $\alpha \cdot X$, where X is the signal power before the first Y-junction. The signal power of the delayed signal is $(1 - l_d) \cdot (1 - \alpha) \cdot X$, where l_d is the loss of the delay line module. By equating the two signal powers, the split ratio can be calculated:

$$\alpha = \frac{1 - l_d}{2 - l_d} \tag{5.4}$$

By configuring the split ratio according to Equation 5.4, the original signal and the delayed (reused) signal will have the same signal power. This design eliminates the need for weight and activation scaling. However, the signal in this design can only be reused once since there is no feedback loop, which is the main limitation of the feedforward optical buffer.

5.4.1.3 Reusing signal or weight

In the context of neural networks, the JTC receives two signals to compute their convolution: inputs (activation) and weights. Consequently, there is a choice of whether to reuse inputs or

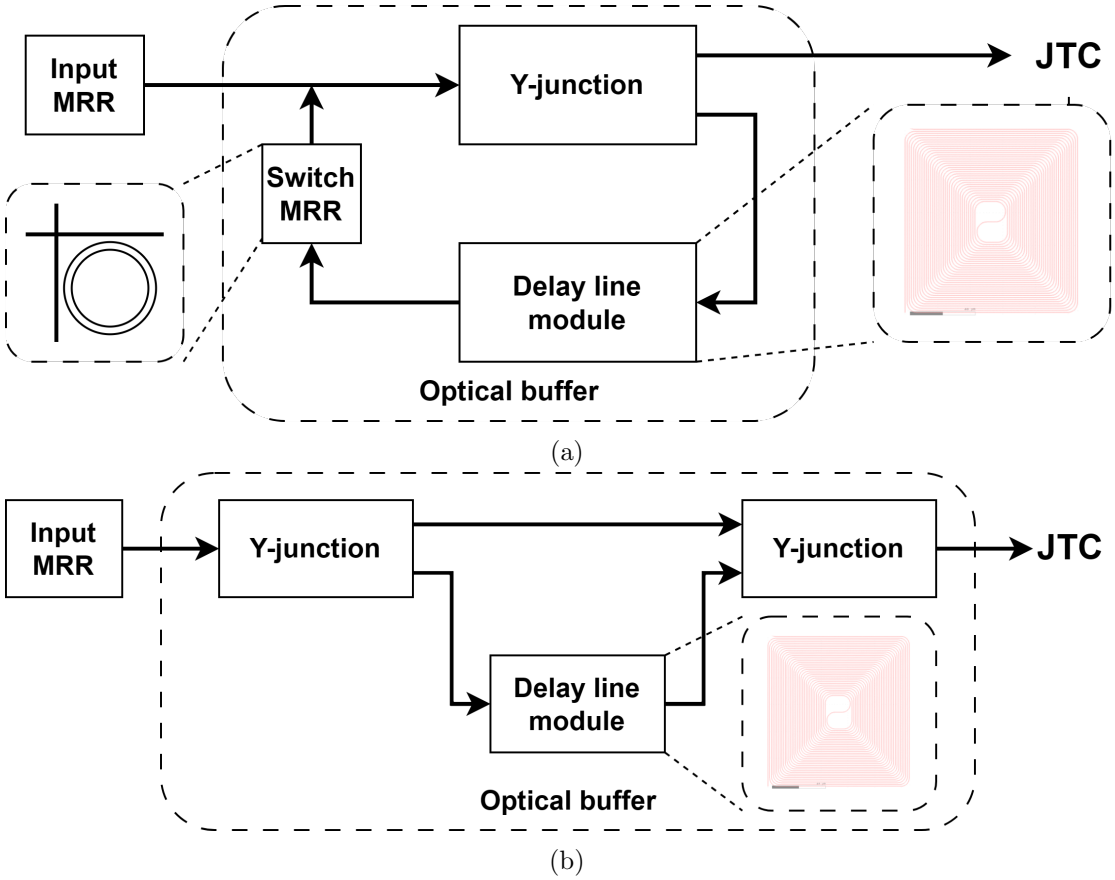


Figure 5.4: Schematic diagram of two versions of optical buffers used in ReFOCUS. (a): Feedback version. (b): Feedforward version.

weights. Assuming the processing of a 3×3 kernel, the number of input DACs is 256, while the number of weight DACs is 9 for a single JTC. Even considering the entire system and assuming input is fully broadcasted to 8 or 16 RFCUs, the number of input DACs remains significantly larger than the number of weight DACs. Therefore, reusing inputs will have a greater impact on power efficiency than reusing weights.

Furthermore, reusing weights can lead to lower-than-expected performance improvement. For inference with a batch size of 1, if weights are reused, the only option is to process different input activation tiles for each iteration since they share the same weight. However, the JTC tile size is usually large (e.g., 256) for more inherent weight reuse within the JTC, while the input activation size can be small for later layers of CNNs due to pooling. For instance,

ResNet-34 has 18 layers with input activation sizes small enough that the entire input can be loaded into a single JTC together, which means there will be no opportunity for temporal weight reuse at all. Reusing inputs will not have this problem, as the number of filters of modern CNNs is far larger than the number of filters that can be executed in parallel on ReFOCUS so that each cycle can process different filters.

5.4.1.4 Longer delay lines

Temporal accumulation can significantly reduce ADC frequency by accumulating the outputs of multiple cycles at photodetectors before the ADC readout, enabling the ADC to operate at much lower power. However, output stationary (OS) dataflow is required for temporal accumulation to function properly, as only the outputs of individual channels can be accumulated. The introduction of optical buffers to reuse inputs means the dataflow needs to be adjusted accordingly. Assuming the inputs are only delayed by 1 cycle, then input stationary dataflow will be enforced, and temporal accumulation cannot be implemented. If input reuse is achieved at the cost of removing temporal accumulation, it will not be an ideal design choice, as the increase in ADC power will have a greater impact than the reduction of DAC power.

Nevertheless, with a longer delay line and dataflow optimization, temporal accumulation can still be implemented by accumulating the results while the reused inputs are traveling through the delay lines. An alternating dataflow (OS + input stationary (IS)) is required to implement temporal accumulation with a delay line. The maximum amount of temporal accumulation that can be achieved in terms of cycles is the same as the delay line length in terms of cycles. The alternating dataflow, choice of the exact length of the delay line, as well as how many times an input signal will be reused for ReFOCUS-FB, will be discussed in detail in Section 5.5.

5.4.1.5 Overhead of optical buffer

The components used in both versions of the optical buffers are passive, except for the switch MRR used in the feedback optical buffer, which consumes significantly less power compared to a high-speed DAC. Therefore, the power overhead of optical buffers is small (excluding laser power). However, the area overhead cannot be ignored, as the delay line modules are large in size, particularly if the signals need to be delayed for an extended period to implement temporal accumulation. Table 5.1 lists the length, area, and loss of the delay line which can delay the signal by one cycle for a 10 GHz system (0.1 ns). The delay line area is around $0.01mm^2$, which constrains the number of inputs that can be delayed and the number of cycles that can be delayed. The area overhead of optical buffers can be compensated through lens sharing and architecture-level optimizations, both of which will be discussed later.

In addition to the area overhead, delay lines also attenuate the signal. The total signal power loss is directly proportional to the delay line length. The average laser power will be higher compared to the case without optical buffers to compensate for the power loss caused by the delay line module, which will be discussed further in Section 5.5. With low-loss on-chip delay lines [LCL12], the delay line loss is not significant for any reasonable delay line lengths.

Table 5.1: The length, area, and loss of a delay line with 0.1 ns delay (1 cycle in 10 GHz system).

# Length (mm)	Area (mm^2)	Loss (dB) [LCL12]
8.57	0.01	6.94e-3

5.4.2 Lens sharing

5.4.2.1 Motivation

The optical buffer allows input reuse, which can reduce the DAC and memory access power of inputs. However, the delay line comes with a non-negligible area overhead. For a 16-RFCU system (each with 256 input waveguides), if the inputs of each RFCU are buffered individually for 8 cycles, the total delay line area is 327.7 mm^2, which is about 3× larger than the whole system area without the delay line and is clearly infeasible. The optical buffer area can be dramatically reduced through input broadcasting. By placing the optical buffer before the Y-junction tree that broadcasts the inputs, the total delay line area can be reduced to 20.48 mm^2 assuming full input broadcasting. Even in this case, the optical buffer still adds around 20% area overhead to the system.

To improve the overall area efficiency of the system, lens optimization is required, which accounts for around 50% of the total system area, as shown in Figure 5.3 (b). Wavelength-division multiplexing (WDM) is a common approach used in photonic designs to enhance parallelism and area efficiency, although it is primarily used in dot-product style on-chip photonic neural network accelerators and has not yet been applied to Fourier optics. WDM works by transmitting multiple data channels encoded into different wavelengths on a single waveguide, thus saving area. Furthermore, operations applied to the waveguide, such as phase change and delay, are effectively broadcasted to all wavelengths (data channels). For JTC, the Fourier transform implemented by the lens can also be broadcasted to the wavelengths through WDM, effectively sharing the lens. In this work, we leverage WDM to share lenses and photodetectors with different wavelengths, significantly improving the area efficiency of ReFOCUS.

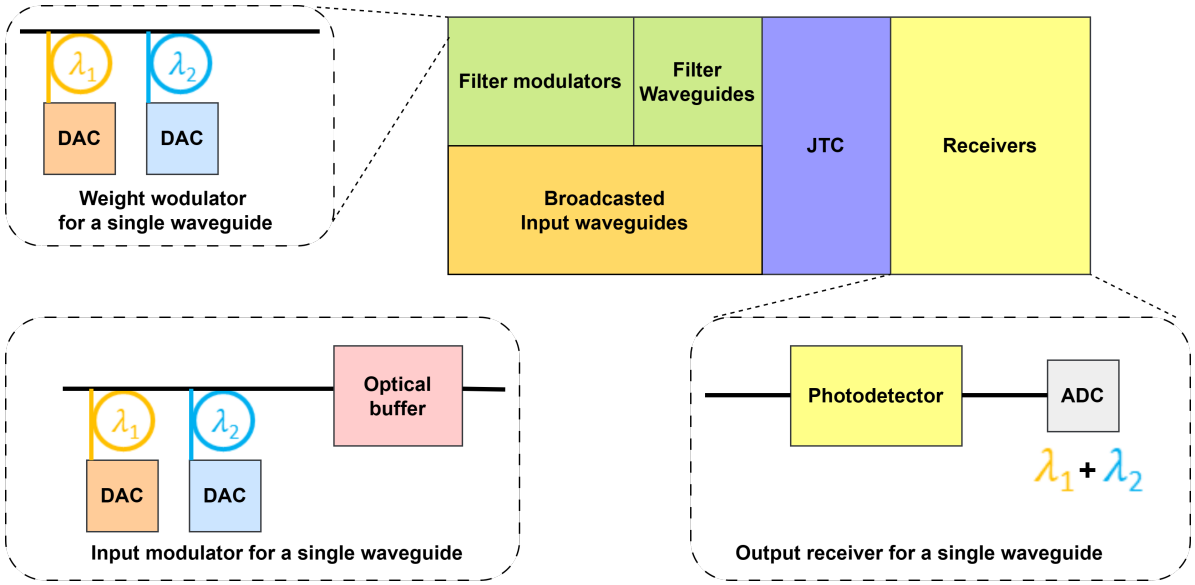


Figure 5.5: Illustration of how WDM is implemented in ReFOCUS (not drawn to scale). Two wavelengths are modulated and encoded into a single waveguide through MRRs with different wavelengths, for both filters and inputs generation. Photodetectors and ADCs receive the sum of the convolution output of the two wavelengths.

5.4.2.2 Implementation

When WDM is used for optical communications, encoders and decoders are required to encode and decode the signals. Both can be implemented with MRR arrays, with each MRR corresponding to one wavelength. For each wavelength, two MRRs are required for modulation/encoding and decoding, and one photodetector, ADC, and DAC. In the context of neural network acceleration, some of the components described above can be shared because of the reuse opportunity inside neural networks. Depending on the exact dataflow, either input DAC/MRR, weight DAC/MRR, or photodetector/ADC can be shared with different wavelengths. In ReFOCUS, each wavelength processes a single convolution channel, hence their convolution results can be directly accumulated. The decoder is no longer required in this case - the waveguide that contains multiple wavelengths is directly connected to a single photodetector and the convolution results of different wavelengths/channels are accumulated by the photodetector. The wavelengths should be selected to be close to each

other so that their convolution results can be detected by a single photodetector. Figure 5.5 illustrates how WDM is implemented in ReFOCUS. In this example, 2 wavelengths are encoded into a single waveguide through MRRs with wavelength λ_1 and λ_2, for both inputs and weight generation. The photodetector receives the sum of the convolution results of both wavelengths.

In this implementation, the photodetector and ADC can be shared, and extra decoding MRRs are not required, which means WDM can improve area efficiency and power efficiency at the same time. We choose to share ADCs rather than input or weight DACs for the following reasons: (1) As previously discussed, broadcasting weights to different activation tiles is not guaranteed, especially for later layers of CNNs, while inputs are already broadcasted to different RFCUs (and further reused through optical buffer). (2): Photodetectors can also be shared when sharing ADCs, which is not possible for the other two cases. Sharing photodetectors further improves area efficiency as they are around 10× larger than MRRs. Define N_λ as the number of wavelengths used, and in this dataflow, N_λ input channel needs to be generated. Since delay lines can also be shared by all wavelengths, processing multiple input channels will not cause excessive area overhead of optical buffers. WDM is applicable to both optical buffers, including the feedback version, as the switch MRR can react to a range of wavelengths.

5.4.2.3 Number of wavelengths

However, there is a limit on how many wavelengths can be used, and is relatively low for ReFOCUS. Having too many wavelengths can cause the spread of the convolution results of all wavelengths too large to be captured by a single photodetector, and our simulation suggests that the number of wavelengths should be less than 4. Besides, more wavelengths will make accessing inputs from memory/buffer challenging due to the huge number of data that needs to be accessed every cycle, as using temporal accumulation means inputs need to be accessed every cycle regardless of optical reuse. Considering both factors, we set $N_\lambda = 2$,

that is using two wavelengths. With WDM, each RFCU essentially contains two 'virtual' JTCs and has 2× throughput, but only requires a single set of lenses and photodetectors.

Table 5.2: Area and normalized area efficiency in terms of frames per second per mm^2 of a 16-RFCU system with different wavelengths.

# wavelengths	Area (mm^2)	Normalized FPS/mm^2
1	111.3	1.00
2	115.2	1.93

Even with just two wavelengths, significant area efficiency can be achieved. Table 5.2 shows the area and normalized area efficiency of a 16-RFCU system with 1 or 2 wavelengths. Adding a second wavelength only increases the area by 3.5%, while doubling the throughput. Combining these together, a 1.93× area efficiency is achieved by WDM. The reason for this extremely low area overhead of WDM is that the Fourier lens and the photodetectors can be shared, which together consume a large proportion of the total system area.

5.5 ReFOCUS Architecture

The high-level architecture and configuration of ReFOCUS are introduced first in this section, followed by optimizations and design choices.

5.5.1 Overall architecture

ReFOCUS has two versions, ReFOCUS-FF (feedforward) and ReFOCUS-FB (feedback). The difference between the two versions is at the RFCU level - ReFOCUS-FB reuses inputs 15 times while ReFOCUS-FF reuses inputs once, and both versions share the same high-level architecture. The architecture diagram of ReFOCUS is illustrated in Figure 5.6. ReFOCUS operates at 10 GHz, supports 8-bit precision, and assumes monolithic integration of CMOS and photonics [RMN20]. There are 16 RFCUs within ReFOCUS, with each RFCU containing 256 input waveguides and processing two wavelengths concurrently through WDM. Input

signals first pass through optical buffers and then broadcast to all RFCUs, while weights are generated within each RFCU. ReFOCUS adopts 16-cycle temporal accumulation to reduce the frequency of ADC and the output processing CMOS circuits to 625 MHz. On the CMOS part, each RFCU has two corresponding CMOS processing units that are used to generate the inputs, process the outputs (reading from ADC, scaling and accumulating the results, and implementing the ReLU non-linearity), and communicate with memory. ReFOCUS has a 4MB global activation SRAM shared with all RFCUs, while each RFCU has its own 512 KB weight SRAM. Input and output data buffers are added to reduce the access energy of the shared activation SRAM. The design choices such as dataflow, number of RFCUs, data buffer configuration, and delay line lengths, will be further discussed in this section.

5.5.2 Memory hierarchy

ReFOCUS adopts a similar top-level memory configuration as [LYW23a], with a 4MB shared activation SRAM and separate local weight SRAMs (one for each RFCU with 512KB size). he activation and weight SRAM sizes are configured to hold the entire activation/layer of weights of common CNNs [SZ14, HZR16] to eliminate the need for writing to DRAMs during execution. This relatively large SRAM size also results in $> 4\times$ access energy compared to weight SRAM. Directly accessing and storing from/to the shared activation SRAM as [LYW23a] leads to excessive SRAM power, as shown in Figure 5.3 (a). In ReFOCUS, we add input and output data buffers to reduce the memory access power. All RFCUs share a single input buffer because of input broadcasting, while each RFCU has its own output buffers. The size and relative access energy of the data buffers depends on the dataflow, and is further discussed in Section 5.5.3.

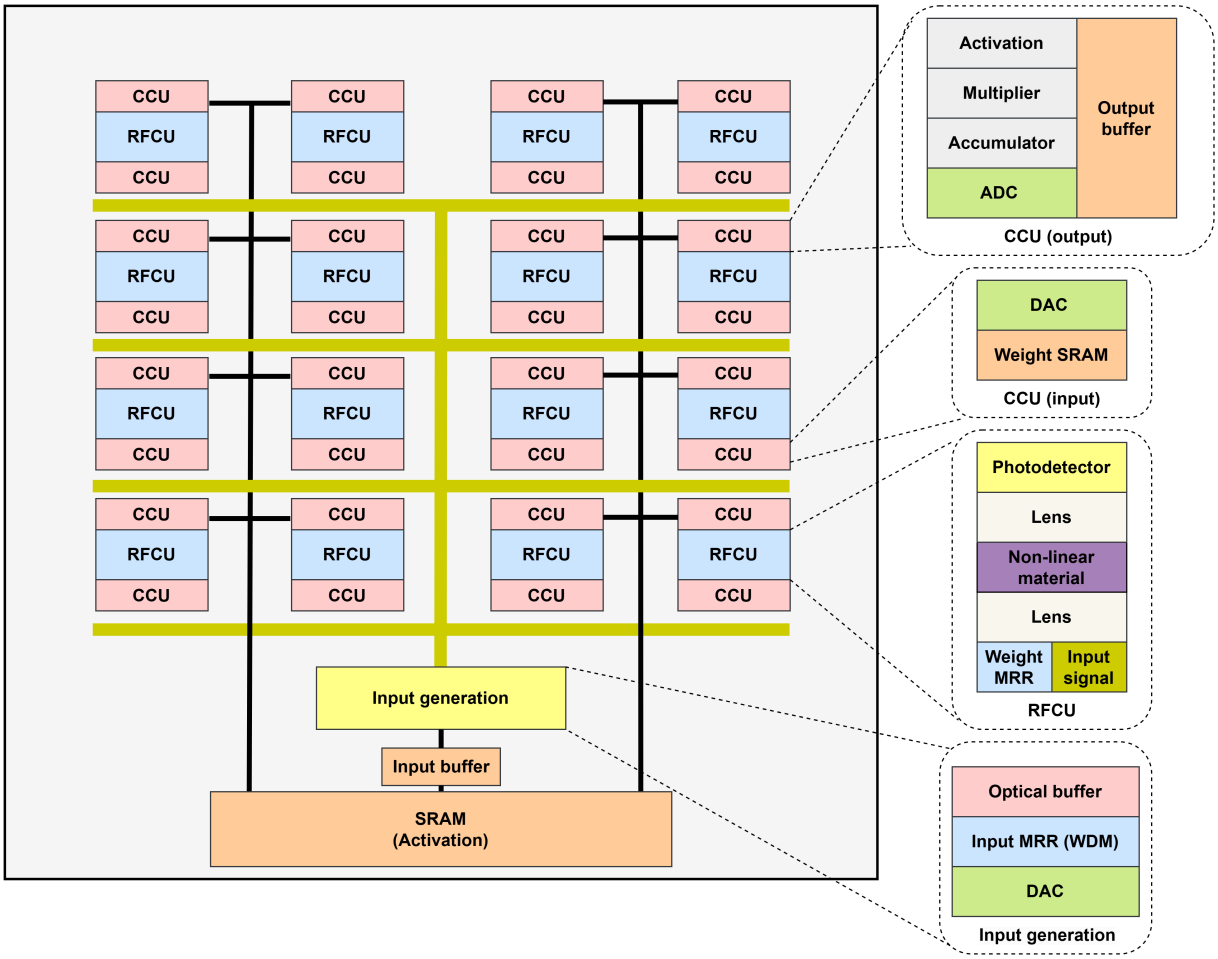


Figure 5.6: High-level architecture diagram of ReFOCUS. CCU stands for CMOS compute unit. Each RFCU has two CCUs, one for input generation and the other for output processing. The diagram is not drawn to scale.

5.5.3 Dataflow

5.5.3.1 Parallelization scheme

Input broadcasting the default parallelization scheme in ReFOCUS, and the main reason is to reduce the input DAC power. Output reuse is achieved through temporal accumulation while weight reuse is inherently achieved by the JTC operation (kernel is 'broadcasted' to the entire input tile). Therefore, inputs are broadcasted to all RFCUs to achieve input reuse. Within an RFCU, WDM is implemented to compute two input channels in parallel,

for reasons discussed in Section 5.4.2.

5.5.3.2 Alternating dataflow

Dataflow plays a critical role in ReFOCUS, as many optimizations have constraints or requirements related to dataflow. Temporal accumulation, which reduces ADC frequency and power, requires an OS dataflow to accumulate the output of different convolution channels using the photodetector. However, the optical buffers, which optically reuse the inputs, enforce an IS dataflow. While the two dataflow seem contradicting, they can be combined together to form an alternating dataflow with some modifications on the optical buffer.

The solution is to increase the delay line length so that inputs will be delayed by M cycles before being reused. Within the M cycles, there are no restrictions on the exact dataflow, and OS dataflow can be used to implement temporal accumulation for M cycles, by processing an input channel group of M channels. After M cycles, the same input channel group is reused, and another filter needs to be processed to achieve input reuse. The dataflow is illustrated in detail in Figure 5.7, which shows the dataflow of an example system with WDM and feedforward optical buffer with $M = 4$. Each RFCU processes a unique filter, and for the 8-RFCU system in this example, 8 filters will be processed in parallel, therefore when the input channel group is reused in RFCU1, filter 9 is processed. Within an RFCU, spatial accumulation is achieved by WDM, where each wavelength processes a different channel group of a filter. With this OS-IS alternating dataflow, output reuse (through temporal accumulation) and input reuse (through optical buffer) can be achieved concurrently.

5.5.3.3 Optimizing for efficient memory accesses

The input channel group can only be reused a limited number of times (reuse once for the ReFOCUS-FF). Thus, after reuse completes and new inputs need to be generated, there is a choice of what should be processed next. There are two dataflow choices: (1) follow the

current pattern to process another filter until all filters are processed for the current input channel group, as illustrated in Figure 5.7 and (2) process another input channel group of the first filter being processed in the RFCU, until all the channels are fully processed for the current filters being processed. These two dataflow choices have different impacts on the SRAM data buffer design and the overall power efficiency. (1) requires relatively small input buffers and large output buffers while (2) requires relatively large input buffers and small output buffers.

Table 5.3: Notations and definitions of common terms used in the analysis.

Notation	Definition
M	Delay line length in terms of cycles
R	How many times the signal is reused
N_{RFCU}	Number of RFCUs
T	Input tile size (number of input waveguides)
N_λ	Number of waveguides.

Some common notations and their definitions used in the analysis are listed in table 5.3. For case (1), the input and output buffer size (per RFCU) in bytes can be calculated as (ignore ping-pong buffer for now):

$$B_{in1} = T \times M \times N_\lambda, B_{out1} = T \times \frac{N_F}{N_{RFCU}}$$

, where N_F is the maximum number of filters per layer of a neural network. For case (2), the input and output buffer size can be calculated as:

$$B_{in2} = T \times N_C \times N_\lambda, B_{out2} = T \times (R + 1)$$

, where N_C is the maximum number of channels per layer of a neural network. In ReFOCUS, we adopt (1) as our dataflow, which favors the input buffer over the output buffer. The reason behind this design choice is the input buffer needs to have a higher frequency than the output buffer and hence has higher constraints on access latency. The input buffer needs

to be accessed every cycle (although when input is being reused the input buffer will not be accessed at all), while the output buffer only needs to be accessed once per M cycle. A large input buffer may not meet the latency requirement. Besides, for ReFOCUS-FF, the input buffer has more accesses overall compared to the output buffer, as there is more output reuse than input reuse (discussed more in Section 5.5.4). Thus, having a smaller input buffer reduces the cost of input buffer accesses, and improves the overall power efficiency of the ReFOCUS-FF.

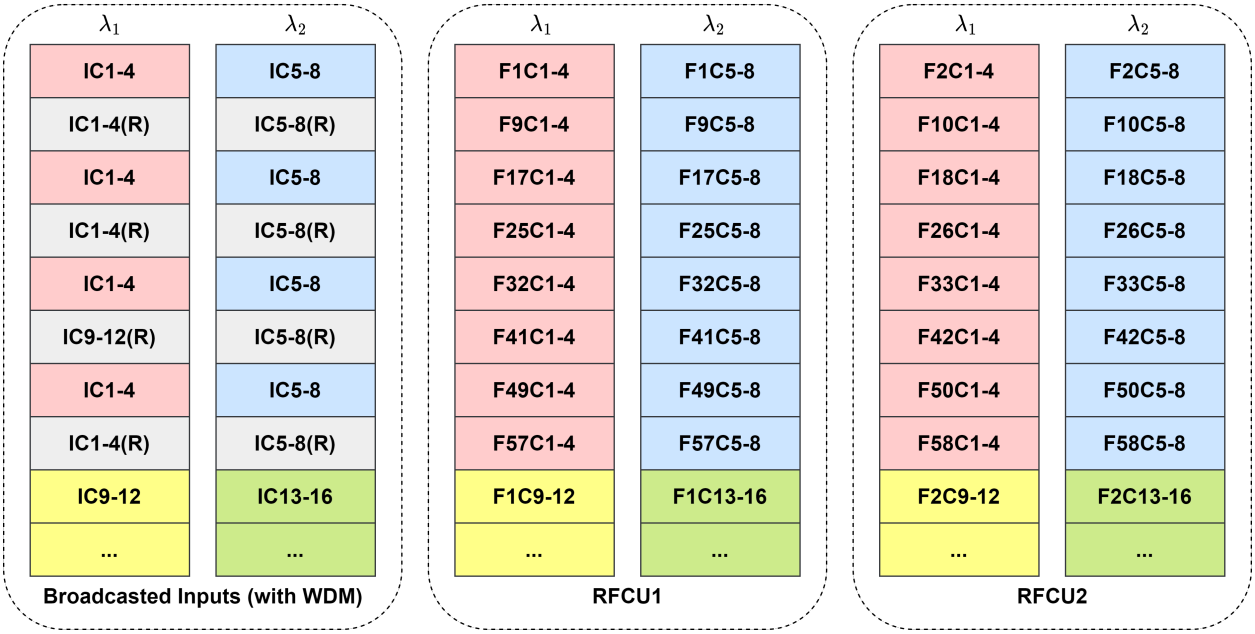


Figure 5.7: Dataflow used in ReFOCUS. An 8-RFCU system that implements feedforward optical buffers with 4-cycle delay lines and WDM with 2 wavelengths is assumed for this example. IC($a-b$) refers to input channel $a-b$, F(x)C($a-b$) refers to channel $a-b$ of filter x. λ_i refers to the ith wavelength in WDM. R means reused input signal through the optical buffer. Difference channel groups are marked with different colors.

5.5.4 Choice of design parameter

Some of the design choices such as which signal to reuse, how WDM is implemented, and the number of wavelengths used are already discussed in Section 5.4. This section discusses other design choices that cannot be determined individually as they have dependence or

impact on each other and require system-level analysis, such as delay line length, number of RFCUs, and how many times the inputs should be optically reused in ReFOCUS-FB.

5.5.4.1 ReFOCUS-FF

From Equation 5.4, the Y-junction split ratio α for the feedforward buffer can be computed. Based on α, it can be derived that the average laser power needs to be $1/2\alpha \times$ larger (divided by 2 because the light is reused once). Based on the delay line loss from Table 5.1, and the fact that laser power per channel is much smaller than the DAC power, the increase in laser power caused by a longer delay line will have a negligible impact on overall power efficiency for any reasonable delay line lengths. Therefore, the primary overhead of longer delay lines is the increase in area. A longer delay line can result in fewer RFCUs that can be placed within a given area limit.

Nearly all previous studies on on-chip photonic neural network accelerators have reported chip areas of less than $200mm^2$ [SMN21, SKB21, LYW23a, LLY19, ZLY20]. Increasing the chip area can lead to yield and cost issues, while providing diminishing returns in terms of performance. Therefore, we set the area budget of the photonic components of ReFOCUS to be $150mm^2$ (leaving some margin for CMOS components), and calculate the maximum number of RFCUs that can be placed for various delay line sizes within the area budget. Since optical buffer has impacts on both power and area, we develop a custom efficiency metric to take into account both power efficiency and area efficiency. The metric is simply the product of frames per second per watt and frames per second per mm^2, and is named PAP (power-efficiency-area-efficiency-product). The geo-mean of relative PAP on four different CNNs (VGG-16, ResNet-18, ResNet-34, ResNet-50) is calculated to determine the optimal delay line length and number of RFCUs, and the results are shown in Table 5.4, along with relative FPS/W and FPS/mm^2,. The change in laser power is modeled in the calculation. The results suggest that when the signals can be delayed by 16 cycles, the optimal efficiency can be achieved, with 18 RFCUs. Thus, we configure ReFOCUS-FF to have 16 RFCUs. We

choose 16 rather than 18 as 16 is a power-of-two value and fits better with neural network execution.

Table 5.4: Number of RFCUs can be placed and relative FPS/W, FPS/mm^2, PAP for different delay line lengths in terms of cycles, for both ReFOCUS-FF and ReFOCUS-FB. The absolute values are shown for the baseline case where $M = 1$.

M	1	2	4	8	16	32
N_{RFCU}	25	24	23	21	18	11
FPS/W (FF)	1 (237)	1.92	2.83	3.71	4.51	4.72
FPS/mm^2 (FF)	1 (196)	1.00	0.97	0.91	0.80	0.53
PAP (FF)	1 (4.6e4)	1.92	2.75	3.39	**3.61**	2.52
FPS/W (FB)	1 (247)	2.00	3.07	4.18	5.20	5.17
FPS/mm^2 (FB)	1 (196)	0.99	0.96	0.91	0.80	0.53
PAP (FB)	1 (4.8e4)	1.98	2.96	3.80	**4.14**	2.75

5.5.4.2 ReFOCUS-FB

There is an additional design choice for ReFOCUS-FB, which is how many times the inputs are reused before generating new ones (R). The choice of R solely depends on the signal loss of the optical buffer, and the related change in average laser power and dynamic range of reused signals (ratio of the initial signal power and the power of the last reused signal).

Unlike ReFOCUS-FF, laser power overhead is not trivial without optimizations for ReFOCUS-FB, even with a low delay line loss. Since the signal power will be smaller for each reuse iteration due to the Y-junction, a relatively large initial laser power is required to make sure the last reused signal (the one with the lowest power) is detectable by the photodetector. In this scheme, all signals except for the last reused signal have higher than the required signal power, which makes the average laser power much higher than the case without optical buffers, especially when the split ratio α is 50%. The average laser power overhead and the dynamic range can be calculated based on Equation 5.3 and are reported in Table 5.5 for different number of reuses and α. Without optimizing the α, reuse 7 or more times is infeasible as it can lead to $> 38\times$ average laser power and $> 153\times$ dynamic range. Even

ignoring the laser power overhead, the dynamic range is too large for an 8-bit ADC which has just 256 levels.

However, this issue can be resolved by setting α to $1/(R+1)$, the optimal Y-junction split ratio for the feedback optical buffer. As shown in Table 5.5, the relative laser power and the dynamic range are both 3.05 for reusing 7 times. Therefore, significantly more optical reuse can be achieved with this modification. If only power-of-2 values are considered (to fit the structure of CNNs better), reusing the signal higher than 15 leads to diminishing returns on overall power efficiency, while increasing the dynamic range of the signal. Thus, ReFOCUS-FB reuses the input signals optically 15 times, to achieve a balance between power efficiency and effective output precision. Once R is determined, the delay line length (M) and the number of RFCUs can be decided in the same way as ReFOCUS-FF, and the results are shown in Table 5.4. The optimal choices of M and N_{RFCU} are the same as ReFOCUS-FF, thus these two designs share the same system architecture.

Table 5.5: Relative laser power when compared to the system without optical buffer and the dynamic range of input signals for different R and α.

R	1	3	7	15	31	63
			$\alpha = 1/(R+1)$			
relative LP.	2.05	2.56	3.05	3.87	5.96	13.7
dynamic range	2.05	2.56	3.05	3.87	5.96	13.7
			$\alpha = 0.5$			
relative LP.	2.05	4.32	38.4	6.0e3	3.0e8	1.5e18
dynamic range	2.05	8.64	153	4.8e4	4.8e9	4.7e19

5.6 Evaluation

We employ Cadence Genus along with a commercial 14nm library to model the power and area of the CMOS components. We use CACTI [MBJ09] to model the area, leakage power, and access energy for all the SRAM memory and buffers used in ReFOCUS. We develop

a custom simulator based on Python to simulate the throughput and energy consumption of ReFOCUS on CNN inferences, and also model the area of ReFOCUS. The simulator integrates the CMOS and SRAM simulation results and then models the photonic part based on the characteristics of photonic components used in ReFOCUS. Table 5.6 lists the power and area of the components used in ReFOCUS. Since there are no reported ADCs/DACs that have the exact same specifications as we assumed in ReFOCUS, we find 8-bit, 14 nm ADC and DAC, with higher frequency than ReFOCUS required, and then linear scale down the power accordingly frequency, which is a conservative approach as the relationship between frequency and power is not linear. The average DAC power is calculated by multiplying the power reported in [LHC22] with the duty cycle of DAC in ReFOCUS. Since JTC-based system can only process positive weights, ReFOCUS implements pseudo-negative processing, which splits a filter into two parts, one positive and one negative. The negative part is processed as a positive filter and the results are subtracted from the results of the positive part digitally. This approach addresses the positive-weight limitation, but doubles inference latency. We only benchmark the convolution layers of these networks, which correspond to more than 99% of total computation.

5.6.1 Power and area

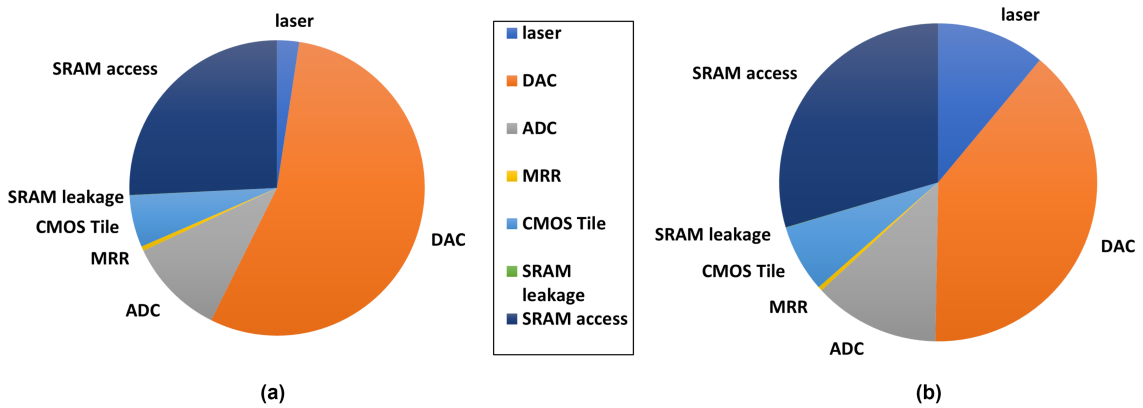


Figure 5.8: (a): Power breakdown of ReFOCUS-FF. (b): Power breakdown of ReFOCUS-FB. The same legend is applied to both pie charts.

Table 5.6: Power of active components and the area of photonic components used in ReFOCUS.

Component power (mW)	
MRR	0.42 [MLW17]
Laser (min) *	0.1 per waveguide
ADC @ 625 MHz	0.93 [LHC22]
DAC @ 10 GHz	35.71 [CMA20]
Optical component area (μm^2)	
MRR	255 [LYW23a]
Photodetector	1920 [LYW23a]
Y-junction	2.6 [ZYL13]
Laser	1.2e5 [DJB13]
Delay line (0.1 ns delay)	1e4
Lens	2e6

*: Minimum laser power required. The average laser power will be higher to compensate for the loss of optical buffers.

For power evaluation, we benchmark ReFOCUS on 5 CNNs (AlexNet [KSH12], VGG-16 [SZ14], ResNet-18,34,50 [HZR16]), and the average system power is calculated. Overall, ReFOCUS-FF and ReFOCUS-FB consume 14.0W and 10.8W average power respectively. The difference is caused by the further reduction of input DAC energy, as ReFOCUS-FB has more optical reuse. Figure 5.8 shows the power breakdown of ReFOCUS-FF and ReFOCUS-FB. In both systems, DAC still consumes the most power, but the proportion is reduced in the FB version. For the FB version, DAC power is dominated by weight DAC, which consumes 90% of total DAC power, preventing further reduction of DAC power through input reuse. As a result of computation becoming more efficient, SRAM access energy consumes a large proportion of total power in both cases, which would be even larger without data buffers. ReFOCUS-FB has significantly higher laser power compared to ReFOCUS-FF, as the laser power needs to be scaled to compensate for the loss of the feedback optical buffer. Further improving the system power requires reducing the weight DAC power, and we will briefly discuss this in Section 5.7.3.

Figure 5.9 shows the area breakdown of ReFOCUS, which is applicable to both versions

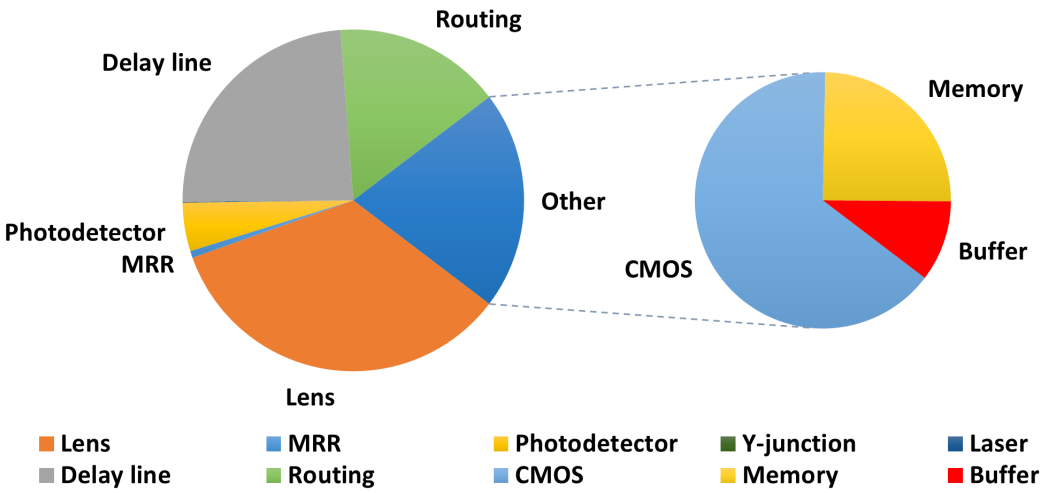


Figure 5.9: Area breakdown of ReFOCUS. The secondary pie chart shows the area breakdown of non-photonic components (CMOS, SRAM memory, and data buffers).

of ReFOCUS as they have the same area. ReFOCUS has a 171.1 mm^2 overall area, with 135.7 mm^2 contributed by the photonic components. SRAM memory and data buffers together consume 12.4 mm^2 area, and the rest chip area is contributed by CMOS logic and ADCs/DACs. On the photonic side, lenses (58.5 mm^2) and delay lines (41.0 mm^2) are the two largest contributors. WDM reduces the lens area of ReFOCUS by 2×, making it possible to fit 256 16-cycle delay lines with no area overhead. Further increasing the delay line length will make its area overhead too large to be compensated, and leads to lower system efficiency.

5.6.2 Effect of optimizations

Table 5.7 shows the potential reuse (spatial and temporal) that can be achieved through different optimizations for the baseline system and two versions of ReFOCUS. With the proposed WDM and optical buffer, both outputs and inputs can be further reused when compared to the baseline, hence reducing the conversion energy.

Figure 5.10 shows the relative FPS/W on ResNet-34[HZR16] of ReFOCUS with different optimizations enabled compared to ReFOCUS-baseline with the same architecture (similar

Table 5.7: Potential reuse can be achieved by different optimizations. OB stands for optical buffer and TA stands for temporal accumulation.

	Input reuse		Output reuse	
	Broadcast	OB	WDM	TA
Baseline	16 ×	N/A	N/A	16×
ReFOCUS-FF	16 ×	2×	2×	16×
ReFOCUS-FB	16×	16×	2×	16×

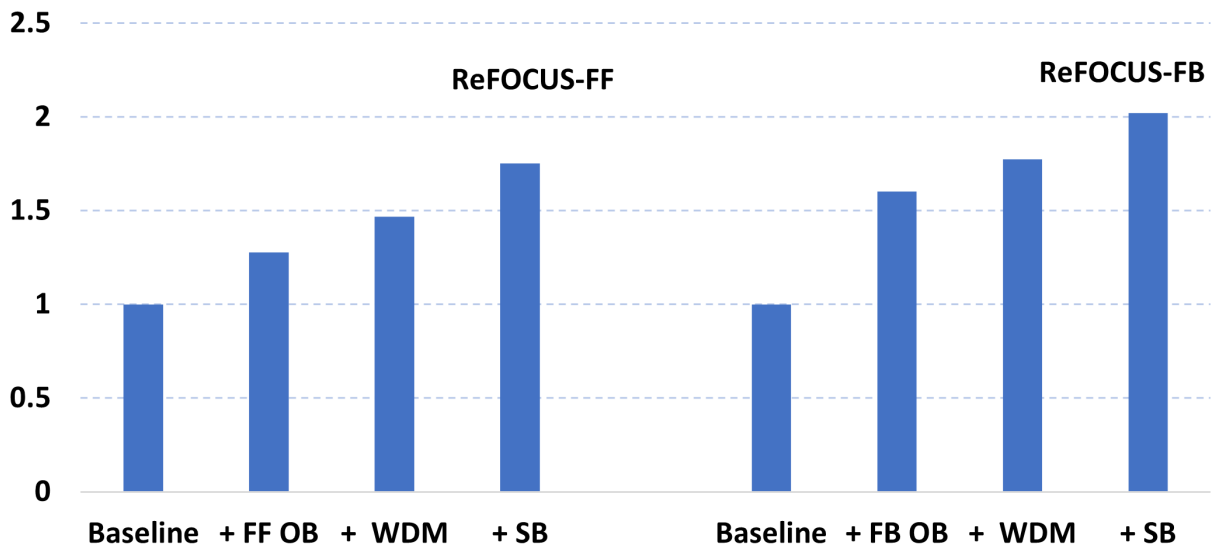


Figure 5.10: Relative FPS/W for ReFOCUS with different optimizations. Each column includes the optimizations that are reported on its left. Resnet-34 is used for this benchmark. OB stands for optical buffer while SB stands for SRAM buffer.

to [LYW23a]). All three optimizations proposed (optical buffers, WDM, and SRAM data buffers) improve the overall power efficiency noticeably. The SRAM buffers provide substantial benefits because the power of ADCs/DACs is optimized by optical buffers, WDM, and temporal accumulation, making SRAM power consume a larger proportion of total system power (36.9% for ReFOCUS-FB without data buffers). When comparing to the baseline system that scaled to the same throughput (with a much larger area), the absolute power of converters (ADC + DAC) for ReFOCUS-FB is 1.72× smaller, demonstrating the effectiveness of optical reuse to reduce the power overhead of A/D and D/A conversions. Overall,

both ReFOCUS versions achieve significant performance improvement compared to the baseline system, with ReFOCUS-FB being 2× more efficient.

5.6.3 Comparison with prior work

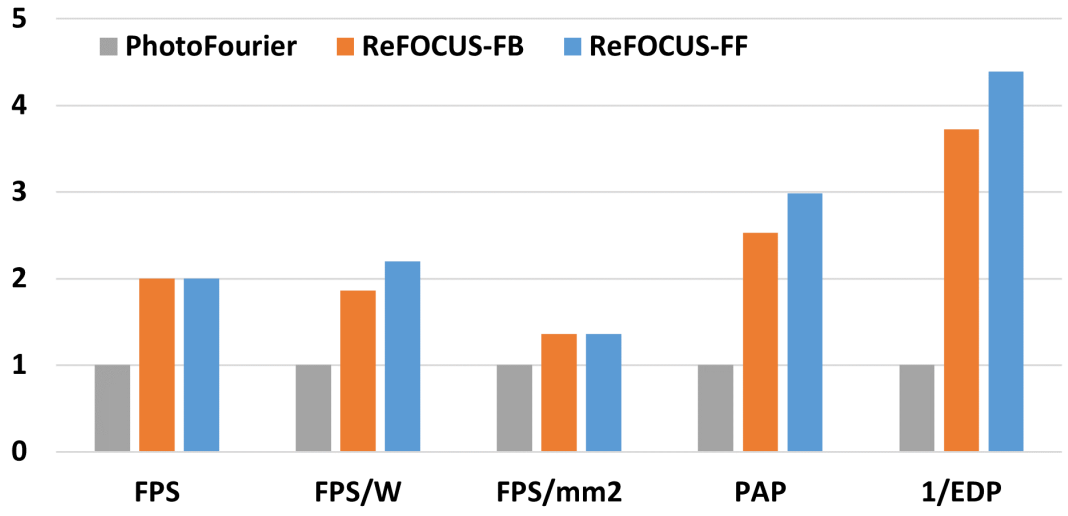


Figure 5.11: Two ReFOCUS versions compared to PhotoFourier, in terms of relative FPS, FPS/W, FPS/mm^2, PAP, and inverse of EDP. Benchmarked on 5 CNNs.

We primarily compare ReFOCUS with PhotoFourier for two reasons - (1) PhotoFourier is the most closely related prior work as both PhotoFourier and ReFOCUS are based on JTC, and (2) PhotoFourier reports state-of-the-art efficiency results for on-chip photonic neural network accelerators. To make the comparison as fair as possible, we obtain the simulator from the authors of PhotoFourier, and implement a slightly modified version of PhotoFourier for comparison, which uses our power and area number for individual components and adopts non-linear material for optical nonlinearity. We evaluate the systems on the 5 CNNs mentioned earlier, and the geometric mean of key metrics is calculated.

Figure 5.11 shows the relative improvements of ReFOCUS over PhotoFourier, in terms of throughput (FPS), power efficiency (FPS/W), area efficiency (FPS/mm^2), and two combined efficiency metrics - PAP (introduced earlier) and $^1/_{\text{EDP}}$ (inverse of energy-delay product).

Both ReFOCUS-FF and ReFOCUS-FB achieve better results on all metrics compared to PhotoFourier, demonstrating the efficiency of ReFOCUS. The FPS of ReFOCUS is roughly doubled since ReFOCUS processes two wavelengths concurrently in each RFCU. For the same reason, ReFOCUS achieves better area efficiency even though delay lines add a large area overhead. Energy-wise, ReFOCUS-FB achieves more than 2× FPS/W compared to PhotoFourier, thanks to the extra input and output reuse achieved through the optical buffer and WDM. ReFOCUS-FF also has close to 2× efficiency. Since ReFOCUS has higher throughput, power, and area efficiency, naturally, ReFOCUS achieves significantly better PAP and $^1/_{\text{EDP}}$.

We also compare ReFOCUS with two other 8-bit precision photonic neural network accelerators, Albireo [SKB21] and Holylight-m [LLY19] in terms of FPS and FPS/W on AlexNet, VGG-16, and ResNet-18. For reference purposes, we further compare ReFOCUS with a digital accelerator (UNPU) [LKK19] and one RRAM-based accelerator [WWL19]. The results are shown in Figure 5.13, some results are missing as some works did not report results on all three networks. Similarly, ReFOCUS achieves the best results on both metrics. ReFOCUS achieves up to 25× power efficiency compared to state-of-the-art MZI/MRR-based photonic neural network accelerator Albireo, and achieves up to 145× power efficiency compared to Holylight-m. The large performance gap between Fourier-optics based accelerators such as ReFOCUS and PhotoFourier and the MZI/MRR style accelerators demonstrates the superiority of Fourier-optics on CNN acceleration.

To better demonstrate the advantage of ReFOCUS, we conduct a comparison with some well-known digital accelerators from both industry and academia, namely, the NVIDIA H100 GPU [h1023], Google TPU V3 [tpu23], Simba [SCV19], and a design from JSSC 20 [ZVS20], on the relatively large ResNet-50 network. The FPS results of H100 and TPU V3 are collected from the MLPerf benchmark [RCK20]. Figure 5.12 illustrates the FPS and FPS/W results. While H100 and TPU V3 exhibit better raw throughput compared to ReFOCUS, it is essential to consider their significantly larger footprint as a contributing factor. However,

in terms of power efficiency (FPS/W), ReFOCUS has a clear advantage over existing GPUs and ASIC accelerators, boasting an efficiency that is $5.6 - 24.5$ times higher.

The efficiency advantage over digital and other photonic accelerators mainly stems from the complexity reduction of JTC, the passive calculation of Fourier transforms, and the reduced conversion cost due to the reusing of light signals. When compared to RRAM-based analog accelerators that have limited write endurance, high write latency/energy, and are usually network-specific due to the necessity to unroll the network into numerous fixed crossbar arrays, ReFOCUS presents much better programmability and flexibility while still having more than $2\times$ efficiency. Rather than being network-specific like RRAM-based accelerators, ReFOCUS allows weights to be fully programmable at high speed during runtime, akin to digital accelerators.

5.7 Discussion

5.7.1 Instruction scheduling

While optical buffers introduce complexity to the system and dataflow, potentially complicating scheduling, the buffer size (delay line length) and latency in ReFOCUS are fixed. Given the strictly first-in-first-out behavior of the optical buffer, its behavior can be predetermined, allowing scheduling to be offloaded to the compiler. Consequently, the compiler can manage the instruction scheduling statically, akin to Very Long Instruction Word (VLIW).

5.7.2 Compensating system noise

Inherent in analog computing, noise and non-idealities cannot be entirely avoided in photonic neural network accelerators. However, system noise can be mitigated through careful design, placement, and calibration of photonic components. Moreover, the noise impact can be further compensated by modeling and injecting noise during training. This approach enables

the trained neural network to learn and adapt to various noise behaviors and non-idealities.

5.7.3 DRAM, weight sharing, and weight DAC

Almost all prior works on photonic neural network accelerators did not report DRAM energy, which is often a major contributor to system power. We discover that when the computation and on-chip memory access are efficient enough, DRAM access power cannot be ignored. For example, DRAM access power can contribute more than 50% of total power in ReFOCUS-FB, when profiled with HBM2 access energy [OCL17]. For neural network layers with small activation sizes but a large number of filters, DRAM energy dominates, even though ReFOCUS already minimizes DRAM accesses (no DRAM writes). Without reducing DRAM access energy, further optimizing computation or on-chip memory access leads to diminishing returns. Besides developing better DRAM technology (e.g., HBM3), there are also software solutions to reduce DRAM access energy, such as weight sharing.

Neural network weight sharing: Weight sharing [SNL18, WWW18, DNO22, LG22, DNO20, UMW17] is an effective compression technique for neural networks that outperforms quantization and pruning while maintaining accuracy. It uses a smaller codebook and index matrix, reducing storage needs. In CNNs, various weight sharing methods exist [SNL18, LG22, WWW18]. Sharing 2D convolution kernels [SNL18] with a trainable scaling factor can achieve a 4.5× compression ratio compared to 8-bit weights in ReFOCUS, with negligible accuracy loss. This method reduces DRAM access energy by 4.5× and overall energy by up to 52%. Weight SRAM access energy is also lowered due to smaller weight memory.

Channel Reordering: In ReFOCUS-FB and ReFOCUS-FF, the weight DAC accounts for 90% and 53% of the DAC power consumption, and 42% and 31% of the total system power on ResNet-34, respectively. Weight sharing in 2D convolution kernels presents an opportunity to decrease weight DAC power and thereby enhance system efficiency. To capitalize on this, we reorder the input channels and group those that are assigned to the same kernel. This minimizes the weight DAC operations, although the degree of reduction is constrained by

factors like input broadcasting and reuse. We further introduce a Simulated Annealing-based algorithm for channel reordering, achieving a 15% reduction in weight DAC power for ReFOCUS-FF under a typical setup and boosting the overall power efficiency by 4.7%.

5.7.4 Non-CNN tasks

While we primarily focus on accelerating CNNs in this work, recently there are many works proposed Fourier-transform based transformer [LAE21, GML21] and convolution-based transformer [WXC21, YGL21], which can be potentially accelerated by JTC-based systems as they share similar underlying operations as CNNs. Further work is required to adapt JTC-based architecture for these transformer models, which will be a part of our future work.

5.7.5 Slow light

One concept that has been used to design area-efficient optical delay lines is called 'slow light'. The speed of light is significantly reduced as it propagates through a medium in this type of delay line, achieved by manipulating the properties of the medium. With a lower light speed, the length of the waveguide, and hence the delay line area can be greatly reduced. There are works that reported 'slow light' based delay lines with promising area efficiency [CBA22, XWN18]. Given the number of cycles that inputs can be delayed is constrained by the delay line area, having more compact delay lines will further improve the system efficiency. Slow slight-based delay lines are not used in ReFOCUS as they currently have relatively large loss [CBA22] and require further development.

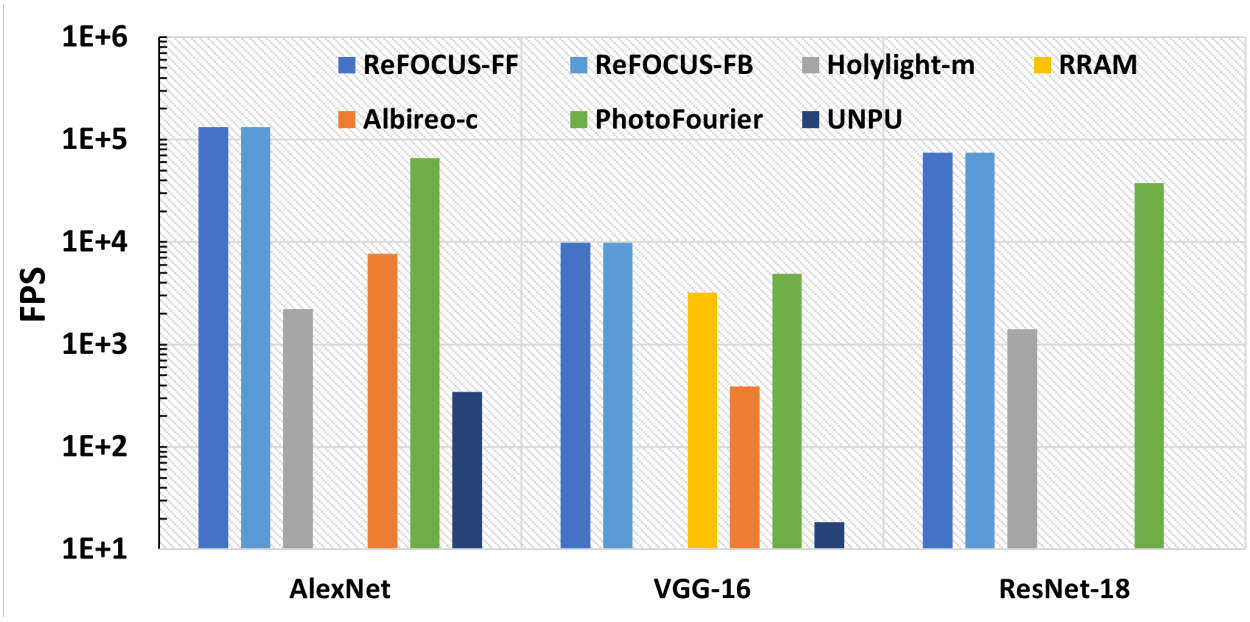
5.8 Related Work

As mentioned in Section 5.1, on-chip photonic neural network accelerators can be roughly split into two categories - dot product or matrix multiplication accelerators based on MRRs/MZIs, and convolution accelerators based on Fourier-optics. PhotoFourier [LYW23a], being the only published Fourier-optics based accelerator so far, is the most closely related prior work. Hence, we extensively compare ReFOCUS to PhotoFourier. PhotoFourier proposed the first on-chip JTC based neural network accelerator and demonstrated state-of-the-art power efficiency. It uses plain JTCs as building blocks that do not feature WDM or optical buffer and the performance advantage mostly comes from the complexity reduction of JTC. In contrast, ReFOCUS innovatively integrates two versions of optical buffers and WDM. This distinct approach substantially enhances both the area and power efficiency of JTC-based accelerators, thus differentiating ReFOCUS from PhotoFourier. Besides, ReFOCUS further optimizes the dataflow and memory hierarchy to improve power efficiency. Other on-chip photonic neural network accelerators [SKB21, SMN21, ZLY20, LLY19, SWK20, MAS18] are fundamentally different than ReFOCUS as ReFOCUS leverages the convolution theorem to reduce the complexity of CNNs through Fourier optics.

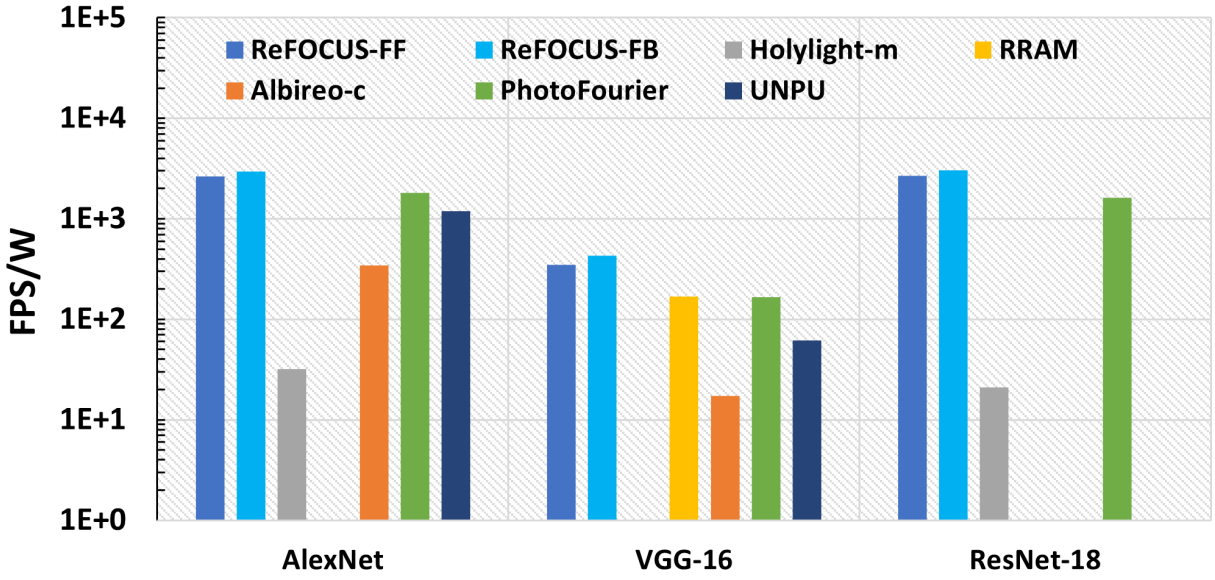
5.9 Conclusion

In this paper, we introduce ReFOCUS, a Fourier-optics on-chip photonic neural network accelerator featuring optical reuse. We present two innovative optical buffer designs tailored to enhance light reuse and energy efficiency. To mitigate the area overhead of optical buffers, we incorporate WDM in ReFOCUS, significantly improving the area efficiency of the system. Compared to state-of-the-art photonic neural network accelerators, ReFOCUS demonstrates remarkable gains: 2× throughput, 2.2× energy efficiency, and 1.36× area efficiency. Furthermore, ReFOCUS achieves over 25× power efficiency when compared to photonic neural network accelerators not utilizing Fourier optics, highlighting its potential

for future high-performance computer-vision applications.

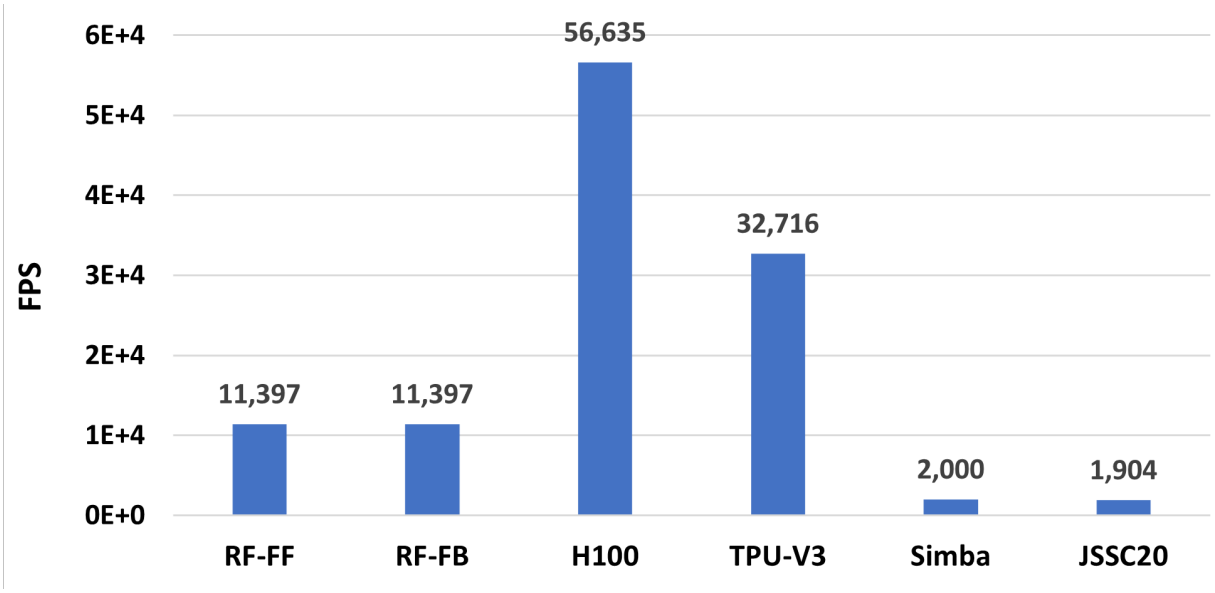


(a)

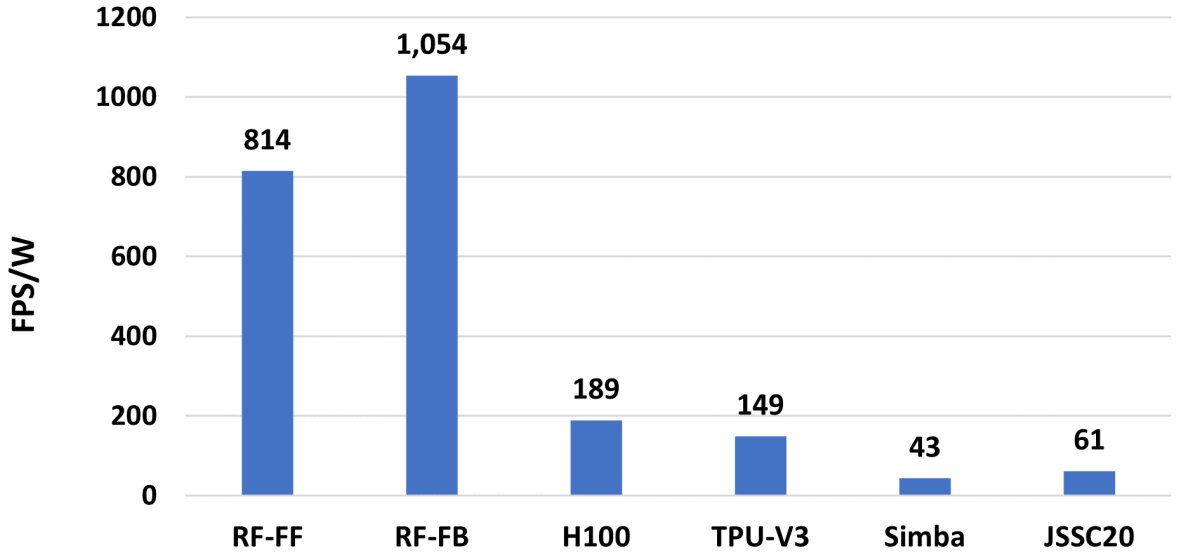


(b)

Figure 5.12: ReFOCUS compared with other accelerators. The logarithmic axis is used. (a): FPS. (b): FPS/W.



(a)



(b)

Figure 5.13: ReFOCUS compared with different digital accelerators on ResNet-50. RF stands for ReFOCUS. (a) FPS. (b) FPS/W.

CHAPTER 6

Bit-serial Weight Pools: Compression and Arbitrary Precision Execution of Neural Networks

Applications of neural networks on edge systems have proliferated in recent years but the ever-increasing model size makes neural networks not able to deploy on resource-constrained microcontrollers efficiently. We propose bit-serial weight pools, an end-to-end framework that includes network compression and acceleration of arbitrary sub-byte precision. The framework can achieve up to 8× compression compared to 8-bit networks by sharing a pool of weights across the entire network. We further propose a bit-serial lookup based software implementation that allows runtime-bitwidth trade-off and is able to achieve more than 2.8× speedup and 7.5× storage compression compared to 8-bit networks, with less than 1% accuracy drop.

6.1 Introduction

The ever-increasing size of neural network models and rapid proliferation of machine learning in resource-constrained edge devices have catalyzed research into a variety of model compression techniques, as well as software and hardware acceleration of deep learning on edge devices.

General-purpose microcontrollers have been a platform of choice for edge devices due to their low power, low cost and programmability. However, this comes at the cost of limited memory: these processors usually do not have any DRAM and often have less than 2MB

total memory (SRAM + Flash); and small available compute power: these processors usually have small datapaths and simple pipelines running at modest clock rates. This makes the execution of complex machine learning models on this ubiquitous class of processors very challenging. A variety of model compression techniques have, therefore, garnered attention in the embedded machine learning community [BCD21].

Weight sharing [NH92] as a model compression technique shares a set of weight vectors across the entire neural network, so that only the indices of the shared weight vectors need to be stored, instead of actual weight values. For convolutional neural networks (CNNs), weight sharing methods can achieve compression ratios between 4-16x, compared to 8-bit baselines. Since weight sharing does not modify the structure nor the precision of the network, it can be combined with other compression techniques like pruning and quantization to further improve compression ratio and runtime. Furthermore, recent works [CWV18, BNH18] have shown that sub-byte quantization of weights and/or activations can achieve inference accuracy comparable to full-precision networks.

Though weight sharing and sub-byte quantization are both promising for storage and runtime improvement, neither has native support in microcontroller-class general purpose processors commonly deployed in edge devices. As a result, these compression techniques can often hurt performance rather than improve it. For instance, processing a neural network with sub-byte precision naively can lead to worse runtime due to bit unpacking overhead [HZL18]. Hence, there is a need for optimized software implementations of weight-shared neural networks, as well as methods that can support and accelerate sub-byte precision neural networks on microcontrollers.

In this chapter, we present a framework for efficiently deploying large neural networks on small microcontrollers. The proposed framework contains two parts. The first part is neural network compression, where a pool of weight vectors (e.g., a 1×8 8-tuple of weights) along channel dimension are shared across the entire network. We refer to networks using our weight sharing method as weight pool networks in the rest of this chapter. The second part

of the framework is the software implementation of weight pool networks on microcontrollers, where we utilize bit-serial lookup tables to support and accelerate weight pool networks with 8-bit or lower activation bitwidth. The main contributions can be summarized as follows.

- We show that z-dimension weight pools, as small as 512 total parameters can realize popular networks such as ResNet and MobileNet with negligible accuracy loss.
- We develop a bit-serial lookup based method for efficient arbitrary-precision execution of weight pool networks on general purpose microcontrollers. This delivers 2.38X speedup (compared to well-optimized ARM CMSIS-NN library [LSC18]) at 8-bit precision and even greater speedup at lower bitwidth on popular neural networks.
- We explore the design-space of weight pool networks experimentally to develop an optimized software implementation of weight pool networks targeted for small, memory-starved microcontrollers.
- We show that weight pool arbitrary precision networks can be 2.8X faster and 6.51X more compact than CMSIS on ResNet-10, with less than 1 percent drop of accuracy on CIFAR-10, and better compression and speedup can be achieved on larger networks.

The next section outlines the motivation behind the bit-serial weight pool approach.

6.2 Addressing Compression and Quantization Challenges for General Purpose Processors

Compression with weight pools. Our weight pool networks essentially store vectors of weights along the channel dimension as one entry. The 3D filters used in CNNs would then be composed of these vectors. For instance, a $3 \times 3 \times 32$ filter would use $3 \times 3 \times 4 (= 36) 1 \times 8$ weight vectors selected from the available pool of weight vectors. There is no limitation on the reuse of vectors. Weight pool networks would reduce the parameter storage from the

total number of parameters in the network to the total size of the weight pool. If done correctly, this can reduce parameter storage requirements of neural networks by orders of magnitude with minimal accuracy drop. Furthermore, the parameter storage here becomes independent of network size.

However, naively implementing weight pool networks would likely worsen inference latency because of additional memory reads (some form of index storage lookup followed by the actual weight lookup) with no reduction in total number of operations. One could try reducing the number of operations by directly storing the results of the (partial) dot product on the weight pools. For a pool vector size of 8, it would replace 8 multiply-accumulate operations with one memory lookup. Unfortunately, for 8-bit activations, this would require a lookup table size of 2^{8^8} entries for just one pool vector which is impractical.

Arbitrary precision computation using bit-serial arithmetic. Like conventional neural networks, the activation bitwidth of weight pool networks can be reduced to sub-byte regions while still achieving decent accuracy on many tasks. The sub-byte activation bitwidth provides an opportunity to improve the runtime and overall energy efficiency.

Sub-byte precision is not well supported in most microcontrollers (or most processors in general). Naively implementing networks with sub-byte activation bitwidth is not useful as it would worsen runtime because of the bit unpacking overhead with no actual compute reduction (since underlying hardware still executes higher precision arithmetic).

To support and accelerate neural networks with sub-byte activation bitwidth, bit-serial multiplication seems to be a suitable candidate since it processes a multiplication serially by looping through all the bits of one operand. The runtime of bit-serial multiplication is proportional to the bitwidth of the bit-unrolled operand. There are many bit-serial multiplication based hardware neural network accelerators [LRG21, JAH16, SPS18], but there is no support of bit-serial multiplication in microcontrollers due to the lack of bit-serial multipliers.

Bit-serial-lookup-based weight pool networks. We address the challenges outlined above by doing bit-serial execution *but* saving computation by lookup of partial dot product results on pool vectors. Since activations are processed one bit at a time (most significant to least significant bit), the dot product lookups only need to be on 1-bit operands. Therefore, the lookup table for activation bitwidth of 8 bits is just 2^8 entries. This would replace 8 multiply-accumulate operations with 8 memory reads and accumulations. Later we show how despite this, substantial runtime reduction can be achieved by careful implementation optimizations leveraging the value reuse properties of weight pools. Furthermore, reducing activation bitwidth now just amounts to truncating the temporal bit-serial execution earlier which gives proportionate further runtime improvement.

6.3 Bit-Serial Weight Pool Methodology

Figure 6.1 shows the high-level flow of the proposed framework, which is split into two parts. The left block shows the compression part, where the input is a pretrained CNN. The corresponding weight pool and weight indices (original weights are converted to indices of the weight pool) are generated and the pretrained CNN is hence compressed. Analysis of minimum activation bitwidth of the compressed CNN is carried out afterward. Finally, the dot product lookup table is generated from the weight pool, and loaded into microcontrollers' flash memory along with weight indices and precision information. The compression part is entirely executed on the host side and the generated weight pool CNN is sent to the microcontroller.

The second part is CNN inference acceleration, which is executed on the microcontroller. At this stage, the original CNN has already been compressed and transformed into weight pool CNN, and the activation bitwidth has been determined. The framework uses a bit-serial lookup table based algorithm to accelerate the inference of weight pool CNNs, and is able to further improve runtime by reducing the activation bitwidth. The rest of this section

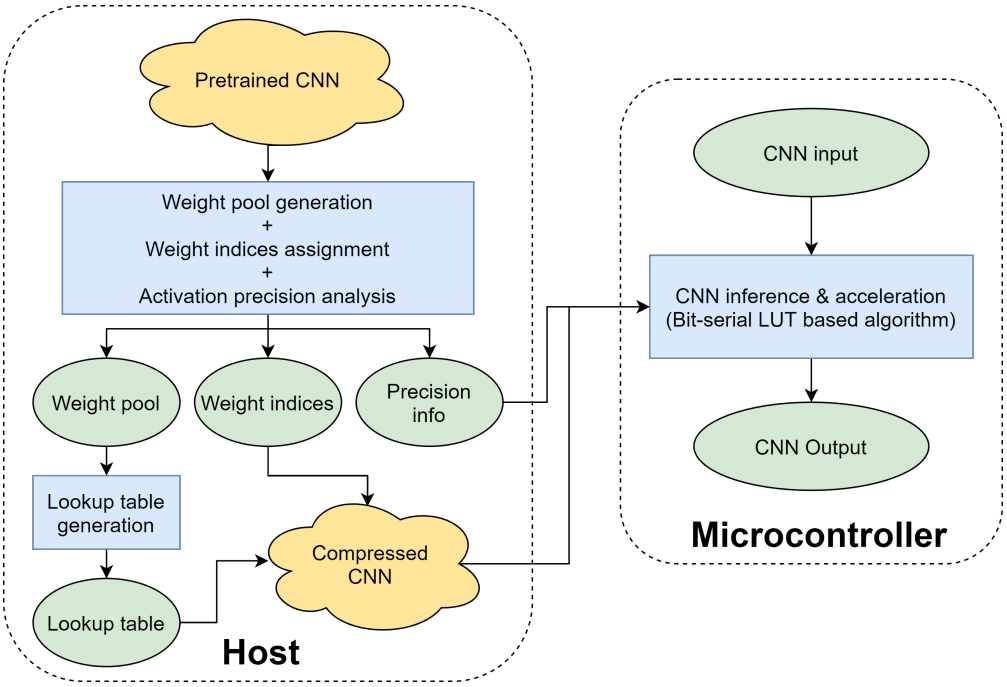


Figure 6.1: High level flow of the proposed framework. Pretrained weights are clustered into weight vectors pool, any fine tuning and activation bitwidth selection are done offline. At inference time, the processor only stores the weight pool dot product results and indices to weight vectors used in the network.

describes each of these steps in detail.

Weight pool networks achieve compression by sharing a fixed pool of weight vectors among all the layers of a network, so that the network only needs to store indices of the weight pool, plus the weight pool itself. In this work we use a weight sharing pipeline similar to [SNL18] to generate weight pool CNNs, but instead of clustering 2D convolutional kernels, we apply the clustering algorithm along the z-dimension of a 3D filter (clustering across the filter channels) as shown in Figure 6.3. Figure 6.2 shows the proposed training pipeline. The pretrained weights are grouped into 1×8 weight vectors along the channel dimension and clustered using K-means clustering (with a cosine distance metric to avoid scaling dependence). After the clustered weight pool is generated, the original CNN's weights are converted to the indices of the weight vectors in the weight pool. The network is retrained to fine-tune the weight indices assignment (with a fixed weight pool) and fully connected layer's weights.

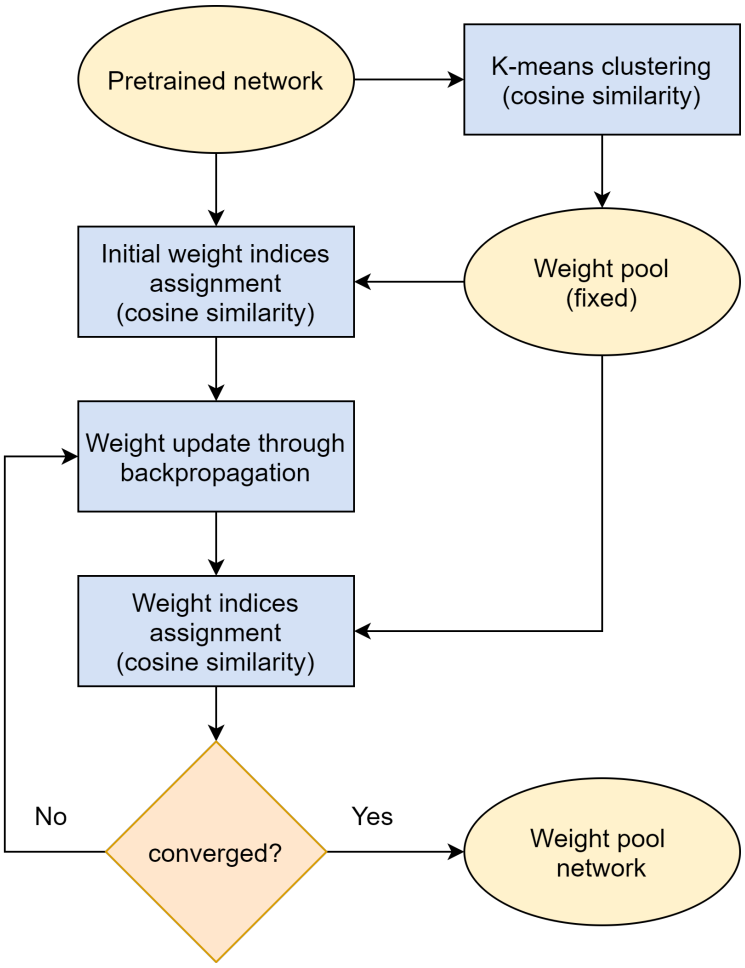


Figure 6.2: Overall flow of generating a weight pool network from a pretrained network.

The backward pass updates the network weights and the forward pass reassigns indices to the nearest weight pool vector. Weight pool network may be further fine-tuned, if needed, for reduced activation bitwidth.

To show the effectiveness of the z-dimension weight pool and determine the optimal pool size, we benchmark the 3×3 kernel weight pool (xy-dimension weight pool) with and without scaling coefficient, as well as the proposed z-dimension weight pool using ResNet-14 (modified ResNet-18 [HZR16] with last block truncated) on the CIFAR-10 dataset. For each setup three weight pool size are tested. The result is shown in Figure 6.4. For all three weight pool sizes, the z-dimension weight pool performs slightly better than the xy-

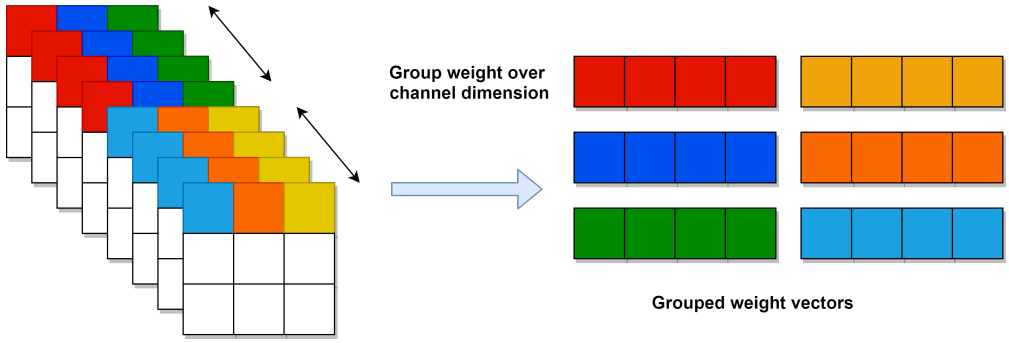


Figure 6.3: Visualization of the z-dimension weight grouping. This example shows a $8 \times 3 \times 3$ filter with weight vector size of 4. The weights are grouped in the channel dimension and same color represent weights in a single group. After the z-dimension grouping, 18 $4 \times 1 \times 1$ weight vectors (6 are shown in the figure) are generated for the given filter.

Group size	4	8	16
Accuracy (%)	91.22	91.13	87.96

Table 6.1: Accuracy of z-dimension weight pool with different group size. The network is ResNet-14 and dataset is CIFAR-10. Original network accuracy is 92.26%.

dimension weight pool with coefficients and significantly better than the xy-dimension weight pool without scaling coefficients. Regarding the pool size, 64 is enough for this network and 32 also achieves a decent result.

The reason for the better accuracy of the z-dimension weight pool is more weights are grouped together in the xy-dimension weight pool than the z-dimension (9 vs 8). Considering a 3x3 convolution layer with weight shape (8,8,3,3), the total number of possible unique weight vectors for 64 weight pool size is 64^{72} for z-dimension and 64^{64} for xy-dimension. Another reason might be if a certain 2D kernel (a channel of an entire filter) has high importance, the z-dimension weight pool can closely reconstruct this kernel by sacrificing other channels, while for xy-dimension it can only be directly chosen from the weight pool.

Table 6.1 shows the accuracy results of different group size (weight vector size) for z-dimension weight pool on ResNet-14. Clearly, a group size of 8 achieves a good balance between compression ratio and network accuracy. We choose 8 as the default group size so

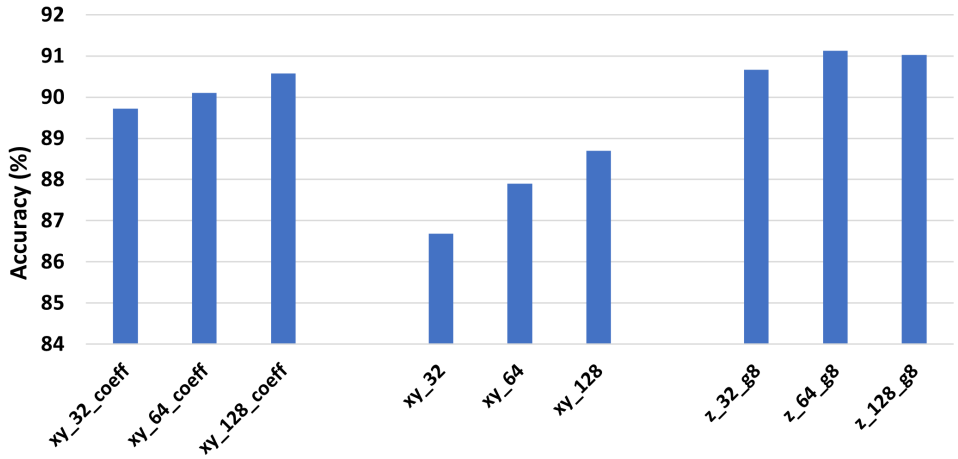


Figure 6.4: Accuracy of weight pool ResNet-14 with different setups, on the CIFAR-10 dataset. For a weight pool with 3×3 kernels, its setups are denoted by xy_n_(coeff), where n means the weight pool size (how many weight vectors in the weight pool) and coeff means the version with scaling coefficients. For the z-dimension weight pool, the setups are denoted by z_n_g8, where n is the weight pool size and g8 means the weight vector size (group size) is 8. The original accuracy is 92.26%.

the weight pool contains multiple 1×8 weight vectors. Compared to clustering 3×3 kernels, clustering along z-dimension has a few advantages:

- It achieves the same or better network performance (accuracy) without the additional scaling coefficient as used in [SNL18], which improves the compression ratio from 4.5× (clustering 3×3 kernels) to 8× over an 8-bit network.
- It is more flexible. It can fit networks with arbitrary kernel sizes including 1×1 kernels, and can apply to fully connected layers as well.

The main rationale behind our choice of using the z-dimension weight pool is not its accuracy but its flexibility. It can work on all filter sizes including 1×1 filters, while the xy-dimension weight pool only works on 3×3 filters. The accuracy for the xy-dimension weight pool is severely impacted for 5x5 filters due to the reduction in representability (>10% accuracy drop on CIFAR-10).

Grouping weights along z-dimension for layers with depth less than 8 (e.g., typical input

layers in image CNNs) incurs underutilization. In most, if not all popular CNNs, such reduced depth layers account for a small fraction of storage and compute. Therefore, we choose to keep such layers (usually just the first layer) uncompressed for better inference accuracy. Not compressing the first layer has minimal impact on compression ratio and runtime for most CNNs since the first layer usually just have three input channels. Another alternative can be grouping all the channels together and zero pad the vector size to 8.

Although the main focus of this work is compressing and accelerating CNNs, we apply the weight pool compression on one dense network to demonstrate the generalization capability of weight pool compression. We evaluate a 3-layer dense network (784-256-128-10) using the FashionMNIST dataset. The original accuracy is 88.65% and after weight pool compression (64 vectors) the accuracy is 88.01% ($< 1\%$ reduction). This is a promising result for adopting the weight pool compression to other types of networks.

6.3.1 Lookup Table Based Bit-serial Computation

As introduced in section 6.2, lookup tables can be used to accelerate convolutions by looking up the vector dot product results directly from memory, instead of computing them. Lookup table offers a trade-off between space complexity and time complexity, and can improve runtime when the memory is large enough and fast enough. However, for dot product operations, the size of lookup table can be huge. Consider the dot product between two 8-element vectors with 8-bit precision, the total number of entries required for the lookup table is $2^{8^{2^8}} = 3.40 \times 10^{38}$. Clearly, such lookup table implementation is not feasible unless the lookup table size can be massively shrunk.

The huge lookup table size is partly caused by both inputs having no restriction on their values, leading to 65536 total input combinations for a simple two-input multiplication. However, this is not the case for weigh-pool networks. Unlike normal neural networks where inputs and weights can be any possible values, weights are fixed for weight pool networks, meaning a single 8-bit multiplication only requires 256 lookup table entries. The lookup table

size for the aforementioned 8-element dot product operation with weight fixed is 1.84×10^{19} entries, which is significantly smaller than 3.40×10^{38}, but still impractical.

To further reduce the lookup table size and support bit-serial multiplication, a key step in our proposed method is *bit decomposition*. For an N-element dot product between input and weight vector (both M bits), the dot product between input (activation) vector and weight vector can be calculated as:

$$\vec{a} \cdot \vec{w} = \sum_{i=0}^{N-1} a_i \times w_i \tag{6.1}$$

Where a_i and w_i are the i-th elements of vectors $\vec{a}$ and $\vec{w}$ respectively and N is the width of the dot product. The input element a_i can be decomposed as:

$$a_i = \sum_{j=0}^{M-1} 2^j \times a_i[j] \tag{6.2}$$

Where $a_i[j]$ is the j-th bit (from LSB) of activation a_i, and M is the bitwidth of the activation. Hence each input element is decomposed into M binary values each representing a single bit, and the input vector is hence decomposed into an $M \times N$ matrix where each row represents a bit position. Each time one row (one bit position) of the input matrix is multiplied with the weight vector by looking up the correct dot product result, and then the result is multiplied with the corresponding bit weight. This step is repeated M times until all the bits are processed and all the results are accumulated to calculate the final result. Doing so, the dot product is effectively calculated in a bit-serial way, and it takes M iterations to compute the original dot product. Figure 6.5 visualizes the decomposition process using the 8-element 8-bit dot product example.

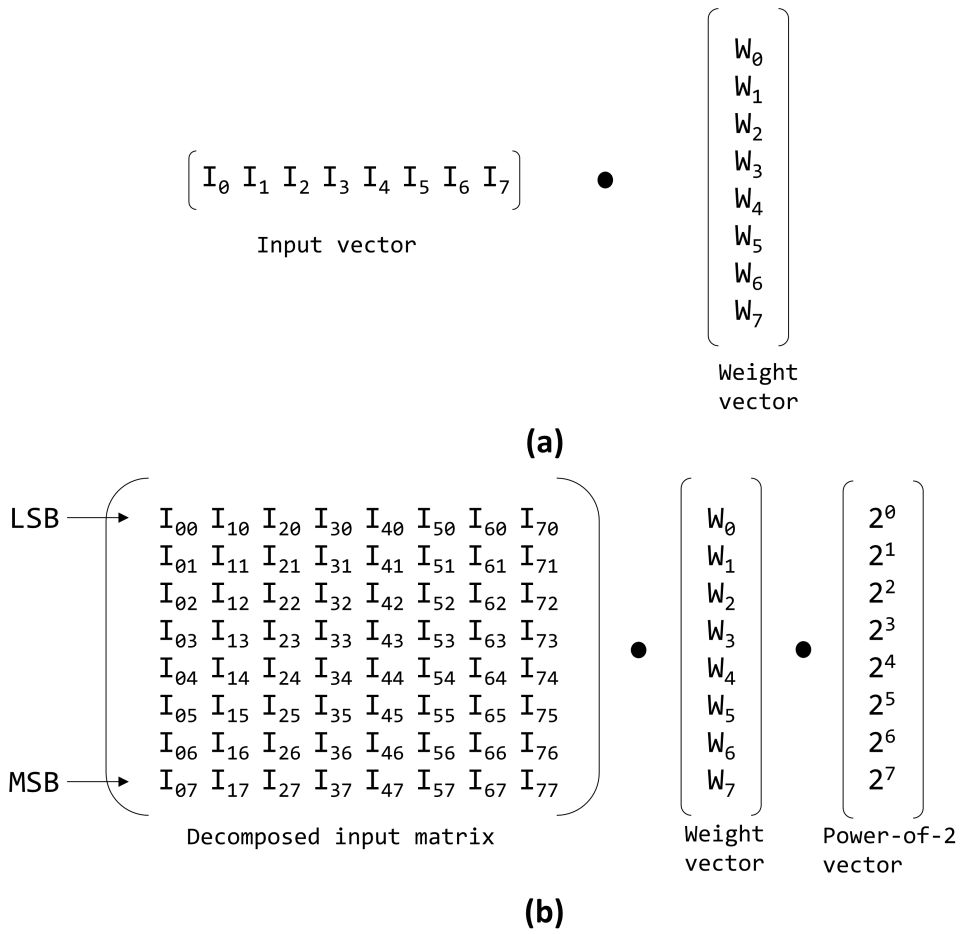


Figure 6.5: Visualization of the bit decomposition step. (a): The original 8-element dot product between input and weight vectors. (b): The original dot product is transformed into matrix-vector multiplication followed by dot product after bit decomposition. I_{mn} means the nth bit (starting from LSB) of the mth element. The original input vector is decomposed into an 8×8 matrix with each element representing a single bit. Each column represents all the bits of an input value while each row represents a unique bit position of all input values. The weight vector is kept the same and is multiplied with all the bit positions of input. The result of the matrix-vector multiplication should be the dot product of input and weight vector at every input bit position. The result is then multiplied with the power-of-two vector which represents bit weights to generate the final dot product result.

6.3.2 Lookup Table Bitwidth and Weight Pool Storage

By decomposing the input vector, the lookup table only needs to store the results of the dot product between N 1-bit input elements and N fixed weight elements. The required lookup table size is thus reduced to 2^N entries, which is 256 entries for the 8-element dot product example. Assuming 64 fixed weight vectors are needed for a weight pool network (we will show later 64 is enough for most cases), and the results are stored in 8-bit precision, the total lookup table storage for the entire network is just 16 kB. Since the lookup table needs to be stored in memory, this storage overhead should be considered when calculating the overall compression ratio of weight pool networks. Besides the activation/weight vector length N, We also denote the lookup table bitwidth by B_l and the size of weight pool by S, the formula for lookup table storage in bits is:

$$Storage_{LUT} = 2^N \times S \times B_l \tag{6.3}$$

For a network with W total weight parameters and weight bitwidth of B_w, the total network storage in bits is $W \times B_w$. Assuming all the weights of the network are compressed by the weight pool method, the maximum compression ratio that can be achieved is:

$$CR = \frac{W \times B_w}{(\frac{W}{N} \times log_2 S + 2^N \times S \times B_l)} \tag{6.4}$$

, where the term $\frac{W}{N} \times log_2 S$ is the weight index storage. $log_2 S$ is the minimum bitwidth required for the weight index, but in actual implementation it may be more efficient to use 8 or 16 bits.

Interestingly, the weight bitwidth of weight pool networks can be arbitrary since the weights are not explicitly stored. The entire weight pool is converted to a lookup table and the dot product results are stored instead of weights. In this case, the lookup table bitwidth matters, as it determines how much memory space is required for storing the lookup table,

as well as the inference accuracy of the network. Storing the lookup table at low bitwidth essentially reduces bitwidth (precision and/or range) of dot-product partial sums and may compromise the inference accuracy. We experimentally show that 8-bit lookup table precision is good enough for most cases. The full results are shown in 6.5.3.

6.3.3 Activation Bitwidth and Weight Pool Network Runtime

In terms of theoretical runtime performance, for the 8-element 8-bit dot product example, the proposed method requires 8 iterations to loop over bit positions and each iteration contains two memory loads (input and result), one shift and one accumulates operation. The weight indices are the same for all the bits and hence can be shared. Normal convolution also requires 8 iterations to loop over individual vector elements and each iteration requires two memory loads (activation and weight), one multiplication and one accumulation. This analysis shows that our proposed method has an almost identical theoretical runtime compared to the 8-bit baseline without considering overheads and optimizations. This is a promising result since the proposed method can have better runtime than the baseline by simply reducing the activation bitwidth below 8 bits. We will show that with various reuse and optimizations, our proposed method has better runtime even at 8-bit activation bitwidth compared to the 8-bit baseline using ARM's CMSIS library [LSC18].

6.4 Weight Pool Implementation: Overheads and Optimizations

There are many runtime overheads associated with software bit-serial processing and weight sharing. Here we discuss these overheads and the corresponding optimizations to overcome them.

6.4.1 Bit Unpacking Overheads and Optimized Dataflow

For software sub-byte precision computation, bit unpacking causes significant runtime overhead since processors typically are byte-addressable. For our bit-serial lookup method, the bit decomposition step needs to unpack each element of the input vector into individual bits, and the same bit position of different input elements (rows of the decomposed input matrix in Figure 6.5) should be grouped together for lookup table computation. Doing this in software requires iterating over all the input elements and for each input element there is an inner loop to extract all the bits. For the 8-element, 8-bit dot product example, 64 iterations are required for a single dot product, while only 8 iterations are required for the actual computation. Implementing bit unpacking for every dot product can significantly slow down the runtime, making it roughly 9× slower than baseline hence negating any potential speedup by reducing the activation bitwidth.

To address the bit unpacking overhead, we utilize input reuse in our dataflow so that the bit unpacking step (activation vector decomposition) can be shared. For CNNs, the same input can be reused for all the filters of a layer, so that the bit unpacking overhead per result lookup is reduced by a factor equal to the number of total filters in a layer. To implement input reuse and share the bit unpacking overhead, we order the loops such that the filter lookup is inside the loops over input channels and filter x, y dimensions. The activation vector decomposition (bit unpacking) is implemented right before the filter loop, so that the decomposed activation matrix can be reused. Algorithm 1 shows the overall flow including the modified loop order. The bit-unpacking step happens at line 7 of Algorithm 1. For a convolution layer with N filters, the time spent on bit unpacking is reduced by a factor of N and is significantly less than the time spent on result lookup for most layers.

Algorithm 1 The simplified algorithm flow of the bit-serial lookup table implementation. Number of input channel group is number of total input channels divided by weight vector size.

```
for loop over batch do
  for loop over output x-dimension do
    for loop over output y-dimension do
      for loop over kernel x-dimension do
        for loop over kernel y-dimension do
          for loop over input channel groups do
            Activation vector decomposition (bit unpacking
            Lookup table caching (flash to ram)
            if Precomputation then
              for loop over weight pool vectors do
                for loop over activation bits do
                  Results lookup
                  Shift and accumulate
                Store results in RAM
              for loop over filters do
                Precomputed results lookup
            else
              for loop over filters do
                for loop over activation bits do
                  Result lookup
                  Shift and accumulate
```

6.4.2 Memory Latency and Lookup Table Caching

In a typical microcontroller, flash memory is used as the main storage and SRAM is used for holding variables during computation. Flash memory has more storage space than SRAM but operates slower. However, due to SRAM's limited size (typically 16-128 kB), it can only be used to hold activations and some temporary variables. The network weights are normally stored in flash memory (size ranges from 128 kB - 2 MB), and during the computation the weights are loaded from the slower flash memory. For weight pool networks, the lookup table size is typically 8-32 kB, which is similar to the SRAM size of some small microcontrollers. For such really tiny, low-cost processors, the lookup table cannot fit in SRAM and need to be stored in flash, hence the result lookup latency will be higher and hurt runtime.

To improve the result lookup latency, we cache the active part of the lookup table in SRAM. Before explaining what is the active part of a lookup table, we first discuss how data can be arranged inside a lookup table. The lookup table of the proposed method contains the dot product results between all weight vectors and all possible input (activation) bit vectors. There are two ways to order the lookup table contents when storing them in memory, one is weight oriented order and the other is input oriented order. Visualization of the two lookup table orders are shown in the appendix. Assume the total number of weight vectors in the weight pool is S and the activation bitwidth is M. For weight oriented order, the lookup table can be split into S smaller concatenated lookup tables, each containing the results of all possible inputs related to a single weight vector. For input oriented order, the lookup table consists of 2^M smaller lookup tables and each of them contains the results of one input with all weight vectors. Input oriented order is more compatible with input reuse dataflow since a few blocks (results corresponding to the bit-vectors generated by the input matrix decomposition) of the lookup table is repeatedly accessed in the filter loop, with other blocks of the lookup table staying idle. We utilize this property and cache the active blocks of the lookup table from flash to SRAM during computation. We use input oriented lookup table in our implementation to reduce the flash access overhead and improve runtime.

In our implementation, the dataflow is configured to boost input reuse, and lookup table accesses can also benefit from this dataflow by caching the lookup table in SRAM. In our input reuse dataflow, after activation decomposition the activation vector is multiplied with corresponding weight vectors for all filters. In this case, only a portion of the lookup table related to the generated activation vectors will be used inside the filter loop. Still considering 8-bit activation bitwidth and weight pool size of 64. After the bit decomposition step, 8 activation bit vectors are generated. For the input oriented lookup table, only 8 blocks of the original lookup table each with 64 entries (weight pool size) that corresponds to the activation bit vectors will be actively used in the filter loop. The total size of the active lookup table is just 512 bytes, which is small enough to fit into most microcontroller SRAMs.

Hence, as shown in line 8 of Algorithm 1, before entering the filter loop, we load the active portion of the lookup table from flash and cache them in SRAM. Figure 6.7 visualizes the lookup table caching process. The overhead of this lookup table caching step is again compensated by sharing it across all the filters. Doing so in the innermost loop of the lookup table results will be loaded from SRAM instead of flash, therefore the overall runtime can be improved.

To validate the analysis, we benchmark the lookup table caching optimization against the implementation without lookup table caching (everything else is the same) on individual layers with a different number of filters. The results are shown in Figure 6.7 (orange bars). The lookup table caching version outperforms baseline for all 4 layer configurations, and the speedup scales with the number of filters in the layer (due to better reuse). While lookup table caching only marginally improves runtime for layers with 32 filters, it achieves more than 1.4× speedup for layers with 192 filters.

6.4.3 Weight Pool Computation Reuse Through Precomputation

The main property of weight pool networks is that a small pool of weight vectors is shared across the entire network. We have shown that using a pool of 32 or 64 8-element weight

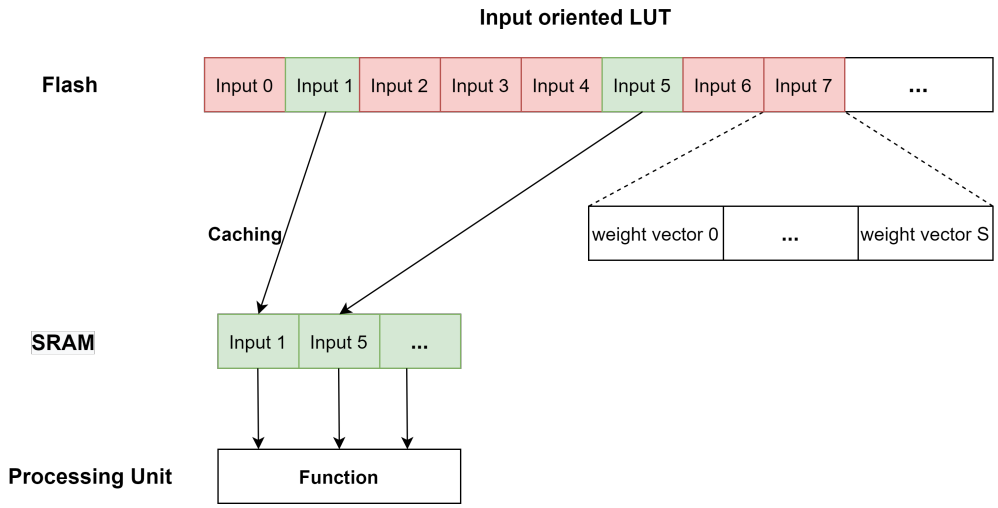


Figure 6.6: Visualization of lookup table caching. Green blocks represent active lookup table regions corresponding to the input vectors that are shared across filters. Red blocks represent the inactive lookup table regions. Active regions are cached into SRAM before the filter loop and the function only accesses lookup table results from SRAM.

vectors is enough for maintaining the accuracy, and such pool sizes are often smaller than the number of filters of a large convolution layer, which can be more than 256. The relatively small pool size offers computation reuse opportunities on large convolution layers to further improve the runtime of weight pool networks.

A property of CNNs is that the same input vector can be reused for all the filters of a convolution layer. For weight pool networks, weights are selected from a group of weight vectors and the total number of distinct weight vectors is the pool size (32 or 64). If a convolution layer has more filters than the pool size, an input vector will inevitably multiply with some weight vectors multiple times when looping over filters. In other words, for a weight pool network, the maximum number of unique dot products that need to be computed for a given input vector is the weight pool size, regardless of the actual number of filters in that layer. To avoid unnecessary computation for large convolution layers, precomputation can be used to only compute the necessary dot products between inputs and weights and store them in another lookup table, hence repeated (bit-serial) computation will be replaced with result lookups. Another way to avoid repeated computation is memoization, where the

dot product results are dynamically memoized during computation (inside the filter loop). We compare and evaluate the two methods (analysis is in appendix) and precomputation performs better. The simplified flow of precomputation is shown in lines 9-16 of Algorithm 1.

Precomputation should only be used for large convolution layers as its benefits rely on a large number of filters (it improves runtime when the number of filters of a layer is larger than the weight pool size). For a given layer, precomputation is used only when the number of filters is *larger* than the pool size. To demonstrate the effectiveness of precomputation, we combine precomputation with lookup table caching and evaluate the speedup against baseline implementation, using the same benchmark in section 6.4.2. The results in figure 6.7 show that for layers that have more filters than the weight pool size, precomputation can further improve the runtime of the lookup table caching version. For a layer with 192 filters, precomputation + lookup table caching achieves 2.45× speedup against baseline implementation and is 1.7× faster than just using lookup table caching. However, for layers with number of filters that are smaller or equal to the weight pool size, precomputation hurts runtime. This result supports our analysis that precomputation should not be used for those layers.

Run-time accuracy trade-off Precomputation not only accelerates wide convolution layers, it also offers another way to make trade-offs between runtime and accuracy, besides adjusting the activation precision. For a relatively wide network that contains layers wider than 32 filters, the runtime can be improved by reducing the weight pool size. Although we observed that a weight pool size of 64 works reasonably well in most cases and we set 64 as

Name	Model	SRAM (kB)	Flash (kB)	Core	Freq. (MHz)
MC-large	F207ZG	128	1024	CM3	120
MC-small	F103RB	20	128	CM3	72

Table 6.2: STM Nucleo family microcontrollers used for benchmarking. Both use ARM Cortex M3 for the core.

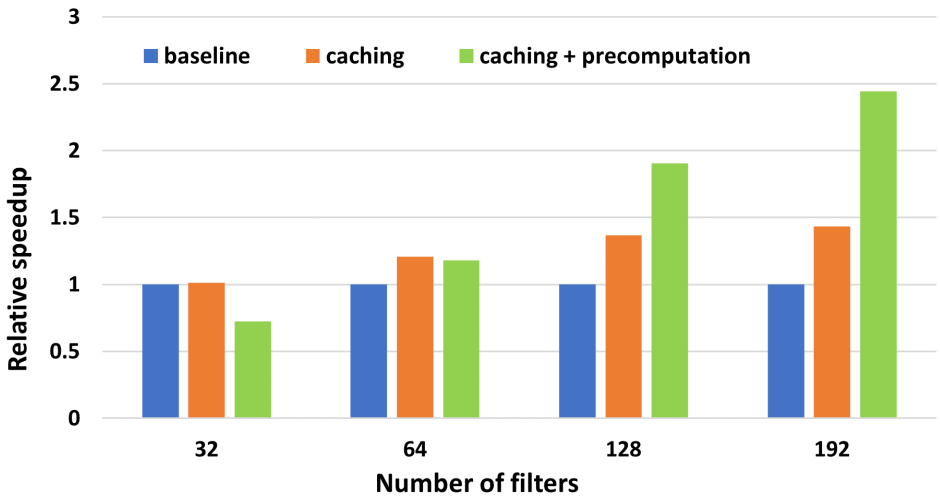


Figure 6.7: Relative speedup of just lookup table caching (orange) and precomputation + lookup table caching (green) against baseline implementation. Four 3×3 convolution layers with different number of filters are tested. The number of channel is set to be same as number of filters and the input size is 16×16. Weight pool size is 64.

the default size, 32 is also good enough for many cases. The runtime can be improved with a tiny drop in accuracy by reducing the weight pool size in such cases.

6.5 Evaluation

6.5.1 Experimental Setup

We evaluate the accuracy and runtime of the z-dimension weight pool method on five different networks: TinyConv [LSC18], MobileNet-v2 [SHZ18], ResNet-10 (ResNet-18 with last two blocks truncated), ResNet-14 (ResNet-18 with last block truncated) and ResNet-s (scaled-down version of ResNet-18 used in [BRT21]). We use 2 datasets, CIFAR-10 and Quickdraw-100 (100 classes), and form 5 network-dataset combinations. All ResNets are tested on CIFAR-10 while MobileNet-v2 and TinyConv are tested on Quickdraw-100. The network structures are adjusted slightly to fit CIFAR-10 and Quickdraw-100. For the weight pool version of MobileNet-v2, only the 1×1 point-wise convolution layers are compressed using

the weight pool. Depth-wise convolution layers are kept uncompressed since they do not fit our proposed implementation. Theoretically the depth-wise layers can be compressed using the xy-dimension weight pool, but it is not necessary - those layers account for a very small portion of storage (2.93%) and runtime.

All the accuracy results are evaluated using the PyTorch framework. For network training and retraining, SGD is used as the optimizer with learning rate scheduling, and batch size set to 128. For runtime results, we use two microcontrollers as shown in Table 6.2. We use ARM Compiler version 6 and runtime is measured using the built-in cycle counter. The frequency is set to maximum frequency for both boards.

6.5.2 Compression Ratio

Network	Total param	CR	LUT overhead
TinyConv	81600	2.32	29.8%
ResNet-s	170928	4.43	29.7%
ResNet-10	665280	6.51	13.8%
ResNet-14	2729664	7.55	4.3%
MobileNet-v2	2249792	6.22	4.5%

Table 6.3: Total number of parameters (uncompressed), overall compression ratio (CR) and lookup table overhead of the selected networks. The lookup table overhead is the proportion of lookup table storage to the total network storage after compression.

Table 6.3 shows the total number of parameters and the overall compression ratio of the networks with weight pool size of 64. The lookup table overhead is also shown and is compression limiting only for small networks such as TinyConv. The compression ratio improves as the network size increases, and is close to the theoretical maximum (8×) for ResNet-14 (and even larger networks). Smaller networks further suffer in compression since the first convolution layer and fully connected layers are not compressed, whose effect is not well amortized. [1]

[1]Compressing fully connected layer with weight pools improves the compression ratio for Resnet-s (TinyConv) to 4.5(3.1) but at the cost of 0.7%(2.8%) additional accuracy drop. These compression ratios improve further to 5.7 (4.2) if weight pool size of 32 is used albeit, again at 0.5%-1% additional accuracy drop. In this

6.5.3 Accuracy Evaluation

6.5.3.1 Weight Pool Size

We first study the impact of weight pool size alone on accuracy without any quantization effects. Table 6.4 shows the accuracy of the z-dimension weight pool compression with three weight pool sizes without any activation quantization compared to an uncompressed floating-point baseline. A weight pool size of 64 ensures little accuracy drop for most networks and is our default for all experiments unless otherwise mentioned. ResNet-s, being already compressed, is tougher to compress without accuracy loss. The results demonstrate the effectiveness of the z-dimension weight pool compression, even for already small CNNs like TinyConv and ResNet-s.

Network	Original	32	64	128
			CIFAR-10	
ResNet-s	85.3	82.0	83.0	84.0
ResNet-10	91.0	89.3	89.8	90.1
ResNet-14	92.3	90.7	91.1	91.0
			Quickdraw-100	
TinyConv	82.2	81.7	82.2	82.3
MobileNet-v2	86.5	86.7	86.8	86.9

Table 6.4: Accuracy (%) of the z-dimension weight pool with different weight pool sizes on selected network-dataset combinations. Original means original network accuracy and 32/64/128 are the weight pool size.

6.5.3.2 Lookup Table Bitwidth

For the proposed bit-serial lookup table implementation, the dot product results between decomposed activation bit-vectors and weight vectors are stored in the lookup table, and the bitwidth of the lookup table may affect inference accuracy.

To evaluate the impact of lookup table bitwidth on network accuracy, we simulate the

work we do not compress them as they do not improve compression for most networks but affect accuracy.

proposed bit-serial lookup implementation using PyTorch. Results in table 6.5 show that a *lookup table bitwidth of 8* loses no accuracy and is the default for our experiments unless otherwise mentioned. Furthermore, since most processors are byte-addressable, using a bitwidth smaller than 8 would incur performance overheads albeit delivering a better storage compression for small networks.

	Lookup table bitwidth			
Network	No-LUT	16	8	4
CIFAR-10				
ResNet-s	83.0	83.0	82.9	82.3
ResNet-10	89.6	89.9	89.9	89.4
ResNet-14	91.1	91.1	91.1	90.4
Quickdraw-100				
TinyConv	82.2	82.2	82.1	81.6
MobileNet-v2	86.8	86.6	86.6	85.5

Table 6.5: Inference accuracy (%) of bit-serial lookup table implementation. No-LUT column shows accuracy that not using lookup table implementation. The activation bitwidth is 8 bit.

6.5.3.3 Activation Bitwidth

Although activation bitwidth does not affect the storage of a weight pool network, it affects the runtime when the weight pool network is implemented using the proposed bit-serial lookup table approach. We use an iterative search algorithm to determine the optimal range when quantizing activations. The weight pool size is 64 and the lookup table bitwidth is 8 for all cases. Table 6.6 shows that for 8-bit activation bitwidth, almost all networks achieve floating point accuracy (i.e.,"64" column in Table 6.4). At 5-bit activation bitwidth, most networks still maintain less than 1% accuracy drop except for MobileNet-v2 which is quantization-unfriendly [SFZ18, YW21]. Moreover, for lower bitwidths, the accuracy drop can be compensated by retraining the network with activation quantization. After retraining, activation bitwidth can go down to 3-4 bit within 1% accuracy drop for all networks except for MobileNet-v2, which requires 5 bits.

Network	Activation bitwidth 8	7	6	5	4	3	Min. bitwidth < 1% a.d
CIFAR-10							
ResNet-s	82.9	83.0	83.1	82.9	82.5	80.4(80.4)	4
ResNet-10	89.9	89.9	89.8	89.6	88.9(89.2)	84.5(87.8)	4
ResNet-14	91.1	91.1	91.0	90.8	90.6(91.0)	88.5(90.2)	3
Quickdraw-100							
TinyConv	82.1	81.8	81.2	79.3(82.0)	69.2(81.2)	36.0(77.4)	4
MobileNet-v2	86.6	86.5	86.0	83.6(85.9)	77.9(84.0)	36.4(73.0)	5

Table 6.6: Inference accuracy (%) of weight pool networks with different activation bitwidths. Results in brackets are accuracy after retraining. The last column shows the minimum activation bitwidth with less than 1% accuracy drop. The lookup table bitwidth is set to 8 bit.

6.5.4 Runtime Evaluation

6.5.4.1 Impact of Activation Bitwidth

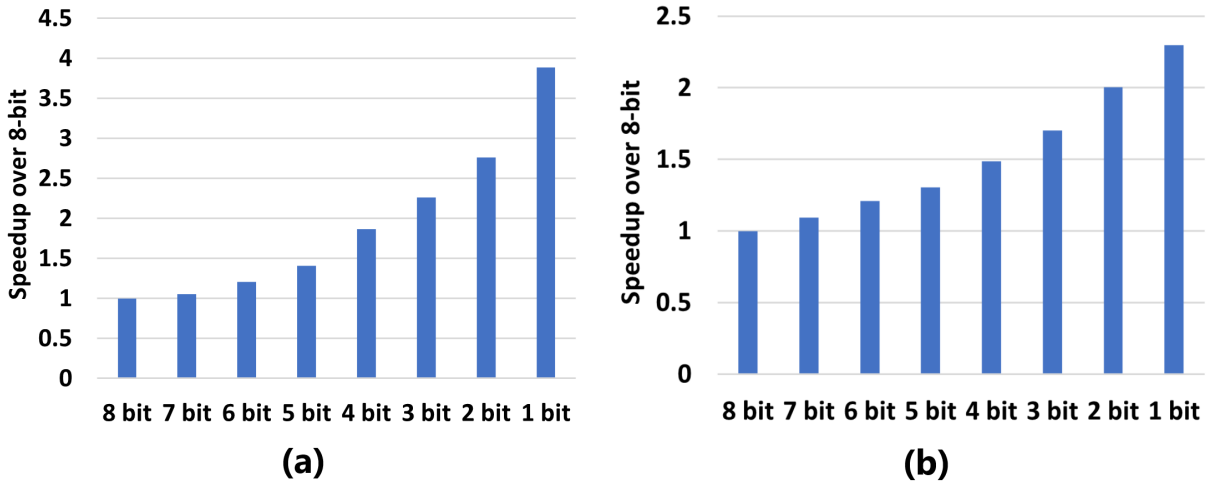


Figure 6.8: Speedup against 8-bit bit-serial lookup implementation for different activation bitwidths. (a): results without precomputation. (b): results with precomputation. The input size is 16×16 and number of channels and filters are both 128. Weight pool size is 64.

One of the main contributions of the proposed framework is the support of accelerating runtime by reducing activation bitwidth. We evaluate the runtime improvement from an 8-bit baseline on a layer with 128 channels and filters and pool size of 64 in Figure 6.8, using MC-large. Without precomputation, the speedup scales linearly according to activation

bitwidth, and is almost 4× for 1-bit activation (less than the 8× theoretical speedup because of the fixed bit unpacking overhead). For the precomputation case, as the activation bitwidth reduces, the runtime of the bit-serial loop during precomputation reduces, but the runtime for precomputed results lookup does not change and starts to dominate the runtime. However, precomputation already accelerates the runtime significantly so the overall speedup is still better for large layers.

6.5.4.2 Full-network Benchmark

To evaluate the overall runtime performance of the proposed method, we evaluate the full-network runtime performance on both microcontrollers with weight pool sizes of 32 and 64, and compare with ARM CMSIS implementation whenever possible. Only convolution layers are benchmarked since we do not apply weight pool on the fully connected layers. The results are shown in Table 6.7. For the minimum activation bitwidth case, the results for the 32-vector weight pool are for reference only, since the minimum bitwidth is determined from the results of the 64-vector weight pool.

Network	CM.	64-8	32-8	64-m	32-m
			MC-large		
TinyConv	1.06	0.83	0.75	0.60	0.57
ResNet-s	0.60	0.49	0.43	0.31	0.28
ResNet-10	5.28	3.00	2.22	1.87	1.61
ResNet-14	/	3.46	2.59	1.92	1.73
MobileNet-v2	/	3.60	3.12	3.07	2.78
			MC-small		
TinyConv	1.95	1.49	1.33	0.99	0.89
ResNet-s	1.24	1.07	0.89	0.63	0.55

Table 6.7: Full-network inference latency (in seconds) with different setups for both microcontrollers. CM. stands for CMSIS implementation, -8 means 8-bit activation precision while -m means minimum activation precision that has less than 1% accuracy drop that determined in Table 6.6. 32 and 64 are the weight pool size. / means the network cannot fit into flash memory.

For all setups, the proposed implementation achieves better runtime than CMSIS and the speedup is better for larger networks. With less than 1% accuracy drop, the "right bitwidth"

weight pools can achieve over 2.8× speedup over CMSIS for medium-sized CNNs like ResNet-10 and around 2× speedup for smaller CNNs like ResNet-s and TinyConv. There are several factors that make the speedup smaller for small CNNs, including lack of precomputation opportunity, more bit unpacking overhead and the relatively larger impact of not accelerating the first layer. Larger CNNs (ResNet-14, MobileNet-v2) do not fit into the microcontroller memory without the weight pool compression and hence a runtime comparison is not possible. Overall, the proposed method improves CMSIS runtime on CNNs regardless of network structure and activation bitwidth, and the speedup is larger for large networks.

6.5.5 Comparison with Binarized Networks

The theoretical compression ratio of a weight pool network is similar to the compression ratio of binarized networks but with much better accuracy. [RLG20] evaluates the implementation of binarized networks on microcontrollers and reports $2-4\times$ speedup compared CMSIS 8-bit implementations. For comparison, we trained the binarized version of TinyConv and the accuracy for CIFAR-10 is barely 66.9% as opposed to 81.2% with weight pools. Our method achieves 14.3% higher accuracy with just 1.24 × runtime overhead.

6.6 Related Work

6.6.1 Neural Network Weight Sharing

The concept of weight sharing in neural networks can be dated back to 1992 [NH92], as an approach to simplify neural networks. Recently, weight sharing has been applied to convolution neural networks, by clustering and sharing 2D convolution kernels [SNL18, WWW18] for all layers of the network. With tiny or almost no drop in accuracy, weight sharing can significantly compress the parameters of the neural network, which leads to 4.5× to 36× reduction in CNN's storage requirement, depending on the exact sharing method and baseline

precision.

6.6.2 Lookup Table Based Vector Multiplication Acceleration

Lookup table is a widely used method to improve runtime by replacing computation with memory lookup. There are many works [DZZ19, SCB20, FFG21] try to accelerate deep neural networks with lookup tables by memorizing vector multiplication results. However, due to the huge lookup table size (GB+) required for memorizing all possible results of a vector-vector multiplication, all of them are DRAM based in-memory accelerators, hence they are not software solutions.

6.6.3 Software Based Convolution Acceleration for Sub-byte Precision

There are a few software-focused works that develop algorithms to deploy sub-byte neural networks on CPUs. [YLD19] utilizes a single multiplication instruction to implement multiple sub-byte multiplications through bit-packing, and is able to show performance improvement for four-bit input and ternary weight network over 16-bit baselines. [CMC18] and [CMC20] share the same main concept and propose a software method and corresponding optimizations for CPUs to compute sub-byte precision more efficiently by utilizing the popcount instruction. However, as their method has a time complexity proportional to the total number of weight bits times the total number of activation bits, moderate speedup over 8-bit baseline can only be demonstrated on very low activation and weight bitwidth (2-3 bits). [UJ17] is another work that targeting extremely low precision CNN acceleration, with a similar idea that utilizes the popcount instruction. Current software methods for accelerating sub-byte neural networks have limited use cases due to their strict requirements on activation and weight bitwidth. For many applications, quantizing both activation and weight to 2-3 bits can severely impact the learning capability of neural networks. Besides, some versions require advanced instructions that are not available for low-power microcon-

trollers like ARM Cortex M0 and M3. We do not directly compare against these works as the target applications and platforms are not the same and they do not offer arbitrary sub-byte precision acceleration.

6.7 Conclusion

We have proposed the first framework for efficiently deploying weight pool networks on resource-constrained processors, with compression, training and execution methodologies. The proposed weight pool networks with bit-serial lookup table implementation support and accelerate arbitrary sub-byte precision execution, and can achieve up to 2.8× speedup and up to 7.5× compression compared to 8-bit networks, with less than 1% drop in accuracy. The proposed framework is more efficient on large networks, both in terms of compression and speedup, therefore is suitable for deploying large neural networks on small microcontrollers. We are able to fit and accelerate relatively large CNNs like MobileNet-v2 on a microcontroller with 1MB Flash memory, which otherwise will not fit in the processor memory.

CHAPTER 7

Characterizing the Effect of Partial Sum Precision on Accuracy and Energy for Analog Neural Network Accelerators

There is a massive interest in analog neural network accelerators like processing in memory and on-chip photonics recently, mainly because of their superior power efficiency. However, ADCs tend to be the power bottleneck of analog neural network accelerators, which makes high-precision ADCs infeasible in terms of power efficiency. Depending on the accelerator configuration and application, ADCs usually quantize the partial sum of neural networks, which usually requires relatively high precision. The impact of partial sum quantization is important for designing accurate and efficient analog neural network accelerators, but it is not thoroughly studied. In this paper, we develop an accurate analytical model of the quantization error for analog neural network accelerators that takes partial sum quantization into account, which provides an efficient and accurate way of exploring the accuracy-performance trade-offs.

7.1 Introduction

The impact of the bitwidth of the weights and activations of neural networks has been thoroughly studied in the past decade [Guo18]. Many quantization schemes have been proposed to reduce the bitwidth of the activations and weights, hence improving the overall efficiency of executing neural networks. However, in contrast to the massive number of

works on various activation and weight quantization methods, there are few works that study the impact of the partial sum (accumulation) bitwidth. The partial sum bitwidth is the bitwidth used to accumulate the individual multiplication results when computing a dot product between two vectors, which is the most commonly used operation for executing neural networks. Most quantization works assume floating-point precision for partial sum accumulation, which is reasonable for Central Processing Units (CPU) and early Graphics Processing Units (GPU), but not for analog neural network accelerators including processing in memory (PIM) accelerators [SNM16, CLX16, LKL21, AHR18] and photonic accelerators [BMM19, SKB21, LLY19, ZLY20] or possibly even for deeply resource-constrained digital accelerators.

Partial sum precision plays an important role in various types of analog accelerators. Analog accelerators typically contain multiple arrays and the accumulation within a single array happens before the Analog-to-Digital Converter (ADC) sensing and can be considered as full precision. For most analog accelerators, ADCs dominate the system power, and the power of an ADC is directly related to its bitwidth. The array size determines how many operations can be computed before the expensive conversion step. Therefore, how the partial sum bitwidth and array size affect the overall energy efficiency and accuracy should be thoroughly studied and understood. The main contributions of this paper can be summarized as follows:

- To the best of our knowledge, this is the first work to propose an analytical model of overall quantization error that models partial sum quantization with both rounding and clipping error.
- We provide a case study of an example analog neural network accelerator to demonstrate how the proposed model can help to optimize the architecture design.

7.2 A quick 10.1145/3007787.3001140r on PIM-style analog neural network accelerators

Though, most of the modeling developed in this work applies to arbitrary accelerators which benefit from computing partial sum with reduced bitwidth, as an exemplar we focus on processing in memory.

PIMs leverage the memory array for the computation of vector-matrix multiplications, and have been widely used to accelerate neural networks for the past 5 years. The advantage of PIMs is power efficiency, as the memory cells can have significantly lower power than digital multiply-and-accumulate (MAC) units. PIMs usually consist of multiple 2D arrays of memory cells. Each memory cell computes a single multiplication between the applied voltage and the encoded resistivity values. The multiplication results are represented as current and the results of each column are accumulated using the current law and accumulated with an ADC. In this paper, we refer to the column size as the hardware dot product size.

ADC applies quantization to the results according to their bitwidth and range, which can affect the overall accuracy. Moreover, ADCs are usually the power bottleneck and can consume more than 60% of total power[SNM16, KLL21]. This typically prevents the use of high-precision ADCs which consume a lot of power in PIMs. There is a work targets the ADC quantization problem of PIMs [WTL21], but is simulation-based and lacks an analytical model.

For most analog neural network accelerators, only positive inputs and weights are supported in hardware. However, neural networks require negative weights and outputs to be properly trained and generate meaningful predictions. A common workaround is to split a normal dot product (with both positive and negative weights) into two positive-only dot products, with one representing the positive part and one representing the negative part. The two dot products can be computed in analog accelerators, and their results are subtracted digitally after passing through ADCs. This split-compute-subtract method can be

either done in parallel or in serial, depending on the exact design. We assume the underlying hardware utilizes this method to handle negative weights in our analytical model.

7.3 Quantization error modeling

For a typical analog neural network accelerator, the partial sum precision is determined by the ADC bitwidth. Unless the hardware dot product size (column size) is large enough to compute the output of a convolution layer directly, partial sums of multiplication results will be computed instead, and requires digital accumulation to obtain the output activation. In such cases, the partial sum will be quantized by the ADC, whose range and bitwidth can significantly impact the overall quantization error and energy efficiency.

There are two ways to set the range for ADCs, (1) Saturation will never happen for all multiplication result combinations, and (2) saturation is allowed to happen. For (1), since both input activations and weights are represented with fixed-point representation (converted by digital-to-analog converters (DAC)), their multiplication results will have a fixed range. Therefore ADC can be configured to avoid any saturation. There will be no clipping error for this case, but the rounding error can be large for small values. For (2), since saturation is allowed, clipping error needs to be considered for large values, while the rounding error should be smaller than (1) for the same ADC bitwidth because of the smaller range.

In this section, we will analyze the quantization error and provide analytical models for the two cases separately. The model targets analog neural network accelerators that implement pseudo-negative accumulation (digital subtraction of positive-only results). We make several reasonable assumptions and approximations to obtain clear, closed-form analytical models. Table 7.1 lists common symbols used in the analysis throughout this paper and their meanings.

Table 7.1: Common symbols used in the analysis and their meanings.

Symbol	Meaning
b	ADC bitwidth
t	Magnitude of a single quantization interval
q_{adc}	Maximum quantization range
h	Hardware dot product size
s	Full dot product size

7.3.1 Case 1: saturation not allowed

In this case, only rounding errors need to be modeled, and the overall quantization error for a full dot product is the sum of the rounding error for each hardware dot product (ADC quantization). We make the following assumption:

Assumption 1 *Rounding error can be modeled by a uniform distribution $X \sim U[-\frac{t}{2}, \frac{t}{2}]$ for the entire range of possible hardware dot product results.*

This assumption implies that quantization error is independent of the input distribution, which is a reasonable approximation for most cases where the b is not too small (e.g., 1-3 bits). The distribution of overall quantization error, which is the sum of rounding errors of individual hardware dot products, can be calculated by summing all Uniform distributions. Note that the sum of n independent uniform distributions with range [0,1] forms the Irwin-Hall distribution, whose PDF is well-defined [JKB95]:

$$f(x;n) = \frac{1}{(n-1)!} \sum_{k=0}^{\lfloor x \rfloor} (-1)^k \begin{pmatrix} n \\ k \end{pmatrix} (x-k)^{n-1} \tag{7.1}$$

Since $\sum_{k=1}^{n} U[-\frac{t}{2}, \frac{t}{2}] = \sum_{k=1}^{n} (-\frac{t}{2} + t\cdot \text{ U}[0,1])$, with a linear transformation, the PDF for $\sum_{k=1}^{n} U[-\frac{t}{2}, \frac{t}{2}]$ can be derived as:

$$g(x;t,n) = \frac{1}{t} f\left(\frac{\frac{2x}{t}+n}{2}; n\right) \tag{7.2}$$

, where f is the PDF for the original Irwin-Hall distribution.

Both t and n in Equation 7.2 are related to h and s. n is the number of hardware dot products required to compute the full dot product of one sign, which equals $\lceil \frac{s}{h} \rceil$. To implement the pseudo-negative representation, the dot product result needs to be subtracted by another dot product with the same distribution. Since $g(x)$ is symmetrical around zero, $g_+(x) - g_-(x) = g_+(x) + g_-(x)$. Therefore the full dot product requires the accumulation of $2\lceil \frac{s}{h} \rceil$ hardware dot product results.

Let a be the range of a single multiplication result ($max - min$), and t_h be the magnitude of a single quantization interval with hardware dot product size of h. Then the range of hardware dot product results is $h \times a$, and $t_h = h \times t_1$, where $t_1 = \frac{a}{2^b}$ Therefore, the Equation 7.2 can be rewritten to

$$g(x; a, h, s, b) = \frac{2^b}{h \times a} f\left(\frac{\frac{2^{b+1}x}{h \times a} + \lceil \frac{s}{h} \rceil}{2}; 2\left\lceil \frac{s}{h} \right\rceil\right) \tag{7.3}$$

, which is the PDF of the overall quantization error in terms of a, h, s, b. Equation 7.3 can be used to compute the relative quantization error for a given scale, hardware dot product size, full dot produce size, and ADC bitwidth. However, the mean of the overall quantization error is 0, which means the mean absolute value should be used to compare different configurations. Since the distribution is symmetric with respect to 0, the mean absolute quantization error can be computed by:

$$\mathbb{E}(|X|) = \int_{-\infty}^{\infty} |x| g(x; a, h, s, b) dx \tag{7.4}$$

Impact of hardware dot product size: By fixing the dot product size, ADC bitwidth, and the scale o f multiplication output (s, b, a), the relationship between overall quantization error and hardware dot product size can be obtained.

Based on Equation 7.3, 7.4, and the results from 7.4.1, we make the following observations: When the ADC range is configured such that no saturation can happen, having a larger hardware dot product size leads to a larger overall quantization error.

This observation is in contrast to one common intuition that having a large array size can reduce the overall quantization error since there are fewer ADC quantization operations for a given dot product computation. The difference between our observation and the common intuition is that the intuition does not consider the fact that the ADC range needs to be scaled according to hardware dot product size to avoid saturation.

7.3.2 Case 2: saturation allowed

In this case, the ADC range does not need to be configured to cover the entire range of possible hardware dot product results. Doing so is essentially trading range for precision, to represent small values more precisely. As a consequence, clipping error will be introduced since any value larger than the ADC range will be clipped at the upper limit of the ADC range. Unlike rounding error, clipping error depends on the input distribution and should be taken into account when modeling the overall quantization error. Due to the nature of clipping error which is hard to model precisely, we make several approximations when modeling the quantization error of the case where saturation is allowed. Although the derived analytical model is not precise, we will show that it is a good approximation for most cases, especially with our optimizations.

For most analog neural network accelerators, only positive input and weights are supported in the analog domain, hence the multiplication results are all positive. To model the quantization error, we first make the following assumption:

Assumption 2 *The multiplication result between a pair of input activation and weight for analog neural network accelerators follows a truncated normal distribution with mean μ and standard deviation σ. The negative portion of the original distribution is truncated.*

Figure 7.1 shows the histogram of multiplication outputs generated by a batch of CIFAR-10 dataset with random weights. It can be seen that the distribution follows a truncated normal distribution. The hardware dot product results are hence the sum of h truncated

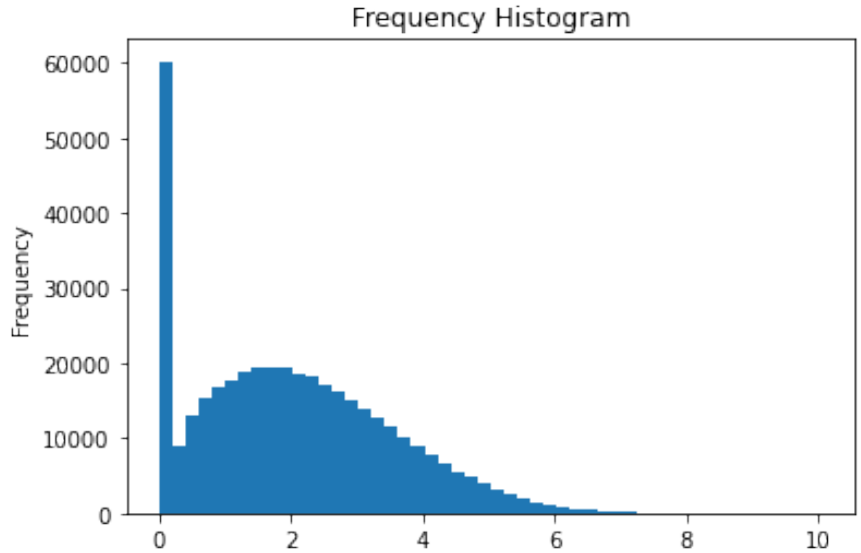


Figure 7.1: Histogram of the distribution of a batch of multiplication outputs.

normal distributions, and the distribution of the sum needs to be modeled to compute the clipping error. For relatively large hardware dot product sizes, the central limit theorem (CLT) can be used to model the distribution of the hardware dot product results. CLT for sums states that when n independent random variables are summed up, the sum tends toward a normal distribution, even if the distribution of the random variables is not normal. A rule of thumb for CLT to be valid is $n > 30$, which is quite common for PIMs ($h > 30$). According to CLT, the mean and variance of the hardware dot product results can be calculated as:

$$\mathbb{E}(\sum X) = h\mu \quad \text{and} \quad Var(\sum X) = h\sigma^2 \tag{7.5}$$

However, we observe that when X is a truncated normal distribution, CLT is still a good approximation for $\sum X$ even if $n < 30$. Therefore, we make the following approximation to make a generic model for all hardware dot product sizes:

Approximation 1 *Let Y be the sum of n independent random variables X which follows a truncated normal distribution (multiplication results). CLT is used to model the distribution of Y for $n > 1$.*

Thus, for $h > 1$, $Y \sim \mathcal{N}(h\mu, h\sigma^2)$.

Note that multiplication results do not have to follow truncated distributions to make this analysis valid, as the central limit theorem is valid for all distributions. In other words, the analysis holds true as long as the multiplication results are positive numbers.

When saturation is allowed, the overall quantization error is a combination of rounding error and clipping error, which will be modeled separately in this analysis. Let the range of ADC be $[0, q_{adc}]$ The clipping error of a single hardware dot product result is defined as:

$$Z_{ch} = \begin{cases} Y - q_{adc} & Y - q_{adc} > 0 \\ 0 & Y - q_{adc} \leq 0 \end{cases} \tag{7.6}$$

, where y is the hardware dot product result. It can be seen from Equation 7.6 that the PDF of the clipping error is not continuous and does not follow a well-defined distribution. To compute the distribution of the sum of clipping errors, we make the following approximation:

Approximation 2 *Use the central limit theorem to approximate the distribution of the sum of n clipping errors, for $n > 1$.*

In Section 7.3.3 we will show that this approximation can be optimized for better estimation, but here we first conduct the analysis using unmodified CLT approximation. In order to apply CLT, sample mean and variance are required, which can be obtained from the PDF of Y which is normally distributed. The mean clipping error can be calculated by:

$$\mathbb{E}(Z_{ch}) = \int_{-\infty}^{\infty} z_{ch} f(z_{ch}) dz_{ch} \tag{7.7}$$

Let $f(y)$ be the PDF of Y, and substitute Z_{ch} with Y based on Equation 7.6, the sample mean can be expressed as:

$$\mathbb{E}(Z_{ch}) = \int_{q_{adc}}^{\infty} (y - q_{adc}) f(y) dy \tag{7.8}$$

Similarly, the sample variance of clipping error can be obtained using the PDF of Z_{ch}.

$$Var(Z_{ch}) = \int_{-\infty}^{\infty} (z_{ch} - \mu)^2 f(z_{ch}) dz_{ch} \tag{7.9}$$

, where μ is the sample mean, which can be obtained using Equation 7.8. The integral can be broken up and rewritten using the PDF of Y based on Equation 7.6:

$$\begin{aligned} Var(Z_{ch}) = &\int_{-\infty}^{q_{adc}} \mathbb{E}(Z_{ch})^2 f(y) dy \\ &+ \int_{q_{adc}}^{\infty} (y - q_{adc} - \mathbb{E}(Z_{ch}))^2 f(y) dy \end{aligned} \tag{7.10}$$

Then the distribution of Z_{cf}, which is the overall clipping error for the full dot product of one sign, can be approximated to a normal distribution:

$$Z_{cf} \sim \mathcal{N}\left(\left\lceil \frac{s}{h} \right\rceil \mathbb{E}(Z_{ch}), \left\lceil \frac{s}{h} \right\rceil Var(Z_{ch})\right) \tag{7.11}$$

For the sum of rounding error, similar to the sum of clipping error, it needs to be modeled as a normal distribution using CLT, so that later it can be combined with clipping error. The rounding error for a single hardware dot product is uniformly distributed according to Assumption 1 for $Y \leq q_{adc}$, and the rounding error is zero for $Y > q_{adc}$. Since the mean rounding error is zero for both parts, the sample mean of the rounding error $\mathbb{E}(Z_{rh})$ is zero. The sample variance can be obtained from the variance of uniform distribution with adjustment to take account into cases where $Y > q_{adc}$. The variance of $U[-\frac{t}{2}, \frac{t}{2}]$ is $\frac{t^2}{12}$ ($t = \frac{q_{adc}}{2^b}$), and the variance is zero for $Y > q_{adc}$ since the sample mean is zero. The sample variance can hence be calculated by:

$$\begin{aligned} Var(Z_{rh}) &= Var(Z_{rh} | Y \leq q_{adc}) \times P(Y \leq q_{adc}) + 0 \\ &= \frac{q_{adc}^2}{12 \times 2^b} \int_0^{q_{adc}} f(y) dy \end{aligned} \tag{7.12}$$

The distribution of the overall rounding error of the full dot product of one sign, can be approximated to a normal distribution:

$$Z_{rf} \sim \mathcal{N}\left(0, \left\lceil \frac{s}{h} \right\rceil Var(Z_{rh})\right) \tag{7.13}$$

To compute the signed full dot product result, the full dot product results of the positive and negative parts need to be subtracted. Therefore, the overall quantization error can be represented as:

$$Z_f = (Z_{rf+} + Z_{cf+}) - (Z_{rf-} + Z_{cf-}) \tag{7.14}$$

According to Equation 7.11 and Equation 7.13, all four errors are normally distributed. The subtraction can be modeled as adding normal distributions with $\mu \leq 0$. Since the sum of normal distributions is also a normal distribution, the distribution of Z_f can be modeled as a normal distribution with mean $\mathbb{E}(Z_f) = \mathbb{E}(Z_{cf+}) - \mathbb{E}(Z_{cf-})$, and variance $Var(Z_f) = Var(Z_{rf+}) + Var(Z_{rf-}) + Var(Z_{cf+}) + Var(Z_{cf-})$. Since the positive and negative parts are essentially the same distribution, the distribution of Z_f is

$$Z_f \sim \mathcal{N}(0, 2Var(Z_{rf}) + 2Var(Z_{cf})) \tag{7.15}$$

The mean absolute value of the overall quantization error can be computed as:

$$\mathbb{E}(|Z_f|) = \int_{-\infty}^{\infty} |z_f| f(z_f) dz_f \tag{7.16}$$

, where $F(z_f)$ is the PDF of $Z(f)$ which follows the normal distribution defined in Equation 7.11. With Equation 7.15, the mean absolute value of the overall quantization error can be represented in terms of $\mu, \sigma, b, q_{adc}, h, s$.

7.3.3 Optimization of case 2

We make approximations in the above analysis about applying CLT to estimate the distribution of sums regardless of the sample size n, for both sums within hardware dot products and between hardware dot products. We observe that in most cases CLT provides good estimations even with a small sample size ($n < 5$), for example, the sum of truncated normal distribution and uniform distribution. Generally speaking, the closer the sample distribution compared to a normal distribution, the smaller the required sample size to make CLT converge. However, for the sum of clipping error, each sample only draws from the tail of the distribution (clipped values), which makes CLT require a larger sample size to converge [Hal80].

We observe that the required sample size to apply CLT on the sum of clipping error depends on the ADC saturation threshold q_{adc} and the distribution of the hardware dot product results Y. Larger q_{adc} compared to Y can make the probability of clipping error smaller, which requires more samples for CLT to be accurate ($n < 100$ for some cases). The results suggest that for these cases, the analytical model can significantly overestimate the clipping error when the number of digital accumulation $\lceil \frac{s}{h} \rceil$ is small.

To make our model more generalizable, we propose an empirical model to improve CLT accuracy on the sum of clipping errors, for cases where ADC saturation is rare and $\lceil \frac{s}{h} \rceil$ is small. In such cases, the CLT tends to overestimate the variance, since the real distribution of sums usually has a large density at zero. Therefore a correction formula to reduce the estimated variance value can improve the accuracy of CLT. Let the hardware dot product results be a normal distribution with mean μ and standard deviation σ, we create a parameter α such that $q_{adc} = \mu + \alpha\sigma$ to represent how far away the ADC threshold is from the mean of results. Through empirical simulation and modeling, we propose a correction coefficient

c to significantly improve the estimation of the sum of clipping errors, which is defined as:

$$c = \frac{\lceil \frac{s}{h} \rceil}{1.5\alpha^{\alpha+2} + \lceil \frac{s}{h} \rceil} \tag{7.17}$$

We can see from the equation that c is small when α is large and $\lceil \frac{s}{h} \rceil$ is small and tends to 1 when $\lceil \frac{s}{h} \rceil$ is large (where CLT is accurate enough). After applying the correction coefficient, the Equation 7.11 becomes:

$$Z_{cf} \sim \mathcal{N}\left(\left\lceil \frac{s}{h} \right\rceil \mathbb{E}(Z_{ch}), c\left\lceil \frac{s}{h} \right\rceil Var(Z_{ch})\right) \tag{7.18}$$

To evaluate the proposed optimization, we randomly generate n normal distributions $\mathcal{N} \sim (\mu, \sigma)$, each representing a single hardware dot product result, and $n = \lceil \frac{s}{h} \rceil$. We set $\mu = 10$ and $\sigma^2 = 3$ in our evaluation, but their exact value does not affect the results. We clip each distribution with a threshold $q_{adc} = \mu + \alpha\sigma$. Figure 7.2 plots the normalized mean absolute clipping error (normalized with respect to simulation results) of the sum of these distributions obtained from (1): the original analytical model, (2): the optimized analytical model with c, and (3): simulation with large enough sample size, for various n and α. Based on the results, we observe that the original analytical model overestimates Z_{cf} when n is small, especially for large α, which agrees with our analysis. The comparison between the optimized analytical model and simulation results suggests that the proposed correction coefficient significantly improves the estimation for small n, and is valid for all common α values. Larger α values are not plotted because when $\alpha = 3$ the clipping threshold is already at the 3σ point, so clipping rarely happens and the rounding error dominates the quantization error.

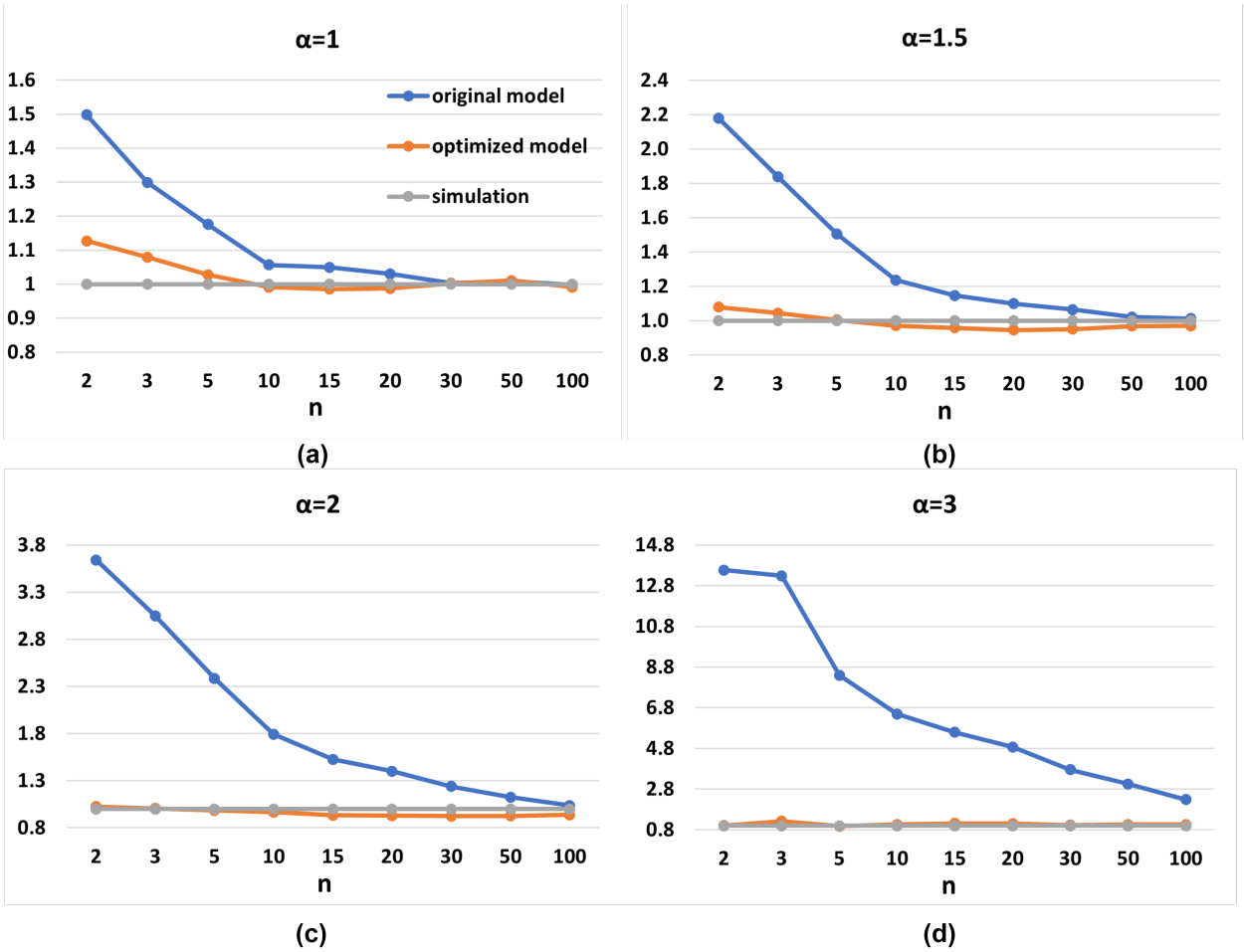


Figure 7.2: Normalized mean absolute clipping error versus number of summations obtained from different methods, for different α. Y-axis is the normalized clipping error.

7.4 Evaluation

In this section, we evaluate the accuracy of the proposed quantization error model against simulation results for both cases introduced in Section 7.3. We also include a case study on an example PIM architecture, showing how the hardware dot product size and ADC bitwidth can affect the quantization error and power efficiency of the system.

7.4.1 Case 1

To evaluate the accuracy of the proposed analytical model for the case where the ADC range is set according to the hardware dot product size such that saturation will never happen, we use mean absolute quantization error as the metric and compare the results generated by the model with simulation results. We set the full dot product size s to be 2000 and sweep the hardware dot product size h to generate the results in Figure 7.3. The results indicate that the analytical model agrees with simulation results for all hardware dot product sizes.

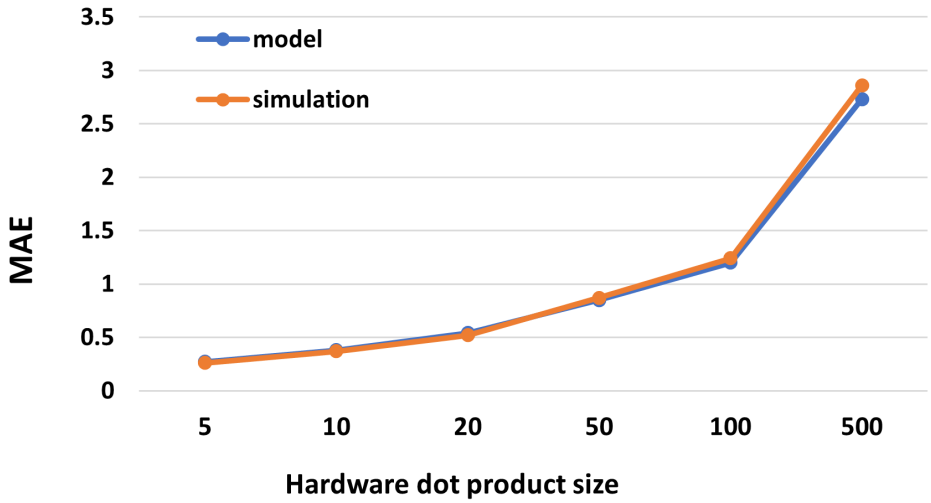


Figure 7.3: Mean absolute clipping error generated with the proposed model and simulation, for different hardware dot product sizes.

7.4.2 Case 2

To evaluate the proposed analytical model for the case where ADC saturation is allowed, we use the model to generate the mean absolute quantization error (rounding error + clipping error) with different values of the model parameters and compare it with the simulation results. Due to the large number of model parameters, it's impossible to generate plots for the cases where multiple/all parameters are changing at the same time. Therefore we assign a default value to each model parameter, and for each plot we only change one parameter from its default value. We fix the full dot product size s and only change the hardware dot product size h. We set q_{adc} by $q_{adc} = \mathbb{E}(Y) + \alpha Var(Y)$, where Y is the hardware dot product result. We generate the truncated normal distribution from the normal distribution with mean μ and variance σ^2. The default parameter values are shown in Table 7.2. Figure 7.4 shows the mean absolute quantization error plot for different values of α, h, b, and μ, respectively. The results suggest that the proposed model is robust as the model provides good estimations of total quantization error for all cases. With the analytical model, the relationship between overall quantization error with different accelerator configurations can be easily visualized for a given application. Although we cannot show the plots of all possible combinations of model parameters, they are easily obtained from the provided analytical model. For the

Table 7.2: Default value of model parameters.

Parameter	s	h	b	α	μ	σ
Default value	2000	50	6	1.5	2	1

7.4.3 A PIM case study

We use a modified version of the classic ISAAC PIM [SNM16] architecture as an example analog neural network accelerator to explore the effect of ADC bitwidth and hardware dot product size on power efficiency and quantization error. To simplify the analysis as well as make it generalizable to most analog neural network accelerators, we adopt the version of

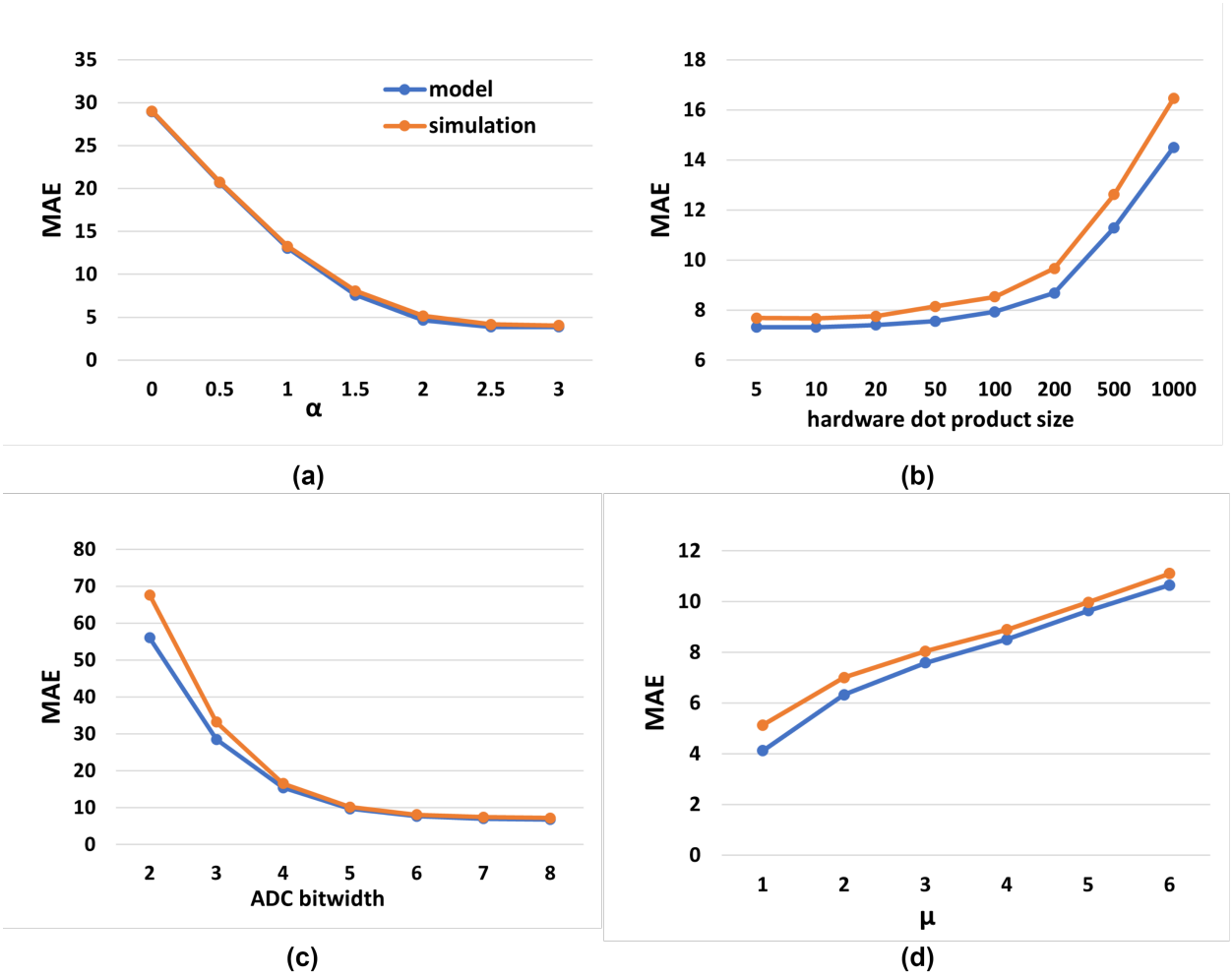


Figure 7.4: Mean absolute clipping error with different model parameters generated using the analytical model and simulation. (a): α (relative ADC threshold). (b): Hardware dot product size. (c) ADC bitwidth. (d): Mean of the original normal distribution before truncation (multiplication outputs).

ISAAC without bit-serial processing. We assume the precision of inputs and weights are 4-bits, which is the configuration used in [HSL16]. In this analysis, we focus on a single memristor array, which contains 128×128 4-bit memory cells. The system can perform one vector-matrix multiplication with size 128×128 per 100 ns. The hardware dot product size in this case is the column size of the memristor array. A single 8-bit 1.28 GHz ADC is shared between all columns of the memristor array to reduce the area and is time-interleaved to convert the output of all columns. Table 7.3 lists the component power in the baseline configuration.

Table 7.3: Summary of component power in the baseline configuration.

Component	Number	Total power
Cell	$128\times$ 128	0.36 mW
DAC	128	0.53 mW
ADC	1	2 mW

We use a simplified power model to model the power of the PIM array, which is essentially the sum of the power of DAC, ADC, and memory cells, which together can consume more than 80% of the total power [SNM16]. The ADC power is adjusted according to the bitwidth b using the relationship: $P_{adc} \propto 2^b$, which can be derived from the Walden ADC figure-of-merit [Wal99]. We use the described configuration of the memristor array as a baseline setup and explore how power efficiency and quantization error scale with the ADC bitwidth and column size. We fix the row size during scaling so that the ADC power per column is constant. We estimate the energy (normalized to the default configuration) of the memristor array for different hardware dot product sizes in Figure 7.5 (a). The default parameter values are kept the same as in Table 7.2, except that s is set to 2048 to fit the column size of the example PIM. When the ADC bitwidth is fixed, it's obvious that a larger column size is better as there are more computations per ADC conversion. However, from Figure 7.3 and Figure 7.4 (b), the optimal column size in terms of quantization error shows a reverse trend: the overall quantization error reduces with column size regardless of whether saturation is allowed or

not. To resolve this discrepancy and explore the optimal column size that takes into account both quantization error and power efficiency, we reduce the ADC bitwidth according to the column size since according to previous results the ADC range can be smaller for smaller arrays. If the ADC bitwidth is scaled directly proportional to the column size, as plotted in Figure 7.5 (a), the system power is the same for all column sizes. Therefore, the optimal column size solely depends on the quantization error, which is plotted in Figure 7.5 (b). The results suggested that with power efficiency roughly on par, a larger column size achieves less overall quantization error, and is preferred as long as the array can be fully utilized.

However, the analysis of power efficiency assumes full array utilization, which cannot be guaranteed for large column sizes. For neural network layers with point-wise convolution and or a small number of filters, the dot product size can be quite small (e.g., 32 to 256), making PIMs with large column sizes have poor power efficiency. The optimal column size still depends on the exact use case and sometimes requires trade-offs between accuracy and efficiency.

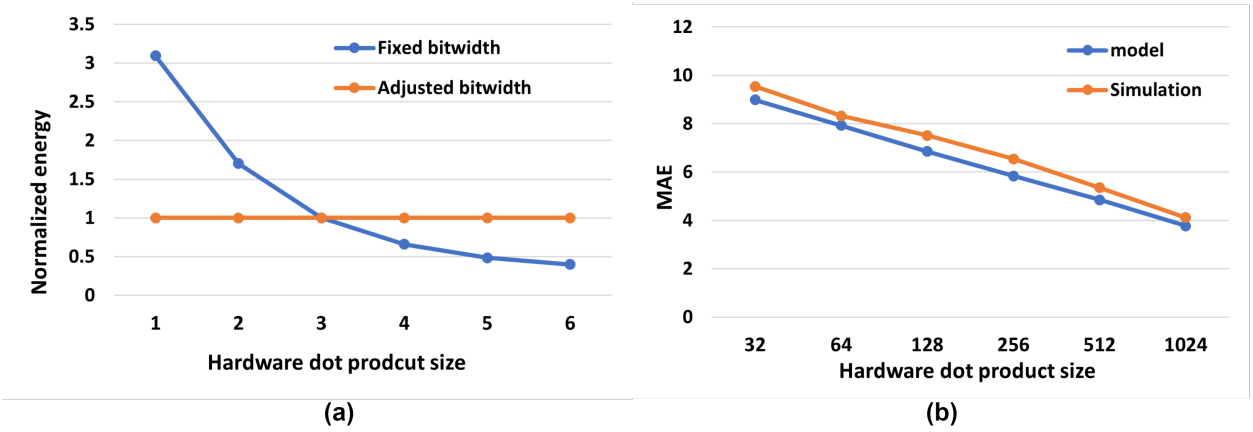


Figure 7.5: (a): Power of the memristor array vs. column sizes for fixed and adjusted ADC bitwidth. (b): Mean absolute quantization error vs. column size for the case with adjusted ADC bitwidth.

7.5 Conclusion

In this work, we propose an accurate analytical model for the overall quantization error of analog neural network accelerators that quantize partial sums. The model provides an efficient and accurate method to explore the impact of hardware dot product size and ADC bitwidth on the overall quantization error. Coupled with an energy/throughput model of the hardware architecture, the developed partial sum error model can be used during the architecture design stage to analyze trade-offs and search for optimal configurations (e.g., ADC bitwidth, column size). Moreover, the model can also be used to find the optimal clipping threshold during quantization that minimizes the overall quantization error for existing analog and digital neural network accelerators. Our ongoing work explores some of these directions to optimize analog accelerators such as PIM and photonic neural network accelerators.

CHAPTER 8

Experimental Analysis of JTC

In this section, the limitations and non-idealities of a real JTC system are discussed, along with their impacts and the methods used to mitigate certain effects. While this dissertation focuses on the architectural aspects of JTC-based neural network accelerators and assumes JTC to be an ideal correlator engine, real optical systems, including JTCs, do not always behave ideally. These non-idealities should be acknowledged and examined. This section provides both simulated and experimental analyses of a real JTC system based on a taped-out, reduced-version prototype of the neural network accelerator proposed in this dissertation. The experimental work is largely credited to my collaborators at both UCLA and the University of Florida (Professor Wong's group and Professor Sorger's group).

We discuss some of the non-idealities and challenges encountered in our JTC prototype, as well as the methods implemented to mitigate their impact. Additionally, the impact of non-linearity within the JTC system, particularly concerning the square function, is analyzed, simulated, and discussed in this section.

For all the accuracy results presented in this chapter, the CIFAR-10 dataset is used as a benchmark, rather than the more commonly utilized MNIST dataset in the field of optical neural network accelerators. The complexity of CIFAR-10 allows for better differentiation of accuracy across different setups and aids in evaluating the impact of various non-idealities in the optical system.

This chapter aims to provide an overview of the non-ideal behaviors of real-world JTC systems, offering directions and insights for future research.

8.1 Non-idealities, Training, and Calibration of the Experimental JTC Setup

8.1.1 JTC Hardware Prototype

Before proceeding with experimental findings and results, it is essential to introduce our fabricated JTC prototype. As a proof-of-concept, the hardware is a fully photonic tape-out of a 16-channel JTC, fabricated by AIM Photonics. It contains only photonic components such as MRRs, lenses, and photodetectors. To achieve the full functionality of a neural network accelerator, the JTC chip must be connected to external ADC and DAC modules via high-speed PCB for signal conversion. It also needs to be connected to the host to access memory and perform non-convolution operations.

Figure 8.1 shows the PCB photo of the prototype with the JTC chip assembled onto it, along with the layout diagram of the JTC chip.

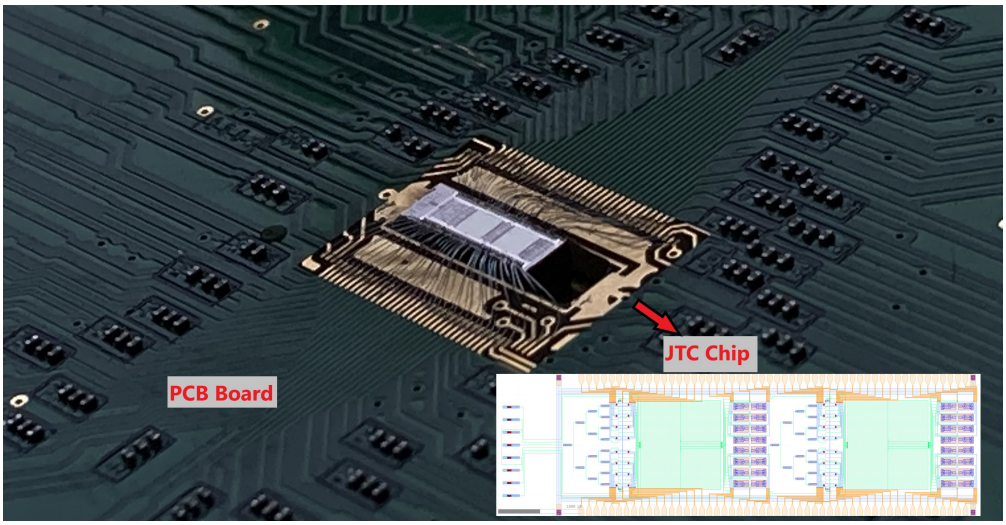


Figure 8.1: PCB photo and layout diagram of the JTC prototype.

8.1.2 Custom training methodology

Unlike the 4F system, JTC outputs not only contain convolution (correlation) results but also include a non-convolution term. This characteristic means that JTC outputs, even in the ideal case, cannot be directly used as convolution results. A more detailed introduction to JTC's operation and output format can be found in Section 4.2. In addition to the unique output format, due to the analog computing nature of JTC, the actual hardware will not generate exact convolution outputs. Consequently, a pretrained neural network cannot be directly executed on a JTC-based accelerator, as pretrained models are typically trained using exact convolution computations. To ensure that neural networks run correctly on JTC for inference tasks, the training flow must account for the behavior of the actual JTC hardware. Similar to the free-space 4F systems discussed earlier in this dissertation, a custom hardware simulator is created and integrated into the PyTorch training flow. In the forward pass of the training process, instead of using the standard convolution function provided by PyTorch, the hardware simulator is called, simulating JTC hardware during the forward pass so that the trained model reflects the underlying hardware characteristics.

The hardware simulator models how the generated light signals propagate through the JTC, interact with the lens, and are eventually received by the photodetectors. To provide fine-grained simulation, each JTC channel is effectively represented by 10 elements in the simulator. The output filtering is also modeled in the simulator, which filters out the first-order non-convolution term. Additionally, the simulator models some non-idealities of the JTC prototype, which are introduced later in this section.

8.1.3 Custom Neural Network Design

Due to the limited number of JTC channels, only 8 input pixels/elements can be processed by the JTC prototype in one cycle (8 channels for input and 8 channels for the filter), making it nearly impossible to process conventional CNNs that typically use 3×3 kernels and much

larger inputs. To address this, we designed a custom CNN model for the JTC prototype to better fit the hardware constraints.

The custom neural network used for evaluation is a single-layer CNN with 8 filters. It consists of a custom convolutional layer followed by two fully connected layers, with 256 and 10 neurons, respectively. The final fully connected layer serves as the classification layer.

A wider or deeper CNN is not adopted due to the limited processing speed of the prototype, which restricts its ability to handle more complex workloads within a reasonable time.

Image Patching Strategy For the CIFAR-10 dataset, where the input image resolution is 32×32, an image patching strategy is employed due to the prototype's limitation of having only 8 channels for inputs and 8 channels for weights. The image is split into 1×8 patches horizontally, resulting in 4 patches per row and 128 patches per image. Each patch is convolved with a 1×8 filter.

To capture information along the vertical dimension as well, the inputs are transposed, and the same convolution operation is performed, effectively patching along the vertical dimension. The results from both the horizontal and vertical convolutions are concatenated and fed into the fully connected layers.

Handling Negative Weights The network uses a pseudo-negative method to handle negative weights. In this approach, the weights are split into positive and negative components, each with the same dimensions. Both components are positive values and are convolved with the inputs separately. The result of the convolution with the negative component is then subtracted from the result of the convolution with the positive component, thereby simulating the behavior of negative weights.

Baseline Accuracy The baseline accuracy achieved by this model is approximately 56

8.1.4 Non-Ideal MRR Outputs

One of the main challenges of the JTC prototype is the non-ideal behavior of the MRR outputs. Ideally, the MRR output should be linearly proportional to the input (the DAC output). However, in practice, the actual MRRs exhibit outputs that follow non-linear response curves. Additionally, real MRR outputs are complex values rather than real values, due to phase modifications.

This discrepancy means that if a network is trained without accounting for the actual MRR behavior, the learned weights may be invalid and fail to produce meaningful results on the hardware. Therefore, it is crucial to model the correct MRR output during the training phase, allowing the neural network to learn this behavior. We implemented an MRR output model using a look-up table and integrated it into the hardware simulator used in the training flow. This ensures that the MRR behavior in the simulator accurately reflects the real-world experimental behavior.

To verify the effectiveness of this approach and evaluate how well the modeling mitigates the issue, we simulated the hardware experiment using a version of the hardware simulation model that includes the actual MRR behavior. The model was trained both with and without consideration of the MRR behavior and evaluated using the same hardware simulation model. However, we currently lack the results from an actual hardware experiment as we do not yet have a fully functional end-to-end setup; this remains a work in progress.

Figure 8.2 presents the simulated hardware accuracy with and without modeling the actual MRR output behavior, compared to the ideal case. The results clearly show that without modeling the actual MRR output during training, the weights are unusable in real hardware, as expected. When the MRR output behavior is modeled during training, the hardware accuracy improves significantly, demonstrating the effectiveness of this approach. Nevertheless, a small gap remains between this accuracy and the ideal case, due to the non-linear quantization effect applied by the MRRs, which results in some information loss.

Part of this accuracy reduction stems from the fact that the MRRs used in this prototype are not state-of-the-art. In future work, this accuracy gap could be further minimized by employing more advanced MRRs with more linear response curves and finer-grained output levels.

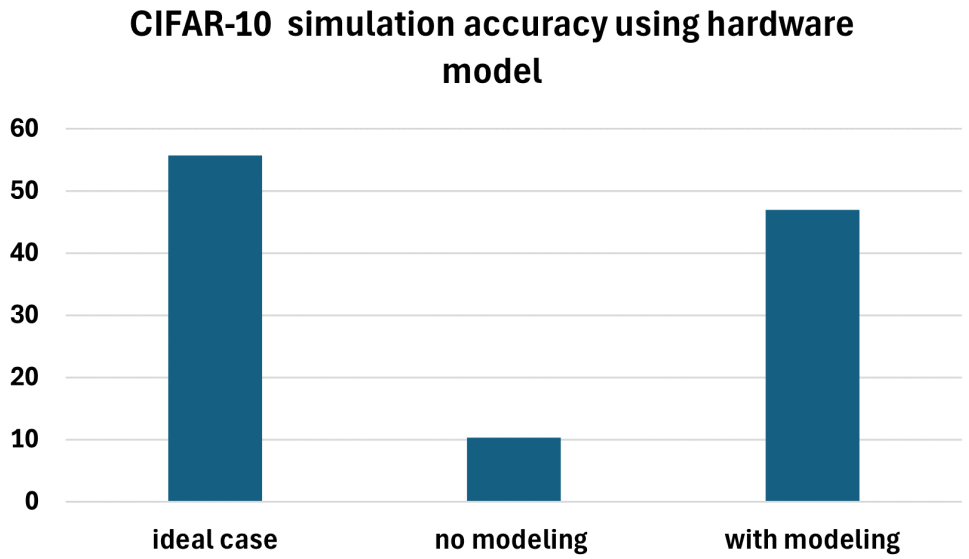


Figure 8.2: CIFAR-10 simulation accuracy using the hardware simulation model, with and without modeling of non-ideal MRR behavior.

8.1.5 Phase difference of input signals

Another major non-ideality observed in the JTC prototype is the phase difference of the light signals. Ideally, the signals from all JTC channels should be generated simultaneously to ensure correct phasing and to avoid any undesired phase differences among the signals. However, in the prototype system, the DACs are off-chip, and their outputs must travel through the PCB to reach the MRRs of the JTC chip. Despite best efforts in PCB design, small but unavoidable differences exist in the wire lengths to each MRR, preventing all signals from being generated at precisely the same time. Although this path difference is minor, it can still result in noticeable unwanted phase differences between JTC channels,

causing the JTC to behave differently than expected.

Similar to the non-ideal MRR outputs, this phase difference in the hardware can invalidate the learned weights if it is not modeled during the training process. To address this, the phase difference is modeled in the hardware simulator and incorporated into the training process, allowing the trained neural network model to account for these phase differences. These differences are obtained from chip measurement results and hard-coded into the hardware simulator. It is important to note that since each JTC hardware system may have its own unique path differences, the neural network model trained using this approach will only be applicable to the specific hardware being measured.

Figure 8.2 shows the simulated hardware accuracy with and without modeling the path difference, compared to the ideal case. Similar to the MRR non-idealities, failing to model the path difference renders the trained neural network unusable, while modeling it during training significantly improves accuracy. However, even with the path difference modeled, there remains a noticeable accuracy drop compared to the ideal case, which is expected since the outputs are not ideal convolution results.

A better solution to this problem would be to calibrate the path difference so that the signals generated by the MRRs exhibit minimal unwanted phase variation. This calibration is feasible since the exact path difference and corresponding phase error can be obtained through testing and measuring the JTC chip. Calibration can be performed using an FPGA or a similar controller to adjust the signals fed into the MRRs on the JTC chip, which is a work in progress.

While calibration effectively mitigates this issue, a more desirable solution would be to eliminate the path difference at its source. This can be achieved by integrating the ADCs and DACs onto the same chip as the JTC, thereby removing the need for signals to travel through PCB wiring, which is the root cause of the path difference. Such monolithic integration is already available from certain foundries, including GlobalFoundries' 90nm and 45nm technology nodes [GNA19, RMN20].

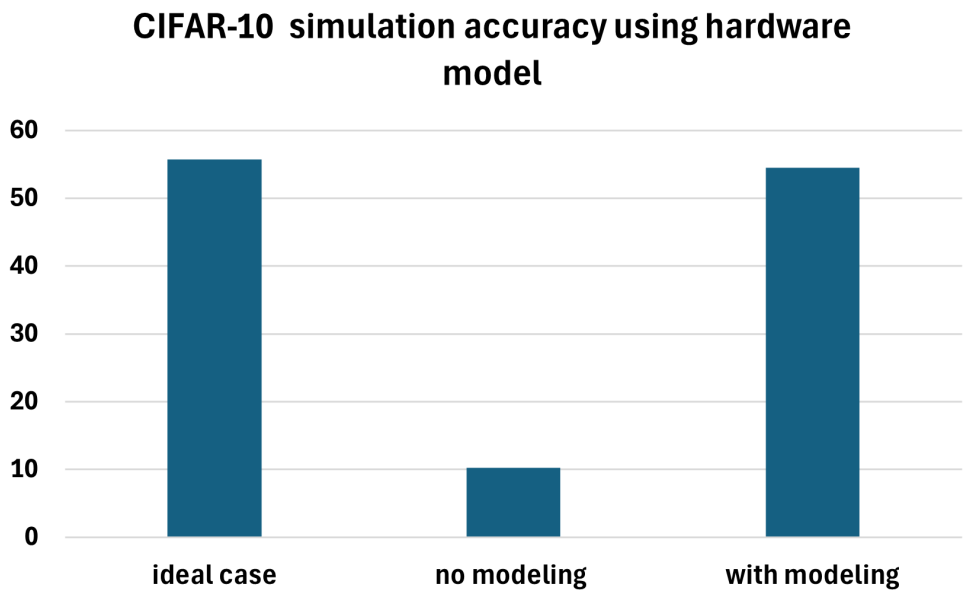


Figure 8.3: CIFAR-10 simulation accuracy using the hardware simulation model, with and without modeling of the path difference between JTC channels.

8.2 Sensitivity of Neural Network Accuracy to Optical System Non-Linearity

The theoretical operation of a classic JTC requires a precise square function to be applied to the signals in the Fourier plane in order to generate mathematically accurate correlation results at its output plane. However, due to the nature of analog computing, various sources can introduce non-linearity into the system, preventing the JTC from behaving in an ideal, theoretical manner. One major source of non-linearity is the photodetectors used to implement the square function in the Fourier plane. Photodetectors, typically made from photodiodes, do not have a perfectly linear response curve between incident light intensity and the generated current.

On one hand, if the incident light intensity is too low, dark current can dominate, significantly reducing the SNR and making the response non-linear. On the other hand, if the incident light intensity is too high, the photodetector can saturate, causing the signals to

be effectively clipped. These additional sources of non-linearity can prevent the JTC from functioning as a classic, mathematically equivalent correlation calculator.

However, this non-linearity may not pose as significant a problem when JTCs are used to implement neural networks, and it could even be beneficial to overall performance. Since the goal of a neural network, such as a CNN, is to extract features, exact convolution is not a strict requirement—any function, including an approximation to convolution, can be used as long as it can extract useful features and is properly modeled during training.

In fact, there is a broad area of research dedicated to non-linear JTCs, where non-linearity is intentionally introduced into the classic JTC setup to improve SNR in vision tasks requiring correlation.

This section provides a brief introduction to non-linear JTCs, followed by an evaluation of how the non-linearity of photodetectors affects system accuracy, based on simulations of our JTC prototype.

8.2.1 Non-Linear JTC

A non-linear JTC is a variant of the classic JTC, which introduces additional non-linearity into the setup at the Fourier plane. Non-linear JTCs have been widely researched for computer vision and encoding tasks that require correlation operations [GSY22, JWT94, PCS97, RDM98, Jeo10, WT98, AK98]. The central concept behind non-linear JTCs is that, despite the added non-linearity, they still act as effective correlation approximators while improving the SNR of the correlation outputs compared to classical JTCs.

[Kuo92] offers a solid theoretical foundation for the correlation signal of non-linear JTCs. According to this work, the correlation signal of a non-linear JTC is the convolution of the conventional correlation signal with a non-linear one. Additionally, unlike conventional JTCs, non-linear JTCs utilize all the interference intensities to construct the correlation signal. This results in an increased correlation signal compared to conventional JTCs, which

allocate the majority of interference intensities to the non-correlation term.

For the JTC-based neural network accelerators proposed in this dissertation, while the initial aim was to use classic JTCs and ensure the system behaves as linearly as possible, the non-linear response of the photodetectors in the Fourier plane effectively transforms the system into a non-linear JTC. The photodetector's non-linearity can be treated as an additional non-linearity introduced at the Fourier plane, similar to the behavior of non-linear JTCs described above.

8.2.2 Evaluation Results

The hypothesis that the non-linearities of photodetectors, used to implement the square function in a classic JTC, may not degrade and might even improve the accuracy of CNNs is evaluated and validated using the simulator of our JTC prototype. The hardware simulator employed for this evaluation, as well as the overall training flow, is the same as the one introduced in Section 8.1.2. The modeling of actual MRR behavior and path differences is incorporated in all evaluations conducted for this section. The only variation between different setups is how the square function is implemented in the Fourier plane, with the baseline case using a perfect square function.

Real Photodetector Transfer Function To assess the impact of photodetector non-linearity on neural network accuracy, we obtained the response curve and transfer function of real photodetectors from GlobalFoundries. The exact response curve is not included here due to potential restrictions on public access to this data. The laser is configured to allow the JTC to operate primarily in the linear region, with some saturation permitted. Table 8.1 presents the simulation accuracy of the JTC prototype modeled with an ideal square function versus the JTC prototype modeled using the actual transfer function of the photodetectors, evaluated on the CIFAR-10 dataset. The accuracy for the actual transfer function is slightly higher than that of the ideal square function.

Ideal square function	Actual transfer function
47.9	48.5

Table 8.1: Simulation accuracy on CIFAR-10 dataset of the JTC prototype using ideal square function versus the JTC prototype using the actual transfer function of photodetectors.

Accuracy Results of Different Non-Linearities While the accuracy results from modeling the actual transfer function of the photodetector suggest that it achieves better accuracy than using the ideal square function, this is only one case, and more data is required to generalize the argument. To further investigate, we conducted two additional sets of experiments to explore the impact of various hypothetical cases of non-linearity at the photodetector, simulating different photodetector behaviors.

The first experiment examines the effect of varying the exact power function applied to the signals in the Fourier plane by the photodetectors. In the ideal case, where a square function is implemented, the power is 2. However, some photodetectors may not implement an exact square function but may instead apply a power of 2.2. This experiment aims to evaluate how such behavior affects the overall accuracy of the neural networks. The saturation effect is not considered in this evaluation.

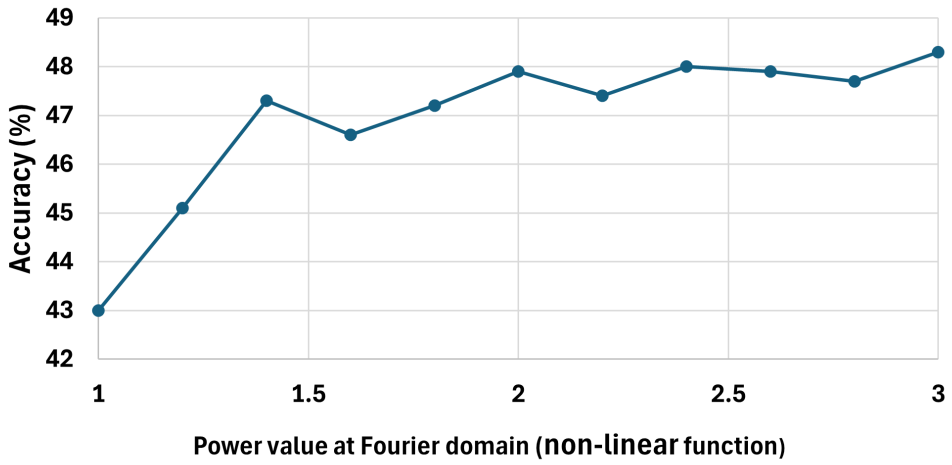


Figure 8.4: CIFAR-10 simulation accuracy using the hardware simulation model, for various power functions applied at the Fourier plane.

Figure 8.4 shows the simulation accuracy for various power functions applied at the Fourier plane. A sweep from 1.0 (identity function) to 3.0 was performed, with a step size of 0.2. The results indicate that accuracy degrades when the power value is less than 2 but can achieve a similar level of accuracy as the ideal square function when the power value is greater than 2. Therefore, as long as the photodetector behavior is reasonably close to a square function, the impact on accuracy is not significant.

The second experiment is to study the impact of saturation of photodetectors, which is likely to happen in actual hardware. Since how many channels could saturate the photodetector depends on many factors such as laser power, photodetector configuration and, number of channels, a sweep is performed to cover cases with different saturation probabilities. Although the saturation effect is not a perfect clip function for real photodetectors, a clip function is used to model saturation in this experiment since the exact saturation behavior is photodetector-dependent. A saturation probability of $X\%$ means the largest $X\%$ values of all the channels after the square function will be clipped to the nearest value of the $X\%$ percentile.

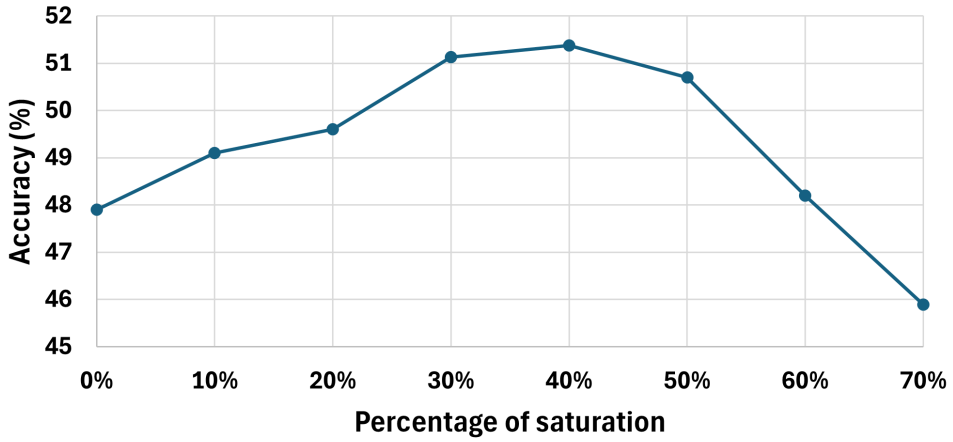


Figure 8.5: CIFAR-10 simulation accuracy using the hardware simulation model, for various photodetector saturation probabilities.

Figure 8.5 shows the simulation accuracy for various photodetector saturation probabil-

ities. The results are surprising yet reasonable. For any saturation probability less than 60%, the accuracy is higher than the baseline case with no saturation at all (0%). The surprising part is that photodetector saturation can actually improve accuracy rather than hurt it. This finding is reasonable in the sense that it verifies the claim that non-linear JTCs can be a better correlator than classic JTCs. The saturation in this case serves as a good non-linearity source for JTCs.

We hope this interesting finding could provide directions and insights for future works, showing that certain non-linearities can be tolerated, or even exploited, to improve the overall performance of JTC-based neural network accelerators.

CHAPTER 9

Conclusion and Directions for Future Work

9.1 Summary of contributions

This dissertation explores the challenges and potential solutions for enhancing the energy efficiency and performance of neural network accelerators, specifically focusing on 4F/JTC-based photonic neural network accelerators. The motivation for this work stems from the limitations of traditional CMOS-based accelerators, constrained increasingly by Moore's Law, alongside the growing computational demands of complex neural networks.

Throughout this dissertation, several key contributions were made, each targeting a critical aspect of the challenge:

9.1.1 Custom training flow and neural network model for optical neural network accelerators

Given the nature of analog computing, photonic neural network accelerators, such as the 4F and JTC-based systems, do not behave in the same manner as digital electronics. They do not produce exact convolution results due to system properties, non-idealities, and noise, rendering standard neural network training flows that utilize ideal convolutions inapplicable. This dissertation proposes a novel training method for photonic-based accelerators by integrating a simulation model of the hardware into the training pipeline, making the neural network cognizant of the underlying hardware constraints. This training approach significantly improves the experimental accuracy of the trained neural network compared to

traditional methods, and has been adopted by both the 4F and JTC-based neural network accelerator projects discussed in this dissertation.

A custom neural network model was also designed specifically for our JTC prototype, as conventional neural networks are not supported due to the limited number of channels available.

9.1.2 Efficient architectures for on-chip photonic neural network accelerators

Building upon the initial proof-of-concept, the second part of the dissertation focuses on on-chip photonic neural network accelerators. These systems, based on the Joint Transform Correlator (JTC) architecture, extend the benefits of the 4F system by integrating photonic components onto a chip. Various architectural optimizations were proposed, including wavelength-division multiplexing (WDM), temporal accumulation, and delay line-based optical buffers, all aimed at mitigating the high power consumption associated with DAC and ADC conversions. The resulting architectures demonstrated significant improvements in both energy efficiency and performance, surpassing prior state-of-the-art photonic neural network accelerators.

Additionally, the evaluation results based on our custom performance model, such as the power and area breakdown, provide useful insights into the capabilities of JTC-based neural network accelerators.

9.1.3 Weight pool compression algorithm

In the third part, a novel compression algorithm, termed weight pool, was introduced to address the memory access energy and DAC power limitations of analog neural network accelerators. This algorithm reduces the storage and memory traffic requirements by sharing weight vectors across the network, enabling up to 8× compression with minimal accuracy loss. The synergy between weight pool compression and photonic accelerators was explored,

highlighting the potential for further reducing DAC power consumption and enhancing the overall energy efficiency of photonic systems.

The proposed weight pool also serves as an effective standalone compression algorithm and can be directly used for almost all types of CMOS-based accelerators to reduce memory storage requirements and DRAM traffic.

9.1.4 Experimental insights and real-world challenges

The final chapters explore the challenges encountered during the experimental implementation of Joint Transform Correlator (JTC)-based photonic systems, including non-idealities such as photodetector non-linearity and device variations. Strategies to mitigate these challenges were presented, setting the stage for future enhancements in the design and optimization of photonic neural network accelerators.

9.2 Directions for future works

9.2.1 Improving the simulation model of JTC hardware

The current simulation model of JTC hardware is quite sophisticated, accurately modeling several behaviors such as the actual transfer function of Micro-Ring Resonators (MRR), the path difference between channels, and JTC output filtering. However, there remain unmodeled error sources, such as lens misalignment and photodetector non-idealities.

Future work could aim to model these error sources in a highly parameterized manner within the system simulator to better approximate actual hardware behavior. Enhancing the simulator's accuracy can also improve training quality and, consequently, the overall accuracy of the models.

9.2.2 A more advanced JTC prototype

As a proof-of-concept, a reduced-scale JTC prototype was fabricated with AIM Photonics, featuring a single JTC chip with a total of 16 channels. While this prototype serves well for demonstration purposes, it has several limitations, including the absence of on-chip ADCs and DACs, a limited number of channels, and noticeable variations in component behavior.

Looking ahead, the development of a larger, more advanced JTC prototype should be considered, preferably employing advanced technology that can integrate optical and CMOS components monolithically on the same chip. Such prototypes could further validate the utility of JTC-based neural network accelerators and bring the architectures proposed in this dissertation closer to practical realization.

9.2.3 Custom deep neural network for 4F/JTC based accelerators

In this dissertation, a custom 1-layer Convolutional Neural Network (CNN) was designed for our JTC prototype. The decision to use a single-layer network was primarily due to the limited processing speed of the prototype. In the future, as more advanced JTC prototypes or even full-scale accelerators are developed, a single-layer CNN will be insufficient. A deeper neural network, customized for JTCs, should be developed to enhance the overall performance of JTC hardware.

9.2.4 Efficient compute-in-memory (CIM) using weight pool

The proposed weight pool compression algorithm also demonstrates intriguing synergy with compute-in-memory (CIM), especially CIMs based on non-volatile memory such as Resistive Random-Access Memory (RRAM) and Magnetoresistive Random-Access Memory (MRAM). These types of CIMs typically suffer from slow write speeds and limited write endurance, necessitating that networks be fully unrolled due to constraints in updating weights. Consequently, such CIMs exhibit minimal run-time programmability, which in turn limits their

potential applications.

However, by employing weight pools, the total number of unique weight vectors can be limited to 256 or fewer, irrespective of the neural network's size. This allows the entire weight pool (comprising unique weight vectors) to be stored in a single CIM array and reused throughout the execution of the entire neural network. With strategic architectural design, this approach can significantly reduce the overall area and enhance the flexibility of non-volatile memory-based CIMs.

REFERENCES

[ADB16] M Zahirul Alam, Israel De Leon, and Robert W Boyd. "Large optical nonlinearity of indium tin oxide in its epsilon-near-zero region." *Science*, **352**(6287):795–797, 2016.

[ADJ17] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. "Bit-pragmatic deep neural network computing." In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 382–394, 2017.

[AGW21] Rubab Amin, Jonathan K. George, Hao Wang, Rishi Maiti, Zhizhen Ma, Hamed Dalir, Jacob B. Khurgin, and Volker J. Sorger. "An ITO–graphene heterojunction integrated absorption modulator on Si-photonics for neuromorphic nonlinear activation." *APL Photonics*, **6**(12):120801, 2021.

[AHR18] Shaahin Angizi, Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator." In *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

[AIM] AIM Photonics. Available online: www.aimphotonics.com.

[AK98] Mohammad S Alam and Jehad Khoury. "Fringe-adjusted incoherent erasure joint transform correlator." *Optical Engineering*, **37**(1):75–82, 1998.

[AUO17] Kota Ando, Kodai Ueyoshi, Kentaro Orimo, Haruyoshi Yonekawa, Shimpei Sato, Hiroki Nakahara, Shinya Takamaeda-Yamazaki, Masayuki Ikebe, Tetsuya Asai, and Tadahiro Kuroda. "BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W." *IEEE Journal of Solid-State Circuits*, **53**(4):983–994, 2017.

[AVR] AVR Optics. "HSP1920." https://www.avr-optics.com/catalog/spatial-light-modulators-scanning-systems/xy-slm/hsp1920.

[BCD21] Anthony Berthelier, Thierry Chateau, Stefan Duffner, Christophe Garcia, and Christophe Blanc. "Deep model compression and architecture optimization for embedded systems: A survey." *Journal of Signal Processing Systems*, **93**(8):863–878, 2021.

[BCX15] Qiaoliang Bao, Jianqiang Chen, Yuanjiang Xiang, Kai Zhang, Shaojuan Li, Xiaofang Jiang, Qing-Hua Xu, Kian Ping Loh, and T Venkatesan. "Graphene nanobubbles: a new optical nonlinear material." *Advanced Optical Materials*, **3**(6):744–749, 2015.

[BMM19] Viraj Bangari, Bicky A Marquez, Heidi Miller, Alexander N Tait, Mitchell A Nahmias, Thomas Ferreira de Lima, Hsuan-Tung Peng, Paul R Prucnal, and Bhavin J Shastri. "Digital electronics and analog photonics for convolutional neural networks (DEAP-CNNs)." *IEEE Journal of Selected Topics in Quantum Electronics*, **26**(1):1–13, 2019.

[BNH18] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. "Post-training 4-bit quantization of convolution networks for rapid-deployment." *arXiv preprint arXiv:1810.05723*, 2018.

[BRT21] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, and Danilo Pau. "MLPerf Tiny Benchmark." *arXiv preprint arXiv:2106.07597*, 2021.

[BSS18] Hengameh Bagherian, Scott Skirlo, Yichen Shen, Huaiyu Meng, Vladimir Ceperic, and Marin Soljacic. "On-chip optical convolutional neural networks." *arXiv preprint arXiv:1808.03303*, 2018.

[CBA22] Alexander Chen, Amir Begović, Stephen Anderson, and Zhaoran Rena Huang. "On-Chip Slow-Light SiN Bragg Grating Waveguides." *IEEE Photonics Journal*, **14**(6):1–6, 2022.

[CCS19] Shane Colburn, Yi Chu, Eli Shilzerman, and Arka Majumdar. "Optical frontend for a convolutional neural network." *Applied optics*, **58**(12):3179–3186, 2019.

[CDS14] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning." *SIGARCH Comput. Archit. News*, **42**(1):269–284, feb 2014.

[CES16] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks." In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 367–379, 2016.

[CLX16] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory." *SIGARCH Comput. Archit. News*, **44**(3):27–39, jun 2016.

[CMA20] Pietro Caragiulo, Oscar Elisio Mattia, Amin Arbabian, and Boris Murmann. "A Compact 14 GS/s 8-Bit Switched-Capacitor DAC in 16 nm FinFET CMOS." In *2020 IEEE Symposium on VLSI Circuits*, pp. 1–2, 2020.

[CMC18] Meghan Cowan, Thierry Moreau, Tianqi Chen, and Luis Ceze. "Automating generation of low precision deep learning operators." *arXiv preprint arXiv:1810.11066*, 2018.

[CMC20] Meghan Cowan, Thierry Moreau, Tianqi Chen, James Bornholt, and Luis Ceze. *Automatic Generation of High-Performance Quantized Machine Learning Kernels*, p. 305–316. Association for Computing Machinery, New York, NY, USA, 2020.

[Col18] N. Collings. *Fourier Optics in Image Processing*. Series in Optics and Optoelectronics. CRC Press, 2018.

[CRC20] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. "Generative pretraining from pixels." In *International conference on machine learning*, pp. 1691–1703. PMLR, 2020.

[CSD18] Julie Chang, Vincent Sitzmann, Xiong Dun, Wolfgang Heidrich, and Gordon Wetzstein. "Hybrid optical-electronic convolutional neural networks with optimized diffractive optics for image classification." *Scientific reports*, **8**(1):1–10, 2018.

[CWV18] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. "Pact: Parameterized clipping activation for quantized neural networks." *arXiv preprint arXiv:1805.06085*, 2018.

[DBK20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929*, 2020.

[DCC19] Lorenzo De Marinis, Marco Cococcioni, Piero Castoldi, and Nicola Andriolli. "Photonic Neural Networks: A Survey." *IEEE Access*, **7**:175827–175841, 2019.

[DJB13] A Descos, C Jany, D Bordel, H Duprez, G Beninca de Farias, P Brianceau, S Menezo, and B Ben Bakir. "Heterogeneously integrated III-V/Si distributed Bragg reflector laser with adiabatic coupling." In *39th European Conference and Exhibition on Optical Communication (ECOC 2013)*, pp. 1–3. IET, 2013.

[DLH16] C. Dong, C. C. Loy, K. He, and X. Tang. "Image Super-Resolution Using Deep Convolutional Networks." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **38**(2):295–307, 2016.

[DNO20] Etienne Dupuis, David Novo, Ian O'Connor, and Alberto Bosio. "On the Automatic Exploration of Weight Sharing for Deep Neural Network Compression." In

2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1319–1322, 2020.

[DNO22] Etienne Dupuis, David Novo, Ian O'Connor, and Alberto Bosio. "A Heuristic Exploration of Retraining-free Weight-Sharing for CNN Compression." In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 134–139, 2022.

[DZZ19] Quan Deng, Youtao Zhang, Minxuan Zhang, and Jun Yang. "Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator." In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[FFG21] João Dinis Ferreira, Gabriel Falcao, Juan Gómez-Luna, Mohammed Alser, Lois Orosa, Mohammad Sadrosadati, Jeremie S Kim, Geraldo F Oliveira, Taha Shahroodi, Anant Nori, et al. "pLUTo: In-DRAM Lookup Tables to Enable Massively Parallel General-Purpose Computation." *arXiv preprint arXiv:2104.07699*, 2021.

[GBM14] Sebastianus A Goorden, Jacopo Bertolotti, and Allard P Mosk. "Superpixel-based spatial amplitude and phase modulation using a digital micromirror device." *Optics express*, **22**(15):17999–18009, 2014.

[GDD13] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation." *CoRR*, **abs/1311.2524**, 2013.

[GJN18] Harshit Gupta, Kyong Hwan Jin, Ha Q Nguyen, Michael T McCann, and Michael Unser. "CNN-based projected gradient descent for consistent CT image reconstruction." *IEEE transactions on medical imaging*, **37**(6):1440–1453, 2018.

[GL22] Puneet Gupta and Shurui Li. "4F optical neural network acceleration: an architecture perspective." In *Proc. of SPIE Vol*, volume 12019, pp. 120190B–1, 2022.

[GLF11] Peng Ge, Qi Li, Huajun Feng, and Zhihai Xu. "Displacement measurement using phase-encoded target joint transform correlator." *Optical Engineering*, **50**(3):038201, 2011.

[GMA18] Jonathan George, Amin Mehrabian, Rubab Amin, Tarek El-Ghazawi, Paul K. Prucnal, and Volker J. Sorger. "Photonic Neural Network Nonlinear Activation Functions by Electrooptic Absorption Modulators." In *Frontiers in Optics / Laser Science*, p. JW3A.123. Optical Society of America, 2018.

[GML21] John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. "Adaptive fourier neural operators: Efficient token mixers for transformers." *arXiv preprint arXiv:2111.13587*, 2021.

[GMS18] Humberto González, Lluís Martínez-León, Fernando Soldevila, Ma Araiza-Esquivel, Jesús Lancis, and Enrique Tajahuerce. "High sampling rate single-pixel digital holography system employing a DMD and phase-encoded patterns." *Optics express*, **26**(16):20342–20350, 2018.

[GNA19] Ken Giewont, Karen Nummy, Frederick A. Anderson, Javier Ayala, Tymon Barwicz, Yusheng Bian, Kevin K. Dezfulian, Douglas M. Gill, Thomas Houghton, Shuren Hu, Bo Peng, Michal Rakowski, Stewart Rauch, Jessie C. Rosenberg, Asli Sahin, Ian Stobert, and Andy Stricker. "300-mm Monolithic Silicon Photonics Foundry Technology." *IEEE Journal of Selected Topics in Quantum Electronics*, **25**(5):1–11, 2019.

[Goo05] Joseph W Goodman. *Introduction to Fourier optics*. Roberts and Company Publishers, 2005.

[GSY22] Jonathan K George, Maria Solyanik-Gorgone, Hangbo Yang, Chee Wei Wong, and Volker J Sorger. "Nonlinear optical joint transform correlator for low latency convolution operations." *arXiv preprint arXiv:2202.06444*, 2022.

[Guo18] Yunhui Guo. "A survey on methods and theories of quantized neural networks." *arXiv preprint arXiv:1808.04752*, 2018.

[GWL18] Fei Gao, Teresa Wu, Jing Li, Bin Zheng, Lingxiang Ruan, Desheng Shang, and Bhavika Patel. "SD-CNN: A shallow-deep CNN for improved breast cancer diagnosis." *Computerized Medical Imaging and Graphics*, **70**:53–62, 2018.

[GZF20] Jiaqi Gu, Zheng Zhao, Chenghao Feng, Mingjie Liu, Ray T. Chen, and David Z. Pan. "Towards Area-Efficient Optical Neural Networks: An FFT-based Architecture." In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 476–481, 2020.

[h1023] "NVIDIA H100 Tensor Core GPU.", 2023.

[Hal80] Peter Hall. "Characterizing the Rate of Convergence in the Central Limit Theorem." *The Annals of Probability*, **8**(6):1037 – 1048, 1980.

[HC19] Mohsen Hassanpourghadi and Mike Shuo-Wei Chen. "A 2-way 7.3-bit 10 GS/s Time-based Folding ADC with Passive Pulse-Shrinking Cells." In *2019 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4, 2019.

[HDP08] L. Haspeslagh, J. De Coster, O. V. Pedreira, I. De Wolf, B. Du Bois, A. Verbist, R. Van Hoof, M. Willegems, S. Locorotondo, G. Bryce, J. Vaes, B. van Drieenhuizen, and A. Witvrouw. "Highly reliable CMOS-integrated 11MPixel SiGe-based micro-mirror arrays for high-end industrial applications." In *2008 IEEE International Electron Devices Meeting*, pp. 1–4, 2008.

[HK92a] G. Healey and R. Kondepudy. "CCD camera calibration and noise estimation." In *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 90,91,92,93,94,95, Los Alamitos, CA, USA, jun 1992. IEEE Computer Society.

[HK92b] Glenn Healey and Raghava V. Kondepudy. "Modeling and calibrating CCD cameras for illumination-insensitive machine vision." In Donald J. Svetkoff, editor, *Optics, Illumination, and Image Sensing for Machine Vision VI*, volume 1614, pp. 121 – 132. International Society for Optics and Photonics, SPIE, 1992.

[HLS] Zibo Hu, Shurui Li, Russell L. T. Schwartz, Maria Solyanik-Gorgone, Mario Miscuglio, Puneet Gupta, and Volker J. Sorger. "High-Throughput Multichannel Parallelized Diffraction Convolutional Neural Network Accelerator." *Laser & Photonics Reviews*, **n/a**(n/a):2200213.

[HLS22] Zibo Hu, Shurui Li, Russell LT Schwartz, Maria Solyanik-Gorgone, Mario Miscuglio, Puneet Gupta, and Volker J Sorger. "High-throughput multichannel parallelized diffraction convolutional neural network accelerator." *Laser & Photonics Reviews*, **16**(12):2200213, 2022.

[HOL] HOLOEYE. "GAEA-2." https://holoeye.com/gaea-4k-phase-only-spatial-light-modulator/.

[HQS14] Xue-ting Hong, Yi-xian Qian, Xiao-wei Shi, and Deng-hui Li. "Measurement of sub-pixel image motion based on joint transform correlator." In *Holography, Diffractive Optics, and Applications VI*, volume 9271, pp. 327–334. SPIE, 2014.

[HSL16] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M. Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R. Stanley Williams. "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication." In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2016.

[HSM22] Chaoran Huang, Volker J Sorger, Mario Miscuglio, Mohammed Al-Qadasi, Avilash Mukherjee, Lutz Lampe, Mitchell Nichols, Alexander N Tait, Thomas Ferreira de Lima, Bicky A Marquez, et al. "Prospects and applications of photonic neural networks." *Advances in Physics: X*, **7**(1):1981155, 2022.

[HX17] Wei Huang and Zhengyuan Xu. "Characteristics and performance of image sensor communication." *IEEE Photonics Journal*, **9**(2):1–19, 2017.

[HZL18] Yuwei Hu, Jidong Zhai, Dinghua Li, Yifan Gong, Yuhao Zhu, Wei Liu, Lei Su, and Jiangming Jin. "BitFlow: Exploiting Vector Parallelism for Binary Neural Networks on CPU." In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 244–253, 2018.

[HZR16] 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.

[JAH16] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. "Stripes: Bit-serial deep neural network computing." In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12. IEEE, 2016.

[Jav90] Bahram Javidi. "Comparison of nonlinear joint transform correlator and nonlinearly transformed matched filter based correlator for noisy input scenes." *Optical Engineering*, **29**(9):1013–1020, 1990.

[Jeo10] Man-Ho Jeong. "Binary nonlinear joint transform correlator with sinusoidal iterative filter in spectrum domain." *Journal of the Optical Society of Korea*, **14**(4):357–362, 2010.

[JKB95] Norman L Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. *Continuous univariate distributions, volume 2*, volume 289. John wiley & sons, 1995.

[JT16] Hojoong Jung and Hong X. Tang. "Aluminum nitride as nonlinear optical material for on-chip frequency comb generation and frequency conversion." *Nanophotonics*, **5**(2):263–271, 2016.

[JWT94] Bahram Javidi, Jun Wang, and Qing Tang. "Nonlinear joint transform correlators." *Pattern recognition*, **27**(4):523–542, 1994.

[KKA94] Jehad Khoury, Jonathan S Kane, George Asimellis, Mark Cronin-Golomb, and Charles Woods. "All-optical nonlinear joint Fourier transform correlator." *Applied optics*, **33**(35):8216–8225, 1994.

[KLL21] Beomseok Kang, Anni Lu, Yun Long, Daehyun Kim, Shimeng Yu, and Saibal Mukhopadhyay. "Genetic Algorithm-Based Energy-Aware CNN Quantization for Processing-In-Memory Architecture." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, **11**(4):649–662, 2021.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[Kuo92] Chung J. Kuo. "Theoretical expression for the correlation signal of nonlinear joint-transform correlators." *Appl. Opt.*, **31**(29):6264–6271, Oct 1992.

[LA04] Chye-Hwa Chye Hwa Loo and Mohammad S Alam. "Invariant object tracking using fringe-adjusted joint transform correlator." *Optical Engineering*, **43**(9):2175–2183, 2004.

[LAE21] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. "Fnet: Mixing tokens with fourier transforms." *arXiv preprint arXiv:2105.03824*, 2021.

[LCL12] Hansuek Lee, Tong Chen, Jiang Li, Oskar Painter, and Kerry J Vahala. "Ultra-low-loss optical delay line on a silicon chip." *Nature communications*, **3**(1):867, 2012.

[LDD01] Hubert K. Lakner, Peter Duerr, Ulrike Dauderstaedt, Wolfgang Doleschal, and Joerg Amelung. "Design and fabrication of micromirror arrays for UV lithography." In M. Edward Motamedi and Rolf Goering, editors, *MOEMS and Miniaturized Systems II*, volume 4561, pp. 255 – 264. International Society for Optics and Photonics, SPIE, 2001.

[LDY23] Kun Liao, Tianxiang Dai, Qiuchen Yan, Xiaoyong Hu, and Qihuang Gong. "Integrated photonic neural networks: Opportunities and challenges." *ACS Photonics*, **10**(7):2001–2010, 2023.

[Lee74] Wai-Hon Lee. "Binary synthetic holograms." *Applied optics*, **13**(7):1677–1682, 1974.

[LG22] Shurui Li and Puneet Gupta. "Bit-serial Weight Pools: Compression and arbitrary precision execution of neural networks on resource constrained processors." *Proceedings of Machine Learning and Systems*, **4**:238–250, 2022.

[LHC22] Juzheng Liu, Mohsen Hassanpourghadi, and Mike Shuo-Wei Chen. "A 10GS/s 8b 25fJ/c-s 2850um¡sup¿2¡/sup¿ Two-Step Time-Domain ADC Using Delay-Tracking Pipelined-SAR TDC with 500fs Time Step in 14nm CMOS Technology." In *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, volume 65, pp. 160–162, 2022.

[LKK01] Jae-Soo Lee, Jung-Hwan Ko, and Eun-Soo Kim. "Real-time stereo object tracking system by using block matching algorithm and optical binary phase extraction joint transform correlator." *Optics Communications*, **191**(3-6):191–202, 2001.

[LKK19] Jinmook Lee, Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sangyeob Kim, and Hoi-Jun Yoo. "UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision." *IEEE Journal of Solid-State Circuits*, **54**(1):173–185, 2019.

[LKL21] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, et al. "Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product." In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 43–56. IEEE, 2021.

[LLK22] Yuan Li, Ahmed Louri, and Avinash Karanth. "SPACX: Silicon Photonics-based Scalable Chiplet Accelerator for DNN Inference." In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 831–845, 2022.

[LLY19] Weichen Liu, Wenyang Liu, Yichen Ye, Qian Lou, Yiyuan Xie, and Lei Jiang. "Holylight: A nanophotonic accelerator for deep learning in data centers." In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1483–1488. IEEE, 2019.

[LNM17] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. "DRISA: A DRAM-based Reconfigurable In-Situ Accelerator." In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 288–301, 2017.

[LRG21] Shurui Li, Wojciech Romaszkan, Alexander Graening, and Puneet Gupta. "SWIS–Shared Weight bIt Sparsity for Efficient Neural Network Acceleration." *arXiv preprint arXiv:2103.01308*, 2021.

[LRY18] Xing Lin, Yair Rivenson, Nezih T Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. "All-optical machine learning using diffractive deep neural networks." *Science*, **361**(6406):1004–1008, 2018.

[LSC18] Liangzhen Lai, Naveen Suda, and Vikas Chandra. "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus." *arXiv preprint arXiv:1801.06601*, 2018.

[LSZ15] Zhaohui Li, Ivan Shubin, and Xiang Zhou. "Optical interconnects: recent advances and future challenges." *Optics express*, **23**(3):3717–3720, 2015.

[Lug74] AV Lugt. "Coherent optical processing." *Proceedings of the IEEE*, **62**(10):1300–1319, 1974.

[LYW23a] Shurui Li, Hangbo Yang, Chee Wei Wong, Volker J Sorger, and Puneet Gupta. "Photofourier: A photonic joint transform correlator-based neural network accelerator." In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 15–28. IEEE, 2023.

[LYW23b] Shurui Li, Hangbo Yang, Chee Wei Wong, Volker J Sorger, and Puneet Gupta. "ReFOCUS: Reusing Light for Efficient Fourier Optics-Based Photonic Neural

Network Accelerator." In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 569–583, 2023.

[LZL21] Chong Li, Xiang Zhang, Jingwei Li, Tao Fang, and Xiaowen Dong. "The challenges of modern computing and new opportunities for optics." *PhotoniX*, **2**:1–31, 2021.

[Mac92] Ludwig Mach. "Ueber einen interferenzrefraktor." *Zeitschrift für Instrumentenkunde*, **12**(3):89, 1892.

[MAS18] Armin Mehrabian, Yousra Al-Kabani, Volker J Sorger, and Tarek El-Ghazawi. "PCNNA: a photonic convolutional neural network accelerator." In *2018 31st IEEE International System-on-Chip Conference (SOCC)*, pp. 169–173. IEEE, 2018.

[MBJ09] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. "CACTI 6.0: A tool to model large caches." *HP laboratories*, **27**:28, 2009.

[Mea] Meadowlark Optics. "1920slm." https://www.meadowlark.com/1920-1152-spatial-light-modulator-p-119.

[MHL20] Mario Miscuglio, Zibo Hu, Shurui Li, Jonathan K. George, Roberto Capanna, Hamed Dalir, Philippe M. Bardet, Puneet Gupta, and Volker J. Sorger. "Massively parallel amplitude-only Fourier neural network." *Optica*, **7**(12):1812–1819, Dec 2020.

[MLW17] Sajjad Moazeni, Sen Lin, Mark Wade, Luca Alloatti, Rajeev J Ram, Miloš Popović, and Vladimir Stojanović. "A 40-Gb/s PAM-4 transmitter based on a ring-resonator optical DAC in 45-nm SOI CMOS." *IEEE Journal of Solid-State Circuits*, **52**(12):3503–3516, 2017.

[MMC13] Mohammad Mirhosseini, Omar S Magana-Loaiza, Changchen Chen, Brandon Rodenburg, Mehul Malik, and Robert W Boyd. "Rapid generation of light beams carrying orbital angular momentum." *Optics express*, **21**(25):30196–30203, 2013.

[MMH18] Mario Miscuglio, Armin Mehrabian, Zibo Hu, Shaimaa I. Azzam, Jonathan George, Alexander V. Kildishev, Matthew Pelton, and Volker J. Sorger. "All-optical nonlinear activation function for photonic neural networks]." *Opt. Mater. Express*, **8**(12):3851–3863, Dec 2018.

[Mur] Boris Murmann. "ADC Performance Survey 1997-2022." Available online: http://web.stanford.edu/ murmann/adcsurvey.html.

[NDT19] Mitchell A Nahmias, Thomas Ferreira De Lima, Alexander N Tait, Hsuan-Tung Peng, Bhavin J Shastri, and Paul R Prucnal. "Photonic multiply-accumulate operations for neural networks." *IEEE Journal of Selected Topics in Quantum Electronics*, **26**(1):1–18, 2019.

[NH92] Steven J Nowlan and Geoffrey E Hinton. "Simplifying neural networks by soft weight-sharing." *Neural computation*, **4**(4):473–493, 1992.

[NJ00] Takanori Nomura and Bahram Javidi. "Optical encryption using a joint transform correlator architecture." *Optical Engineering*, **39**(8):2031–2035, 2000.

[NON15] Andres D Neira, Nicolas Olivier, Mazhar E Nasir, Wayne Dickson, Gregory A Wurtz, and Anatoly V Zayats. "Eliminating material constraints for nonlinearity with plasmonic metamaterials." *Nature communications*, **6**(1):1–8, 2015.

[OAN15] E. Olieman, A. Annema, and B. Nauta. "An Interleaved Full Nyquist High-Speed DAC Technique." *IEEE Journal of Solid-State Circuits*, **50**(3):704–713, 2015.

[OCL17] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. "Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems." In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 41–54, 2017.

[PCS97] Elisabet Pérez, Katarzyna Chałasińska-Macukow, Krzysztof Styczyński, Rafał Kotyński, and Maria Sagrario Millán. "Dual nonlinear correlator based on computer controlled joint transform processor: digital analysis and optical results." *Journal of Modern Optics*, **44**(8):1535–1552, 1997.

[Phaa] Phantom. "Flex4k Camera." https://www.phantomhighspeed.com/products/cameras/4kmedi

[Phab] Phantom. "S990 Camera." https://www.phantomhighspeed.com/products/cameras/machinev

[PHS19] Cheng Peng, Ryan Hamerly, Mohammad Soltani, and Dirk R. Englund. "Design of high-speed phase-only spatial light modulators with two-dimensional tunable microcavity arrays." *Opt. Express*, **27**(21):30669–30680, Oct 2019.

[PMY23] Nicola Peserico, Jiawei Meng, Hangbo Yang, Xiaoxuan Ma, Shurui Li, Hamed Dalir, Puneet Gupta, Chee Wei Wong, and Volker J Sorger. "Design and testing of silicon photonic 4F system for convolutional neural networks." In *Integrated Optics: Devices, Materials, and Technologies XXVII*, volume 12424, pp. 112–121. SPIE, 2023.

[PRM17] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. "SCNN: An accelerator for compressed-sparse convolutional neural networks." *ACM SIGARCH computer architecture news*, **45**(2):27–40, 2017.

[RBH09] Edgar Rueda, John F Barrera, Rodrigo Henao, and Roberto Torroba. "Optical encryption with a reference wave in a joint transform correlator architecture." *Optics communications*, **282**(16):3243–3249, 2009.

[RCK20] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. "Mlperf inference benchmark." In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 446–459. IEEE, 2020.

[RDG16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[RDM98] Ben Rodriguez, Alfredo Dubra, Carlos Martinez, Gonzalo Escuder, and Jose A Ferrari. "Nonlinear joint transform correlator using one-dimensional Fourier transformation." *Optical Engineering*, **37**(10):2742–2747, 1998.

[RLG20] Wojciech Romaszkan, Tianmu Li, and Puneet Gupta. "3PXNet: Pruned-Permuted-Packed XNOR Networks for Edge Machine Learning." *ACM Transactions on Embedded Computing Systems (TECS)*, **19**(1):1–23, 2020.

[RMN20] Michal Rakowski, Colleen Meagher, Karen Nummy, Abdelsalam Aboketaf, Javier Ayala, Yusheng Bian, Brendan Harris, Kate Mclean, Kevin McStay, Asli Sahin, Louis Medina, Bo Peng, Zoey Sowinski, Andy Stricker, Thomas Houghton, Crystal Hedges, Ken Giewont, Ajey Jacob, Ted Letavic, Dave Riggs, Anthony Yu, and John Pellerin. "45nm CMOS — Silicon Photonics Monolithic Technology (45CLO) for Next-Generation, Low Power and High Speed Optical Interconnects." In *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, pp. 1–3, 2020.

[Sam94] Jeffrey B. Sampsell. "Digital micromirror device and its application to projection displays." In *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures*, volume 12, pp. 3242–3246, Nov 1994.

[SB17] Aaron Stillmaker and Bevan Baas. "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm." *Integration*, **58**:74–81, 2017.

[SCB20] Purab Ranjan Sutradhar, Mark Connolly, Sathwika Bavikadi, Sai Manoj Pudukotai Dinakarrao, Mark A Indovina, and Amlan Ganguly. "pPIM: A programmable processor-in-memory architecture with precision-scaling for deep learning." *IEEE Computer Architecture Letters*, **19**(2):118–121, 2020.

[SCV19] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney,

Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler. "Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture." In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, p. 14–27, New York, NY, USA, 2019. Association for Computing Machinery.

[SDD14] Jan-Uwe Schmidt, Ulrike A. Dauderstaedt, Peter Duerr, Martin Friedrichs, Thomas Hughes, Thomas Ludewig, Dirk Rudloff, Tino Schwaten, Daniela Trenkler, Michael Wagner, Ingo Wullinger, Andreas Bergstrom, Peter Bjoernangen, Fredrik Jonsson, Tord Karlin, Peter Ronnholm, and Torbjorn Sandstrom. "High-speed one-dimensional spatial light modulator for Laser Direct Imaging and other patterning applications ." In Wibool Piyawattanametha and Yong-Hwa Park, editors, *MOEMS and Miniaturized Systems XIII*, volume 8977, pp. 167 – 176. International Society for Optics and Photonics, SPIE, 2014.

[SFZ18] Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, and Mickey Aleksic. "A Quantization-Friendly Separable Convolution for MobileNets." In *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pp. 14–18, 2018.

[SHS17] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, and Dirk Englund. "Deep learning with coherent nanophotonic circuits." *Nature Photonics*, **11**(7):441–446, 2017.

[SHZ18] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "Mobilenetv2: Inverted residuals and linear bottlenecks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[SKB21] Kyle Shiflett, Avinash Karanth, Razvan Bunescu, and Ahmed Louri. "Albireo: Energy-efficient acceleration of convolutional neural networks via silicon photonics." In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 860–873. IEEE, 2021.

[SLM21] Meer Sakib, Peicheng Liao, Chaoxuan Ma, Ranjeet Kumar, Duanni Huang, Guan-Lin Su, Xinru Wu, Saeed Fathololoumi, and Haisheng Rong. "A high-speed micro-ring modulator for next generation energy-efficient optical networks beyond 100 Gbaud." In *Conference on Lasers and Electro-Optics*, p. SF1C.3. Optica Publishing Group, 2021.

[SMN21] Febin Sunny, Asif Mirza, Mahdi Nikdast, and Sudeep Pasricha. "CrossLight: A Cross-Layer Optimized Silicon Photonic Neural Network Accelerator." In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1069–1074, 2021.

[SNL18] Sanghyun Son, Seungjun Nah, and Kyoung Mu Lee. "Clustering convolutional kernels to compress deep neural networks." In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 216–232, 2018.

[SNM16] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars." *ACM SIGARCH Computer Architecture News*, **44**(3):14–26, 2016.

[Spea] Specialized Imaging. "Kirana7M Camera." https://www.specialised-imaging.com/products/video-cameras/kirana.

[Speb] Specialized Imaging. "SIMX Camera." https://www.specialised-imaging.com/products/framing-cameras/simx.

[SPS18] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network." In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 764–775. IEEE, 2018.

[STN21] Febin P Sunny, Ebadollah Taheri, Mahdi Nikdast, and Sudeep Pasricha. "A survey on silicon photonics for deep learning." *ACM Journal of Emerging Technologies in Computing System*, **17**(4):1–57, 2021.

[SWK20] Kyle Shiflett, Dylan Wright, Avinash Karanth, and Ahmed Louri. "PIXEL: Photonic neural network accelerator." In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 474–487. IEEE, 2020.

[SWL14] Alireza Shafaei, Yanzhi Wang, Xue Lin, and Massoud Pedram. "FinCACTI: Architectural analysis and modeling of caches with deeply-scaled FinFET devices." In *2014 IEEE Computer Society Annual Symposium on VLSI*, pp. 290–295. IEEE, 2014.

[SZ14] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556*, 2014.

[SZW18] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. "Scale-sim: Systolic cnn accelerator simulator." *arXiv preprint arXiv:1811.02883*, 2018.

[TCD21] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. "Training data-efficient image transformers & distillation through attention." In *International conference on machine learning*, pp. 10347–10357. PMLR, 2021.

[TDZ17] Alexander N Tait, Thomas Ferreira De Lima, Ellen Zhou, Allie X Wu, Mitchell A Nahmias, Bhavin J Shastri, and Paul R Prucnal. "Neuromorphic photonic networks using silicon photonic weight banks." *Scientific reports*, **7**(1):1–10, 2017.

[Texa] Texas Instruments. "DLP470TE datasheet." https://www.ti.com/document-viewer/DLP470TE/datasheet/features-dlps1611610.

[Texb] Texas Instruments. "DLP9500 datasheet." https://www.ti.com/document-viewer/DLP9500/datasheet.

[TFG90] Eddy Chi-Poon Tam, TS Francis, Don A Gregory, and Richard D Juday. "Autonomous real-time object tracking with an adaptive joint transform correlator." *Optical Engineering*, **29**(4):314–320, 1990.

[TL19] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

[tpu23] "Google Cloud TPU.", 2023.

[TS98] Renu Tripathi and Kehar Singh. "Pattern discrimination using a bank of wavelet filters in a joint transform correlator." *Optical Engineering*, **37**(2):532–538, 1998.

[UJ17] Yaman Umuroglu and Magnus Jahre. "Streamlined deployment for quantized neural networks." *arXiv preprint arXiv:1709.04060*, 2017.

[UMW17] Karen Ullrich, Edward Meeds, and Max Welling. "Soft weight-sharing for neural network compression." *arXiv preprint arXiv:1702.04008*, 2017.

[UOO11] Erdem Ulusoy, Levent Onural, and Haldun M Ozaktas. "Full-complex amplitude modulation with binary spatial light modulators." *JOSA A*, **28**(11):2310–2321, 2011.

[vis23] "ImageNet Benchmark.", 2023.

[VLB18] Adam Van Etten, Dave Lindenbaum, and Todd M Bacastow. "Spacenet: A remote sensing dataset and challenge series." *arXiv preprint arXiv:1807.01232*, 2018.

[VMP13] Juan M Vilardy, María S Millán, and Elisabet Pérez-Cabré. "Joint transform correlator-based encryption system using the Fresnel transform and nonlinear filtering." In *8th Iberoamerican Optics Meeting and 11th Latin American Meeting on Optics, Lasers, and Applications*, volume 8785, pp. 429–435. SPIE, 2013.

[VMP20] Juan M Vilardy, María S Millán, and Elisabet Pérez-Cabré. "Experimental optical encryption scheme for the double random phase encoding using a nonlinear joint transform correlator." *Optik*, **217**:164653, 2020.

[Wal99] Robert H Walden. "Analog-to-digital converter survey and analysis." *IEEE Journal on selected areas in communications*, **17**(4):539–550, 1999.

[WCT15] Liang Wu, Shubo Cheng, and Shaohua Tao. "Simultaneous shaping of amplitude and phase of light in the entire output plane with a phase-only hologram." *Scientific reports*, **5**:15426, 2015.

[WG66] CS Weaver and Joseph W Goodman. "A technique for optically convolving two functions." *Applied optics*, **5**(7):1248–1249, 1966.

[WHM20] I. A. D. Williamson, T. W. Hughes, M. Minkov, B. Bartlett, S. Pai, and S. Fan. "Reprogrammable Electro-Optic Nonlinear Activation Functions for Optical Neural Networks." *IEEE Journal of Selected Topics in Quantum Electronics*, **26**(1):1–12, 2020.

[WM19] Kelvin H Wagner and Sean McComb. "Optical rectifying linear units for backpropagation learning in a deep holographic convolutional neural network." *IEEE Journal of Selected Topics in Quantum Electronics*, **26**(1):1–18, 2019.

[WT98] David C Weber and James D Trolinger. "Novel implementation of nonlinear joint transform correlators in optical security and validation." In *Algorithms, Devices, and Systems for Optical Information Processing II*, volume 3466, pp. 290–300. SPIE, 1998.

[WTL21] Qizhe Wu, Linfeng Tao, Huawen Liang, Wei Yuan, Teng Tian, Shuang Xue, and Xi Jin. "Software-Hardware Co-Optimization on Partial-Sum Problem for PIM-based Neural Network Accelerator." In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7, 2021.

[WWL19] Qiwen Wang, Xinxin Wang, Seung Hwan Lee, Fan-Hsuan Meng, and Wei D. Lu. "A Deep Neural Network Accelerator Based on Tiled RRAM Architecture." In *2019 IEEE International Electron Devices Meeting (IEDM)*, pp. 14.4.1–14.4.4, 2019.

[WWW18] Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. "Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions." In *International Conference on Machine Learning*, pp. 5363–5372. PMLR, 2018.

[WXC21] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. "Cvt: Introducing convolutions to vision transformers." In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 22–31, 2021.

[XJ23] Xiao-Yun Xu and Xian-Min Jin. "Integrated photonic computing beyond the von neumann architecture." *ACS Photonics*, **10**(4):1027–1036, 2023.

[XRL14] Li Xu, Jimmy SJ Ren, Ce Liu, and Jiaya Jia. "Deep Convolutional Neural Network for Image Deconvolution." In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pp. 1790–1798. Curran Associates, Inc., 2014.

[XWN18] Xingyuan Xu, Jiayang Wu, Thach G. Nguyen, Tania Moein, Sai T. Chu, Brent E. Little, Roberto Morandotti, Arnan Mitchell, and David J. Moss. "Photonic microwave true time delays for phased array antennas using a 49 GHz FSR integrated optical micro-comb source." *Photon. Res.*, **6**(5):B30–B36, May 2018.

[YGL21] Kun Yuan, Shaopeng Guo, Ziwei Liu, Aojun Zhou, Fengwei Yu, and Wei Wu. "Incorporating convolution designs into visual transformers." In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 579–588, 2021.

[YLC19] Geoffrey Yeap, S. S. Lin, Y. M. Chen, H. L. Shang, P. W. Wang, H. C. Lin, Y. C. Peng, J. Y. Sheu, M. Wang, X. Chen, B. R. Yang, C. P. Lin, F. C. Yang, Y. K. Leung, D. W. Lin, C. P. Chen, K. F. Yu, D. H. Chen, C. Y. Chang, H. K. Chen, P. Hung, C. S. Hou, Y. K. Cheng, J. Chang, L. Yuan, C. K. Lin, C. C. Chen, Y. C. Yeo, M. H. Tsai, H. T. Lin, C. O. Chui, K. B. Huang, W. Chang, H. J. Lin, K. W. Chen, R. Chen, S. H. Sun, Q. Fu, H. T. Yang, H. T. Chiang, C. C. Yeh, T. L. Lee, C. H. Wang, S. L. Shue, C. W. Wu, R. Lu, W. R. Lin, J. Wu, F. Lai, Y. H. Wu, B. Z. Tien, Y. C. Huang, L. C. Lu, Jun He, Y. Ku, J. Lin, M. Cao, T. S. Chang, and S. M. Jang. "5nm CMOS Production Technology Platform featuring full-fledged EUV, and High Mobility Channel FinFETs with densest 0.021µm¡sup¿2¡/sup¿ SRAM cells for Mobile SoC and High Performance Computing Applications." In *2019 IEEE International Electron Devices Meeting (IEDM)*, pp. 36.7.1–36.7.4, 2019.

[YLD19] Jiecao Yu, Andrew Lukefahr, Reetuparna Das, and Scott Mahlke. "TF-net: Deploying sub-byte deep neural networks on microcontrollers." *ACM Transactions on Embedded Computing Systems (TECS)*, **18**(5s):1–21, 2019.

[YLM22] Hangbo Yang, Shurui Li, Xiaoxuan Ma, Jonathan K. George, Puneet Gupta, Volker J. Sorger, and Chee Wei Wong. "Programmable On-chip Photonic Machine Learning System Based on Joint Transform Correlator." In *Conference on Lasers and Electro-Optics*, p. JW3B.19. Optica Publishing Group, 2022.

[YW21] Stone Yun and Alexander Wong. "Do All MobileNets Quantize Poorly? Gaining Insights Into the Effect of Quantization on Depthwise Separable Convolutional Networks Through the Eyes of Multi-Scale Distributional Dynamics." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 2447–2456, June 2021.

[YZL17] Q. Ye, Y. Zhang, X. Li, and Y. Zhang. "A 12-bit 10GSps ultra high speed DAC in InP HBT technology." In *2017 2nd IEEE International Conference on Integrated Circuits and Microsystems (ICICM)*, pp. 9–13, 2017.

[ZLY20] Farzaneh Zokaee, Qian Lou, Nathan Youngblood, Weichen Liu, Yiyuan Xie, and Lei Jiang. "LightBulb: A photonic-nonvolatile-memory-based accelerator for binarized convolutional neural networks." In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1438–1443. IEEE, 2020.

[ZLZ19] Yu Zhang, Yi-Chun Ling, Kaiqi Zhang, Cale Gentry, David Sadighi, Greg Whaley, James Colosimo, Paul Suni, and SJ Ben Yoo. "Sub-wavelength-pitch silicon-photonic optical phased array for large field-of-regard coherent optical beam steering." *Optics express*, **27**(3):1929–1940, 2019.

[ZVS20] Brian Zimmer, Rangharajan Venkatesan, Yakun Sophia Shao, Jason Clemons, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel S. Emer, C. Thomas Gray, Stephen W. Keckler, and Brucek Khailany. "A 0.32–128 TOPS, Scalable Multi-Chip-Module-Based Deep Neural Network Inference Accelerator With Ground-Referenced Signaling in 16 nm." *IEEE Journal of Solid-State Circuits*, **55**(4):920–932, 2020.

[ZW14] Long Zhu and Jian Wang. "Arbitrary manipulation of spatial amplitude and phase using phase-only spatial light modulators." *Scientific reports*, **4**:7441, 2014.

[ZYL13] Yi Zhang, Shuyu Yang, Andy Eu-Jin Lim, Guo-Qiang Lo, Christophe Galland, Tom Baehr-Jones, and Michael Hochberg. "A compact and low loss Y-junction for submicron silicon waveguide." *Optics express*, **21**(1):1310–1316, 2013.