Throughput-Optimized Frequency Domain CNN with Fixed-Point Quantization on FPGA

Weiyi Sun
Department of Microelectronics and Nanoelectronics
Tsinghua University
swy15@mails.tsinghua.edu.cn

Hanqing Zeng
Ming Hsieh Departmnet of Electrical Engineering
University of Southern California
zengh@usc.edu

Yi-hua Edward Yang
Machine Intelligence Technology
Alibaba Group
edward.yang@alibaba-inc.com

Viktor Prasanna
Ming Hsieh Department of Electrical Engineering
University of Southern California
prasanna@usc.edu

***Abstract*—State-of-the-art hardware accelerators for large scale CNNs face two challenges: high computation complexity of convolution, and high on-chip memory consumption by weight kernels. Two techniques have been proposed in the literature to address these challenges: frequency domain convolution and space domain fixed-point quantization. In this paper, we propose frequency domain quantization schemes to achieve high throughput CNN inference on FPGAs. We first analyze the impact of quantization bit width on the accuracy of a frequency domain CNN, via the metric of Signal-to-Quantization-Noise-Ratio (SQNR). Taking advantage of the reconfigurability of FPGAs, we design a statically-reconfigurable and a dynamically-reconfigurable architecture for the quantized convolutional layers. Then, based on the SQNR analysis, we propose quantization schemes for both types of architectures, achieving optimal tradeoff between throughput and accuracy. The proposed quantizer allocates the number of bits for each convolutional layer under various design constraints, including overall SQNR, available DSP resources, on-chip memory and off-chip bandwidth. Experiments on AlexNet show that our designs improve the CNN inference throughput by $1.45\times$ to $8.44\times$, with negligible ($< 0.5\%$) loss in accuracy.**

***Index Terms*—Convolutional Neural Networks; FPGA; Fixed-Point Quantization; Frequency Domain Convolution;**

I. Introduction

Convolutional deep neural networks (CNNs) have shown outstanding performance in non-trivial machine learning problems such as image classification and object detection. However, the improved performance of state-of-the-art CNNs comes at the cost of higher computation complexity and hardware resource consumption. These expenses hinder the application of CNNs in real-time embedded systems, where both computation time and memory size are strictly constrained.

Two approaches have been developed to lower the barrier of deploying CNNs on embedded hardware: (1) reduce the computation complexity of a CNN through the use of frequency-domain convolution [1]; and (2) reduce both computation complexity and memory size of a CNN with fixed-point quantization [2]. While individually each approach has been shown to improve the computation and memory efficiency of CNNs, few efforts have been made to combine the strengths of both in a single framework. On the other hand, we observe that both approaches are not mutually exclusive. In fact, the increased memory usage of a frequency-domain CNN (by the FFT and IFFT matrices) makes the optimization of its fixed-point quantization an even more critical issue.

Motivated by the observations above, we adapt the fixed-point quantization method proposed in [2] and utilize it to enhance the convolution throughput of the frequency-domain CNN architecture in [1]. We analyze the effect of quantization on the frequency domain CNNs in terms of both increased Signal-to-Quantization-Nose-Ratio (SQNR) and reduced resource utilization. The analysis allows us to represent both the quantization error and the inference throughput as functions of quantization bitwidths across all convolutional layers. We then solve for the maximum throughput of the CNN, given constraints on SQNR and available resource, and derive optimal bitwidth allocations and design parameters for the fixed-point quantized CNN architecture in [1]. Finally, we verify the predicted accuracy-throughput improvements with experimental results from actual CNN designs.

The main contributions of this work are:

- We propose a fixed-point quantization scheme for frequency domain CNNs on FPGAs. The scheme allocates bit widths optimally to various convolutional layers, to optimize throughput under the constraints on SQNR loss, available hardware resources, and dynamic reconfigurability.
- We quantitatively analyze the noise introduced by the hardware Fast Fourier Transform (FFT) module, and propose a design that uses minimal additional hardware resources to achieve little SQNR loss when performing FFT.
- We tailor the accelerator in [1] design for the proposed data quantization scheme. The accelerator includes high

This work was supported by the US NSF under grants CNS-1643351 and ACI-1339756. Equipment grant from Intel Hardware Accelerator Research Program is gratefully acknowledged.

TABLE I
MODEL PARAMETERS FOR THE 5 CONVOLUTIONAL LAYERS OF ALEXNET

Layer i	Image size $l_{\text{img}}^{(i)}$	Kernel size $l_{\text{kern}}^{(i)}$	Input channels $f_{\text{in}}^{(i)}$	Output channels $f_{\text{out}}^{(i)}$
1	227	11	3	96
2	28	5	96	256
3	13	3	256	384
4	13	3	384	384
5	13	3	384	128

precision FFT and flexible Hadamard product modules that work for various bit widths.

- We implement the proposed accelerator on Intel Stratix-V. For AlexNet, our design achieves throughputs 1.45x to 8.44x of state-of-the-art designs, with negligible increase in classification error ($< 0.5\%$).

II. BACKGROUND

A. Convolutional Neural Networks

In general, a CNN consists of multiple *convolutional layers*, each optionally followed by a *pooling layer* and an *activation layer* of simple non-linear functions. The outputs of the convolutional layers are then fed into subsequent *fully connected layers* to complete the classification procedure. Among the various operations performed by a CNN, convolution dominantly contributes to most of the computation workload. Therefore, in this work, we focus on accelerator optimization for convolutional layers by applying fixed-point quantization.

We select AlexNet [3] as a representative of the various state-of-the-art large-scale CNNs [4], [3], [5]. Table I summarizes the model parameters for the 5 convolutional layers of AlexNet. Convolutional layer i is defined by 4 parameters: image size $l_{\text{img}}^{(i)}$, kernel size $l_{\text{kern}}^{(i)}$, number of input channels $f_{\text{in}}^{(i)}$, and number of output channels $f_{\text{out}}^{(i)}$ (where $f_{\text{out}}^{(i)} = f_{\text{in}}^{(i+1)}$). During inference, layer i takes as input an image with $f_{\text{in}}^{(i)}$ channels. Each channel contains a matrix of width and height $l_{\text{img}}^{(i)}$. The input image is convolved with $f_{\text{in}}^{(i)} \times f_{\text{out}}^{(i)}$ number of kernel filters. Each filter is of width and height $l_{\text{kern}}^{(i)}$. The output of layer i is the filtered image with $f_{\text{out}}^{(i)}$ channels, where each channel consists of a $l_{\text{img}}^{(i)} \times l_{\text{img}}^{(i)}$ matrix. A convolutional layer has two optional parameters for kernel striding and image padding. These two parameters are not involved in this paper.

B. Frequency Domain CNN Accelerator

The biggest challenge in accelerating CNN inference lies in the high computation complexity of spatial convolution. It is known that performing convolution in frequency domain largely reduces the number of operations of a CNN [6]. Recently, authors in [1] propose a high throughput accelerator optimized for frequency domain CNNs, which, for AlexNet and VGG16, achieves about $5\times$ higher throughput compared with other state-of-the-art designs using spatial convolution. The design and analysis in [1] form the basis of our work.

The innovations of [1] lie in various data tiling and partitioning techniques that greatly improve hardware efficiency.

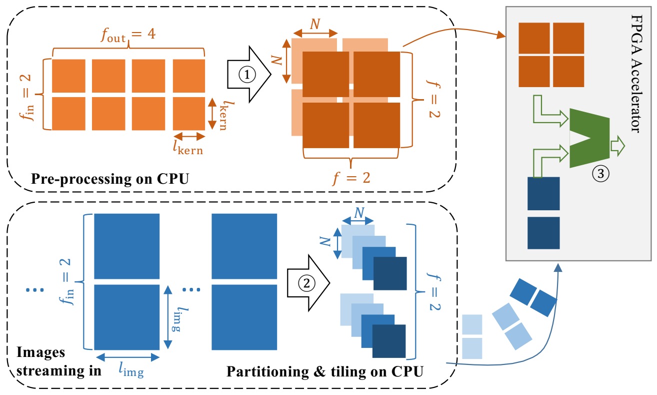


Fig. 1. Illustration of the design in [1]

The design involves collaboration between a CPU and an FPGA, which is illustrated in Figure 1. Suppose we would like to compute convolutional layer i using $N^{(i)}$-point FFT in hardware. First, the CPU performs pre-processing and light-weight data layout rearrangements to prepare data blocks for the FPGA. As shown by Step 1 in Figure 1, CPU computes $N^{(i)}$-point 2D FFT on all of the $f_{\text{in}}^{(i)} \times f_{\text{out}}^{(i)}$ kernel filters, and then performs tiling along the $f_{\text{in}}^{(i)}$, $f_{\text{out}}^{(i)}$ dimensions. The $f_{\text{in}}^{(i)} \times f_{\text{out}}^{(i)} \times l_{\text{kern}}^{(i)} \times l_{\text{kern}}^{(i)}$ kernel data then becomes tiles of $f^{(i)} \times f^{(i)} \times N^{(i)} \times N^{(i)}$ data blocks, ready to be loaded onto FPGA. Note that although the Fourier transform on kernels is computationally intensive, this is only a pre-processing step due to the immutability of kernels during inference. Afterwards, to process images of shape $f_{\text{in}}^{(i)} \times l_{\text{img}}^{(i)} \times l_{\text{img}}^{(i)}$ that are pipelined into the system, the CPU performs partitioning and tiling of the images (Step 2). The partitioning step uses the Overlap-and-Add (OaA) technique [7], [8], and the tiling on images is along the $f_{\text{in}}^{(i)}$ dimension. Step 2 outputs image data blocks of shape $f^{(i)} \times N^{(i)} \times N^{(i)}$, which are continuously loaded onto FPGA via double buffering. Steps 1 and 2 transform kernels and images of various $f_{\text{in}}^{(i)}$, $f_{\text{out}}^{(i)}$, $l_{\text{img}}^{(i)}$, $l_{\text{kern}}^{(i)}$ parameters into data blocks whose shape exploits most of the FPGA processing efficiency. Therefore, the FPGA computes frequency domain convolution with high throughput (Step 3).

The illustration above involves two design parameters: the layer i FFT size $N^{(i)}$ and the tiling factor $f^{(i)}$. It is shown in [1] that, a throughput-optimal design should exhaust all types of available resources on board (external bandwidth B, logic/DSP resources L and on-chip memory M). Therefore, on a target device with constant amount of resources B, L, M, given a choice of $N^{(i)}$, the value of $f^{(i)}$ is correspondingly selected. Thus, $N^{(i)}$ is the only independent design parameter to be tuned based on the target FPGA device and CNN model.

Throughput of the system depends on $f_{\text{in}}^{(i)}$, $f_{\text{out}}^{(i)}$, $l_{\text{img}}^{(i)}$ and $l_{\text{kern}}^{(i)}$ of the CNN, B, L and M of the target FPGA, and the design parameter $N^{(i)}$. Specifically, the system throughput of layer i is bottlenecked by either the computation bound $T_{\text{comp}}^{(i)}$ or the communication bound $T_{\text{comm}}^{(i)}$, such that $T_{\text{sys}}^{(i)} =$

$\min\{T_{\text{comp}}^{(i)}, T_{\text{comm}}^{(i)}\}$, where [1]:

$$T_{\text{comp}}^{(i)} = \frac{1}{K_{\text{CNN}}^{(i)}} \cdot L \cdot \left(\frac{N^{(i)} - l_{\text{kern}}^{(i)} + 1}{N^{(i)}}\right)^2 \tag{1}$$

$$T_{\text{comm}}^{(i)} = \frac{1}{K_{\text{CNN}}^{(i)}} \cdot \frac{B\sqrt{M}}{2N^{(i)}} \cdot \left(\frac{N^{(i)} - l_{\text{kern}}^{(i)} + 1}{N^{(i)}}\right)^2 \tag{2}$$

and $K_{\text{CNN}}^{(i)}$ is a CNN model constant defined as:

$$K_{\text{CNN}}^{(i)} = f_{\text{in}}^{(i)} \cdot f_{\text{out}}^{(i)} \cdot \left(l_{\text{img}}^{(i)} + l_{\text{kern}}^{(i)} - 1\right)^2 \tag{3}$$

C. Fixed-Point Quantization on CNNs

Most of the state-of-the-art CNN models contain tens of millions of parameters that cannot fit on-chip. Thus, it is necessary to compress the model parameters to boost the performance of FPGA devices. Quantization of kernel filters and input images is an efficient way to achieve significant compression. The work in [2] proposes a theoretical framework to analyze the overall Signal-to-Quantization-Noise-Ratio (SQNR) of the quantized CNNs. They also discuss a strategy to allocate bit width according to the number of model parameters of each convolutional layer. Their bit width allocation scheme achieves optimal model compression rate under a given budget of SQNR. The work in [9] proposes another quantization algorithm that applies to both convolutional layers and fully connected layers. Their target platform, however, is general purpose processors rather than application specific accelerators. Therefore, their quantization scheme is sub-optimal if applied directly to FPGAs. The work in [10] proposes an FPGA-accelerator designed for quantized CNNs. Although their accelerator is optimized for computing quantized CNN, their quantization scheme is empirical and may not apply to a variety of CNN models.

It is worth noticing that the above 3 projects all focus on quantizing layers using spatial convolution. To the best of our knowledge, our work is the first in the literature to study the effect of quantization on frequency domain convolution.

III. Problem Definition

Given a multi-layer CNN and a target FPGA device with fixed amount of resources, find a bit width allocation scheme for each convolutional layer (in frequency domain) so that:

1) The Signal-to-Quantization-Noise-Ratio of the quantized CNN does not exceed a given budget; and
2) Inference throughput of the whole CNN is maximized.

There are a few points worth noticing. First of all, Signal-to-Quantization-Noise-Ratio (SQNR) is an important metric evaluating the quality of a data quantization scheme. In general, the higher the SQNR, the lesser additional wrong classification will be output by the quantized CNN [2]. Secondly, tradeoff exists to allocate different number of bits to different convolutional layers. Intuitively, from the hardware performance perspective, we would like to use less number of bits to quantize layers with more amount of model parameters, since it will yield higher compression rate. We will show in Section VI how to verify this intuition in a mathematically rigorous manner. Thirdly, for the target FPGA, we study two hardware models: the *Dynamically-Reconfigurable Model* and *Statically-Reconfigurable Model*. The Statically-Reconfigurable Model is more restrictive to our bit width allocation scheme than the Dynamically-Reconfigurable Model. We discuss both in detail in Section VI, and present the corresponding experimental evaluation in Section VII.

[1]The expression of $T_{\text{comm}}^{(i)}$ is slightly different from that in [1]. The difference is due to an extra approximation $\frac{N \quad l_{kern}+1}{N} \approx 1$ proposed in [1]. We neglect this approximation for convenience of analysis.

IV. SQNR Analysis on Frequency-Domain CNNs

A. Preliminary

Quantizer: We consider symmetric uniform quantizer, which means that the mid-level stands for value 0 and each quantization level covers the same amount of value range. A data quantization scheme is determined by: bit width, step size and dynamic range, where $Range = Stepsize \cdot 2^{Bidwidth}$.

SQNR γ: Suppose a set of values x_i is quantized to value x_i'. SQNR $\gamma := \frac{\mathbb{E}[x_i^2]}{\mathbb{E}[(x_i - x_i')^2]}$, where $\mathbb{E}$ gives the expectation.

Optimal Quantizer for Gaussian Data: Suppose that the original data follows a zero-mean Gaussian distribution. For the symmetric uniform quantizer, empirical optimal step size for a given a bit width β is show in [2]. Approximately, SQNR γ_{dB} (in the unit of Decibel) is linear with the bit width:

$$\gamma_{dB} \approx \kappa \cdot \beta \tag{4}$$

where $\gamma_{dB} = 10\log_{10}(\gamma)$ and $\kappa \approx 3$ dB/bit.

For spatial convolution, it is shown that the input images and kernel weights of each convolutional layer follows a zero-mean Gaussian distribution [2]. We show in Section IV-B that this conclusion holds for frequency domain convolution as well.

SQNR of a Linear System: Let us start from the SQNR analysis for multiplication. Given two variables a, b and their SQNR γ_a, γ_b, then SQNR γ_c of the product $c = a \cdot b$ is $\frac{1}{\gamma_c} = \frac{1}{\gamma_a} + \frac{1}{\gamma_b}$ [2]. For a Linear Time-Invariant (LTI) system $y = h(x, w) = \Sigma x_i \cdot w_i$, it follows directly that:

$$\frac{1}{\gamma_y} = \frac{1}{\gamma_x} + \frac{1}{\gamma_w} \tag{5}$$

Now we compose two LTI systems: $y = h_2\big(h_1(x, w_1), w_2\big)$. Note that the composed system is still LTI. We have:

$$\frac{1}{\gamma_y} = \frac{1}{\gamma_x} + \frac{1}{\gamma_{w_1}} + \frac{1}{\gamma_{w_2}} = \frac{1}{\gamma_x} + \frac{1}{\gamma_{\text{LTI}}} \tag{6}$$

where γ_{LTI} is the SQNR intrinsic to the composed LTI system.

We make the following observations:

1) A convolutional layer is an LTI system, and a CNN is approximately composition of multiple LTI systems (if we ignore the activation layers, as assumed in [2]).
2) SQNR γ_{LTI} depends on the distribution of weights w_i, and is independent of the number of weight parameters.

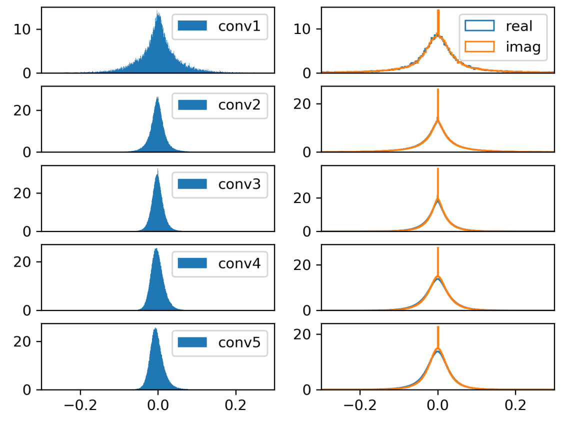


Fig. 2. Kernel value distribution of the 5 convolutional layers of AlexNet. The left 5 sub-plots show the PDF function of kernel values in space domain. The right 5 sub-plots are for the kernel values in frequency domain (real and imaginary parts of the complex numbers are plotted correspondingly).

Observation 1 forms the basis of Section IV-D. Regarding Observation 2, as an example, consider a convolutional layer 1 with $1M$ kernel weight parameters and a convolutional layer 2 with only $1K$ kernel weight parameters. If we use β bits to quantize weights for both layers, and these weights follow the same Gaussian distribution, then the resulting SQNRs of the two layers are identical. This observation directly motivates our optimization in Section VI: the amount of weight parameters of a layer affects hardware performance, not SQNR.

SQNR After Re-Quantization: Suppose high-bit-width data x with SQNR γ_1 is processed by a β_2-bit quantizer with SQNR γ_2 (calculated by Equation 4). By [2], SQNR after the two quantization is:

$$\frac{1}{\gamma_{\text{overall}}} = \frac{1}{\gamma_1} + \frac{1}{\gamma_2} \tag{7}$$

B. Value Distribution of Kernel Weights and Input Images

As discussed above in section IV.A, the accuracy of our SQNR analysis depends on the fact that the data (weights, inputs and outputs) are zero mean Gaussian distributed.

Lemma 1. *Let $X_1, \ldots, X_n$ be n jointly multivariate Gaussian distributed random variables with zero mean. Let $Y_1, \ldots, Y_n$ be outputs of an n-dimensional linear transformation on $X_1, \ldots, X_n$. Then, $Y_1, \ldots, Y_n$ are also jointly multivariate Gaussian and have zero mean. Specifically, if $X_1, \ldots, X_n$ are independent and have identical variance, and if the transformation is orthogonal, then $Y_1, \ldots, Y_n$ are also independent and have identical variance.*

Lemma 1 directly follows from Equation 7.3 in [11].

Therefore, both the space domain images and weights [2], and their frequency domain counterparts are approximately multivariate, zero-mean, Gaussian random variables.

Figure 2 shows the empirical distributions of the kernel weights in space domain as well as in frequency domain for the 5 convolutional layers of AlexNet. The distribution plots for images are omitted here due to space constraints.

Therefore, we use the optimal quantizer for Guassian data (Section IV-A) for quantization on frequency domain CNNs.

C. SQNR of a Convolutional Layer

According to Figure 1, the procedure to compute a convolutional layer in frequency domain is summarized as:

$$I^{\text{output}} = \mathcal{F}^{-1}\Big(\mathcal{F}(I^{\text{input}}) \circ K^{\text{freq}}\Big) \tag{8}$$

where K^{freq} is the pre-processed kernel data in frequency domain, I^{input} is the image data in space domain. $\mathcal{F}$ and $\mathcal{F}^{-1}$ stands for Fourier transform and its inverse. Symbol $\circ$ stands for Hadamard product (element-wise matrix multiplication).

One convolutional layer is the composition of three LTI subsystems, responsible for FFT, Hadamard product and IFFT respectively. Note that there can be various data re-quantization steps among the three sub-systems as well (Section V-A). By Equations 6 and 7, SQNR of the layer output is:

$$\frac{1}{\gamma_{\text{layer}}} = \frac{1}{\gamma_{\text{input}}} + \left(\frac{1}{\gamma_{\text{FFT}}} + \frac{1}{\gamma_{\text{Hadamard}}} + \frac{1}{\gamma_{\text{IFFT}}}\right) + \frac{1}{\gamma_{\text{re-quan}}} \tag{9}$$

Next, we derive the expressions for γ_{FFT}, γ_{Hadamard} and γ_{IFFT} respectively. We derive the expression of $\gamma_{\text{re-quan}}$ in Section V.

SQNR Analysis on 2D FFT: A N-point FFT can be decomposed into two $\frac{N}{2}$-point FFTs. Applying this idea recursively, an efficient hardware implementation of FFT involves $\log N$ computation stages [12], [13]. Each stage of the FFT computation applies constant weights called "twiddle factors". Taking into account the data re-quantization between adjacent stages, SQNR intrinsic to the FFT module is:

$$\frac{1}{\gamma_{\text{FFT}}} = \sum^{\log N - 1} \frac{1}{\gamma_{\text{re-quan}}} + \sum^{\log N} \frac{1}{\gamma_{\text{twiddle}}} \tag{10}$$

SQNR γ_{FFT} decreases as N grows. This restricts the choices of the design parameter N. To overcome this limitation, we adopt a high-precision FFT module in our design (Section V).

Also notice that Equation 10 applies to 2D FFT as well, since 2D FFT is the concatenation of a N-point 1D FFT for rows and a N-point 1D FFT for columns [1].

SQNR Analysis on Hadamard Product: Different from 2D FFT, the Hadamard product operation involves only one stage of multiplication. Thus, the SQNR of the sub-system is:

$$\gamma_{\text{Hadamard}} = \gamma_{K^{\text{freq}}} \tag{11}$$

where $\gamma_{K^{\text{freq}}}$ is the SQNR for quantizing kernel filters K^{freq}.

D. SQNR of a CNN

Using the same assumption as [2] to ignore the activation and pooling layers, a CNN is the composition of multiple convolutional layers. SQNR of a l-layer CNN follows directly:

$$\frac{1}{\gamma_{\text{CNN}}} = \left(\frac{1}{\gamma_{\text{input}}}\right)^{(1)} + \sum_{i=1}^{l}\left(\frac{1}{\gamma_{\text{FFT}}} + \frac{1}{\gamma_{\text{Hadamard}}} + \frac{1}{\gamma_{\text{IFFT}}} + \frac{1}{\gamma_{\text{re-quan}}}\right)^{(i)} \tag{12}$$

There are differences between SQNRs of spatial and frequency domain CNNs. The middle term of Equation 12 captures the SQNR intrinsic to the system (target CNN). For spatial CNNs, the intrinsic SQNR is only contributed by the kernel quantization [2]. However, for frequency domain

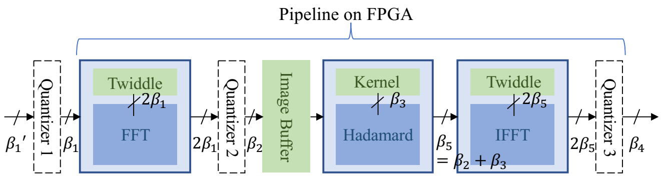


Fig. 3. Overall FPGA Architecture

CNNs, the intrinsic SQNR is due to both kernel quantization ($\gamma_{\text{Hadamard}} = \gamma_{K^{\text{freq}}}$) and the FFT process. In other words, given the same quantization bit width, frequency domain CNNs likely suffer higher quantization noise. We address this problem by an architecture carefully designed with the tradeoff of SQNR and throughput in mind (Section V-A).

V. Accelerator Design and Performance Model

A. Overall System Architecture

Figure 3 shows the overall architecture design on FPGA, which largely follows the design in [1]. The processing pipeline on FPGA performs the computation specified by Equation 8. Input image data is pre-processed by CPU as specified by Figure 1. Using the same notations as defined in Section II-B, below is a concise description of the computation procedure on FPGA, while a detailed description is presented in [1]. Assume tiling factor $f^{(i)}$ and FFT size $N^{(i)}$ for layer i. Before FPGA starts processing, a tile of $f^{(i)} \times f^{(i)} \times N^{(i)} \times N^{(i)}$ of kernel is pre-loaded into on-chip buffer of FPGA. Image tiles of shape $f^{(i)} \times N^{(i)} \times N^{(i)}$ is then sequentially fed into the FFT module, one tile at a time. The image tile on-chip is buffered so that it can be reused when the Hadamard module computes using kernel tiles and image tiles. The results of Hadamard product are processed by the IFFT module, whose outputs flow directly back to external memory. Note that to consume an image tile of $f^{(i)}$ channels, the Hadamard module needs to iterate over $f^{(i)} \times f^{(i)}$ channels of kernel data. This indicates that the workload performed by Hadamard module is much larger than that by the FFT or IFFT module. To match the throughput of the three modules on the pipeline, it is necessary to allocate much more hardware resources to the Hadamard module than to the FFT and IFFT modules. This observation is consistent with the analysis in [1]. We will compare the actual resource consumption of the three modules in our implementation in Section VII-C.

Note that FFT and IFFT modules are not resource hungry, and the bit widths of these two modules have a significant impact on the overall SQNR (Equation 10). Therefore, we use high-precision FFT/IFFT designs in our system. In other words, if the input image tile to the FPGA is represented by β-bit data, then all the internal $\log N$ stages of the FFT/IFFT pipeline use 2β data for computation. The twiddle factors are also quantized by 2β bits. Due to the large bit width within the FFT/IFFT modules, the intrinsic SQNR γ_{twiddle} and the re-quantization SQNR $\gamma_{\text{re-quan}}$ in Equation 10 are effectively orders of magnitudes lower than the other SQNR terms of the system. Therefore, as an approximation, $\frac{1}{\gamma_{\text{FFT}}} = \frac{1}{\gamma_{\text{IFFT}}} \approx 0$.

The Hadamard module performs most of the on-chip computation. Thus, it is essential to make the hardware design flexible enough to handle various bit widths efficiently. Since there is no data dependency during the Hadamard product operation, we simply design the Hadamard module as a length h array of Multiply-ACcumulate (MAC) units. To achieve higher throughput for lower bit widths, we increase the array length h. For example, for our experimental platform, $h = 128$ for 16-bit quantization and $h = 256$ for 11-bit quantization.

There are three quantizers in Figure 3. Quantizer 1 is inserted so that adjacent convolutional layers don't necessarily have the same bit width. The motivation for Quantizer 2 is that Hadamard module consumes most of the logic/DSP resources on chip. It is not resource efficient to directly compute multiplication on the $2\beta_1$ data output by FFT. Quantizer 3 helps save the resource on external bandwidth, since the output of Quantizer 3 is transferred directly to external memory. Notice that there is no quantizer between the Hadamard and IFFT modules, since the IFFT module can process $2\beta_5$-bit input data without consuming much of the total on-chip resources.

B. Performance Model

This section derives the throughput performance model for quantized frequency domain CNNs. Define β_0 as the bit width where we consider a CNN as un-quantized. For quantized bit widths, use the same notations (β_1, β_2, β_3 and β_4) as defined in Figure 3. Denote variables for layer i by superscript (i).

First, recall the performance model of the un-quantized frequency domain CNNs defined by Equation 2. Optimal design parameter $N^{(i)}$ should be chosen at the intersection point of computation and communication bounds ($T^{(i)}_{\text{comp}}$, $T^{(i)}_{\text{comm}}$):

$$N^{(i)}_{\text{opt}} = \frac{1}{2} \cdot \frac{B\sqrt{M}}{L} \tag{13}$$

To derive the performance model for quantized CNNs based on Equation 2, we define the concept of *effective resources*. Image a device 1 with resources B, L and M, executing the quantized CNN, and a device 2 with resources B_{eff}, L_{eff} and M_{eff}, executing the un-quantized CNN. The *effective resources* of B, L and M are said to be B_{eff}, L_{eff} and M_{eff}, if throughput of devices 1 equals that of device 2.

Effective resources are easily derived by noting that:

- Bandwidth and memory consumption of a data word is proportional to the bit width of the data.
- Logic or DSP consumption is proportional to the bit widths of the multiplicand and the multiplier (since multiplication consumes most of the logic/DSP resources).
- Most of the on-chip memory is used to store kernel tiles, and most of the logic/DSP resources are used to calculate Hadamard product (see Section V-A and [1] for details).

Therefore, the effective resources are:

$$B^{(i)}_{\text{eff}} = B \cdot \frac{2\beta_0}{\beta^{(i)}_1 + \beta^{(i)}_4} \tag{14}$$

$$L^{(i)}_{\text{eff}} = L \cdot \frac{\beta_0^2}{\beta^{(i)}_2 \cdot \beta^{(i)}_3} \tag{15}$$

$$M_{\text{eff}}^{(i)} = M \cdot \frac{\beta_0}{\beta_3^{(i)}} \tag{16}$$

Substituting $B_{\text{eff}}^{(i)}$, $L_{\text{eff}}^{(i)}$ and $M_{\text{eff}}^{(i)}$ into Equation 13, we get the optimal FFT size $N^{(i)}$ for quantized CNN:

$$N_{\text{opt}}^{(i)} = \frac{1}{2} \cdot \frac{B\sqrt{M}}{L} \cdot \sqrt{\frac{\beta_3^{(i)}}{\beta_0} \frac{2\beta_2^{(i)}}{\beta_1^{(i)} + \beta_4^{(i)}}} \tag{17}$$

The resulting throughput of the quantized CNN is thus:

$$T_{\text{sys,quan}}^{(i)} = \min\{T_{\text{comp,quan}}^{(i)}, T_{\text{comm,quan}}^{(i)}\} \tag{18}$$

where $T_{\text{comp,quan}}^{(i)}$ and $T_{\text{comm,quan}}^{(i)}$ is derived by directly plugging in Equation 2 the updated $B_{\text{eff}}^{(i)}$, $L_{\text{eff}}^{(i)}$ and $M_{\text{eff}}^{(i)}$.

The optimal throughput takes place at $N_{\text{opt}}^{(i)}$:

$$T_{\text{sys-opt,quan}}^{(i)} = T_{\text{comp-opt,quan}}^{(i)}(N_{\text{opt}}^{(i)}) = T_{\text{comm-opt,quan}}^{(i)}(N_{\text{opt}}^{(i)}) \tag{19}$$

VI. Throughput-Optimized Bit Width Allocation Scheme

Based on the SQNR analysis and the performance model of the quantized CNN, we can formulate an optimization problem to allocate different bit widths to different convolutional layers. Specifically, we identify the optimal $\beta_1^{(i)}$, $\beta_2^{(i)}$, $\beta_3^{(i)}$ and $\beta_4^{(i)}$ to maximize Equation 18 under the constraint of Equation 12.

A. Simplification on the Bit Width Allocation Scheme

Notice that the current bit width allocation scheme involves 4 independent variables for each layer. By simplification and approximation, we can reduce the number of independent variables without sacrificing the quality of our design.

Consider the effect of quantization on the optimal FFT size. In practice, hardware requires the FFT size to be power of 2. Therefore, the quantizer has to change its bit width significantly to round the optimal N value from the current 2^c point to the adjacent $2^{c\pm1}$ point (where c is a positive integer). In other words, quantization barely affects the value of $N_{\text{opt}}^{(i)}$ under a reasonable quantizer. On the other hand, due to the configurability of memory, logic and DSP blocks on chip, quantization does greatly affect the $B_{\text{eff}}^{(i)}$, $L_{\text{eff}}^{(i)}$, $M_{\text{eff}}^{(i)}$ values.

Therefore, to analyze the effect of quantization on $T_{\text{comp,quan}}^{(i)}$ and $T_{\text{comm,quan}}^{(i)}$, we approximately treat the common term $\frac{N_{\text{opt}}^{(i)} - l_{\text{kern}}^{(i)} + 1}{N_{\text{opt}}^{(i)}}$ as a constant under various bit widths. Additionally, note that $\beta_2^{(i)}$ and $\beta_3^{(i)}$ play a symmetric role on $L_{\text{eff}}^{(i)}$ (and thus $T_{\text{comp,quan}}^{(i)}$); $\beta_1^{(i)}$ and $\beta_4^{(i)}$ play a symmetric role on $B_{\text{eff}}^{(i)}$ (and thus $T_{\text{comm,quan}}^{(i)}$). We reasonably conclude that:

$$\beta_1^{(i)} = \beta_4^{(i)}; \qquad \beta_2^{(i)} = \beta_3^{(i)} \tag{20}$$

B. Model of the Target Hardware

We consider two models of the target FPGA:

- *Statically-Reconfigurable Model*: The on-chip memory, logic and DSP of the target FPGA cannot be reconfigured across layers. Thus, $\beta_1^{(i)} = \beta_1^{(j)}$ and $\beta_2^{(i)} = \beta_2^{(j)}$.
- *Dynamically-Reconfigurable Model*: The on-chip memory, logic and DSP are fully reconfigurable across layers. $\beta_1^{(i)}$ and $\beta_2^{(i)}$ are independent variables (no dependencies between $\beta_1^{(i)}$, $\beta_1^{(j)}$ and $\beta_2^{(i)}$, $\beta_2^{(j)}$, for $i \neq j$).

C. Bit Width Allocation for Statically-Reconfigurable Model

As stated in [1], FFT size $N_{\text{opt}}^{(i)}$ should be much larger than $l_{\text{kern}}^{(i)} - 1$ to make frequency domain convolution efficient. Thus, $x = \frac{l_{\text{kern}}^{(i)} - 1}{N_{\text{opt}}} \ll 1$, which validates the approximation:

$$(1+x)^n \approx 1 + n \cdot x; \qquad (1-x)^n \approx 1 - n \cdot x \tag{21}$$

Applying Equation 21 and the simplifications in Section VI-A to Equation 19, average time for layer i to compute an input image can be calculated as:

$$t_{\text{opt}}^{(i)} = \frac{1}{T_{\text{sys-opt,quan}}^{(i)}} \approx \frac{K_{\text{CNN}}^{(i)}}{L} \left(\left(\frac{\beta_2}{\beta_0}\right)^2 + \frac{4(l_{\text{kern}}^{(i)} - 1)}{B \cdot M^{1/2} \cdot L^{-1}} \sqrt{\frac{\beta_2}{\beta_0}} \frac{\beta_1}{\beta_0} \right) \tag{22}$$

We ignore superscript for β due to statically-reconfigurable model.

Average time for a CNN to process an input image is:

$$t_{\text{opt-CNN}} = \sum_{i=1}^{l} t_{\text{opt}}^{(i)} \tag{23}$$

which forms the objective of our optimization problem.

The constraint to the optimization problem is the overall SQNR specified by Equation 12. Note that $\frac{1}{\gamma_{\text{FFT}}} = \frac{1}{\gamma_{\text{IFFT}}} = 0$ (Section V-A); $\gamma_{\text{Hadamard}} = \gamma_{K^{\text{freq}}} = \gamma_{\beta_2}$ (Equation 11); $\frac{1}{\gamma_{\text{re-quan}}} = \frac{1}{\gamma_{\beta_1}} + \frac{1}{\gamma_{\beta_2}} + \frac{1}{\gamma_{\beta_4}} = \frac{2}{\gamma_{\beta_1}} + \frac{1}{\gamma_{\beta_2}}$ (Figure 3). Therefore:

$$\frac{1}{\gamma_{\text{CNN}}} = \left(\frac{1}{\gamma_{\text{input}}}\right)^{(1)} + \frac{2l}{\gamma_{\beta_1}} + \frac{2l}{\gamma_{\beta_2}} = \frac{2l}{\gamma_{\beta_1}} + \frac{2l}{\gamma_{\beta_2}} \tag{24}$$

where $\left(\frac{1}{\gamma_{\text{input}}}\right)^{(1)} = 0$ since the original image is not quantized before feeding into layer 1 of the CNN.

The optimization problem for the statically-reconfigurable model is formulated as:

$$\begin{aligned} \underset{\beta_1, \beta_2}{\text{minimize}} \quad & t_{\text{opt-CNN}}(\beta_1, \beta_2) \\ \text{subject to} \quad & \frac{2l}{\gamma_{\beta_1}} + \frac{2l}{\gamma_{\beta_2}} \leq \frac{1}{\gamma_0}. \end{aligned}$$

where γ_0 is a given budget for SQNR.

To solve the optimization problem, we first need Equation 4 to relate bit width β with SQNR γ. By applying further first order approximations on the objective function, we use the Lagrange multiplier method to derive an analytical solution to the optimization problem. We derive a simple and insightful relation between β_1 and β_2:

$$\beta_1 \approx \frac{10 \log_{10} \eta}{\kappa} + \beta_2 \tag{25}$$

where $\eta = \frac{B\sqrt{M}}{L} \cdot \frac{\sum K_{\text{CNN}}^{(i)}}{\sum K_{\text{CNN}}^{(i)} \cdot 2(l_{\text{kern}}^{(i)} - 1)}$; Equation 4 defines κ.

We conclude that the bit width for images should be $\frac{10 \log_{10} \eta}{\kappa}$ higher than that for kernels. For example, when $\eta = 10$ and $\kappa = 3$ dB/bit, $\beta_1 = \beta_2 + 3$.

D. Bit Width Allocation for Dynamically-Reconfigurable Model

Assuming dynamically-reconfigurable model, it is not necessary that $\beta_1^{(i)} = \beta_1^{(j)}$ and $\beta_2^{(i)} = \beta_2^{(j)}$ for $i \neq j$. This is the only difference from the statically-reconfigurable scenario.

The optimization problem in this case can be easily derived:

$$\begin{aligned} \underset{\beta_1^{(i)},\beta_2^{(i)}}{\text{minimize}} \quad & t_{\text{opt-CNN}}(\beta_1^{(i)}, \beta_2^{(i)}) \\ \text{subject to} \quad & \sum_{i=1}^{l} \frac{2}{\gamma_{\beta_1}^{(i)}} + \sum_{i=1}^{l} \frac{2}{\gamma_{\beta_2}^{(i)}} \leq \frac{1}{\gamma_0}. \end{aligned}$$

Using the same Lagrange multiplier technique, we derive the relation between $\beta_1^{(i)}$ and $\beta_2^{(i)}$ (where C is a constant):

$$\begin{aligned} \beta_1^{(i)} &\approx \frac{10\log_{10}\eta^{(i)}}{\kappa} + \beta_2^{(i)} \\ \beta_2^{(i)} + \frac{10\log_{10}\left(\beta_2^{(i)} K_{\text{CNN}}^{(i)}\right)}{\kappa} &= C \end{aligned} \tag{26}$$

Base on Equation 26, we first observe that kernels and images of the same layer should be quantized using different number of bits. The difference in bit width is captured by $\frac{10\log_{10}\eta^{(i)}}{\kappa}$, where $\eta^{(i)} = \frac{B\sqrt{M}}{L} \cdot \frac{1}{2(l_{\text{kern}}^{(i)}-1)}$. This result is consistent with the analysis on the statically-reconfigurable model. Secondly and more importantly, layers with larger number of CNN model parameters (categorized by K_{CNN}) should be assigned with less number of bits. This conclusion is consistent with the intuition in Sections III and IV-A.

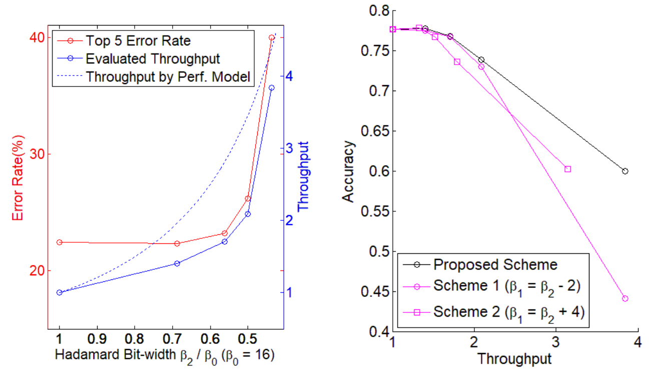
VII. Experimental Evaluation

A. Experimental Setup

We use Intel Stratix-V GXA7 device as our experimental FPGA platform, which has 6 MB on-chip memory, 534720 ALMs and 256 DSPs. We use the Quartus 18.0 tool for synthesis. We choose AlexNet as the CNN model under evaluation. Table I shows the specifications. As a reference, the floating point Top 5 accuracy of AlexNet is 78.2%.

Same as [1], we assume an FPGA-CPU heterogeneous execution model, where FPGA computes the convolutional layers and CPU computes the other layers. When measuring throughput, we only consider the FPGA execution. Thus, we use "throughput" to refer to throughput of convolutional layers. In addition, we calculate throughput as the total number of operations for *spatial* convolution, divided by the average execution time per image for our *frequency domain* design. Such definition of throughput is proposed by [1], and enables fair comparison with other designs using spatial convolution.

Apart from throughput, classification accuracy is another important metric. We have developed a software tool in `Python3` , which precisely simulates the execution of our accelerator and the corresponding hardware quantizers. The accuracy results below are obtained from our tool.



(a) Error rate and throughput under various bit-widths (b) Comparison of various quantization schemes

Fig. 4. Statically-reconfigurable architecture

B. Optimization for Statically-Reconfigurable Architecture

In this section, we evaluate the proposed quantization scheme for statically-reconfigurable architecture. We set κ as 3 dB/bit (Equation 4). Thus, $\beta_1 = \beta_2 + 2$.

Figure 4(a) shows the throughput and accuracy under various bit widths using our optimized quantization scheme. The left y-axis shows the overall classification error rates of AlexNet, which are obtained from our software simulator. The right y-axis shows the throughput of FPGA, which are obtained from performance modeling of the hardware. The dashed line shows the theoretical throughput based on the performance model in Section V-B. The solid blue line shows the evaluated throughput based on a much more precise performance model with additional hardware constraints (such as power of two FFT sizes, integer Hadamard array length, etc.) in mind. Note that throughput in this plot is normalized by the throughput of the 16-bit design. Figure 4(a) shows the tradeoff between accuracy and throughput. We observe $1.4\times$ speedup in throughput with negligible error rate penalty ($< 0.5\%$) when quantized from 16-bit to 11-bit, and $1.7\times$ speedup in throughput with limited accuracy loss ($< 5\%$) when quantized to 8-bit. Once the bit-width falls below 8-bit, throughput increases significantly at the cost of error rates infeasible to practical applications. In addition, the evaluated throughput matches reasonably well with the theoretical throughput. The discrepancy is mainly due to the limited flexibility in hardware.

Figure 4(b) shows the advantage of the proposed quantization scheme over other unoptimized schemes. We show the tradeoff between accuracy and throughput for various bit widths. The proposed scheme sets $\beta_1 = \beta_2 + 2$. Scheme 1 sets $\beta_1 = \beta_2 - 2$. Scheme 2 sets $\beta_1 = \beta_2 + 4$. Clearly, our proposed quantization scheme achieves the best accuracy and throughput. Throughput of Scheme 2 is limited by the communication bound. Scheme 1 suffers high accuracy loss at low bit widths. Such results are consistent with our analysis.

C. 11-Bit Statically-Reconfigurable CNN Accelerator

Because of the lack of flexibility in DSP blocks, we only implement the 11-bit statically-reconfigurable accelerator. Below we present the post place-and-route results.

TABLE II
COMPARISON WITH STATE-OF-THE-ART ALEXNET IMPLEMENTATIONS

	[15]	[14]	[1]	Proposed
Approach	Space	Frequency	Frequency	Frequency
FPGA	Stratix-V GXA7	Stratix-V GXA7	Stratix-V GXA7	Stratix-V GXA7
Frequency (MHz)	100	200	200	200
Precision (Fixed-point)	16-bit	16-bit	16-bit	11-bit
DSP Usage	256 (100%)	256 (100%)	256 (100%)	256 (100%)
Logic Usage (ALM)	$121K$ (52%)	$70K$ (30%)	$107K$ (46%)	$133K$ (57%)
On-chip RAM	1152 (61%)	1679 (89%)	1377 (73%)	1415 (75%)
Throughput (GOPS)	134.1	274.5	780.6	1131.9

Table II summarizes the comparison with state-of-the-art designs. The keyword "Space" denote implementations based on space domain convolution, and "Frequency" denote implementations based on frequency domain convolution. All designs are based on Stratix V GXA7 device. All of the three baseline designs use 16-bit fixed point arithmetic, while ours uses 11-bit. Throughput improvement of our implementation is due to both quantization and the low complexity frequency domain convolution algorithm. We achieve $1.45\times$, $4.12\times$ and $8.44\times$ speedup compared with [1], [14] and [15]. Our 11-bit design incurs negligible accuracy loss ($< 0.5\%$). Besides, FFT and IFFT modules only contribute to 18% of the total ALM/DSP consumption, which verifies the Section V-A analysis.

D. *Optimization for Dynamically-Reconfigurable Architecture*

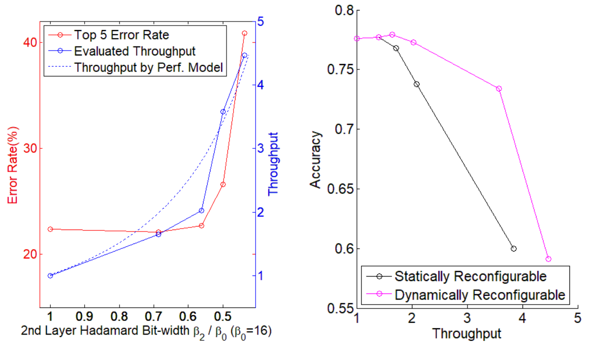
In this section, we evaluate the proposed quantization scheme for dynamically-reconfigurable architecture. We assume that dynamic reconfiguration across CNN layers can be done with low cost. Thus, in this section, we ignore all of the resource and time overhead due to dynamic reconfiguration.

Figure 5(a) shows the throughput and error rate under various bit widths. For the sake of notation, here "bit width" of x-axis refers to the bit width of the layer 2 Hadamard module. Similar as Figure 4(a), the left y-axis shows the classification error rate of the overall CNN, and the right y-axis shows the FPGA throughput (normalized by the throughput of the 16-bit design). 11-bit quantization leads to $1.65\times$ speedup in throughput with negligible error rate increment ($< 0.5\%$). 8-bit quantization leads to $3.57\times$ speedup in throughput with limited accuracy loss ($< 5\%$). The accelerator becomes infeasible to practical applications for bit width lower than 8.

Figure 5(b) shows the advantage of our dynamically-reconfigurable architecture over the optimized statically-reconfigurable architecture discussed in Section VII-B. Clearly, the dynamically-reconfigurable architecture leads to better performance in terms of accuracy as well as throughput. It remains a question, however, how to implement such dynamic reconfiguration with little resource and time overhead.

VIII. CONCLUSION

In this paper, we presented a high throughput CNN accelerator using fixed-point quantization in frequency domain. Based on the SQNR analysis and performance model, high throughput is achieved due to the optimized quantization schemes for statically and dynamically reconfigurable architectures.



(a) Error rate and throughput under various bit-widths

(b) Comparison of statically and dynamically-reconfigurable schemes

Fig. 5. Dynamically-reconfigurable architecture

In the future, we will explore system design that efficiently realizes dynamic reconfiguration for various bit widths. Besides, we will also explore sparsity in frequency domain kernels of CNNs. Compression on the high frequency components of frequency domain kernels potentially leads to even higher throughput without significant accuracy loss.

REFERENCES

[1] H. Zeng, R. Chen, C. Zhang, and V. Prasanna, "A framework for generating high throughput cnn implementations on fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: ACM, 2018.

[2] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning*.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[5] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014.

[6] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," *CoRR*, vol. abs/1312.5851, 2013.

[7] A. Daher and et al., "Overlap-save and overlap-add filters: Optimal design and comparison," vol. 58, no. 6. IEEE, 2010.

[8] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on cpu-fpga shared memory system," in *Proceedings of the 2017 ACM/SIGDA Intl. Symp. on Field-Programmable Gate Arrays*, ser. FPGA '17, 2017.

[9] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[10] M. Sit, R. Kazami, and H. Amano, "Fpga-based accelerator for losslessly quantized convolutional neural networks," in *2017 International Conference on Field Programmable Technology (ICFPT)*, Dec 2017.

[11] S. Ross, *A First Course in Probability*. Collier Macmillan, 2014.

[12] M. Püschel, J. M. F. Moura, B. Singer, J. Xiong, J. Johnson, D. Padua, M. Veloso, and R. W. Johnson, "Spiral: A generator for platform-adapted libraries of signal processing algorithms," *Int. J. High Perform. Comput. Appl.*, vol. 18, no. 1, pp. 21–45, Feb. 2004.

[13] R. Chen, H. Le, and V. K. Prasanna, "Energy efficient parameterized fft architecture," in *2013 23rd International Conference on Field programmable Logic and Applications*, Sept 2013, pp. 1–7.

[14] H. Zeng, C. Zhang, and V. Prasanna, "Fast generation of high throughput customized deep learning accelerators on fpgas," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*.

[15] Y. Ma, N. Suda, Y. Cao, J. Seo, and S. Vrudhula, "Scalable and modularized rtl compilation of convolutional neural networks onto fpga," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–8.