24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

# On Minimizing Diagonal Block-wise Differences for Neural Network Compression

**Yun-Jui Hsu**[1] and **Yi-Ting Chang**[1] and **Chih-Ya Shen**[1] and **Hong-Han Shuai**[2] and **Wei-Lun Tseng**[2] and **Chen-Hsu Yang**[1]

**Abstract.** Deep neural networks have achieved great success on a wide spectrum of applications. However, neural network (NN) models often include a massive number of weights and consume much memory. To reduce the NN model size, we observe that the structure of the weight matrix can be further re-organized for a better compression, i.e., converting the weight matrix to the *block diagonal structure*. Therefore, in this paper, we formulate a new research problem to consider the structural factor of the weight matrix, named *Compression with Difference-Minimized Block Diagonal Structure (COMIS)*, and propose a new algorithm, *Memory-Efficient and Structure-Aware Compression (MESA)*, which effectively prunes the weights into a block diagonal structure to significantly boost the compression rate. Extensive experiments on different models show that MESA achieves $135\times$ to $392\times$ compression rates for different models, which are $1.8$ to $3.03$ times the compression rates of the state-of-the-art approaches. In addition, our approach provides an inference speed-up from $2.6\times$ to $5.1\times$, a speed-up up to $44\%$ to the state-of-the-art approaches.

## 1 Introduction

With the acceleration of matrix processing brought by the graphic processing unit (GPU), deep neural networks have achieved great success and enabled a wide spectrum of applications. However, deep learning models usually include a massive number of weights (e.g., 240 MB for ResNet-152, 248 MB for AlexNet, and 552 MB for VGG16 on ImageNet dataset) and consume much memory and transmission bandwidth, especially for mobile devices or embedded platforms [13]. Therefore, reducing the model size has been an active research topic in recent years. To reduce the model size, different approaches have been proposed, such as [1], [18], [25], [23], [24], which compress the model with limited degradation in inference accuracy.

However, we observe that the weight matrices can be reorganized for a better compression. Specifically, conventional approaches only consider to preserve the important weights[3], which then prune the unimportant ones and re-train the model [6], [25], [24]. However, the remaining non-zero weights are randomly scattered in the weight matrices, and the irregular sparse matrices lead to inefficient computation and ineffective compression.

In contrast to the prune-and-retrain approaches above, a recent work [3] proposes that dense feed-forward networks contain sub-networks (*lottery ticket hypothesis*) that are able to reach comparable test accuracy to the original network when *trained from scratch* with a similar number of iterations, i.e., removing certain weights before training. On top of that, Supic et al. [22] show that *pruning the weights arbitrarily before training without considering their importance* (*arbitrary pruning* for short) retains nearly the same performance of the model (detailed in Sec. 3). Inspired by these works, in this paper, we explore the new idea to arbitrarily prune the weights in fully-connected layers into the *block diagonal structure* before training, i.e., organizing the non-zero weights as blocks on the diagonal in the matrix, in order to significantly improve the compression rate and reduce the model inference time.

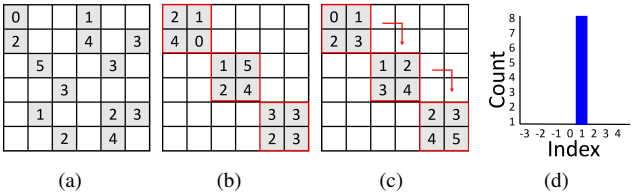



**Figure 1.** Motivating example. (a) Irregular structure. (b) Block diagonal structure. (c) Preferred structure. (d) Distribution of Fig. 1(c).

Consider Fig. 1 as an example. Fig. 1(a) is an $n \times n$ sparse matrix (storing the weights of the NN). While compressed sparse row (CSR) and compressed sparse column (CSC) formats can be employed to reduce the storage overhead, organizing the weights in a *block diagonal structure* is more promising in terms of compression rate and inference efficiency. That is, if Fig. 1(a) is stored in CSR format, an overhead of 13 indices and 6 pointers are required on top of the weight values. In contrast, when the non-zero weights are organized in same-sized blocks in the diagonal as in Fig. 1(b), dense blocks can be stored by appending each dense block together sequentially, requiring no extra indices and pointers. In adition, for inference efficiency, each block can be computed in parallel on a GPU device to its own corresponding column, which significantly boosts the efficiency of model inference. For example, the block diagonal structure in Fig. 1(b) reduces the computation complexity from $O(n^2)$ to $O(\frac{n^2}{3})$ for real-time processing. We also show in the experiments section that pruning into block diagonal structure shows a speed-up up to $44\%$ compared to the unstructured pruning method [21] for model inference.

Further, we observe that organizing the original sparse matrix to block diagonal structure still leaves a large room for improvement. We argue that the entropy of the weight distribution in blocks should

[1] National Tsing Hua University, Taiwan. Email: chihya@cs.nthu.edu.tw.
[2] National Chiao Tung University, Taiwan. Email: hhshuai@nctu.edu.tw, erictseng.eed06g@nctu.edu.tw.
[3] In this paper, we use *weight* and *parameter* interchangeably.

also be well addressed to further compress the model. Consider a given weight matrix that the weights are quantized to 3 bits, with representing index in $[0, 7]$, as shown in Fig. 1(c). Fig. 1(c) is well arranged with correlated small entropy (the entropy value is 0) with all the differences as 1 on all the $2 \times 2$ blocks, whereas the entropy value of Fig. 1(b) is $2.46$ along the diagonal. By doing so, encoding by the blocks' pair-wise index differences in Fig. 1(c) leads to a concentrated distribution as shown in Fig. 1(d). By applying lossless compression, such as Huffman coding, we only need $8$ bits to encode the remaining matrix, while those in Figs. 1(a) and 1(b) require $82$ (with CSR and Huffman) and $30$ bits, respectively.

With the observations above, in this paper, we aim to reduce the model size by considering the block diagonal structure and the minimum data entropy of sequential block differences (*minimum block-wise differences* for short). Unlike most existing approaches, such as pruning and quantization [1], [23], [16], [19], [6], [25], [24], that focus only on removing redundant weights or reducing the allocated bits for weights, we make the first attempt to consider the structural properties of the matrix to effectively reduce the memory consumption, which is particularly useful for embedded systems and mobile devices. To our best knowledge, this is the first work that considers the block diagonal structure with minimum block-wise differences to compress the model. It is worth noting that, although we present our results for fully-connected layers in this paper, our ideas can be easily extended to compress convolutional layers.

Specifically, we formulate a new research problem, named *COmpression with Difference-MInimized Block Diagonal Structure (COMIS)* that takes as input an NN model and transforms it to a memory-efficient representation (i.e., compressed model) with a limited accuracy drop. We design a new algorithm, named *Memory-Efficient and Structure-Aware Compression (MESA)*, which employs the ideas of *lottery ticket hypothesis* and *arbitrary pruning* with a new penalty function to minimize the block-wise differences in the block diagonal structure to significantly improve the compression rate. We conduct extensive experiments to validate our ideas and evaluate our algorithm. Our algorithm is able to achieve compression rates of $135\times$ to $392\times$ on various models, tripling the compression rates of the state-of-the-art approaches [21], [22]. For the inference time, the models compressed by MESA achieve $2.6\times$ to $5.1\times$ speed-up compared to the full precision model and up to $44\%$ speed-up compared to the state-of-the-art NN compression approaches [21], [22]. The contributions of this paper are summarized below.

- We observe that the block diagonal structure with the minimized block differences is able to boost the compression rate and the inference speed for NN model compression, and we validate our observations with analysis on multiple models.
- We propose a new research problem, named *Compression with Difference-Minimized Block Diagonal Structure (COMIS)* problem with a new algorithm, *Memory-Efficient and Structure-Aware Compression (MESA)*, to significantly improve the compression rate. This is the first work that compresses the NN model with a difference-minimized block diagonal structure, which is complementary with most existing compression methods.
- Extensive experiments on multiple models are conducted. The results show that MESA achieves $135\times$ to $392\times$ compression rates, tripling those of the state-of-the-art approaches. Additionally, the models compressed by MESA achieve $2.6\times$ to $5.1\times$ speed-up (inference time) compared to the full precision model, and up to $44\%$ of speed-up compared to the state-of-the-art approaches.

This paper is organized as follows. Sec. 2 discusses the relevant works. Sec. 3 formulates the problem and discusses our observations. Sec. 4 details the algorithm design, and Sec. 5 presents the experimental results. Finally, Sec. 6 concludes this paper.

## 2 Related Works

Compressing NN models has been actively studied in the past few years. The main objective is to minimize the memory consumption in the inference phase with a very limited accuracy drop, where the works can be basically categorized into *quantization*, *pruning*, and *low-rank* approaches.

The quantization-based approaches, such as binarization quantization [1], [23] and fixed-point quantization [16], [5], use fewer bits to represent the original weights for reducing the memory consumption. In addition, approaches that employ sharing index such as [21], [19] are also proposed to enable an efficient lookup mechanism for performance improvements.

For pruning-based approaches [6], [24], [25], [26], the less important weights in the model are pruned to reduce the complexity. By putting pruning and quantization together, DeepCompression [21] proposed a three-stage pipeline with pruning, quantization, and Huffman coding for the quantized index. Further, MPDCompress [22] proposes to store the sparse weights as diagonal blocks to facilitate parallel computation on GPU devices. However, since none of the above works considers the structural factor to minimize the block-wise differences in the weight matrix, our ideas and algorithms are complementary to these works for improving the compression rate.

In addition, low-rank matrix factorization and tensor decomposition are also employed for compression [15], [11]. Also, a recent line of studies aims at reducing the vast convolutional layers by reformulating its standard structure, such as SqueezeNet [9], MobileNet [7], and EffNet [4].

To summarize, although the above works achieve good performance, however, they do not consider the block diagonal structure with the minimum block-wise differences, which are critical to further reduce the model size. In this paper, we make the first attempt to optimize the model compression by incorporating these ideas.

## 3 Problem Formulation and Analysis

**Problem Formulation.** The COMIS problem is formulated as follows. We are given a neural network with $n$ fully-connected layers[4], a set of target pruning rates $\mathbb{P} = \{p_1, \cdots, p_n\}$ and a set of quantization bits $\mathbb{Q} = \{q_1, \cdots, q_n\}$ for each fully-connected layer, where $p_i$ of weights should be kept (i.e., $(1-p_i)$ of weights should be pruned) in each layer $1 \leq i \leq n$, and all the weights in layer $i$ are quantized from 32-bit floating point numbers to $q_i$ bits of weight index[5]. The goal of COMIS is to minimize the model size with a limited accuracy drop, such that the non-zero weights for each fully-connected layer are organized as non-overlapping blocks in the diagonal of the weight matrix (*diagonal requirement*), and the block-wise differences of the adjacent blocks should be small (*min-difference requirement*).

Here, the parameters $\mathbb{P} = \{p_1, \cdots, p_n\}$ and $\mathbb{Q} = \{q_1, \cdots, q_n\}$ can be assigned to consider different scenarios. A smaller $p_i$ or $q_i$ may lead to a smaller model size. However, when $p_i$ or $q_i$ is set too small, it becomes more difficult (or infeasible) to compress the model

[4] Even if the model contains both fully-connected and convolutional layers, the proposed COMIS problem and algorithm can still be applied to compress the fully-connected layers. Moreover, our approach can be easily extended to compress the convolutional layers by slightly changing the penalty function.

[5] $\mathbb{P}$ and $\mathbb{Q}$ can be set according to [21] and will be discussed in the experiments section.

with a limited accuracy drop. Since previous work has shown that the performance remains the same if the neural networks are compressed with the same $\mathbb{P}$ and $\mathbb{Q}$, the settings of $\mathbb{P}$ and $\mathbb{Q}$ are based on the previous compression results by other approaches in practice, such as DeepCompression [21], MPDCompress [22]. Please note that as shown in the experimental results, our proposed approach significantly outperforms DeepCompression and MPDCompress under the same $p_i$ and $q_i$ settings.

In this paper, our proposed algorithm is able to prune the network more effectively to enhance the compression rate by satisfying the two requirements above, i.e., *diagonal requirement* and *min-difference requirement*, which are added to allow run-length coding to significantly reduce the model size, as illustrated in Fig. 1. These requirements are designed to follow our observations below. Please note that, although our observations and ideas are presented with fully-connected layers, the proposed ideas can be easily extended to compress the convolutional layers.

**Observations and Analysis.** As mentioned earlier, arranging the sparse matrix (representing the weights of a fully-connected layer) to a block diagonal structure is critical for effectively reducing the model size. However, reducing the model size while satisfying the diagonal requirement is very challenging because conventional pruning-based approaches, such as [25], [24], [26], usually fail to generate a pruned weight matrix where non-zero weights are grouped into blocks on the diagonal. This is because they aim to prune out the less important weights defined in their work, but such less important weights often scatter irregularly in the weight matrix.

Recently, Frankle et al. and Liu et al. [17], [3] both point out that given the assigned pruned model structure from the conventional pruning-based approaches, the pruned network structure can be trained from scratch and provide a similar or even better model accuracy. On top of that, Supic et al. [22] demonstrate the effectiveness of arbitrary pruning strategy on fully-connected layers. In their experiments, they show that given the percentage of weights to be pruned from each layer, one can arbitrarily prune off that number of weights (i.e., the *arbitrary pruning* strategy) and train the pruned network from scratch to provide a model accuracy comparable with the original full-precision model. This observation motivates us to explore a new direction to effectively prune the weight matrix in order to extract the winning ticket subnetwork [3] (i.e., the subnetwork that matches the test accuracy of the original network when trained in isolation from scratch) from the original network while satisfying the diagonal requirement of the COMIS problem.

To validate and gain more insights for the arbitrary pruning strategy, we conduct a preliminary experiment on 4 models, i.e., LeNet-5 [14], VGG16 on CIFAR-10 [20], VGG16 on CIFAR-100 [20], and AlexNet [12][6]. For each model, we apply one of the state-of-the-art compression approaches, DeepCompression [21], and record its pruning rate for each individual fully-connected layer without any accuracy drop. The second column of Table 1 reports the pruning rates of DeepCompression, which are then used as the hyperparameters for testing other approaches. For example, the pruning rates for LeNet-5 are $10\%$ and $20\%$, indicating that DeepCompression keeps $10\%$ and $20\%$ of weights in the first and second fully-connected layers, respectively. Moreover, in this pilot experiment, all the weights for the compared approaches are quantized to 5 bits. For each model, we compare the performance of three approaches: i) DeepCompression (`DeepC`) [21]; ii) The arbitrary pruning strategy to prune the fully-connected layers randomly with the same pruning rate as in DeepCompression for each individual layer (`Arb`). iii) Moreover, we prune the weights not residing in the blocks on the diagonal with the same pruning rate as in DeepCompression, which is a special case of the arbitrary pruning strategy (referred to as `Diag`).

[6] We also present the experimental results on non-computer vision models later in the experiments section.

**Table 1.** Preliminary results for different pruning strategies

| | Pruning rates | `DeepC` | `Arb` | `Diag` |
|---|---|---|---|---|
| LeNet-5 | 10%, 20% | 99.35 | 99.03 | 99.11 |
| VGG16 CIFAR-10 | 12.5%, 12.5%, 10% | 91.38 | 91.1 | 91.44 |
| VGG16 CIFAR-100 | 12.5%, 12.5%, 10% | 72.33 | 72.35 | 72.12 |
| AlexNet | 10%, 10%, 25% | 53.23 | 54.16 | 53.37 |

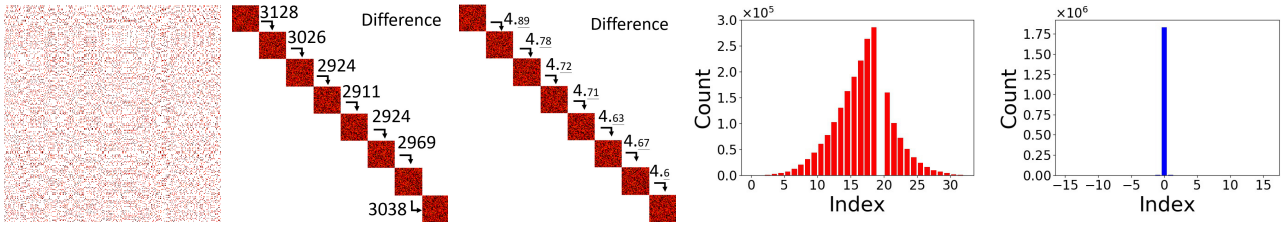
The results are reported in Table 1, where the third, fourth, and fifth columns in Table 1 respectively present the accuracy of the model compressed by `DeepC`, `Arb`, and `Diag`, all with the same number of weights. `Diag` is very close to `DeepC` and `Arb` in all models, which manifests that arbitrary pruning the same ratio of weights to generate block diagonal structure produces similar or better accuracy in these models. This observation validates the feasibility and effectiveness of transforming the weight matrix into a block diagonal structure for better model compression. Based on this important observation, we propose the algorithm for COMIS in the next section.

## 4 Algorithm Design

To solve the proposed COMIS problem, in this paper, we propose algorithm *Memory-Efficient and Structure-Aware Compression (MESA)*, which includes four steps as detailed below.

**Step 1. Pruning with Block Diagonal Mask (PM).** Given the pruning rate $p_i$ of the $i$-th weight matrix, the conventional pruning approaches [25], [24], [26], [6] identify the importance of the weights and then prune the less important ones until the desired number of weights are pruned. Although these approaches may be able to retain the accuracy eventually, there is no guaranteed structure for the weight matrix after the pruning. Consider a running example in Fig. 2, given the weight matrix of size $4096 \times 4096$ and a pruning rate of $12.5\%$, a trivial pruning approach that keeps $12.5\%$ of weights without considering the block diagonal structure may obtain the irregular sparse matrix as shown in Fig. 2(a). However, as mentioned earlier, the block diagonal structure is critical to model efficiency. Therefore, in the first step, after the pruning rate of each layer is identified by the threshold pruning method (e.g., [6]), we directly train the block diagonal sparse matrix with a mask. In this way, the redundant weights can be structurally pruned and compressed. Compared with unstructured pruning methods, block diagonal sparse matrix requires no extra pointer to record the location of weights in the pruned sparse matrix, which can further enhance the compression rate under the same pruning rate of each layer. Furthermore, each block can be independently computed to improve the parallelism of the model inference.

Specifically, for the $i$-th weight matrix $W_i$, we denote $x_i$ and $y_i$ the row and column sizes of $W_i$, respectively. Given pruning rate $p_i$, we first generate a binary mask $M_i^B$ that has the same size as $W_i$. The mask $M_i^B$ has $\lfloor \frac{1}{p_i} \rfloor$ dense blocks assigned as 1 (binary True) along the diagonal. Each of the $\lfloor \frac{1}{p_i} \rfloor$ dense blocks along the diagonal axis in $M_i^B$ has the size of $(\lfloor x_i \cdot p_i \rfloor) \times (\lfloor y_i \cdot p_i \rfloor)$, and the remaining values outside the diagonal blocks are assigned as 0 (binary False). Under the same pruning rate $p_i$, each block is set to the same size instead of using blocks with irregular sizes because of the following reasons: 1) When the blocks come with different sizes, extra information is required for recording the size of each block, while same-sized blocks only need to record the size of the first block. 2) The




(a) Irregular sparse matrix (by trivial approach) (b) Diagonal blocks (step 1) (c) Minimized block-wise differences (step 2) (d) Distribution of weight indices without delta-coding (e) Distribution of weight indices with delta-coding (step 3)

**Figure 2.** Running example on fc2 (second fully-connected layer) in VGG16 on CIFAR-10 (pruning rate: 12.5%)

memory needed for model inference is related to the largest dense block since the matrix multiplication can be processed by loading the weights block-by-block. Therefore, the size of the blocks in each fully-connected layer is set to $\lfloor x_i \cdot p_i \rfloor \times \lfloor y_i \cdot p_i \rfloor$.

Next, we apply this mask and train the model from scratch. Each mask $M_i^B$ covers the weight matrix $W_i$, with an AND operation masking out the weights not residing on the diagonal block structure. Let $W_i^M$ be the weight matrix between layers $i$ and $i+1$ after the mask is applied, $W_i^M = W_i \; AND \; M_i^B$. By applying the mask $M_i^B$, we force the weights to propagate forward only if the corresponding position on the mask is assigned as 1 (binary True). This ensures that in the back-propagation stage, only the corresponding position with binary True is updated, while other weights remain zero. The idea of applying this binary mask can be regarded as a special case of the arbitrary pruning strategy, which implements the operation similar to `Diag` in Table 1. Therefore, the model accuracy is not likely to incur a sharp drop after applying this mask.

**Example 1** *Fig. 2 presents an example of pruning with the proposed block diagonal mask. After applying the generated mask on the weight matrix of size* $4096 \times 4096$*, the new matrix is shown in Fig. 2(b), which contains 8 non-overlapping blocks from top left to bottom right. We also label the block-wise differences between adjacent blocks in this figure.*

**Algorithm 1** Pruning with Block Diagonal Mask (PM)

**Input:** Fully-connected weight matrix: $W$ with size $(x \times y)$, pruning rate $p$
**Output:** Pruned fully-connected layer: $W^M$

1: Generate $M^B$ by the size of $W$, with $\lfloor \frac{1}{p} \rfloor$ dense blocks assigned as binary True along the diagonal axis having the same size $\lfloor x \cdot p \rfloor \times \lfloor y \cdot p \rfloor$
2: $W^M \leftarrow W$ AND $M^B$
3: block-number: $bn \leftarrow \lfloor \frac{1}{p} \rfloor$
4: **return** $W^M, bn$

After training each weight matrix with the mask $M_i^B$, the original matrix is transformed into a new matrix such that only independent blocks (with non-zero weights) exist on its diagonal. Given the pruning rate of the $i$-th weight matrix $p_i$, let $bn_i = \lfloor \frac{1}{p_i} \rfloor$ denote the block number of the $i$-th weight matrix, and let $\{B_1^i, B_2^i, \cdots, B_{bn_i}^i\}$ denote the blocks on the diagonal of the matrix in each fully-connected layer $i$, where $B_1^i$ is the top left block. Instead of storing the whole weight matrix, considerable memory space can be saved by storing only the blocks $\{B_1^i, B_2^i, \cdots, B_{bn_i}^i\}$ of each masked weight matrix $W_i^M$. This is because the locations of the non-zero weights are pre-defined, i.e., they are arranged into same-sized blocks on the diagonal, and thus we can easily reconstruct the original weights by aligning $\{B_1^i, B_2^i, \cdots, B_{bn_i}^i\}$ sequentially along the diagonal axis. Actually, since each $B_j^i, \forall 1 \leq j \leq bn_i$, can be computed independently, there is no need to reconstruct the original weight matrix. The pseudocode of our first step (PM) is presented in Algorithm 1.

**Step 2. Difference Minimization for Neighboring Blocks (DM).** In step 1, for each $W_i^M$, $\forall 1 \leq i \leq n$, only the blocks $\{B_1^i, B_2^i, \cdots, B_{bn_i}^i\}$ are stored. Although this approach achieves good compression rate, it still leaves a large room for further improvement. After the weights in each $B_j^i$ are quantized, we can further minimize the block-wise differences between $B_j^i$ and $B_{j+1}^i, \forall j \in [1, bn_i - 1]$. By doing so, we control the entropy of the pair-wise distances between the blocks by concentrating the distance close to zero, and then in step 3, we can employ the delta-coding techniques [8] to encode the differences to further reduce the model size.

**Example 2** *Fig. 2(b) shows the masked weight matrix and the total difference of adjacent blocks without considering to minimize the differences between adjacent blocks. In contrast, the same masked weight matrix after minimizing the differences is shown in Fig. 2(c). The differences between adjacent blocks are reduced by approximately 600 times in Fig. 2(c), as compared to Fig. 2(b). After quantization, the minimized differences between the blocks shrink the entropy from* 4.44 *to* 0.05.

Therefore, to minimize the block-wise differences between the adjacent blocks, we propose a new penalty function, namely, *Loss of Distance Penalty* $\mathcal{L}_{DP}$, as follows.

$$\mathcal{L}_{DP} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{bn_i - 1} \frac{||B_{j+1}^i - B_j^i||^2}{(bn_i - 1)},$$

where $n$ is the number of fully-connected layers, $bn_i = \lfloor \frac{1}{p_i} \rfloor$ is the number of diagonal blocks in the $i$-th masked weight matrix $W_i^M$ generated by step 1 (recall $p_i$ is the pruning rate of layer $i$), and $B_j^i$ is the $j$-th block in $W_i^M$. When the difference between $B_j^i$ and $B_{j+1}^i$ or $B_{j-1}^i$ is large, the loss increases. Based on $\mathcal{L}_{DP}$, we design the loss function $\mathcal{L}_{total}$ for training as follows.

$$\mathcal{L}_{total} = \mathcal{L}_{acc} + \alpha \mathcal{L}_{DP},$$

where $\mathcal{L}_{acc}$ is the accuracy loss[7], and $\alpha$ is a hyper-parameter designed to control the trade-off penalty between the compression rate and the model accuracy. A larger $\alpha$ value allows the normalization

[7] Loss can be cross-entropy or MSE according to the learning task.

factor to become more dominant which makes the blocks more similar and more likely to be quantized with the same code. Thus, the compression rate becomes higher with delta-coding since the entropy of the delta weight distribution is smaller, enabling Huffman coding to be more memory-efficient. In contrast, a smaller $\alpha$ allows the weights to be more likely to fit the original error loss and thus the model accuracy can be higher, since minimizing the error loss becomes a more important factor in the loss function.

In practice, when the goal is to achieve the best compression rate without accuracy loss, $\alpha$ is set as large as possible until the accuracy drops. On the other hand, in the case that the memory or bandwidth is extremely limited and slight accuracy loss can be tolerated, $\alpha$ can be set as a large value to strike a good balance between the accuracy requirement and the size of the model. As $\mathcal{L}_{total}$ considers the similarity between blocks as a chain and therefore after the model converges, reorganizing the sequence of delta-coded blocks is not necessary for the subsequent steps. The pseudocode of our second step (DM) is listed in Algorithm 2.

**Algorithm 2** Difference Minimization for Neighboring Blocks (DM)

**Input:** Masked weights for each layer: $W = \{W_1^M, \cdots, W_n^M\}$, block number for each layer: $BN = \{bn_1, \cdots, bn_n\}$
**Output:** Loss of Distance Penalty: ($\mathcal{L}_{\mathcal{DP}}$)

$i \leftarrow 1, j \leftarrow 1, \mathcal{L}_{\mathcal{DP}} \leftarrow 0$
**for** $i \leq n$ **do**
  **for** $j < bn_i$ **do**
    $\mathcal{L}_{\mathcal{DP}} \leftarrow \mathcal{L}_{\mathcal{DP}} + \frac{||B_{j+1}^i - B_j^i||^2}{b_{n_i}-1}$
    $j \leftarrow j + 1$
**return** $\mathcal{L}_{\mathcal{DP}}$

**Steps 3 and 4. Post-Quantization Cyclic-distance Assignment (PQA) and Huffman Encoding with Organized Distribution (HE).** At step 3, after the model converges, we quantize the weight matrix of each fully-connected layer $i$ according to the given parameter $q_i$ with the $k$-means clustering algorithm and train the model until it recovers the accuracy. Next, for the $bn_i$ sequence of blocks $B_j^i$ in the $i$-th fully-connected layer, we delta-code $B_2^i$ to $B_{bn_i}^i$ by the cyclic-distance of the quantized index from its previous block on the diagonal. For example, given two $(1 \times 2)$ 3-bit quantized blocks, $\mu_1$ and $\mu_2$, where $\mu_1 = [0, 1]$ and $\mu_2 = [7, 1]$. $\mu_2$ in this case is delta-coded as $[-1, 0]$ to ensure that the bit range remains the same as encoded by the original quantized bits.

Putting penalty, quantization, and cyclic-distance delta-coding together, we can adjust carefully the differences between adjacent blocks to make each block more similar to its nearby blocks and can also encode cyclic-distance between blocks after quantization, where cyclic-distance is concentrated in a small range as shown in Fig. 2(e). The concentrated distribution can be further compressed by run-length coding, e.g., Huffman coding in the next step.

**Example 3** *Continue our running example in Fig. 2, where Fig. 2(c) shows the blocks (and the total of their difference values) after step 2. After the quantization mentioned above, the distributions of the weight indices and the cyclic-distance weights indices are shown in Figs. 2(d) and 2(e), respectively.*

At step 4, the last step, we generate two codebooks using Huffman coding for each fully-connected layer. The first codebook encodes the absolute indices of the first block in each masked fully-connected layer, i.e., $B_1^i$ in $W_i^B$, and the second codebook encodes the differences of block-wise indices along the diagonal block sequence, i.e., $||B_2^i - B_1^i||, \cdots, ||B_{bn_i}^i - B_{bn_i-1}^i||$, where $bn_i$ is the number of blocks in $W_i^B$. Our algorithm then outputs the encoded $B_1^i$ block along with the $(bn_i - 1)$ delta-coded sequence for each fully-connected layer $i$ as the final compressed model.

**Example 4** *For VGG16 on CIFAR-10 in Fig. 2, if each weight is quantized to 5 bits, the weights of the fully-connected layers consume 10 MB of memory. After steps 1-3, the model is compressed to 1.25 MB. If encoded with cyclic-distance in step 3, the compressed model (after Huffman coding) takes 394 KB, while the model size (after Huffman coding) is 901 KB without cyclic-distance delta-coding.*

In our experiments, by employing Huffman coding on the cyclic-distance delta-coding, our approach outperforms the direct encoding with quantized indices by reducing up to $60\%$ of the model size. The pseudocode of step 3 is presented in Algorithm 3. Finally, combining the 4 steps altogether, the pseudocode of algorithm MESA is presented in Algorithm 4.

**Algorithm 3** Post-Quantization Cyclic-distance Assignment (PQA)

**Input:** Quantized diagonal blocks: $B^*$, $B$, maximum weight index range: $r$
**Output:** Delta-encoded form of $B^*$: $B^R$

$B^R$= size($B^*$)
**for** $i, j$ in size of $B^*$ **do**
  $B_{i,j}^R \leftarrow \min\{|B_{i,j}^* - B_{i,j}|, r - |B_{i,j}^* - B_{i,j}|\}$
**return** $B^R$

**Algorithm 4** Memory-Efficient and Structure-Aware Compression (MESA)

**Input:** $n, W = \{W_1, \cdots, W_n\}, \mathbb{P} = \{p_1, \cdots, p_n\}, \mathbb{Q} = \{q_1, \cdots, q_n\}$
**Output:** Compressed NN weights after MESA is applied: $W^C$

**for each** NN layer $W_i$ **do**
  $W_i^M, bn_i \leftarrow$ **PM**($W_i, p_i$), initialize pruned weights $W_i^M$
For each layer, $BN \leftarrow$ all $bn_i$; $W^M \leftarrow$ all $W_i^M$
**for each** training epoch **do**
  Input batch data and calculate the $MSE$
  $Loss \leftarrow MSE$; $Loss \leftarrow Loss +$ **DM**($W^M$,$BN$)
  Compute gradient, back propagation
**for each** Converged weight: $W_i^M$ **do**
  $W_i^M \leftarrow$ Quantized $W_i^M$ from 32 bits to $q_i$ bits
**for each** Quantized weight matrix: $W_i^M$ **do**
  range $\leftarrow 2_i^q$
  **for** j from 1 to $(bn_i - 1)$ **do**
    $B_{j+1}^i \leftarrow$ **PQA**($B_{j+1}^i$, $B_j^i$, range)
  Compressed weight $W_i^C \leftarrow$ Huffman code $B_1^i$ and $\{B_1^i, \cdots, B_m^i\}$ respectively from $W_i^M$
**return** Compressed NN weights $W^C$

## 5 Experimental Results

**Models and Datasets.** We compare our algorithm with other baselines on multiple NN models. Similar to most model compression works, well-known computer vision models and datasets are used, including LeNet-5 on MNIST (*LeNet-5* for short) [14], VGG16 on CIFAR-10 and CIFAR-100 (*VGG16 (C10)* and *VGG16 (C100)* for short, respectively) [20], AlexNet on CIFAR-100 (*AlexNet* for short) [12], and AlexNet on ImageNet (*AlexNet (ImageNet)* for short) [10].

In addition, to demonstrate that the proposed approach can be applied to the NN models for other tasks, we conduct experiments on predicting the category of the input text content with Char-ConvNet [28] on AG's News and DBPedia, abbreviated as *C-CNN (AG)* and *C-CNN (DB)*, respectively. Char-ConvNet comprises 6 convolutional layers and 3 FC layers. AG's News dataset from web[8] contains news

[8] http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html.

articles in 4 categories, where each category contains 30000 training samples and 1900 testing samples. Moreover, DBPedia dataset contains 14 non-overlapping ontology classes from DBPedia 2014 with each class having 40000 random training samples and 5000 testing samples.

**Baselines.** We compare the proposed MESA with five other baselines: i) DeepCompression (DeepC) [21], one of the state-of-the-art algorithms that effectively compresses the deep neural networks; ii) MPDCompress (MPDC) [22], which is able to achieve high compression rates by considering the block diagonal structure of the weights, but does not consider to minimize the block-wise differences between adjacent blocks; iii) MPDCompress with quantization and Huffman coding (M+Q+H), which quantizes the weight matrix compressed by MPDC to 5 bits, then encodes the quantized weights with the absolute quantization indices, and applies Huffman coding on the quantized indices; iv) Original fully-connected layer (FC), which is the comparison basis with no operation performed on the weights; v) Original fully-connected layer with quantization and Huffman coding (FC+Q+H), which quantizes the original weights to 5 bits and then applies Huffman coding on them without pruning any weights. For fair comparisons, we set all quantization bits $q_i$ of MESA to 5 bits according to the suggestions from DeepCompression [21], which strikes a good balance between the compression rate and accuracy[9]. Moreover, MESA and all the baseline approaches are trained to allow only a tiny accuracy drop as compared to FC, i.e., within 1%, in the experiments.

**Measures.** For fair comparisons, the pruning rates ($p$) for each model, i.e., the ratio of the number of the remaining weights to that of the original weights, are set the same as those for threshold pruning methods. More specifically, For LeNet-5 and AlexNet (ImageNet), we set the pruning rates according to those reported in DeepCompression [21]. For VGG16 (C10), VGG16 (C100), C-CNN (AG), C-CNN(DB), and AlexNet, we set the pruning rates such that DeepCompression [21] has its accuracy loss within 1% compared to the original uncompressed models (our proposed approach also has accuracy loss within 1%). Moreover, *Average bits (avg. bits)* denotes the model size (in bits) divided by the total number of unpruned weights, which shows how many bits are needed for the information of each unpruned weight (including the pointers to locate the weights if any). *Compression Rate (CP Rate)* measures the overall performance of the compression algorithms. Similar to most works in neural network compression [27, 2], our algorithm and the compared baselines only compress the weights in the fully-connected layers. Therefore, the compression rate is calculated as the memory size of the original weights in the fully-connected layers, divided by the memory size of the compressed fully-connected layers generated by the compression algorithm (including the codebook size if any). A larger compression rate indicates a better performance for the algorithm.

The experiments are conducted with Pytorch 0.4.1 framework. The models are trained on a server equipped with a 3.60 GHz Intel Core i7-7700 CPU, a GeForce RTX 2080Ti, and 64 GB RAM. The default $\alpha$ (hyper-parameter for penalty) is 0.1.

## 5.1 Compression Rates

Fig. 3 compares the compression rates of the proposed MESA with other baselines, while the corresponding accuracy of each approach is listed in Table 2. The compression rates are calculated based on the weights of the original fully-connected layer (FC) without any compression. MESA significantly outperforms the other baselines on all the models. Its compression rates range from 135× to 392× due to the different pruning rates of the models, which vary significantly according to the redundancy of each model. For example, VGG16 and AlexNet do not have any accuracy drop when keeping only 12.5% (8× compression) and 10% (10× compression) of weights, respectively. For C-CNN on both datasets, the fully-connected layers can be pruned to keep only 6.5% (16× compression) of weights while retaining the same accuracy.

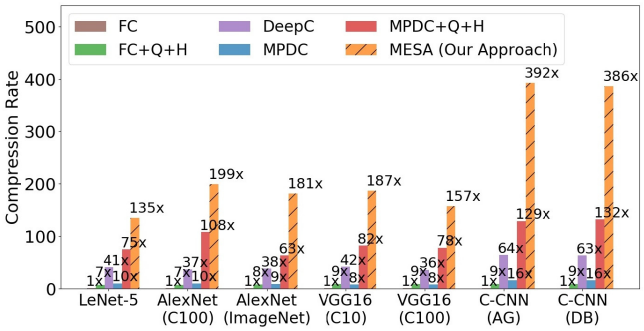



**Figure 3.** Compression rates of different approaches

We further analyze the compression rates of MESA contributed by different steps. First, after performing steps 2-4 of MESA, the delta-coding on the difference-minimized blocks along with the subsequent Huffman coding introduces the extra 2.1×, 2.3×, 3.1×, 3.65×, 3.06×, 3.8×, and 3.7× compression rates on AlexNet, AlexNet (ImageNet), LeNet-5, VGG16 (C10), VGG16 (C100), C-CNN (AG), and C-CNN (DB), respectively. Please note that Huffman coding achieves only about 1.5× compression rate in other baselines, which indicates that our idea of delta-coding on the blocks with minimized block-wise differences indeed facilitates the performance of Huffman coding.

On the other hand, MPDC, which constructs block diagonal structure by permutation, shows low (8× to 16×) compression rates because the block diagonal structure of MPDC is mainly used to improve the efficiency of matrix multiplication instead of generating blocks with minimized block-wise differences. To understand how quantization and Huffman coding can enhance MPDC, we also present the results of M+Q+H, which show a significant improvement on the compression rates (75× to 132×) compared to MPDC. However, without minimizing block-wise differences on the block diagonal structure, the compression rates are still much inferior to those of MESA.

FC+Q+H is compared to show the performance of quantization and Huffman coding, which has only 7× to 9× compression rates, where 6.4× of compression rate is contributed by quantization. Since the weight matrices are neither pruned nor rearranged to optimize the performance of Huffman coding, Huffman coding contributes a very limited compression rate, from 1.09× to 1.4×. Finally, DeepC, one of the state-of-the-art compression algorithm, achieves 36× to 64× compression rates. The pruning and its CSR (or CSC) format effectively reduce the model sizes. However, since i) CSR (or CSC) formats require additional information to locate the non-zero weights, and ii) DeepC does not jointly consider the pruning and quantization for optimizing the compression rate of Huffman coding, there is a large gap between DeepC and MESA on the compression rates.

[9] It is possible to set different quantization bits for different fully-connected layers, which can be viewed as a hyper-parameter optimization problem. We leave the discussion of this topic in the future work.

**Table 2.** Accuracy of different approaches

| | FC | FC+Q+H | DeepC | MPDC | M+Q+H | *MESA* |
|---|---|---|---|---|---|---|
| LeNet-5 | 99.34 | 99.28 | 99.29 | 99.08 | 99.03 | *99.04* |
| VGG16 (C10) | 91.32 | 91.40 | 91.44 | 91.14 | 90.53 | *91.14* |
| VGG16 (C100) | 71.40 | 72.33 | 72.57 | 71.84 | 71.91 | *71.46* |
| AlexNet | 52.68 | 52.77 | 54.04 | 53.69 | 53.83 | *53.47* |
| AlexNet (ImageNet) | 77.12 | 76.51 | 77.58 | 76.38 | 76.27 | *76.55* |
| C-CNN (AG) | 86.61 | 86.68 | 86.76 | 86.51 | 87.22 | *87.21* |
| C-CNN (DB) | 97.84 | 97.90 | 97.94 | 97.92 | 98.05 | *97.92* |

## 5.2 Layer-wise Statistics of Different Models

**Table 3.** Layer-wise average bits in LeNet-5

| | Size | p | DeepC Avg. bits | M+Q+H Avg. bits | MESA Avg. bits | MESA CP Rate |
|---|---|---|---|---|---|---|
| fc1 | 1.52MB | 10% | 7.6 | 4.2 | **2.3** | 138× |
| fc2 | 19.53KB | 10% | 14.2 | **6** | 6.7 | 47× |
| total | 1.54MB | 10% | 7.7 | 3.1 | **2.3** | 135× |

**LeNet-5 on MNIST.** In the following, *Size* represents the original size of the 32-bit float representation of the weights in fully-connected layers. For LeNet-5, as shown in Table 2, MESA achieves the accuracy of 99.04% with a slight accuracy drop compared to the original FC with 99.34% accuracy. Table 3 presents the average bits for each approach. MESA outperforms the other baselines in average bits. In fc1 (the first fully-connected layer), DeepC requires 7.6 average bits because additional bits are required to locate the non-zero weights in the sparse matrix. While M+Q+H lowers the average bits to 4.2 bits, it still consumes nearly twice memory compared to MESA (2.3 bits).

One may observe that the average bits for fc2 (the second fully-connected layer) are high for all the algorithms. This is a special case where the fully-connected layer after pruning becomes too small, and the codebook overhead dominates the representation. For example, in fc2, the codebook consumes 103 Bytes, but only 29 Bytes are needed to represent the weights by MESA. Even so, MESA still achieves a 47× compression rate in fc2.

**Table 4.** Layer-wise average bits in C-CNN (AG)

| | Size | p | DeepC Avg. bits | M+Q+H Avg. bits | MESA Avg. bits | MESA CP Rate |
|---|---|---|---|---|---|---|
| fc1 | 34MB | 6.25% | 7.9 | 3.9 | **1.3** | 401× |
| fc2 | 4MB | 6.25% | 8.9 | 4.1 | **1.5** | 341× |
| fc3 | 16KB | 25% | 9.8 | 5.3 | **4.2** | 31× |
| total | 38.01MB | 6.28% | 8 | 3.95 | **1.3** | 392× |

**C-CNN on AG's news.** Table 4 compares the results of MESA and other baselines for text classification tasks. We observe that the layer-wise pruning rates for the first two fully-connected layers, i.e., $p_1$, $p_2$, for C-CNN can be set to 6.25% without an accuracy drop. This indicates that the weights for fc1 and fc2 in C-CNN have a much higher redundancy on AG's News compared to other models. One major advantage of MESA is that even the baseline compression methods

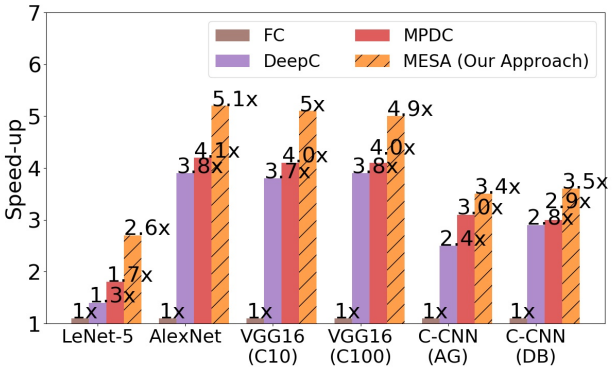



**Figure 4.** Inference speed-up of different approaches

reach their pruning and quantization limits, weight distribution adjustment (in step 2 of the algorithm) by MESA still provides extra compression. The block-wise delta coding between adjacent blocks provides over 3× memory size reduction compared to M+Q+H. All together, C-CNN compressed by MESA requires only around 1.3 bits for each non-zero weight, and the compression rate can be as high as 392× for AG's news.

**Table 5.** Layer-wise average bits in C-CNN (DB)

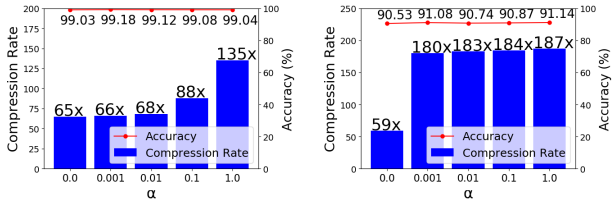
| | Size | p | DeepC Avg. bits | M+Q+H Avg. bits | MESA Avg. bits | MESA CP Rate |
|---|---|---|---|---|---|---|
| fc1 | 34MB | 6.25% | 8.0 | 3.9 | **1.3** | 396× |
| fc2 | 4MB | 6.25% | 9.2 | 4.0 | **1.4** | 361× |
| fc3 | 56KB | 50% | 4.7 | **1.8** | 2.3 | 28× |
| total | 38.05MB | 6.38% | 8.1 | 3.8 | **1.3** | 386× |

**C-CNN on DBPedia.** For C-CNN on DBPedia, the average bits for non-zero weights in fc3 of MESA is slightly larger than M+Q+H. This is because when our Loss of Distance Penalty ($\mathcal{L}_{DP}$) minimizes the distances, it may sometimes sacrifice the compression rate of the much-smaller fully-connected layers in order to reduce the accuracy loss. However, this does not affect the performance of the overall compression rate for MESA. The overall average bits for model compressed by MESA is still around 3× smaller than M+Q+H and 6.2× smaller than DeepC. For the layer-wise average bits of VGG16 and AlexNet, they show similar trends to the models described above where MESA outperforms all other baselines significantly. We omit the results due to their similarity trends.

In summary, the results on NLP datasets (C-CNN on AG's News and DBPedia) are better than those on image datasets (CIFAR-10 and CIFAR-100) since visual features are more complicated than sentence embeddings, and it is thus more effective to diagonalize the weight matrix of the NN for extracting features from NLP datasets.

## 5.3 Inference Speed-up on GPU

As mentioned earlier, the block diagonal structure can also improve the efficiency of model inference. To validate this claim, we present




(a) LeNet-5 (b) VGG16 (C10)

**Figure 5.** Penalty evaluation

the inference time of the models compressed by MESA and other baseline approaches. Here, *speed-up* represents the ratio of the inference time of the original uncompressed model (FC) to the inference time of the model compressed by a certain approach. A larger speed-up value indicates a smaller inference time compared to the uncompressed model, i.e., FC. The inference time of each model is calculated on a single GeForce RTX 2080Ti with CUDA 10.0. We employ cuBLAS GEMV kernel on the dense layers in FC and each dense blocks in MPDC and MESA. For DeepC, each sparse layer is stored in CSR format and cuSPARSE CSRMV kernel is used for general unstructured matrix operations in CSR format[10].

Fig. 4 shows that the speed-up of MESA outperforms the other baselines, i.e., the average speed-up of MESA, DeepC, and MPDC are $4.08\times$, $2.97\times$, and $3.28\times$, respectively. The speed-up of MESA is around $26\%$ higher than MPDC because the pruned fully-connected layers of MESA are originally trained with block diagonal structure and do not require inverse matrix permutations to reconstruct the output, which allows a better scheduling and memory organization. Yet, MPDC still shows $10\%$ better speed-up results compared to DeepC because the latency of inverse permutations is minor compared to the reduced time of model inference with block diagonal structure.

## 5.4 Penalty Evaluation

To understand the impact of $\alpha$, i.e., the hyper-parameter controlling the trade-off between the accuracy and compression rate, we present the compression rates of MESA with different $\alpha$ values on LeNet-5 and VGG16 (C10). Fig. 5 shows that the compression rates increase significantly without obvious accuracy drops as $\alpha$ increases. With a larger $\alpha$, the weights in adjacent blocks are easily to be quantized into the same index, and the weight distribution after delta-coding can thus become more skew with a lower entropy. Table 6 shows the entropy of the delta-coded weight distribution for LeNet-5 and VGG16 (C10). For LeNet-5, when $\alpha$ equals to 1, the entropy values of fc1 and fc2 decrease from 4.86 and 4.97 to 2 and 3.42, respectively (LeNet-5 has only two fc layers). Please note that $\alpha$ controls the trade-off and cannot be set too large. For example, setting $\alpha = 10$ for LeNet-5 and VGG16 (C10) results in an accuracy drop of more than $1\%$. Fig. 5(b) shows that in VGG16 on CIFAR-10, MESA works well with a small penalty, i.e., $\alpha = 0.001$, while the model accuracy retains (original model accuracy: $91.32\%$). When $\alpha$ becomes larger, the compression rate of MESA, although outperforming other baselines as shown in Fig. 3, does not improve significantly. After a careful examination, we find that $98\%$ of the delta-coded weight indices are 0 when $\alpha = 0.001$, resulting in a very low entropy value for its distribution, as shown in Table 6. Therefore, a larger $\alpha$ value does not significantly improve the compression rate.

[10] Please note that CUDA libraries do not support the indirect lookup for the quantized models. Therefore, we present the comparison results on the compressed model without quantization.

**Table 6.** Layer-wise data entropy on LeNet-5 and VGG16

| | LeNet-5 | | | | VGG16 (C10) | | | |
|---|---|---|---|---|---|---|---|---|
| | Orig. index | Delta-coded index | | | Orig. index | Delta-coded index | | |
| | | α=0.01 | α=0.1 | α=1.0 | | α=0.001 | α=0.01 | α=1.0 |
| fc1 | 4.86 | 4.58 | 3.39 | **2.00** | 2.94 | 0.2 | **0.07** | 0.1 |
| fc2 | 4.97 | 4.91 | 4.85 | **3.42** | 2.81 | 0.08 | **0.03** | 0.08 |
| fc3 | N/A | N/A | N/A | N/A | 3.35 | 2.22 | 0.95 | **0.21** |

## 6 Conclusion and Future Work

In this paper, we propose the *COMIS* problem to consider the block-wise differences in block diagonal structure for NN model compression. We devise algorithm *MESA*, which effectively prunes the weights into a block diagonal structure and minimizes the block-wise differences to boost the compression rate. Experiments show that MESA achieves up to $392\times$ compression rates, tripling those of the state-of-the-art approaches, while significantly improving the inference efficiency. In the future, we plan to explore the ideas of i) fully leveraging the block-diagonal structure to reduce the memory access on ASIC, ii) learning to adaptively adjust the pruning rates/quantization bits for each layer of different models, and iii) proposing/experimenting different penalty functions for different models/datasets.

## ACKNOWLEDGEMENTS

This work was supported in part by Ministry of Science and Technology (MOST), Taiwan, under MOST 109-2636-E-007-019-, MOST 108-2636-E-007-009-, MOST 108-2218-E-468-002-, MOST 107-2218-E-002-010-, MOST 108-2218-E-009-050-, and MOST 108-2218-E-009-056-.

## REFERENCES

[1] Yash Akhauri, 'Hadanets: Flexible quantization strategies for neural networks', *arXiv preprint arXiv:1905.10759*, (2019).
[2] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee, 'Universal deep neural network compression', in *arXiv preprint arXiv:1802.02271*, (2018).
[3] Jonathan Frankle and Michael Carbin, 'The lottery ticket hypothesis: Finding sparse, trainable neural networks', in *International Conference on Machine Learning*, (2019).
[4] Ido Freeman, Lutz Roese-Koerner, and Anton Kummert, 'Effnet: An efficient structure for convolutional neural networks', in *IEEE International Conference on Image Processing*, (2018).
[5] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan, 'Deep learning with limited numerical precision', in *International Conference on Machine Learning*, (2015).
[6] Song Han, Jeff Pool, John Tran, and William Dally, 'Learning both weights and connections for efficient neural network', in *Advances in Neural Information Processing Systems*, (2015).
[7] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, 'Mobilenets: Efficient convolutional neural networks for mobile vision applications', *arXiv preprint arXiv:1704.04861*, (2017).
[8] James J Hunt, Kiem-Phong Vo, and Walter F Tichy, 'Delta algorithms: An empirical analysis', *ACM Transactions on Software Engineering and Methodology*, (1998).
[9] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer, 'Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size', in *arXiv preprint arXiv:1602.07360*, (2016).

[10] S. Satheesh H. Su A. Khosla J. Deng, A. Berg and L. Fei-Fei, 'Ilsvrc-2012', in *http://www.image-net.org/challenges/LSVRC/2012/*, (2019).
[11] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin, 'Compression of deep convolutional neural networks for fast and low power mobile applications', in *International Conference on Learning Representations*, (2016).
[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, 'Imagenet classification with deep convolutional neural networks', in *Advances in Neural Information Processing Systems*, (2012).
[13] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar, 'Squeezing deep learning into mobile and embedded devices', *IEEE Pervasive Computing*, (2017).
[14] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger, et al., 'Comparison of learning algorithms for handwritten digit recognition', in *International Conference on Artificial Neural Networks*, (1995).
[15] Dongsoo Lee, Se Jung Kwon, Byeongwook Kim, and Gu-Yeon Wei, 'Learning low-rank approximation for cnns', *arXiv preprint arXiv:1905.10145*, (2019).
[16] Zhisheng Li, Lei Wang, Shasha Guo, Yu Deng, Qiang Dou, Haifang Zhou, and Wenyuan Lu, 'Laius: An 8-bit fixed-point cnn hardware inference engine', in *IEEE International Symposium on Parallel and Distributed Processing with Applications and IEEE International Conference on Ubiquitous Computing and Communications*, (2017).
[17] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell, 'Rethinking the value of network pruning', in *International Conference on Learning Representations*, (2019).
[18] Antonio Polino, Razvan Pascanu, and Dan Alistarh, 'Model compression via distillation and quantization', in *International Conference on Learning Representations*, (2018).
[19] Günther Schindler, Wolfgang Roth, Franz Pernkopf, and Holger Fröning, 'N-ary quantization for cnn model compression and inference acceleration', in *International Conference on Learning Representations*, (2019).
[20] Karen Simonyan and Andrew Zisserman, 'Very deep convolutional networks for large-scale image recognition', in *International Conference on Learning Representations*, (2015).
[21] Han Song, Huizi Mao, and William J Dally, 'Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding', in *International Conference on Learning Representations*, (2016).
[22] Lazar Supic, Rawan Naous, Ranko Sredojevic, Aleksandra Faust, and Vladimir Stojanovic, 'Mpdcompress-matrix permutation decomposition algorithm for deep neural network compression', in *arXiv preprint arXiv:1805.12085*, (2018).
[23] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers, 'Finn: A framework for fast, scalable binarized neural network inference', in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, (2017).
[24] Qing Yang, Wei Wen, Zuoguan Wang, Yiran Chen, and Hai Li, 'Integral pruning on activations and weights for efficient neural networks', in *International Conference on Learning Representations*, (2019).
[25] Shaokai Ye, Tianyun Zhang, Kaiqi Zhang, Jiayu Li, Kaidi Xu, Yunfei Yang, Fuxun Yu, Jian Tang, Makan Fardad, Sijia Liu, et al., 'Progressive weight pruning of deep neural networks using admm', in *International Conference on Learning Representations*, (2019).
[26] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis, 'Nisp: Pruning networks using neuron importance score propagation', in *IEEE Conference on Computer Vision and Pattern Recognition*, (2018).
[27] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao, 'On compressing deep models by low rank and sparse decomposition', in *IEEE Conference on Computer Vision and Pattern Recognition*, (2017).
[28] Xiang Zhang, Junbo Zhao, and Yann LeCun, 'Character-level convolutional networks for text classification', in *Advances in neural information processing systems*, pp. 649–657, (2015).