Layer-wise Performance Bottleneck Analysis of Deep Neural Networks

Hengyu Zhao, Colin Weinshenker*, Mohamed Ibrahim*, Adwait Jog*, Jishen Zhao
*University of California, Santa Cruz, *The College of William and Mary*
hzhao28@ucsc.edu

***Abstract*—Deep neural networks (DNNs) are becoming the inevitable part of a wide range of applications domains, such as visual and speech recognition. Recently, Graphics Processing Units (GPUs) are showing great success in helping meet the performance and energy efficiency demands of DNNs. In this paper, we identify GPU performance bottlenecks via characterizing the data access behaviors of AlexNet and VGG16 models in a layer-wise manner. The goal of this work is to find the performance bottlenecks of DNNs. We obtain following findings: (i) Backward propagation is more performance critical than forward propagation. (ii) The working set of convolutional inter-layers does not fit in L1 cache, while convolutional input layer can exploit L1 cache sufficiently. (iii) Interconnect network can also be a performance bottleneck that substantially increase GPU memory bandwidth demand.**

I. Introduction

Deep neural networks (DNNs) are finding their way into increasingly wider range of application domains, such as visual recognition [1], speech recognition [2], and natural language processing [3]. Deep learning has two phases: training and inference – inference leverages the trained neural network to infer things about new data it is presented with. As a result, training DNNs typically has much higher demand for compute power than inference and is becoming increasingly data intensive with larger, deeper training models. In fact, graphic processing units (GPUs) are extensively used in the training of modern DNN models, due to their tremendous compute horsepower and strong backend support for DNN software libraries such as cuDNN [4].

With DNN, larger and deeper neural networks with more parameters typically result in better accuracy. Yet, this also substantially increases the hardware demand on both compute power and data access efficiency. In order to maintain a reasonable training performance with the continuous scaling of DNN models, it is critical to ensure that future GPU systems can keep up with the increase of the hardware demand.

Our goal in this paper is to investigate the performance bottlenecks during training of DNNs in commodity GPU hardware. To this end, we make efforts toward characterizing the performance of DNN models on GPU systems. First, we analyze the performance of two popular ImageNet models – AlexNet [1] and VGG-16 [5] – built with Caffe [6] deep learning framework on one of the latest graphics card product. Second, we examine layer-wise execution time, stall reasons, cache access behavior, and memory bandwidth utilization of the two ImageNet models. This paper makes the following contributions:

- We build a layer-wise model for training VGG-16 and AlexNet on GPUs.
- We identify GPU performance bottlenecks in compute and cache resources, by characterizing the performance and data access behaviors of AlexNet and VGG-16 models in a layer-wise manner.

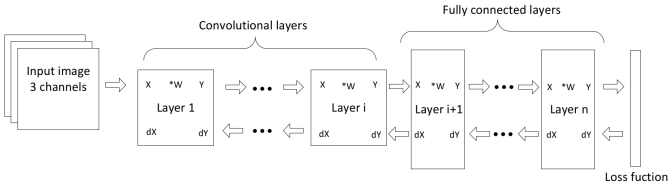

Fig. 1: DNN architecture.

II. Deep Neural Networks (DNNs)

DNNs are combined with several different types of layers, such as convolutional layers, activation layers, pooling layers, and fully connected layers. Typically, these layers can be divided into two categories: (i) feature extraction layers that extract input features and (ii) classification layers that analyze features and classify input images into groups. DNNs have two stages: training and inference. Training allows DNNs to learn and update weights of each layer. Training of DNNs typically performs both forward propagation and backward propagation [7] to generate weights. As a result, training phase is typically much more compute and data intensive than inference phase. As such, this paper focuses on studying training phase. As the name implies, the traverse direction is in reverse between forward and backward propagation. When training a DNN model, we can divide input images into several sets that are processed independently. Each set of images is called a batch.

Figure 1 shows a typical DNN architecture. In this paper, we evaluate AlexNet and VGG-16, which are two popular deep learning models used in image classification. The two models share the similar layout of neural network layers as shown in Figure 1. AlexNet has five convolutional layers, while VGG16 adopts a deeper neural network with 13 convolutional layers.

DNN Training. Training is necessary for DNNs before we use them to do certain tasks. During the training phase, lots of weights of every layer need to be updated real-timely by performing forward and backward propagation.

Forward propagation. In Figure 1, when performing forward propagation, several feature vectors are passed to the input layer. Each layer multiplies its inputs(x) by the weight matrix connecting and outputs y to the next layer. A non-linear transformation (e.g., a sigmoid or rectified linear unit function) is applied to the result. This process is repeated, propagating the input signal through the network. The transformed signal that reaches the output layer is interpreted as an encoded output – an image classification, for example.

Backward propagation. In Figure 1, in the backward propagation phase, the correct output for the given input is used with a loss function to compute the error of the network's prediction. Seeking derivative from the latter layer with input gradient map(dY) to the former layer with output gradient map(dX). A simple squared error function can suffice.

$$E = \frac{1}{2}\sum_{i=0}^{n}(t_i - y_i)^2$$

Where E is the sum of the network's prediction error over all n training examples in the training set, t_i is the true label for input sample i, and y_i is the network's predicted classification for input i.

After determining the prediction error on an observation, the weights of the network are updated. The functions by which the inputs determine the error of the network and their gradients with respect to the network's last predicted output are known. Thus the chain rule can be applied to each function in the network to create a map of how network error changes with respect to any individual weight [8], [9]. Network weights are then updated to improve network performance for the last seen observation.

To train a neural network practically, we can feed it a labeled "training" dataset (i.e., a dataset consisting of input features and known correct outputs). For each input, the error of the network's output is computed. After summing the error over a certain number of observations (referred to as the mini-batch size), the weights are updated. Over many updates, the weights in a network form a relationship between the probability distribution of the model and the probability distribution of the underlying data.

III. Experimental Setup

A. Workloads

AlexNet. AlexNet [1] is a type of neural network to do image classification tasks using ImageNet dataset. It has five convolutional layers, three pooling layers and three fully connected layers.

VGG-16. VGG-16's [5] main task is also to do image classification and localization, but it has more layers than AlexNet. It has thirteen convolutional layers and three fully connected layers.

Datasets. ImageNet is a huge image dataset which contains millions of images belong to thousands of categories. Begin at 2010, ImageNet Large Scale Visual Recognition Challenge (ILSVRC) has been held annually. ILSVRC exploits a subset of ImageNet with 1.3 million training images, 50000 validation images, 150000 testing images in 1000 categories. We evaluate ImageNet with four different batch size: 32, 64, 128, and 256. We measure the performance for 100 iterations when training.

Framework. Caffe [6] is a popular deep learning framework, which is produced by Berkeley AI Research.

TABLE I: System configuration.

CPU	Intel Xeon E5-2620 V3@2.4GHz
Main memory	16GB DDR4
Operating system	Ubuntu 16.04.2
GPU	NVIDIA GeForce GTX 1080 Ti (Pascal)
GPU cores	28 SMs, 128 CUDA cores per SM, 1.5GHz
L1 cache	24KB per SM
L2 cache	4096KB
Memory interface	8 memory controllers, 352-bit bus width
GPU main memory	11GB GDDR5X

B. Real Machine Configuration

We run AlexNet and VGG-16 on GTX 1080Ti graphic card with Dell Precision T7810 Tower Workstation. GTX 1080Ti combines L1 cache and texture cache together as unified cache, so we show unified cache performance (note that we denote it as L1 cache in our results sections). We conducted experiments on Nvidia's GTX 1080Ti graphic card, which has 3584 cuda cores, 11GB memory capacity, 484 GB/s memory bandwidth, and Pascal architecture. Table I lists our system configuration.

IV. Real Machine Characterization

In this section, we characterize AlexNet and VGG-16 on real GPU hardware. In order to explore GPU system performance bottlenecks and the criticality of various architecture components, we analyze the execution time, data operations, and cache and memory access behaviors of each neural network layers across various image input batch sizes. We also show several key observations of our analysis on the major computational kernels executed in each layer.

A. Execution Time and Instruction Distribution

Figure 2 through Figure 4 explore a layer-wise performance landscape of our workloads.

Layer-wise execution time. Figure 2 shows our evaluation of execution time breakdown in various ways. While the overall trend of execution time (Figure 2(a) and (d), (e)) is inline with most previous studies on the same workloads [10], [11], we make four observations from our experiments. First, convolutional (CONV) layers execute for much longer time than fully connected (FCN) layers. Second, CONV inter-layers dominate the execution time of all CONV layers; these inter-layers also execute more instructions than other layers (Figure 3(a), Figure 4(a)). Third, execution time and instruction count increases as we increase the batch size from 32 to 256. Finally, with both CONV and FCN layers, the execution time of backpropagation can be over 2× of forward

propagation. What is more, the computation latency ratio of VGG-16 is larger than AlexNet. (Figure 2(d), (e)) [10], [11].
Findings: Our results show that (1) CONV inter-layers dominate both execution time and instruction count; (2) backpropagation is more performance critical than forward propagation.

Stalls due to data access. CONV layers are typically considered as much more compute-intensive than FCN layers. However, we identify three observations that speaks for the volume of the performance impact of data access. First, Figure 2(c) shows that data access imposes substantial stalls during the execution time. Such stalls stay above 30% of total stalls across various batch sizes, even among the most compute-intensive CONV layers with the lowest load/store instruction ratios shown in Figure 3(b) and Figure 4(b). In fact, based on Figure 2(b) and (c), data stall time of CONV inter-layers can be up to 38% longer than FCN layers. Second, the number of stalls increase as we increase batch size. This is consistent with our observation on GPU main memory access, where the number of main memory requests scales up almost linearly with image batch size. Finally, because CONV inter-layers execute much more instructions than FCN layers (Figure 3(a), Figure 4(a)), these inter-layers can generate more data requests despite less data intensive than FCN layers.
Findings: Based on our investigation on stall time and instruction breakdown, data access – which is performed along the path of caches, interconnect network, and memory interface – is performance critical to both CONV inter-layers and FCN layers.

B. Cache Access Behavior

To study the performance impact of data access in caches, we characterize read and write access behaviors in GPU cache hierarchy.

L1 cache access behavior. To study L1 cache access behavior, we evaluate the access throughput (GB/s) and layer-wise hit rates as shown in Figure 5 and Figure 6. We make four observations. First, CONV inter-layers have much lower L1 cache access throughput than other layers, despite issuing much more L1 accesses (with the load and store instructions based on Figure 3 and Figure 4). Second, CONV inter-layers have much lower L1 hit rate than other layers. This observation is consistent with the long data access stall time of these layers. The reason can be either or both of the two: a) their working set does not fit in L1 caches; b) they have low data access locality (our later evaluation on L2 access behavior demonstrates that this is not the case). Third, CONV input layer (CONV1) has a high L1 hit rate, but L1 hit rate drops in as CONV layers get deeper. Finally, L1 throughput and hit rate appear stable across various batch sizes with CONV layers. While FCN layers also have stable L1 hit rates as the batch size alters, L1 throughput significantly increases when the batch size is increased from 32 to 64. We notice that executed cuDNN kernels are changed when we change the batch size between the two.
Findings: As such, we conclude that 1) the working set of CONV inter-layers do not fit in L1 cache, while CONV input layer can utilize L1 cache effectively; 2) L1 cache access behavior remains stable across various batch sizes.

L2 cache access behavior. Due to CONV inter-layer's low hit rate in L1 caches, L2 cache can be performance critical to these layers. We make several observations based on our L2 cache access evaluation shown in Figure 7 and Figure 8. First, most CONV inter-layers generate similar read and write throughput with L2 cache, while yield high read hit rates (on average 68%) and lower write hit rates (on average 45%). Second, we notice that *tensor_add* and *convolution* are the two most time consuming operations executed in these layers. In particular, tensor_add yields almost 0% read hit rate but high write hit rate in L2 cache, whereas the read/write hit rates of convolution appears completely in reverse. Tensor_add adds the scaled values of a bias tensor to another tensor and writes the result to a third tensor. With nearly 0% read hit rate, the two tensors are always not in the cache. The convolution operation takes an input image and a convolution kernel and write the convoluted result to another memory location. As such, this operation always misses when it writes to the new memory locations. Third, CONV input and FCN layers yield much higher read than write throughput. With each layer, the overall L2 throughput is similar to L1 cache and remains similar with various batch sizes. Because the change of the kernels in FCN layers mainly impacts read operations, L2 write throughput remains stable when we increase batch size from 32 to 64. Finally, FCN layers have much higher write hit rates than reads, whereas CONV layers are in reverse.
Findings: It appears that CONV inter-layers yield much higher hit rates in the 4MB L2 cache than the 24KB L1 caches. As such, these layers have sufficient locality, especially with read requests, if the GPU can integrate large caches to accommodate their working set. Furthermore, given FCN layer's low write throughput yet high write hit rates, L2 cache can efficiently accommodate write requests of FCN layers.

C. Memory Bandwidth Demand

As shown in Figure 9 and Figure 10, GPU off-chip memory bandwidth demand is consistent with L2 cache misses. In particular, CONV inter-layers generates more write traffic on the memory bus than reads. Due to the high L2 write hit rates of FCN layers, a substantial portion of off-chip memory traffic is reads.

V. Related Work

To the best of our knowledge, this is the first work that characterizes performance and cache behaviors of DNNs on commodity GPU systems in a layer-wise manner. In this section, we describe previous studies that are closely related to ours.

Virtualized DNN (vDNN) [10] proposes a runtime memory manager that virtualizes the memory usage of DNNs such that both GPU and CPU memory can simultaneously be utilized for training larger DNNs. The GPU performance characteristics of five popular deep learning frameworks: Caffe, CNTK, TensorFlow, Theano, and Torch in AlexNet has been analyzed

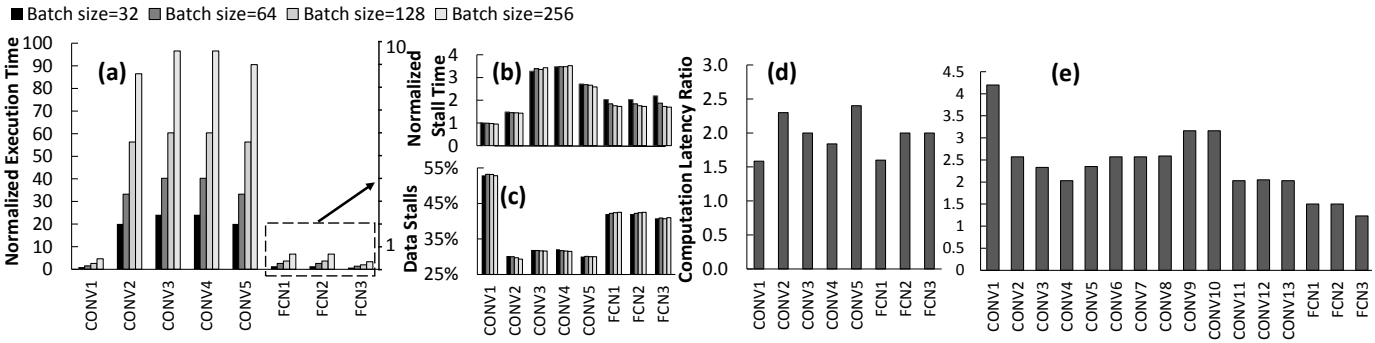

Fig. 2: AlexNet (a) Execution time and (b) total stall time normalized to *CONV1* layer with a batch size of 32. (c) Percentage of stalls due to data access in AlexNet. (d) Backpropagation to forward propagation computation latency ratio with a 256 batch size in (d) AlexNet and (e) VGG-16.

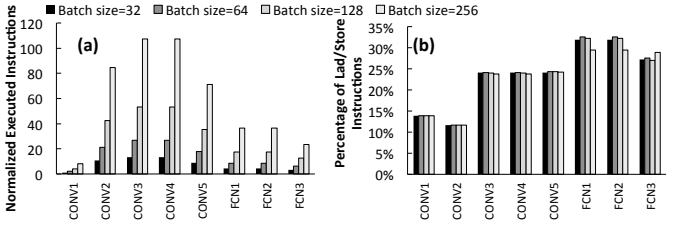

Fig. 3: (a) The number of total executed instructions and (b) percentage of load and store instructions in AlexNet (normalized to *CONV1* layer with batch a size of 32).

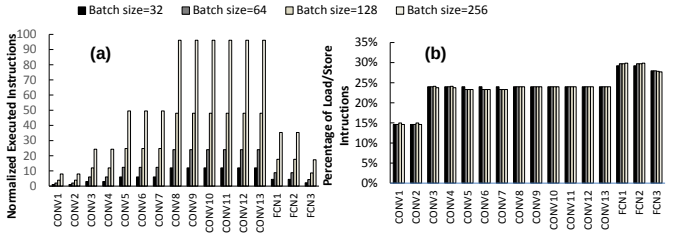

Fig. 4: (a) The number of total executed instructions and (b) percentage of load and store instructions in VGG-16 (normalized to *CONV1* layer with batch a size of 32).

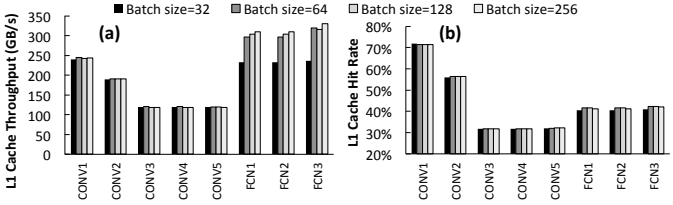

Fig. 5: L1 cache (a) throughput and (b) hit rate of each layer in AlexNet.

in [11], and this paper also evaluates the GPU performance characteristics of four different convolution algorithms and suggest criteria to choose proper convolutional algorithms to build efficient deep learning model. Efficient inference engine (EIE) [12] performs inference on the compressed network model and accelerates the resulting sparse matrix-vector multiplication with weight sharing. Jia *et.al* demonstrate that GPU caches can be detrimental to the performance of many GPU applications and characterize the impact of L1 caches on the

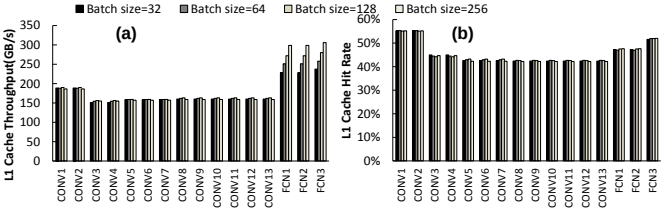

Fig. 6: L1 cache (a) throughput and (b) hit rate of each layer in VGG-16.

performance of the Rodinia suite benchmarks. However, they also show that fully-connected neural network backpropagation benefits significantly from added cache [13]. Tian et. al propose techniques for having memory that is unlikely to benefit from temporal locality bypass GPU cache [14]. Singh *et.al* and Wang et. al pursue better cache performance through cache coherence policies [15], [16].

VI. Conclusion

We perform a layer-wise characterization of a set of DNN workloads that execute on commodity GPU systems. By investigating layer-wise cache and memory access behavior, we draw the following observations:

- The execution time of convolutional inter-layers dominates the total execution time. In particular, backpropagation of these inter-layers consumes significantly longer execution time than forward propagation.
- The working set of convolutional inter-layers does not fit in L1 cache, while convolutional input layer can exploit L1 cache sufficiently.
- Interconnect network can also be a performance bottleneck that substantially increase GPU memory bandwidth demand.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] O. Abdel-Hamid, A.-R. Mohamed, and H. Jiang et al., "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 22, pp. 1533–1545, Oct. 2014.

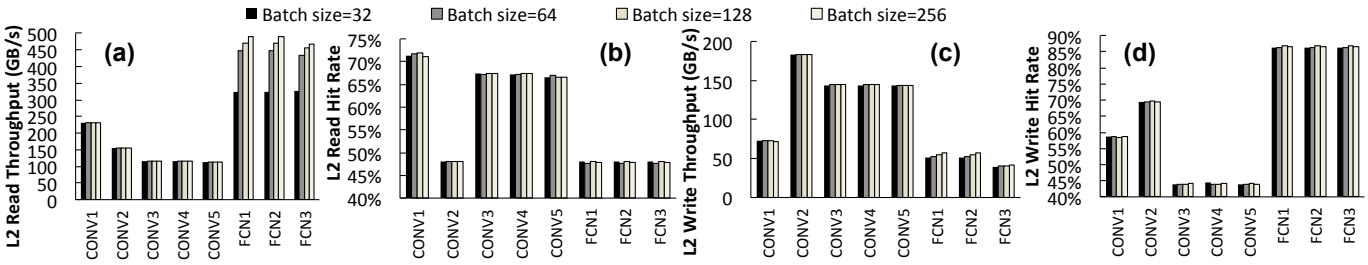

Fig. 7: L2 cache (a) read throughput, (b) read hit rate, (c) write throughput, and (d) write hit rate of each layer in AlexNet.

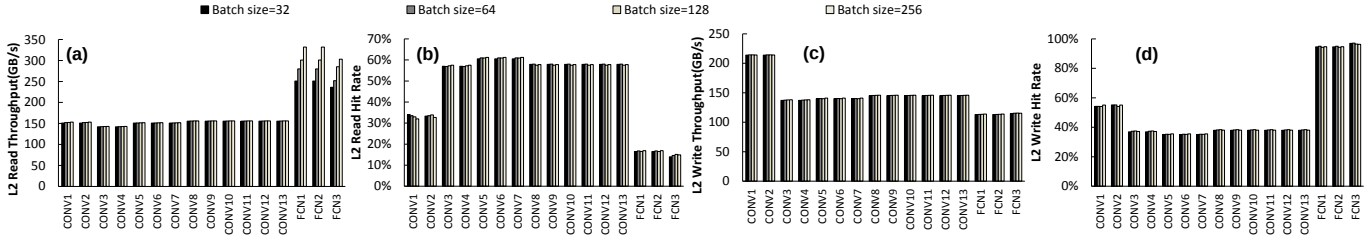

Fig. 8: L2 cache (a) read throughput, (b) read hit rate, (c) write throughput, and (d) write hit rate of each layer in VGG-16.

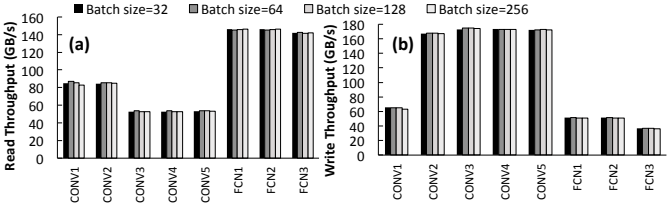

Fig. 9: GPU main memory (a) read and (b) write throughput of each layer in AlexNet.

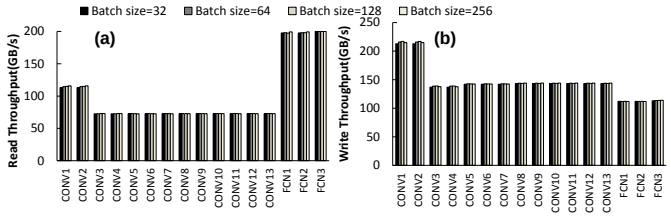

Fig. 10: GPU main memory (a) read and (b) write throughput of each layer in VGG-16.

[3] R. Collobert, J. Weston, and L. Bottou et al., "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, Nov. 2011.

[4] "NVIDIA, cuDNN: GPU accelerated deep learning," 2016.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] Y. Jia, E. Shelhamer, and J. D. et al., "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[7] Y. LeCun, L. Bottou, and Y. B. et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[8] A. Karpathy, "Hacker's Guide to Neural Networks." http://karpathy.github.io/neuralnets/.

[9] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "Virtualizing deep neural networks for memory-efficient neural network design," *arXiv preprint arXiv:1602.08124*, 2016.

[10] M. Rhu, N. Gimelshein, and J. C. et al., "vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *MICRO*, 2016.

[11] H. Kim, H. Nam, W. Jung, and J. Lee, "Performance analysis of CNN frameworks for GPUs," in *ISPASS*, 2017.

[12] S. Han, X. Liu, and H. M. et al., "EIE: efficient inference engine on compressed deep neural network," in *ISCA*, pp. 243–254, 2016.

[13] W. Jia, K. A. Shaw, and M. Martonosi, "Characterizing and Improving the Use of Demand-fetched Caches in GPUs," in *Proceedings of the 26th ACM International Conference on Supercomputing*, pp. 15–24, 2012.

[14] Y. Tian, S. Puthoor, and J. G. et al., "Adaptive GPU cache bypassing," in *Proceedings of the 8th Workshop on General Purpose Processing Using GPUs*, GPGPU-8, pp. 25–35, 2015.

[15] I. Singh, A. Shriraman, and W. W. L. F. et al., "Cache coherence for GPU architectures," in *HPCA*, pp. 578–590, 2013.

[16] H. Wang, V. Sathish, and R. S. et al., "Workload and Power Budget Partitioning for Single-Chip Heterogenous Processors," in *PACT*, 2012.