# FFConv: An FPGA-based Accelerator for Fast Convolution Layers in Convolutional Neural Networks

AFZAL AHMAD and MUHAMMAD ADEEL PASHA, Lahore University of Management Sciences (LUMS), Pakistan

Image classification is known to be one of the most challenging problems in the domain of computer vision. Significant research is being done on developing systems and algorithms improving accuracy, performance, area, and power consumption for related problems. Convolutional Neural Networks (CNNs) have shown to give outstanding accuracies for problems such as image classification, object detection, and semantic segmentation. While CNNs are pioneering the development of high accuracy systems, their excessive computational complexity presents a barrier for a more permeated deployment. Although Graphical Processing Units (GPUs), due to their massively parallel architecture, have shown to give performance orders of magnitude better than general purpose processors, the former are limited by their high power consumption and generality. Consequently, Field Programmable Gate Arrays (FPGAs) are being explored to implement CNN architectures, as they also provide massively parallel logic resources but with a relatively lower power consumption than GPUs. In this article, we present FFConv, an efficient FPGA-based fast convolutional layer accelerator for CNNs. We design a pipelined, high-throughput convolution engine based on the Winograd minimal filtering (also called Fast Convolution) algorithms for computing the convolutional layers of three popular CNN architectures: VGG16, Alexnet, and Shufflenet. We implement our accelerator on a Virtex-7 FPGA platform where we exploit the computational parallelization to the maximum while exploring optimizations aimed at improving performance. The resultant design loses only 0.43%, 0.47%, and 0.61% Top-1 classification accuracy for VGG16, Alexnet, and Shufflenet-v1, respectively, while significantly improving throughput, resource, and power efficiency compared to previous state-of-the-art designs.

CCS Concepts: • **Hardware → Hardware accelerators**;

Additional Key Words and Phrases: FPGA, convolutional neural networks, hardware acceleration

**ACM Reference format:**
Afzal Ahmad and Muhammad Adeel Pasha. 2020. FFConv: An FPGA-based Accelerator for Fast Convolution Layers in Convolutional Neural Networks. *ACM Trans. Embed. Comput. Syst.* 19, 2, Article 15 (March 2020), 24 pages.
https://doi.org/10.1145/3380548

## 1 INTRODUCTION

With the advent of Artificial Intelligence (AI), the popularity of Deep Neural Networks (DNNs) has exploded in the past few years with applications in various fields from computer vision and pattern recognition to natural language processing. Convolutional Neural Networks (CNNs) are

Authors' addresses: A. Ahmad and M. A. Pasha, Department of Electrical Engineering, Lahore University of Management Sciences (LUMS), Lahore, Pakistan; emails: {afzal.ahmad, adeel.pasha}@lums.edu.pk.

1539-9087/2020/03-ART15 $15.00
https://doi.org/10.1145/3380548

state-of-the-art algorithms for computer vision tasks such as image classification, object detection, and semantic segmentation [14, 16, 21]. While the accuracy achieved by CNNs on these tasks is unparalleled, their extreme computational budgets limit wider implementation. While Graphical Processing Units (GPUs) are being used to deploy CNN architectures exploiting the algorithmic parallelism over the many-cores that they provide [6], their power consumption is high and their architectures are more generic.

Owing to their reconfigurability, Field Programmable Gate Array (FPGA)-based implementations [22, 29, 41] are being explored to design parallel and pipelined network architectures that give improved performance and power efficiency compared to general-purpose processors (CPUs) and GPUs. While more custom solutions in the form of Application-Specific Integrated Circuits (ASICs) can be implemented that further improve the performance and power efficiency compared to their FPGA-based counterparts [3], ASIC-based designs are rigid, hence may only be justified at the stage of final implementation when thorough testing and prototyping has been done on a more reconfigurable FPGA-based platform.

Significant research effort is also being put into optimizing different layers of CNNs to gain improvements in the performance metrics for a wider range of CNN architectures and hardware platforms. Fast Fourier Transforms (FFT)-based convolutions have shown significant gains in reducing the computational complexity of convolutional layers that use large kernel sizes ($\geq 7 \times 7$) implemented on GPU platforms [31]. Although this reduction in computational complexity offered by FFT-based convolution is significant for large kernel sizes, modern neural network architectures, such as VGGNet [28], ResNet [16], MobileNets [17], and GoogLeNet [30], tend towards smaller kernel sizes and deeper topologies. FFT-based convolutions have actually been shown to increase the overall computation time of layers that use smaller kernel sizes by as much as 16× [31].

Winograd minimal filtering [34] based fast convolution algorithms (we will refer to them as "fast-conv" from here onwards) have also been proposed and have shown significant improvements for small kernel sizes, applicable to most modern networks [20]. Fast-conv algorithms work by reducing the computational complexity of expensive operations while adding transform stages that increase the number of cheaper operations involved in the convolution. Furthermore, the arithmetic cost of the additional transform stages can be amortized over layer dimensions, leading to an overall reduction in computation complexity.

In this article, we present FFConv, an efficient FPGA-based fast-conv accelerator for CNNs. We explore custom bitwidth quantization schemes in fast-conv and their impact on classification accuracy of the system. We follow through with a modular hardware implementation that not only allows the system to run at high frequency but also uses the resources efficiently for a tradeoff in throughput and accuracy. To this objective, we explore challenges that bottleneck the performance of our design and find optimizations to curb them and give a balance between performance and accuracy. The main contributions of this work are as follows:

- We model losses in classification accuracy for fast-conv based VGG16-D [28], AlexNet [19], and Shufflenet-v1 [43] for different quantization levels for feature and kernel maps.
- We propose FFConv, an FPGA-based optimized pipelined fast-conv accelerator for CNNs.
- We explore limitations in terms of memory-compute tradeoffs and introduce optimizations to our base design, resulting in performance improvements while costing a minute loss in classification accuracy.

The rest of the article is structured as follows: In Section 2, we present a short primer on CNNs, spatial convolution, and fast-conv while also surveying the previous works. Section 3 covers a design space exploration to find appropriate parameters and control knobs of fast-conv algorithms to be used in our hardware design. We also model the losses in classification accuracy for

different quantization levels for fast-conv based VGG16-D, AlexNet, and Shufflenet-v1 architectures. In Section 4, we present stage-wise implementation of FFConv while discussing challenges and optimizations along with a qualitative discussion of the novelties of our design compared to the state-of-the-art. Section 5 contains a detailed discussion and comparison of implementation results of FFConv against the state-of-the-art implementations in terms of four metrics: accuracy, throughput, resource, and power efficiency. The article is then concluded in Section 6.

## 2 BACKGROUND

CNNs are a form of supervised learning algorithms that work in two stages: training and inference. During training, the network adjusts its weights through back-propagation, learning its appropriate internal representations and yielding a trained model that is stored and later utilized for inference. During inference, an input image is passed though a series of layers to generate class labels that the image could belong to, with their probabilities. A set CNN architecture has a predefined configuration of layers, parameters, and hyper-parameters that define the network, and hence its accuracy, memory requirements, and computational complexity. Convolutional (conv) and Fully Connected (FC) are two fundamental types of layers in CNNs that are dependent on learned parameters called filters or kernels obtained from the training phase.

While FC layers are memory-hungry, having much more parameters than conv layers, the latter are compute-intensive, requiring lots of Multiply and Accumulate (MACC) operations to compute their output activations. Given that conv and FC layers contribute towards 90% and 5%–10% of the total computational budget, respectively [8], our work focuses on optimizing conv layers. Moreover, though fast-conv can be used to speed up the training process, the purpose of this work is to accelerate the inference stage to have a better run-time performance.

Several CNN architectures are being used to overcome various computer vision challenges. GoogLeNet [30], VGGNet [28], and ResNets [16] are some of the common network architectures used as benchmarks due to their high accuracies, while MobileNets [17] and Shufflenet [43] are network architectures geared towards computational efficacy.

GoogLeNet, a deep CNN architecture employing inception modules that used multiple receptive fields in each module to detect features effectively, achieved Top-5 error rate of 6.7%. VGGNet only utilized $3 \times 3$ filters to design deep CNN architectures having up to 16 conv layers, achieving a Top-5 error rate of about 7.5%. ResNet introduced skip connections that allowed training very deep networks of up to 152 layers by alleviating the issue of vanishing gradients, achieving Top-5 error rate of 3.6%. Shufflenet employed efficient convolution methodologies such as depthwise convolution and pointwise grouped convolutions to surpass accuracy of MobileNets at much lower computation cost. In this work, we use two relatively older and more established state-of-the-art CNNs, VGG16-D and Alexnet, and one relatively newer CNN, Shufflenet, as benchmarks to show the efficacy of our proposed design for a variety of CNN architectures. Since all these networks utilize the same convolution operation at their core, hardware architectures of conv layers that utilize other parameters can easily be designed using our approach.

### 2.1 Fast Convolution (fast-conv)

*2.1.1 Convolutional (conv) Layer.* A conv layer takes a feature map $D$ and a filter map $W$ as inputs and generates an output feature map $Y$ that is then passed to the next layer. The output feature map of layer $i$, $Y^i$, is the input feature map to the next layer $D^{i+1}$. The filter or kernel map is a tensor of parameters learned during the training phase. Conv layers learn high-level features of the input feature maps to be used in classification.

In spatial convolution, corresponding elements of an input feature map tile and a filter map, each of size $C \times r \times r$, undergo a MACC operation to generate a single pixel of the output feature

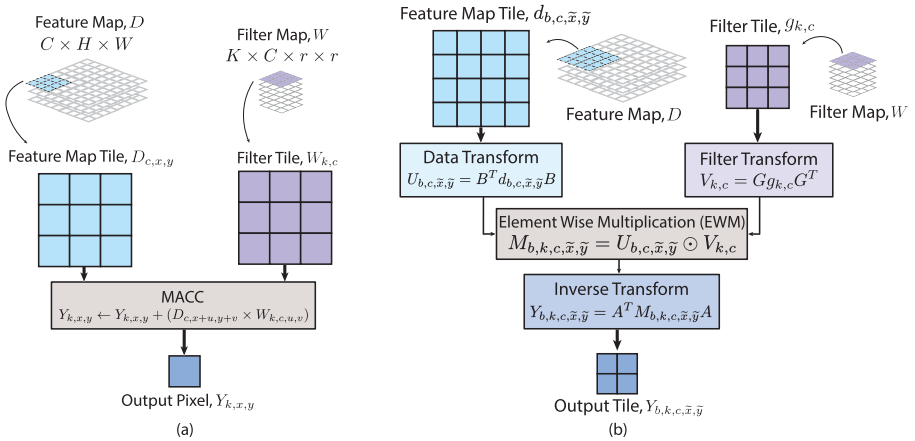


Fig. 1. Convolution of a single channel tile of a feature map with a filter tile to generate (a) a single output pixel using spatial-conv and (b) an output tile, $m, r = 2, 3$ using fast-conv.

map, where $C$ is the number of channels and $r$ is the kernel dimension. The same kernel is then swept across the whole input feature map with a predefined stride to generate a single channel of the output feature map. The process is repeated with $K$ different kernel maps to generate $K$ channels of the output feature map. Let $D_{c,x,y}$ and $W_{k,c,u,v}$ be origins of $r \times r$ tiles of input feature map and kernel map, respectively, pixel $Y_{k,x,y}$ of the output feature map can be calculated as,

$$Y^i_{k,x,y} = \sum_{c=1}^{C} \sum_{v=1}^{r} \sum_{u=1}^{r} D^i_{c,x+u,y+v} \; W^i_{k,c,u,v}, \tag{1}$$

where $i$ is the layer number, $x$ and $y$ are origin coordinates of feature map tiles, $u$ and $v$ are the iterators over the coordinates of the input feature map tile and the kernel map, $k$ is the kernel index, which after convolution translates to the channel of the output feature map. Figure 1(a) shows this calculation in a pictorial representation for a single-channel, single kernel map tile configuration.

*2.1.2 Motivation for Fast-conv.* While spatial convolution computes a single pixel of the output feature map before the kernel is strode to the next location, fast-conv algorithms compute a two-dimensional (2D) tile of output feature map. A 2D fast-conv algorithm, represented as $F(m \times m, r \times r)$, takes an input feature map tile of size $t \times t$ where $t = m + r - 1$ and kernel tile of size $r \times r$ and computes an output feature map tile of $m \times m$ pixels before making a stride to the next location. The number of multiplications required to calculate these $m^2$ units of output is equal to the input tile size, $t^2$. In contrast, spatial convolution takes $r^2 \times m^2$ multiplications to compute $m^2$ outputs. For a common kernel size $r = 3$ and taking output tile size, $m = 2$, this reveals savings in multiplication complexity of 2.25× as calculated below:

$$\frac{m^2 r^2}{(m + r - 1)^2} = \frac{2^2 \times 3^2}{(2 + 3 - 1)^2} = 2.25 \times . \tag{2}$$

These savings in multiplication complexity can be leveraged, especially on hardware platforms such as FPGAs with dedicated Digital Signal Processor (DSP) units that give very low latency for the operation while allowing parallel instantiation to increase throughput.

*2.1.3 Fast-conv Basics.* A 2D fast-conv algorithm, represented as $F(m \times m, r \times r)$ is given by:

$$Y_{b,k,c,\tilde{x},\tilde{y}} = A^T [(B^T d_{b,c,\tilde{x},\tilde{y}} B) \odot (G g_{k,c} G^T)] A, \tag{3}$$

where $B$, $G$, and $A$ are data, filter, and inverse transform constant matrices, $d_{b,c,\widetilde{x},\widetilde{y}}$ is input feature map tile of dimensions $t \times t$ originating at coordinates $(b, c, \widetilde{x}, \widetilde{y})$, $g_{k,c}$ is filter map tile of dimensions $r \times r$ originating at coordinates $(k, c)$, $Y_{b,k,c,\widetilde{x},\widetilde{y}}$ is output feature map tile of dimensions $m \times m$ originating at coordinates $(b, k, c, \widetilde{x}, \widetilde{y})$, $b$ represents the batch index, $k$ is the kernel index, $c$ is the channel index, and $(\tilde{x}, \tilde{y})$ are the column and row coordinates of the feature map tile.

Fast algorithms work by transforming input feature maps and filter maps to Winograd domain and then performing Element-Wise Multiplication (EWM) between the transformed maps, which is equivalent to matrix multiplication in spatial domain. The result is, then, inverse transformed back to the spatial domain. Input data transform $U_{b,c,\widetilde{x},\widetilde{y}} = B^T d_{b,c,\widetilde{x},\widetilde{y}} B$, filter transform $V_{k,c} = G g_{k,c} G^T$, and inverse transform $Y_{b,k,c,\widetilde{x},\widetilde{y}} = A^T M_{b,k,c,\widetilde{x},\widetilde{y}} A$ are the additional three transform stages of fast-conv algorithm that add to the arithmetic complexity of the conv layer compared to spatial convolution, where $M_{b,k,c,\widetilde{x},\widetilde{y}}$ is the result of EWM stage. The EWM stage, $M_{b,k,c,\widetilde{x},\widetilde{y}} = U_{b,c,\widetilde{x},\widetilde{y}} \odot V_{k,c}$ is the Winograd domain multiplication equivalent of the spatial convolution direct multiplication of single tiles of kernel maps with feature maps but fast-conv algorithm outputs a larger tile, $m \times m$ per stride of the kernel map instead of a single pixel generated by the spatial convolution as depicted by Figure 1(b).

### 2.2 Related Works

Extensive research effort is being put into making CNNs more attractive for mobile applications, decreasing their complexity by network pruning, approximation, and quantization. Reference [38] demonstrated energy-efficient network pruning by sparsifying the filters, reducing energy consumption of AlexNet and GoogLeNet by 3.7× and 1.6×, respectively, while incurring less than 1% Top-5 accuracy loss. Reference [18] proposed approximations to full-rank filters of CNNs as groups of rank-1 filters, achieving approximately 4.5× speedup while incurring less than 1% drop in accuracy of a CNN that performed character classification in a scene. Reference [35] proposed quantization of filter maps and FC layer weights by estimating and correcting error of each layer's response, achieving 4.06× speed-up and 20.34× compression of VGG16 model at the cost of only 0.58% increase in Top-5 error rate. Reference [15] presented Deep Compression, a combination of network pruning for removing insubstantial weights, quantization to reduce bitwidths of pruned model, and Huffman coding for further compressing the model, achieving model reduction of 35× and 49× for AlexNet and VGG16, respectively, while also improving throughput and energy efficiency. Reference [9] proposed Binarized Neural Networks (BNNs), which utilized concatenation of 32 binary trained parameters in a single 32-bit register to maximize efficacy of memory utilization while achieving high speedups. Reference [44] designed a BNN-accelerator on an FPGA platform based on the work of Reference [9], exploiting the cheap binary arithmetic operations suited for FPGA platforms instead of complex operations such as multiplications, reducing the memory footprint, and accelerating the inference stage.

Several research works have also focused on exploiting algebraic frameworks of conv layers to reduce computation complexity of CNNs. Reference [31] used FFTs with convolution to reduce the arithmetic complexity of conv layers. Unfortunately, FFTs only showed savings in arithmetic complexity for large filters while modern state-of-the-art CNN architectures tend towards small filter sizes. Reference [8] used Strassen algorithm for matrix multiplication to decrease arithmetic complexity of conv layers. Reference [20] proposed fast-conv, which is based on Winograd minimal filtering algorithms first proposed in Reference [34]. Reference [20] also implemented $F(2 \times 2, 3 \times 3)$ on a GPU platform and achieved 7.28× speedup for a batch size of 8 relative to cuDNN [27].

After Reference [20] proposed fast-conv algorithms for CNNs, several software and hardware implementations have been proposed exploiting the savings in arithmetic complexity. Reference [37] proposed a Winograd-based Reconfigurable Architecture (WRA) for Internet of Things

Table 1. State of GPU vs FPGA-based CNN Implementations

| | [7] | [23] | | [2] | [42] | [24] | |
|---|---|---|---|---|---|---|---|
| Device | TitanX | TitanX | ZCU102 | Arria 10 | Arria 10 | Jetson[a] | ZCU102 |
| Network Arch. | AlexNet | VGG16 | VGG16 | AlexNet | VGG16 | YOLOv2 | YOLOv2[b] |
| Precision | float32 | float32 | fixed16 | float16 | fixed16 | float32 | binary |
| Throughput (GOPS/s) | 6,937 | 5,600 | 2,941 | 1,382 | 1,790 | 51.7 | 1,424.5 |
| Frame rate (fps) | 5,120 | 182.5 | 95.8 | 1,020 | 58.3 | 1.48 | 40.8 |
| Power (W) | 227 | 134 | 23.6 | 45 | 37.5 | 7.0 | 4.5 |
| Power Efficiency (GOPS/s/W) | 30.56 | 41.80 | 124.60 | 30.71 | 47.78 | 7.39 | 316.56 |

[a]NVIDIA Jetson TX2 Pascal GPU implementation.
[b]Lightweight, custom implementation.

(IoTs), implementing their proposed accelerator for ShuffleNet [43], a mobile CNN architecture, and achieved high throughput on a Zync 7-Series FPGA platform. Reference [39] proposed a fast-conv based, instruction driven, cross-layer CNN accelerator, which reduced data transfer among layers of VGG16 using $F(2 \times 2, 3 \times 3)$. Reference [36] developed a fusion architecture for reusing inter-layer data in CNNs based on fast-conv and designed a high-level synthesis (HLS) tool-chain for easing the mapping of their designs to FPGA platforms. In Reference [25], a scalable and pipelined design of $F(2 \times 2, 3 \times 3)$ was proposed, which reused inter-tile overlap in fast-conv algorithm. Reference [23] proposed a line buffer structure for caching portions of feature map data tiles for improving reuse and designed an automated tool for efficiently mapping fast-conv based CNN on FPGAs. Reference [2] implemented an OpenCL-based accelerator system for AlexNet on Intel Arria 10 based on 1D fast-conv $F(4, 3)$ achieving throughput of 1,382 GFLOPS/s and power efficiency similar to NVIDIA's TitanX GPU. Reference [1] explored the design space associated with fast-conv algorithms, demonstrating improvements in throughput and power efficiency of different fast-conv algorithms for FPGA-based implementations.

### 2.3 State of FPGA-based Embedded CNN Solutions

GPUs have shown unprecedented performance results for CNNs in many applications. Reference [7] performed GPU benchmarks for several conv-net implementations, achieving forward pass latency up to 25 *ms* for a batch of 128 images for AlexNet, corresponding to a whopping 5,120 frames per second (fps). Similarly, Reference [23] found throughput of up to 5.6 TOPS/s (182.5 fps) for VGG16 on a TitanX GPU. While these throughput numbers are significantly higher than FPGA-based designs, the latter have demonstrated higher power efficiency, which makes them suitable for embedded vision applications. Table 1 compares performance and power efficiency of a selection of image classification and object detection algorithms on GPU and FPGA platforms. Reference [23] implemented their VGG16 architecture on both TitanX GPU and Ultrascale+ FPGA and found the power efficiency of the FPGA-based design to be 2.98× better while achieving 0.53× the throughput. Similarly, Reference [42] implemented VGG16 on Intel Arria 10 FPGA to achieve a throughput of 1,790 GOPS/s at a power consumption of only 37.5 W. Reference [24] demonstrated 27.5× better throughput and 42.9× better power efficiency for YOLOv2, a state-of-the art object detection algorithm, on Xilinx Ultrascale+ FPGA compared to NVIDIA Jetson Embedded GPU platform. This survey of previous implementations demonstrates that while both GPU- and FPGA-based implementations meet real-time constraints of 30 fps, FPGA-based implementations give significantly higher power efficiencies in most cases, which would lead towards the future green (energy-aware) implementations.

## 3 DESIGN SPACE EXPLORATION

In this section, we will present the design space exploration steps taken to fine-tune different design knobs. The section will start with the selection of optimal fast-conv parameters, which is followed by the exploration of quantization schemes for feature and kernel maps.

### 3.1 Fast-conv Parameters

The two parameters of fast-conv algorithms, $m$ and $r$, play a key role in defining the performance and applicability of the conv layers based on fast-conv. Although a high-level design of fast-conv is possible, which allows easy deployment for any values of $m$ and $r$, the design generated by HLS tools is unoptimized with long critical paths and high resource utilization, resulting in reduction in throughput and resource efficiency. To get the best possible performance from the hardware platform, we did a brief background study of the two parameters of fast-conv algorithms. Parameter $m$, output tile size, directly affects throughput and precision of the convolution hardware to be designed while parameter $r$, kernel size, affects the applicability of the designed hardware [20].

A plethora of highly successful CNN architectures are tending towards deeper topologies while utilizing shorter kernel sizes with $r = 3$ being a common choice. The reason is that deeper networks with small kernels achieve better accuracy than shallower networks with large kernels [28]. Two of the variants of ResNet [16], 18-layer and 34-layer, both utilize $r = 3$ in all of their layers except the first layer. GoogLeNet [30] also utilizes $r = 3$ in its inception modules along with other small kernels. Our selected CNN architectures for benchmarking, VGG16, AlexNet, and Shufflenet, also utilize $r = 3$ to a great extent. VGG16-D uses $r = 3$ in all five of its conv group layers, while AlexNet uses $r = 3$ in last three of its five conv layers. Shufflenet utilizes both depthwise convolutions ($r = 3$) and pointwise convolutions ($r = 1$) in its architecture. The improved applicability of fast-conv to layers with small kernel sizes compared to FFT-based approaches coupled with the fact that most modern CNN architectures tend towards small kernel sizes led us to select $r = 3$ as the kernel size parameter in designing our accelerator.

While increasing $m$ increases the throughput, References [1, 20] have shown that the arithmetic complexity of transforms increases exponentially with increase in $m$ for VGG16. Reference [1] showed that beyond $m = 4$, arithmetic complexity of transform stages outweighs the savings from the reduction in multiplication complexity. Furthermore, Reference [20] showed that added transform stages result in loss in precision relative to spatial convolution. Increasing $m$ results in more complex transform stages and hence higher losses in precision as well as larger input tiles, leading to higher memory-bandwidth requirements. This led us to select $m = 4$ as the output feature size.

To conclude, we decided to implement a fast-conv engine using $m, r = 4, 3$ as the parameters of choice as a balance between decrease in multiplication complexity, increase in transforms' complexity, and loss in precision relative to spatial convolution due to the complex transform stages. For this set of $m, r$, the input feature map tile is of size $t \times t = 6 \times 6$ (where $t = m + r - 1$), while the kernel map tile $r \times r = 3 \times 3$. The output tile generated by the convolution engine would be of size $m \times m = 4 \times 4$.

### 3.2 Quantization in Fast-conv

Deep Neural Networks are known to give high accuracies, even for very constrained data precisions [5]. Although low precision training and inference in CNNs using spatial-conv layer has been extensively studied [10, 45], fast-conv algorithms affect the classification accuracy differently due to the added transform stages. Studying quantization schemes helps realize a system that utilizes the resources efficiently while giving an acceptable compromise in the classification accuracy. To this end, we implemented fast-conv layers–based VGG16, AlexNet, and Shufflenet-v1 architectures in software to model the loss in classification accuracy with decrease in quantization bitwidths of

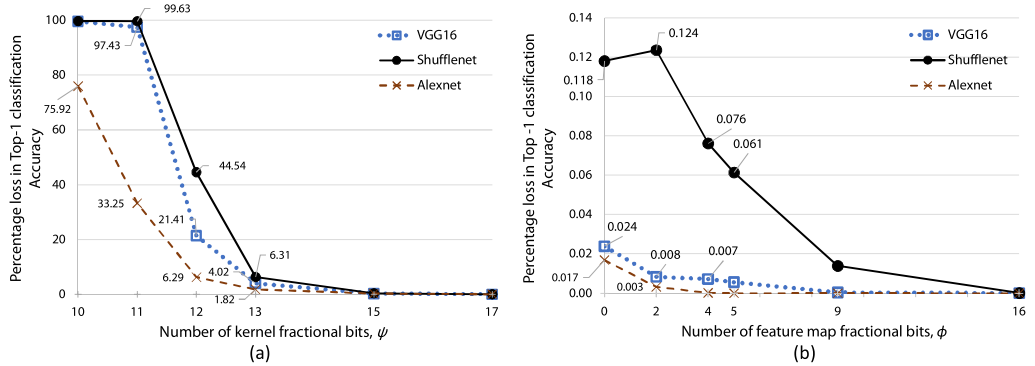


Fig. 2. Average % loss in Top-1 classification accuracy with (a) kernel quantization $Q1.\psi$ and (b) feature map quantization $Q12.\phi$.

pretrained models, running our tests on a subset of Imagenet 2012 test dataset [11]. Please note that for $r = 1$ in Shufflenet, a transform has been used that changes $r = 1$ convolutions to $r = 3$ convolutions and for AlexNet, the last three layers that utilize $r = 3$ are based on fast-conv, while the first two layers that utilize $r = 11$ and $r = 5$ are implemented as spatial-conv in single-precision floating point (i.e., `float32`).

*3.2.1 Kernel Map Quantization.* To model the impact of kernel map quantization on the classification probabilities, we used different fixed-point bitwidths for the kernel maps while using 32-bit fixed-point quantization (Q16.16) for the feature map inputs. Since the kernel weights are centered around zero with a low standard deviation, it is appropriate to assign a high number of bits to the fractional portion of the fixed-point representation of the kernel maps. We assign only one bit to the integer portion and all remaining bits to the fractional portion to achieve maximum precision of the network based on fixed-point data types. Figure 2(a) shows the variation in percentage (%) loss in Top-1 classification accuracy with the fractional bitwidth of kernel map input ($\psi$). It can be clearly noted that although the average % loss in Top-1 classification accuracy relative to a `float32` implementation increases with decreasing number of fractional bits, this increase in average loss largely occurs when the fractional bitwidth gets too small.

Going from Q1.31 to Q1.17, we did not observe a measurable loss in accuracy for the test dataset on any of the three networks—VGG16, AlexNet, and Shufflenet—but going from Q1.13 to Q1.11, we get over 90% drop in accuracy for VGG16 and Shufflenet, and over 30% drop for AlexNet. This is simply reminiscent of the fact that improvement in representation of small floating-point numbers in fixed-point notation increases exponentially with the number of fractional bits. We see a high average loss in accuracy for Shufflenet compared to VGG16, since the former utilizes efficient convolution methodologies aimed at reducing the arithmetic complexity, hence are not as rigid to perturbations in bitwidths compared to a more computationally complex network such as VGG16. We also observe a low average loss in classification accuracy for AlexNet compared to both VGG16 and Shufflenet, because the first two layers of AlexNet were computed in `float32` using spatial-conv.

Based on this analysis, we conclude that $\psi = 15$ is the lower-bound on the number of fractional bits for kernel maps for non-critical accuracy constraint systems such as mobile applications and AI-based home automation systems. Kernel map quantization of $\psi \geq 15$ would give a feasible improvement in resource utilization and performance while costing very little in accuracy where $\psi = 15$ costs only 0.272%, 0.291%, and 0.411% increase in percentage loss in classification accuracy for VGG16, AlexNet, and Shufflenet, respectively, compared to a `float32` implementation.

*3.2.2 Feature Map Quantization.* To model the impact of feature map quantization, we used different fixed-point bitwidths for feature maps while using 32-bit fixed-point quantization (Q16.16) for kernel maps. Feature map inputs depend on the input image or outputs of previous layer and have high absolute values, since sub-sampling using max-pool operation after each conv layer conserves only the maximum value at each stride of the pooling receptive field on the output feature map, the result of which is passed as input feature map to the next layer.

Figure 2(b) shows the variation in % loss in Top-1 classification accuracy with the fractional bitwidth of feature map input ($\phi$). It is evident that as long as the number of bits assigned to the integer portion is enough to hold the integer without overflowing, there is minimal loss in accuracy. This means that a simple rounding-to-nearest-integer operation for quantization of feature map inputs is feasible and adding extra bits for the fractional portion has little impact on the classification accuracy. Figure 2(b) shows that quantization scheme Q12.0, which is enough to hold the integer portion of the feature map inputs and entirely truncates the fractional portion, giving only approximately 0.024%, 0.017%, and 0.118% drop in average classification accuracy for VGG16, AlexNet, and Shufflenet, respectively, compared to `float32`. For contrast, a single precision float (32-bits) uses 2.67× more bits than Q12.0, requiring proportional resources to implement in hardware. Furthermore, adding extra bits to the integer portion would not have any impact on the average loss in accuracy, since any extra significant bits would go unused in all subsequent computations.

Based on this analysis, we conclude that $\phi = 0$ is the lower-bound on number of fractional bits for feature maps in applications with non-critical accuracy constraints and any value of $\phi \geq 0$ would give minute loss in classification accuracy while improving resource utilization and performance.

## 4 PROPOSED HARDWARE DESIGN

In this section, we present FFConv, our fixed-point quantized conv layer accelerator based on fast-conv $F(4 \times 4, 3 \times 3)$ algorithm. The design of the accelerator is separated into four modules, corresponding to the four stages: data transform, filter transform, EWM, and inverse transform. We will discuss the design choices that resulted in improved throughput, area, and power efficiency while ensuring operation of the system at a high frequency. We will also discuss memory-compute tradeoffs and limitations of our design and devised optimizations that make our design efficient.

### 4.1 Hardware Platform

To maximize throughput, resource, and power efficiency, the convolution accelerator is designed while keeping the available hardware resources in view. We chose a very popular Xilinx Virtex 7 VC707 Evaluation Platform that uses an XC7VX485T FPGA; however, other similar FPGAs from Xilinx or Intel can also be used as a target platform. The Xilinx FPGA device has 2,800 high-speed, low-power DSP48E1s, each having a dedicated $25 \times 18$ bit multiplier. This DSP slice is found in several devices, including Xilinx 7 series FPGA family lineup, including Artix, Kintex, Spartan, and Virtex, hence allowing our proposed design to be ported to other FPGAs with minor changes. The Intel FPGA lineup including Stratix, Arria, and Cyclone series incorporate DSP blocks, each having two 18-bit multipliers that can either be configured independently or in a single 27-bit multiplier mode. Our quantization analysis in Section. 3.2 showed that an 18-bit quantization scheme also incurs minimal loss in accuracy, which makes our design feasible for two 18-bit multipliers per DSP configuration on most Intel FPGA platforms.

### 4.2 Base Design

*4.2.1 Data Transform Stage.* The data transform stage takes input feature map tile, $d_{b,c,\widetilde{x},\widetilde{y}}$, of size $t \times t = 6 \times 6$ as an input and implements $U_{b,c,\widetilde{x},\widetilde{y}} = B^T d_{b,c,\widetilde{x},\widetilde{y}} B$ where for $m, r = 4, 3$, $B^T$ is

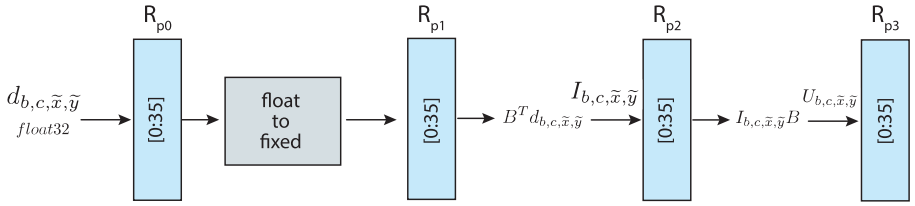


Fig. 3. Data transform implementation as a three-stage pipeline.

given as [20]:

$$B^T = \begin{pmatrix} 4 & 0 & -5 & 0 & 1 & 0 \\ 0 & -4 & -4 & 1 & 1 & 0 \\ 0 & 4 & -4 & -1 & 1 & 0 \\ 0 & -2 & -1 & 2 & 1 & 0 \\ 0 & 2 & -1 & -2 & 1 & 0 \\ 0 & 4 & 0 & -5 & 0 & 1 \end{pmatrix}.$$

Constant multiplications are simple operations implemented using shift operator and require negligible logic resources. The output of the data transform stage, $U_{b,c,\widetilde{x},\widetilde{y}}$, is a tile of size $6 \times 6$.

We implemented the data transform module as a three-stage pipeline where the first stage converts input `float32` feature map tile to fixed-point while the remaining two stages perform the data transforms as shown in Figure 3. This is because implementing two dense matrix multiplications as a single combinatorial module results in a datapath with a long critical delay due to the high number of constant multiplications and additions involved in it. Splitting it into two stages allows our module to operate at a higher frequency. Although adding three pipeline stages incurs an overhead in register logic consumption, our design is unconstrained by the register logic utilization.

An HLS-based implementation of this stage would simplify the two matrix multiplications into a set of 36 equations corresponding to each output whose implementation would yield a combinatorial block with a long critical path. This reduces the maximum operating frequency of the design to create a bottleneck at this stage. Although an HLS-based combinatorial module might give a shorter critical path for lower values of $m$ where the transform stages are less dense, for $m = 4$ and beyond, HLS-based implementation fails to give an optimized module for this stage, justifying our need to design a low-level, Hardware Descriptive Language (HDL)-based pipelined design to achieve shorter critical path and higher operating frequency.

Loss in accuracy occurs only when converting from `float32` to fixed-point notation. The data transform stage itself causes no loss in accuracy, since the inputs are in fixed-point and the elements of the data transform matrix are also integers.

*4.2.2 Filter Transform Stage.* The filter transform stage takes a kernel map tile, $g_{k,c}$, of size $r \times r = 3 \times 3$ as an input and implements $V_{k,c} = Gg_{k,c}G^T$ where for $m, r = 4, 3$, filter transform matrix $G$ is given as [20]:

$$G = \begin{pmatrix} \frac{1}{4} & 0 & 0 \\ -\frac{1}{6} & -\frac{1}{6} & -\frac{1}{6} \\ -\frac{1}{6} & \frac{1}{6} & -\frac{1}{6} \\ \frac{1}{24} & \frac{1}{12} & \frac{1}{6} \\ \frac{1}{24} & -\frac{1}{12} & \frac{1}{6} \\ 0 & 0 & 1 \end{pmatrix}.$$

Although filter transform can be computed offline, since it is dependent only on the learned parameters from the training phase, i.e., kernel maps $g_{k,c}$, we implemented this stage in hardware

to save on memory bandwidth requirement of the design. Filter transforms expand kernel tiles from $g_{k,c}$, which is of dimensions $3 \times 3$ to $V_{k,c}$, which is of dimensions $6 \times 6$. Loading $V_{k,c}$ from the external memory instead of $g_{k,c}$ would lead to 4× higher memory bandwidth requirement for kernel maps, making memory bandwidth a bottleneck in our design, since fast-conv requires loading feature tiles of size $6 \times 6$ compared to $3 \times 3$ in spatial convolution. The issue of memory bandwidth is discussed in detail in Section 4.3.

This stage is also implemented as a three-stage pipeline where the first stage performs the float to fixed conversion while the subsequent two stages compute $V_{k,c}$. The transformation matrix $G$ is converted to fixed-point offline before being utilized by the convolution engine. There is no latency overhead of implementing filter transforms online, since data and filter transforms are independent and are computed in parallel.

*4.2.3 Element-wise Multiplication (EWM) Stage.* The EWM stage takes the output of the data and filter transform stages, $U_{b,c,\tilde{x},\tilde{y}}$ and $V_{k,c}$, and computes $M_{b,k,c,\tilde{x},\tilde{y}} = U_{b,c,\tilde{x},\tilde{y}} \odot V_{k,c}$ where both $U_{b,c,\tilde{x},\tilde{y}}$ and $V_{k,c}$ are $6 \times 6$ tiles. This stage is implemented using DSPs exclusively for speed and power efficiency. Although each DSP48E1 has only one dedicated $25 \times 18$ bit multiplier, multiple DSPs can be chained to generate multipliers of higher bitwidths as done in Reference [1]. Based on the fixed-point precision analysis of fast-conv algorithm in Section 3.2, we opted for a single multiplier per DSP, as this results in a high resource efficiency while incurring negligible loss in accuracy.

*4.2.4 Inverse Transform Stage.* The inverse transform stage takes the result of the EWM stage, $M_{b,k,c,\tilde{x},\tilde{y}}$, of size $6 \times 6$ as an input and implements $Y_{b,k,c,\tilde{x},\tilde{y}} = A^T M_{b,k,c,\tilde{x},\tilde{y}} A$ where for $m, r = 4, 3$, inverse transform matrix $A^T$ is given as [20]:

$$A^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 2 & -2 & 0 \\ 0 & 1 & 1 & 4 & 4 & 0 \\ 0 & 1 & -1 & 8 & -8 & 1 \end{pmatrix}.$$

Similar to the data and filter transform stages, we implemented inverse transform as a two-stage pipeline (similar to that shown in Figure 3). $X_{b,k,c,\tilde{x},\tilde{y}} = A^T M_{b,k,c,\tilde{x},\tilde{y}}$ is computed in the first pipeline stage, while $Y_{b,k,c,\tilde{x},\tilde{y}} = X_{b,k,c,\tilde{x},\tilde{y}} A$ is computed in the second stage to allow the design to operate at a high frequency.

Since the EWM stage output increases in bitwidth due to multiplications, truncation is employed at the output of the EWM stage to equate the bitwidth of the output with the bitwidth of the input. This is done to minimize the contribution of write bandwidth towards the total bandwidth requirement, as discussed in Section 4.3.

*4.2.5 Overall Base Design.* To summarize, Figure 4 shows the base design of our fast-conv accelerator $F(4 \times 4, 3 \times 3)$, which consists of a data transform, filter transform, EWM, and an inverse transform stage. The system takes a `float32` feature map tile $d_{b,c,\tilde{x},\tilde{y}}$ of size $6 \times 6$ and a `float32` kernel tile $g_{k,c}$ of size $3 \times 3$. The first pipeline stages of data and filter transform convert the inputs `float32` to fixed-point. The subsequent two stages compute the transforms, $U_{b,c,\tilde{x},\tilde{y}}$ for data and $V_{k,c}$ for filter transform. EWM operation is performed on these transforms and the result $M_{b,k,c,\tilde{x},\tilde{y}}$, is inverse transformed in two pipeline stages to generate $Y_{b,k,c,\tilde{x},\tilde{y}}$, a $4 \times 4$ output tile.

### 4.3 Memory-computation Optimization

While fast-conv offers significant savings in arithmetic complexity relative to spatial convolution, there is an overwhelming demand for memory bandwidth, since larger feature map tiles need to

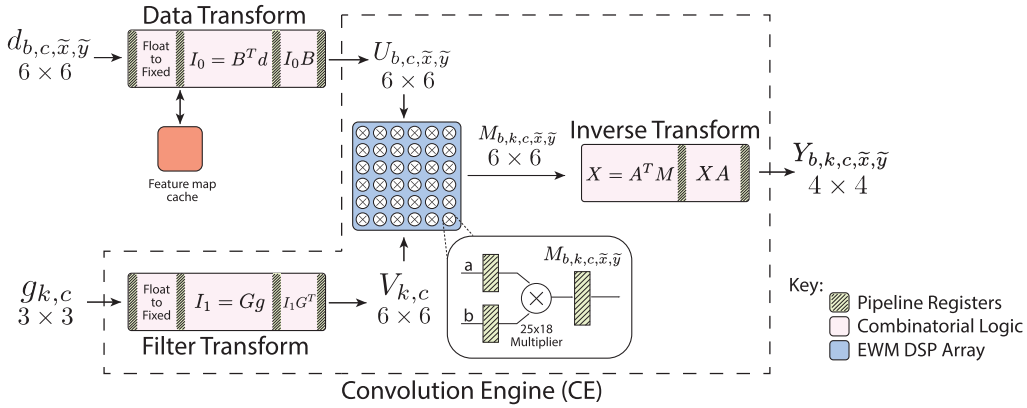


Fig. 4. Base design for fast-conv accelerator, $F(4 \times 4, 3 \times 3)$.

be loaded in each iteration. Furthermore, filter transforms expand each $3 \times 3$ input kernel to $6 \times 6$, which would result in 4× the memory bandwidth for loading of kernel data from the external memory if filter transform is implemented offline. This strengthens our case for implementing hardware-based filter transform stages for each CE. Although there is an associated logic cost since the stage has to be replicated for each CE for parallel processing, there is no latency overhead of implementing online filter transform stages, since they are computed in parallel to the data transform stage.

We quantitatively assess the memory-computation relation of our accelerator design applied to three CNN architectures—VGG16, AlexNet, and Shufflenet—while constrained with the available resources of our target platform. We use roofline model [33] for this analysis due to its effectiveness in capturing performance parameters of a wide variety and to provide insights into potential algorithmic and hardware limitations and directions for optimization of design for improving performance.

We use a Virtex 7 VC707 with DDR3 memory with a standard bus-width of 64-bits for memory bandwidth ceiling computation, achieving peak uni-directional bandwidth of 12.8 GB/s. The compute bounds are calculated by using the network parameters, available FPGA resources, and our base design for CE. Figures 5(a), (b), and (c) show the rooflines for VGG16, AlexNet, and Shufflenet, respectively, with attainable performance on the vertical axis and arithmetic intensity on the horizontal axis (log-log scale). Arithmetic intensity is a measure of the computational workload (GOPs) per unit of memory access (byte). The available memory bandwidth is solely dependent on the hardware platform, while the computation roof is dependent on both available resources and efficiency of hardware design. The points where the lines steepen depict the transition point between memory bottleneck and compute bottleneck. The arithmetic intensity at this point is 346.5 and 21.1 GOPs/byte for VGG16/AlexNet and Shufflenet, respectively. It also gives an insight into the maximum amount of data that can be transferred between the FPGA and external memory during the time the network output is computed for avoiding a memory-bound in the design.

The computation ceilings for VGG16 and AlexNet are the same, since all the evaluated layers utilize the same $r = 3$ kernel and our accelerator base design maps with equal efficiency to all different layer with $r = 3$. The maximum achievable performance for Shufflenet is much lower than VGG16/AlexNet, since the former utilizes $r = 1$ to a great extent, which maps sub-optimally to $r = 3$ kernel after transforming the $r = 1$ convolutions to $r = 3$.

It can be observed from Figure 5 that the memory burden for a design based on spatial convolution would be significantly lower compared to our design based on fast-conv due to the fact that

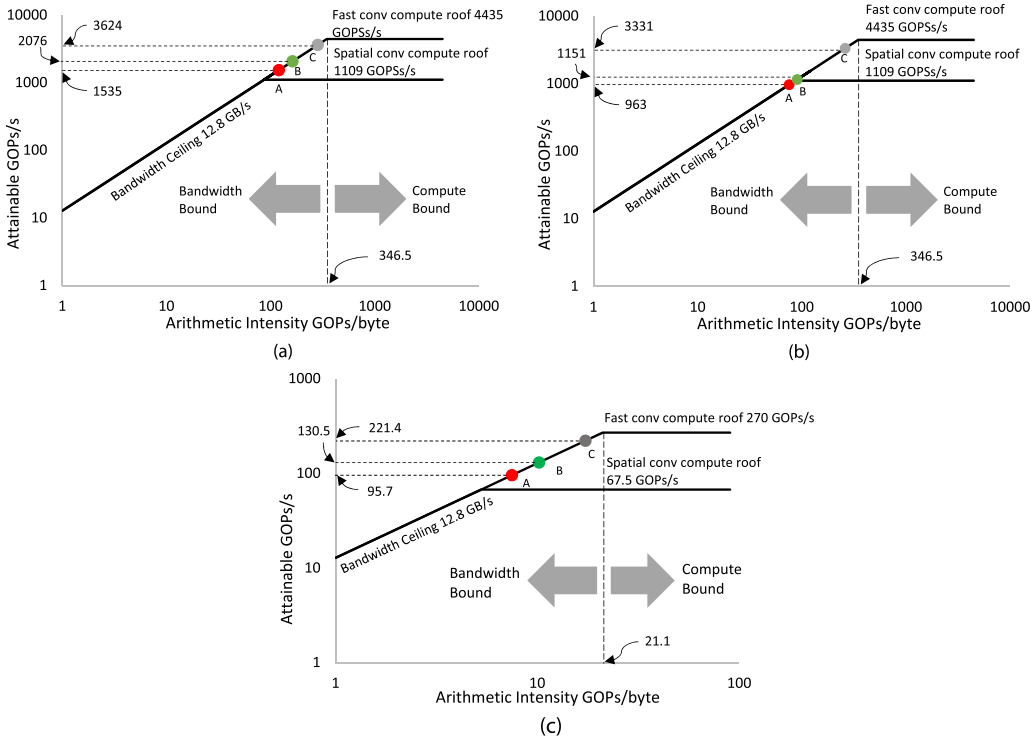


Fig. 5. Roofline model for (a) VGG16, (b) AlexNet, and (c) Shufflenet depicting compute and bandwidth ceilings for both fast-conv based and spatial-conv based CNNs.

smaller tiles need to be loaded in spatial convolution, although the peak achievable throughput for such a design would also be much lower due to higher number of multiplications required by spatial convolution. This results in a resource constrain on compute. To alleviate the high memory bandwidth requirement of our design, we optimize two sources of high bandwidth: data reuse and quantization.

*4.3.1 Kernel Map Reuse.* To minimize the amount of off-chip memory accesses for kernel data, we propose full kernel map reuse policy. In effect, the goal of this policy is to ensure that every kernel tile has to be loaded only once during a full forward pass. To enforce this policy, the channel of the feature map tile corresponding to the currently loaded kernel tile is fully convolved before the kernel tile is evicted and replaced. This ensures that the net amount of kernel data that needs to be loaded from the external memory is equal to the number of kernel weights in the CNN layers for each run of forward pass.

We also exploit kernel map reuse over batch size to amortize the cost of kernel bandwidth over multiple inputs. Every loaded kernel tile is fully reused until the current channel of the full batch of feature maps is convolved with the kernel tile. At higher batch sizes, the cost of kernel bandwidth is effectively negligible compared to the feature map bandwidth requirement and plateaus at roughly a batch size of 16.

*4.3.2 Feature Map Overlap Reuse.* Reference [20] reported that neighboring feature map tiles overlap by $r - 1 = 2$ units in both $H$ and $W$ dimensions. To exploit this overlap of input feature tiles, we utilize registers to temporarily cache $6 \times 2$ units of overlap of feature maps in the $W$ dimension

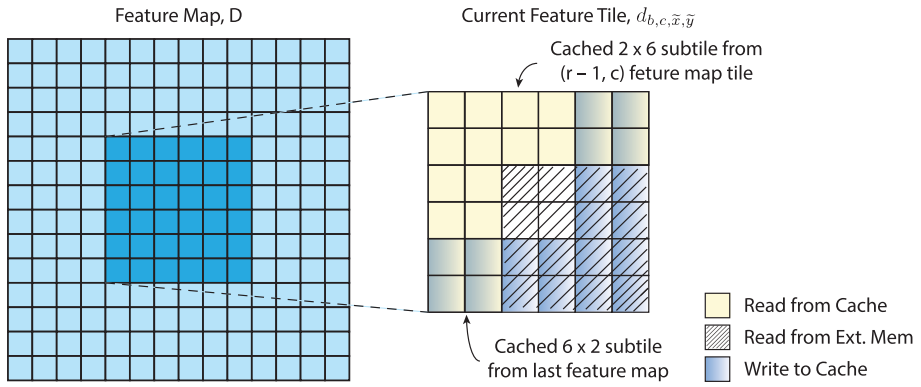


Fig. 6. Illustration showing optimal feature map reuse pattern, loading of second tile of the second row of feature map. Feature map tiles are processed in row major format.

and $2 \times 6$ units of overlap in the $H$ dimension. The cache size for caching overlap in $W$ dimension is only $6 \times 2$, since the cached overlap is utilized in each successive cycle owing to processing of feature maps in a row major format. The overlap cache size for $H$ dimension is proportional to layer width, hence layer with the largest feature map $H$, i.e., first layer, is utilized. Caching overlap of feature maps in the $W$ dimension ensures cache hits in overlap data in $W$ dimension for each feature map tile except the first tile of each row while the overlap in $H$ dimension ensures cache hits in all rows except the first row. This ensures a reasonable level of exploitation of temporal locality of the overlap data while putting less burden on external memory. Using feature map overlap, most external memory accesses would load $4 \times 4$ feature map tiles instead of $6 \times 6$. This corresponds to reduction in feature map bandwidth contribution by 2.25×. Figure 6 shows feature map reuse pattern employed in FFConv. Algorithm 1 shows a pseudocode of FFConv, employing both feature and kernel reuse using loop optimizations.

*4.3.3 Quantization Choice.* Quantization bitwidth is another parameter of optimization for reduction in memory bandwidth requirement. Based on our discussion in Section 3.2, we found that quantization schemes Q1.15 and Q12.0 for input features and kernel maps, respectively, result in minimal loss in accuracy of the system for the three CNNs. From the perspective of bandwidth, it is feasible to utilize bitwidths that are exponents of 2, since transfers to and from the external memory occur in exponent of 2 sized data chunks. While there is very little difference in accuracy for networks employing 18-bit kernel maps compared to 16-bit kernel maps, there is a sizable difference in bandwidth between these two configurations, since 18-bit kernels maps would need more than 2 bytes per transfer. However, we found the percentage loss in classification accuracy at 16-bit quantization (Q1.15 for feature maps and Q12.4 for kernel maps) to be 0.33%, 0.31%, and 0.52% for VGG16, AlexNet, and Shufflenet, respectively. 16-bit quantization (Q1.15/Q12.4) is hence justified, as it ensures efficient memory-bandwidth utilization while incurring insignificant loss in classification accuracy.

Since multiplication increases the bitwidth of the output relative to the inputs, we employ truncation after EWM stage to ensure the contribution of write bandwidth towards the net memory transfers is also kept at a minimum. Truncating the output to 16-bits (Q13.3) results in net accuracy loss of 0.43%, 0.47%, and 0.61% for VGG16, AlexNet, and Shufflenet, respectively.

*4.3.4 Attainable Performance.* Point A (red) on Figure 5 shows the expected performance with kernel map reuse (only over channels; batch size 1), no feature map reuse, 16-bit quantization for

**ALGORITHM 1:** Method stub for fast-conv using $P$ CEs, exploiting both feature overlap and kernel reuse

**Input**: $D, W$
**Output**: $Y$
**Result**: Input features $D$ convolved with kernels $W$ using $P$ CEs and fast-conv

```
for c ← 0 to C do
    for k ← 0 to K by P do
        forall k̃ ← k ... k + P do
            g_{k̃,c} ← W_{k̃,c}                          // Load P kernel tiles in parallel
            V_{k̃,c} ← G g_{k̃,c} G^T
        /* Use loaded kernels over current feature channel of entire batch        */
        for b ← 0 to batch_size do
            for x̃ ← 0 to H do
                for ỹ ← 0 to W do
                    overlap ← cache                     // Load cached feature map subtile
                    d_{b,c,x̃,ỹ} ← concat(overlap, D_{b,c,x̃,ỹ})        // Use loaded subtile
                    cache ← extract_overlap(d_{b,c,x̃,ỹ})      // Cache subtile of new map
                    U_{b,c,x̃,ỹ} ← B^T d_{b,c,x̃,ỹ} B
                    forall k̃ ← k ... k + P do
                        M_{b,k̃,c,x̃,ỹ} ← U_{b,c,x̃,ỹ} ⊙ V_{k̃,c}
                        Y_{b,k̃,c,x̃,ỹ} ← A^T M_{b,k̃,c,x̃,ỹ} A
```

both features and kernels, and no truncation of EWM output. Point B (green) shows the expected performance with both kernel map (batch size 1) and feature map overlap reuse, 16-bit quantization for features, and kernels but no truncation at EWM output. Point C (grey) shows performance with all optimizations employed; both kernel (batch size 16) and feature map reuse, 16-bit quantization for input features, kernels and truncation of EWM output (Q13.3), a configuration we employed in our implementation and will use in all future discussions. The net accuracy loss for this configuration is 0.43%, 0.47%, and 0.61%, respectively. Please note that all three of these points correspond to a system utilizing $P = 64$, where $P$ is the number of CEs instantiated in parallel.

The difference in attained performance for points B and C is highest in AlexNet, since it benefits the most from kernel map reuse over batch size, as all the three evaluated layers have small dimensions $H = W = 13$, leading to low computation to communication (CTC) ratio for these layers. Kernel map reuse over batch size significantly improves CTC ratio and hence performance.

Although the chosen configuration still faces a bandwidth bottleneck, the achieved performance is much higher compared to that possible using an unoptimized system and also to the compute ceiling of a system based on spatial convolution.

### 4.4 FFConv Architecture

Figure 7 shows the complete architecture of our accelerator system, FFConv. The system loads an input feature map tile $d_{b,c,\tilde{x},\tilde{y}}$ from external memory where for the first clock cycle, only $d_{0,0,0,0}$ is loaded, which denotes the first channel of the first $6 \times 6$ tile in batch 0. Each of the 64 CEs load its own kernel tile from the off-chip memory, $g_{k,c}$, where in the first cycle, kernel tiles ranging from $g_{0,0}$ to $g_{63,0}$ are loaded. Data and filter transforms then occur in parallel after which the CE computes the EWM output. In each subsequent cycle, tile from the same channel in the same batch

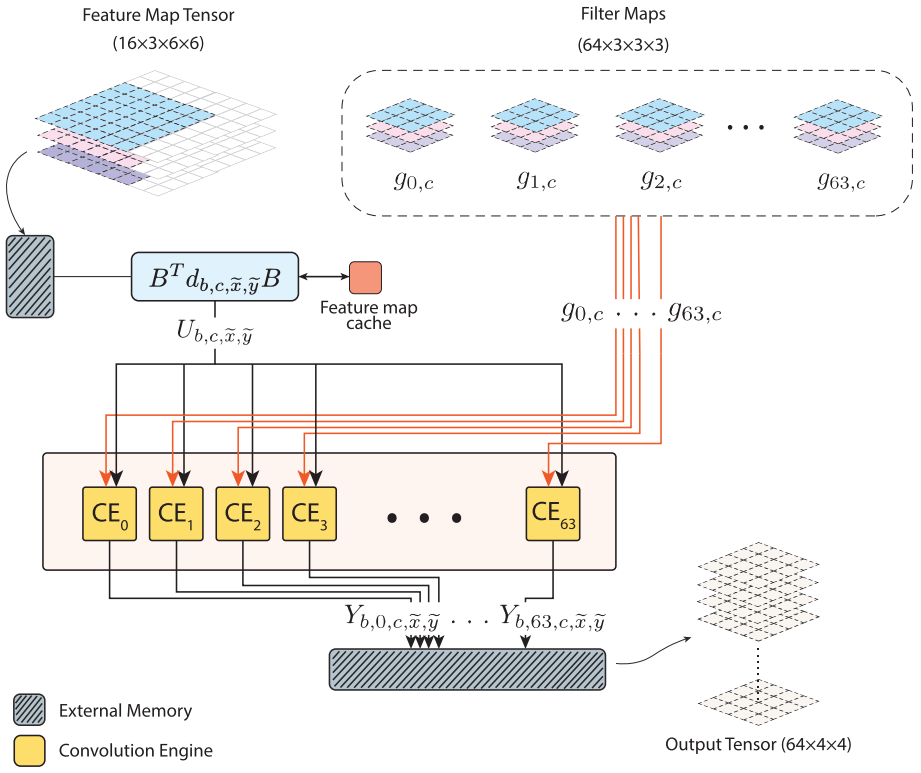


Fig. 7. FFConv system architecture with convolution core having 64 CEs. Each CE contains a filter transform, EWM, and an inverse transform stage.

Table 2. Qualitative Feature Comparison of FPGA-based Fast-conv Implementations

| | [1] | [23] | [25] | [2] | FFConv |
|---|---|---|---|---|---|
| $m, r$ | 4,3 | 4,3 | 2,3 | 4,3[a] | 4,3 |
| Quantization | No | Limited | No | Limited | Yes |
| Multipliers Per DSP | 0.25 | 1 | 0.25 | 2[b] | 1 |
| Accuracy Measurement | No | No | No | Yes | Yes |
| Stage Level Optimizations | No | Limited | No | Limited | Yes |

[a]1D fast-conv, $F(m, r)$.
[b]DSP blocks in most Intel FPGAs allow such a configuration.

index is loaded until the current channel of the batch index is computed. Same channel of the next batch index is processed next for kernel map reuse over batch size. The loaded kernel tiles are retained until the entire channel of the full batch is processed. The system then loads the next set of kernel tiles. In each clock cycle, 64 output tiles, $Y_{b,k,c,\tilde{x},\tilde{y}}$ are computed.

### 4.5 Qualitative Comparison with State-of-the-art

In this subsection, we will highlight the novel design and implementation choices that resulted in a better performance w.r.t. existing FPGA-based fast-conv implementations in the literature. Table 2 shows a qualitative comparison of features of different fast-conv implementations on

FPGAs. For quantization, "limited" refers to the fact that the previous works give limited information on the breakdown of components of fixed-point data-types. Stage-level optimizations refer to improvements in design at stage-level. This encapsulates improvements in resource utilization, power efficiency, and exploration of memory-compute tradeoffs.

*4.5.1 Quantization and Accuracy.* To the best of our knowledge, we are the first to investigate the variations in classification accuracy with the level of quantization of fixed-point feature and kernel maps in fast-conv–based designs. This is our first contribution, as we treated each of these variables separately, measuring the loss in accuracy with quantization bitwidths, while previous works used the same quantization for both feature and kernel maps for simplicity [1, 2, 23, 25]. While we evaluate custom bitwidth quantization schemes, previous works have used either half- or full-precision fixed-point with complete disregard to its impact on accuracy. Moreover, they give limited information on the distribution of fixed-point bitwidths between integer and fractional portions [1, 23, 25]. Some previous works have also attempted full-precision floating-point implementations that resulted in significantly degraded performance [40].

*4.5.2 Improvement in Throughput.* Our main improvement in throughput comes from maximizing the utilization and efficiency of resources while exploring memory limitations of our design and employing optimizations aimed at reducing bottlenecks. Our DSPs are completely utilized in every clock cycle, performing one multiplication per DSP per clock cycle. While some previous works have designed architectures for single-precision accelerators for simplicity and accuracy [1, 25], their designs are severely bottlenecked by the inefficient utilization of DSP resources. Furthermore, $F(4 \times 4, 3 \times 3)$ gives more savings in multiplication complexity compared to $F(4, 3)$ [2] and $F(2 \times 2, 3 \times 3)$ [25], leading to higher throughput, albeit at a high bandwidth cost. Reference [2] used DSP blocks on an Intel Arria 10 FPGA platform in a two multipliers per DSP configuration, which gave them significant gains in throughput and DSP efficiency. Although our design is still bandwidth-bound despite employing a number of optimizations, there is potential for further performance improvements using FPGA platforms with faster off-chip memory accesses. Most modern high-end FPGAs use DRR4 memory, which achieves over 18 GB/s uni-directional bandwidth. Lowering quantization using retraining could also help negate bandwidth limitations.

*4.5.3 Improvement in Resource Efficiency.* Custom bitwidth quantization significantly saves the logic resources required to implement the transform stages. Truncation after EWM stage allows our design to scale with reduced logic resources per CE compared to a design without truncation. Despite implementing data and filter transforms online leading to higher logic cost per CE, our design is unconstrained by the logic resources and DSPs but by the bandwidth. High DSP efficiency is resultant from quantization of feature and kernel maps to utilize one DSP per multiplier compared to full-precision implementation, which requires four DSPs per multiplier [1, 25].

*4.5.4 Improvement in Power Efficiency.* Quantization and efficient data reuse leads to a simpler and more power-efficient design. Quantization makes our transform stages sparse, utilizing lower number of logic resources, leading to lower power consumption. Truncation of the output to compute inverse transform on lower precision data also improves the power efficiency of our design as opposed to full precision inverse transform implementations [1, 23, 25].

## 5 EXPERIMENTS AND RESULTS

In this section, we present a quantitative analysis of the improvements discussed in previous section on four design metrics: accuracy, throughput, resource, and power efficiency. While the accuracy of a CNN architecture depends on the network choice itself, optimization steps—such as adding custom bitwidth quantization, performing truncation, and algorithmic changes—result in

changes in classification accuracy. For computing the $r = 1$ convolutions in Shufflenet, we transform feature and kernel maps such that an equivalent $r = 3$ convolution is possible, albeit suboptimally. Kernel maps are transformed from $1 \times 1 \times C$ to $3 \times 3 \times \frac{C}{9}$ maps where each 9 units along the channels are reshaped into $3 \times 3$ tiles. The feature maps are transformed in a similar way from $H \times W \times C$ to $3H \times 3W \times \frac{C}{9}$. The reshaped feature and kernel maps are stride 3 convolved and output of size $H \times W$ is obtained, which is equivalent to that obtained using $r = 1$.

The numbers discussed for throughput, resource, and power efficiency are post-implementation results of Xilinx Virtex 7 VC707 evaluation platform for conv layers of VGG16, AlexNet, and Shufflenet. Power consumption numbers are estimated using the Xilinx Power Estimator (XPE). Deep pipelining allows our design to meet timing constraints with maximum allowable frequency of 236.1 MHz, hence we operate our FPGA at the base oscillator frequency of 200 MHz.

### 5.1 Accuracy

Our fast-conv layer incurs loss in classification accuracy not only due to quantization of feature and kernel map data but also due to subsequent truncation of the EWM stage output, $M_{b,k,c,\tilde{x},\tilde{y}}$. To measure the net losses in accuracy due to the quantization and truncation, we used our fast-conv VGG16, AlexNet, and Shufflenet architectures implemented in software on an High Performance Computing (HPC) cluster integrating 48, 12-Core Intel® Xeon® E5-2680 v3 physical processors, 128 Gigabytes of physical memory, and an NVIDIA® Tesla K40M GPU. We also synthesized and simulated individual layer hardware designs on the same cluster using Icarus Verilog [32] for compilation and GTKWave [4] for simulation and functional verification of our fast-conv accelerator. We ran accuracy tests on the same subset of the Imagenet 2012 test dataset [11] as used in Section 3.2 to find the net loss in Top-1 classification accuracy.

For our chosen set of quantization and truncation schemes, Q12.4 for data transform, Q1.15 for filter transform output, and Q13.3 for inverse transform, we found the average % loss in Top-1 classification accuracy to be 0.43% for VGG16, 0.47% for AlexNet, and 0.61% for Shufflenet. Shufflenet loses the most accuracy, since its already low-complexity layers afford less leeway for quantization, as they hold important learned representations that are more prone to precision loss.

### 5.2 Throughput

As previously discussed, quantization of feature and kernel maps allows us to fit the design into one DSP per multiplier configuration. This enables us to instantiate more CEs in parallel. Previous works either implemented full-precision CEs that utilize more DSPs per multiplier [1, 25] or used half-precision implementations without regard to loss in accuracy [23]. Our custom precision implementation, utilizing an appropriate mix of quantization and truncation at different stages, maps the design with the highest efficiency to the FPGA platform for better utilization of resources, improved power efficiency, and an acceptable compromise in classification accuracy.

Exploring the limitations of our design in terms of memory-compute tradeoff helped point us towards optimizations such as kernel and feature data reuse, and further quantization that helped improve our performance by alleviating the high bandwidth requirements of fast-conv accelerators.

*5.2.1 VGG16.* All layers of VGG16 utilize $r = 3$, which enables excellent mapping of the conv layers onto our accelerator. Our system computes all five conv group layers of VGG16 in 8.47 ms compared to 10.08 ms of Reference [23] and 28.05 ms of Reference [1] and 163.4 ms of Reference [26] as shown in Table 3. This translates into speedups of 1.19×, 3.31×, and 19.3×, respectively. Reference [1] designed CEs for full-precision inputs, which lowered the number of parallel Processing Elements (PEs) that could be instantiated with the available DSP resources, resulting in a lower throughput. Overall, our accelerator system deals efficiently with the exorbitant amount

Table 3. Performance Comparison for VGG16-D

| | [26] | [1] | [23] | FFConv |
|---|---|---|---|---|
| $m, r$ | - | 4,3 | 4,3 | 4,3 |
| Number of DSPs | 780 | 2,736 | 2,520 | 2,304 |
| Number of LUTs | 182.62K | 107.84K | 600K | 337.3K |
| Data precision (bits) | 16 | 32 | 16 | 16 |
| Frequency (MHz) | 150 | 200 | 200 | 200 |
| Inference Time (ms) | 163.4 | 28.05 | 10.08 | 8.47 |
| Throughput (GOPS/s) | 187.8 | 1,094.3 | 3,044.7 | 3,623.9 |
| DSP efficiency (GOPS/s/DSP) | 0.241 | 0.400 | 1.208 | 1.573 |
| Slice LUT efficiency (GOPS/s/1000LUTs) | 1.03 | 10.15 | 5.08 | 10.74 |
| Power (W) | 9.63 | 36.32 | 23.60 | 26.91 |
| Power efficiency (GOPS/s/W) | 19.50 | 30.13 | 124.60 | 134.67 |

Table 4. Performance Comparison for AlexNet

| | [40] | [23] | [2] | FFConv |
|---|---|---|---|---|
| $m, r$ | - | 4,3 | 4,3[a] | 4,3 |
| Number of DSPs | 2,240 | 2,520 | 1,476[b] | 2,304 |
| Number of LUTs | 303.56K | 600K | 153.75K | 337.3K |
| Data precision (bits) | 32 | 16 | 16 | 16 |
| Frequency (MHz) | 100 | 200 | 303 | 200 |
| Inference Time (ms) | 8.600 | 0.4757 | 0.3526 | 0.2029 |
| Throughput (GOPS/s) | 78.23 | 1,414.3 | 1,908.1 | 3,331.6 |
| DSP efficiency (GOPS/s/DSP) | 0.035 | 0.561 | 1.294 | 1.446 |
| Slice LUT efficiency (GOPS/s/1000LUTs) | 0.258 | 2.36 | 12.41 | 9.88 |
| Power (W) | 16.81 | 23.6 | 45.0 | 26.91 |
| Power efficiency (GOPS/s/W) | 4.654 | 59.93 | 42.40 | 123.81 |

[a]1D Fast-conv, $F(m, r)$.
[b]Two $18 \times 18$ multipliers per DSP.

of compute and bandwidth requirements of VGG16, given the network utilizes $r = 3$ convolutions exclusively.

*5.2.2 AlexNet.* Our design of FFConv is only applicable to the last three conv layers of AlexNet, since layers 1 and 2 use kernel sizes 11 and 5, respectively. Hence, we only measure and compare the performance of these layers to previous works in Table 4. Latency and throughput numbers considered for all works in Table 4 are only for these three layers.

Our implementation takes 0.202 ms to compute the output of the three conv layers compared to 0.353 ms, 0.476 ms, and 8.600 ms taken by Reference [2], Reference [23], and Reference [40], respectively. This translates into speedups of 1.75×, 2.36×, and 42.59× compared to the respective works. The design of Reference [2] was clocked at 303 MHz and utilized DSP blocks configured in 2 multipliers per DSP mode, resulting in higher throughput compared to Reference [23]. Reference [2] implemented their accelerator system using 1D fast-conv $F(4, 3)$ to make fast-conv applicable on all layers of AlexNet, albeit sub-optimally on Conv 1 and 2 due to different kernel sizes. At an algorithmic level, the savings in multiplication complexity from utilizing 2D fast-conv relative to spatial-conv are exponent of 2 of savings from utilizing 1D fast-conv, which results in proportionally higher throughput. The architecture of Reference [40] used an operating frequency

Table 5. Performance Comparison for Shufflenet

| | [13] | [12] | FFConv |
|---|---|---|---|
| CNN | MobileNets V2[a] | DS-CNN | Shufflenet[b] |
| Complexity (MOPs) | 800 | 1.5 | 137 |
| Frequency (MHz) | 100 | 180 | 200 |
| Number of DSPs | 728 | 712[c] | 2,304 |
| Number of LUTs | 148.5K | 534.0K | 337.3K |
| Data precision (bits) | 8 | 16 | 16 |
| Overall latency (ms) | 15.4 | 0.016 | 0.62 |
| Throughput (GOPs/s) | 51.9 | 98.9 | 221.4 |
| DSP efficiency (GOPs/s/DSP) | 0.071 | 0.139 | 0.096 |
| Slice LUT efficiency (GOPs/s/1000LUTs) | 0.349 | 0.185 | 0.656 |
| Power (W) | 9.72 | 8.52 | 26.91 |
| Power efficiency (GOPs/s/W) | 5.3 | 11.6 | 8.2 |

[a]MobileNets-V2 SSDLite.
[b]Pointwise convolutions transformed to $r = 3$.
[c]Two $18 \times 18$ multipliers per DSP.

of 100 MHz, which severely decreased their achievable throughput. Furthermore, they designed their architecture for `float32` data-types, which reduced the number of parallel compute elements that they could instantiate with the available DSP resources.

*5.2.3 Shufflenet.* Due to different conv methodologies employed in Shufflenet compared to VGG16 and AlexNet, we compare our Shufflenet results with DS-CNN [12] and MobileNets V2 [13], FPGA accelerators. These designs utilize efficient convolution methodologies such as depthwise and pointwise convolutions similar to Shufflenet and are more appropriate for a fair comparison. The results are tabulated in Table 5. Owing to the fact that over 85% of the conv complexity in Shufflenet is attributed to pointwise convolutions ($r = 1$), we achieve a much lower throughput compared to that of our VGG16 due to a sub-optimal mapping of $r = 1$ to our accelerator. Despite this inefficient mapping, we achieve 2.24× the throughput of Reference [12] and 4.17× the throughput of Reference [13] owing to our more optimized architecture and better reuse of data.

### 5.3 Resource Efficiency

We utilize quantization and truncation at each stage to keep our data bitwidths low to alleviate bandwidth utilization. This leads to sparse transform stages that utilize less resources and also give high throughput but at a minor accuracy loss. The logic cost of our accelerator scales with the number of CEs due to both filter and inverse transforms, since they are replicated for each CE while a single data transform stage feeds data to all the CEs. Our current design is memory-constrained, hence implementing on a platform with faster off-chip accesses such as a Xilinx Ultrascale+ with a DDR4 memory would achieve a higher throughput for similar resource utilization.

*5.3.1 VGG16.* For VGG16, we utilize 0.914× the DSPs and 0.562× the LUTs to achieve 1.19× the throughput compared to Reference [23]. Reference [1] implemented full-precision conv layers that utilized 4 DSPs per multiplier, hence giving very low DSP efficiency. Owing to efficient utilization of DSPs due to custom bitwidth quantization, we achieve 1.30×, 3.93×, and 6.53× more DSP efficiency compared to Reference [23], Reference [1], and Reference [26], respectively. In slice LUT utilization, we achieve 10.74 compared to 5.08, 10.15, and 1.03 GOPS/s/1000LUTs achieved by

Reference [23], Reference [1], and Reference [26], respectively. This translates into 2.11×, 1.06×, and 10.43× better slice LUT efficiency of our design compared to the three designs for VGG16.

*5.3.2 AlexNet.* Since we utilize the same system to compute $r = 3$ conv layers of both VGG16 and AlexNet, our resource utilization is same, although the resource efficiency is different due to throughput difference originating from different bandwidth constraints. AlexNet is more bandwidth-constrained, since its small layer dimensions $H$ and $W$ reduce the CTC ratio reducing achieved throughput. Reference [2] configured DSPs in 2 multipliers per DSP configuration, achieving roughly 2.35× the DSP efficiency compared to the AlexNet implementation of Reference [23]. Furthermore, the transform stages of 1D fast-conv are sparser compared to those of 2D fast-conv, resulting in lower slice LUT utilization for Reference [2]. Reference [40] implemented a `float32` accelerator, which utilized high DSP and slice LUT resources and achieved poor throughput. Overall, we achieve 1.12×, 2.58×, and 41.31× better DSP efficiency, and 0.80×, 4.19×, and 38.29× better slice LUT efficiency compared to Reference [2], Reference [23], and Reference [40], respectively.

*5.3.3 Shufflenet.* The authors of Reference [12] implemented an accelerator for a custom low-complexity CNN architecture, DS-CNN. For Shufflenet, we achieve 31% lower DSP efficiency compared to DS-CNN [12], since DS-CNN utilized an Intel FPGA incorporating DSP blocks with the ability to be configured in two 18-bit multipliers per DSP configuration. However, we achieve 3.55× and 1.88× better slice LUT efficiency compared to Reference [12] and Reference [13], respectively, owing to lower resource utilization due to singular data transform and bandwidth-aware quantization.

### 5.4 Power Efficiency

Optimization of memory-compute tradeoff leads to a more power-efficient design, since we minimized off-chip memory transfers for reducing bandwidth requirements. This includes improving data reuse to minimize memory accesses and quantization to reduce the required transfers. Truncation for reducing the bitwidth of the output of EWM stage also helped improve power efficiency, since lower bitwidth transform stages require less logic resources to implement.

*5.4.1 VGG16.* For VGG16, owing to our customized design and efficient mapping of $r = 3$ kernels, we achieved power efficiency of 134.7 GOPS/s/W, which significantly supersedes previous works. Reference [1] implemented a 32-bit CE, which decreased the throughput significantly due to DSP resources becoming a bottleneck. Since Reference [1] designed a system that employed low number of parallel PEs, their logic resource utilization was low while employing higher DSP resources than our design. Owing to quantization, truncation, and efficient data reuse, we achieve 1.08×, 4.48×, and 6.91× better power efficiency compared to Reference [23], Reference [1], and Reference [26], respectively.

*5.4.2 AlexNet.* Though the relative power efficiency of FFCONV for AlexNet is lesser than VGG16 due to lower throughput, owing to better data reuse and hence minimal off-chip memory accesses, we achieve an overall 2.92×, 2.07×, and 26.60× better power efficiency compared to Reference [2], Reference [23], and Reference [40], respectively.

*5.4.3 Shufflenet.* For Shufflenet, we achieve 29% less power efficiency than DS-CNN. Two factors contribute toward a higher power efficiency of DS-CNN: (i) their target FPGA is using a 20nm process technology compared to 28nm of our FPGA, and (ii) the two multipliers per DSP configuration Intel FPGA deployed in their approach also makes it more power-efficient. In comparison,

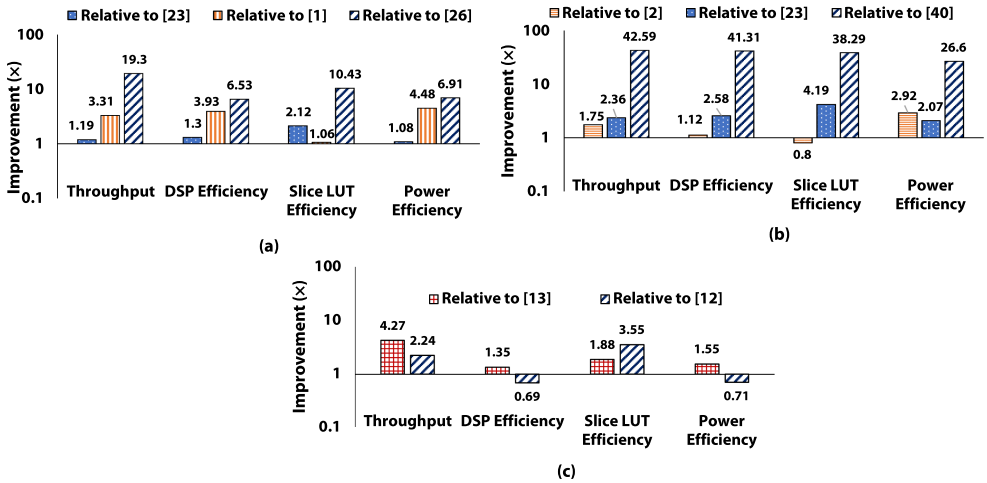


Fig. 8. Performance improvements provided by FFConv for (a) VGG16, (b) AlexNet, and (c) Shufflenet.

we achieve 1.55× better power efficiency than the design presented in Reference [13], which is using a similar Xilinx FPGA to ours.

The overall improvements resulted by our proposed hardware accelerator, FFConv, compared with the state-of-the-art implementations are summarized in Figure 8 for VGG16, AlexNet, and Shufflenet for all four performance metrics.

## 6 CONCLUSION

In this article, we presented FFConv, an FPGA-based convolutional layer accelerator that is based on fast-conv and is designed to utilize resources efficiently while improving performance. We explored memory bandwidth-compute tradeoffs, optimized and constrained our design to the FPGA platform of our choice to achieve significantly high throughput, resource, and power efficiency. We explored quantization and truncation at different stages of the design, optimizing memory bandwidth requirements while achieving an acceptable compromise in classification accuracy. Efficient data reuse schemes were utilized for both feature and kernel maps, resulting in high computation to communication (CTC) ratio. We benchmarked our design using three popular CNN architectures, VGG16-D, AlexNet, and Shufflenet, achieving 1.19× to 42.59× the throughput, 0.69× to 41.31× DSP efficiency, and 0.71× to 26.6× power efficiency for different networks compared to the previous state-of-the-art designs.

## ACKNOWLEDGMENTS

The authors thank LUMS HPC Cluster for compute support for accuracy testing and functional verification.

## REFERENCES

[1] A. Ahmad and M. A. Pasha. 2019. Towards design space exploration and optimization of fast algorithms for convolutional neural networks (CNNs) on FPGAs. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'19)*. IEEE, 1106–1111.

[2] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C. Ling, and Gordon R. Chiu. 2017. An OpenCL™deep learning accelerator on Arria 10. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, 55–64.

[3] Andrew Boutros, Sadegh Yazdanshenas, and Vaughn Betz. 2018. You cannot improve what you do not measure: FPGA vs. ASIC efficiency gaps for convolutional neural network inference. *ACM Trans. Reconfig. Technol. Syst.* 11, 3, Article 20 (Dec. 2018).

[4] Tony Bybell. 2010. GtkWave Electronic Waveform Viewer. http://gtkwave.sourceforge.net/.

[5] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 2017. Deep learning with low precision by half-wave Gaussian quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*. IEEE, 5406–5414.

[6] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient primitives for deep learning. *CoRR* abs/1410.0759 (2014).

[7] Soumith Chintala. 2017. convnet-benchmarks. *Technical Project (GitHub)*. https://github.com/soumith/convnet-benchmarks.

[8] Jason Cong and Bingjun Xiao. 2014. Minimizing computation in convolutional neural networks. In *Proceedings of the International Conference on Artificial Neural Networks and Machine Learning (ICANN'14)*, Stefan Wermter, Cornelius Weber, Włodzisław Duch, Timo Honkela, Petia Koprinkova-Hristova, Sven Magg, Günther Palm, and Alessandro E. P. Villa (Eds.). Springer International Publishing, Cham, 281–290.

[9] Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training deep neural networks with weights and activations constrained to +1 or −1. *CoRR* abs/1602.02830 (2016).

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024* 2015 (2015).

[11] J. Deng, W. Dong, R. Socher, and L. Li. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*. IEEE, 248–255.

[12] Wei Ding, Zeyu Huang, Zunkai Huang, Li Tian, Hui Wang, and Songlin Feng. 2019. Designing efficient accelerator of depthwise separable convolutional neural network on FPGA. *J. Syst. Archit.* 97 (2019), 278–286. DOI: https://doi.org/10.1016/j.sysarc.2018.12.008

[13] H. Fan, S. Liu, M. Ferianc, H. Ng, Z. Que, S. Liu, X. Niu, and W. Luk. 2018. A real-time object detection accelerator with compressed SSDLite on FPGA. In *Proceedings of the International Conference on Field-Programmable Technology (FPT'18)*. IEEE, 14–21. DOI: https://doi.org/10.1109/FPT.2018.00014

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. 2016. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 38, 1 (Jan. 2016), 142–158.

[15] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. In *Proceedings of the 4th International Conference on Learning Representations (ICLR'16)*. http://arxiv.org/abs/1510.00149.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. IEEE, 770–778.

[17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR* abs/1704.04861 (2017).

[18] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. *CoRR* abs/1405.3866 (2014).

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *ACM Commun.* 60, 6 (May 2017), 84–90.

[20] Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. IEEE, 4013–4021.

[21] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2014. Fully convolutional networks for semantic segmentation. *CoRR* abs/1411.4038 (2014).

[22] L. Lu, Y. Liang, Q. Xiao, and S. Yan. 2017. Evaluating fast algorithms for convolutional neural networks on FPGAs. In *Proceedings of the IEEE 25th International Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*. IEEE, 101–108. DOI: https://doi.org/10.1109/FCCM.2017.64

[23] L. Lu, Y. Liang, Q. Xiao, and S. Yan. 2017. Evaluating fast algorithms for convolutional neural networks on FPGAs. In *Proceedings of the IEEE 25th International Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*. IEEE, 101–108.

[24] Hiroki Nakahara, Haruyoshi Yonekawa, Tomoya Fujii, and Shimpei Sato. 2018. A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'18)*. ACM, 31–40.

[25] A. Podili, C. Zhang, and V. Prasanna. 2017. Fast and efficient implementation of convolutional neural networks on FPGA. In *Proceedings of the IEEE 28th International Conference on Application-specific Systems, Architectures, and Processors (ASAP'17)*. IEEE, 11–18.

[26] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going deeper with embedded FPGA platform for convolutional neural network. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'16)*. ACM, 26–35.

[27] Dominik Scherer, Hannes Schulz, and Sven Behnke. 2010. Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors. In *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III (ICANN'10)*. Springer-Verlag, 82–91.

[28] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *CoRR* abs/1409.1556 (2014).

[29] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'16)*. ACM, 16–25.

[30] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*. IEEE, 1–9.

[31] Nicolas Vasilache, Jeff Johnson, Michaël Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. 2014. Fast convolutional nets with fbfft: A GPU performance evaluation. *CoRR* abs/1412.7580 (2014).

[32] Stephen Williams. 2006. Icarus Verilog. http://iverilog.icarus.com/.

[33] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (Apr. 2009), 65–76. DOI: https://doi.org/10.1145/1498765.1498785

[34] Shmuel Winograd. 1980. *Arithmetic Complexity of Computations*. Vol. 33. Siam, Philadelphia, PA.

[35] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. IEEE, 4820–4828.

[36] Q. Xiao, Y. Liang, L. Lu, and S. Yan. 2017. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs. In *Proceedings of the 54th ACM/EDAC/IEEE Design Automation Conference (DAC'17)*. IEEE, 1–6.

[37] C. Yang, Y. Wang, X. Wang, and L. Geng. 2018. A reconfigurable accelerator based on fast Winograd algorithm for convolutional neural network in Internet of Things. In *Proceedings of the 14th IEEE International Conference on Solid-state and Integrated Circuit Technology (ICSICT'18)*. IEEE, 1–3.

[38] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2016. Designing energy-efficient convolutional neural networks using energy-aware pruning. *CoRR* abs/1611.05128 (2016).

[39] J. Yu, Y. Hu, X. Ning, J. Qiu, K. Guo, Y. Wang, and H. Yang. 2017. Instruction driven cross-layer CNN accelerator with Winograd transformation on FPGA. In *Proceedings of the International Conference on Field Programmable Technology (ICFPT'17)*. IEEE, 227–230.

[40] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'15)*. ACM, 161–170.

[41] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong. 2018. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. *IEEE Trans. Comput.-aided Des. Integr. Circ. Syst.* 38, 11 (Nov. 2018), 2072–2085. DOI: https://doi.org/10.1109/TCAD.2017.2785257

[42] Jialiang Zhang and Jing Li. 2017. Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, 25–34.

[43] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*.

[44] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. 2017. Accelerating binarized convolutional neural networks with software-programmable FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. ACM, 15–24.

[45] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless CNNs with low-precision weights. In *Proceedings of the International Conference on Learning Representations, (ICLR'17)*.

Received March 2019; revised November 2019; accepted January 2020