Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks

Naveen Suda, Vikas Chandra[‡], Ganesh Dasika[*], Abinash Mohanty, Yufei Ma, Sarma Vrudhula[†], Jae-sun Seo, Yu Cao
School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, USA
[†]School of Computing, Informatics, Decision Systems Engineering, Arizona State University, Tempe, USA
[‡]ARM Inc., San Jose, USA; *ARM Inc., Austin, USA.
E-mail: {naveen.suda, abinash.mohanty, yufeima, vrudhula, jaesun.seo, yu.cao}@asu.edu, {vikas.chandra, ganesh.dasika}@arm.com

ABSTRACT

Convolutional Neural Networks (CNNs) have gained popularity in many computer vision applications such as image classification, face detection, and video analysis, because of their ability to train and classify with high accuracy. Due to multiple convolution and fully-connected layers that are compute-/memory-intensive, it is difficult to perform real-time classification with low power consumption on today's computing systems. FPGAs have been widely explored as hardware accelerators for CNNs because of their reconfigurability and energy efficiency, as well as fast turn-around-time, especially with high-level synthesis methodologies. Previous FPGA-based CNN accelerators, however, typically implemented generic accelerators agnostic to the CNN configuration, where the reconfigurable capabilities of FPGAs are not fully leveraged to maximize the overall system throughput. In this work, we present a systematic design space exploration methodology to maximize the throughput of an OpenCL-based FPGA accelerator for a given CNN model, considering the FPGA resource constraints such as on-chip memory, registers, computational resources and external memory bandwidth. The proposed methodology is demonstrated by optimizing two representative large-scale CNNs, AlexNet and VGG, on two Altera Stratix-V FPGA platforms, DE5-Net and P395-D8 boards, which have different hardware resources. We achieve a peak performance of 136.5 GOPS for convolution operation, and 117.8 GOPS for the entire VGG network that performs ImageNet classification on P395-D8 board.

Categories and Subject Descriptors

C.3 [**SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS**]: Signal processing systems.

Keywords

FPGA, OpenCL, Convolutional Neural Networks, Optimization.

FPGA'16, February 21–23, 2016, Monterey, CA, USA.
© 2016 ACM. ISBN 978-1-4503-3856-1/16/02…$15.00
DOI: http://dx.doi.org/10.1145/2847263.2847276

1. INTRODUCTION

Convolutional Neural Networks (CNNs), inspired by visual cortex of the brain, are a category of feed-forward artificial neural networks. CNNs, which are primarily employed in computer vision applications such as character recognition [1], image classification [2] [9] [16] [17], video classification [3], face detection [4], gesture recognition [5], etc., are also being used in a wide range of fields including speech recognition [6], natural language processing [7] and text classification [8]. Over the past decade, the accuracy and performance of CNN-based algorithms improved significantly, mainly due to the enhanced network structures enabled by massive training datasets and increased raw computational power aided by CMOS scaling to train the models in a reasonable amount of time.

A typical CNN architecture has multiple convolutional layers that extract features from the input data, followed by classification layers. The operations in CNNs are computationally intensive with over billion operations per input image [9], thus requiring high performance server CPUs and GPUs to train the models. However, they are not energy efficient and hence various hardware accelerators have been proposed based on FPGA [10]-[13], SoC (CPU + FPGA) [14] and ASIC [15]. FPGA based hardware accelerators have gained momentum owing to their reconfigurability and fast development time, especially with the availability of high-level synthesis (HLS) tools from FPGA vendors. Moreover, FPGAs provide flexibility to implement the CNNs with limited data precision which reduces the memory footprint and bandwidth requirements, resulting in a better energy efficiency (e.g. GOPS/Watt).

Previous FPGA-based CNN accelerator designs primarily focused on optimizing the computational resources without considering the impact of the external memory transfers [10] [11] or optimizing the external memory transfers through data reuse [12] [13]. The authors of [13] proposed a design space exploration methodology for CNN accelerator by optimizing both computation resources and external memory accesses, but implemented only convolution layers. In this work, we present a systematic methodology for maximizing the throughput of an FPGA-based accelerator for an entire CNN model consisting of all CNN layers: convolution, normalization, pooling and classification layers.

The key contributions of this work are summarized as follows:

- CNN with fixed-point operations are implemented on FPGA using OpenCL framework. Critical design variables that impact the throughput are identified for optimization.
- Execution time of each CNN layer is analytically modeled as a function of these design variables and validated on FPGA.

Logic utilization is empirically modeled using FPGA synthesis data for each CNN layer as a function of the design variables.

- A systematic methodology is proposed to minimize total execution time of a given CNN algorithm, subject to the FPGA hardware constraints of logic utilization, computational resources, on-chip memory and external memory bandwidth.
- The new methodology is demonstrated by maximizing the throughput of two large-scale CNNs: AlexNet [16] and VGG [17] (which achieved top accuracies in ImageNet challenges 2012 and 2014, respectively), on two Altera FPGA platforms with different hardware resources.

The rest of the paper is organized as follows. Section 2 briefly describes the operations of CNNs using AlexNet as an example. Section 3 presents the challenges in implementing a large-scale CNN on FPGAs. It also studies the impact of precision of the weights on the accuracy of AlexNet and VGG models. Section 4 briefly presents the OpenCL implementation of CNN layers and describes the design variables used for acceleration. Section 5 describes our proposed methodology for design space exploration to maximize the throughput of the CNN accelerator with limited FPGA resources. Section 6 presents the experimental results of two CNNs optimized on two Altera FPGA platforms and compares them with prior work. Section 7 concludes the paper.

2. BASIC OPERATIONS OF CNN

A typical CNN is comprised of multiple convolutional layers, interspersed by normalization, pooling and non-linear activation function. These convolution layers decompose the input image to different features maps varying from low-level features such as edges, lines, curves, etc., in the initial layers to high-level/abstract features in the deeper layers. These extracted features are classified to output classes by fully-connected classification layers that are similar to multi-layer perceptrons. For example, Figure 1 shows the architecture of AlexNet CNN [16], which won the ImageNet challenge in 2012. It consists of 5 convolutional layers each with a Rectified Linear Unit (ReLU) based activation function, interspersed by 2 normalization layers, 3 pooling layers and concluded by 3 fully connected layers which classify the input 224×224 color images to 1,000 output classes. The ImageNet database-based models are characterized by top-1 and top-5 accuracies, which represent whether the input image label matches with top-1 and top-5 predictions, respectively.

2.1 Convolution

Convolution is the most critical operation of CNNs and it constitutes over 90% of the total operations in AlexNet model [13]. It involves 3-dimensional multiply and accumulate operation of N_{if} input features with $K\times K$ convolution filters to get an output feature neuron value as shown in Equation (1).

$$out(f_o,x,y)=\sum_{f_i=0}^{N_{if}}\sum_{k_x=0}^{K}\sum_{k_y=0}^{K} wt(f_o,f_i,k_x,k_y)\times in(f_i,x+k_x,y+k_y) \quad (1)$$

where $out(f_o,x,y)$ and $in(f_i,x,y)$ represent the neurons at location (x,y) in the feature maps f_o and f_i, respectively and $wt(f_o,f_i,k_x,k_y)$ is the weights at position (k_x,k_y) that gets convolved with input feature map f_i to get the output feature map f_o.

2.2 Normalization

Local Response Normalization (LRN) or normalization implements a form of lateral inhibition [16] by normalizing each neuron value by a factor that depends on the neighboring neurons. LRN across neighboring features and within the same feature can be computed as shown in Equations (2) and (3), respectively.

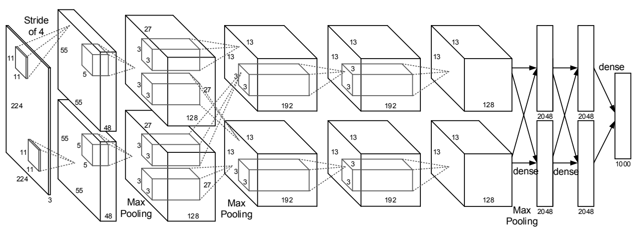


Figure 1: Architecture of AlexNet CNN [16].

$$out(f_o,x,y)=\frac{in(f_o,x,y)}{\left(1+\frac{\alpha}{K}\sum_{f_i=f_o-K/2}^{f_o+K/2} in^2(f_i,x,y)\right)^{\beta}} \quad (2)$$

$$out(f_o,x,y)=\frac{in(f_o,x,y)}{\left(1+\frac{\alpha}{K^2}\sum_{k_x=x-K/2}^{x+K/2}\sum_{k_y=y-K/2}^{y+K/2} in^2(f_o,x+k_x,y+k_y)\right)^{\beta}} \quad (3)$$

where K in Equation (2) is the number of feature maps used for LRN computation, K in Equation (3) is the number of neurons in x, y directions in the same feature, while α and β are constants.

2.3 Pooling

Spatial pooling or subsampling is utilized to reduce the feature dimensions as we traverse deeper into the network. As shown in Equation (4), pooling computes the maximum or average of neighboring $K\times K$ neurons in the same feature map, which also provides a form of translational invariance [18]. Although max-pooling is popularly used, average pooling is also used in some CNN models [18]. Reducing the dimensionality of lower-level features while preserving the important information, the pooling layer helps abstracting higher-level features without redundancy.

$$out(f_o,x,y)=\underset{0\le(k_x,k_y)<K}{\text{max/average}}\left(in(f_o,x+k_x,y+k_y)\right) \quad (4)$$

2.4 Activation Functions

The commonly used activation functions in traditional neural networks are non-linear functions such as tanh and sigmoid, which require a longer training time in CNNs [16]. Hence, Rectified Linear Unit (ReLU) defined as $y = max(x,0)$ has become the popular activation function among CNN models as it converges faster in training. Moreover, ReLU has less computational complexity compared to exponent functions in tanh and sigmoid, also aiding hardware design.

2.5 Fully Connected Layer

Fully-connected layer or inner product layer is the classification layer where all the input features (N_{if}) are connected to all of the output features (N_{of}) through synaptic weights (wt). Each output neuron is the weighted summation of all the input neurons as shown in Equation (5).

$$out(f_o)=\sum_{f_i=0}^{N_{if}} wt(f_o,f_i)\times in(f_i) \quad (5)$$

The outputs of the inner-product layer traverse through ReLU based activation function to the next inner-product layer or directly to a Softmax function that converts them to probability in the range (0, 1). The final accuracy layer compares the labels of the top probabilities from softmax layer with the actual label and gives the accuracy of the CNN model.

Table 1. Operations in AlexNet CNN model [16]

Layer	#Features	Feature dimensions		Stride	Kernel/weight dimensions				#Operations
		X	Y		#Output features	#Input features	X	Y	
Input image	3	224	224						
Convolution-1/ReLU-1	96	55	55	5	96	3	11	11	211120800
Normalization-1	96	55	55			5[a]			3194400
Pooling-1	96	27	27	2			3	3[b]	629856
Convolution-2/ReLU-2	256	27	27	1	256	48[c]	5	5	448084224
Normalization-2	256	27	27			5[a]			2052864
Pooling-2	256	13	13	1			3	3[b]	389376
Convolution-3/ReLU-3	384	13	13	1	384	256	3	3	299105664
Convolution-4/ReLU-4	384	13	13	1	384	192[c]			224345472
Convolution-5/ReLU-5	256	13	13	1	384	192[c]			149563648
Pooling-5	256	6	6	2			3	3[b]	82944
Fully connected-6/ReLU-6	4096				4096	9216			75501568
Fully connected-7/ReLU-7	4096				4096	4096			33558528
Fully connected-8	1000				1000	4096			8192000
Total Operations									1455821344

[a] Normalization across 5 neighboring channels.
[b] Max-pooling across 3x3 window.
[c] Convolution performed in 2 groups.

3. CNN MODEL STUDY AND FPGA DESIGN DIRECTIONS

3.1 FPGA Implementation Challenges

While CNNs are proven indispensable in many computer vision applications, they consume significant amount of storage, external memory bandwidth, and computational resources, which makes it difficult to implement on an embedded platform. The challenges in implementation of a large-scale CNN on FPGAs are illustrated using AlexNet model as an example. The different layers in AlexNet along with the number of features in each layer, feature dimensions, number of synaptic weights and the total number of operations in each layer are summarized in Table 1. It has over 60 million model parameters, which needs ~250MB of memory to store the weights using 32-bit floating point representation and hence they cannot be stored in on-chip memory of commercially available FPGAs. They need to be stored in an external memory and transferred to the FPGA during computation, which could become a performance bottleneck. The AlexNet model consists of 5 convolution layers, 2 LRN layers, 3 pooling layers and 3 fully connected layers, where each layer has different number of features and dimensions. If they are implemented independently without resource sharing, it would be either hardware-inefficient or may not fit on the FPGA due to the limited logic resources. The problem gets exacerbated in the state of the art models such as VGG [19] and GoogLeNet [9], which have a larger number of layers. To efficiently share hardware resources, repeated computation (e.g. convolution) should be implemented with a scalable hardware [13], such that the same hardware is reused by iterating the data through them in software.

The performance limitation due to the external memory bandwidth can be alleviated by using reduced precision model weights. Hence we performed a precision study by sweeping model weights and chose the precision values that have minimal impact on the classification accuracy.

3.2 Precision Study for FPGA Primitives

Traditionally CNN models are trained in CPU/GPU environments using 32-bit floating point data. Such high precision is not necessarily required in the testing or classification phase, owing to the redundancy in the over-parameterized CNN models [19]. Reducing data precision of the weights/data without any impact on the accuracy directly reduces the storage requirement as well as the energy for memory transfers.

Using AlexNet and VGG models, we explored the precision requirements of convolution and fully connected layer weights using Caffe tool framework [20]. We obtained the pre-trained models from Caffe, rounded the convolution weights and inner product weights separately, and tested the models using the ImageNet-2012 validation dataset of 50K images. Although data precision is reduced, Caffe tool still performs CNN operations in 32-bit floating point precision using the rounded-off weights. Figure 2 shows the top-1 and top-5 accuracies of the model for a precision sweep of the weights. It shows that the accuracy steeply drops if the weight precision reduces below 8 bits. We use a common precision for the weights in all convolution layers, as the same hardware block will be reused for all the convolution layer iterations. We choose 8-bit precision for the convolution weights and 10-bit precision for inner product weights, which degrades the accuracy by only <1% compared to full precision weights. Similarly, we choose 16-bit precision for the intermediate layer data by performing the precision study.

3.3 FPGA Accelerator Design Directions

In our FPGA design, we first developed computing primitives of CNNs using OpenCL framework. A scalable convolution module is designed based on matrix multiplication operation in OpenCL, so that it can be reused for all convolution layers with different input and output dimensions. Similarly, we developed scalable hardware modules for normalization, pooling, and fully-connected layers. We identified key design variables

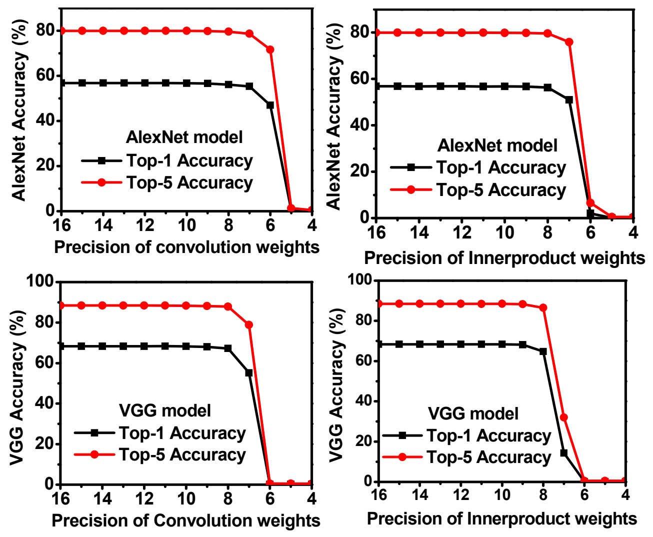


Figure 2: AlexNet and VGG model classification accuracies are shown for different weight precisions of convolution and inner-product layers.

such as loop-unroll factor and SIMD vectorization factor, which determine hardware parallelization and thus directly impact the throughput, external memory bandwidth requirement, and computational resource utilization.

Intuitively, assigning more computational resources to performance-critical operations in convolution and fully connected layers would maximize the overall throughput of the system. However, it may not be a global optimal solution, because each layer has different feature dimensions and the computational resources are limited. Hence, there is a great need for a design space exploration methodology that maximizes the throughput by optimally distributing the FPGA resources among various scalable CNN hardware blocks.

We propose a design space exploration framework based on both analytical and empirical models of CNN layer performance and resource utilization, to find the optimal values of the key design variables that maximize the throughput of a generic CNN model on a given FPGA board with limited computation resources, on-chip memory, and external memory bandwidth.

4. CNN LAYERS IN OPENCL

In this section, we briefly introduce the OpenCL framework, then present the implementation of the CNN layers in OpenCL and explain the key design variables that need to be optimized to maximize the overall throughput of the CNN accelerator.

4.1 OpenCL Framework

High Level Synthesis (HLS) tools are gaining popularity in the FPGA community, as they enable faster hardware development by automatically synthesizing an algorithm in high-level language (e.g. C) to RTL/hardware. There is a recent interest in using OpenCL, a C-based programming language, for FPGAs because of its parallel programming model [21] which matches with the parallel computation capabilities of FPGAs. Moreover, the same OpenCL codes can easily be ported to different platforms: CPUs, GPUs, DSPs or heterogeneous systems consisting of a combination of them. OpenCL compilers not only compile an OpenCL code to RTL, but also integrate it with the interfacing IPs for external memory and for communication between host CPU and FPGA accelerator board. They abstract the designer/user from the intricacies of traditional FPGA design flow such as RTL coding, integration with interfacing IPs and timing closure, which considerably reduces the design time, while achieving performance comparable to the traditional flow, but possibly at the expense of higher on-chip memory utilization [22]. The design flow of the OpenCL based FPGA accelerator for CNN used in this work is shown in Figure 3. It consists of an FPGA accelerator board that is integrated into the PCIe slot of a desktop CPU that acts as the OpenCL host. In general, OpenCL framework consists of two components (a) an OpenCL code that is compiled and synthesized to run on the FPGA accelerator and (b) a C/C++ based host code with vendor-specific application program interface (API) to communicate with the FPGA board.

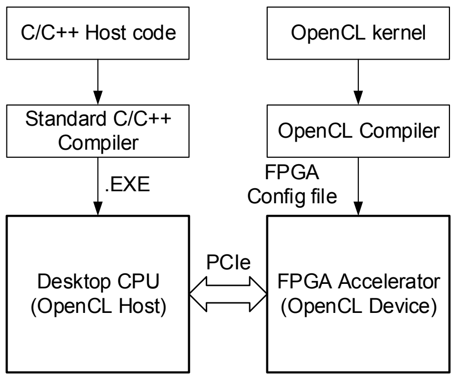


Figure 3: Design flow of OpenCL based FPGA accelerator for CNN.

In this work, we use Altera OpenCL software development kit (SDK) for compilation of OpenCL code to RTL, which takes a few minutes for initial compilation, followed by full synthesis which could take hours depending on the size of the design. The tool-kit provides support for emulation, which runs the OpenCL code on host CPU, thus allowing for quick functional verification before going to the full FPGA implementation. The Altera SDK for OpenCL provides different synthesis constructs to enable acceleration of OpenCL kernels such as loop unroll factor and Single-Instruction-Multiple-Data (SIMD) vectorization factor. The details about how these factors improve the performance of the OpenCL kernels and impact the logic utilization are discussed in the following sections.

4.2 3-D Convolution as Matrix Multiplication

Convolutions are the most performance-critical operations in CNNs, involving computationally intensive 3-D multiply and accumulate (MAC) operations of the input features with the convolution weights as given in Equation (1). To maximize the overall throughput of the accelerator and also make the design portable to any other CNN model, a scalable convolution block is needed such that the data can be iterated through it in software.

We implemented the scalable convolution block by mapping the 3-D convolutions as matrix multiplication operations similar to that in [23] by flattening and rearranging the input features. As an example, Figure 4 illustrates how Convolution-1 layer in AlexNet is mapped from 3 input features with dimensions 224×224 to a rearranged matrix with dimensions of (3×11×11) × (55×55). The input features from the first convolution window of 11×11 are flattened and arranged vertically as shown in Figure 4. Similarly, the entire rearranged matrix can be generated by sliding the 11×11 convolution filter across the input features. After input features are rearranged, the convolution operation transforms to a

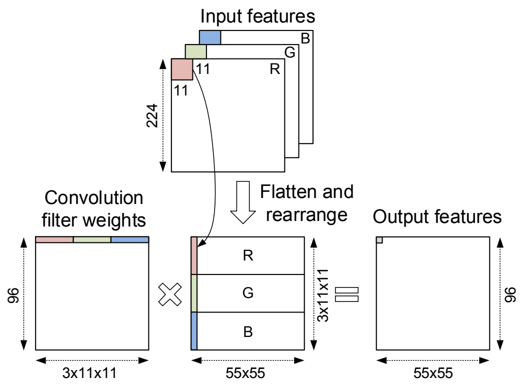


Figure 4: Mapping 3D convolutions to matrix multiplications.

```
1.  Get current work-item/thread identifiers (x, y).
2.  For each $N_{CONV}$ elements width-wise in weight matrix:
3.      Compute address locations for input features and weights.
4.      Fetch input features to inputs[x][y] in local memory.
5.      Fetch convolution weights to weights[y][x] in local memory.
6.      Wait till $N_{CONV} \times N_{CONV}$ inputs and weights are loaded.
7.      Do the following $N_{CONV}$ MAC operations in parallel:
8.          convolution output += weight[x][k]*input[y][k].
9.      Wait till all work-items complete computation on fetched data.
10. Save convolution output to output buffer.
```

Figure 5: Pseudo code for convolution implementation.

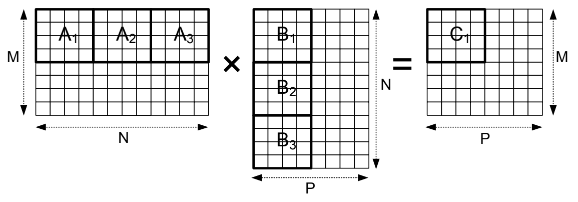
generic matrix multiplication operation. Note that we perform the input feature rearrangement on-the-fly by storing them in the FPGA on-chip memory before performing matrix multiplication, which reduces the external memory requirement by eliminating the need to store the entire rearranged input feature matrix.

The pseudo-code for matrix multiplication based convolution implementation in OpenCL is shown in Figure 5. It can be summarized as the following three basic operations which are repeated over each row of the weight matrix.

a) Fetch the convolution weights to the local memory which is implemented using FPGA on-chip memory.
b) Compute the input feature actual address locations before flattening and fetch them to local memory.
c) Compute N_{CONV} multiply and accumulate operations in parallel on the weights and inputs from local memory.

We utilized matrix multiplication OpenCL code from [24] and appended the input feature rearranging operation. Understanding the matrix multiplication OpenCL implementation is critical for acceleration of the convolution operation. The implementation of matrix multiplication operation in OpenCL is illustrated in Figure 6, which consists of convolution weight matrix A (M×N), multiplied by the rearranged input feature matrix B (N×P) to compute the output feature matrix C (M×P). It consists of $N_{CONV} \times N_{CONV}$ threads or OpenCL work-items, which fetch the first $N_{CONV} \times N_{CONV}$ inputs to the local memory where $N_{CONV}=4$ in this example. Each work-item performs N_{CONV} parallel multiply and accumulate (MAC) operations on the local memory data, which is accomplished by loop unrolling that replicates the hardware resources for acceleration. This process is repeated by sliding the $N_{CONV} \times N_{CONV}$ window column-wise in matrix A and row-wise in matrix B and performing the MAC operations to get $N_{CONV} \times N_{CONV}$ elements in the product matrix C.

From Figure 6, we see that the input and output matrix dimensions must be a multiple of N_{CONV}, which might not always be possible because of different number of input and output features and different feature dimensions in different convolution



$C_1 = A_1 \times B_1 + A_2 \times B_2 + A_3 \times B_3$

Figure 6: Accelerating matrix multiplications in OpenCL.

layers. Hence we use zero padding in the input matrices to make their dimensions a multiple of N_{CONV}. Increasing N_{CONV} boosts the throughput as it fetches larger number of inputs to the local memory and performs computations on them without having to wait for external data. On the other hand, it increases the logic utilization and execution time if the zero-padding is excessive in some layers.

We use SIMD vectorization factor (S_{CONV}), as another design variable to accelerate the convolution operation, which represents the factor by which computational resources are vectorized to execute in a Single-Instruction-Multiple-Data fashion. This factor improves the throughput by a factor of S_{CONV}. Depending on the model configuration parameters such as number of features and their dimensions and the number of CNN layers, choosing an appropriate combination of (N_{CONV}, S_{CONV}) maximizes the overall throughput of the CNN.

4.3 Normalization Layer

Local response normalization (LRN) implementation requires an exponent operation as shown in Equation (2), which is expensive to precisely implement in hardware. Hence we implement the exponent function $f_1(x_o)$ shown in Equation (6) using a piece-wise linear approximation function $pwlf(x_o)$.

$$out(f_o, x, y) = in(f_o, x, y).f_1(x_o) \tag{6}$$

$$f_1(x_o) = (1+x_o)^{-\beta};\ x_o = \frac{\alpha}{K}\sum_{f_i=f_o-K/2}^{f_o+K/2} in^2(f_i, x, y) \tag{7}$$

Here K represents the number of features used for normalization. Using the AlexNet model data as an example, the exponent function $f_1(x_o)$ is approximated using a piece-wise linear function using 20 points with a maximum error of 1%. Because of the wide dynamic range of values involved in x_i computation, normalization is implemented in 32-bit floating point representation. The exponent function and the piece-wise linear approximate function along with the approximation error are plotted in Figure 7. Normalization is implemented as a single-threaded OpenCL code using loop unroll factor (N_{NORM}), which represents the number of normalization operations it performs in a single cycle. The Altera OpenCL compiler automatically infers pipelining whenever there are no data dependencies between multiple iterations. The pseudo code for normalization is shown in

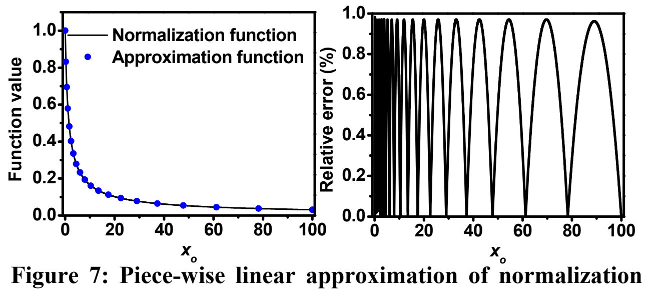


Figure 7: Piece-wise linear approximation of normalization operation kernel with a maximum error of 1%.

```
1. Compute sum_of_squares of first K/2 features.
2. For each input_feature i:
3.     For each neuron j in feature i:
4.         Do the following for $N_{NORM}$ neurons in parallel:
5.             Compute sum_of_squares[j] += input_feature[i+K/2][j]
6.             Compute output_feature[i][j] = input_feature[i][j]
7.                         *pwlf(α/K*sum_of_squares[j])
8.             Update sum_of_squares[j] –= input_feature[i–K/2][j]
```

Figure 8: Pseudo-code for normalization implementation.

Figure 8. It uses local memory to store the sum of squares of a sliding window of K input features, while performing the normalization operation on the computed sum of squares using the piece-wise linear approximation function, $pwlf(x_o)$.

4.4 Implementation of other Layers

Pooling is implemented using a single work-item kernel where acceleration is achieved by unrolling the loop to generate N_{POOL} parallel outputs in a single cycle. Fully-connected layer or inner-product layer is also implemented as single work-item kernel, where acceleration is achieved by performing N_{FC} parallel multiply and accumulate operations, which accelerates the performance by a factor of N_{FC}. Nonlinear activation function ReLU, which performs the function $y=max(x,0)$ is incorporated at the output of convolution and inner product implementations with a flag to enable or disable it.

5. DESIGN SPACE EXPLORATION

Choosing the best combination of the design variables (N_{CONV}, S_{CONV}, N_{NORM}, N_{POOL}, N_{FC}) that maximizes the performance of the CNN accelerator, while still being able to fit in the limited FPGA resources is a non-trivial task, which emphasizes the need for a systematic design space exploration methodology. Optimization framework that relies on full FPGA synthesis at each design point may not be feasible especially because of the long run time, which could take hours, or potential synthesis failures that occur due to utilization of hardware resources. Hence we model the performance and resource utilization and use them for fast design space exploration.

In this section, we first formulate the optimization problem and present the analytical and empirical modeling of the performance and FPGA resource utilization as a function of the design variables for each CNN layer.

5.1 Problem Formulation

The resource-constrained throughput optimization problem can be formulated as follows.

$$\text{Minimize} \sum_{i=0}^{TL} runtime_i\,(N_{CONV}, S_{CONV}, N_{NORM}, N_{POOL}, N_{FC}) \quad (8)$$

$$\text{Subject to} \sum_{j=0}^{L} DSP_j \le DSP_{MAX} \quad (9)$$

$$\sum_{j=0}^{L} Memory_j \le Memory_{MAX} \quad (10)$$

$$\sum_{j=0}^{L} Logic_j \le Logic_{MAX} \quad (11)$$

where TL represents the total number of CNN layers including the repeated layers, L denotes the total number of CNN layer types and $runtime_i$ is the execution time of the layer-i. DSP_{MAX}, $Memory_{MAX}$, and $Logic_{MAX}$ represent the total DSP, on-chip memory and FPGA logic resources, respectively, available in a given FPGA.

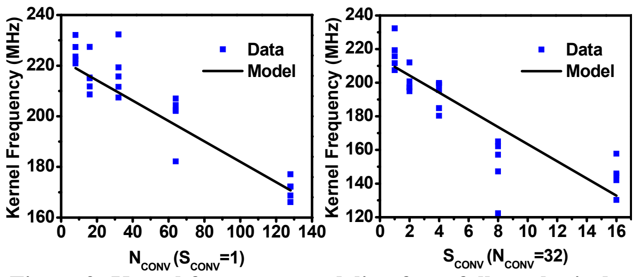


Figure 9: Kernel frequency modeling from full synthesis data at 5 random seeds. RMS error of the fit: 12.57 MHz

5.2 Performance Modeling

The execution time of each CNN layer is analytically modeled as a function of the design variables and validated by performing full synthesis at selective design points and running them on the FPGA accelerator.

5.2.1 Convolution time

The execution time of convolution layer-i is modeled as follows.

$$Convolution\ Runtime_i = \frac{No.\,of\ Convolution\ Ops_i}{N_{CONV} \times S_{CONV} \times Frequency} \quad (12)$$

$$\begin{aligned} &No.\,of\ Convolution\ Ops_i \\ &= PAD_{NCONV}(\text{Conv filter dimensions} \times \text{No.of input features}) \\ &+ PAD_{NCONV}(\text{No. of output features}) \\ &+ PAD_{NCONV}(\text{output feature dimensions}) \end{aligned} \quad (13)$$

where PAD_{NCONV} ceils its inputs to the multiple of N_{CONV}. Maximum frequency of the kernel, which is also a function of N_{CONV} and S_{CONV}, is modeled empirically from the synthesis data with different random seeds, as shown in Figure 9. The execution time model and the measured execution time of convolution layers 1-5 of AlexNet implementation for a sweep of N_{CONV} at different S_{CONV} values are compared in Figure 10.

5.2.2 Other layers

Similarly, the execution time of normalization, pooling and fully connected layers are modeled as functions of their respective loop unroll factors used for acceleration as follows.

$$Runtime_i = \frac{\#Operations_i}{Unroll\ factor \times Frequency} \quad (14)$$

The execution time model vs. measured run time of normalization and fully connected classification layers are shown in Figure 11.

5.2.3 Memory Bandwidth

Input data, weights, intermediate data and final output data are stored in the external memory that is present on the FPGA accelerator board. To enable efficient data transfer to and from external memory, Altera OpenCL compiler generates complex load/store units similar to those in GPUs, which combine multiple external memory accesses into a single burst access, known as memory coalescing. This ensures the efficient use of available external memory bandwidth with less contention for memory accesses between multiple computational blocks. On the other hand, this makes it difficult to model the external memory bandwidth usage with respect to the design variables used for acceleration. This problem is aggravated by the reuse of the scalable hardware blocks in multiple iterations of CNN layers

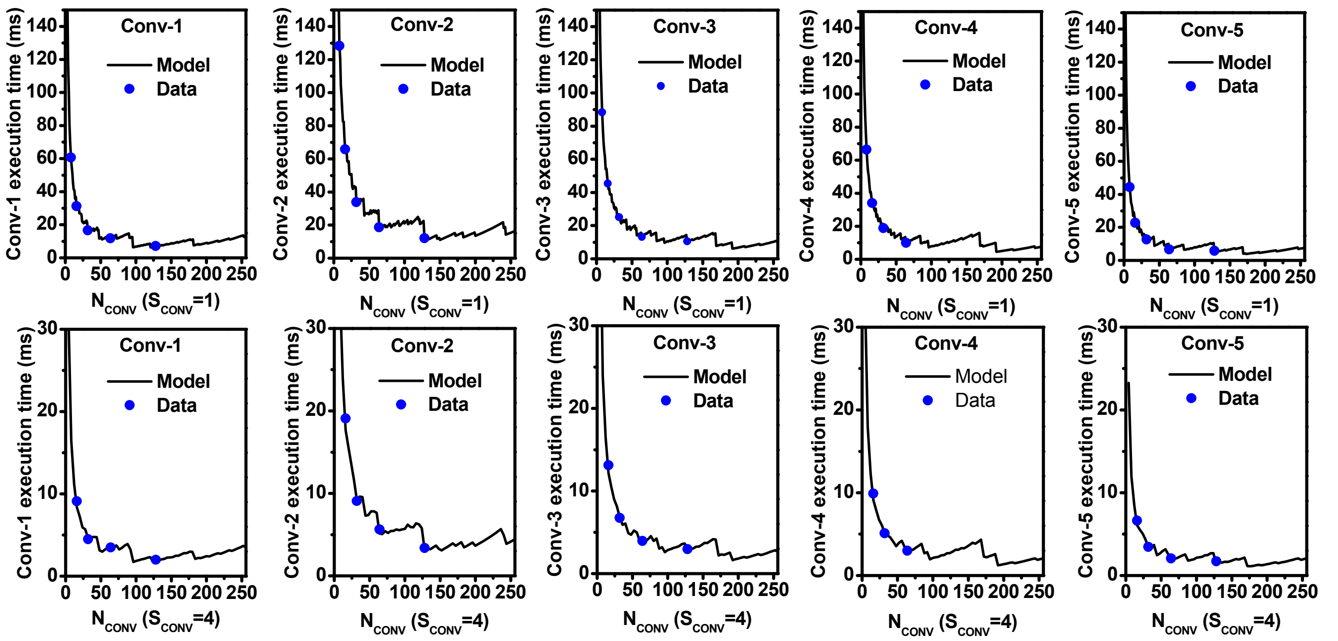


Figure 10: Run time model vs. measured time of convolution layers 1-5 for a sweep of matrix multiplication block size (N_{CONV}) for SIMD vectorization factor, S_{CONV} = 1 and 4.

with different input dimensions, which will have different access patterns. For example, the execution time of fully connected layers 6 and 7 of AlexNet model shown in Figure 11 shows that the model matches well with the measured time till N_{FC}=100. For N_{FC}>100, the measured time increases slightly, but the model still shows a reduction in execution time. This discrepancy is caused by the bandwidth limitation of the FPGA board used for the model validation. Hence we use the bandwidth limitation of the FPGA board to define the upper limits for the design variables in our optimization framework.

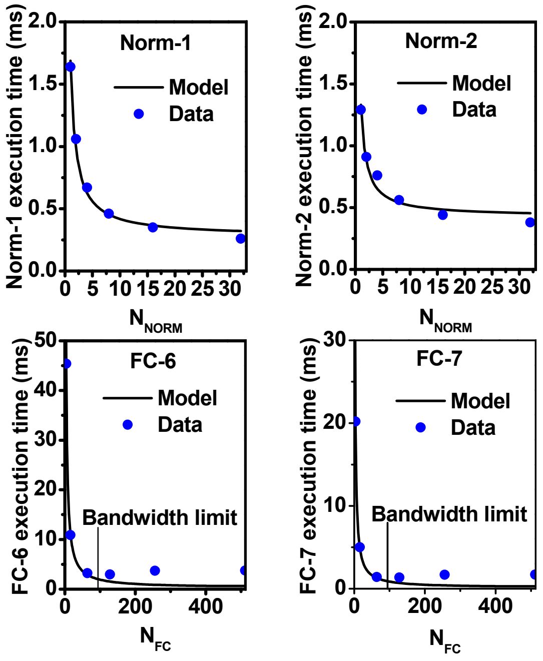


Figure 11: The execution time model vs. measured data of normalization and fully connected layers in AlexNet for sweep of loop unroll factors N_{NORM} and N_{FC}.

5.3 Resource Utilization Modeling

Analytically modeling the FPGA resource utilization of an algorithm in a high-level language such as OpenCL may not be feasible because of the optimizations performed in the HLS tools. Hence, we use synthesis results to empirically model the FPGA resource utilization. DSP block usage, on-chip memory and logic utilization from synthesis results of each CNN layer are fitted to linear regression models as a function of their design variables.

For example, resource utilization models of normalization block are shown in Figure 12. Logic element and DSP utilization from the synthesis data in Figure 12 show a linear increase with the swept design variable N_{NORM}. On the other hand, on-chip memory utilization model shows small discrepancy with the synthesis data at intermediate points because of implementation of coalescing load/store units in which the memory resource utilization depends on whether the external memory data width is an integer multiple of the design variables i.e. N_{NORM}.

5.4 Optimization Framework

From the convolution run time model in Figure 10, we see that it is non-monotonic, because of the differences in dimensions of the CNN layers. Although exhaustive search of all the design variables could be done using the performance and resource utilization models, it may not be feasible if the number of design variables and/or the FPGA resources increase substantially. This

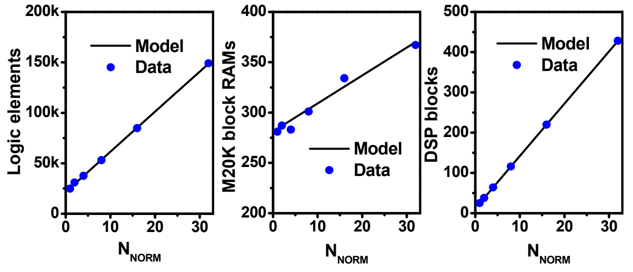


Figure 12: Resource utilization empirical models for normalization block.

calls for global optimization methodologies such as simulated annealing, genetic algorithm or particle swarm optimization with integer variables and multiple inequality constraints. In this work, we use genetic algorithm with integer constraints from the global optimization toolbox in Matlab for the design space exploration.

Genetic algorithm is a stochastic optimization technique that mimics the biological evolution process and is popularly used to find the global minimum of an objective function subject to a set of constraints. It can also handle mixed integer programming problems, where some of the design variables are integers. It iteratively improves the quality of the solution by generating a set of candidate solutions at each iteration or generation from a combination of the best solutions from the previous generation based on a set of genetic rules – selection, crossover and mutation. The solutions that violate the constraints (i.e. Equations (9)-(11)) are penalized with a higher objective function value to ensure convergence of the feasible solutions to a global minimum.

The design space of the OpenCL-based FPGA accelerator design is illustrated in Equation (15).

$$\begin{gathered} S_{CONV} = 1, 2, 4, 8 \; or \; 16 \\ N_{CONV} = N \times S_{CONV}, 0 < N < N_{MAX} \\ 0 < N_{NORM} < N_{NORM(MAX)} \\ 0 < N_{POOL} < N_{POOL(MAX)} \\ 0 < N_{FC} < N_{FC(MAX)} \end{gathered} \tag{15}$$

where all the design variables are integers, and upper limits of the design space exploration such as N_{MAX}, $N_{NORM(MAX)}$, $N_{POOL(MAX)}$, and $N_{FC(MAX)}$ are determined by the external memory bandwidth of the FPGA board. For example, in a fully connected layer implementation where k bytes are required for each MAC operation, N_{FC} of an accelerator board with external memory bandwidth of M_{BW} is computed as shown in Equation (16).

$$N_{FC(MAX)} = \frac{Memory\ bandwidth\ (M_{BW})}{k \times Frequency} \tag{16}$$

For an FPGA system with 6 GB/s external memory bandwidth, requiring 2 bytes per MAC operation in a fully connected layer with 100MHz kernel frequency, the upper limit for N_{FC} can be computed from Equation (16) as 30. Similarly, the upper limits of other blocks can be computed based on the number of external memory transfers required for each operation.

6. EXPERIMENTAL RESULTS

In this section, we present the validation results of the proposed optimization framework by implementing and accelerating two large-scale CNN models: AlexNet and VGG (16-layer) models on two FPGA boards with different hardware resources. The hardware specifications of the two Altera Stratix-V based boards are summarized in Table 2.

Both networks are implemented in OpenCL with fixed-point operations using 8-bit weights for convolution and fully connected layers as obtained from the precision study in Section 3. Although 10-bit precision is chosen for inner product weights, they are still represented using 8-bits as the 2 bits in MSB side are zeros in all the weights. Using the performance and resource utilization models (Sections 5.2 and 5.3) and the maximum hardware resources available in the two boards, optimization framework is run on both AlexNet and VGG models to find the optimal combination of design variables (N_{CONV}, S_{CONV}, N_{NORM}, N_{POOL}, N_{FC}) that maximizes the throughput. For example, Figure 13 shows the execution time of the best solution of each iteration during the optimization of AlexNet implementation on DE5-Net FPGA board. Table 3 shows the execution time from the model, measured execution time on FPGA and the FPGA resource utilization at chosen points A, B and C in Figure 13. The final design variables for both networks optimized for the two FPGA boards are shown in Table 4. VGG model does not include normalization layers, hence the corresponding kernel is removed for the FPGA implementation.

Using Altera OpenCL SDK, the OpenCL kernel codes for AlexNet and VGG models are compiled for the two boards using the corresponding optimized parameters from Table 3. Using the host code APIs, FPGA is programmed and the CNN model is run by queueing the OpenCL implemented CNN kernels with appropriate arguments that consist of input/output buffer address locations and the layer dimensions. The execution time of each kernel and the entire model are measured and throughput is computed as (total number of operations)/(execution time).

Table 2: Comparison of FPGA accelerator boards.

Specification	P395-D8 [25]	DE5-Net [26]
FPGA	Stratix-V GSD8	Stratix-V GXA7
Logic elements	695k	622k
DSP blocks	1,963	256
M20K RAMs	2,567	2,560
External memory	4× 8GB DDR3	2× 2GB DDR3

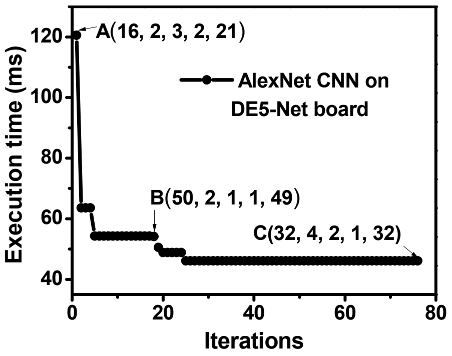


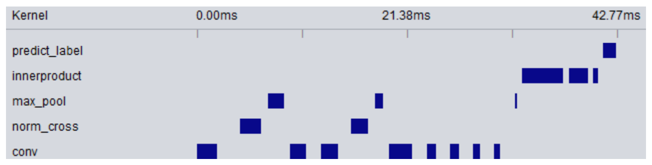
Figure 13: Optimization progress of AlexNet implementation. Design variables (N_{CONV}, S_{CONV}, N_{NORM}, N_{POOL}, N_{FC}) are shown at points A, B and C.

Table 3: Summary of Execution time and Utilization.

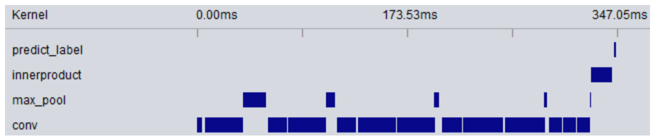
	A	B	C
Exec. time (model)	120.6 ms	54.3 ms	46.1 ms
Exec. time (measured)	117.7 ms	52.6 ms	45.7 ms
Logic elements	158k	152k	153k
M20K memory blocks	1,439	1,744	1,673
DSP blocks	164	234	246

Table 4: Optimized parameters.

	P395-D8 board		DE5-Net board	
	AlexNet	VGG	AlexNet	VGG
N_{CONV}	64	64	32	64
S_{CONV}	8	8	4	2
N_{NORM}	2	-	2	-
N_{POOL}	1	1	1	1
N_{FC}	71	64	32	30



(a)



(b)

Figure 14: The execution time of CNN layers in (a) AlexNet and (b) VGG models on P395-D8 FPGA accelerator.

The execution time of the CNN layers in AlexNet and VGG models implemented on P395-D8 board with kernel profiling support) is shown in Figure 14. The final classification time without kernel profiling will be significantly lower than that shown in Figure 14 because of the delay involved with kernel profiling itself. The execution of fully-connected layers can be overlapped with the initial convolution layers of the next image, which increases the overall throughput of the accelerator (by 27% in AlexNet implementation on P395-D8). The next input image transfer from the OpenCL host to the off-chip memory on the FPGA board is overlapped with current CNN operations, thus not hampering the throughput. The initial model weight transfer from the host to the board, which only occurs once in the beginning, is not included for throughput computation.

The total classification time per image and overall throughput of AlexNet and VGG models on P395-D8 and DE5-Net boards are compared with Caffe tool [20] running on Intel core i5-4590 CPU (3.3 GHz) as shown in Table 5. Although both FPGAs have similar number of logic elements and on-chip memory blocks, the smaller number of DSP blocks in DE5-Net accounts for its lower throughput compared to that of P395-D8. The software implementation in Caffe tool uses libraries optimized for basic vector and matrix operations (i.e., ATLAS [27]) for performing CNN operations. Our OpenCL based FPGA implementations on P395-D8 achieve 9.5x and 5.5x speedups for AlexNet and VGG models, respectively, compared to the CPU implementation in Caffe tool.

The execution time, throughput and the resource utilization of each kernel type of the AlexNet implementation on P395-D8 and DE5-Net FPGA accelerator boards are shown in Figure 15. VGG implementation on P395-D8 achieves a peak throughput of 136.5 GOPS for convolution layers, and 117.8 GOPS including all layers and operations while performing image classification. From the implementation results, we see that throughput of the accelerator is largely proportional to the number of DSP blocks

Table 5: Classification time/image and overall throughput.

	FPGA	Classification time/image (ms)	Throughput (GOPS)
AlexNet	P395-D8	20.1	72.4
	DE5-Net	45.7	31.8
	CPU	191.9	7.6
VGG	P395-D8	262.9	117.8
	DE5-Net	651.2	47.5
	CPU	1437.2	21.5

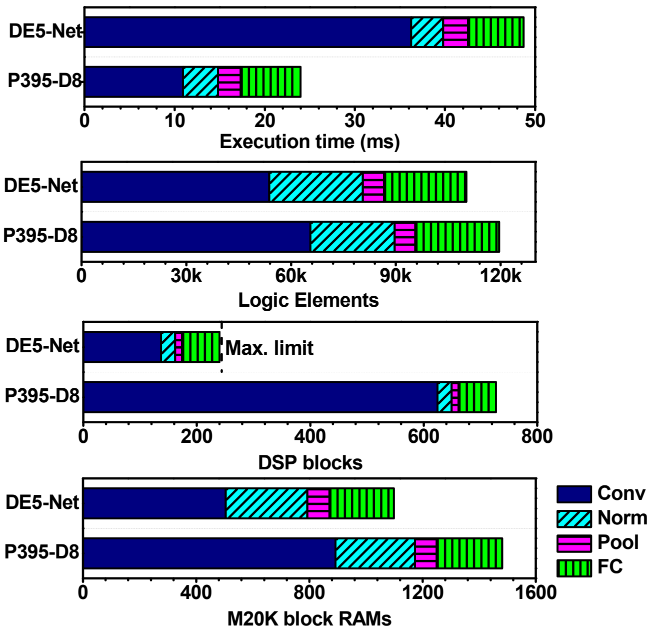


Figure 15: Execution time and resource utilization of each CNN layer type for AlexNet implementation on P395-D8 and DE5-Net FPGA boards.

used in the implementation. AlexNet implementation on P395-D8 board is limited by the number of available M20K block RAMs, while only 727 out of 1963 available DSP blocks are utilized. On the other hand, throughput on DE5-Net FPGA board is limited by the lower number of available DSP blocks, although the on-chip memory resources and logic elements are not fully utilized.

Our optimization framework reports the hardware resource that causes the performance bottleneck, such that the user can choose another FPGA hardware, which has larger number of the specific hardware resources (e.g. DSP blocks). This methodology can also be used to find the ideal specifications of an FPGA suited for CNN, by performing optimization with relaxed constraints for the bottleneck hardware resource. For example, increasing the on-chip memory resources on P395-D8 FPGA by 10% directly increases the throughput of AlexNet implementation by ~10%. This work assumes that MAC operations are implemented using the DSP blocks only. However, we can potentially enhance the throughput further by using the remaining logic elements to implement MAC operations, which will be studied in future work.

The top-1 and top-5 accuracies of FPGA implementation of AlexNet and VGG models compared to those of the full-precision Caffe models are summarized in Table 6. The accuracy degradation due to fixed-point operations in FPGA implementation is <2% for top-1 accuracy and <1% for top-5 accuracy for both AlexNet and VGG models.

Both DE5-Net and P395-D8 boards are connected to a PCIe slot of a desktop computer whose CPU operates as the OpenCL host. Since the FPGA board receives power from external power port as well as PCIe slot, the power measurement of the FPGA

Table 6: Model accuracy comparison.

Accuracy	Full precision in Caffe tool		Fixed-point FPGA implementation	
	Top-1	Top-5	Top-1	Top-5
AlexNet	56.82%	79.95%	55.41%	78.98%
VGG	68.35%	88.44%	66.58%	87.48%

Table 7: Comparison with previous implementations.

	[12]	[13]	This work
FPGA	Virtex-6 VLX240T	Virtex-7 VX485T	Stratix-V GSD8
Frequency	150 MHz	100 MHz	120 MHz
CNN size	2.74 GMAC	1.33 GOP	30.9 GOP
Precision	fixed	float (32b)	fixed (8-16b)
Throughput	17 GOPS[b]	61.6 GOPS[a]	136.5 GOPS[a] 117.8 GOPS[b]

[a]convolution operation only
[b]all operations for image classification

board itself is not straightforward. We attempted to block the power connection through PCIe and have the FPGA board powered only through the external power port. This way, the average power consumption of DE5-Net board was measured as 24.2W after programming AlexNet configuration, and as 25.8W while performing classification. On the other hand, the same measurement method was not feasible on P395-D8 board as it was designed to use both power supplies. Nonetheless, we measured its power consumption as 19.1W after programming with AlexNet configuration file, using a utility function provided by board manufacturer that measures the steady state power of the board.[1] We compare the performance of VGG model implementation on P395-D8 FPGA board to the existing FPGA based CNN accelerators in Table 7. For the entire VGG model with 30.9 GOP, our FPGA accelerator achieves overall throughput of 117.8 GOPS for ImageNet classification.

7. CONCLUSION

In this work, we implemented scalable CNN layers on FPGA using OpenCL framework and identified the key design variables for hardware acceleration. Further, we proposed a design space exploration methodology based on a combination of analytical and empirical models for performance and resource utilization, to find the optimal design variables that yield maximum acceleration of any CNN model implementation using limited FPGA resources. Using the proposed methodology, we implemented two large-scale CNNs, AlexNet and VGG, on P395-D8 and DE5-Net FPGA boards and achieved superior performance compared to previous work.

8. ACKNOWLEDGEMENTS

This project was partially supported by Samsung Advanced Institute of Technology.

9. REFERENCES

[1] Y. LeCun, et al. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, 396-404, 1990.

[2] O. Russakovsky, et al. ImageNet large-scale visual recognition challenge. In *Int. J. Computer Vision*, 2015.

[3] A. Karpathy, et al. Large-scale video classification with convolutional neural networks. In *CVPR*, 1725-1732, 2014.

[1]The power consumption difference between the desktop computer without FPGA and with FPGA running AlexNet is measured as 26W for DE5-Net and 35W for P395-D8 boards. Note that this difference includes the power consumption of CPU running the OpenCL host code, which could be much smaller with embedded processors in FPGA chips.

[4] H. Li, Z. Lin, X. Shen, J. Brandt and G. Hua. A convolutional neural network cascade for face detection. In *CVPR*, 5325-5334, 2015.

[5] P. Barros, S. Magg, C. Weber and S. Wermter. A multichannel convolutional neural network for hand posture recognition. In *Int. Conf. on Artificial Neural Networks (ICANN),* 403-410, 2014.

[6] O. Abdel-Hamid, et al. Convolutional neural networks for speech recognition. In *IEEE Trans. on Audio, Speech and Language Processing,* 1533-1545, Oct 2014.

[7] R. Collobert and J. Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Int. Conf. on Machine Learning*, 160-167, 2008.

[8] S. Lai, L. Xu, K. Liu and J. Zhao. Recurrent convolutional neural networks for text classification. In *AAAI Conf. on Artificial Intelligence*, 2267-2273, 2015.

[9] C. Szegedy, et al. Going deeper with convolutions. In *CVPR,* 1-9, 2015.

[10] C. Farabet, et al. Hardware accelerated convolutional neural networks for synthetic vision systems. In *ISCAS,* 257-260, 2010.

[11] S. Chakradhar, et al. A dynamically configurable coprocessor for convolutional neural networks. In *ISCA*, 247-257, 2010.

[12] M. Peemen, et al. Memory-centric accelerator design for convolutional neural networks. In *ICCD*, 13-19, 2013.

[13] C. Zhang, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *ACM Int. Symp. On Field-Programmable Gate Arrays,* 161-170, 2015.

[14] V. Gokhale, et al. A 240 G-ops/s mobile coprocessor for deep neural networks. In *CVPR Workshops,* 696-701, 2014.

[15] Y. Chen, et al. DaDianNao: A machine-learning supercomputer. In *IEEE/ACM Int. Symp. on Microarchitecture*, 602-622, 2014.

[16] A. Krizhevsky, et al. ImageNet classification with deep convolutional neural networks. In *NIPS,* 1097-1105, 2012.

[17] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556.

[18] Y.L. Boureau, et al. A Theoretical Analysis of Feature Pooling in Visual Recognition. In *Int. Conf. on Machine Learning*, 2010.

[19] M. Denil, et al. Predicting parameters in deep learning. In *NIPS*, 2148–2156, 2013.

[20] Y. Jia, et al. Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093.

[21] Khronos OpenCL Working Group. The OpenCL Specification, version 1.1.44, 2011.

[22] M. S. Abdelfattah, et al. Gzip on a chip: high performance lossless data compression on FPGAs using OpenCL. In *Int. Workshop on OpenCL 2014.*

[23] K. Chellapilla, S. Puri and P. Simard. High performance convolutional neural networks for document processing. In *Int. Workshop on Frontiers in Handwriting Recognition,* 2006.

[24] Altera OpenCL design examples. Available online at https://www.altera.com/support/support-resources/design-examples/design-software/opencl.html

[25] Nallatech P395-D8 OpenCL FPGA accelerator cards. http://www.nallatech.com/wp-content/uploads/openclcardspb_v1_51.pdf

[26] DE5-Net FPGA kit user manual. Available online at ftp://ftp.altera.com/up/pub/Altera_Material/Boards/DE5/DE5_User_Manual.pdf

[27] R.C. Whaley and J.J. Dongarra. Automatically tuned linear algebra software. In *Proc. SuperComputing 1998: High Performance Networking and Computing*, 2001.