5-2023

On-device deep multi-task inference via multi-task zipping

Xiaoxi HE
ETH Zurich

Xu WANG
Tsinghua University

Zimu ZHOU
Singapore Management University, zimuzhou@smu.edu.sg

Jiahang WU
Tsinghua University

Zheng YANG
Tsinghua University

See next page for additional authors

On-Device Deep Multi-Task Inference via Multi-Task Zipping

Xiaoxi He, *Student Member, IEEE,* Xu Wang, *Student Member, IEEE,* Zimu Zhou, *Member, IEEE,* Jiahang Wu, *Student Member, IEEE,* Zheng Yang, *Senior Member, IEEE,* and Lothar Thiele, *Member, IEEE*

Abstract—Future mobile devices are anticipated to perceive, understand and react to the world on their own by running multiple correlated deep neural networks locally on-device. Yet the complexity of these deep models needs to be trimmed down both within-model and cross-model to fit in mobile storage and memory. Previous studies squeeze the redundancy within a single model. In this work, we aim to reduce the redundancy across multiple models. We propose Multi-Task Zipping (MTZ), a framework to automatically merge correlated, pre-trained deep neural networks for cross-model compression. Central in MTZ is a layer-wise neuron sharing and incoming weight updating scheme that induces a minimal change in the error function. MTZ inherits information from each model and demands light retraining to re-boost the accuracy of individual tasks. MTZ supports typical network layers (fully-connected, convolutional and residual) and applies to inference tasks with different input domains. Evaluations show that MTZ can fully merge the hidden layers of two VGG-16 networks with a 3.18% increase in the test error averaged on ImageNet for object classification and CelebA for facial attribute classification, or share 39.61% parameters between the two networks with $< 0.5\%$ increase in the test errors. The number of iterations to retrain the combined network is at least $17.8\times$ lower than that of training a single VGG-16 network. Moreover, MTZ can effectively merge nine residual networks for diverse inference tasks and models for different input domains. And with the model merged by MTZ, the latency to switch between these tasks on memory-constrained devices is reduced by $8.71\times$.

Index Terms—Deep Neural Networks; Model Compression; Multi-Task Learning;

1 Introduction

AI-powered mobile applications increasingly demand *multiple* deep neural networks for *correlated* tasks to be performed continuously and concurrently on resource-constrained mobile devices such as wearables, smartphones, and drones [1], [2], [3], [4], [5], [6]. Examples include wearable cameras that recognise objects and identify people for the visually impaired and drones that detect vehicles and identify road signs for traffic surveillance. While many pre-trained models for different inference tasks are available [7], [8], [9], it is often infeasible to deploy them directly on mobile devices due to their large memory footprints. For instance, VGG-16 models for object classification [9] and facial attribute classification [10] both contain over 130M parameters. Packing multiple such models easily strains mobile storage and memory at inference time.

Model compression [11] is an effective approach to radically reduce the size of a deep neural network without sacrificing its accuracy by pruning unimportant operations (pruning) [12], [13], [14], [15] or reducing the precision of operations (quantization) [16], [17]. However, all these proposals focus on *single-model compression*. Consequently, they generate sub-optimally compressed neural networks for multiple correlated inference tasks because there can still be notable redundancy across models due to task relatedness. For example, deep neural networks trained for different visual tasks tend to learn similar low-level features that resemble either Gabor filters or colour blobs [18]. Sharing information *among tasks* holds potential to further reduce the sizes of multiple correlated models without incurring drop in individual task inference accuracy.

We study information sharing in the context of *cross-model compression*, which seeks *effective* and *efficient* information sharing mechanisms among *pre-trained* models for multiple tasks to reduce the size of the combined model without accuracy loss in each task (see Fig. 1). A solution to cross-model compression is multi-task learning (MTL), a paradigm that jointly learns multiple tasks to improve the robustness and generalisation of tasks. However, most MTL studies use heuristically configured shared structures, which may lead to dramatic accuracy loss due to improper sharing of knowledge [19], [20]. Some recent proposals [21], [22] automatically decide "what to share" in deep neural networks. Yet deep MTL usually involves enormous training overhead [19]. Hence it is inefficient to ignore the already trained parameters in each model and apply MTL for cross-model compression.

In this paper, we propose Multi-Task Zipping (MTZ), a framework to automatically and adaptively merge correlated, well-trained deep neural networks for cross-model compression via neuron sharing. It decides the optimal sharable pairs of neurons on a layer basis and adjusts their

- *X. He and L. Thiele are with Computer Engineering and Networks Laboratory, ETH Zurich, Zurich, Switzerland, 8052. E-mail: {hex, thiele}@ethz.ch*
- *X. Wang, J. Wu and Z. Yang are with TNList and School of Software, Tsinghua Univeristy, Beijing, China, 100084. E-mail: xu-wang15@mails.tsinghua.edu.cn, jiahangok@gmail.com, yangzheng@tsinghua.edu.cn*
- *Z. Zhou is with School of Computing and Information Systems, Singapore Management University, Singapore, Singapore, 178902. E-mail: zimuzhou@smu.edu.sg*
- *Corresponding Author: Zimu Zhou.*

Manuscript received September, 2020; revised May, 2021; September, 2021.

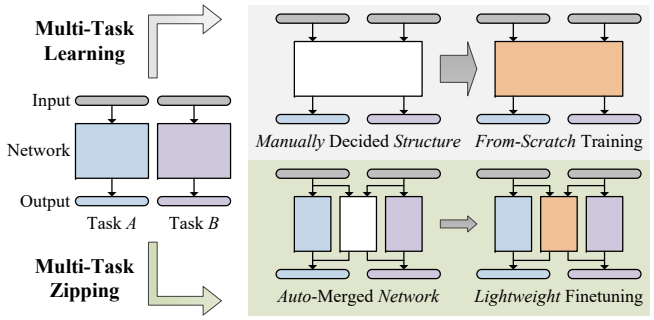


Fig. 1. Differences between multi-task learning (MTL) and our multi-task zipping (MTZ) as solutions to cross-model compression (illustrated with two tasks). Given two pre-trained models, MTL manually decides the shared structures, and then retrain the multi-task model from scratch *i.e.*, the original weights of the individual models are discarded. In contrast, our MTZ automatically determines what to share, and only a lightweight finetuning is necessary to re-boost the accuracy on each task because the original weights of the individual models are either kept (for non-shared neurons) or analytically calculated (for shared neurons).

incoming weights such that minimal errors are introduced in each task. Unlike MTL, MTZ inherits the parameters of each model and optimises the information to be shared among models such that only light retraining is necessary to resume the accuracy of individual tasks. In effect, it squeezes the *inter-network redundancy* from multiple already trained deep neural networks. MTZ may be further integrated with existing proposals for *single-model compression*, which reduce the *intra-network redundancy* via network pruning [12], [13], [14], [15] or network quantization [16], [17].

The contributions and results of this work are as follows.

- We propose MTZ, a framework that automatically merges multiple correlated, pre-trained deep neural networks. It squeezes the task relatedness across models via layer-wise neuron sharing, while requiring light retraining to re-boost the accuracy of the combined model. We also extend MTZ to support different layer types and tasks with different input domains. To the best of our knowledge, this is one of the first studies on cross-model compression for deep neural networks.
- MTZ managed to share 39.61% parameters between the two VGG-16 networks pre-trained for object classification (on ImageNet [23]) and facial attribute classification (on CelebA [24]), while incurring less than 0.5% increase in test errors. Even when all the hidden layers are fully merged, there is a moderate (averaged 3.18%) increase in test errors for both tasks. MTZ achieves the above performance with at least $17.9\times$ fewer iterations than training a single VGG-16 network from scratch [9].
- MTZ can merge models of different input domains (*e.g.*, audio- and video-based models), and is able to share 90% of the parameters among nine ResNets on nine different visual recognition tasks while inducing negligible loss on accuracy. Furthermore, with the joint model merged by MTZ, the latency to switch between these inference tasks on memory-constrained devices can be reduced by $8.71\times$.

A preliminary version of this work is presented in [25]. This paper has made the following additional contributions:

- We enhance the theoretical analysis of MTZ by showing that the accumulated error at the output layer in our *layer-wise* neuron sharing is bounded (Sec. 3.5).
- We propose an optimised network zipping scheme for ResNets (Sec. 4.2.2 to support batch normalisation layers and Sec. 4.2.3 to support residual blocks).
- We empirically show that MTZ can support different input domains *e.g.*, audio- and image-based models (Sec. 5.3) and is scalable in merging more than two networks (Sec. 5.4). Experimental results show that MTZ can merge 9 ResNets pre-trained for diverse visual inference tasks, which reduce the model storage from $9\times$ to only $1.8\times$ of a single ResNet, with marginal loss in all the 9 inference tasks. In addition, MTZ can reduce the latency by $8.71\times$ when switching between the 9 inference tasks on memory-constrained embedded platforms.

In the rest of this paper, we first review related work in Sec. 2, and then introduce our MTZ framework in Sec. 3 and its extensions in Sec. 4. We present the evaluations of MTZ in Sec. 5 and finally conclude in Sec. 6.

2 Related Work

MTZ compresses multiple well-trained deep neural networks of correlated inference tasks. It is relevant to research on multi-task learning and single-model compression. Our work belongs to the emerging field of cross-model compression and is complementary to resource scheduling of deep neural networks and edge-assisted inference.

2.1 Multi-Task Learning

Multi-task learning (MTL) jointly trains multiple correlated tasks to achieve higher accuracy than training each task individually. Determining "what to share" among tasks is a central issue in MTL, where can take place at different levels [19]. For MTL with neural networks, common techniques include hard or soft parameter sharing of the hidden layers [20]. Hard parameter sharing enforces sharing most or all of the parameters among all tasks while keeping a few task-specific output layers [26]. It causes notable accuracy drop when many tasks are trained jointly [27]. In soft parameter sharing, individual tasks are connected via information sharing [28]. The two sharing schemes can also be combined for more flexible parameter sharing *i.e.*, adaptively sharing a subset of parameters in the hidden layers [22].

The shared topology in most MTL studies is heuristically configured, which may lead to improper knowledge transfer [18]. Only a few schemes [21], [22] optimise *what to share among tasks*, especially for deep neural networks. Our MTZ resembles these automatic shared structure optimisation studies for MTL in effect, but differs in objectives. MTL jointly trains multiple tasks to improve their generalisation and accuracy, while MTZ aims to compress multiple *already trained* tasks with mild training overhead. Specifically, MTZ inherits the parameters directly from each pre-trained network when optimising the neurons shared among tasks in each layer and demands light retraining.

2.2 Single-Model Compression

There have been various model compression proposals to reduce the size of a *single* neural network without incurring loss in accuracy [11]. Pruning-based methods compress a deep neural network by eliminating unimportant operations such as weights [13], [14] or neurons [12], [29]. Neuron-level pruning is more desirable since it leads to regular sparsity in the pruned networks, and thus avoids the need for customised hardware [11]. The memory footprint of a neural network can be further reduced by lowering the precision of parameters (network quantization) [16], [17].

Unlike previous research that deals with the *intra-redundancy* of a single network, our work reduces the *inter-redundancy* among multiple networks. In principle, our method is a neuron-level *cross-model* pruning scheme. Our work may be integrated with single-model compression to further reduce the size of the combined neural network.

2.3 Cross-Model Compression

Cross-model compression aims to construct an accurate and compact multi-task neural network for efficient inference on resource-constrained platforms. Georgiev *et al.* [1] are the first to explore cross-model compression. They directly apply MTL techniques by heuristically configuring the shared structure and training the multi-task network from scratch. Our preliminary version [25] and NeuralMerger [3] are among the earliest studies to merge well-trained neural networks without training from scratch. NeuralMerger [3] utilises a joint encoding scheme for weight sharing, which can be understood as a cross-network *quantization*. Our technique is orthogonal to [3] since we focus on network *merging*. Neural weight virtualisation (NWV) [5] and ZipperNet [6] are two latest studies that explore merging for cross-model compression. NWV [5] shares all parameters among tasks and retrains to recover the accuracy *i.e.*, hard parameter sharing. ZipperNet [6] relaxes the constraint by a layer-wise merging strategy, *i.e.*, all the parameters are shared till a given layer. In contract, MTZ allows partial merging in each hidden layer. In addition, ZipperNet [6] adopts a heuristic neuron similarity metric and only applies to convolutional layers. In contrast, our MTZ shares neurons and updates weights via sensitivity analysis, and our method supports not only convolutional layers, but also fully connected layers, batch normalisation layers and residual blocks. We compare the performance with NWV [5] and ZipperNet [6] in Sec. 5.

2.4 Resource Scheduling for Deep Neural Networks

Orthogonal to reducing the complexity of deep neural networks themselves, resource scheduling algorithms enables efficient on-device execution of deep neural networks. DeepX [30] is a software accelerator that splits deep neural networks into blocks to be executed across multiple co-processors. DeepEye [2] proposes to interleave the execution of convolutional layers and fully-connected layers from multiple deep neural networks to improve the runtime efficiency of multi-model execution. NestDNN [4] designs a dynamic model pruning and recovery scheme and a resource-aware runtime scheduler to adaptively select the best models and allocate them to the available resources to maximise the overall inference accuracy and minimise the overall latency of concurrently running deep neural networks. As with [2], [4], our work also focuses on optimising multiple deep neural networks. However, our approach is complementary, which aims to reduce the memory footprint of multiple models by enforcing neuron sharing rather than scheduling their executions.

2.5 Edge-Assisted Deep Inference

In addition to on-device execution, offloading is also a popular strategy to run deep neural networks in the era of edge computing [31]. Particularly, the memory- or computation-intensive portion of a deep model can be offloaded to the edge to meet the resource constraints on end devices. For example, DeepDecision [32] dynamically decides whether to execute the model on-edge or on-device according to the available resources. Neurosurgeon [33] explores the optimal layer to partition a deep neural network for collaborative execution between the edge and the device that minimises latency and energy consumption. EdgeDuet [34] runs a full model on-edge and a compressed version on-device and only uploads image tiles to the full model when necessary. EalgeEye [35] partitions the multiple-model pipeline for face identification both spatially and temporally and runs the partitions in parallel on both the edge and the device. Magnum [36] adopts a lightweight blockchain-based framework to enable transfer learning in industrial IoT applications.

Our work is complementary to model partition. On the one hand, cross-model compression can be combined with model partition schemes for higher efficiency when running multiple tasks in edge-device collaborative inference. For instance, AMVP [37] proposes an adaptive scheduler that integrates single- and cross-model compression with model partition for multi-task video processing at the edge. On the other hand, model compression is preferable over model partition to applications where communication with the edge is prohibited due to data privacy or unreliable network connections [31], [38], [39], [40].

3 Layer-wise Network Zipping

This section explains the principles and details of our network zipping method with two feed-forward networks of dense fully connected (FC) layers. We discuss the extensions to other layers and settings in Sec. 4.

3.1 Problem Statement

Consider two inference tasks A and B with the corresponding *well-trained* deep neural networks M^A and M^B, *i.e.*, trained to a local minimum in error. We assume the same input domain and the same number of layers in M^A and M^B. Performing multiple correlated inference tasks on the same input domain is common in mobile applications (*e.g.*, face recognition, age and gender identification from a wearable camera [2], [5]; or speaker identification and ambient scene analysis from a smartphone microphone [1]). Note that our method also works for different input domains (see Sec. 5.3). The assumption on the same number of layers follows the practice in multi-task learning for ease of joint

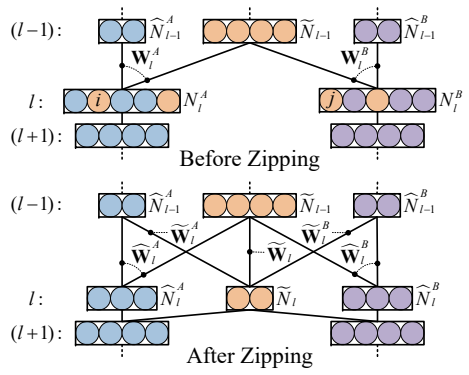


Fig. 2. An illustration of neurons and the corresponding weight matrices before and after zipping the l-th layers of M^A and M^B.

training [19]. Note that the models for different tasks can vary in the widths in their layers. Our goal is to construct a combined model M^C by sharing as many neurons between layers in M^A and M^B as possible such that *(i)* M^C has minimal loss in inference accuracy for the two tasks and *(ii)* the construction of M^C involves minimal retraining. As with other studies on cross-model compression [1], [3], [5], [6], the process to construct the combined model, *i.e.*, model merging and retraining, takes place offline on the cloud or the edge before model deployment. The combined model is then deployed to resource-constrained devices for accurate multi-task inference. Although extensive model training is affordable on the cloud/edge, it is still desirable to minimise the retraining overhead to allow fast model deployment and to serve more model merging requests at the same time.

3.2 Layer Zipping via Neuron Sharing

We take a layer-wise approach to the neuron sharing problem described in Sec. 3.1. This subsection presents the procedure of zipping the l-th layers ($1 \leq l \leq L-1$) in models M^A and M^B given the previous $(l-1)$ layers of the two models have been merged (see Fig. 2).

Denote the input layers as the 0-th layers. The L-th layers are the output layers of M^A and M^B. Denote the weight matrices of the l-th layers in M^A and M^B as $\mathbf{W}_l^A \in \mathbb{R}^{N_{l-1}^A \times N_l^A}$ and $\mathbf{W}_l^B \in \mathbb{R}^{N_{l-1}^B \times N_l^B}$, where N_l^A and N_l^B are the numbers of neurons in the l-th layers in M^A and M^B. Assume $\tilde{N}_{l-1} \in [0, \min\{N_{l-1}^A, N_{l-1}^B\}]$ neurons are shared between the $(l-1)$-th layers in M^A and M^B. Hence there are $\hat{N}_{l-1}^A = N_{l-1}^A - \tilde{N}_{l-1}$ and $\hat{N}_{l-1}^B = N_{l-1}^B - \tilde{N}_{l-1}$ task-specific neurons left in the $(l-1)$-th layers in M^A and M^B. Zipping the l-th layers in in M^A and M^B consists of two steps: *neuron sharing* and *weight matrices updating*.

3.2.1 Neuron Sharing

To enforce neuron sharing between the l-th layers in M^A and M^B, we calculate the *functional difference* (details in Sec. 3.3) between the i-th neuron in layer l in M^A, and the j-th neuron in the same layer in M^B. The functional difference is measured by a metric $d[\tilde{\mathbf{w}}_{l,i}^A, \tilde{\mathbf{w}}_{l,j}^B]$, where $\tilde{\mathbf{w}}_{l,i}^A, \tilde{\mathbf{w}}_{l,j}^B \in \mathbb{R}^{\tilde{N}_{l-1}}$ are the incoming weights of the two neurons from the *shared* neurons in the $(l-1)$-th layer. We do not alter incoming weights from the non-shared neurons in the $(l-1)$-th layer because they are likely to contain task-specific information only.

To zip the l-th layers in M^A and M^B, we first calculate the functional difference for each pair of neurons (i, j) in layer l and select $\tilde{N}_l \in [0, \min\{N_l^A, N_l^B\}]$ pairs with the smallest functional difference. These pairs of neurons form a set $\{(i_k, j_k)\}$, where $k = 0, \cdots, \tilde{N}_l$ and each pair is merged into one neuron. Thus the neurons in the l-th layers in M^A and M^B fall into three groups: $\tilde{N}_l$ shared, $\hat{N}_l^A = N_l^A - \tilde{N}_l$ specific for A and $\hat{N}_l^B = N_l^B - \tilde{N}_l$ specific for B.

3.2.2 Weight Matrices Updating

After neuron sharing, the weight matrices $\mathbf{W}_l^A$ and $\mathbf{W}_l^B$ are re-organised as follows. The weights vectors $\tilde{\mathbf{w}}_{l,i_k}^A$ and $\tilde{\mathbf{w}}_{l,j_k}^B$, where $k = 0, \cdots, \tilde{N}_l$, are merged and replaced by a matrix $\tilde{\mathbf{W}}_l \in \mathbb{R}^{\tilde{N}_{l-1} \times \tilde{N}_l}$, whose columns are $\tilde{\mathbf{w}}_{l,k} = f(\tilde{\mathbf{w}}_{l,i_k}^A, \tilde{\mathbf{w}}_{l,j_k}^B)$, where $f(\cdot)$ is an *incoming weight update function*. $\tilde{\mathbf{W}}_l$ represents the task-relatedness between A and B from layer $(l-1)$ to layer l. The incoming weights from the $\hat{N}_{l-1}^A$ neurons in layer $(l-1)$ to the $\hat{N}_l^A$ neurons in layer l in M^A form a matrix $\hat{\mathbf{W}}_l^A \in \mathbb{R}^{N_{l-1}^A \times \hat{N}_l^A}$. The remaining columns in $\mathbf{W}_l^A$ are packed as $\tilde{\mathbf{W}}_l^A \in \mathbb{R}^{\hat{N}_{l-1}^A \times \tilde{N}_l}$. Matrices $\hat{\mathbf{W}}_l^A$ and $\tilde{\mathbf{W}}_l^A$ contain the task-specific information for A between layer $(l-1)$ and layer l. For task B, we organise matrices $\hat{\mathbf{W}}_l^B \in \mathbb{R}^{N_{l-1}^B \times \hat{N}_l^B}$ and $\tilde{\mathbf{W}}_l^B \in \mathbb{R}^{\hat{N}_{l-1}^B \times \tilde{N}_l}$ in a similar manner. We also adjust the order of rows in the weight matrices in the $(l+1)$-th layers, $\mathbf{W}_{l+1}^A$ and $\mathbf{W}_{l+1}^B$, to maintain the correct connections among neurons.

The above layer zipping process can reduce $\tilde{N}_{l-1} \times \tilde{N}_l$ weights from $\mathbf{W}_l^A$ and $\mathbf{W}_l^B$. Essential in MTZ are the neuron functional difference metric $d[\cdot]$ and the incoming weight update function $f(\cdot)$. They are designed to demand only light retraining to recover the original accuracy.

3.3 Deriving Neuron Functional Difference Metric $d[\cdot]$ and Incoming Weight Update Function $f(\cdot)$

This subsection introduces our neuron functional difference metric $d[\cdot]$ and weight update function $f(\cdot)$ leveraging previous research on parameter sensitivity analysis [13], [14].

3.3.1 Preliminaries

A naive approach to accessing the impact of a change in some parameter vector $\boldsymbol{\theta}$ on the objective function (training error) E is to apply the parameter change and re-evaluate the error on the entire training data. An alternative is to exploit second order derivatives [13], [14]. Specifically, the Taylor series of the change δE in training error due to certain parameter vector change $\delta\boldsymbol{\theta}$ is [14]:

$$\delta E = \left(\frac{\partial E}{\partial \boldsymbol{\theta}}\right)^\top \cdot \delta\boldsymbol{\theta} + \frac{1}{2}\delta\boldsymbol{\theta}^\top \cdot \mathbf{H} \cdot \delta\boldsymbol{\theta} + O(\|\delta\boldsymbol{\theta}\|^3) \quad (1)$$

where $\mathbf{H} = \partial^2 E / \partial \boldsymbol{\theta}^2$ is the Hessian matrix containing all the second order derivatives. For a network trained to a local minimum in E, the first term vanishes. The third and higher order terms can also be ignored [14]. Hence:

$$\delta E = \frac{1}{2}\delta\boldsymbol{\theta}^\top \cdot \mathbf{H} \cdot \delta\boldsymbol{\theta} \quad (2)$$

Eq.(2) approximates the deviation in error due to parameter changes. However, it is still a bottleneck to compute and store the Hessian matrix $\mathbf{H}$ of a modern deep neural network. For instance, applying the weight pruning scheme proposed in [14] on a VGG-16 model [9] trained on the ImageNet ILSVRC-2012 dataset [23] requires the calculation of a Hessian matrix with approximately $(138 \times 10^6)^2 = 1.9044 \times 10^{16}$ elements.

As next, we harness the trick in [13] to break the calculations of Hessian matrices into layer-wise, and propose a Hessian-based neuron difference metric as well as the corresponding weight update function for neuron sharing.

3.3.2 Our Method

Inspired by [13] we define the error functions of M^A and M^B in layer l as

$$E_l^A = \frac{1}{n_A} \sum \|\tilde{\mathbf{y}}_l^A - \mathbf{y}_l^A\|^2 \tag{3}$$

$$E_l^B = \frac{1}{n_B} \sum \|\tilde{\mathbf{y}}_l^B - \mathbf{y}_l^B\|^2 \tag{4}$$

where $\mathbf{y}_l^A$ and $\tilde{\mathbf{y}}_l^A$ are the *pre-activation* outputs of the l-th layers in M^A before and after layer zipping, evaluated on one instance from the training set of A; $\mathbf{y}_l^B$ and $\tilde{\mathbf{y}}_l^B$ are defined in a similar way; $\|\cdot\|$ is l^2-norm; n_A and n_B are the number of training samples for M^A and M^B, respectively; Σ is the summation over all training instances. Since M^A and M^B are trained to a local minimum in training error, E_l^A and E_l^B will have the same minimum points as the corresponding training errors.

We further define an error function of the combined network in layer l as

$$E_l = \alpha E_l^A + (1-\alpha) E_l^B \tag{5}$$

where $\alpha \in (0,1)$ is used to balance the errors of M^A and M^B. The change in E_l with respect to neuron sharing in the l-th layer can be expressed in a similar form as Eq.(2):

$$\delta E_l = \frac{1}{2}(\delta\tilde{\mathbf{w}}_{l,i}^A)^\top \cdot \tilde{\mathbf{H}}_{l,i}^A \cdot \delta\tilde{\mathbf{w}}_{l,i}^A + \frac{1}{2}(\delta\tilde{\mathbf{w}}_{l,j}^B)^\top \cdot \tilde{\mathbf{H}}_{l,j}^B \cdot \delta\tilde{\mathbf{w}}_{l,j}^B \tag{6}$$

where $\delta\tilde{\mathbf{w}}_{l,i}^A$ and $\delta\tilde{\mathbf{w}}_{l,j}^B$ are the adjustments in the weights of i and j to merge the two neurons; $\tilde{\mathbf{H}}_{l,i}^A = \partial^2 E_l / (\partial\tilde{\mathbf{w}}_{l,i}^A)^2$ and $\tilde{\mathbf{H}}_{l,j}^B = \partial^2 E_l / (\partial\tilde{\mathbf{w}}_{l,j}^B)^2$ denote the *layer-wise* Hessian matrices. Similarly to [13], the layer-wise Hessian matrices can be calculated as

$$\tilde{\mathbf{H}}_{l,i}^A = \frac{\alpha}{n_A} \sum \mathbf{x}_{i-1}^A \cdot (\mathbf{x}_{i-1}^A)^\top \tag{7}$$

$$\tilde{\mathbf{H}}_{l,j}^B = \frac{1-\alpha}{n_B} \sum \mathbf{x}_{j-1}^B \cdot (\mathbf{x}_{j-1}^B)^\top \tag{8}$$

where $\mathbf{x}_{i-1}^A$ and $\mathbf{x}_{j-1}^B$ are the outputs of layer $(l-1)$ in M^A and M^B, respectively.

When sharing the i-th and j-th neurons in the l-th layers of M^A and M^B, our aim is to minimize δE_l, which can be formulated as the optimization problem below:

$$\min_{(i,j)} \{ \min_{(\delta\tilde{\mathbf{w}}_{l,i}^A, \delta\tilde{\mathbf{w}}_{l,j}^B)} \delta E_l \} \text{ s.t. } \tilde{\mathbf{w}}_{l,i}^A + \delta\tilde{\mathbf{w}}_{l,i}^A = \tilde{\mathbf{w}}_{l,j}^B + \delta\tilde{\mathbf{w}}_{l,j}^B \tag{9}$$

For the inner minimization problem:

$$\min_{(\delta\tilde{\mathbf{w}}_{l,i}^A, \delta\tilde{\mathbf{w}}_{l,j}^B)} \delta E_l \quad \text{s.t. } \tilde{\mathbf{w}}_{l,i}^A + \delta\tilde{\mathbf{w}}_{l,i}^A = \tilde{\mathbf{w}}_{l,j}^B + \delta\tilde{\mathbf{w}}_{l,j}^B \tag{10}$$

we form Lagrange multipliers with the second order approximation in (2):

$$\begin{aligned} L = &\frac{1}{2}(\delta\tilde{\mathbf{w}}_{l,i}^A)^\top \cdot \tilde{\mathbf{H}}_{l,i}^A \cdot \delta\tilde{\mathbf{w}}_{l,i}^A + \frac{1}{2}(\delta\tilde{\mathbf{w}}_{l,j}^B)^\top \cdot \tilde{\mathbf{H}}_{l,j}^B \cdot \delta\tilde{\mathbf{w}}_{l,j}^B \\ &+ \boldsymbol{\lambda}^\top \cdot (\tilde{\mathbf{w}}_{l,i}^A + \delta\tilde{\mathbf{w}}_{l,i}^A - \tilde{\mathbf{w}}_{l,j}^B - \delta\tilde{\mathbf{w}}_{l,j}^B) \end{aligned} \tag{11}$$

where $\boldsymbol{\lambda}$ is the vector of Lagrange undetermined multipliers. By taking functional derivatives and employing the constraints of Eq.(9), we have closed-form solutions:

$$\begin{aligned} \delta\tilde{\mathbf{w}}_{l,i}^{A,opt} = &(\tilde{\mathbf{H}}_{l,i}^A)^{-1} \cdot \Big((\tilde{\mathbf{H}}_{l,i}^A)^{-1} + (\tilde{\mathbf{H}}_{l,j}^B)^{-1} \Big)^{-1} \\ &\cdot (\tilde{\mathbf{w}}_{l,j}^B - \tilde{\mathbf{w}}_{l,i}^A) \end{aligned} \tag{12}$$

$$\begin{aligned} \delta\tilde{\mathbf{w}}_{l,j}^{B,opt} = &(\tilde{\mathbf{H}}_{l,j}^B)^{-1} \cdot \Big((\tilde{\mathbf{H}}_{l,i}^A)^{-1} + (\tilde{\mathbf{H}}_{l,j}^B)^{-1} \Big)^{-1} \\ &\cdot (\tilde{\mathbf{w}}_{l,i}^A - \tilde{\mathbf{w}}_{l,j}^B) \end{aligned} \tag{13}$$

$$\begin{aligned} \delta E_l^{opt} = &\frac{1}{2}(\tilde{\mathbf{w}}_{l,i}^A - \tilde{\mathbf{w}}_{l,j}^B)^\top \cdot \Big((\tilde{\mathbf{H}}_{l,i}^A)^{-1} + (\tilde{\mathbf{H}}_{l,j}^B)^{-1} \Big)^{-1} \\ &\cdot (\tilde{\mathbf{w}}_{l,i}^A - \tilde{\mathbf{w}}_{l,j}^B) \end{aligned} \tag{14}$$

We define the neuron functional difference metric as:

$$d[\tilde{\mathbf{w}}_{l,i}^A, \tilde{\mathbf{w}}_{l,j}^B] = \delta E_l^{opt} \tag{15}$$

and the weight update function as:

$$f(\tilde{\mathbf{w}}_{l,i}^A, \tilde{\mathbf{w}}_{l,j}^B) = \tilde{\mathbf{w}}_{l,i}^A + \delta\tilde{\mathbf{w}}_{l,i}^{A,opt} = \tilde{\mathbf{w}}_{l,j}^B + \delta\tilde{\mathbf{w}}_{l,j}^{B,opt}. \tag{16}$$

3.4 MTZ Framework

Algorithm 1 outlines the process of MTZ on two tasks of the same input domain, *e.g.*, images. We first construct a joint input layer. In case the input layer dimensions are not equal in both tasks, the dimension of the joint input layer equals the larger dimension of the two original input layers, and fictive connections (*i.e.*, weight 0) are added to the model whose original input layers are smaller. Afterwards we begin layer-wise neuron sharing and weight matrix updating from the first hidden layer. The two networks are "zipped" layer by layer till the last hidden layer and we obtain a combined network. After merging each layer, the networks are retrained to re-boost the accuracy.

Practical Issues. We make the following notes on the practicability of MTZ.

- *How to set the number of neurons to be shared?* One can directly set $\tilde{N}_l$ neurons to be shared for the l-th layers, or set a layer-wise threshold ε_l instead. Given a threshold ε_l, MTZ shares pairs of neurons where $\{(i_k, j_k) | d[\tilde{\mathbf{w}}_{l,i_k}^A, \tilde{\mathbf{w}}_{l,j_k}^B] < \varepsilon_l\}$. In this case $\tilde{N}_l = |\{(i_k, j_k)\}|$. One can set $\{\tilde{N}_l\}$ if there is a hard constraint on storage or memory. Otherwise $\{\varepsilon_l\}$ can be set if accuracy is of higher priority. Note that $\{\varepsilon_l\}$ controls the layer-wise error δE_l, which correlates to the accumulated errors of the outputs in layer L $\tilde{\varepsilon}^A = \frac{1}{\sqrt{n_A}} \sum \|\tilde{\mathbf{x}}_L^A - \mathbf{x}_L^A\|$ and $\tilde{\varepsilon}^B = \frac{1}{\sqrt{n_B}} \sum \|\tilde{\mathbf{x}}_L^B - \mathbf{x}_L^B\|$ [13].
- *How to execute the combined model for each task?* During inference, only task-related connections in the combined model are enabled. For instance, when performing inference on task A, we only activate $\{\hat{\mathbf{W}}_l^A\}$, $\{\tilde{\mathbf{W}}_l^A\}$ and $\{\tilde{\mathbf{W}}_l\}$, while $\{\tilde{\mathbf{W}}_l^B\}$ and $\{\hat{\mathbf{W}}_l^B\}$ are disabled (*e.g.*, by setting them to zero).

Algorithm 1: Multi-task Zipping via Layer-wise Neuron Sharing

input : $\{\mathbf{W}_l^A\}, \{\mathbf{W}_l^B\}$: weight matrices of M^A, M^B; $\mathbf{X}^A, \mathbf{X}^B$: training datum of task A and B (including labels); α: coefficient to balance M^A and M^B; $\{\tilde{N}_l\}$: number of neurons to be shared in layer l

1 **for** $l = 1, \ldots, L-1$ **do**
2 Calculate inputs for the current layer $\mathbf{x}_{l-1}^A$ and $\mathbf{x}_{l-1}^B$ using training data from $\mathbf{X}^A$ and $\mathbf{X}^B$ and forward propagation
3 $\tilde{\mathbf{H}}_{l,i}^A \leftarrow \frac{\alpha}{n_A} \sum \mathbf{x}_{i-1}^A \cdot (\mathbf{x}_{i-1}^A)^\top$
4 $\tilde{\mathbf{H}}_{l,j}^B \leftarrow \frac{1-\alpha}{n_B} \sum \mathbf{x}_{j-1}^B \cdot (\mathbf{x}_{j-1}^B)^\top$
5 Select $\tilde{N}_l$ pairs of neurons $\{(i_k, j_k)\}$ with the smallest $d[\tilde{\mathbf{w}}_{l,i}^A, \tilde{\mathbf{w}}_{l,j}^B]$
6 **for** $k \leftarrow 1, \ldots, \tilde{N}_l$ **do**
7 $\tilde{\mathbf{w}}_{l,k} \leftarrow f(\tilde{\mathbf{w}}_{l,i_k}^A, \tilde{\mathbf{w}}_{l,j_k}^B)$
8 Re-organize $\mathbf{W}_l^A$ and $\mathbf{W}_l^B$ into $\tilde{\mathbf{W}}_l$, $\hat{\mathbf{W}}_l^A$, $\tilde{\mathbf{W}}_l^A$, $\hat{\mathbf{W}}_l^B$ and $\tilde{\mathbf{W}}_l^B$
9 Permute the order of rows in $\mathbf{W}_{l+1}^A$ and $\mathbf{W}_{l+1}^B$ to maintain correct connections
10 Conduct a light retraining on task A and B to re-boost accuracy of the joint model

output: $\{\hat{\mathbf{W}}_l^A\}, \{\tilde{\mathbf{W}}_l^A\}, \{\tilde{\mathbf{W}}_l\}, \{\tilde{\mathbf{W}}_l^B\}, \{\hat{\mathbf{W}}_l^B\}$: weights of the zipped multi-task model M^C

- *How to zip more than two neural networks?* MTZ is able to zip more than two models by sequentially adding each network into the joint network, and the calculated Hessian matrices of the already zipped joint network can be reused. Therefore, MTZ is scalable in regards to both the depth of each network and the number of tasks to be zipped. Also note that since calculating the Hessian matrix of one layer requires only its layer input, only one forward pass in total from each model is needed for the merging process (excluding retraining).

Complexity Analysis. We only analyse the complexity of the main merging process (Line 3 to 9) and ignore the complexity of the forward and backward propagation of neural networks (Line 2 and 10). For simplicity, we consider fully merging two layers from two networks, which both have n neurons in every hidden layer. Line 3 and 4 take $O(n^2)$ time and $O(n^2)$ memory. Line 5 takes $O(n^5)$ time and $O(n^2)$ memory (using in-place matrix inversion) to calculate all the paring distances, and then $O(n^4)$ to sort them. This is the most time consuming step, as the calculation of the merging criterion (15) involves inversion of the Hessian matrices. There are $O(n)$ iterations in Line 6 and 7, and line 7 takes $O(n^3)$ time and $O(n^2)$ memory. Line 8 and 9 take $O(n^2)$ time and $O(n^2)$ memory. Therefore, the total time cost is $O(n^5)$ and memory cost is $O(n^2)$.

3.5 Propagation of Layer-wise Error

Note that we define layer-wise error function Eq.(5) to avoid calculating the entire Hessian matrix. In this subsection, we demonstrate the effectiveness of such a layer-wise formulation by proving that the accumulated error at the output layer is bounded.

To analyse the accumulated error at the output layer, we investigate how the error Eq.(5) propagates from layer l to the final output layer. Note that the error function in layer l consists of two parts, E_l^A and E_l^B, as defined in Eq.(3) and Eq.(4), which are defined with pre-activation outputs $\mathbf{y}_l^A$ and $\tilde{\mathbf{y}}_l^A$. In order to understand the propagation of errors, however, we need to take activation function into consideration. We define:

$$\mathcal{E}_l^A = \frac{1}{n_A} \sum \|\tilde{\mathbf{z}}_l^A - \mathbf{z}_l^A\|^2 \tag{17}$$

$$\mathcal{E}_l^B = \frac{1}{n_B} \sum \|\tilde{\mathbf{z}}_l^B - \mathbf{z}_l^B\|^2 \tag{18}$$

$$\mathcal{E}_l = \alpha \cdot \mathcal{E}_l^A + (1-\alpha) \cdot \mathcal{E}_l^B \tag{19}$$

where $\mathbf{z}_l^A = \sigma(\mathbf{y}_l^A)$, $\tilde{\mathbf{z}}_l^A = \sigma(\tilde{\mathbf{y}}_l^A)$, $\mathbf{z}_l^B = \sigma(\mathbf{y}_l^B)$ and $\tilde{\mathbf{z}}_l^B = \sigma(\tilde{\mathbf{y}}_l^B)$ are *post-activation* layer outputs with activation function $\sigma(\cdot)$. In this paper, we consider the widely adopted activation function: rectified linear unit (ReLU).

After merging the l-th layer, there are three groups of neurons: $\hat{N}_l^A$ task-A-specific neurons, $\hat{N}_l^B$ task-B-specific neurons, and $\tilde{N}_l$ shared neurons. When task A is performed, only task-A-specific and shared neurons are activated. The connections between the task-A-specific and shared neurons in the $l-1$-th and l-th layer have weights $\mathbf{W}_l'^A$:

$$\mathbf{W}_l'^A = \left[\begin{array}{c|c} \hat{\mathbf{W}}_l^A & \begin{matrix}\tilde{\mathbf{W}}_l^A \\ \tilde{\mathbf{W}}_l\end{matrix} \end{array}\right] \tag{20}$$

Similarly, we can define $\mathbf{W}_l'^B$. Furthermore, we denote the vectorisation of the weight matrices $\mathbf{W}_l'^A$ and $\mathbf{W}_l'^B$ as $\mathcal{V}_l^A$ and $\mathcal{V}_l^B$, respectively.

Adapting the conclusions in [13] to multiple neural networks, the propagation of the layer-wise error in MTZ can be described by the following theorem:

Theorem 1. For a multi-task network merged via Algorithm 1 with L layers, the accumulated error of the last layer output is upper-bounded by:

$$\begin{aligned}\mathcal{E}_L \leq \sum_{i=1}^{L-1} \Bigg(& \alpha \cdot \prod_{j=i+1}^{L-1} \|\mathcal{V}_j^A\| \sqrt{\delta E_i^A} \\ & + (1-\alpha) \cdot \prod_{j=i+1}^{L-1} \|\mathcal{V}_j^B\| \sqrt{\delta E_i^B} \Bigg)\end{aligned} \tag{21}$$

Proof. From similar derivation as in [13], we have:

$$\mathcal{E}_L^A \leq \sum_{i=1}^{L-1} \left(\prod_{j=i+1}^{L} \|\mathcal{V}_j^A\| \sqrt{\delta E_i^A} \right) + \sqrt{\delta E_L^A} \tag{22}$$

and the same holds if we switch A with B. However, as the last layer, *i.e.*, the output layer is untouched, we have $\sqrt{\delta E_L^A} = \sqrt{\delta E_L^B} = 0$. Therefore:

$$\mathcal{E}_L^A \leq \sum_{i=1}^{L-1} \left(\prod_{j=i+1}^{L-1} \|\mathcal{V}_j^A\| \sqrt{\delta E_i^A} \right) \tag{23}$$

which also holds if we swap A and B. Finally, since $\mathcal{E}_l = \alpha \cdot \mathcal{E}_l^A + (1-\alpha) \cdot \mathcal{E}_l^B$, Eq.(21) holds. □

4 MTZ Extensions

In this section, we explain how to extend MTZ to support sparse models (Sec. 4.1), other commonly used layers in computer vision *e.g.*, convolutional (CONV) layers, batch normalisation (BN) layers and residual blocks (Sec. 4.2).

4.1 Support for Sparse Models

Since the pre-trained neural networks may have already been sparsified via weight pruning, we also extend MTZ to support sparse models. Specifically, we use sparse matrices, where zeros indicate no connections, to represent such sparse models. Then the incoming weights from the previous shared neurons $\tilde{\mathbf{w}}^A_{l,i}, \tilde{\mathbf{w}}^B_{l,j}$ still have the same dimension. Therefore $d[\tilde{\mathbf{w}}^A_{l,i}, \tilde{\mathbf{w}}^B_{l,j}]$, $f(\tilde{\mathbf{w}}^A_{l,i}, \tilde{\mathbf{w}}^B_{l,j})$ can be calculated as before. However, we also calculate two mask vectors $\tilde{\mathbf{m}}^A_{l,i}$ and $\tilde{\mathbf{m}}^B_{l,j}$, whose elements are 0 when the corresponding elements in $\tilde{\mathbf{w}}^A_{l,i}$ and $\tilde{\mathbf{w}}^B_{l,j}$ are 0, and 1 otherwise. We pick the mask vector with more $1's$ and apply it to $\tilde{\mathbf{w}}_l$. This way the combined model always have a smaller number of weights than the sum of the original two models.

4.2 Extension to Other Layers

This subsection introduces how to extend MTZ from FC layers to CONV layers, BN layers and residual blocks.

4.2.1 Extensions to Convolutional Layers

The layer zipping procedure of two convolutional layers are similar to that of two fully connected layers. The only difference is that sharing is performed on *kernels* rather than *neurons*. Take the i-th kernel of size $k_l \times k_l$ in layer l of M^A as an example. Its incoming weights from the previous shared kernels are $\tilde{\mathbf{W}}^{A,in}_{l,i} \in \mathbb{R}^{k_l \times k_l \times \tilde{N}_{l-1}}$. The weights are then flatten into a vector $\tilde{\mathbf{w}}^A_{l,i}$ to calculate functional differences. As with in Sec. 3.2, after layer zipping in the l-th layers, the weight matrices in the $(l+1)$-th layers need careful permutations regarding the flattening ordering to maintain correct connections among neurons, especially when the next layers are fully connected layers.

4.2.2 Extensions to Batch Normalisation Layers

BN layers are typically applied on the pre-activation outputs of CONV layers. After training, the output of the BN layer applied on the i-th channel of layer l is:

$$BN(y_{l,i}) = \gamma_{l,i} \cdot \frac{y_{l,i} - \mu_{l,i}}{\sqrt{\sigma^2_{l,i} + \epsilon}} + \beta_{l,i} \tag{24}$$

where $y_{l,i}$ is the pre-activation output of the CONV layer, $\gamma_{l,i}$ and $\beta_{l,i}$ are the two learnable parameters (scaling and shifting) for the BN layer, $\mu_{l,i}$ and $\sigma_{l,i}$ are the pre-calculated mean and standard deviation.

Since all the parameters are fixed after training, the effect of the BN layer can be replaced by multiplying the incoming weight $\mathbf{w}_{l,i}$ by a scalar $\frac{\gamma_{l,i}}{\sigma_{l,i}}$ and adding $\beta_{l,i} - \frac{\gamma_{l,i} \cdot \mu_{l,i}}{\sigma_{l,i}}$ to the bias $b_{l,i}$. The calculation of the Hessian matrices (7) and (8) remains the same, and in the closed-form solutions Eq.(12), Eq.(13) and Eq.(14) the new weights and bias should be used. Later in the retraining phase, newly initialised BN layers need to be applied.

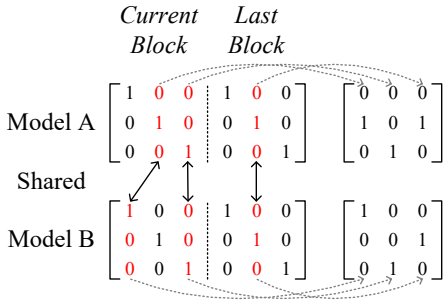


Fig. 3. An example of the weight matrices when merging residual blocks with output dimension of three.

4.2.3 Extensions to Residual Blocks

At the end of each residual block, the output vector of the last CONV layer is added with the identity shortcut vector. This addition can be considered as a layer of neurons (channels) with binary weights (1 or 0) fully connected to the last convolutional layer and the last shortcut addition layer (or in the case of the first residual block, it connect to the pre-convolutional layer). However, in order to continue the chain of MTZ, the neurons at this addition layer should be marked as shared/unshared. Since the neuron sharing situation of the last CONV layer in the current residual block can be different from of the addition layer of the last residual block, there might be conflicts. We propose an exact and an approximate method to combine residual blocks.

Exact Method. We illustrate the exact method via an example residual block with output dimension of three. In the last CONV layer of the current block, the first neuron in model A is shared with the second neuron in model B, and the third neuron in model A is shared with the third neuron in model B. In the addition layer of the last block, the second neuron in model A is shared with the second neuron in model B. Fig. 3 illustrates the weight matrices of the current addition layers of model A and B, where the red column indicates the weights from merged neurons, and the weight matrices from the merged neurons.

As shown in this example, we mark the neurons in the addition layer as shared/unshared as follows:

- If a neuron in the addition layer is not connecting (*i.e.*, having non-zero weights) to any shared convolutional neuron or shared shortcut neuron, *e.g.*, the first row in the matrices on the right of Fig. 3, this neuron is marked as unshared.
- In other cases, these addition neurons are merged by analysing their incoming weights from merged neurons, *e.g.*, the matrices on the right of Fig. 3.

Approximate Method. The exact method above requires an additional layer to be processed by MTZ, and actually adds more weights to the model. To avoid this problem, we propose an approximate method to merge residual blocks. The idea is to use the neuron sharing scheme of the last CONV layer in the current residual block as the reference. In other words, in the last CONV layer in current residual block, if the i-th neuron in model A is merged with j-th neuron in model B, then their corresponding neurons in the addition layer are also marked as merged.

5 EVALUATIONS

We first evaluate the performance of MTZ on zipping two networks pre-trained for the same task (Sec. 5.1) and different tasks (Sec. 5.2). We mainly assess the test errors of each task after network zipping and the retraining overhead involved. We then show that MTZ can merge models for different input domains (Sec. 5.3). Finally we show that MTZ is scalable and reduces the execution time of neural networks on resource-constrained mobile devices (Sec. 5.4). MTZ is implemented with TensorFlow. All experiments are conducted on a workstation equipped with Nvidia Titan X Maxwell GPU. The execution time of neural networks is measured on the Jetson Nano embedded platform [41] equipped with a 128-core Maxwell GPU, a Quad-core ARM A57 (1.43GHz) CPU, and 4GB 64-bit LPDDR4 (25.6GB/s).

5.1 Zipping Two Networks for the Same Task

This experiment validates the effectiveness of MTZ by merging two differently trained models for the same task. Ideally, two models trained to different local optimums should function the same on the test data. Therefore their hidden layers can be fully merged without incurring any accuracy loss. This experiment aims to show that, by finding the correct pairs of neurons which shares the same functionality, MTZ can achieve the theoretical limit of compression ratio *i.e.*, 100%, even without any retraining involved.

Dataset and Settings. We test on MNIST dataset with the LeNet-300-100 and LeNet-5 networks [7] to recognise handwritten digits from 0 to 9. LeNet-300-100 is a fully connected network with two hidden layers (300 and 100 neurons each), reporting an error from 1.6% to 1.76% on MNIST [7], [13]. LeNet-5 is a convolutional network with two convolutional layers and two fully connected layers, which achieves an error from 0.8% to 1.27% on MNIST [7], [13].

We train two LeNet-300-100 networks of our own with errors of 1.57% and 1.60%; and two LeNet-5 networks with errors of 0.89% and 0.95%. All the networks are initialised randomly with different seeds, and the training data are also shuffled before every training epoch. After training, the ordering of neurons/kernels in all hidden layers is once more randomly permuted. Therefore the models have completely different parameters (weights). The training of LeNet-300-100 and LeNet-5 networks requires 1.05×10^4 and 1.1×10^4 iterations in average, respectively.

For sparse networks, we apply one iteration of L-OBS [13] to prune the weights of the four LeNet networks. We then enforce all neurons to be shared in each hidden layer of the two dense LeNet-300-100 networks, sparse LeNet-300-100 networks, dense LeNet-5 networks, and sparse LeNet-5 networks, using MTZ.

Results. Fig. 4a plots the average error after sharing different amounts of neurons in the first layers of two dense LeNet-300-100 networks. Fig. 4b shows the error by further merging the second layers. We compare MTZ with a random sharing scheme, which shares neurons by first picking (i_k, j_k) at random, and then choosing randomly between $\tilde{\mathbf{w}}^A_{l,i_k}$ and $\tilde{\mathbf{w}}^B_{l,j_k}$ as the shared weights $\tilde{\mathbf{w}}_{l_k}$. When all the 300 neurons in the first hidden layers are shared, there is an increase of 0.95% in test error (averaged over the two models) even without retraining, while random sharing induces an error of 33.47%. We also use MTZ to fully merge the hidden layers in the two LeNet-300-100 networks without any retraining *i.e.*, without line 10 in Algorithm 1. The averaged test error increases by only 1.50%.

TABLE 1
Test errors on MNIST by sharing all neurons in two LeNet networks.

Model	err_A	err_B	re-err_C	# re-iter
LeNet-300-100-Dense	1.57%	1.60%	1.64%	550
LeNet-300-100-Sparse	1.80%	1.81%	1.83%	800
LeNet-5-Dense	0.89%	0.95%	0.93%	600
LeNet-5-Sparse	1.27%	1.28%	1.29%	1200

Table 1 summarises the errors of each LeNet pair before zipping (err_A and err_B), after fully merged with retraining (re-err_C) and the number of retraining iterations involved (# re-iter). MTZ consistently achieves lossless network zipping on FC and CONV networks, either they are dense or sparse, with 100% parameters of hidden layers shared. Meanwhile, the number of retraining iterations is approximately $19.0\times$ and $18.7\times$ fewer than that of training a dense LeNet-300-100 network and a dense LeNet-5 network, respectively.

5.2 Zipping Two Networks for Different Tasks

This experiment evaluates the performance of MTZ to automatically share information among two neural networks for different tasks. We investigate: *(i)* what the accuracy loss is when all hidden layers of two models for different tasks are fully shared (in purpose of maximal size reduction); *(ii)* how much neurons and parameters can be shared between the two models by MTZ with at most 0.5% increase in test errors allowed (in purpose of minimal accuracy loss); *(iii)* how MTZ performs compared with the state-of-the-art cross-model compression schemes [5], [6].

Dataset and Settings. We first test the performance of MTZ for either *maximal size reduction* or *minimal accuracy loss*. We merge two VGG-16 networks trained on the ImageNet ILSVRC-2012 dataset [23] for object classification and the CelebA dataset [24] for facial attribute classification. The ImageNet dataset contains images of $1,000$ object categories. The CelebA dataset consists of 200 thousand celebrity face images labelled with 40 attribute classes. VGG-16 has 13 convolutional layers and 3 fully connected layers. We adopt the pre-trained weights from the original VGG-16 model [9] for the object classification task, which has a 10.31% error. For the facial attribute classification task, we train a second VGG-16 model following a similar process as in [10]. We initialise the convolutional layers of a VGG-16 model using the pre-trained parameters from imdb-wiki [8], and train the remaining 3 fully connected layers till the model yields an error of 8.50%, which matches the accuracy of the VGG-16 model in [10] on CelebA.

We conduct two experiments with the two VGG-16 models. *(i)* All hidden layers in the two models are 100% merged using MTZ. *(ii)* Each pair of layers in the two models are adaptively merged using MTZ allowing an increase ($< 0.5\%$) in test errors on the two datasets.

Results. Table 2 summarises the performance when each pair of hidden layers are 100% merged. The test errors of

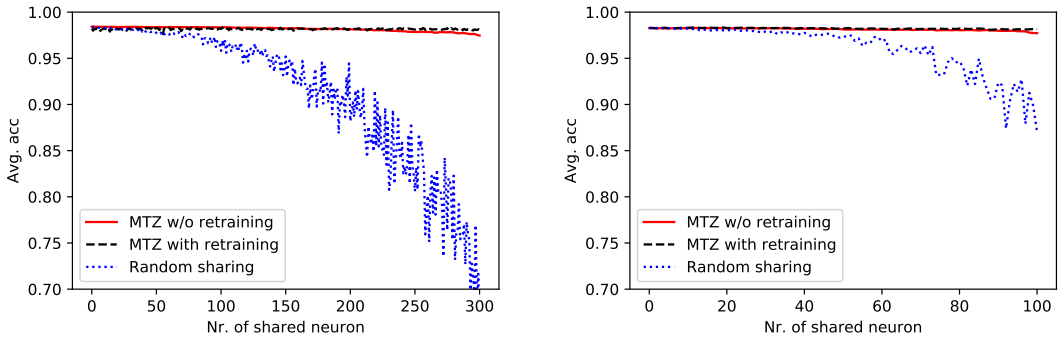


Fig. 4. Test error on MNIST by continually sharing neurons in (a) the first and (b) the second fully connected layers of two dense LeNet-300-100 networks till the merged layers are fully shared.

TABLE 2
Test errors and retraining iterations of sharing all neurons (output layer fc8 excluded) in two VGG-16 networks for ImageNet and CelebA.

Layer	N_l^A	ImageNet (Top-5 Error)		CelebA (Error)		# re-iter
		w/o-re-err$_C$	re-err$_C$	w/o-re-err$_C$	re-err$_C$	
conv1_1	64	10.59%	10.61%	8.45%	8.43%	50
conv1_2	64	11.19%	10.78%	8.82%	8.77%	100
conv2_1	128	10.99%	10.68%	8.91%	8.82%	100
conv2_2	128	11.31%	11.03%	9.23%	9.07%	100
conv3_1	256	11.65%	11.46%	9.16%	9.04%	100
conv3_2	256	11.92%	11.83%	9.17%	9.05%	100
conv3_3	256	12.54%	12.41%	9.46%	9.34%	100
conv4_1	512	13.40%	12.28%	10.18%	9.69%	400
conv4_2	512	13.02%	12.62%	10.65%	10.25%	400
conv4_3	512	13.11%	12.97%	12.03%	10.92%	400
conv5_1	512	13.46%	13.09%	12.62%	11.68%	400
conv5_2	512	13.77%	13.20%	12.61%	11.64%	400
conv5_3	512	36.07%	13.35%	13.10%	12.01%	1×10^3
fc6	4096	15.08%	15.17%	12.31%	11.71%	2×10^3
fc7	4096	15.73%	14.07%	11.98%	11.09%	1×10^4

both tasks gradually increase during the zipping procedure from layer conv1_1 to conv5_2 and then the error on ImageNet surges when conv5_3 are 100% shared. After $1,000$ iterations of retraining, the accuracies of both tasks are resumed. When 100% parameters of all hidden layers are shared between the two models, the joint model yields test errors of 14.07% on ImageNet and 11.09% on CelebA, *i.e.*, increases of 3.76% and 2.59% in the original test errors.

Table 3 shows the performance when each pair of hidden layers are adaptively merged. MTZ achieves an increase in test errors of 0.44% on ImageNet and 0.45% on CelebA. Approximately 39.61% of the parameters in the two models are shared (56.94% in the 13 CONV layers and 38.17% in the 2 FC layers). The zipping procedure involves $20,650$ iterations of retraining. For comparison, at least 3.7×10^5 iterations are needed to train a VGG-16 network from scratch [9]. That is, MTZ is able to inherit information from the pre-trained models and construct a combined model with an increase in test errors of less than 0.5%. And the process requires at least $17.9\times$ fewer (re)training iterations than training a joint network from scratch.

For comparison, we also trained a fully shared multi-task VGG-16 with two split classification layers jointly on both tasks. The test errors are 14.88% on ImageNet and 13.29% on CelebA. This model has exactly the same topology and amount of parameters as our model constructed by MTZ, but performs slightly worse on both tasks.

Comparison with State-of-the-Arts. We compare our MTZ against two recent cross-model compression schemes, Neural Weight Virtualisation [5] (NWV for short) and ZipperNet [6] in the *fully shared* setting, *i.e.*, in purpose of *maximal size reduction*. We choose the fully shared setting because NWV adopts hard parameter sharing, *i.e.*, parameters are shared across all tasks. ZipperNet allows partial parameter sharing but all the weights *within* a layer are fully shared.

To compare with ZipperNet [6], we apply it to merge two VGG-16 networks pre-trained on ImageNet and CelabA as above. Specifically, we perform filter alignment by Hungarian algorithm for each CONV layer and then retrain the merged network as [6]. The merging process starts with the first CONV layer and continues until all CONV layers are merged. Since ZipperNet does not apply to FC layers, we leave the two FC layers in the VGG-16 separate. For fair comparison, the number of retraining iterations for each CONV layer is set to the same as our MTZ (detailed numbers see Table 2). Fig. 5 plots the test errors after fully

TABLE 3
Test errors, number of shared neurons, and retraining iterations of adaptively zipping two VGG-16 networks for ImageNet and CelebA.

Layer	N_l^A	$\tilde{N}_l$	ImageNet (Top-5 Error)		CelebA (Error)		# re-iter
			w/o-re-err$_C$	re-err$_C$	w/o-re-err$_C$	re-err$_C$	
conv1_1	64	64	10.28%	10.37%	8.39%	8.33%	50
conv1_2	64	64	10.93%	10.50%	8.77%	8.54%	100
conv2_1	128	96	10.74%	10.57%	8.62%	8.46%	100
conv2_2	128	96	10.87%	10.79%	8.56%	8.47%	100
conv3_1	256	192	10.83%	10.76%	8.62%	8.48%	100
conv3_2	256	192	10.92%	10.71%	8.52%	8.44%	100
conv3_3	256	192	10.86%	10.71%	8.83%	8.63%	100
conv4_1	512	384	10.69%	10.51%	9.39%	8.71%	400
conv4_2	512	320	10.43%	10.46%	9.06%	8.80%	400
conv4_3	512	320	10.56%	10.36%	9.36%	8.93%	400
conv5_1	512	436	10.42%	10.51%	9.54%	9.15%	400
conv5_2	512	436	10.47%	10.49%	9.43%	9.16%	400
conv5_3	512	436	10.49%	10.24%	9.61%	9.07%	1×10^3
fc6	4096	1792	11.46%	11.33%	9.37%	9.18%	2×10^3
fc7	4096	4096	11.45%	10.75%	9.15%	8.95%	1.5×10^4

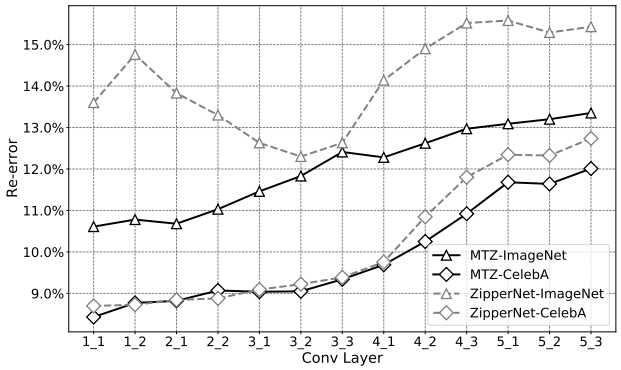


Fig. 5. Test errors after merging each CONV layer in two VGG-16 networks for ImageNet and CelebA using ZipperNet [6].

TABLE 4
Test errors of the joint network merged by NWV [5] and MTZ. The joint model is fully shared except the last classification layer.

	GTSRB	SVHN	Average
NWV	7.04%	7.65%	7.35%
MTZ	1.05%	6.86%	3.96%

TABLE 5
Test errors of the joint network by merging ResNet-28 (pretrained on GTSRB) and ResNet-34 (pretrained on SVHN) using MTZ. The joint model is merged layer-by-layer from the first layers.

	GTSRB	SVHN	Average
MTZ	1.14%	6.80%	3.97%

merging each CONV layer in the two VGG-16 networks. In general, ZipperNet performs worse than MTZ in terms of accuracy after merging. The difference gets increasingly evident after merging the conv4_2 layer. In the end, ZipperNet only achieves a test errors of 15.43% on ImageNet and 12.74% on CelebA, 2.08% and 0.73% worse than MTZ.

For comparison with NWV [5], we merge two ResNet-28 networks [42] to cover diverse model architectures. The two ResNet-28 networks are pre-trained on German Traffic Sign Recognition Benchmark (GTSRB) [43] and Street View House Numbers (SVHN) [44]. Table 7 provides a summary of the datasets. When merging the two ResNet-28 networks with NWV, the weights are organised into $5,822$ weight-pages with a page size of $1,000$. Then the weight-pages are retrained as in NWV to recover the inference accuracy. Table 4 lists the test errors of the fully shared ResNet-28 model merged by NWV and MTZ. MTZ achieves better accuracy than NMV on both tasks after merging. Compared with NWV, MTZ yields 5.99% lower error on GTSRB and 0.79% on SVHN.

To demonstrate the feasibility of MTZ on merging models of similar architectures yet different depth, we further merge a ResNet-28 network and a ResNet-34 network, which are pretrained respectively on GTSRB and SVHN. The two networks are merged layer-by-layer from the first layers and the extra layers in the ResNet-34 are left independent. Table 5 shows the test errors of the merged model. We can see that the merged model performs well on both tasks, showing the effectiveness of MTZ merging networks with different depth.

5.3 Zipping Two Networks for Different Input Domains

This experiment evaluates the performance of MTZ to merge two models for different input domains. The aim is to show the potential memory saving to share information among different models, even if they are designed for different input domains, *e.g.*, one for audio and the other for visual input. This is common in mobile and ubiquitous computing with multiple sensing modalities.

Dataset and Settings. We experiment with an audio-visual emotion classification task, where we perform emotion recognition from speech and facial expression [45]. We use the same models as [45], where one ResNet-28 network is used to extract audio representations and the other ResNet-28 network is used to extract facial features from video frames. The audio and video features are concatenated and fed to full connected layers for emotion recognition.

The performance of emotion recognition is assessed on the RML audio-visual dataset [46]. The RML database contains 720 utterances from 8 participants with 6 emotions: anger, disgust, fear, joy, sadness, and surprise.

We first train the two ResNet-28 networks for audio emotion classification (ResNet-28-Audio) and video emotion classification (ResNet-28-Video) respectively. The two models are then merged via full connected layers and fine-tuned for audio-emotion classification (ResNet-28-Fusion). Finally we enforce sharing 90% of the neurons in the two networks (the last classification layer excluded) via MTZ, which leads to a joint model of $1.1\times$ the size of a single ResNet-28 network.

Results. Table 6 shows the accuracy of different models on emotion classification. Comparing ResNet-28-Fusion with ResNet-28-Audio and ResNet-28-Video, the emotion recognition error drops significantly from around 40% to about 25%. However, this accuracy gain is at the cost of double the size of the model, *i.e.*, having twice the number of parameters of a single ResNet-28 network. Our MTZ method is able to enforce information sharing between ResNet-28-Audio and ResNet-28-Video. Specifically, compared with ResNet-28-Fusion, our joint model only has 1.1 times the number of the parameters (in contrast to 2 times), with only a 3.68% drop in emotion recognition accuracy. The results indicate that *inter-redundancy* is not limited to models with the same input domain and MTZ is able to suppress inter-redundancy among models for different input domains.

5.4 Zipping More Than Two Networks

This experiment evaluates the scalability of MTZ and the benefit of cross-model compression for running multiple models on embedded platforms. We investigate: *(i)* what the accuracy loss is if more than two models are merged; and *(ii)* what is the reduction in the model-switching time on resource-limited devices if multiple models are merged.

Dataset and Settings. We adopt the models and datasets in [47], a recent work on multi-task learning with ResNets. Specifically, nine ResNet-28 networks [42] are trained for diverse image recognition tasks, including CIFAR100 [48], German Traffic Sign Recognition Benchmark (GTSRB) [43], Omniglot [49], Street View House Numbers (SVHN) [44], UCF101 [50], Flowers102 [51], Daimler Mono Pedestrian Classification Benchmark (DPed) [52], Describable Texture Dataset (DTD) [53] and FGVC-Aircraft [54]. Table 7 provides a brief summary of the datasets.

To test the scalability of MTZ, we enforce sharing 90% of the neurons in a single ResNet-28 network with the other eight models, and evaluate the accuracy of the joint model on each of the nine task.

To show the benefit of executing a compact joint model, we measure the delay when switching between the nine inference tasks on the Jetson Nano embedded platform [41]. A 32GB Sandisk microSD is connected to the platform to storage the neural networks. To perform inference tasks on-device, the corresponding model should be loaded from the microSD to the memory. When a new task is performed, the parameters of the new model are loaded and the old parameters in the memory are overwritten, as the memory resource is limited. Fig. 6 illustrates the setup to measure the execution time of the visual inference tasks on the embedded platform. To measure the model-switching time, tasks are performed in a random sequence but each one from the 9 tasks is performed 10 times.

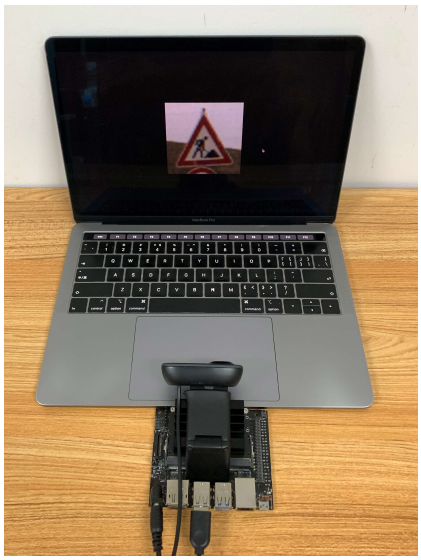


Fig. 6. Hardware setup to measure the execution time of the nine inference tasks (the photo was for the traffic sign recognition task, *i.e.*, the GTSRB dataset [43]) on the Jetson Nano embedded platform.

Results. Table 8 shows the accuracy of each individual pre-trained model and the joint model on the nine tasks. Compared with each individual model, the accuracy of the joint model only drops by 0.46% (averaged across the nine tasks). However, the total storage for the nine models decreases from $9\times$ to only $1.8\times$ of a single ResNet-28 network. The results show that MTZ is able to enforce neuron sharing among dozens of models for diverse tasks while retaining the inference accuracy on each task.

Fig. 7 shows the averaged time needed to update the parameters in the memory for new tasks. As 90% of the parameters are shared, only 10% of the parameters in the memory are needed to be updated when we use the joint model. Hence the model-switching time when using the joint model is significantly lower than that when using individual models. In general, the joint model is able to achieve $8.71\times$ lower model-switching time.

6 Conclusion

We propose MTZ, a framework to automatically merge multiple correlated, well-trained deep neural networks for cross-model compression via neuron sharing. It selectively shares neurons and optimally updates their incoming weights on a layer basis to minimise the errors induced to each individual task. Only light retraining is necessary to resume the accuracy of the joint model on each task. Evaluations show that MTZ can fully merge two VGG-16 networks with an error increase of 3.76% and 2.59% on ImageNet for object classification and on CelebA for facial attribute classification, or share 39.61% parameters between the two models with $< 0.5\%$ error increase. The number of iterations to retrain the combined model is $17.9\times$ lower than that of training a single VGG-16 network. Meanwhile, MTZ

TABLE 6
Test errors of audio emotion classification network, visual emotion classification network and the joint network merged by MTZ. $1\times$ is the number of parameters of one single ResNet excluding the last classification layer.

Model	ResNet-28-Audio	ResNet-28-Video	ResNet-28-Fusion	MTZ
#par.	$1.0\times$	$1.0\times$	$2.0\times$	$1.1\times$
Test Error	43.02%	39.65%	24.74%	28.42%

TABLE 7
A brief description of the datasets that each ResNet-28 network is trained on.

Dataset	Brief Description
CIFAR100 [48]	$60,000$ colour images of 100 different object categories
GTSRB [43]	$50,000+$ images for 43 traffic sign classes in different resolutions
Omniglot [49]	$1,623$ different handwritten characters from 50 different alphabets
SVHN [44]	$70,000$ images of digits cropped from street views
UCF101 [50]	$13,320$ videos clips collected from YouTube for 101 action categories
Flowers102 [51]	$8,189$ images of 102 categories of flowers
DPed [52]	$50,000$ grey-scale images of pedestrians and non-pedestrians
DTD [53]	$5,640$ texture images of 47 terms (categories) *e.g.*, bubbly
FGVC-Aircraft [54]	$10,000$ images of 100 different aircraft models *e.g.*, Airbus A310

TABLE 8
Test errors of pre-trained single ResNets and the joint network merged by MTZ. The joint model is compressed to $1.8\times$ of a single model. $1\times$ is the number of parameters of a single ResNet-28 excluding the last classification layer. Without MTZ the joint model would have a size of $9\times$.

	CIFAR100	GTSRB	Omniglot	SVHN	UCF101	Flowers102	DPed	DTD	FGVC-Aircraft	**Average**
Single	28.97%	0.56%	14.97%	6.04%	36.68%	37.45%	0.56%	67.61%	58.75%	**27.96%**
Joint	31.88%	0.51%	16.94%	6.70%	36.22%	37.35%	0.51%	67.71%	58.30%	**28.42%**

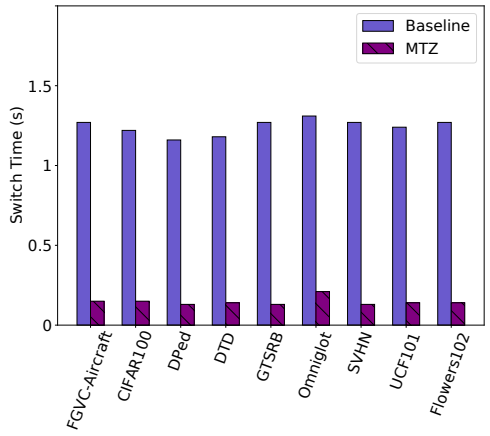


Fig. 7. Averaged model-switching time between the nine tasks.

is able to share 90% of the parameters among nine ResNets on nine different visual recognition tasks while inducing negligible loss on accuracy. The joint model also reduces the model-switching time between these inference tasks on memory-constrained devices by $8.71\times$. Experiments also show that MTZ is applicable to sparse networks and models for different input domains.

REFERENCES

[1] P. Georgiev, S. Bhattacharya, N. D. Lane, and C. Mascolo, "Low-resource multi-task audio sensing for mobile and embedded devices via shared deep neural network representations," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, pp. 50:1–50:19, 2017.

[2] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proceedings of ACM Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 68–81.

[3] Y.-M. Chou, Y.-M. Chan, J.-H. Lee, C.-Y. Chiu, and C.-S. Chen, "Unifying and merging well-trained deep neural networks for inference stage," in *Proceedings of International Joint Conference on Artificial Intelligence*, 2018, pp. 2049–2056.

[4] B. Fang, X. Zeng, and M. Zhang, "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proceedings of ACM Annual International Conference on Mobile Computing and Networking*, 2018, pp. 115–127.

[5] S. Lee and S. Nirjon, "Fast and scalable in-memory deep multitask learning via neural weight virtualization," in *Proceedings of ACM Annual International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 175–190.

[6] C.-E. Wu, J.-H. Lee, T. S. Wan, Y.-M. Chan, and C.-S. Chen, "Merging well-trained deep cnn models for efficient inference," in *Proceedings of IEEE Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, 2020, pp. 1594–1600.

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

[8] R. Rothe, R. Timofte, and L. Van Gool, "Deep expectation of real and apparent age from a single image without facial landmarks," *International Journal of Computer Vision*, vol. 126, no. 2-4, pp. 144–157, 2018.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[10] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. Feris, "Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification," in *Proceedings of IEEE/CVF Con-*

ference on Computer Vision and Pattern Recognition, 2017, pp. 5334–5343.

[11] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: a comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[12] B. Dai, C. Zhu, B. Guo, and D. Wipf, "Compressing neural networks using the variational information bottleneck," in *Proceedings of ACM International Conference on Machine Learning*, 2018, pp. 1143–1152.

[13] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 2017, pp. 4860–4874.

[14] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 1993, pp. 164–171.

[15] S. Liu, J. Du, K. Nan, Z. Zhou, H. Liu, Z. Wang, and Y. Lin, "Adadeep: A usage-driven, automated deep model compression framework for enabling ubiquitous intelligent mobiles," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[16] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

[17] Z. Qu, Z. Zhou, Y. Cheng, and L. Thiele, "Adaptive loss-aware quantization for multi-bit networks," in *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7988–7997.

[18] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems*, 2014, pp. 3320–3328.

[19] Y. Zhang and Q. Yang, "An overview of multi-task learning," *National Science Review*, vol. 5, no. 1, pp. 30–43, 2018.

[20] S. Ruder, "An overview of multi-task learning in deep neural networks," 2017.

[21] Y. Yang and T. Hospedales, "Deep multi-task representation learning: A tensor factorisation approach," in *Proceedings of International Conference on Learning Representations*, 2016.

[22] X. Sun, R. Panda, R. Feris, and K. Saenko, "Adashare: Learning what to share for efficient deep multi-task learning," in *Advances in Neural Information Processing Systems*, 2020, pp. 8728–8740.

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[24] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of IEEE International Conference on Computer Vision*, 2015, pp. 3730–3738.

[25] X. He, Z. Zhou, and L. Thiele, "Multi-task zipping via layer-wise neuron sharing," in *Advances in Neural Information Processing Systems*, 2018, pp. 6019–6029.

[26] H. Bilen and A. Vedaldi, "Integrated perception with recurrent multi-task neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 235–243.

[27] T. Standley, A. Zamir, D. Chen, L. Guibas, J. Malik, and S. Savarese, "Which tasks should be learned together in multi-task learning?" in *Proceedings of ACM International Conference on Machine Learning*, 2020, pp. 9120–9132.

[28] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3994–4003.

[29] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 264–11 272.

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

[31] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[32] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proceedings of IEEE International Conference on Computer Communications*, 2018, pp. 1421–1429.

[33] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[34] X. Wang, Z. Yang, J. Wu, Y. Zhao, and Z. Zhou, "Edgeduet: Tiling small object detection for edge assisted autonomous mobile vision," in *Proceedings of IEEE International Conference on Computer Communications*, 2021, pp. 1–1.

[35] J. Yi, S. Choi, and Y. Lee, "Eagleeye: Wearable camera-based person identification in crowded urban spaces," in *Proceedings of ACM Annual International Conference on Mobile Computing and Networking*, 2020, pp. 1–14.

[36] P. K. Deb, S. Misra, T. Sarkar, and A. Mukherjee, "Magnum: A distributed framework for enabling transfer learning in b5g-enabled industrial-iot," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 10, pp. 7133–7140, 2020.

[37] M. Chao, R. Stoleru, L. Jin, S. Yao, M. Maurice, and R. Blalock, "Amvp: Adaptive cnn-based multitask video processing on mobile stream processing platforms," in *Proceedings of IEEE/ACM Symposium on Edge Computing (SEC)*, 2020, pp. 96–109.

[38] Z. Ouyang, J. Niu, Y. Liu, and M. Guizani, "Deep cnn-based real-time traffic light detector for self-driving vehicles," *IEEE Transactions on Mobile Computing*, vol. 19, no. 2, pp. 300–313, 2020.

[39] C. Wang, Y. Xiao, X. Gao, L. Li, and J. Wang, "A framework for behavioral biometric authentication using deep metric learning on mobile devices," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[40] Y. Zhang, T. Gu, and X. Zhang, "Mdldroidlite: a release-and-inhibit control approach to resource-efficient deep neural networks on mobile devices," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[41] NVIDIA, "Jetson nano developer kit," https://developer.nvidia.com/embedded/jetson-nano-developer-kit, 2020.

[42] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016.

[43] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, vol. 32, pp. 323–332, 2012.

[44] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2, 2011, p. 5.

[45] S. Zhang, S. Zhang, T. Huang, and W. Gao, "Multimodal deep convolutional neural network for audio-visual emotion recognition," in *Proceedings of ACM on International Conference on Multimedia Retrieval*, 2016, pp. 281–284.

[46] Y. Wang and L. Guan, "Recognizing human emotional state from audiovisual signals," *IEEE Transactions on Multimedia*, vol. 10, no. 5, pp. 936–946, 2008.

[47] S.-A. Rebuffi, H. Bilen, and A. Vedaldi, "Learning multiple visual domains with residual adapters," in *Advances in Neural Information Processing Systems*, 2017, pp. 506–516.

[48] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[49] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[50] K. Soomro, A. R. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," 2012.

[51] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Proccesings of IEEE Indian Conference on Computer Vision, Graphics & Image Processing*, 2008, pp. 722–729.

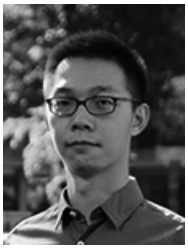
[52] S. Munder and D. M. Gavrila, "An experimental study on pedestrian classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1863–1868, 2006.

[53] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3606–3613.

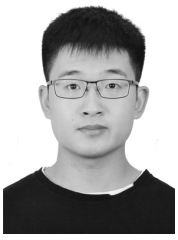
[54] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," 2013.

Xiaoxi He received his B.Sc. degree in electrical engineering and information technology from Leibniz University Hannover in 2015, and the M.Sc degree from ETH Zurich in 2017. He has been a Ph.D. Candidate with the group of Prof. L. Thiele, Computer Engineering and Networks Laboratory, ETH Zurich, since 2017. His current research interests include deep learning theory and machine learning applications in mobile embedded system.

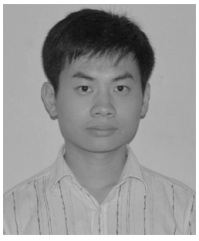
Xu Wang received his B.E. degree and Ph.D. in School of Software from Tsinghua University in 2015 and 2020. He is currently a postdoctoral researcher in School of Software at Tsinghua University. He is a member of the Tsinghua National Lab for Information Science and Technology. His research interests include mobile computing and machine learning.



Zimu Zhou received his B.E. in 2011 in the Department of Electronic Engineering, Tsinghua University and his Ph.D. in 2015 in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He is currently an assistant professor at the School of Information Systems, Singapore Management University. His research interests include mobile and ubiquitous computing.



Jiahang Wu received his B.E. degree in School of Software from Beijing Jiaotong University in 2020. He is currently an M.E. student in School of Software at Tsinghua University. He is a member of the Tsinghua National Lab for Information Science and Technology. His research interests include Internet of Things and edge computing.



Zheng Yang received a B.E. degree in computer science from Tsinghua University in 2006 and a Ph.D. degree in computer science from Hong Kong University of Science and Technology in 2010. He is currently a professor in Tsinghua University. His main research interests include wireless ad-hoc/sensor networks and mobile computing.

Lother Thiele joined ETH Zurich, Switzerland, as a full professor of computer engineering in 1994, where he currently leads the Computer Engineering and Networks Laboratory. He received his DiplomIngenieur and Dr.-Ing. degrees in Electrical Engineering from the Technical University of Munich in 1981 and 1985 respectively. His research interests include models, methods and software tools for the design of embedded systems, embedded software and bioinspired optimization techniques. Lothar Thiele is associate editor of IEEE Transaction on Industrial Informatics, IEEE Transactions on Evolutionary Computation, Journal of Real-Time Systems, Journal of Signal Processing Systems, Journal of Systems Architecture, and INTEGRATION, the VLSI Journal. In 1986 he received the Dissertation Award of the Technical University of Munich, in 1987, the Outstanding Young Author Award of the IEEE Circuits and Systems Society, in 1988, the Browder J. Thompson Memorial Award of the IEEE, and in 2000/2001, the IBM Faculty Partnership Award. In 2004, he joined the German Academy of Sciences Leopoldina. In 2005, he was the recipient of the Honorary Blaise Pascal Chair of University Leiden, The Netherlands. Since 2009 he is a member of the Foundation Board of Hasler Foundation, Switzerland. Since 2010, he is a member of the Academia Europaea. In 2013, he joined the National Research Council of the Swiss National Science Foundation. Lothar Thiele received the "EDAA Lifetime Achievement Award" in 2015. Since 2017, Lothar Thiele is Associate Vice President of ETH for Digital Transformation.