BlinkNet: Software-Defined Deep Learning Analytics with Bounded Resources

Brian Koga
Washington State University
Vancouver, WA, 98686, USA
brian.koga@wsu.edu

Theresa Vanderweide
Washington State University
Vancouver, WA, 98686, USA
theresa.vanderweide@wsu.edu

Xinghui Zhao
Washington State University
Vancouver, WA, 98686, USA
x.zhao@wsu.edu

Xuechen Zhang
Washington State University
Vancouver, WA, 98686, USA
xuechen.zhang@wsu.edu

***Abstract*—Deep neural networks (DNNs) have recently gained unprecedented success in various domains. In resource-constrained systems, QoS-aware DNNs are designed to meet the latency requirements of mission-critical deep learning applications. However, none of the existing DNNs have been designed to satisfy both latency and memory bounds simultaneously as specified by end-users in the resource-constrained systems. In this paper, we propose BLINKNET, a runtime system that can guarantee both latency and memory/storage bounds via efficient QoS-aware per-layer approximation. We implement BLINKNET in Apache TVM and evaluate it using Cifar10-quick and VGG network models. Our experimental results show that BLINKNET can meet the latency and memory requirements with 2% accuracy loss on average.**

***Index Terms*—Machine Learning, Approximation, Quality of Services**

I. INTRODUCTION

Deep neural networks (DNNs) have been attracting attention in computer vision [1], natural language processing [2], robotics [3], and many other fields. With the need for privacy, security, short latency, and limitations of communication bandwidth, the trained DNN models are gradually increasingly deployed to edge end such as mobile devices and wearable devices [4]–[8].

However, the deployment of DNN models on resource-constrained computing systems (e.g., embedded or IoT platforms) is pretty challenging. On the one hand, due to the dense computation intensity of neural networks, the application of neural networks need a guarantee of latency bounds on the systems. On the other hand, the limited memory and storage resources are the obstacles for the embedded or IoT platform to support the inference tasks which require efficiently moving a large amount of data in its memory hierarchy. Therefore, it is in much more urgent need for QoS-aware DNN models that can satisfy both latency and memory bounds on the resource-constrained computing platforms.

Most of the existing DNN frameworks do not enforce latency bounds or memory bound. For example, model compression techniques reduce the weight storage and computation cost using weight quantization [9]–[14] and pruning [15]–[20]. They do not consider the trade-off between accuracy loss, memory and storage cost, and latency bounds. Most recently, ApNet was designed to meet the latency bounds of applications through layer-specific approximation [21]. Nevertheless, it does not consider the memory cost of DNN models.

In this paper, we propose a new QoS-aware deep learning framework, named BLINKNET. It has three novel features. First, BLINKNET provides a performance interface for users to specify performance requirements as latency and memory bounds. It allows users to set memory bounds for individual deep learning applications or collectively for a group of related applications in a deep learning workflow. Second, it automatically creates an approximation model corresponding to the basic model provided by end-users. BLINKNET runs the approximation model during the inference phase to enforce latency bounds and memory bounds. The model is dynamically generated by TVM [22] and stored in a catalog database. These approximation models in the catalog can be reused for other applications having similar QoS requirements. Finally, the approximation model is periodically refined according to the user's input and the characteristics of workloads.

We have implemented a prototype of BLINKNET and integrated it with Apache TVM. Our evaluation with representative DNNs (e.g., VGG16 [23] and Cifar10-quick [24]) shows that BLINKNET can provide a performance guarantee for both deep learning applications and dynamically allocate resources according to their memory demands.

II. RELATED WORK

Research on the model approximation of DNNs. Weight pruning is one of the key model compression techniques. It is widely used to reduce the memory and storage cost of DNNs and accelerate the DNNs with ignorable accuracy loss. The key idea of weight pruning is to remove the relatively unimportant weights in the weight matrix in order to reduce the redundancy. According to the granularity, network pruning can be classified into two categories: unstructured pruning [25]–[27] and structured pruning [28]–[30]. The unstructured pruning methods prune the individual parameters resulting in an irregular compressed model. Though it can yield a higher pruning ratio, it leads to irregular memory access and an imbalanced load on each parallel processing engine. Compared to unstructured pruning, structured pruning approaches remove the filters or the channels. Intuitively, this kind of pruning is more aggressive and is more likely to degrade accuracy. However, in these works [31], the accuracies are comparable

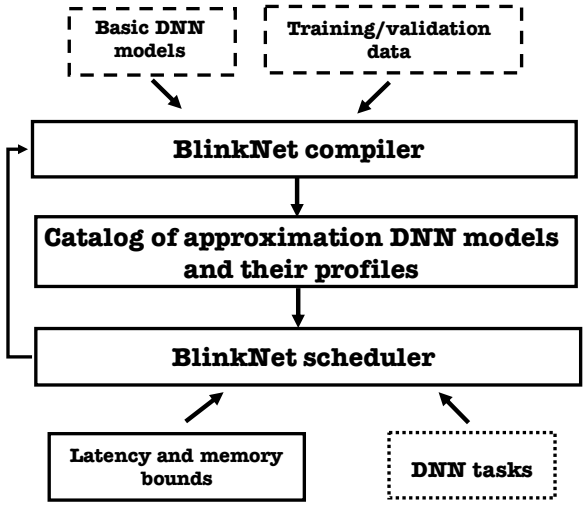

Fig. 1. Architecture overview of BLINKNET. The components in the solid rectangular are the new functionalities added for BLINKNET and the components in the rectangular of dashed lines are the existing functionalities in TVM.

to unstructured pruning methods or even higher. BLINKNET supports a variety of model approximation schemes to enforce the latency and memory bounds.

Research on resource-constrained DNNs. MCDNN is designed to replace a full DNN model with its approximated model considering memory size, energy budget, and cloud cost budget [32]. TVM is an end-to-end DNN optimization framework [22]. It uses compilers to reduce DNN latency on resource-constrained hardware. New DNN models have been designed for resource-constrained hardware. For example, MobileNets presents a class of DNN models for mobile and embedded vision applications. It allows users to make trade-offs between latency and accuracy [33]. However, none of them enforces latency bounds for real-time applications. Most recently, Bateni et al. designed the ApNet framework which enforces latency bounds for real-time applications. It maximizes the accuracy potential by applying different approximation schemes to individual layers of DNNs. ApNet does not consider resource constraints including memory and storage bounds, which should be enforced in resource-constrained systems. BLINKNET enforces both latency and memory bounds simultaneously. It is integrated with TVM to support a variety of hardware platforms leveraging its end-to-end compiling framework.

III. DESIGN OF BLINKNET

A. Overview of BLINKNET

The architecture of the BLINKNET framework is shown in Figure 1. It has four major components: *compiler*, *catalog database*, *scheduler*, and *performance interface*. They work coordinately to enforce latency and memory bounds specified by end-users of deep learning applications.

- **BLINKNET compiler**. Given a basic DNN model, the compiler is used to generate a set of approximation DNN models. The set of DNN models is managed as a DNN catalog. The compiler currently supports two model approximation schemes: *low-rank* (LR) [34] and *deep-compression* (DC) [35]. *Low-rank* performs low-rank decomposition in feature maps to replace a basis filter with a sequence of horizontal and vertical filters. *Deep-compression* applies pruning, quantization, and Huffman coding to reduce the storage requirement of DNNs. Given a new basic model of N layers, the BLINKNET compiler applies *low-rank* and *deep-compression* to layer i ($1 \leq$ i $\leq N$), respectively. For a DNN, initially, its catalog $Catalog$ includes only $Set_{LR} = \{DNN^i_{LR}\}$ where only layer i is compressed using the *low-rank* scheme and $Set_{DC} = \{DNN^i_{DC}\}$ where only layer i is compressed using the *deep-compression* scheme. All the approximation models in $Catalog$ are then executed given the training and testing datasets to get their characteristic profile $Profile_{DNN^i_x}(x \in \{LR, DC\})$ for models in Set_{LR} and Set_{DC}. Currently, their profiles include classification accuracy, memory cost, and inference latency. At runtime, the BLINKNET compiler will store new approximation models to $Catalog$ as specified by the BLINKNET scheduler.
- **Catalog database**. The $Catalog$ and $Profile$ data of a DNN is stored in a catalog database. We use an in-memory database to support low-latency retrieval of the approximation models and their corresponding profiles.
- **BLINKNET scheduler**. Given the user's input of latency and memory bounds, the scheduler determines an approximated model $Model_{approximated}$ to enforce the bounds. The algorithm used by the scheduler is described in Section III-B. Its output is $Model_{approximated} = \{Layer^i_x\}$, where $x \in [LR, DC]$, $1 \leq$ i $\leq N$, and $Layer^i_x$ denotes approximation scheme x is applied to layer i. For example, if its output is $Model_{approximated} = \{Layer^1_{DC}, Layer^2_{LR}, Layer^3_{LR}\}$, BLINKNET will apply *deep-compression* to layer 1 and *low-rank* to layer 2 and 3, respectively. If $Model_{approximated}$ is available in the catalog database, the BLINKNET runtime will use it for the deep learning application. Otherwise, the $Model_{approximated}$ will be generated online by the compiler and stored in the database for future reference. When the scheduler cannot enforce the bounds because of the low availability of system resources, it will serve a task on a best-effort basis.
- **Performance interface**. We implement the performance interface in the relay python frontend in Apache

```
1 n_layer = int(name.split('-')[1])
2 mod,params = relay.testing.vgg.get_workload(num_layers=n_layer,
..., latency_bound=latency, memory_bound=mem, group_ID=gid)
3 ...
```

Fig. 2. Performance interface of BLINKNET.

TVM [36]. An example is shown in Figure 2. Users can specify the latency and memory bounds when a network is created. Further, they need to provide a *group_ID*, which is assigned to correlated machine learning applications belonged to the same workflow. BLINKNET uses *group_ID* to further refine the allocation of system resources and alleviate performance interference between groups of applications. Providing a TVM-compatible interface will significantly accelerate the adoption of BLINKNET.

B. QoS-Aware Scheduling Algorithm

The general goal of the scheduling algorithm in BLINKNET is to analyze the approximated versions of DNNs (i.e., those in Set_{LR} and Set_{DC}) at a per-layer level in order to maximize accuracy while meeting the latency bound and other resource constraints. For example, if resource and accuracy information were known about a basic network model (e.g., VGG) and its approximated variants (e.g., DNN^i_{DC} and DNN^i_{LR}) the scheduling algorithm will exam layer by layer analyzing the trade-offs between accuracy and resource usage for each variant, deciding if an approximated version of the network should be used or not. The output of the scheduling algorithm would be $\{Layer^i_x\}$ ($x \in [LR, DC]$), a list of the layer configuration with which variant of the network to be used. Only one approximation algorithm would be applied to an individual layer. As the future work, we plan to support more compression algorithms in BLINKNET.

In order to determine which variant is optimal, it is necessary to develop a metric to quantify the trade-off between resources saved and accuracy lost. We define a layer's *time-saving potential* ($Time\_potential^i_x$) as

$$Time\_potential^i_x = \frac{\Delta Time^i_x}{Approx(\%)},$$

where

$$\Delta Time^i_x = \frac{Time^i_x - Time^i_{baseline}}{Time^i_{baseline}}$$

Here, $Time^i_x$ is the execution time of layer i of the approximated model using approximation scheme x. $Time^i_{baseline}$ is the execution time of layer i of the corresponding basic model. $Approx$ is percentile accuracy loss. Similar values can be calculated for other resources, such as memory (*memory saving potential*). Let $Size^i_x$ be memory usage of layer i, then

$$Mem\_potential^i_x = \frac{\Delta Size^i_x}{Approx(\%)},$$

where

$$\Delta Size^i_x = \frac{Size^i_x - Size^i_{baseline}}{Size^i_{baseline}}$$

These saving potentials can be combined to produce a more general *resource-saving potential* for users having performance requirements on other dimensions (e.g., energy budget).

$$\Pi^i_x = Time\_potential^i_x + Mem\_potential^i_x$$

This allows the future inclusion of other resources and the potential to weight different resources as more important in the BLINKNET framework.

Algorithm 1: BLINKNET scheduler

Input: $Time\_limit$: time bound of the task;
$Memory\_limit$: memory bound of the task;
$Time^i_j$ and $Size^i_j$: execution time and memory cost of layer i of the approximated model using approximation scheme j.
Output: $MinSet^i$: configuration of layer i.

```
// Choose the approximation schemes which meet the memory bound
for layer i in 1, ..., n do
    Set^i ← LR ∪ DC;
    for j in LR, DC do
        if Size^i_j > memoryLimit then
            Set^i ← Set^i − j;
totalTime ← 0;
temp ← INF;
// Choose the approximation scheme which has the minimum execution time
for layer i in 1, ..., n do
    MinSet^i ← NULL;
    for j in Set^i do
        temp ← min(Time^i_j, temp);
        if temp = Time^i_j then
            min ← j;
    MinSet^i ← MinSet^i + j;
    totalTime ← totalTime + temp;
// Improve the accuracy after meeting the latency and time bounds
timeLeft ← Time_limit − totalTime;
while timeLeft > 0 do
    temp ← INF;
    // Choose the layer which has the minimum time-saving potential
    for layer i in 1, ..., n do
        temp ← min(Time_potential^i, temp);
        if temp = Time_potential^i then
            m ← i;
    MinSet^m ← NULL;
    // For layer m, select the scheme that has the best accuracy
    n ← best_accuracy(m);
    timeLeft ← timeLeft + Time^m_{MinSet^m} − Time^m_n;
    if timeLeft > 0 then
        MinSet^m ← MinSet^m + n;
```

We design a heuristic algorithm that achieves an optiaml schedule given the list of possible compression algorithms for each layer. We describe the scheduling algorithm of

BLINKNET in Algorithm 1. The algorithm takes as inputs a time limit, a memory bound, and execution time and memory cost of each layer of the approximated models. Its output $MinSet^i$ is a list of configurations (i.e. approximation schemes) of individual layers. First, the algorithm needs to choose the configurations Set^i which meet the memory bound for each layer (Lines 1-5). Then it determines the configuration which has the minimum execution time (Lines 8-15). The configuration data is stored in $MinSet^i$. Finally, without violating the time bound, it further improves the accuracy of the approximated models using time-saving potentials (Lines 16-26). Specifically, it identifies the configuration of layer m which has the minimum time-saving potential. Then it replaces m with the configuration n, which may lead to higher accuracy. Then n replaces m in $MinSet^i$. BLINKNET keeps the accuracy improvement process until the total time cost is larger than the time bound.

IV. EVALUATION

Basic DNN models. We implement the BLINKNET prototype in Python and evaluate it using two basic DNN models, including VGG [23] and Cifar10-quick [24]. Table I shows the configurations of the two models. It also shows the maximum accuracy loss after applying the *low-rank* and *deep-compression* approximation schemes to all the layers of the models, respectively.

TABLE I
DNN CONFIGURATION EVALUATED IN THIS PAPER.

DNN	*Layers*	*Accuracy loss*
VGG16	16	≈1%
Cifar10-quick	3	≈5%

Model catalogs. In the experiments, we utilize caffe [37] to train our models and run the models through TVM's AutoRelay [22] and BLINKNET compiler to get the catalog of per-layer approximation DNN models and their profiles. Let's take VGG as an example. We create 16 models by applying *low-rank* to one layer of VGG [23] at a time, only adjusting one layer per model. Then we conduct a similar process again using *deep-compression*. It is an unstructured pruning algorithm, which zeroes out the lowest magnitude weights. It also uses 256 bits for quantization. The process ends with 32 total models in the catalog of VGG.

Datasets. We choose two datasets in the evaluation. The CIFAR10 dataset consists of 60000 32*32 color images in 10 classes with 6000 images per class [38]. The dataset is divided into five training batches and one test batch. The MNIST dataset comes from a database of handwritten digits [39]. It contains 60,000 training images and 10,000 testing images.

System configuration. The computer system has an Intel processor of Core 2 Duo CPU E8400 3.00 GHz and 16 GB DRAM. We run Ubuntu 18.04.5 LTS on the server with the latest version of Caffe and TVM 6.0. We use this server to emulate different types of resource-constrained computing systems with varying capacity of DRAM and CPU speed through dynamic voltage scaling [40]. We only use CPUs in the evaluation in the paper. As future work we will study its performance on GPU platforms.

A. Performance with VGG

TABLE II
EXECUTION TIME OF VGG WITH DIFFERENT LATENCY BOUNDS.

Latency bound	*Memory bound*	*Measured latency*
800 ms	4.25 MB	808 ms
1000 ms	4.25 MB	997 ms

In this experiment, we use VGG with the CIFAR10 dataset. We set the latency bounds as 800 ms and 1000 ms respectively and memory bound as 4.25 MB. As shown in Table II, BLINKNET achieves the performance goals of users with up to 1% accuracy loss. The assignment of the approximation schemes for each layer of VGG is presented in Table III.

TABLE III
ASSIGNMENT OF PER-LAYER APPROXIMATION SCHEMES WHEN THE LATENCY BOUNDS ARE 800 MS AND 1000 MS RESPECTIVELY. B: LATENCY BOUND; DC: DEEP COMPRESSION; LR: LOW RANK; AND NA: NO APPROXIMATION.

Layers	*B=800ms*	*B=1000ms*
1	DC	DC
2	LR	LR
3	NA	LR
4	LR	LR
5	LR	LR
6	LR	LR
7	LR	LR
8	LR	LR
9	DC	DC
10	DC	LR
11	DC	DC
12	DC	DC
13	LR	LR

B. Performance with Cifar10-quick

We evaluate BLINKNET using the Cifar10-quick model with both CIFAR10 and MNIST datasets. The results are shown in Table IV. We can observe that BLINKNET enforces the latency bounds given both of the datasets. Its per-layer assignment is given in Table V.

TABLE IV
EXECUTION TIME OF CIFAR10-QUICK WITH DATASETS.

Dataset	*Latency bound*	*Memory bound*	*Measured latency*
CIFAR10	60 ms	4.2 MB	54 ms
MNIST	6 ms	3.8 MB	5 ms

TABLE V
ASSIGNMENT OF PER-LAYER APPROXIMATION SCHEMES FOR THE CIFAR10 AND MNIST DATASETS RESPECTIVELY. B: LATENCY BOUND; DC: DEEP COMPRESSION; LR: LOW RANK; AND NA: NO APPROXIMATION.

Layers	*CIFAR10*	*MNIST*
1	NA	DC
2	NA	LR
3	LR	LR

V. CONCLUSION

This paper presents BLINKNET, a user-configurable deep learning framework. It leverages the per-layer analysis of the approximated DNN models to meet user's performance requirements while maximizing the accuracy of approximated models given the resource constraints. Our evaluation results with VGG and Cifar10-quick show that BLINKNET can automatically and efficiently enforce the performance policies specified by end-users.

For future work, we plan to support more compression schemes for building the model catalog. We will evaluate BLINKNET with other state-of-the-art networks (i.e., ResNet and MobileNet) and datasets (i.e., MNIST and ImageNet). We will also use other hardware platforms (i.e., FPGAs and GPUs) for the evaluation.

ACKNOWLEDGMENT

We are grateful to the anonymous reviewers. This research was supported by US National Science Foundation under CNS 1906541. This work was also funded in part by DoE Electricity Industry Technology and Practices Innovation Challenge.

REFERENCES

[1] C. Seifert, A. Aamir, A. Balagopalan, D. Jain, A. Sharma, S. Grottel, and S. Gumhold, "Visualizations of deep neural networks in computer vision: A survey," in *Transparent Data Mining for Big and Small Data*. Springer, 2017, pp. 123–144.

[2] X. Liu, P. He, W. Chen, and J. Gao, "Multi-task deep neural networks for natural language understanding," *arXiv preprint arXiv:1901.11504*, 2019.

[3] H. A. Pierson and M. S. Gashler, "Deep learning in robotics: a review of recent research," *Advanced Robotics*, vol. 31, no. 16, pp. 821–835, 2017.

[4] Y. Deng, "Deep learning on mobile devices: a review," in *Mobile Multimedia/Image Processing, Security, and Applications 2019*, vol. 10993. International Society for Optics and Photonics, 2019, p. 109930A.

[5] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2016, pp. 1–12.

[6] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[7] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Computing*, vol. 16, no. 3, pp. 82–88, 2017.

[8] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[9] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[10] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[12] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[13] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[14] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 8612–8620.

[15] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[16] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[17] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[18] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 925–938.

[19] X. Ma, F.-M. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang, "Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices." in *AAAI*, 2020, pp. 5117–5124.

[20] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 907–922.

[21] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 67–79.

[22] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Q. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: end-to-end optimization stack for deep learning," *arXiv preprint arXiv:1802.04799*, vol. 11, p. 20, 2018.

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

[24] "The cifar10-quick model," Website, https://github.com/BVLC/caffe/tree/master/examples/cifar10.

[25] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[26] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances in neural information processing systems*, 2016, pp. 1379–1387.

[27] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 184–199.

[28] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.

[29] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 1–18, 2017.

[30] T. Zhang, K. Zhang, S. Ye, J. Li, J. Tang, W. Wen, X. Lin, M. Fardad, and Y. Wang, "Adam-admm: A unified, systematic framework of structured weight pruning for dnns," *arXiv preprint arXiv:1807.11091*, vol. 2, no. 3, 2018.

[31] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the regularity of sparse structure in convolutional neural networks," *arXiv preprint arXiv:1705.08922*, 2017.

[32] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 123–136. [Online]. Available: https://doi.org/10.1145/2906388.2906396

[33] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[34] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[35] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[36] "Introduction to relay ir," Website, https://tvm.apache.org/docs/dev/relay_intro.html.

[37] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[38] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[39] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[40] C.-H. Hsu and W. Feng, "Effective dynamic voltage scaling through cpu-boundedness detection," vol. 3471, 01 2004, pp. 135–149.