Modularizing while Training: A New Paradigm for Modularizing DNN Models

Binhang Qi
SKLSDE Lab, Beihang University
China
binhangqi@buaa.edu.cn

Hailong Sun†
SKLSDE Lab, Beihang University
China
sunhl@buaa.edu.cn

Hongyu Zhang
Chongqing University
China
hyzhang@cqu.edu.cn

Ruobing Zhao
SKLSDE Lab, Beihang University
China
rbingzhao@buaa.edu.cn

Xiang Gao†
SKLSDE Lab, Beihang University
China
xiang_gao@buaa.edu.cn

ABSTRACT

Deep neural network (DNN) models have become increasingly crucial components of intelligent software systems. However, training a DNN model is typically expensive in terms of both time and computational resources. To address this issue, recent research has focused on reusing existing DNN models - borrowing the concept of software reuse in software engineering. However, reusing an entire model could cause extra overhead or inherit the weaknesses from the undesired functionalities. Hence, existing work proposes to decompose an already trained model into modules, i.e., *modularizing-after-training*, to enable module reuse. Since the trained models are not built for modularization, modularizing-after-training may incur huge overhead and model accuracy loss. In this paper, we propose a novel approach that incorporates modularization into the model training process, i.e., *modularizing-while-training* (MwT). We train a model to be structurally modular through two loss functions that optimize intra-module cohesion and inter-module coupling. We have implemented the proposed approach for modularizing Convolutional Neural Network (CNN) models. The evaluation results on representative models demonstrate that MwT outperforms the existing state-of-the-art modularizing-after-training approach. Specifically, the accuracy loss caused by MwT is only 1.13 percentage points, which is less than that of the existing approach. The kernel retention rate of the modules generated by MwT is only 14.58%, with a reduction of 74.31% over the existing approach. Furthermore, the total time cost required for training and modularizing is only 108 minutes, which is half the time required by the existing approach. Our work demonstrates that MwT is a new and more effective paradigm for realizing DNN model modularization, offering a fresh perspective on achieving model reuse.

ICSE 2024, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04…$15.00
https://doi.org/10.1145/3597503.3608135

KEYWORDS

DNN Modularization, Model Reuse, Modular Training, Convolutional Neural Network

ACM Reference Format:
Binhang Qi, Hailong Sun†, Hongyu Zhang, Ruobing Zhao, and Xiang Gao†. 2024. Modularizing while Training: A New Paradigm for Modularizing DNN Models. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE 2024), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3597503.3608135

1 INTRODUCTION

Software reuse [11, 15, 30] can facilitate software development. According to Gartner[41], over 95% of companies use at least one open-source software component in building their business products, highlighting the importance of software reuse. Like open-source software, more and more deep neural network (DNN) models, trained for various tasks [34, 44, 50], are also publicly available on model sharing platforms such as GitHub and Hugging Face [3]. These DNN models have become increasingly crucial components of intelligent software systems. Reusing these models to facilitate the development of intelligent software systems has sparked interests in recent years [12, 18, 20, 24, 32, 33, 46].

However, reusing a whole DNN model causes extra overhead [32, 33] or inherits the weakness from the undesired functionalities [19, 33, 52]. Modularization [26, 27, 45] is a software design technique that involves separating a program into independent and reusable modules. Each module is self-contained and responsible for executing a part of the desired functionality. This paradigm enables software reuse by allowing developers to reuse specific packages or functions in their programs. Borrowing the idea of modularization in software engineering, researchers propose to modularize DNN models, then reuse the resultant modules. For example, Qi et al. [32, 33] and Pan et al. [23, 24] proposed to decompose a trained DNN model into modules by identifying neurons or weights that are responsible for classifying each class and removing irrelevant neurons and weights. They then use the resultant modules for constructing, patching, or transferring DNN models.

Existing work [23, 24, 32, 33, 35, 52] on modularizing DNN models follows the paradigm of *modularizing after training*, i.e., they first train a model as a whole or find an already trained model, then

†Corresponding authors: Hailong Sun and Xiang Gao. Hailong Sun is also with Hangzhou Innovation Institute, Beihang University, China.

decompose the model into modules. However, since the model is not directly trained for modularization, decomposing trained models into modules has three main limitations [24, 32, 33]. First, the obtained modules may share a large portion of weights, meaning that the weights relevant to implementing different functionalities have large overlaps. When decomposing a trained model for a certain prediction task, the corresponding modules may retain a large number of weights. For instance, modules generated by [23] retain 76.21% of the weights. CNNSplitter [32] achieves better performance, but its resulting modules still retain 56.76% of the convolution kernels. Second, decomposing trained models could cause huge resource and time consumption. For instance, CNNSplitter costs at least 3 hours to decompose a simple trained model with 4,224 convolution kernels. Last but not least, the modularization process may cause non-trivial accuracy loss, e.g., the average loss of accuracy caused by CNNSplitter is 2.89%. Those limitations affect the practical usability of the paradigm of *modularizing after training*.

To overcome the above limitations, in this paper, we propose *Modularizing while Training* (MwT), a new paradigm for modularizing DNN models. It is well known that modular software development requires (1) *high cohesion*, which means keeping code that is closely related to each other in a single module, and (2) *low coupling*, which means separating unrelated code into different modules [1]. Inspired by the principle of modular software development, MwT trains models with the goal of achieving *higher cohesion* and *lower coupling* as much as possible in the *modular training* stage. In our paradigm, high cohesion means keeping the weights relevant to a certain prediction task into a small portion of weights of the whole model, while low coupling means the weights used for different prediction tasks have small overlaps. Given a DNN model that is trained with high cohesion and low coupling, the *modularizing* stage can achieve more efficient and effective decomposition than the existing approach of modularizing after training [23, 24, 32].

To realize this idea, we first define cohesion and coupling in the context of DNN modularization. MwT measures cohesion by computing the overlap between weights corresponding to the samples belonging to the same class. MwT measures the coupling by computing the overlap between weights corresponding to different modules. To train high-cohesion and low-coupling models, we design cohesion loss and coupling loss functions and integrate them with the cross-entropy loss. Then to decompose a trained model, MwT simply needs to remove the irrelevant weights to construct modules. Based on the MwT framework, we design and implement a concrete approach for CNN modularization. The reason for choosing CNN is that the CNN model is a mainstream DNN model that has been the focus of existing DNN modularization work.

We evaluate MwT using four representative CNN models on two widely-used datasets. The experimental results demonstrate that MwT can maintain sufficient model classification accuracy while effectively improving cohesion and reducing coupling. Compared to the existing state-of-the-art modularizing-after-training approach [32], the loss of accuracy caused by MwT is only 1.13 percentage points, which is 1.76 percentage points less than that of the existing approach. Notably, the kernel retention rate of modules generated by MwT is only 14.58%, with a reduction of 74.31% over the existing approach. The total time cost of training and modularizing is 108 minutes, which is half the time required by the existing approach. Moreover, module reuse leads to significantly lower inference cost than reusing the entire model.

The main contributions of this work are as follows:

- We propose a new paradigm for modularizing DNN models, called *modularizing while training*, which achieves more efficient and effective decomposition than the current paradigm of *modularizing after training*.
- We propose a framework called MwT and implement a concrete tool for CNN modularization based on MwT. We propose strategies for recognizing relevant kernels, and then design loss functions for evaluating cohesion and coupling.
- We conduct extensive experiments on two widely-used datasets using four representative CNN models. The results demonstrate that MwT can maintain the model's classification accuracy while improving cohesion and reducing coupling. Moreover, our experiments show that MwT outperforms the state-of-the-art approach in terms of both effectiveness and efficiency.

2 PRELIMINARIES

2.1 Mainstream neural networks

Neural networks (NNs) [6, 9, 14] are a type of machine learning models that comprise interconnected layers of neurons. The connections between neurons are represented by weights, which are the essential structure and learnable parameters that determine the functionality and performance of the models. The most basic NN is the fully connected neural network, in which each neuron in one layer is connected to all neurons in the next layer.

With the advancement of neural networks, various network structures have been developed for processing different types of data. Among them, CNN [16, 42, 43] is a popular type of NN specifically designed for image data and has been widely adopted in computer vision [25, 48] and various software engineering tasks [10, 17, 40, 47]. A CNN model typically consists of convolutional layers, pooling layers, and fully connected layers, with convolutional layers being the core of CNNs [14, 49]. Each convolutional layer contains numerous convolution kernels, which consists of a group of weights. Each kernel learns to recognize local features of an input tensor and outputs a feature map that reflects the degree of matching between the kernel and the input tensor.

2.2 Problem Formulation

Given a trained N-class classification model $\mathcal{M} = (\mathcal{N}, \mathcal{W})$, where $\mathcal{N}$ and $\mathcal{W}$ are the sets of neurons and weights in the model, respectively, and a training sample set of $\mathcal{M}$, denoted as $\mathcal{S} = \{\mathcal{S}_n\}_{n=1}^{N}$, where $\mathcal{S}_n$ represents the set of samples belonging to class n, the formal definition of DNN modularization is as follows:

Definition 1. ***DNN Modularization*** *aims to compute a set of modules, denoted as* $\{m_n\}_{n=1}^{N}$. *To compute* m_n, *modularization recognizes subsets* $\mathcal{N}_n \subset \mathcal{N}$ *and* $\mathcal{W}_n \subset \mathcal{W}$ *used to classify* $\mathcal{S}_n$.

The set of weights $\mathcal{W}_n$ can consist of individual weights or substructures of NNs. Specifically, since weights are fundamental structures of NNs, weight sets based on individual weights are applicable to all DNN models for modularization. In addition, weight sets based on substructures such as convolution kernels for CNNs or attention heads for Transformers are applicable to the DNN

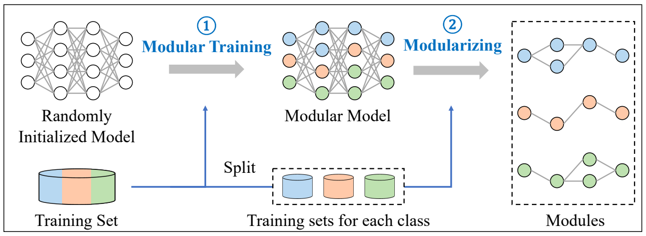

Figure 1: An overview of the proposed framework *modularizing while training.*

models constructed mainly with the corresponding NNs for modularization. To evaluate modularization, the definitions of cohesion and coupling in the context of DNN modularization are as follows:

Definition 2. ***Cohesion** of module m_n is the degree of overlap between the sets of weights $\{\mathcal{W}_n^i\}_{i=1}^{|\mathcal{S}_n|}$, where $\mathcal{W}_n^i$ represents the set of weights used to classify the i-th sample in $\mathcal{S}_n$.*

Definition 3. ***Coupling** between two modules m_n and m_k is the degree of overlap between sets of weights $\mathcal{W}_n$ and $\mathcal{W}_k$.*

In the end, the cohesion and coupling of the result of modularization are calculated as the average cohesion of all modules and the average coupling across all pairs of modules, respectively.

3 MODULARIZING WHILE TRAINING

In this section, we present the detailed methodology of Modularizing-While-Training (MwT). Specifically, Section 3.1 introduces the general framework that incorporates modularization into the model training process. A concrete approach for modularizing convolutional neural networks using MwT is then presented in Section 3.2 and Section 3.3. Additionally, Section 3.4 introduces the concept of on-demand model reuse based on MwT.

3.1 Overview

The goal of DNN modularization is to decompose a model into modules, each corresponding to a class and containing only the weights necessary for classifying samples of that class. To achieve this goal and overcome the limitations of *modularizing after training* approaches, we design a novel *modularizing while training* framework, called MwT, as shown in Figure 1. From a high-level perspective, MwT consists of two stages: *modular training* and *modularizing*.

The *modular training* stage is responsible for training a modular model that has sufficient classification ability and performs well in terms of modularity metrics (i.e., coupling and cohesion). Enhancing classification accuracy while improving model performance in coupling and cohesion during training is the core of modular training. During modular training, MwT recognizes the weights required to classify each training sample. Then, it evaluates the cohesion of the modular model by computing the overlap between weights corresponding to training samples belonging to the same class. MwT evaluates the coupling by computing the overlap between weights corresponding to training samples belonging to different classes. Based on the computation of coupling and cohesion, MwT designs coupling loss and cohesion loss functions. While minimizing the classification loss (e.g., cross-entropy loss) using gradient descent, MwT also minimizes the coupling loss and cohesion loss to optimize the performance of the modular model in terms of coupling and cohesion. Through iterative optimization, the resulting modular model has sufficient classification capability and achieves high cohesion and low coupling in terms of modularity.

The *modularizing* stage decomposes the resulting modular model into modules using the training samples of each class. Each module retains only the weights responsible for the corresponding class.

In this work, we apply MwT to CNN models. CNN is a mainstream neural network model, whose modularization has been the focus of existing works [24, 32, 33]. Following existing modularization work, we propose a concrete approach for CNN modularization based on MwT. It is worth mentioning that, inspired by CNNSplitter [32], MwT generates modules at the granularity of convolution kernels rather than individual weights. In this way, the generated module contains fewer weights than the model, and the module can perform well without the requirement of special libraries [32, 33].

3.2 Modular Training

Training a modular CNN model involves three phases: (1) recognition of relevant convolution kernels, (2) evaluation of the cohesion and coupling, and (3) optimization of the modular model's performance with regard to cohesion and coupling.

3.2.1 Recognition of relevant convolution kernels. As illustrated in Figure 2, to recognize the relevant kernels for an input sample, a *relevant kernel recognition* is appended to the original CNN model and trained jointly with the model. The relevant kernel recognition consists of *mask generators*, each of which is a fully connected (FC) layer and corresponds to a convolutional layer, thereby enabling it to recognize the kernels responsible for the input sample based on the convolutional layer's input. Taking an example of a convolutional layer in the dashed box in Figure 2, the input sample to the convolutional layer, denoted as *input*, is an image or the output of the previous convolutional layer. Suppose *input* has a dimension of (C, H, W), where C represents the number of channels, H denotes the height of input planes in pixels, and W indicates the width in pixels. First, *input* is fed simultaneously to the convolutional layer *Conv* and the mask generator. *Conv* comprises N_k kernels, which produce N_k feature maps, denoted as *FMs*. Meanwhile, the mask generator outputs a mask with a dimension of N_k based on *input*. The N_k elements of the mask correspond to the N_k convolution kernels, representing whether each kernel is relevant to *input*. The activation functions of the mask generator are $Tanh$ and $ReLU$, which map the value of its outputs to $[0, 1)$, resulting in the mask. The kernels associated with the elements in the mask having values greater than 0 are relevant. Specifically, *FMs* are multiplied by the mask, which could filter out the feature maps produced by irrelevant kernels. Based on the produced masks, MwT recognizes the relevant kernels for each sample.

Besides, to reduce the overhead for training relevant kernel recognition, before being fed to the mask generator, *input* is subject to an average pooling operation to obtain a tensor $input'$ with a smaller

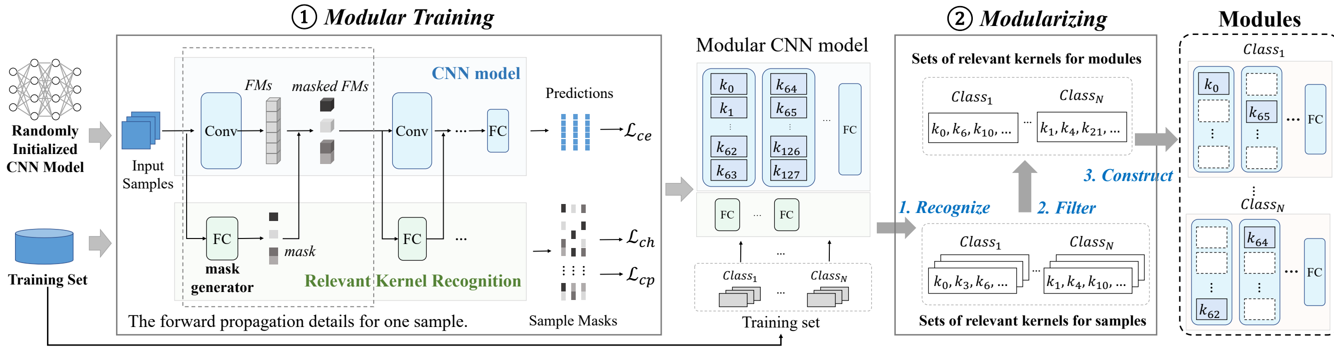

Figure 2: The workflow of MwT for CNN models.

dimension of C. Reducing the dimension of input could significantly lower the computational cost.

3.2.2 Calculation of cohesion and coupling. Using the generated sample masks as described above, we can obtain the set of relevant kernels for each sample. Specifically, for a certain class c_i, the n_i samples belonging to c_i are denoted as $\{s_i^1, s_i^2, \dots s_i^{n_i}\}$, and their corresponding kernel sets are represented as $\{sK_i^1, sK_i^2, \dots sK_i^{n_i}\}$. The kernel set mK_i constituting module m_i is calculated as $mK_i = \bigcup_{k=1}^{n_i} sK_i^k$, meaning that if one kernel is useful for any sample of class c_i, it will be retained in the corresponding module m_i. MwT evaluates the cohesion of m_i by measuring the similarity between the kernel sets sK_i^j and sK_i^k for $0 < j < k \leq n_i$. The similarity metrics are defined as the Jaccard Index (JI), which is also utilized by existing work [24, 32] to measure the similarity between modules. Specifically, the JI between sets A and B is obtained by dividing the size of the intersection of two sets by the size of the union:

$$JI(A, B) = \frac{|A \cap B|}{|A \cup B|}. \quad (1)$$

$JI(A, B) = 1$ indicates that the two sets are exactly the same; conversely, a value of 0 indicates that there is no overlap between the two sets. Based on JI, the cohesion of the module m_i responsible for the class c_i is calculated as follows:

$$Cohesion(m_i) = \frac{2}{n_i \times (n_i - 1)} \times \sum_{0<j<k\leq n_i} JI(sK_i^j, sK_i^k). \quad (2)$$

In other words, the cohesion of module m_i is calculated by the averaged JI of each pair of sK_i^j and sK_i^k. On the other hand, MwT evaluates the coupling between m_i and m_j by measuring the overlap between their relevant kernel sets mK_i and mK_j. Specifically, the coupling between m_i and m_j is calculated as follows:

$$Coupling(m_i, m_j) = JI(mK_i, mK_j). \quad (3)$$

In the end, the cohesion and coupling of the modular model are calculated as the average cohesion of all modules and the average coupling across all pairs of modules, respectively.

3.2.3 Modular training to optimize accuracy, cohesion, and coupling. To continuously improve the performance of the modular model in terms of cohesion and coupling during training, a straightforward way is to integrate the two metrics into the loss function. Then, the optimization goal is to boost the modular model's classification accuracy while improving its cohesion and reducing its coupling. Through gradient descent, the cohesion and coupling losses, as well as the cross-entropy loss, are minimized to optimize the CNN model and relevant kernel recognition. Common gradient descent methods can only perform optimization in continuous space, but convolution kernel sets are discrete, which makes it hard to use the Jaccard Index-based cohesion and coupling as the loss function. Therefore, we calculate the cohesion and coupling losses by computing the cosine similarity between the masks of samples. Specifically, given a batch of training samples, the cohesion loss $\mathcal{L}_{ch}$ and coupling loss $\mathcal{L}_{cp}$ are computed as follows:

$$\mathcal{L}_{ch} = 1 - \frac{1}{N} \times \sum_{i=1}^{N} \left(\sum_{0<j<k\leq n_i} Cos(sM_i^j, sM_i^k) \div \frac{n_i \times (n_i - 1)}{2} \right), \quad (4)$$

$$\mathcal{L}_{cp} = \frac{2}{N^2 - N} \times \sum_{0<h<i\leq N} \left(\frac{1}{n_h \times n_i} \times \sum_{j=1}^{n_h} \sum_{k=1}^{n_i} Cos(sM_h^j, sM_i^k) \right), \quad (5)$$

where N is the number of classes, n_i is the number of samples in the batch belonging to the class c_i, sM_i^j represents the mask of the j-th sample of class c_i, and $Cos(*, *)$ is the cosine similarity between two tensors. Since the range of element values in the mask is $[0, 1)$, the cosine similarity between two masks is within $[0, 1]$. Thus the ranges of both $\mathcal{L}_{ch}$ and $\mathcal{L}_{cp}$ are $[0, 1]$. With all loss functions, the objective function is defined as follows:

$$O = \mathcal{L}_{ce} + \alpha \times \mathcal{L}_{ch} + \beta \times \mathcal{L}_{cp}, \quad (6)$$

where $\mathcal{L}_{ce}$ is the cross-entropy loss, α and β are weighting factors.

To minimize O, modular training simultaneously minimizes $\mathcal{L}_{ce}$, $\mathcal{L}_{ch}$, and $\mathcal{L}_{cp}$ through mini-batch gradient descent. By minimizing $\mathcal{L}_{ce}$, the modular model learns to use the corresponding kernels (i.e., modules) to classify different classes of samples. By minimizing $\mathcal{L}_{ch}$, the modular model tends to generate identical masks for samples belonging to the same class, i.e., improving the cohesion of the modular model. By minimizing $\mathcal{L}_{cp}$, the modular model tends to generate masks with zero cosine similarity for samples belonging to different classes. Therefore, minimizing the coupling loss can reduce the coupling of the modular model. Modular training iterates specified epochs and outputs the modular model.

3.3 Modularizing

During modular training, a modular model including the CNN model and relevant kernel recognition is generated. To decompose

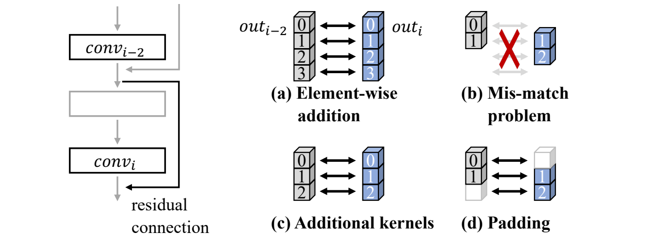

Figure 3: The illustration of residual connections and dimension mismatch problem.

the trained modular model into modules, MwT first generates masks for each module using the trained relevant kernel recognition. To generate a mask for module m_i, the basic idea is (1) for all training sample $\{s_i^1, s_i^2, \dots s_i^{n_i}\}$ belonging to class c_i, using relevant kernel recognition to generate sets of sample masks $\{sM_i^1, sM_i^2, \dots, sM_i^{n_i}\}$, and (2) calculate the module's mask as $mM_i = \{\frac{kernel\_count}{n_i} > \tau : 1, 0 \mid kernel\_count \in \sum_{j=1}^{n_i} sign(sM_i^j)\}$, meaning that if the frequency of a kernel occurring in samples' kernel sets exceeds a threshold τ, the element in the module's mask corresponding to the kernel will be marked as 1, otherwise, it will be marked as 0. The kernels corresponding to the elements of mM_i marked as 1 are relevant to m_i. MwT filters out kernels by setting a reasonable threshold τ because of the inherent randomness of neural network models. It is challenging for the mask generator to ensure identical masks for different samples of the same class. Especially when the number of training samples is large, it is inevitable that some individual relevant kernels of a sample are irrelevant to other samples. Simply regarding all relevant kernels of samples as module's kernel may result in a module containing numerous redundant kernels.

After obtaining the relevant kernels of a module, irrelevant kernels in the CNN model are removed, and the relevant kernel recognition is eliminated, resulting in a module responsible for class c_i. In simple CNN models, such as VGG [39], which consist of stacked convolutional layers, the removal of irrelevant kernels is straightforward. Specifically, when removing irrelevant kernels, the retained kernels' channels corresponding to the irrelevant kernels in the previous convolutional layer are also eliminated. However, for complex CNN models containing residual connections (such as ResNet [8]), removing irrelevant kernels may cause two layers connected with a residual connection to produce mismatched outputs. Figure 3 presents a residual network, where layers $conv_i$ and $conv_{i-2}$ have a residual connection. In the residual network, the outputs out_i and out_{i-2} of $conv_i$ and $conv_{i-2}$ are element-wise added, as shown in Figure 3-(a). As shown in Figure 3-(b), when the irrelevant kernels are removed from $conv_i$ and $conv_{i-2}$, their outputs may mismatch in the dimension of channel, resulting in failed element-wise addition. To address this problem, existing work CNNSplitter retains more kernels to keep out_i and out_{i-2} consistent. More specifically, in addition to retaining their respective relevant kernels, $conv_i$ and $conv_{i-2}$ must also retain extra irrelevant kernels to ensure that both out_i and out_{i-2} have the same number of channels, as depicted in Figure 3-(c). Nevertheless, these additional kernels result in extra convolution computation, which could lead to much overhead. To avoid this issue, we design a padding-based method to address the mismatch problem. As illustrated in Figure 3-(d), $conv_i$ and $conv_{i-2}$ each retain their relevant kernels. Before the element-wise addition of out_i and out_{i-2}, zero padding is applied to them to ensure that they have the same number of channels. Since the computation of the padding operation is extremely low, the overhead of our padding-based method is lower than that of the existing method.

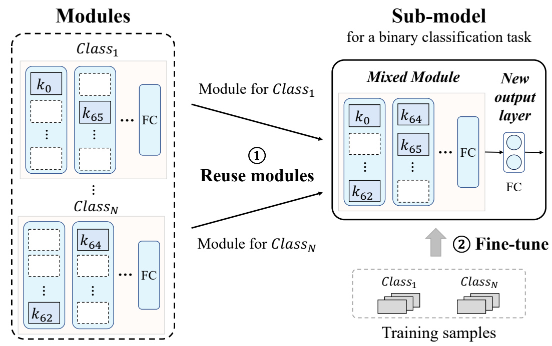

Figure 4: The illustration of on-demand model reuse.

3.4 On-demand Model Reuse

MwT can better achieve on-demand model reuse, which means dynamically selecting modules (rather than the whole model) from the model and composing them according to specific user requirements. As users usually do not need all the classification functionality, on-demand model reuse can construct a sub-model with fewer weights than the original model, resulting in lower inference costs.

Specifically, as illustrated in Figure 4, to reuse the N-class model on demand on a sub-task containing M classes (i.e., $Class_1$ and $Class_N$ in the example), the M modules responsible for these M classes are reused. When the reused modules originate from the same trained model, MwT constructs a *mixed module* based on the union of convolution kernel sets of these modules, in contrast to the existing work [23, 24] that constructs a composed model by treating each module as an individual classifier. As different modules could contain some of the same weights, a mixed module can avoid redundant retention of the same weights compared to a composed model. Since the output layer of the mixed module still corresponds to the N classes of the original task, a randomly initialized FC layer with a dimension of (N, M) is appended as a *new output layer* after the mixed module, mapping the N-dimension output to an M-dimension output. As a result, an M-class classification sub-model is constructed, which contains only the kernels responsible for the M classes. The sub-model only needs to be fine-tuned for a few epochs (e.g., 5 epochs) using samples belonging to these M classes from the training set of the original model, thus achieving comparable accuracy to the original model on the target task. Note that the fine-tuning of the sub-model does not use additional data beyond the training samples of the original model.

When reused modules come from different trained models, mixed modules can be separately constructed based on each model's modules. Subsequently, similar to existing work [24], a composed model could be constructed by combining these mixed modules and be fine-tuned. We will explore this reuse scenario in future work.

4 EXPERIMENTS

To verify the effectiveness of MwT, we present the benchmarks and experimental setup as well as the experimental results. Specifically, we evaluate MwT by answering the following research questions:

- RQ1: How effective is MwT in training and modularizing CNN models?
- RQ2: How efficient is MwT in training and modularizing CNN models?
- RQ3: How effective is MwT in reusing CNN modules?
- RQ4: How do the major hyper-parameters influence the performance of MwT?

4.1 Experimental Setup

Models. Two representative CNN models including ResNet18 [8] and VGG16 [39], and two CNN models, SimCNN and ResCNN from the baseline [32] are utilized to evaluate the effectiveness of MwT.

Datasets. Two public classification datasets are used to train and modularize the CNN models, including CIFAR10 [13] and Street View House Number (SVHN) [22], which are also used in the baseline approaches [24, 32]. The CIFAR10 [13] dataset contains natural images with resolution 32 × 32, which are drawn from 10 classes including *airplanes*, *cars*, *birds*, *cats*, *deer*, *dogs*, *frogs*, *horses*, *ships*, and *trucks*. The training and test sets contain 50,000 and 10,000 images respectively. The SVHN dataset contains colored digit images 0 to 9 with resolution 32 × 32. The training and test sets contain 604,388 and 26,032 images respectively. For both CIFAR10 and SVHN, the training set is used to train and modularize models, and the test set is used to evaluate the trained models and the resulting modules.

Baselines. (i) Standard training. Standard training optimizes CNN models using mini-batch stochastic gradient descent with cross-entropy loss. (ii) CNNSplitter [32]. CNNSplitter is the state-of-the-art approach following the paradigm of modularizing after training. In particular, same as MwT, CNNSplitter also produces modules by retaining relevant convolution kernels.

Metrics. (i) Classification accuracy (ACC), which is calculated as the percentage of correct predictions on the test set out of the total number of predictions made on the test set. (ii) Kernel retention rate (KRR) [32], which is calculated as the average number of kernels retained in modules divided by the total number of kernels in the model. (iii) Cohesion, which is the average cohesion of all modules (Eq. 2). (iv) Coupling, which is the average coupling across all pairs of modules (Eq. 3).

Hyper-parameters. In standard training, VGG16 and ResNet18 are trained for 200 epochs using a mini-batch size of 128. We set the learning rate to 0.05 and Nesterov's momentum to 0.9. Additionally, we use common data augmentation [38], such as random cropping and flipping. As for SimCNN and ResCNN, pre-trained models published by the baseline [32] are utilized directly. For MwT, the hyperparameters involved in modular training include all the hyperparameters in standard training, and their settings are the same as those in standard training. In addition, the settings of weighting factors α and β are shown in Table 1, and the threshold τ is set to 0.9. The effect of α, β, and τ will be investigated in RQ4.

All the experiments are conducted on Ubuntu 20.04 server with 64 cores of 2.3GHz CPU, 128GB RAM, and NVIDIA Ampere A100 GPUs with 40 GB memory.

Table 1: The settings of weighting factors α and β.

	VGG16		ResNet18		SimCNN		ResCNN	
	CIFAR10	SVHN	CIFAR10	SVHN	CIFAR10	SVHN	CIFAR10	SVHN
α	0.5	1.0	0.5	1.3	0.5	0.5	1.0	1.0
β	1.5	2.0	1.5	1.5	3.5	2.0	2.2	1.8

Table 2: The comparison of standard training and the proposed MwT.

Model	Dataset	#Kernels	Standard Model ACC	Modular Model ACC	Modules		
					KRR	Cohesion	Coupling
VGG16	CIFAR10	4224	92.29	90.86	17.28	0.9758	0.1751
	SVHN		95.84	94.74	14.15	0.9687	0.2246
ResNet18	CIFAR10	3904	93.39	91.59	24.74	0.9437	0.2412
	SVHN		95.84	95.95	25.89	0.9663	0.3115
Average		**4064**	**94.34**	**93.29**	**20.52**	**0.9636**	**0.2381**

4.2 Experimental Results

RQ1: How effective is MwT in training and modularizing CNN models?

To evaluate the effectiveness of MwT in training and modularizing CNN models, we evaluate the classification accuracy of modular models and measure the kernel retention rate (KRR), cohesion, and coupling of trained modules. Table 2 presents the results of standard training, modular training, and modularizing on four models. Column "#Kernels" shows the number of convolution kernels for VGG16 and ResNet18, and the fourth and fifth columns show the accuracy of standard and modular models on testing set, respectively. On average, the standard and modular models achieved accuracy of 94.34% and 93.29%, respectively. The average loss of accuracy is 1.05 percentage points, demonstrating that modular training does not cause much loss of performance in terms of accuracy. The last three columns display KRR, cohesion and coupling degree of obtained modules by modularizing with a threshold of 0.9, respectively. On average, the modules of all four models retain 20.52% of convolution kernels. The averaged cohesion degree is 0.9636, indicating that a module uses essentially the same convolution kernels to predict samples belonging to the corresponding class. The averaged coupling between modules is 0.2381, indicating that the convolution kernels of different modules do not overlap much.

More specifically, Figure 5 depicts the convergence process of training for the VGG16 model on the CIFAR10 dataset, where the dashed orange lines represent the standard training and solid blue lines represent modular training. During modular training, the coupling and cohesion of the modular model in each epoch are calculated as the average values across all iterations within that epoch. The coupling and cohesion in each iteration are calculated using Eq. 3 and Eq. 2. Note that, the calculation of coupling and cohesion during modular training does not involve filtering, as the batch of samples (consisting of 128 samples) in each iteration may not accurately estimate the occurrence frequency of a kernel in the entire training dataset. Regarding the coupling and cohesion of the standard model during standard training, relevant kernels for a sample are identified based on whether their outputs contain non-zero elements. If a kernel's output contains non-zero elements,

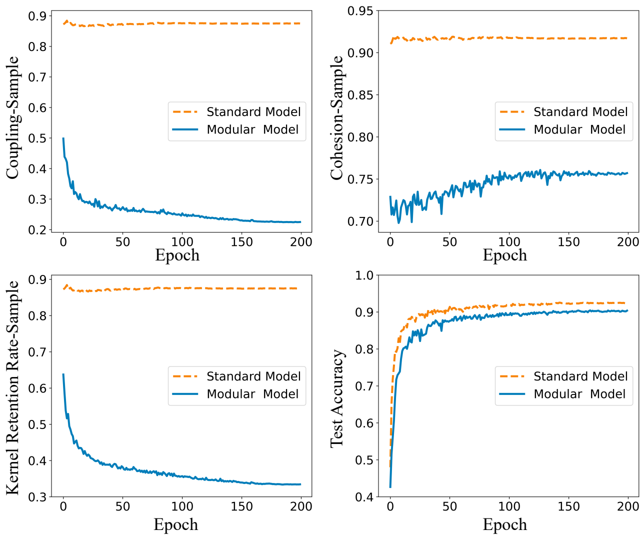

Figure 5: The convergence process of Modular Training on VGG16-CIFAR10.

it is considered relevant because its output may affect the classification of the sample. Otherwise, the kernel is considered irrelevant. Overall, the process of identifying relevant kernels during standard training is similar to that during modular training. In both cases, the determination is based on whether the output of a kernel will impact the classification of the sample.

The top left sub-figure plots the convergence trend of coupling degree. Regarding the modular model, the coupling decreases rapidly in the first 50 epochs and gradually converges. In contrast, the coupling degree of the standard model is consistently high. This indicates that the proposed loss function can effectively reduce the coupling degree between kernels that are relevant to different classes during the training process. The top right sub-figure shows the convergence process of cohesion degree. The cohesion degrees of both modular and standard models are maintained at a relatively high level. However, this does not imply that the designed cohesion loss function is unimportant. In our experiments, when eliminating the cohesion loss, the cohesion degree of the modular model would quickly drop as the coupling degree decreases. The reduction of the cohesion degree would result in the increase of module size, thus increasing the reuse overhead of the modules. Compared to the standard training that maintains both high cohesion and coupling degree, MwT is able to maintain a high cohesion degree while reducing the coupling degree. When predicting samples belonging to different classes, the standard model always uses most of the convolution kernels, while the modular model uses only a corresponding portion of the convolution kernels.

The bottom left sub-figure gives the averaged rate of convolution kernels used for predicting samples. The standard model always uses the majority of convolution kernels to make predictions. In contrast, the modular model tends to use as few convolution kernels as possible, thus reducing the coupling degree. The fewer convolution kernels used for predicting a sample, the less likely that the convolution kernels for different prediction tasks overlap. Similar

Table 3: The comparison of CNNSplitter and MwT. "S.M.", "Coup.", and "Cohe." indicate "Standard Model ACC", "Coupling", and "Cohesion", respectively.

Model	Dataset	S.M. ACC	CNNSplitter				MwT			
			ACC	Coup.	Cohe.	KRR	ACC	Coup.	Cohe.	KRR
SimCNN	CIFAR10	89.77	86.07	0.5277	0.9326	61.96	88.84	0.1372	0.8682	11.58
	SVHN	95.41	93.85	0.6161	0.9619	52.79	93.56	0.1434	0.9580	11.85
ResCNN	CIFAR10	90.41	85.64	0.5648	0.8462	58.26	89.82	0.2781	0.9601	21.52
	SVHN	95.06	93.52	0.6046	0.8828	54.03	93.88	0.3306	0.9731	13.37
Average		**92.66**	**89.77**	**0.5783**	**0.9059**	**56.76**	**91.53**	**0.2223**	**0.9399**	**14.58**

to the convergence trend of coupling degree, the KRR decreases rapidly in the first 50 epochs and gradually converges.

Despite the differences in cohesion and coupling degrees, the modular model achieves competitive classification accuracy with the standard model. The convergence trends of the accuracy of the modular and standard models on the test dataset are similar, with final classification accuracy of 90.86% and 92.29%, respectively. Modular training results in a small accuracy loss of 1.43 percentage points.

Moreover, we also compare MwT with the state-of-the-art modularization approach CNNSplitter [32]. Specifically, we directly compared these two approaches on the models and modules published by CNNSplitter. Table 3 shows the performance on four metrics, including accuracy, kernel retention rate, cohesion, and coupling. Note that, CNNSplitter's ACC represents the test accuracy of the "composed" model constructed by combining all the decomposed modules. The composed model is similar to the modular model. On average, the accuracy of CNNSplitter's composed model and MwT's modular model are 89.77% and 91.53%, respectively, with the latter achieving an improvement of 1.76 percentage points. Compared with the accuracy of the standard model of 92.66%, the accuracy losses caused by CNNSplitter and MwT are 2.89 percentage points and 1.13 percentage points, respectively, indicating that MwT causes much less loss of accuracy. To compare cohesion and coupling, we calculate the cohesion degree of CNNSplitter's modules in the same way as standard model. Since a module generated by CNNSplitter retains only a part of the convolution kernels of the standard model, the calculation of the coupling degree of CNNSplitter's modules is the same as that of MwT, which is obtained by calculating the Jaccard similarity between modules. Overall, compared with CNNSplitter, MwT achieves better results in both coupling and cohesion metrics, with an average improvement of 0.3560 and 0.0340, respectively. Regarding convolution kernel retention rate, MwT is significantly better than CNNSplitter (56.76% vs 14.58%), with a reduction of 74.31% in KRR.

Similar to MwT, CNNSplitter uses Jaccard similarity to increase the differences between modules and ensure modules retain only relevant convolution kernels. However, CNNSplitter is an approach that modularizes after training. Standard training methods do not consider the differences (i.e., coupling) between modules, which results in small differences between modules. Therefore, modularizing-after-training approaches are inherently limited in terms of coupling and KRR. In contrast, MwT is a modularizing-while-training approach, which considers reducing the coupling between modules

Table 4: The comparison of MwT and modularizing-after-training in time costs of training and modularizing. "Mo." indicates "Modularizing".

Model	Dataset	MwT			Modularizing-after-Training		
		Modular Training	Mo.	Total	Standard Training	CNNSplitter	Total
SimCNN	CIFAR10	24s/e x 200e	14s	80m	12s/e x 200e	83s/e x 123e	210m
	SVHN	36s/e x 200e	21s	120m	17s/e x 200e	95s/e x 79e	182m
ResCNN	CIFAR10	28s/e x 200e	15s	94m	14s/e x 200e	80s/e x 185e	293m
	SVHN	41s/e x 200e	21s	137m	20s/e x 200e	93s/e x 107e	233m
Average		**107m**	**18s**	**108m**	**53m**	**177m**	**229m**

during the modular training process, leading to better results in terms of coupling and KRR.

The coupling, cohesion, and KRR of the modules generated by MwT are 0.2223, 0.9399, and 14.58%, respectively, which are decreased by 0.3560, increased by 0.0340, and reduced by 74.31%, respectively, compared to the state-of-the-art.

RQ2: How efficient is MwT in training and modularizing CNN models?

One advantage of MwT over Modularizing-after-training is to reduce the runtime overhead. In this experiment, we take CNNSplitter as the example of modularizing-after-training. The runtime overhead of MwT includes modular training time and modularizing time, and the runtime overhead of CNNSplitter includes standard training time and modularizing time. Table 4 shows the runtime overhead for MwT and modularizing-after-training on SimCNN-CIFAR10, SimCNN-SVHN, ResCNN-CIFAR10, and ResCNN-SVHN. For instance, in the case of SimCNN-CIFAR10, the modular training time of MwT amounts to $24s/e \times 200e$, indicating that the model is trained for 200 epochs, with each epoch taking 24 seconds. The modularizing time of MwT is mainly attributed to the generation of masks through forward propagation, which takes 14 seconds. Consequently, the runtime overhead of MwT amounts to 80 minutes. Regarding CNNSplitter, the standard training time amounts to $12s/e \times 200e$. The modularizing time of CNNSplitter amounts to $83s/e \times 123e$, i.e., the modularization iterates 123 epochs, with each epoch taking 83 seconds. As a result, the runtime overhead of modularizing-after-training amounts to 210 minutes. On average, MwT's runtime overhead is 108 minutes, while modularizing-after-training's runtime overhead is 229 minutes.

Additionally, we found that the average modular training time of MwT (i.e., 107 minutes) is almost twice the standard training time (i.e., 53 minutes). This is mainly due to the introduction of additional parameters (i.e., mask generators) and loss functions (i.e., cohesion loss and coupling loss). Moreover, in our experiments, we noticed that the GPU utilization for modular training is less than 70%, while the GPU utilization for standard training is nearly 100%. This suggests that optimizing the implementation of MwT could further enhance the GPU utilization, thereby reducing the time overhead of modular training. We leave this as future work.

Table 5: The convolution kernel retention rate of MwT in reusing CNN modules. All results in %.

Target Task	VGG16		ResNet18		Average
	CIFAR10	SVHN	CIFAR10	SVHN	
2-class	30.34	27.38	35.94	42.44	34.03
3-class	43.39	38.65	50.51	68.18	50.18
4-class	52.93	47.00	65.12	70.38	58.86
5-class	58.29	53.15	71.41	72.28	63.78
6-class	63.75	57.31	74.58	74.01	67.41
7-class	72.57	61.27	79.79	78.36	73.00
8-class	79.88	63.65	82.35	79.56	76.36
9-class	88.34	66.25	86.87	80.60	80.52
10-class	94.11	68.61	88.75	81.52	83.25

On average, the time overhead of MwT is 108 minutes, only half of the time for modularizing-after-training.

RQ3: How effective is MwT in reusing CNN modules?

In this RQ, we investigate the effectiveness of MwT in on-demand model reuse, which is one of the key benefits of model modularization. Specifically, we reuse both the module and the standard model on sub-tasks derived from the original classification task that a standard model solves. There are two 10-class classification tasks corresponding to the CIFAR10 and SVHN datasets. Each task can be divided into sub-tasks with a number of categories ranging from 2 to 10. An M-class sub-task consists of M categories from the 10-class classification task, resulting in a total of C_{10}^{M} M-class sub-tasks. For instance, a 5-class sub-task has 252 (C_{10}^{5}) possibilities in total. We randomly select 10 sub-tasks for each 2-class to 9-class classification sub-task. For each M-class sub-task, we reuse the standard model and corresponding modules (see Sec. 3.4) that can classify the M categories, then compare the number of convolution kernels, computational cost, and accuracy of model and modules.

Table 5 presents the KRR of the modules on different classification sub-tasks. For instance, as shown in the third row, for a 2-class classification sub-task of CIFAR10, the corresponding modules of VGG16-CIFAR10 only retain 30.34% of the model's convolution kernels. On average, for 2-class classification sub-tasks, the corresponding modules retain only 34.03% of the kernels. As the number of categories of sub-tasks increases, the number of kernels retained by the corresponding modules also increases. However, we found that even when the number of categories increases to the maximum (i.e., the sub-task is the original task), the corresponding modules contain fewer kernels than the model, with an average retention rate of 83.25%. The reason is that the *modularizing* stage filters out unimportant convolution kernels (see Sec. 3.3).

Lower KRR means that on-demand model reuse incurs a lower computational cost. To compare the computational cost directly, we follow the baseline [32] and use the open-source tool fvcore [4] to calculate the number of floating point operations (FLOPs) required by the model and module to make predictions. The VGG16 and ResNet18 models require 315.11 million and 558.50 million FLOPs, respectively. Table 6 displays the number of FLOPs (million) required by modules, as well as the percentage reduction (%) compared to reusing the model. For instance, the VGG16-CIFAR10 modules for 2-class sub-tasks require 70.97 million FLOPs, resulting in a

Table 6: The computational costs of MwT in reusing CNN modules.

Target Task	Metric	VGG16		ResNet18		Average
		CIFAR10	SVHN	CIFAR10	SVHN	
2-class	FLOPs (M)	70.97	31.85	129.76	136.15	92.18
	Reduction (%)	77.48	89.89	76.77	75.62	79.94
3-class	FLOPs (M)	121.21	52.93	189.50	329.49	173.28
	Reduction (%)	61.54	83.20	66.07	41.00	62.95
4-class	FLOPs (M)	166.35	73.31	336.49	336.41	228.14
	Reduction (%)	47.21	76.73	39.75	39.77	50.87
5-class	FLOPs (M)	179.93	85.82	360.62	344.77	242.79
	Reduction (%)	42.90	72.77	35.43	38.27	47.34
6-class	FLOPs (M)	198.71	91.03	376.75	352.28	254.69
	Reduction (%)	36.94	71.11	32.54	36.93	44.38
7-class	FLOPs (M)	230.41	99.47	398.82	372.42	275.28
	Reduction (%)	26.88	68.43	28.59	33.32	39.31
8-class	FLOPs (M)	249.33	103.30	407.57	377.54	284.44
	Reduction (%)	20.87	67.22	27.03	32.40	36.88
9-class	FLOPs (M)	279.40	107.36	432.79	382.93	300.62
	Reduction (%)	11.33	65.93	22.51	31.44	32.80
10-class	FLOPs (M)	293.88	111.10	442.67	388.09	308.94
	Reduction (%)	6.74	64.74	20.74	30.51	30.68

cost reduction of 77.48% ($(1-70.97/315.11)\times 100$) compared to the original VGG16 model. Reusing the modules of ResNet18-CIFAR10 for 2-class sub-tasks requires 129.76 million FLOPs, resulting in a 76.77% cost reduction compared to reusing the model. As shown in the last column, on average, reusing modules reduces the computational cost by 79.94% for 2-class classification sub-tasks. Overall, the experimental results demonstrate that on-demand model reuse significantly reduces the reuse cost. In particular, the average computational cost can be reduced by more than 50% when the number of categories in a sub-task is less than half of the original task.

In practice, on-demand model reuse must balance the need to reduce reuse overhead and the need to maintain classification accuracy. Table 7 presents the accuracy of both the standard model and modules on the sub-tasks (i.e., *M-ACC* and *m-ACC*). Meanwhile, row *loss* presents the accuracy loss caused by module reuse, which is the difference in accuracy between the module and the model. Evaluation results show that module reuse causes more accuracy loss in sub-tasks with over half of the original task's categories, e.g., the modules for 9-class sub-tasks result in an accuracy loss of approximately 1 percentage point. In contrast, sub-tasks with less than half categories have a smaller accuracy loss (<1 percentage point). The reason could be that as the category number increases, the classification task becomes more challenging, and it requires more classifier's parameters. Furthermore, the table indicates that the accuracy loss is higher for CIFAR10 sub-tasks compared to SVHN sub-tasks, possibly due to the greater complexity of the CIFAR10 task, which requires a classifier to be equipped with more parameters. Overall, on-demand reuse of modules causes a negligible average accuracy loss of only 0.8 percentage points across all cases, demonstrating that MwT can strike a balance between classification accuracy and computational cost.

We further compare MwT with CNNSplitter in terms of on-demand model reuse. Similar to MwT, which uses masks to represent modules, CNNSplitter uses a binary bit string to represent a module, where each bit represents whether the corresponding convolution kernel is retained or not. By retaining kernels in the model based on bit vectors, the corresponding module can be constructed. Table 8 compares the results of KRR for CNNSplitter-based and MwT-based module reuse. The results show that MwT achieves lower KRR than CNNSplitter on all sub-tasks. For example, in the

Table 7: The test accuracy results of MwT in reusing CNN modules. All results in %.

Target Task	Metric	VGG16		ResNet18		Average
		CIFAR10	SVHN	CIFAR10	SVHN	
2-class	M-ACC	99.35	98.89	99.40	99.34	99.25
	m-ACC	99.20	99.17	99.30	99.33	99.25
	loss	0.15	-0.28	0.10	0.01	0.00
3-class	M-ACC	98.23	98.48	98.23	98.91	98.46
	m-ACC	97.40	98.58	97.23	98.82	98.01
	loss	0.83	-0.10	1.00	0.09	0.45
4-class	M-ACC	96.70	97.35	96.63	97.96	97.16
	m-ACC	94.98	97.28	95.50	97.64	96.35
	loss	1.72	0.07	1.13	0.32	0.81
5-class	M-ACC	95.22	97.01	95.66	97.47	96.34
	m-ACC	93.82	97.01	94.34	97.55	95.68
	loss	1.40	0.00	1.32	-0.08	0.66
6-class	M-ACC	92.53	96.73	93.77	96.98	95.00
	m-ACC	91.05	96.44	91.27	96.92	93.92
	loss	1.48	0.29	2.50	0.06	1.08
7-class	M-ACC	92.43	96.44	93.60	96.74	94.80
	m-ACC	90.90	95.78	91.86	96.45	93.75
	loss	1.53	0.66	1.74	0.29	1.06
8-class	M-ACC	92.28	96.18	93.48	96.32	94.57
	m-ACC	91.03	95.52	91.71	96.25	93.63
	loss	1.25	0.66	1.77	0.07	0.94
9-class	M-ACC	92.36	96.04	93.52	96.09	94.50
	m-ACC	90.89	95.20	91.48	96.00	93.39
	loss	1.47	0.84	2.04	0.09	1.11
10-class	M-ACC	92.29	95.84	93.39	95.84	94.34
	m-ACC	90.86	94.74	91.59	95.95	93.29
	loss	1.43	1.10	1.80	-0.11	1.06

Table 8: The comparison of MwT and CNNSplitter in reusing CNN modules in terms of KRR. All results in %.

Target Task	Approach	SimCNN		ResCNN		Average
		CIFAR10	SVHN	CIFAR10	SVHN	
2-class	CNNSplitter	84.61	76.04	79.57	78.26	79.62
	MwT	20.34	34.13	28.69	19.98	25.79
3-class	CNNSplitter	93.09	86.88	89.79	89.86	89.91
	MwT	30.77	43.10	39.28	42.33	38.87
4-class	CNNSplitter	97.09	93.42	95.22	94.73	95.12
	MwT	33.80	51.49	53.46	45.64	46.10
5-class	CNNSplitter	98.63	96.64	97.69	97.50	97.62
	MwT	37.67	56.97	56.30	47.18	49.53
6-class	CNNSplitter	99.34	98.37	98.79	98.72	98.81
	MwT	41.25	64.69	58.62	49.34	53.48
7-class	CNNSplitter	99.74	99.05	99.32	99.51	99.41
	MwT	52.00	68.75	66.45	51.04	59.56
8-class	CNNSplitter	99.88	99.43	99.77	99.81	99.72
	MwT	57.14	72.55	68.37	52.37	62.61
9-class	CNNSplitter	99.95	99.69	99.88	99.86	99.85
	MwT	67.52	75.94	72.44	53.72	67.41
10-class	CNNSplitter	99.98	99.76	99.91	99.88	99.88
	MwT	72.40	78.82	74.22	54.71	70.04

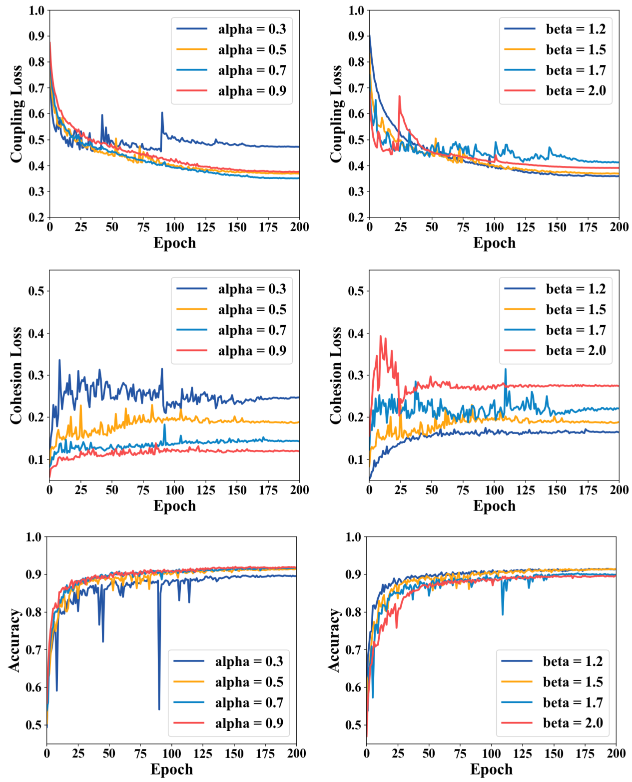

Figure 6: The effect of α and β on modular training.

2-class classification sub-task of CIFAR10, the KRR of the SimCNN-CIFAR10 modules constructed by MwT and CNNSplitter are 20.34% and 84.61%, respectively. The former contains significantly fewer kernels than the latter, resulting in a reduction of 75.96%. On average, the modules constructed by MwT retain 29.88% (the 10-class sub-task) to 75.96% (the 2-class sub-task) kernels less than the modules constructed by CNNSplitter. Moreover, it is worth mentioning that even for 2-class classification sub-tasks, the modules constructed by CNNSplitter retain most of the convolution kernels, with an average of 79.62%, while the modules constructed by MwT only retain 25.79% of the convolution kernels. This is mainly because MwT is a modularizing-while-training approach, while CNNSplitter relies on modularizing-after-training paradigm. As a result, when reusing modules for M-class classification sub-tasks, the modules constructed by MwT retain much fewer convolution kernels.

> Based on MwT, reusing modules on-demand can significantly reduce the computational cost of reuse by up to 79.94% with a negligible average accuracy loss of only 0.8 percentage points.

RQ4: How do the major hyper-parameters influence the performance of MwT?

In this research question, we investigate the effect of major parameters on MwT, including α and β (the weighting factors in Eq. 6), as well as τ (the threshold value, as described in Sec. 3.3). Due to the page limit, we only present and discuss the results of ResNet18 on CIFAR10. Nonetheless, we conducted experiments on the remaining three models, and the results are similar, as detailed on the project webpage [31].

The values of α and β will directly influence the modular training process. The left three sub-figures in Figure 6 illustrate the modular training convergence of coupling loss, cohesion loss, and accuracy on ResNet18-CIFAR10 with the values of α as 0.3, 0.5, 0.7 and 0.9. After 200 epochs, as the value of α increases, the cohesion loss gradually decreases. Moreover, coupling loss and accuracy exhibit insensitivity to changes in α when the value of α lies within a reasonable range (e.g., 0.5 to 0.9). Nonetheless, too small values of α may affect the modular training process. For instance, when α=0.3, the coupling loss is greater than that of α=0.9, and the oscillation of accuracy is more prominent. Additionally, the right three sub-figures in Figure 6 depict the convergence of the three metrics with the values of β as 1.2, 1.5, 1.7 and 2.0. We observe that changes in β do not significantly affect coupling loss and accuracy. Nevertheless, cohesion loss increases notably with an increase in β. Overall, under different parameter settings, the performance of MwT is predictable within certain ranges, as the coupling loss and accuracy are not affected significantly, and the trend of change in cohesion loss is obvious. The results also show that our default settings (i.e., α=0.5 and β=1.5) are appropriate.

The value of τ directly affects the results of modularizing and module reuse. As the threshold increases from 0.1 to 0.9, the KRR of the modules gradually decreases, from 37.36% to 24.74%, the cohesion increases from 0.8572 to 0.9437, and the coupling decreases from 0.3594 to 0.2412. Regarding the effect on module reuse, as the threshold increases, the KRR of the module decreases, from 72.57% to 50.51%. Nonetheless, the decrease in KRR has a negligible impact on the accuracy of the module, which only drops from 97.77% to 97.23%. The details are available on the project webpage [31].

> The performance of MwT is predictable within certain ranges in terms of cohesion, coupling, and accuracy under different parameter settings, making it easy to configure to various models.

5 DISCUSSION

5.1 The Generality of MwT

We argue that MwT is generalizable to diverse DNN models for the following reasons. First, the calculation of the coupling and cohesion based on sets of relevant weights is general for DNN models. Sets of weights can be constructed at the granularity of individual weights for any neural network, and at the granularity of sub-structures for neural networks with sub-structures. Regarding the recognition of relevant weights, existing works have explored methods such as neuron activations [23] and weight coverage frequency [52] to identify relevant weights for different types of neural network models. Consequently, sets of relevant weights can be constructed for any neural network model to evaluate cohesion and coupling. In addition, modular training adds cohesion and coupling loss functions on top of standard training, which is a common way to improve the performance of models in certain aspects by adding new loss functions.

More specifically, a concrete approach generalizable for DNN models could be designed at the granularity of individual weights.

The elements in a mask represent the relevance of the corresponding weights to a class. The feasibility of this idea could be supported to some extent by the existing work [33]. Moreover, for neural networks with substructures, a specialized approach could be designed at the granularity of substructures. In addition to CNNs, taking Transformers as an example, the elements of a mask can represent the relevance of corresponding attention heads and word embeddings to a class. Some related work [21, 37] could support the feasibility of the idea to some extent.

5.2 Threats to Validity

External validity: Threats to external validity relate to the generalizability of our results. In this paper, we have only evaluated MwT on CNNs, and the effectiveness on other types of DNNs, such as LSTM and Transformer, remains to be evaluated. However, as discussed above, MwT is considered to be general, and we will further investigate it in our future work. Moreover, MwT is not validated on larger scale datasets such as ImageNet [2], due to the huge resource and time consumption for training models on large-scale datasets. Additionally, our experiments did not consider situations where the original models are not highly accurate. We leave these as future work.

Internal validity: An internal threat comes from the choice of models and datasets. To mitigate this threat, we use CIFAR-10 and SVHN datasets as well as VGG16 and ResNet18 models from PyTorch [28], which are well organized and widely used.

Construct validity: A threat relates to the suitability of our evaluation metrics. The concepts of cohesion and coupling in the context of DNN modularization are first proposed in this paper, thus evaluating cohesion and coupling of DNN models remains an open problem. Other metrics for calculating overlap may also be suitable for measuring cohesion and coupling; however, Jaccard Index is a representative metric and has been widely used in existing work to measure differences between modules [23, 24, 32]

6 RELATED WORK

6.1 DNN Modularization

Existing DNN modularization studies [23, 24, 32] and other related efforts [33, 51, 52] identify neurons or weights in the pre-trained model that is relevant to the target class and retain only these relevant neurons or weights to construct a module responsible for the target class. These works can be classified into two categories based on their identification of relevant neurons and weights: *neuron activation-based* [23, 24, 51, 52] and *search-based* [32, 33] modularization approaches. The neuron activation-based approach determines whether a neuron is activated by the samples belonging to the target class based on whether its output is greater than zero, and further measures the relevance of the neuron or its associated weights to the target class. For instance, ReMos [52] measures relevance of weights to the target class mainly by computing neuron coverage frequency [29] and weight coverage frequency, and the weights with relevance above a threshold are considered relevant.

The search-based approach measures the relevance of weights retained in a candidate module to the target class based on the recognition ability of the candidate for the target class and the candidate's size or difference. For instance, CNNSplitter [32] employs genetic algorithm to search for relevant convolution kernels. It assesses the relevance of kernels in a candidate module by evaluating the classification accuracy of the candidate for the target class and measuring the difference between candidates with Jaccard distance. The weights contained in the resulting candidate with the highest classification accuracy and difference are considered relevant.

Unlike all existing works, modularizing while training proposed in this paper is a new paradigm. By considering the cohesion and coupling in the training process, MwT can overcome the limitations of the paradigm of modularizing after training and achieve more efficient and effective modularization than existing works.

6.2 DNN Pruning

DNN pruning techniques, such as iterative magnitude pruning [5, 7, 36], remove some of the weights that are not important for the whole task to generate a smaller model, thus reducing the resources and time required for inference on the whole task. In contrast, our work removes some of the convolution kernels that are irrelevant to the sub-task to decompose the model into modules, thus facilitating module reuse. The above difference in objectives further leads to a difference in loss functions. The loss functions for DNN pruning techniques usually involve the number of weights in the whole model and the magnitude of the weights. However, in our work, the cohesion loss and coupling loss functions involve the overlap between the weights corresponding to different samples.

7 CONCLUSION

In this work, we propose MwT, a novel paradigm for realizing modularization of DNN models to improve their reusability. To overcome the limitations of the existing methods that achieve modularization on trained models, we take a different approach that incorporates the modularization into the model training process. In doing so, our MwT integrates the cohesion loss and the coupling loss with the normal cross-entropy training loss, which drives the training process to optimize the modular characteristics as well as the accuracy at the same time. Specifically, we implement MwT on CNN models. The experimental results demonstrate that MwT substantially reduces the size of the resultant modules and the time cost of modularization while incurring less performance loss than the state-of-the-art approach [32]. For instance, the kernel retention rate of modules generated by MwT is only 14.58%, with a reduction of 74.31% over the state of the art. Furthermore, the time cost required for training and modularizing is 108 minutes, half of the time required by the state-of-the-art approach.

In the future, we plan to extend MwT to other types of DNNs, such as LSTM and Transformer. Additionally, we will explore various scenarios of model reuse, including the reuse of modules from different pretrained models.

Data Availability: Our source code and experimental data are available at: **https://github.com/qibinhang/MwT**.

ACKNOWLEDGMENTS

This work was supported partly by National Natural Science Foundation of China Under Grant Nos (61972013 and 61972013) and partly by Guangxi Collaborative Innovation Center of Multi-source Information Integration and Intelligent Processing .

REFERENCES

[1] Grady Booch, Robert A Maksimchuk, Michael W Engle, Bobbi J Young, Jim Connallen, and Kelli A Houston. 2008. Object-oriented analysis and design with applications. *ACM SIGSOFT software engineering notes* 33, 5 (2008), 29–29.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.

[3] Hugging Face. 2023. Hugging Face. https://huggingface.co.

[4] FAIR. 2023. fvcore. https://github.com/facebookresearch/fvcore.

[5] Jonathan Frankle and Michael Carbin. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations*.

[6] Alex Graves and Navdeep Jaitly. 2014. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*. PMLR, 1764–1772.

[7] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* 28 (2015).

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[9] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[10] Xuan Huo, Ferdian Thung, Ming Li, David Lo, and Shu-Ting Shi. 2019. Deep transfer bug localization. *IEEE Transactions on software engineering* 47, 7 (2019), 1368–1380.

[11] Ivar Jacobson, Martin Griss, and Patrik Jonsson. 1997. *Software Reuse: Architecture, Process and Organization for Business Success*. ACM Press/Addison-Wesley Publishing Co., USA.

[12] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. 2018. Model-reuse attacks on deep learning systems. In *ACM SIGSAC conference on computer and communications security*. 349–363.

[13] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* 25 (2012), 1097–1105.

[15] Charles W Krueger. 1992. Software reuse. *ACM Computing Surveys (CSUR)* 24, 2 (1992), 131–183.

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of IEEE* 86, 11 (1998), 2278–2324.

[17] Jaehyung Lee, Kisun Han, and Hwanjo Yu. 2022. A Light Bug Triage Framework for Applying Large Pre-trained Language Model. In *37th IEEE/ACM International Conference on Automated Software Engineering*. 1–11.

[18] Yuanchun Li, Ziqi Zhang, Bingyan Liu, Ziyue Yang, and Yunxin Liu. 2021. ModelDiff: Testing-Based DNN Similarity Comparison for Model Reuse Detection. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 139–151.

[19] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *Research in Attacks, Intrusions, and Defenses: 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings 21*. Springer, 273–294.

[20] Linghan Meng, Yanhui Li, Lin Chen, Zhi Wang, Di Wu, Yuming Zhou, and Baowen Xu. 2021. Measuring Discrimination to Boost Comparative Testing for Multiple Deep Learning Models. In *43rd International Conference on Software Engineering*. IEEE, 385–396.

[21] Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems* 32 (2019).

[22] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. (2011).

[23] Rangeet Pan and Hridesh Rajan. 2020. On decomposing a deep neural network into modules. In *28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 889–900.

[24] Rangeet Pan and Hridesh Rajan. 2022. Decomposing Convolutional Neural Networks into Reusable and Replaceable Modules. In *44th International Conference on Software Engineering*. 524–535.

[25] Xuran Pan, Chunjiang Ge, Rui Lu, Shiji Song, Guanfu Chen, Zeyi Huang, and Gao Huang. 2022. On the integration of self-attention and convolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 815–825.

[26] David Lorge Parnas. 1972. On the criteria to be used in decomposing systems into modules. *Commun. ACM* 15, 12 (1972), 1053–1058.

[27] David Lorge Parnas. 1976. On the design and development of program families. *IEEE Transactions on software engineering* 1 (1976), 1–9.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[29] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *26th Symposium on Operating Systems Principles*. 1–18.

[30] Jeffrey S. Poulin. 1996. *Measuring Software Reuse: Principles, Practices, and Economic Models*. Addison-Wesley Longman Publishing Co., Inc., USA.

[31] Binhang Qi. 2023. MwT. https://github.com/qibinhang/MwT.

[32] Binhang Qi, Hailong Sun, Xiang Gao, and Hongyu Zhang. 2022. Patching Weak Convolutional Neural Network Models through Modularization and Composition. In *37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[33] Binhang Qi, Hailong Sun, Xiang Gao, and Hongyu Zhang. 2023. Reusing Deep Neural Network Models through Model Re-engineering. In *International Conference on Software Engineering (ICSE) 2023*.

[34] Binhang Qi, Hailong Sun, Wei Yuan, Hongyu Zhang, and Xiangxin Meng. 2021. DreamLoc: A deep relevance matching-based framework for bug localization. *IEEE Transactions on Reliability* 71, 1 (2021), 235–249.

[35] Xiaoning Ren, Yun Lin, Yinxing Xue, Ruofan Liu, Jun Sun, Zhiyong Feng, and Jin Song Dong. 2023. DeepArc: Modularizing Neural Networks for the Model Maintenance. In *Proceedings of the 45th International Conference on Software Engineering*. 1008–1019.

[36] Jonathan S Rosenfeld, Jonathan Frankle, Michael Carbin, and Nir Shavit. 2021. On the predictability of pruning across scales. In *International Conference on Machine Learning*. 9075–9083.

[37] Jieke Shi, Zhou Yang, Bowen Xu, Hong Jin Kang, and David Lo. 2022. Compressing Pre-trained Models of Code into 3 MB. In *37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.

[38] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of big data* 6, 1 (2019), 1–48.

[39] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*.

[40] Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, and Lu Zhang. 2019. A grammar-based structural cnn decoder for code generation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 7055–7062.

[41] Gartner Symposium. 2023. Software development shifting to 'assembly and integration'. https://www.gartner.com/en/documents/3998759.

[42] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI Conference on Artificial Intelligence*.

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[44] Renshuai Tao, Yanlu Wei, Xiangjian Jiang, Hainan Li, Haotong Qin, Jiakai Wang, Yuqing Ma, Libo Zhang, and Xianglong Liu. 2021. Towards real-world X-ray security inspection: A high-quality benchmark and lateral inhibition module for prohibited items detection. In *Proceedings of the IEEE/CVF international conference on computer vision*. 10923–10932.

[45] Peri Tarr, Harold Ossher, William Harrison, and Stanley M Sutton Jr. 1999. N degrees of separation: Multi-dimensional separation of concerns. In *Proceedings of the 21st international conference on Software engineering*. 107–119.

[46] Yu-Xiong Wang and Martial Hebert. 2015. Model recommendation: Generating object detectors from few samples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1619–1628.

[47] Yueming Wu, Deqing Zou, Shihan Dou, Wei Yang, Duo Xu, and Hai Jin. 2022. VulCNN: an image-inspired scalable vulnerability detection system. In *Proceedings of the 44th International Conference on Software Engineering*. 2365–2376.

[48] Xingxing Xie, Gong Cheng, Jiabao Wang, Xiwen Yao, and Junwei Han. 2021. Oriented R-CNN for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3520–3529.

[49] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. 2018. Convolutional neural networks: an overview and application in radiology. *Insights into imaging* 9, 4 (2018), 611–629.

[50] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 7370–7377.

[51] Ziqi Zhang, Yuanchun Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2020. Dynamic slicing for deep neural networks. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 838–850.

[52] Ziqi Zhang, Yuanchun Li, Jindong Wang, Bingyan Liu, Ding Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2022. ReMoS: Reducing Defect Inheritance in Transfer Learning via Relevant Model Slicing. In *2022 IEEE/ACM 44th International Conference on Software Engineering*. 1856–1868.