


2018 INFORMS International Conference
Taipei, Taiwan, June 17–20, 2018
c 2018 INFORMS|ISBN 978-0-9906153-1-6
https://doi.org/10.1287/infp.2018.XXXX
pp. 000–000

# Multi-Stage DNN Model Compression for IoT-enabled Edge Devices

Ching-Hu Hu
Electrical Engineering Department, National Taiwan University of Science and Technology (NTUST)
No.43, Keelung Rd., Sec.4, Da'an Dist., Taipei 106, Taiwan, R.O.C.
{jhluh@ieee.org}

Jen-Wei Wang
Electrical Engineering Department, National Taiwan University of Science and Technology (NTUST)
{onlywe0620@gmail.com}

Abstract With the advances in edge computing for the Internet of Things (IoT) applications, increasing researchers have tried to reduce the parameters and size of a deep neural network (DNN) model through mostly single-stage model compression. This enables a resource-constrained edge device to accommodate a compressed DNN model and to mitigate the computing burden of its back-end servers. However, existing researches often faced the problem of poorer accuracy due to model compression. To address the issue, our study proposed multi-stage model compression, which compresses a DNN meanwhile maintaining as much accuracy. The experiment results have shown that the accuracy of the resultant compressed model through the multi-stage model compression is better than those only used the one-stage compression, given the same model size.

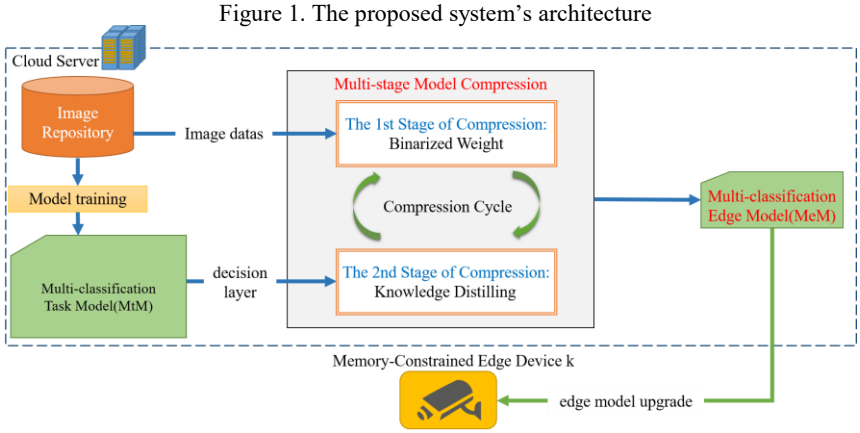
Keywords multi-stage model compression; deep neural network; edge computing; IoT

## 1. Introduction

With the advances in deep neural network (DNN), a wide range of video monitoring has been widely used in various smart-living scenarios. However, a large number of video-surveillance systems will generate plethora of data, which can be represented by increasing depth of DNN models, yet leading to growing parameters and more complicated structures of the DNN models. If all images of a video-surveillance system completely rely on traditional back-end (cloud) servers to process, the back-end servers and the bandwidth will eventually be over-whelmed. Therefore, with the development of Internet of Things (IoT) applications, the demand for edge computing is quickly increasing. One advantage of edge computing is that its decisions and services can be closer to its users, which makes uploading raw images to the servers unnecessary for video-surveillance applications over the Internet to the cloud. This design can significantly reduce transmission bandwidth, thus increasing decision-making speed and energy saving.

Although a DNN model often has high accuracy in image recognition, its computing requirement cannot be met by most existing edge devices. In order to let a resource-constrained edge device run a DNN model, increasing compression methods have been proposed. For example, Han et al. [1] proposed Deep Compression to make their DNN models executable on mobile devices. Hinton et al. [2] proposed Knowledge Distilling, which requires a teacher model and a student one. The student model, with a more simplified DNN structure, can learn from the teacher model to improve its accuracy, and can directly reside on a mobile device. Hubara et al. [3] proposed Binarized Neural Networks (BNN) with their weights binarized to reduce the model size so that they can fit on memory-constrained edge devices.

Figure 1. The proposed system's architecture




The above studies have shown that BNN models can be qualified for edge devices yet at the cost of compromising accuracy. In addition, most previous studies made only one kind of compression method. In order to more comprehensively compress DNN models for an edge device meanwhile maintain as much accuracy, we propose multi-stage model compression, is divided into two cyclic stages in the current phase.

## 2. The proposed system

The proposed architecture of this study is illustrated in Figure 1. Before the execution of the cyclic multi-stage model compression, the system needs a deep network model, which hereafter is referred to a multi-classification task model (MtM). As a techer model, the MtM is trained by more completed data, so it often can have higher accuracy but with more complex DNN structure. Since the MtM often demands high memory space, it often needs to execute on a cloud-level server.

As for the first stage of compression, the compression is completed by binarizing all weights in a DNN model. Each weight will be binarized to either +1 or -1 via the banalization function [4] as follows:

$$w_b = Sign(w) = \begin{cases} +1 & \text{if } w \geq 0, \\ -1 & \text{otherwise.} \end{cases} \quad (1)$$

where $w$ is a weight of the original DNN model, and $w_b$ is its binarized weight. After all weights are binarized, each output layer needs to adjusted using the equation below:

$$y^n = BatchNorm(y^{n-1} * w_b^{n-1}) \quad (2)$$

where $y^n$ is the $n$-th layer's output and *BatchNorm*() is the batch normalization method proposed in [5]. Batch normalization can avoid propagating larger output values to the next layer, which may often causes gradient vanishing, and also accelerates the training process. Through this compression stage, the size of the model can be effectively reduced. However, using this compression method often compromises the accuracy. Therefore, this study incorporates another compression method as the second compression stage to mitigate the issue.

The 2nd compression stage uses knowledge distilling [2], which has proven its usefulness in helping simpler DNN models perform better. In a traditional training process, training labels are the ground truth, also known as hard targets. But, this distilling method uses soft targets generated by an MtM as the ground-truth to train another model, later referred to as a multi-classification edge model (MeM). An MeM is trained using the same dataset. Distilling knowledge from the MtM, a compressed MeM took the simpler (or smaller) form of the MtM but with less neurons

Soft targets are actually the output probability of the softmax layer of the MtM. This study names the softmax layer as the decision layer, each of its output is defined as:

$$s_i = \frac{\exp(z_i / T)}{\sum_{j=1}^{C} \exp(z_j / T)}, \; i \in \{1, \ldots, C\} \quad (3)$$

TABLE I. Accuracy and model size comparison

| Epoch = 300, Batch size=1000, Learning rate = 0.001 | | | |
|---|---|---|---|
| **Model name** | **Testing Accuracy(%)** | | **Model size(MB)** |
| | **Training without MtM** | **Training with MtM** | |
| NN_1200 | 97.82 (MtM alone as the baseline) | | 9.14 |
| BNN_1200 | 89.31 | 92.05 | 9.14 |
| BNN_800 | 88.67 | 91.43 | 4.89 |
| BNN_600 | 87.78 | 90.78 | 3.21 |

where $i$ is an index of a class, $s_i$ is the probability of class $i$, and $z_i$ is an output value from the final output layer via a logit function. Knowledge distillation introduces a temperature parameter $T$ for the decision layer. By adjusting this parameter, the mapping curve of the decision layer become more smooth, so the distribution of the mapping probabilities of a hard target output will be more concentrated and smoother, and this is why a soft target is named from.

Before the model compression, the temperature parameter $T$ of the decision layer of the MtM need adjustment to determine the probabilities of the soft-target labels, which will serve as the ground truth for an MeM. Next, the MeM also uses the same temperature parameter to evaluate its loss, as defined below:

$$Loss = \alpha \cdot soft_loss + (1-\alpha) \cdot hard_loss \quad (4)$$

where *soft_loss* is a loss reckoned using the soft-target labels from the MtM, and *hard_loss* is a loss using the original ground truth. The hyperparameter $\alpha$ controls the learning ratio. Through the help from the 2nd compression stage, it is possible to further improve the accuracy and reduce the size of a DNN model.

## 3. Experiment Results

The multi-stage model compression was implemented using PyTorch [6], which is a popular deep learning toolkit in Python. It supports most commonly used calculation libraries for deep learning and has many built-in optimization and loss functions. Besides it can execute on different types of CPUs or GPUs. Its *model export* function supports the open-source neural network exchange (ONNX) format. This feature enables us to easily migrate a DNN model to different deep learning toolkits that also support ONNX (e.g., Caffe2 or MXNet).

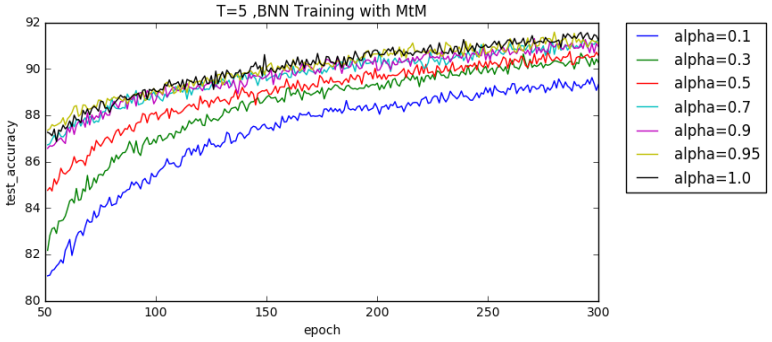
Our study used the MNIST [7] dataset, a commonly used image benchmark. The experiment consists of 60K training data and 10K testing data. Each data is a 28 × 28 gray-scale image represented by digits ranging from 0 to 9. Before the multi-level model compression, we used the MNIST dataset to train a MtM, which takes the form of a Multilayer Perceptron (MLP) consisting of 2 hidden layers of 1,200 neurons, referred to as NN_1200. In the evaluation, the settings of an MeM include BNN_1200, BNN_800, and BNN_600. To optimize model parameters, all models used stochastic gradient descent (SGD) [8], and the loss evaluation in (4) is cross entropy.

We first evaluated the accuracy of the model to see if its performance through the multi-stage model compression is better than that only used binarized weights. The result is summarized in TABLE I, where the numbers of the epoch, batch size, and learning rate are determined by a pilot test. The results show that through the multi-stage model compression, all of the MeM models outperform those only used one-stage compression. With the help from the MtM, a smaller sized model can even outperform those with larger model sizes without the knowledge from the MtM (e.g., BNN_600 outperforming BNN_1200).

Next, we compared the accuracy using different $\alpha$ in the loss function (4). The result show in Figure 2, where the BNN_800 (MeM) with a fixed temperature parameter was evaluated. We can observe that higher $\alpha$ leads to better testing accuracy, but the highest accuracy occurs when $\alpha$ is 0.95. Therefore, we conclude that it is helpful to train an M2M model by utilizing the knowledge from a teacher. However, allowing the MeM to use the hard-target data rather than purely relying on soft-target data could further improve performance due to allowing exploration of the solution space.

As for the reduction in model sizes, because this study made use of the existing popular deep-learning toolkits (e.g., PytTorch, Tensorflow and Keras), which support only real-number rather than bitwise model format as well as parameter representation. Therefore, additional effort needs to made

Figure 2. Compare different $\alpha$ in BNN_800




on the realization of bitwise BNN. Theoretically, NN_1200 and BNN_1200 are the same model size. If the toolkits support bit-wise representation, the size of a compressed DNN model can be reduced by up to 32 times.

## 4. Conclusion and Future work

In order to enable a DNN model to fit on a memory-constrained edge device, our study proposed a cyclic multi-stage model compression. The experiment result has shown that the accuracy of the resultant DNN models through the proposed approach is better than those only used binarized weights compression. Our method can indeed improve the accuracy of a BNN, and also allows the BNN to have a good accuracy with a smaller memory requirement. The experiment results also showed that the MeM can perform better through utilizing the knowledge from the MtM.

In the Future work, we will test this method in different datasets and different DNN models. We will also develop bit-wise model representation to reduce model size, and the resultant compressed the MeM will be tested on a real memory-constrained edge device.

## 5. Acknowledgement

This work was supported by the Ministry of Science and Technology, Taiwan under Grants MOST 106-2221-E-011-029.

## Reference

[1] 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.

[2] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531,* 2015.

[3] 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.

[4] 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.

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167,* 2015.

[6] A. Paszke, S. Gross, and A. Lerer, "Automatic differentiation in PyTorch," in *Workhop in neural information processing systems*, 2017.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, pp. 2278-2324, 1998.

[8] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *arXiv preprint arXiv:1606.04838,* 2016.