Multi-Dimensional Pruning: A Unified Framework for Model Compression

Jinyang Guo Wanli Ouyang Dong Xu
School of Electrical and Information Engineering, The University of Sydney
{jinyang.guo,wanli.ouyang,dong.xu}@sydney.edu.au

Abstract

In this work, we propose a unified model compression framework called Multi-Dimensional Pruning (MDP) to simultaneously compress the convolutional neural networks (CNNs) on multiple dimensions. In contrast to the existing model compression methods that only aim to reduce the redundancy along either the spatial/spatial-temporal dimension (e.g., spatial dimension for 2D CNNs, spatial and temporal dimensions for 3D CNNs) or the channel dimension, our newly proposed approach can simultaneously reduce the spatial/spatial-temporal and the channel redundancies for CNNs. Specifically, in order to reduce the redundancy along the spatial/spatial-temporal dimension, we downsample the input tensor of a convolutional layer, in which the scaling factor for the downsampling operation is adaptively selected by our approach. After the convolution operation, the output tensor is upsampled to the original size to ensure the unchanged input size for the subsequent CNN layers. To reduce the channel-wise redundancy, we introduce a gate for each channel of the output tensor as its importance score, in which the gate value is automatically learned. The channels with small importance scores will be removed after the model compression process. Our comprehensive experiments on four benchmark datasets demonstrate that our MDP framework outperforms the existing methods when pruning both 2D CNNs and 3D CNNs.

1. Introduction

With the popularity of Convolutional Neural Networks (CNNs) for various computer vision applications, several model compression technologies were developed (see Sec. 2 for more details) to deploy CNNs on resource constrained platforms. Among these techniques, the channel pruning methods [8, 23, 38] aim to reduce the redundancy along the channel dimension of CNNs. However, substantial redundancy also exists on the spatial/spatial-temporal dimension of CNNs (*i.e.,* spatial dimension for 2D CNNs, spatial and temporal dimensions for 3D CNNs), which is not considered by the existing channel pruning methods.

Figure 1(a) shows the output feature maps of four consecutive frames generated from the output tensor of the first convolutional layer in the C3D model [31]. This figure shows the scenario that a person is diving, with the spatial resolution of 112×112 per frame. From Figure 1(a), we notice that there is minimal information in the background (the blue part) and the person (the red part) is mostly identical in the four frames, which indicates that the redundancies in CNNs also exist along the spatial and temporal dimensions. We downsample this output tensor on both spatial and temporal dimensions. As shown in Figure 1(b), after downsampling, the spatial resolution of each frame is halved (*i.e.,* from 112×112 to 56×56) and four consecutive frames are reduced to two frames. The information in Figure 1(b) is almost the same as Figure 1(a) and we can still recognize that the person is diving after the downsampling operation, which indicates that the spatial-temporal redundancy (STR) can be reduced with minimal information loss by downsampling the tensors in CNNs.

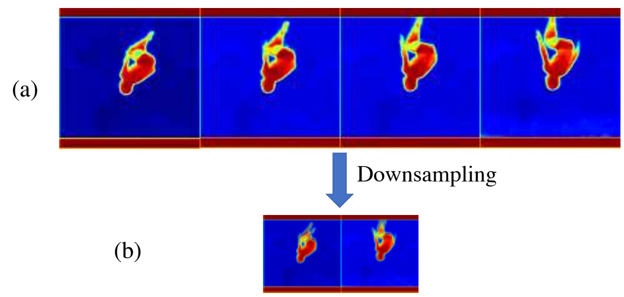

Figure 1: (a) Feature maps of four consecutive frames generated from the output tensor of the first convolutional layer (the C3D model [31] is used for illustration). The hand position of the person is mostly identical in these frames. (b) The downsampled tensor, in which the information is almost the same as the original output tensor.

In order to reduce the redundancies along multiple dimensions, one can perform different model compression algorithms in a step-by-step fashion by firstly pruning the channels and then reducing the STR. However, this solution is often sub-optimal because the dependency from different pruning stages is not well explored in the step-by-step pruning process. For exmaple, if we prune the channels in the first step and then reduce the STR in the next step, the channels considered as unimportant in the first pruning stage may become important after reducing the STR of CNNs. However, we cannot recover these channels in the second pruning stage because they have already been removed in the first stage, which degrades the performance of the compressed model. Different channels in CNNs often pay attention to different parts of feature maps. Before reducing the STR, the channels with detailed information may be more important than those with overall information as the details in the high resolution feature maps can provide rich information to CNNs. Therefore, the channels with overall information will be removed. After reducing the STR, the resolution of the feature maps will be reduced and the details in these feature maps will be lost. In this case, the channels with less details might become more informative when compared with the high resolution case. As a result, we should keep the channels with overall information.

To address the aforementioned issue, we propose a new unified framework called Multi-Dimensional Pruning (MDP) to simultaneously reduce the spatial/spatial-temporal and the channel redundancies in CNNs in an end-to-end fashion. Specifically, our MDP framework consists of three stages: the searching stage, the pruning stage, and the fine-tuning stage. In the searching stage, we construct an over-parameterized network by expanding each convolutional layer to multiple parallel branches, in which different branches correspond to information processing at different spatial/spatial-temporal resolutions. The information from different branches will be aggregated based on their importance scores. We also introduce a gate for each channel to indicate its importance. The importance scores of the branches and the channels are automatically learned in the searching stage. In the pruning stage, we prune the branches and channels based on their importance scores. We finally fine-tune the compressed model to recover from the accuracy drop.

To the best of our knowledge, this is the first unified model compression framework that can simultaneously reduce the spatial and channel redundancy for 2D CNNs, and the spatial, temporal, and channel redundancy for 3D CNNs. When compared with the existing channel pruning methods [21, 38] or other methods that aim at reducing the STR [4, 33, 39], our MDP framework has several advantages: **(1)** The MDP framework is a unified model compression framework, which is suitable for both 2D CNNs and 3D CNNs. **(2)** The optimal combination of selected features from multiple dimensions (*i.e.,* the selected branches for the spatial/spatial-temporal dimension, and the selected channels for the channel dimension) can be automatically learned in the searching stage, which can solve the sub-optimal problem in the alternative approach that sequentially uses different model compression algorithms in a step-by-step fashion.

The experiments on four benchmark datasets demonstrate the effectiveness of our proposed MDP framework for both image classification and video classification tasks.

2. Related Work

Channel pruning. Channel pruning technologies [23, 8, 38, 35, 24, 7, 22, 37] aim to reduce the channel-wise redundancy in CNNs. In [18], Lin *et al.* proposed to use adversarial learning to prune the redundant structures in CNNs. Guo *et al.* [5] pruned the channels by using the guidance from the classification loss and feature importance. However, the existing channel pruning methods ignore the STR in CNNs. Unlike these channel pruning methods, our MDP framework can additionally reduce the redundancy along the spatial/spatial-temporal dimension.

Spatial/spatial-temporal redundancy reduction. Recently, many methods [4, 12] were proposed to reduce the spatial redundancy when designing the architectures of CNNs. For example, Chen *et al.* [4] proposed to replace the vanilla convolution by the octave convolution. On the other hand, several network architectures [33, 39, 17] were proposed to reduce the temporal redundancy in 3D CNNs. TSN [33] uses the sparse temporal sampling strategy to reduce the computational cost for long-term temporal structure. ECO [39] mixes 2D and 3D networks to save computation. The goals of these methods [4, 12, 33, 39, 17] are to design new type of convolution operations or network architectures instead of compressing a given network. In contrast, we aim to compress a given model by jointly pruning the network along the spatial and channel dimensions for 2D CNNs, and along the spatial, temporal, and channel dimensions for 3D CNNs. Due to the newly designed convolution operations or network structures, these methods are not suitable for model compression as they need to train the models from scratch. Therefore, these approaches cannot transfer the information from the pre-trained model to the compressed model, which will degrade the performance of the compressed models. Moreover, these methods only focus on how to reduce the redundancy along one dimension (spatial or temporal). In contrast, our approach can jointly reduce the redundancies along multiple dimensions, which thus achieves better performance.

Multi-scale representation learning. Multi-scale representation learning [10, 30, 19] has been demonstrated to be effective for many computer vision tasks. For example, Elastic-Net [32] introduces the elastic module in CNNs to extract multi-scale feature representations. In [16], the features from multiple scales are concatenated to obtain the information from different scales. The goal of these multi-scale representation learning approaches is to capture the information of features at different scales, but our MDP framework aims to compress a given CNN to obtain a more efficient network.

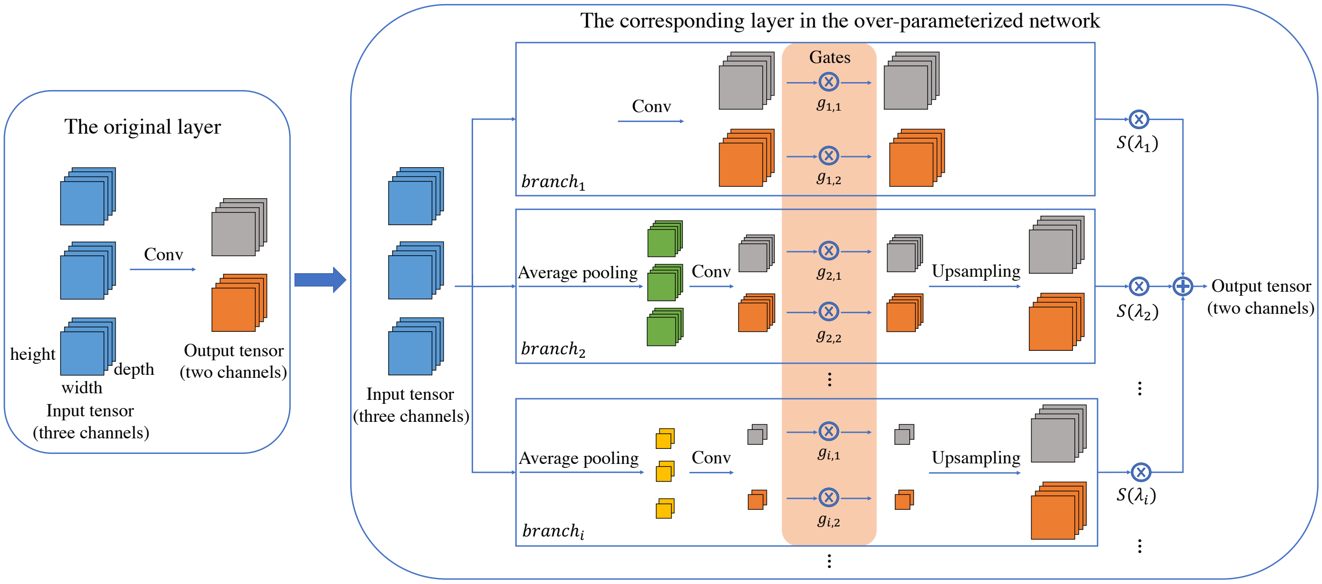

Figure 2: Construction of the over-parameterized network. We expand each convolutional layer into multiple parallel branches and introduce a gate to each channel of the output tensor at each branch. In this figure, $\mathcal{S}(\cdot)$ and $g_{\cdot,\cdot}$ are the importance scores for the branches and the gate values for the gates, respectively. The input tensors (with 3 channels) and the output tensors (with 2 channels) and other tensors are all 4th order tensors, which are shown as three or two 3rd order tensors for better presentation. For the convolution operation, we use two filters to generate the output tensors with two channels.

Network architecture search. While our MDP method is also related to the network architecture search methods [20, 2], we aim to reduce the redundancies in CNNs by pruning a given model along different dimensions instead of searching the optimal network architecture as in [20, 2].

3. Multi-Dimensional Pruning

In this section, we take the process of compressing 3D CNNs as an example to introduce our MDP framework, which is a more general case. The algorithm for compressing 2D CNNs can be readily obtained.

3.1. Overview

Our MDP framework consists of three stages: the searching stage, the pruning stage, and the fine-tuning stage. In the searching stage, we firstly construct an over-parameterized network from any given original network to be pruned and then train this over-parameterized network by using the objective function introduced in Sec. 3.2.2. In the pruning stage, we prune the unimportant branches and channels in this over-parameterized network based on the importance scores learned in the searching stage. In the fine-tuning stage, we fine-tune the pruned network to recover from the accuracy drop.

3.2. The searching stage

3.2.1 Overview of the over-parameterized network

In the searching stage, we firstly construct an over-parameterized network from any given original network. The compressed model can be obtained by pruning this over-parameterized network. Figure 2 shows one convolutional layer (left) in the original network and its corresponding layer (right) in the over-parameterized network. We expand each convolutional layer into several parallel branches in the corresponding layer of the over-parameterized network. Since we only focus on one convolutional layer in most places in this section, we omit the index of this layer when introducing each operation except in Eq. (3), where the layer index is denoted as the superscript $\cdot^{(l)}$ for the l-th layer.

In each branch, we perform the following operations: **(1)** We downsample the input tensor of the convolutional layer by applying average pooling along the spatial and/or temporal dimension. In our implementation, we downsample the input tensor to 4 different resolutions along the spatial dimension with four scaling factors 1, 2, 3, and 4, where the scaling factor of 1 means there is no downsampling operation (*i.e.*, identity mapping). We also downsample the input tensor with four different scaling factors (1, 2, 3, and 4) along the temporal dimension. Therefore, the total number of branches in each layer of the over-parameterized network is $4 \times 4 = 16$ after performing the downsampling operation along the spatial and temporal dimensions. In Figure 2, we have three channels of the input tensor of this layer $\mathbf{X}$ (the blue part) and each channel is represented as one blue block. There are 4 frames along the temporal dimension in each channel, which is represented as the depth of each blue block. For $branch_2$ in Figure 2, we downsample $\mathbf{X}$ by using the scaling factor of 2 and 1 along the spatial dimension and the temporal dimension, respectively. We obtain the downsampled tensor $\mathbf{I}^2$ in $branch_2$, which is marked as green. In this case, the height/width of $\mathbf{I}^2$ becomes half of the height/width of $\mathbf{X}$ while the number of frames is 4 in both $\mathbf{X}$ and $\mathbf{I}^2$. Similar to the downsampling operation in $branch_2$, we downsample $\mathbf{X}$ by using the scaling factor of 4 and 2 along the spatial dimension and the temporal dimension in $branch_i$, respectively. The downsampled input tensor of this branch $\mathbf{I}^i$ is marked as yellow. In this case, the height/width of $\mathbf{I}^i$ become a quarter of the height/width of $\mathbf{X}$ and 4 frames in $\mathbf{X}$ are downsampled as 2 frames. **(2)** The downsampled tensor of each branch is fed into the convolutional layer. The parameters of the convolutional layers in all branches of the over-parameterized network are copied from the original network. **(3)** After the convolutional layer, we multiply each channel by a gate, which is indicated as $g_{\cdot,\cdot}$ in Figure 2. The gate is used for evaluating the importance of each channel and its value is simultaneously learned with the parameters of convolutional layers in the searching stage. **(4)** After the convolution operation, we upsample the output tensor of the convolutional layer to the original size so that the output tensors from different branches can be integrated without resolution mismatch. In our implementation, we choose the simple nearest neighbour interpolation method for the upsampling operation because it does not require any computation and achieves good results in our proposed framework.

The upsampled tensors from multiple branches are integrated into the final output tensor of this layer in the over-parameterized network. Inspired by DARTS [20], we use weighted sum as the integration strategy and the weights are indicated as $\mathcal{S}(\cdot)$ in Figure 2.

3.2.2 Formulation

Formally, for constructing the over-parameterized network of 3D CNNs at each layer, let us denote the input tensor at this layer as $\mathbf{X} \in \mathbb{R}^{c_{in} \times d_{in} \times h_{in} \times w_{in}}$ where c_{in} is the number of input channels for this layer, d_{in} is the length of the input tensor along the temporal dimension, and h_{in} and w_{in} are the height and width of the input tensor, respectively. Similarly, the output tensor at this layer can be denoted as $\mathbf{Y} \in \mathbb{R}^{c_{out} \times d_{out} \times h_{out} \times w_{out}}$, where c_{out} is the number of output channels for this layer, d_{out} is the length of the output tensor along the temporal dimension, and h_{out} and w_{out} are the height and the width of the output tensor, respectively. The convolutional layer connects the input tensor $\mathbf{X}$ and the output tensor $\mathbf{Y}$ by a given transformation (*e.g.,* a convolution operation) $\mathcal{T}$, where $\mathbf{Y} = \mathcal{T}(\mathbf{X})$. Suppose we have B branches for this layer in the over-parameterized network, the output tensor of this layer in the over-parameterized network can be written as follows:

$$\mathbf{Y} = \sum_{i=1}^{B} \mathcal{S}(\lambda_i) \cdot \mathcal{U}_i\{\mathcal{T}_i[\mathcal{A}_i(\mathbf{X})]\}, \tag{1}$$

where $\mathcal{A}_i$, $\mathcal{T}_i$, and $\mathcal{U}_i$ are the average pooling operation, the transformation function, and the upsampling operation in the i-th branch at this layer, respectively. λ_i is the learnable parameter in the i-th branch and $\mathcal{S}(\cdot)$ is the softmax function, namely, $\mathcal{S}(\lambda_i) = \frac{\exp(\lambda_i)}{\sum_{b=1}^{B}\exp(\lambda_b)}$. Note $\mathcal{S}(\lambda_i)$ represents the importance score of the i-th branch at this layer.

Denote $\mathbf{I}^i$ as the pooled input tensor of the i-th branch at this layer after using the average pooling operation, *i.e.,* $\mathbf{I}^i = \mathcal{A}_i(\mathbf{X})$. Denote $\mathbf{I}^i_{j,:,:,:}$ as the j-th channel of $\mathbf{I}^i$. In the i-th branch, the output after performing the transformation operation $\mathcal{T}_i$ can be written as follows:

$$\mathbf{O}^i_{k,:,:,:} = g_{i,k} \cdot \sum_{j=1}^{c_{in}} \mathbf{I}^i_{j,:,:,:} * \mathbf{W}^i_{k,j,:,:,:}, \tag{2}$$

where $\mathbf{O}^i$ is the output tensor before the upsampling operation in the i-th branch and $\mathbf{O}^i_{k,:,:,:}$ is the k-th channel of $\mathbf{O}^i$. $\mathbf{W}^i \in \mathbb{R}^{c_{out} \times c_{in} \times d_{kr} \times h_{kr} \times w_{kr}}$ is the weight tensor for the convolution operation where the subscript kr denotes the kernel/filter. $g_{i,k}$ is the gate value for the k-th channel in the i-th branch. $*$ is the convolution operation. We omit the bias term and the activation function in Eq. (2) for better presentation.

Objective function. After independently constructing the over-parameterized network at each layer, we train the over-parameterized network by using the following objective function, which is inspired by [21, 2]. For better presentation, we additionally introduce the superscript $\cdot^{(l)}$ to denote the corresponding symbols of the l-th layer in the following objective function.

$$\begin{aligned} &\arg\min_{\mathbf{\Theta},\boldsymbol{\lambda},\mathbf{G}} \mathcal{L} = \mathcal{L}_c + \alpha\mathcal{L}_{st} + \eta\mathcal{L}_{gate}, \\ &\text{where } \mathcal{L}_{st} = \sum_{l=1}^{L}\sum_{i=1}^{B^{(l)}} \mathcal{S}(\lambda_i^{(l)}) \cdot \mathcal{F}(\mathcal{T}_i^{(l)}), \\ &\mathcal{L}_{gate} = \sum_{l=1}^{L}\sum_{i=1}^{B^{(l)}}\sum_{k=1}^{c_{out}^{(l)}} \|g_{i,k}^{(l)}\|_1. \end{aligned} \tag{3}$$

$\mathcal{L}_c$ is the standard cross-entropy loss. $\mathcal{L}_{st}$ is the penalty to control the computational complexity, with which we expect the importance scores of most branches are close to zero so that most branches will be removed in the pruning process. $\mathcal{L}_{gate}$ is the l_1-norm based regularizer that enforces the gate values $g_{i,k}^{(l)}$ from most branches and channels to be close to zero so that these channels can be safely removed in the pruning process. L is the total number of layers in our network and $B^{(l)}$ is the number of branches at the l-th layer. $\mathbf{\Theta}$ is the parameters of the whole network. At the searching stage, $\mathbf{\Theta}$ is initialized from the original network to be pruned. $\boldsymbol{\lambda}$ is the set containing the learnable parameters $\lambda_i^{(l)}$ from all the layers, $\boldsymbol{\lambda} = \{\lambda_1^{(1)}, \lambda_2^{(1)}, \ldots, \lambda_i^{(l)}, \ldots\}$

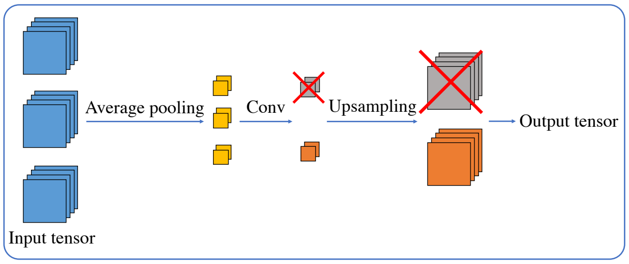

Figure 3: Illustration of the pruning stage (the selected branch of one layer is used as an example for illustration). For this layer, we select the i-th branch and the second channel of the i-th branch in the over-parameterized network in Figure 2.

where $\lambda_i^{(l)}$ is the learnable parameter to obtain the importance score for the i-th branch at the l-th layer. $\mathbf{G}$ is the set containing the gate values for all channels from all layers, $\mathbf{G} = \{g_{1,1}^{(1)}, g_{1,2}^{(1)}, \ldots, g_{i,k}^{(l)}, \ldots\}$ where $g_{i,k}^{(l)}$ is the gate value for the k-th channel of the i-th branch at the l-th layer. $\mathcal{F}(\mathcal{T}_i^{(l)})$ denotes the number of floating point operations (FLOPs) of the transformation $\mathcal{T}_i^{(l)}$ at the l-th layer, which is widely used for computational complexity measurement. α and η are two coefficients to balance different losses. $\|\cdot\|_1$ denotes the l_1 norm. We can control the compression ratio by adjusting the values of α and η. Specifically, higher values of α and η will result in higher compression ratio.

3.3. The pruning stage

We perform the pruning process after the searching stage is finished. At each layer, we only select the branch with the largest importance score and prune other branches, and we also prune the channels with small gate values in the selected branch. In Figure 3, we show the pruning process for one convolutional layer. Here, we ignore the superscript $\cdot^{(l)}$ again for better presentation. Suppose the i-th branch in Figure 2 has the largest importance score $\mathcal{S}(\lambda_i)$. In this case, in the pruning stage, we keep the i-th branch and remove other branches at this layer. At the same time, let us assume the second channel of this branch has the largest importance score $g_{i,2}$. Therefore, we preserve the second channel and remove the first channel. As shown in Figure 3, we finally arrive at the pruned network for this layer after the pruning stage.

3.4. The fine-tuning stage

We perform the fine-tuning process on the pruned network to recover from the accuracy drop. After the fine-tuning stage, the compressed model is obtained.

3.5. Compressing 2D CNNs

Compressing 2D CNNs by using our MDP framework is a special case of compressing 3D CNNs. For 2D CNNs, we only apply the average pooling operation for downsampling along the spatial dimension. Similar to the process of compressing 3D CNNs, we downsample the input tensor by using the scaling factors of 1, 2, 3, 4. In this case, we have 4 branches for each layer in the over-parameterized network. At the same time, we only upsample the output tensor of each branch along the spatial dimension. After constructing the over-parameterized network, the following stages are the same as those for pruning 3D CNNs.

3.6. Comparison with other methods

Our work is related to multi-scale representation learning approaches [16, 32] and channel pruning methods [21, 18]. However, our work is different from these methods in both motivation and formulation. The multi-scale representation learning approaches aim to capture the multi-scale information to improve the accuracy of CNNs. Therefore, multiple branches are selected and preserved in the final model in [16, 32]. In contrast to these two methods, our work aims to compress CNNs and only one branch is preserved in the compressed model. Although the channel pruning methods in [21, 18] also prune the channels based on the learnable importance scores, the spatial-temporal redundancy is not explored in their works [21, 18].

4. Experiments

In order to demonstrate the effectiveness our MDP framework, we compare our MDP approach with several state-of-the-art model compression methods, including ThiNet [23], Channel Pruning (CP) [8], Slimming [21], Width-multiplier (WM) [9], DCP [38], GAL [18], Taylor Pruning (TP) [24], Filter Pruning (FP) [15], and Regularization-based pruning (RBP) [36] on four benchmark datasets: CIFAR-10 [13], ImageNet [26], UCF-101 [29], and HMDB51 [14].

The percentage of the number of floating point operations [#FLOPs(%)] in this section refers to the ratio of the FLOPs from the pruned network over that from the original network, which is a commonly used criterion for computational complexity measurement.

Datasets. CIFAR-10 and ImageNet are used to evaluate the effectiveness of our proposed method when pruning 2D CNNs for the image classification task. CIFAR-10 consists of 50k training images and 10k testing images from 10 classes. ImageNet is a large dataset, which contains over 1 million training images and 50k testing images from 1000 categories. On the other hand, UCF-101 and HMDB51 are used to evaluate the performance of our MDP method when pruning 3D CNNs for the video classification task. Specifically, UCF-101 consists of 13,320 videos from 101 classes, while HMDB51 contains 6,766 videos from 51 classes.

Implementation details. Based on the original network, we apply our MDP approach to compress the model along

Model		ThiNet [23]	CP [8]	Slimming [21]	WM [9]	DCP [38]	**MDP**
VGGNet (Baseline 93.99%)	#FLOPs (%)	50.00	50.00	49.02	50.00	34.97	24.84
	Acc. (%)	93.85	93.67	93.80	93.61	94.57	**94.57**
ResNet-56 (Baseline 93.74%)	#FLOPs (%)	50.25	50.00	-	50.25	50.25	45.11
	Acc. (%)	92.98	91.80	-	93.24	93.49	**94.29**
MobileNet-V2 (Baseline 95.02%)	#FLOPs (%)	-	-	-	73.53	73.53	71.29
	Acc. (%)	-	-	-	94.02	94.69	**95.14**

Table 1: Comparison of Top-1 accuracies from different model compression methods for compressing VGGNet, ResNet-56, and MobileNet-V2 on the CIFAR-10 dataset. When using the CP method [8], we directly quote the results in the original work [8] for compressing ResNet-56. The other results are copied from the work in [38].

Model		ThiNet [23]	CP [8]	WM [9]	DCP [38]	DCP+SP	GAL [18]	**MDP**
ResNet-50 (Baseline 92.94%)	#FLOPs (%)	44.44	50.00	44.44	44.44	45.47	45.06	44.29
	Top-5 Acc. (%)	90.02	90.80	91.31	92.32	92.46	90.82	**92.66**
MobileNet-V2 (Baseline 90.56%)	#FLOPs (%)	55.25	-	55.25	55.25	-	-	56.85
	Top-5 Acc. (%)	86.44	-	85.51	86.34	-	-	**88.86**

Table 2: Comparison of Top-5 accuracies from different model compression methods for compressing ResNet-50 and MobileNet-V2 on ImageNet. For the state-of-the-art works, we directly quote the results from [38].

multiple dimensions. We adjust the #FLOPs by choosing different values of α and η. For image classification, we use the SGD optimizer with nesterov for optimization at the searching stage. On CIFAR-10, the initial learning rate, the batch size, and the momentum are set to 0.1, 256, and 0.9, respectively. The settings on ImageNet are the same as CIFAR-10 except that the initial learning rate is set to $1e^{-2}$. In the fine-tuning stage, we follow [18] to fine-tune the pruned network with hint [25] from the last layer. The other settings are the same as the searching stage.

For the C3D model [31] used in the video classification task, we follow [36] to split each video into several non-overlapped clips with 16 frames as the input of the network. For Inflated 3D (I3D) ConvNet [34], we pre-train the model [1] based on kinetics [3, 11] and fine-tune the pre-trained model on UCF-101 and HMDB51 to obtain the original models before model compression. For fair comparison, we follow [1] to use the frame length of 32 as the input size. The initial learning rate, the batch size, and the weight decay are set to $1e^{-3}$, 32, and $5e^{-4}$, respectively. The other settings are the same as ImageNet.

4.1. Results on CIFAR-10

On CIFAR-10, we evaluate the effectiveness of our MDP method by using three widely used models: VGGNet [28], ResNet-56 [6], and MobileNet-V2 [27]. The results are shown in Table 1. For VGGNet, our MDP method achieves the accuracy of 94.57% with only 24.84% #FLOPs, while it takes 34.97% #FLOPs to achieve the same accuracy for DCP [38]. For ResNet-56 and MobileNet-V2, our proposed MDP approach achieves higher accuracies with lower #FLOPs when compared with other baseline methods. The results clearly demonstrate the effectiveness of our MDP method on small-scale datasets. It is also worth mentioning that our MDP approach outperforms the pre-trained ResNet-56 and MobileNet-V2 by 0.55% and 0.12%, respectively. Similar results are also reported in the DCP work [38]. One possible explanation is that the overfitting problem on small-scale datasets like CIFAR-10 can be partially solved by compressing the models.

4.2. Results on ImageNet

In order to compare our proposed approach with other state-of-the-art methods on large-scale datasets, we compress ResNet-50 [6] and MobileNet-V2 [27] on ImageNet. We follow the setting in [8] to compare the Top-5 accuracy with other methods and the results are shown in Table 2.

To investigate the advantages by simultaneously reducing the redundancies along multiple dimensions, we also report the results from an alternative approach by using the step-by-step pruning strategy, which prunes the channels by using the DCP method in the first step and then reduces the spatial redundancy by using our MDP approach at the second step. The result is referred as DCP+SP in Table 2.

From Table 2, we have the following observations: **(1)** For ResNet-50, our MDP method outperforms other existing approaches, which indicates that it is beneficial to compress the model by using our MDP approach. **(2)** For MobileNet-V2, our proposed method surpasses other state-of-the-art approaches by more than **2.4%** when #FLOPs are comparable, which is a significant improvement on the ImageNet dataset. **(3)** When comparing the DCP+SP approach with the DCP method, the DCP+SP approach performs better, which indicates that it is beneficial to additionally reduce the redundancy along the spatial dimension. **(4)** Our MDP framework outperforms the DCP+SP approach by 0.2%, which demonstrates the effectiveness of our MDP method to jointly prune models along multiple dimensions.

Method	#FLOPs (%)	Acc. (%)
C3D [31] (Baseline)	100.00	82.10
TP [24]	49.49	72.48
FP [15]	49.45	77.58
DCP [38]	50.37	77.77
MDP	49.89	**80.17**
I3D [1, 34] (Baseline)	100.00	93.47
TP [24]	48.90	84.48
FP [15]	50.62	85.38
DCP [38]	49.13	85.20
MDP	47.02	**88.03**

Table 3: Video-level accuracy comparison from different model compression methods on UCF-101 (Split 1). The results of the existing methods are based on our implementation. For the I3D model, we use ResNet-50 as the backbone.

The results on the ImageNet dataset clearly demonstrate that it is effective to use our proposed approach to simultaneously reduce the redundancies along multiple dimensions on large-scale datasets.

4.3. Results on UCF-101

In order to evaluate the performance of our MDP approach when pruning 3D CNNs, we conduct the experiments to compress C3D [31] and I3D [1, 34] on UCF-101 [29] (Split 1). We follow the setting in [34] to report the video-level accuracies. For the I3D model, we use ResNet-50 as the backbone.

The results are shown in Table 3. From Table 3, we observe that our MDP approach outperforms other channel pruning methods for compressing both the C3D and I3D models. Since the work RBP [36] only reports the clip-level accuracies for compressing the C3D model, we compare our MDP approach with the RBP method in terms of the clip-level accuracy. The clip-level accuracy is 76.38% with 50.00% #FLOPs for the RBP approach, while it is 78.06% with 49.89% #FLOPs for our MDP method. The results demonstrate the effectiveness of our MDP method for pruning 3D CNNs.

4.4. Results on HMDB51

We also conduct more experiments on the HMDB51 dataset to further evaluate the performance of our MDP approach for pruning 3D CNNs. Similar to the experiments on UCF-101, we compress C3D and I3D on HMDB51 (Split 1) and report the video-level accuracies. The results are shown in Table 4. Again, our MDP approach consistently outperforms other baseline methods with lower #FLOPs, which further demonstrate the effectiveness of the proposed approach for compressing 3D CNNs.

4.5. Ablation study

4.5.1 Ablation study for 2D CNNs

In this section, we compress ResNet-56 on CIFAR-10 to investigate different components in our MDP approach for compressing 2D CNNs.

Method	#FLOPs (%)	Acc. (%)
C3D [31] (Baseline)	100.00	47.39
TP [24]	47.43	41.90
FP [15]	49.45	43.53
DCP [38]	49.45	42.22
MDP	47.15	**44.97**
I3D [1, 34] (Baseline)	100.00	69.41
TP [24]	48.84	59.48
FP [15]	50.62	57.84
DCP [38]	49.13	58.24
MDP	48.00	**62.88**

Table 4: Video-level accuracy comparison from different model compression methods on HMDB51 (Split 1). The results of the existing methods are based on our implementation. For the I3D model, we use ResNet-50 as the backbone.

Method	#FLOPs (%)	Acc. (%)
MDP w/o CP	45.19	92.86
MDP w/o SP	45.41	93.39
MDP	45.11	94.29

Table 5: Accuracies of our MDP approach and its variants for pruning ResNet-56 on CIFAR-10, which includes MDP without pruning the channels (MDP w/o CP), and MDP without reducing the spatial redundancy (MDP w/o SP).

Effect of channel pruning. To investigate the effectiveness for pruning the channels in our MDP method, we perform the experiment to only reduce the spatial redundancy without pruning the channels, which is referred to as MDP w/o CP in Table 5. From Table 5, our MDP approach outperforms MDP w/o CP method by 1.43%, which indicates that it is effective to prune the channels in our MDP approach.

Effect of spatial pruning. To investigate the effectiveness for reducing the spatial redundancy in our MDP framework, we perform the experiment to only prune the channels without removing the spatial redundancy, which is referred to as MDP w/o SP in Table 5. From Table 5, our MDP method surpasses the MDP w/o SP approach by 0.9%, which indicates that it is effective to reduce the redundancy along the spatial dimension in our MDP approach.

The experimental results in Table 5 show that either the channel pruning or the spatial pruning alone does not perform better than our MDP method, which jointly performs both of them. Therefore, it is beneficial to jointly prune the models along multiple dimensions by using our MDP method.

Effect of α and η. We conduct more experiments to investigate the performance when choosing different values of α and η. For fair comparison, we adjust α and η in Eq. (3) to obtain the compressed models with similar #FLOPs and compare their performance. The results are shown in Table 6. From Table 6, we have several observations: **(1)** The models with similar #FLOPs can be obtained by increasing one of the coefficients and decreasing the other one. **(2)**

α	η	#FLOPs (%)	Acc. (%)
$2e^{-15}$	$1e^{-10}$	43.44	94.18
$2e^{-14}$	$1e^{-11}$	43.53	94.26
$2e^{-13}$	$1e^{-12}$	45.11	94.29
$2e^{-12}$	$1e^{-13}$	45.47	94.23
$2e^{-11}$	$1e^{-14}$	45.03	94.22

Table 6: Accuracies of our MDP method by using different values of α and η when pruning ResNet-56 on CIFAR-10.

Method	#FLOPs (%)	Acc. (%)
MDP w/o CP	49.54	76.92
MDP w/o STP	49.50	77.66
MDP	49.89	80.17

Table 7: Video-level accuracies of our MDP approach and its variants for pruning C3D on UCF-101, which includes MDP without pruning the channels (MDP w/o CP) and MDP without reducing the spatial-temporal redundancy (MDP w/o STP).

With similar #FLOPs, the performance of the compressed model obtained by using different values of α and η are comparable, which shows the performance of our MDP approach is not sensitive to the coefficients α and η.

4.5.2 Ablation study for 3D CNNs

To investigate the effectiveness of different components in our MDP method for compressing 3D CNNs, we perform the experiments to compress C3D on UCF-101 and report the video-level accuracies. In 3D CNNs, the spatial-temporal redundancy exists along both the spatial and the temporal dimensions.

We firstly perform the experiment to only reduce the spatial-temporal redundancy without pruning the channels in the C3D model, which is referred to as MDP w/o CP in Table 7. We also perform the experiment to only prune the channels without reducing the spatial-temporal redundancy in the C3D model, which is referred to as MDP w/o STP. From Table 7, our MDP method outperforms the alternative approach MDP w/o CP, which indicates that it is effective to prune the channels for compressing 3D CNNs in our MDP method. Our MDP method also surpasses the alternative approach MDP w/o STP by 2.51%, which demonstrates the effectiveness of our MDP approach for reducing the spatial-temporal redundancy when compressing 3D CNNs.

4.6. Branch selection analysis

In Figure 4, we report the index of selected branch (*i.e.,* the selected scaling factor for the downsampling operation) for each convolutional layer in the compressed ResNet-50 model on the ImageNet dataset. From Figure 4, it is interesting to see that the branches with higher downsampling scaling factor (*e.g.,* the index of selected branch is 4) tend to appear in the shallower layers (close to the input of CNNs),

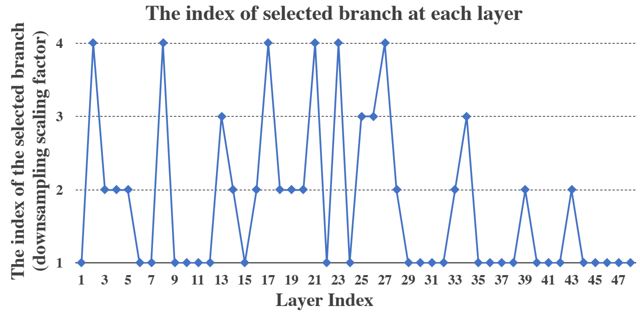

Figure 4: The index of selected branch (*i.e.*, the selected scaling factor for the downsampling operation) in each convolutional layer when pruning ResNet-50 on ImageNet. The layer indices 1 to 48 correspond to the first convolutional layer in the first residual block to the last convolutional layer in the last residual block in ResNet-50. The branches with larger scaling factors tend to appear in shallower layers, while the branches with smaller scaling factors tend to appear in the deeper layers.

while the branches with lower downsamling scaling factor (*e.g.,* the index of selected branch is 1) tend to appear in the deeper layers (close to the output of CNNs). We hypothesize that the feature maps in the shallower layers have higher resolutions, which indicates more redundancy along the spatial dimension. On the other hand, the feature maps in the deeper layers have lower resolutions along the spatial dimension, which indicates there is less redundancy along this dimension. It is also worth mentioning that we do not reduce the resolution (*i.e.*, the index of selected branch is 1) for most of the branches from layer 29 to layer 48, which suggests that the spatial redundancy in the deeper layers of the ResNet-50 model can be neglected possibly because of low spatial resolutions in these deep layers.

5. Conclusion

In this work, we have proposed a unified model compression framework called Multi-Dimensional Pruning (MDP) to compress 2D CNNs along the channel and spatial dimensions, and 3D CNNs along the channel, spatial, and temporal dimensions. In contrast to the existing model compression approaches that only reduce the redundancy along one certain dimension, our proposed framework can simultaneously reduce the redundancies along multiple dimensions and thus significantly accelerate CNNs when they are deployed on resource constrained platforms. Comprehensive experiments on four benchmark datasets demonstrate the effectiveness of our MDP method for pruning both 2D CNNs and 3D CNNs.

Acknowledgement. This work was supported by the Australian Research Council (ARC) Future Fellowship under Grant FT180100116. Wanli Ouyang was supported by the ARC Discovery Project under Grant DP200103223.

References

[1] pytorch-resnet3d. https://github.com/Tushar-N/pytorch-resnet3d. 2019. 6, 7

[2] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 3, 4

[3] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*, 2017. 6

[4] Yunpeng Chen, Haoqi Fang, Bing Xu, Zhicheng Yan, Yannis Kalantidis, Marcus Rohrbach, Shuicheng Yan, and Jiashi Feng. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. In *ICCV*, 2019. 2

[5] Jinyang Guo, Wanli Ouyang, and Dong Xu. Channel pruning guided by classification loss and feature importance, 2020. 2

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 6

[7] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: Automl for model compression and acceleration on mobile devices. In *ECCV*, pages 815–832, 2018. 2

[8] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, pages 1398–1406, 2017. 1, 2, 5, 6

[9] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 5, 6

[10] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017. 2

[11] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017. 6

[12] Tsung-Wei Ke, Michael Maire, and Stella X Yu. Multigrid neural architectures. In *CVPR*, 2017. 2

[13] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 5

[14] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *ICCV*, 2011. 5

[15] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2016. 5, 7

[16] Yi Li, Zhanghui Kuang, Yimin Chen, and Wayne Zhang. Data-driven neuron allocation for scale aggregation networks. In *CVPR*, 2019. 2, 5

[17] Ji Lin, Chuang Gan, and Song Han. Temporal shift module for efficient video understanding. *ICCV*, 2019. 2

[18] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *CVPR*, 2019. 2, 5, 6

[19] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 2

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019. 3, 4

[21] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *CVPR*, 2017. 2, 4, 5, 6

[22] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. *ICCV*, 2019. 2

[23] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5068–5076, 2017. 1, 2, 5, 6

[24] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *ICLR*, 2017. 2, 5, 7

[25] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *ICLR*, 2015. 6

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 5

[27] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 6

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2014. 6

[29] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. 2012. 5, 7

[30] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. *CVPR*, 2019. 2

[31] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015. 1, 6, 7

[32] Huiyu Wang, Aniruddha Kembhavi, Ali Farhadi, Alan Yuille, and Mohammad Rastegari. Elastic: Improving cnns with instance specific scaling policies. In *CVPR*, 2019. 2, 5

[33] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *ECCV*. Springer, 2016. 2

[34] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018. 6, 7

[35] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and

Larry S Davis. NISP: Pruning networks using neuron importance score propagation. In *CVPR*, 2018. 2

[36] Yuxin Zhang, Huan Wang, Yang Luo, and Roland Hu. Three dimensional convolutional neural network pruning with regularization-based method. In *NeurIPS Workshop*, 2018. 5, 6, 7

[37] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. Variational convolutional neural network pruning. In *CVPR*, 2019. 2

[38] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *NeurIPS*, 2018. 1, 2, 5, 6, 7

[39] Mohammadreza Zolfaghari, Kamaljeet Singh, and Thomas Brox. Eco: Efficient convolutional network for online video understanding. In *ECCV*, 2018. 2