Merging Deep Neural Networks for Mobile Devices

Yi-Min Chou[1,2], Yi-Ming Chan[1,2], Jia-Hong Lee[1,2], Chih-Yi Chiu[3], and Chu-Song Chen[1,2]

[1]Institute of Information Science, Academia Sinica, Taipei, Taiwan,
Email: {chou, yiming, honghenry.lee, song}@iis.sinica.edu.tw
[2]MOST Joint Research Center for AI Technology and All Vista Healthcare
[3]National Chiayi University, No.300 Syuefu Rd., Chiayi City, Taiwan,
Email: chihyi.chiu@gmail.com

Abstract

In this paper, a novel method to merge convolutional neural networks for the inference stage is introduced. When two feed-forward networks already trained for handling different tasks are given, our method can align the layers of these networks and merge them into a unified model by sharing the representative weights. The performance of the merged model can be restored or improved via re-training. Without needing high-performance hardware, the proposed method effectively produces a compact model to run the original tasks simultaneously on resource-limited devices. The system development time, as well as training overhead, is substantially reduced because our method leverages the co-used weights and preserves the general architectures of the well-trained networks. The merged model is jointly compressed and can be implemented faster than the original models with a comparable accuracy. When combining VGG-Avg and ZF-Net models, our approach can achieve higher than 12 and 2.5 times of compression and speedup ratios compared to the original whole models, respectively, while the accuracy remains approximately the same.

1. Introduction

Deep neural networks are successfully applied to a wide range of applications, including computer vision, medical imaging, and multimedia processing. We usually design different network models and train them with particular datasets separately to handle various tasks, and thus they can behave well for specific purposes. In practical artificial intelligence (AI) applications, however, it is common to handle multiple tasks simultaneously, resulting in a high demand for the computation resource in both training and inference stages. Consequently, how to effectively integrate multiple network models in a system is a fundamental problem towards thriving AI applications.

This paper undertakes the issues of merging multiple existing feed-forward networks into a unified but compact one. The original networks, whose architectures may not be identical, can be of either single or multiple input sources. After unification, the merged network should be capable of managing the original tasks but is more condensed than the whole original models.

Topologies of existing deep neural network models may be very different. For example, a feed-forward network contains only the layers in a cascade, whereas a recurrent neural network (RNN) has loops among the layers. Currently, this study focuses on merging feed-forward networks, while merging networks with loops remains a future work. A modern feed-forward network consists of several kinds of layers, including convolution, pooling, and full-connection, referred to as a convolutional network (CNN) in general. When merging two CNNs, our approach aligns the same-type layers (convolution; full-connection) into pairs. The layers in a pair are merged into a single layer that shares a common weight codebook through an encoding scheme. The merged single model can be further trained via back-propagation algorithm; it thus can be fine-tuned to seek for performance improvement.

We introduce a method, NeuralMerger, to merge neural-nets for inference. It consists of two stages.

Alignment and encoding phase: First, we align the architectures of neural network models and encode the weights such that they are shared among the networks. The common parts of the network weights are found so that the filters and weights of different neural networks are able to be co-used.

Fine-tuning phase: Second, we fine-tune the merged model with few or all labeled data. Following the concept of distilling dark knowledge of neural networks in [8], our

method employs the original models outputs to guide the training of the merged model in this phase.

1.1. Motivation of Our Study

Merging existing neural networks has various applications to real-world problems. For instance, in intelligent agents or robots, numerous recognition tasks based on same or different signal sources (eg., image, sound) are often required. Even when using a single source only (eg. image), visual classification tasks of different types such as object recognition, face identification, hand gesture prediction and scene-text classification could also be involved in an intelligent system. When many models of individual functionalities are available, merging them on the inference stage is helpful to build a system toward strong AI.

To tackle multiple recognition tasks in a single system based on either unique or various signal sources, a typical approach is to design a new model and train the model on the union datasets of these tasks, eg., [12, 3]. Such "learn-them-all" approaches train a single complex model to handle multiple tasks simultaneously. Nevertheless, two issues may arise. First, it is hard to choose a suitable architecture for learning all the tasks well in advance; hence, a trial-and-error process is required to conduct suitable neural-net architectures. Second, learning from a random initial with a large set of training data of different types or sources could be demanding. To tackle these issues, the network models with bridging layers among them is combined to conduct the architecture in [19]. Notwithstanding, increased complexity and size of the joint model hinder their availability on edge devices.

As many models trained for various tasks are available nowadays, a practical way to integrate different functionalities in a system would be leveraging on these individual-task models. In this paper, we introduce an approach that merges different neural networks by removing the co-redundancy among their filters or weights. The proposed NeuralMerger can take advantage of existing well trained models. Our approach merges them via finding and sharing the representative codes of weights; the shared codes can still be refined by learning. To our knowledge, this is the first study on merging known-weights neural networks into a more compact model. Because our approach compresses the networks for weight sharing and redundancy removal, it is useful for a deep-learning embedded system or edge computing for the inference stage.

1.2. Overview of Our Approach

When merging two different feed-forward CNN models C_A and C_B, the output is a CNN model consisting of jointly encoded convolution (**E**-Conv) and fully-connected (**E**-FC) layers. An overview of our approach is illustrated in Fig. 1. An example of merging two models via our approach is given in Fig. 2.

Contributions of this paper are summarized as follows:

- Given well-trained CNN models, the introduced NeuralMerger can merge them for multi-tasks. The merging process preserves the general architectures of the well-trained networks and removes their redundancy. It avoids the cumbersome re-design and trial-and-error process raised by the learn-them-all approaches.
- The proposed method produces a more compact model to handle the original tasks simultaneously. The compact model consumes less computational time and storage than the compound model of the original networks. It has a great potential to be fitted in low-end systems.
- The experiments evaluate diferent CNN models and datasets, and the result demonstrates a satisfactory performance. In overall, the proposed method can achieve higher than 2.5x speedup and 12x compression ratios, while the accuracy remains nearly the same when merging the VGG-Avg and ZF-Net models.

2. Related Work

In this section, we briefly review multi-task deep learning, network compression, and model calibration.

Multi-task Deep Models

To simultaneously achieve various tasks via a single neural-net model, a typical way is to increase the output nodes (for multi-tasks) of a pre-chosen neural-net structure, and train it from an initialization. In [14], a joint facial age estimation and expression classification model is proposed. Learn-them-all approaches have been proposed to solve multi-tasks across various domains. In [12], MultiModel architecture is introduced to allow input data to be images, sound waves, and text of different dimensions, and then converts them into a unified representation. The convolution, attention, and sparsely-gated mixture-of-experts layers are incorporated to get good performance on various problems. In [3], a deep CNN leverages massive synchronized data (sound and sentences paired with images) to learn an aligned representation. The aligned representation can be shared across modalities for multi-modal tasks such as cross-modal retrieval and classification. Nevertheless, as mentioned earlier, applying the learn-them-all approaches has to pay cumbersome training effort and intensive inference computation.

Neural-Net Compression

Compressing a neural-net is an active direction to deploy the compact model on resource-limited embedded systems. To reduce the representation, binary weights and bitwise operations are used in [9] and [18]. Han et al. [6] introduce a three-stage pipeline: pruning redundant network connections, quantizing weights with a codebook, and

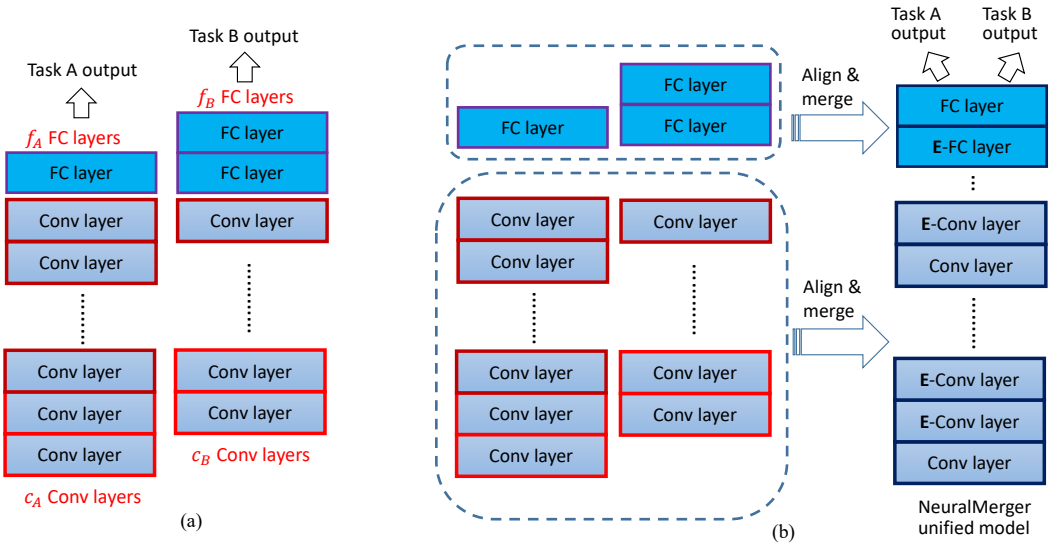

Figure 1. (a) Two tasks accomplished by feed-forward networks, where model A (or B) consists of c_A (or c_B) convolution and f_A (or f_B) fully-connected layers, respectively. (b) Our NeuralMerger unifies the two models into a single model composed by $max(c_A, c_B)$ convolution and $max(f_A, f_B)$ fully-connected layers for the model inference; **E**-Conv and **E**-FC are referred to as the *Jointly-**E**ncoded convolution* and *fully-connected* layers, respectively.

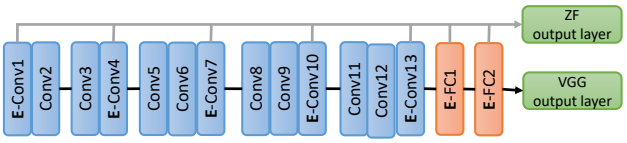

Figure 2. Example of merging tw models, ZF and VGG-Avg into a single one for the inference stage via our approach.

Huffman encoding weights and index, to reduce the storage required by CNN. However, it induces irregular sparsity in the pruned networks and requires special libraries or hardware for speedup. Quantized CNN (Q-CNN) [22] is proposed to address both the speed and compression issues, which splits the input layer space and applies vector quantization to each subspace. For each subspace, the inner products between the input and codewords can be precomputed and stored in a lookup table; it effectively reduces the computation and storage for the layer. Researchers also try to prune filters and feature maps to directly reduce the computational cost [17][15][7]. In [17], it models the pruning problem as a filter combinatorial problem and solves through a greedy approach that removes the least important neuron from the network. Similarly, [15] prunes the filters with small L1-norm values.

Network Distilling and Small-model Retraining

Transferring the learned knowledge from a large network to a small network is another important direction towards effective network compression. In [8], distilling the knowledge of an ensemble model into a smaller model is introduced, where 3% data is enough to train a small model in some cases. The class probabilities of the ensemble model are distilled with a high temperature of the final softmax as soft targets for training. Since the soft targets provide more information and less gradient variance, the small model can be trained with less data. In [7], only 1/10 iterations in fine-tune stage is enough to recover the performance of pruned models.

Instead of compressing a single network, the goal of this study is to merge multiple networks simultaneously. Besides, our method can restore the performance of the jointly compressed models by fine-tuning it with training samples via the giudances of the outputs of every layer of the original models.

3. Deep Model Integration

Assume that model A (or B) consists of c_A (or c_B) convolution (Conv) followed by f_A (or f_B) fully connected (FC) layers, with their weights already trained. Let $c_{min} = min(c_A, c_B)$. In our approach, a correspondence $(c_{A(i)}, c_{B(i)})$ is established between the Conv layers for the alignment of the two models, $i \in \{1 : c_{min}\}$; $A(\cdot)$ is a strictly increasing mapping from $\{1 : c_{min}\}$ to $\{1 : c_A\}$, and $B(\cdot)$ is a strictly increasing mappings from $\{1 : c_{min}\}$ to $\{1 : c_B\}$. Likewise, a correspondence $(f_{A(i)}, f_{B(i)})$ is also established between the FC layers for $i \in \{1 : f_{min}\}$.

In our method, the merged layers have to be of the same type (Conv or FC). Given two layers, one in model A and the other in model B, the principle of merging them is to find a set of (fewer) exemplar codewords that represent the weights of the layers with small quantization errors. The

layers are thus jointly compressed for redundancy removal. Below, we first consider unifying the Conv layers, and then the FC layers.

3.1. Merging Convolution Layers

Assume that some Conv layer in model A and some other Conv layer in model B are to be merged. The layer of model A has the input volume size $N_A \times M_A \times d_A$, where $N_A \times M_A$ is the spatial size and d_A is the depth (number of channels). The input volume is convolved with p_A convolution kernels, where the size of each kernel is $n_A \times m_A \times d_A$.

Without loss of generality, assume that padded (with zero) convolutions are used. The output of the Conv layer in model A is thus a volume of $N_A \times M_A \times p_A$. Likewise, similar notations apply to the respective layer in model B. An input volume of the size $N_B \times M_B \times d_B$ are convolved by p_B convolution kernels of the size $n_B \times m_B \times d_B$. The output volume of that layer is of the size $N_B \times M_B \times p_B$.

We aim to jointly encode the convolution coefficients. As there are p_A (or p_B) convolution kernels in the layers of A (or B), we hope to find a new set of fewer (than $p_A + p_B$) exemplars to express the original ones so that the models are fused and the redundancy between them is removed. To this end, a viable way is to perform vector quantization (such as k-means clustering) on the convolution kernels, and find a smaller number of codewords ($p < p_A + p_B$) to jointly represent the kernels compactly. However, it is demanding to make this method practicable because the kernel dimensions could be inconsistent (i.e., $n_A \neq m_A$ or $n_B \neq m_B$).

To address this issue, we unify the different convolution kernels by using spatially 1×1 convolutions, so that merging CNNs with different convolution kernel sizes is achievable. In the following, we review the operations in a Conv layer at first, and then show how to separate the dimensions so that different layers are unified and jointly encoded.

3.1.1 Operations in Convolution Layer

The operations in a Conv layer of CNNs are reviewed as follows. Suppose $x \in R^{N \times M \times d}$ is the input volume (a.k.a. 3D tensor) to a Conv layer and $y \in R^{N \times M \times p}$ is the output volume. Assume that p convolution kernels of size $n \times m \times d$ are applied to the layer, denoted as

$$\{g^{(t)} \in R^{n \times m \times d} | t = 1 \cdots p\}. \tag{1}$$

Then, the t-th channel output is obtained as $y_t = x \star g^{(t)}$, the volume convolution of x and $g^{(t)}$, and the output y is the concatenation of y_t,

$$y = [y_1\ y_2\ \cdots\ y_p]. \tag{2}$$

Let $x_u \in R^{N \times M}$ and $g_u^{(t)} \in R^{n \times m}$ repectively be the u-th channel of x and $g^{(t)}$. The volume convolution is formed by summing the 2D-convolution results of the d channels:

$$x \star g^{(t)} = \sum_{u=1}^{d} x_u * g_u^{(t)}, \tag{3}$$

where $*$ denotes the 2D convolution operator.

In CNNs, various n and m (eg., $n = m = 3, 5, 7 \cdots$) are used in existing networks. Particularly, when $n = m = 1$, the volume convolution of size $1 \times 1 \times d$ is often referred to as a 1×1 convolution in CNNs for all d.

3.1.2 Kernel Decomposition in Spatial Directions

In the above, the volume convolution is computed as a spatially sliding operation (2D convolution) followed by a channel-wise summation along the depth direction. In this section, we show that, no matter what n and m are, it can be equivalently represented by 1×1 convolutions via decomposing the kernel along the spatial directions as follows.

Given the kernel $g^{(t)}$, let $g^{(t)}_{[i_0,j_0],u}$ specify its entry at the spatial location (i_0, j_0) of the u-th channel. In particular, $g^{(t)}_{[i_0,j_0]} \in R^d$ stands for the $1 \times 1 \times d$ volume convolution associated with the (i_0, j_0)-th spatial site of the kernel $g^{(t)}$; e.g., a kernel of spatial size 5×5 consists of 25 kernels of spatial size 1×1, $(i_0, j_0) \in \{1:5\} \times \{1:5\}, \forall d$.

Following the notation, we decompose an $n \times m \times d$ volume convolution into multiple $1 \times 1 \times d$ convolutions and combine them with shift operators: Without loss of generality, we assume that the spatial sizes n, m of the kernel are odd numbers and replace them with $w = (n-1)/2$ and $h = (m-1)/2$. The t-th channel output y_t can be equivalently represented as

$$y_t = \sum_{i_0=-w}^{w} \sum_{j_0=-h}^{h} S_{-i_0,-j_0}[x \star g^{(t)}_{[i_0,j_0]}], \tag{4}$$

where $g^{(t)}_{[i_0,j_0]}$ $(-w \leq i_0 \leq w, -h \leq j_0 \leq h)$ are the $1 \times 1 \times d$ convolutions depicted above; in Eq.(4), S is the shift operator satisfying that

$$S_{i_0,j_0}[x](i, j, u) = x(i - i_0, j - j_0, u), \forall i, j, u, \tag{5}$$

with i, j the spatial location and u $(1 \leq u \leq d)$ the channel index. Hence, for all n, m, the t-th channel output of the volume convolution can be decomposed as the shifted sum of nm 1×1 convolutions via Eq. (4). Then, the output y is obtained via the concatenation in Eq. (2).

To address the issue caused by dimension mismatch in merging two convolution layers, we then propose to take the representation of 1×1 convolutions for both layers. Hence, a kernel in model A is decomposed into $C_A = n_A m_A$ convolutions of size $1 \times 1 \times d_A$ and that in model B is decomposed into $C_B = n_B m_B$ convolutions of size $1 \times 1 \times d_B$. The kernel is then unified into 1×1 in the spatial domain no matter whether n_A (or m_A) equals to n_B (or m_B).

3.1.3 Kernel Separation along Depth Direction

Then, we seek to jointly express the C_{AB} 1×1 convolutions by a compact representation so that the two layers are co-compressed, where $C_{AB} = p_A C_A + p_B C_B$ and p_A, p_B are the numbers of kernels of the layers in A and B, respectively. Though the subspace dimensions in the spatial domain are consistent (1×1) now, they are still inconsistent in the depth direction (d_A vs. d_B) and thus crucial to be jointly clustered. To address this problem, we simply separate the $1 \times 1 \times d$ kernel $g^{(t)}_{[i_0,j_0]}$ into non-overlapping $1 \times 1 \times r$ kernels along the depth direction ($r < d$). As the convolution in CNNs are summation-based in the depth direction, we divide the kernel $g^{(t)}_{[i_0,j_0]} \in R^d$ into $\lceil d/r \rceil$ vectors of dimension r,

$$g^{(t)}_{[i_0,j_0];\langle v \rangle} \in R^r, v = 1, \cdots, \lceil d/r \rceil, \tag{6}$$

where $g^{(t)}_{[i_0,j_0];\langle v \rangle}$ (of size $1 \times 1 \times r$) is the v-th segment of the original kernel. The output y_t in Eq. (4) then becomes

$$y_t = \sum_{v=1}^{\lceil d/r \rceil} \sum_{i_0=-w}^{w} \sum_{j_0=-h}^{h} S_{-i_0,-j_0}[x_{\langle v \rangle} \star g^{(t)}_{[i_0,j_0];\langle v \rangle}], \tag{7}$$

where $x_{\langle v \rangle} \in R^{N \times M \times r}$ is the v-th sub-volume of the input x for $d = d_A$ or d_B. Specifically, a spatially 1×1 kernel is respectively segmented into $\rho_A = \lceil d_A/r \rceil$ (or $\rho_B = \lceil d_B/r \rceil$) kernels of dimension r in model A (or B), where the last segment is padded with zero if necessary.

Let $\rho = min(\rho_A, \rho_B)$. There are then C_{AB} kernels of size $1 \times 1 \times r$ for the segment $1 \le v \le \rho$. To jointly represent the kernels of both layers, we use C codewords ($C < C_{AB}$) in the dim-r space to encode the convolution coefficients compactly. We run the k-means algorithm with various initials for the C_{AB} vectors and then select the results yielding the least representation error to produce the C codewords (i.e., cluster centers of k-means), for $v \in \{1, \cdots, \rho\}$. [1]

3.1.4 E-Conv Layer and Weights Co-use

The merged convolution layer (called the **E**-Conv layer), is a newly-formed layer where the weights are co-used among the convolution kernels: Denote the C codewords in the v-th subspace to be $\{b_{c,v} \in R^r | c = 1 : C\}$. We then replace each dim-r kernel at the spatial site (i_0, j_0) in the subspace v (namely, $g^{(t)}_{[i_0,j_0];\langle v \rangle} \in R^r$) with $b_{\pi(i_0,j_0,v,t);v}$, the closest codeword in the dim-r space, where $\pi(i_0, j_0, v, t) \in \{1 : C\}$ is the code-assignment mapping. Eq. (7) is then simplified as

$$y_t = \sum_{v=1}^{\lceil d/r \rceil} \sum_{i_0=-w}^{w} \sum_{j_0=-h}^{h} S_{-i_0,-j_0}[x_{\langle v \rangle} \star b_{\pi(i_0,j_0,v,t);v}]. \tag{8}$$

Because the number of codewords C is fewer than that of the total kernel vectors C_{AB}, Eq. (8) can be executed more efficiently via computing the 1×1 convolutions of the C codewords at first:

$$x_{\langle v \rangle} \star b_{c,v} \quad (\forall c = 1 \cdots C, v = 1 \cdots \rho), \tag{9}$$

and then storing the results in a lookup table. The run-time operation of 1×1 convolution is thus replaced by table indexing. Hence, the convolution kernels of the two models A and B are representationally shared in a compact codebook, $\{b_{c,v} | v = 1 : C\}$, and the computation time is saved. An illustration of the **E**-Conv layer is given in Fig. 3.

When choosing the coderwords C fewer, the amount of convolution coefficients is reduced to C/C_{AB}. The merged model is thus co-compressed as it consumes less storage than the total required for the two convolution layers. As for the computational speed, each $x_{i,j;\langle v \rangle} \in R^r$ is replaced with an index and there are NMd/r entries for indexing in $x_{\langle v \rangle}$, where i, j are the spatial location. Let τ_x and be the time unit for a table-indexing operation and τ_r be the time unit for a $1 \times 1 \times r$ convolution. The speedup ratio is then $(C\tau_r + NMd\tau_x/r)/(C_{AB}\tau_r)$ in terms of the complexity. Hence, when the codewords C is fewer or the subspace dimension r is larger, the speedup is getting higher.

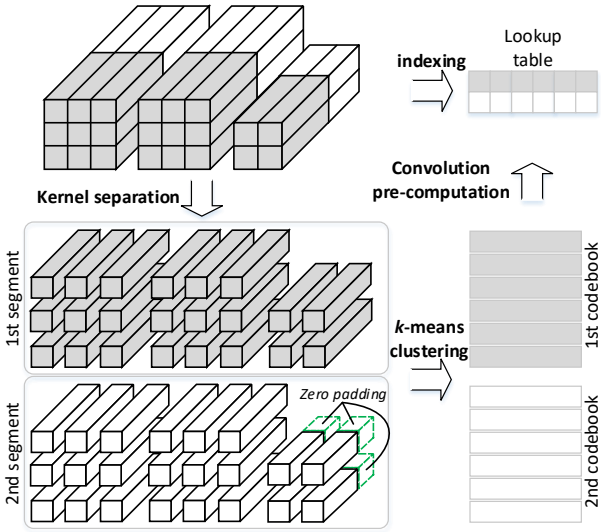

Figure 3. Illustration of merging two models' Conv layers having the kernels of spatial size 3×3 and 2×2, respectively; each layer is divided into 2 segments. They are decomposed into spatially 1×1 kernels and the kernels in every segment is clustered via k-means clustering to build a coodbook. The convolutions are precomputed on the codebook at runtime, and a lookup table is built for indexing the results.

[1] For those remaining segments, $\rho + 1 \le v \le max(\rho_A, \rho_B)$, we also use C codewords to encode the $p_A C_A$ (or $p_B C_B$) dim-r vectors in the respective subspaces if $d_A > d_B$ (or $d_A < d_B$).

3.1.5 Derivatives of the Merged Layer

Besides condensing and unifying the convolution operations, the **E**-Conv layer is also differentiable and thus end-to-end back-propagation learning still remains realizable. However, evaluating the derivatives would be hard based on the table-lookup structure as the indices are uncontinuous. Hence, the table is used only for the inference stage in our approach. While for learning, we slightly change the form of Eq. (8) to conduct the derivatives of $y_{i,j;t}$ (the output at the spatial location i, j of channel t) to $\{b_{c,v}\}$ (the codewords). From Eq. (8), $y_{i,j;t}$ can be written as

$$y_{i,j;t} = \sum_{v=1}^{\lceil d/r \rceil} \sum_{i_0=-w}^{w} \sum_{j_0=-h}^{h} \langle x_{i+i_0,j+j_0,\langle v \rangle}, b_{\pi(i_0,j_0,v,t);v} \rangle, \tag{10}$$

where $\langle \cdot, \cdot \rangle$ is the inner product. Let $\Phi = [b_{c,v}] \in R^{r \times C}$ be the matrix whose columns are the dim-r codewords of the v-th segment. Let $\beta_{i_0,j_0,v,t} \in R^C$ be the one-hot vector where the c-th entry in $\beta_{i_0,j_0,v,t}$ is 1 if $c = \pi(i_0, j_0, v, t)$; otherwise the entry is 0. Then, the codeword mapping $b_{\pi(i_0,j_0,v,t);v}$ in Eq. (8) can be replaced by $b_{\pi(i_0,j_0,v,t);v} = \Phi\beta_{i_0,j_0,v,t}$. Hence, the derivative of the **E**-Conv layer is conducted as

$$\frac{\partial y_{i,j;t}}{\partial \Phi} = \sum_{v=1}^{\lceil d/r \rceil} \sum_{i_0=-w}^{w} \sum_{j_0=-h}^{h} x_{i+i_0,j+j_0,\langle v \rangle} \beta^T_{i_0,j_0,v,t}. \tag{11}$$

As Φ is the matrix consisting of the codewords $\{b_{c,v}\}$, they can then be fine-tuned via the gradients for learning.

3.2. Merging Fully-connected Layers

The volume input to a FC layer is re-shaped to a vector in general. Let $x_F \in R^{N_I}$ be the input vector of a FC layer and $y_F \in R^{N_O}$ be its output. Then $y_F = \mathbf{W}x_F$, where $\mathbf{W} \in R^{N_O \times N_I}$ are the weights of the FC layer.

Unlike Conv layers that have sliding operations, all the operations in FC are summation-based. Thus, given the two weight matrices of models A and B, namely, $\mathbf{W}_A$ and $\mathbf{W}_B$, we simply divide them into length-r segment along the row direction. The dim-r weight vectors in the same segment are then clustered via k-means algorithm and C codewords are found. In this way an **E**-FC layer is built as well, and It is easy to show that the **E**-FC layer is also differentiable.

3.3. End-to-end Fine Tuning

As both **E**-Conv and **E**-FC layers are differentiable, once their codebooks are constructed, we can then fine-tune the entire model from some training data through end-to-end back-propagation learning.

Two error terms are combined for the minimization in calibration training. One is the classification (or regression) error employed in the original models A and B. The other is the layer-wise output mismatch error defined as follows: When applying the input x_I to the model A (or B), the output of every layer in the merged model should be close to the output of the associated layer in A (or B), and L_1-norm is used to measure this error.

The training data employed to fine-tune the merged model are called *calibration data* and the fine-tuning process is referred to as the *calibration training* in our approach. In the experiments, two settings about the size of training data are studied. As our method is used for the inference stage, sometimes we do not want to re-train the merged model through all the training data. Inspired by the network-distilling work [8], we only use a very limited set of data to fine-tune the model in this setting (1000 random samples per class in the experiments), and show that satisfactory results can be obtained. In the second setting, all the training data are used in our calibration training. Although it takes a longer time for training since the data amount is larger, but the accuracy is further improved.

In the calibration training, we use a framework (TensorFlow [1]) for the implementation. In the inference, to make the approach generalizable to mobile devices that may not contain GPUs, we utilize the C++ codes from [22] (which is based on Caffe's[10] CPU implementation) to realize the merged model on CPUs, with the OpenBLAS library [26, 21] used for matrix operations. Note that the codebooks are end-to-end trainable in our method, which is unlike [22] that only a single layer is tunable at one time, and thus our calibration training can be more efficiently realized via existing deep learning frameworks (such as TensorFlow). To make fair comparisons when inference, the Conv layers of the individual models compared with ours are realized via the unrolling convolution [4, 2] that converts the volume convolution into a single matrix product, which is commonly used as an efficient implementation for the Conv layer (eg., Caffe's CPU mode). In the experiments, we report the computation speed via the C++ codes, and show the test accuracy via Tensorflow. The accuracies of the same model could be different on the C++ (based on Caffe's CPU mode) and Tensorflow due to their framework details. The mismatch can be resolved by fine-tuning the model in the CPU mode in the future. Our codes will be availabe at https://github.com/ivclab/NeuralMerger.

4. Experiments

Merging Sound and Image Recognition CNNs

In the first experiment, we merge two CNNs of heterogeneous sources: image and sound. Though the sources are different, the same network (LeNet [13]) is used, which is applied to the following datasets of two tasks:

Sound dataset [11]: This dataset contains 13 classes of sounds recorded from drum kit and guitar. The 1D raw sig-

nals of sound are converted to 2D spectrograms and then resized to 32×32 images. There are more than 480 examples per class in the dataset, which are split into 70% training, 20% validation and 10% testing sets.

Fashion-MNIST [23]: This dataset contains a collection of gray-level images of dressing style of 10 classes. It has 60,000 training images and 10,000 testing images.

LeNet is a simple CNN consisting of 2 Conv layers with 32 and 64 kernels of size $5 \times 5 \times 1$ and $5 \times 5 \times 32$, respectively, each followed by a 2×2 max-pooling, and then a FC layer of 1024 units and a γ-way output (γ is the number of classes). Before merged, LeNet can achieve 95.4% and 91.6% accuracy on the above sound and image recognition tasks, respectively. The models are simply aligned and merged layer by layer since they are identical. As the input layer's depth of LeNet is 1, we set $r = 1$ for the first Conv layer of both models and choose $C = 64$ for it. Then, we alter the parameters of $r \in \{8, 16, 32\}$ and $C \in \{64, 128, 256\}$ for the remaining Conv- and FC-layers (except for the classification layer).

The overall performance and the average accuracy drop of Sound and Fashion-MNIST datasets are shown in Table 1. We select two sets of parameters. One is the 'ACCU' setting that prefers accuracy, and the other is the 'LIGHT' setting that prefers speedup. First, we use 1000 images per class for the calibration training in the fine-tuning phase. With the setting of ACCU, over 10 times compression of the joint model size and 4.17 times speedup per task are achieved with only a small accuracy drop (i.e., 1.25%). If the computation resource is limited, our approach can achieve over 15 times of joint model-size compression and 6.18 times speedup under LIGHT setting with 3.14% accuracy drop. Later, to further boost the performance, we employ all training samples for calibration training, and refer the two settings to as ACCU$^+$ and LIGHT$^+$. As can be seen, both the ACCU$^+$ and LIGHT$^+$ yield the merged models with only negligible accuracy drops, 0.34% and 0.59%, respectively, when leveraging on all the training data. The results suggest that our approach is effective for model merging for the inference stage. Note that the speedup ratio may vary under different hardware; here we use the CPU of NVIDIA Jetson TX1 in the single thread mode.

In the following, we detail the parameter settings of different layers for performance analysis. First, we fix the FC weights and jointly encode only the Conv layers. With the 1st Conv layer's r/C chosen as above, the 2nd layer's are varying and three settings (under $C = 128$) are selected as shown in Table 2(a) (with the FC weights fixed). As expected, the speedup is increased with r because the table indexing time is reduced; the accuracy is higher when r is lower because a finer subspace is divided. The best r/C, $8/128$ for the accuracy and $32/128$ for the compression and speedup, are chosen as the Conv-layer settings of 'ACCU' and 'LIGHT' in Table 1. Likewise, we also vary the parameters of the 1st FC layer with the Conv coefficients fixed, and select $r/C = 8/128$ and $8/64$ for the ACCU and LIGHT settings of the FC layer. To save the parameter setting times, only 1000 calibration data are used in the fine-tuning phase. Note that the speedup and compression in this table are evaluated based on the Conv or FC layers only.

Table 1. Overall Performance of the merged model. ACCU is the setting of $r/C = 8/128$ for both the 2nd Conv and 1st FC layers. LIGHT is the setting of $r/C = 32/128$ for the 2nd Conv and $8/64$ for the 1st FC layers of LeNet. The 1-st Conv layer is $1/64$ for both settings, while the classification layers are not co-compressed. 1000 training samples are used in the ACCU and LIGHT sttings, and all training samples are used in the ACCU$^+$ and LIGHT$^+$ settings for the calibration training, respectively.

Para.	Compres.	Speedup	Acc. Drop
ACCU	10.39 ×	4.17 ×	1.25%
ACCU$^+$			**0.34%**
LIGHT	15.34 ×	6.18 ×	3.14%
LIGHT$^+$			**0.59%**

Table 2. Performance of (a) varying the parameters of the 2nd Conv layer with the FC layers fixed and (b) varying the parameters of the 1st FC layer with the Conv layers fixed in the LeNet models. Note that the compression and speedup ratios are respectively evaluated for the Conv and FC layers only.

(a) Convolution Layer Settings

r/C	Compres.	Speedup	Acc. Drop
32/128	**20.41** ×	**7.2** ×	5.96%
16/128	17.60 ×	5.00 ×	3.86%
8/128	13.81 ×	3.00 ×	**1.81%**

(b) FC Layer Settings

r/C	Compres.	Speedup	Acc. Drop
8/64	**15.94** ×	**12.01** ×	0.47%
8/128	10.64 ×	10.63 ×	**0.19%**
8/256	6.39 ×	7.69 ×	0.43%

As our approach finds several codewords per layer and then performs end-to-end calibration training to refine the codewords, it is also applicable to an individual model. One may wonder how the individual models perform when they are compressed via our approach. For an even comparison, we use the same r/C settings to encode the single sound (or image) model and also fine-tune it using 1000 calibration samples. Hence, the compressed single model consumes the same resource as our merged one (i.e., they have exactly the same model size and execution time). As shown in Table 3, when the sound-only model is encoded via our approach, the accuracy drop remains to be 0.44%; that is, our merged model can achieve the same accuracy while an additional functionality (image recognition) is added. On the other hand, when the image-only model is encoded, the accuracy-drop only changes 0.25% (from

Table 3. Comparison of the merged model (ACCU parameter) with the individual sound and image models compressed via our approach. The individual models and the merged model use the same parameters of r/C and have identical model size and execution time, but more tasks can be done in the merged model.

Para.	Image Drop	Sound Drop	Avg. Drop
ACCU	2.06%	0.44%	**1.25%**
Sound only	N/A	0.44%	N/A
Image only	1.81%	N/A	N/A

Table 4. The parameters of r/C for 'ACCU' and 'LIGHT' settings in the clothing and gender merged model. Each group is a stack of two or three 3×3 Conv layers defined in VGG-16 model.

Para.	Group1,2	Group3	Group 4,5	Fc1	Fc2
ACCU	32/256	16/256	8/256	4/128	4/64
LIGHT	32/128	32/128	16/128	8/128	8/64

1.81% to 2.06%) with an extra functionality (sound recognition) supplemented. We owe that the two models have co-redundancy in between, and thus jointly encoding the models can take advantage of the additional inter-redundancy to achieve a better representation.

Merging Clothing and Gender CNN Classifiers

In the second experiment, we merge two image classifiers (one for clothing and the other for gender recognition) into a single model holding double functionalities. The clothing classifier adopted is [24] which substitutes the first FC layer in the VGG-16 model [20] with average-pooling; it has been shown that VGG-Avg achieves similar performance to VGG-16 while consuming far less storage. The gender classifier adopted is ZF-Net [25] that has fewer layers than VGG-Avg. The overall structures and alignment between them can be found in Fig. 2.

The clothing and gender datasets employed are depicted as follows. The Adience Dataset [5] contains face pictures of different genders and ages, where 11,136 images (each resized to 227×227) are used for training and the other 3,000 are for testing. The Multi-View-Clothing dataset [16] contains clothing images of different views with over 250 attributes organized in a tree, where the frontal images (37,485 for training and 3,000 for testing) and the 13 attributes of the first two layers are used to construct a classifier of 13 classes. Before merged, ZF-Net and VGG-Avg achieve 83.4% and 89.8% accuracy on the gender and clothing classification tasks, respectively. Because the kernel sizes in the two models are not always consistent, they are merged via the decomposition into $1 \times 1 \times r$ convolutions as described in Section 3.1.4. Similar to the process in the first experiment, we choose the parameters r/C and conduct two settings 'ACCU' and 'LIGHT', where the former stresses accuracy and the later enforces compression/speed. The detailed parameter settings are reported in Table 4.

Table 5. Overall Performance of the merged model for image classification of clothing and gender. (ACCU$^+$ and LIGHT$^+$ represent the calibration training using all samples)

Para.	Compres.	Clothing		Gender	
		Speedup	Acc. ↓	Speedup	Acc. ↓
ACCU	12.10×	2.58×	0.53%	1.87×	1.17%
ACCU$^+$			**-0.83%**		**0.27%**
LIGHT	20.10×	5.16×	2.80%	2.50×	3.27%
LIGHT$^+$			**0.50%**		**1.58%**

As the model sizes of both VGG-Avg and ZF-Net are larger than that of LeNet, there are more redundancies between the models. Hence, an even better performance is achieved than that of experiment 1. The overall performance is reported in Table 5. In the ACCU setting, the compression ratio exceeds 12 times with merely a slight accuracy drop 0.53% in the clothing task when 1000 calibration data are used. When calibration with all training data (ACCU$^+$ setting), the performance can be even better than that of the original model (with negative accuracy drop). In LIGHT/LIGHT$^+$ settings, the compression ratio is further enhanced with an acceptable accuracy drop.

Besides, when compressing the individual model (VGG-Avg for Clothing) by using our approach under the same setting, the accuracy drop is 0.58%, which is nearly the same as that achieved in our merged model (0.53%), but the merged model holds an extra capability: gender recognition. The results show that our approach can exploit both the intra- and inter-models redundancy and demonstrates satisfactory performance on merging deep learning models.

5. Conclusions

In this paper, we present a method that can unify multiple feed-forward models into a single but more compact one. To our best knowledge, this is the first study on merging deep models for the inference stage. The unified model is still differentiable and can be fine-tuned to restore or enhance the performance. Experimental results show that the merged model can be extensively compressed under negligible accuracy drops, which makes our method suitable for deep model inference on low-end devices.

A characteristic of our method is that the overall architectures of the original models are preserved when merging. The merged model may serve as a base model. For the cases when the resource in the inference stage is not an issue, a possible way to boost the multi-task performance is to extend the base model to a more complex architecture (eg., with a larger codebook and/or bridging layers added), which remains a future work. Besides, we also plan to incrementally merge more than two models in the future.

Acknowledgement: This work was supported in part by the MOST under the grant MOST 107-2634-F-001-004.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv*, 2016.

[2] S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017.

[3] Y. Aytar, C. Vondrick, and A. Torralba. See, hear, and read: Deep aligned representations. *CoRR*, abs/1706.00932, 2017.

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

[5] E. Eidinger, R. Enbar, and T. Hassner. Age and gender estimation of unfiltered faces. *IEEE TIFS*, 9(12):2170–2179, 2014.

[6] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[7] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, volume 2, page 6, 2017.

[8] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NIPS Workshops*, 2014.

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

[10] 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.

[11] A. Juliani. Recognizing sounds (a deep learning case study), 2016.

[12] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit. One model to learn them all. *CoRR*, abs/1706.05137, 2017.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] G. Levi and T. Hassner. Age and gender classification using convolutional neural networks. In *CVPR Workshops*, 2015.

[15] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *ICLR*, 2017.

[16] K. Liu, T. Chen, and C. Chen. Mvc: A dataset for view-invariant clothing retrieval and attribute prediction. In *Proceedings of ACM on International Conference on Multimedia Retrieval*, pages 313–316, 2016.

[17] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *ICLR*, 2017.

[18] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[19] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

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

[21] Q. Wang, X. Zhang, Y. Zhang, and Q. Yi. Augem: automatically generate high performance dense linear algebra kernels on x86 cpus. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 25. ACM, 2013.

[22] 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*, pages 4820–4828, 2016.

[23] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[24] H.-F. Yang, K. Lin, and C.-S. Chen. Supervised learning of semantics-preserving hash via deep convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 40(2):437–451, 2018.

[25] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision*, pages 818–833, 2014.

[26] X. Zhang, Q. Wang, and Y. Zhang. Model-driven level 3 blas performance optimization on loongson 3a processor. In *IEEE International Conference on Parallel and Distributed Systems*, pages 684–691. IEEE, 2012.