Computation-Performance Optimization of Convolutional Neural Networks with Redundant Filter Removal

Chih-Ting Liu, Tung-Wei Lin, Yi-Heng Wu, Yu-Sheng Lin, Heng Lee, Yu Tsao, and Shao-Yi Chien *Member, IEEE*

***Abstract*—Convolutional Neural Networks (CNNs) are widely employed in modern computer vision algorithms, where the input image is convolved iteratively by many filters to extract the knowledge behind it. However, while the depth of convolutional layers gets deeper and deeper in recent years, the enormous computational complexity makes it difficult to be deployed on embedded systems with limited hardware resources. In this paper, inspired by rate-distortion optimization in image and video coding, we propose a computation-performance optimization (CPO) method to remove the redundant convolution filters in a CNN with performance constraints. To prove the effectiveness of the proposed method, CPO is applied to the networks for image super-resolution (SR) and image classification. Under almost the same PSNR drop and accuracy drop for performance evaluation in these two tasks, we can achieve the best parameter and computation reduction when compared with previous works.**

***Index Terms*—Filter Pruning, Redundancy Reduction, Computation-Performance Optimization, Convolutional Neural Networks.**

I. Introduction

Convolutional Neural Network (CNN) has been widely used since it attained significant improvement the first time on ImageNet Classification Challenge [1]. In recent years, various advanced architectures of Convolutional Neural Network (CNN) are proposed [2], [3], which achieve state-of-the-art performance on many computer vision tasks, such as image segmentation [4], object detection [5], and image super resolution [6]. The general trend of designing a well-performed network is making it deeper and more complex. However, it also increases the number of parameters and convolution operations at the same time, which means it will consume substantial storage and computational resources. Commonly, we can conduct the training stage of deep CNNs on high-performance GPU clusters, but for the Internet of Things (IoT) applications, we consider more about the inference stage on local devices with lower computation ability, such as mobile phones or surveillance cameras. Local computation on embedded system is more preferred than cloud-based solution owing to the real-time processing, better privacy, and no transmission bandwidth constraint. Under these considerations, it is much more difficult to employ those high computation-demanding models on edge devices.

C.-T. Liu, Y.-H. Wu, Y.-S. Lin were with the Graduate Institute of Electroncs Engineering and Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan

Manuscript received April 19, 2005; revised August 26, 2015.

For these purposes, various works focus on optimizing the deep neural networks by removing the redundancy. LeCun *et al.* propose compressing models by pruning model weights [7], and in the past few years, Han *et al.* achieve impressive compression rates on VGGNet [2] by pruning parameters with small magnitudes [8]. With these compression methods, we can efficiently reduce the parameters in fully-connected layers or in the filters of convolutional layers [9]. However, the pruning result on convolutional layers leads to sparse weight matrix with the same model architecture. Therefore, without alternative libraries or specific hardware accelerators conducting sparse operations, the compressed network with weight pruning cannot actually help reduce the computation time on general processors, such as GPUs and DSPs.

Rather than weight pruning, filter removal (or filter pruning) is another aspect of pruning which is beneficial for general computing platforms. CNNs with large capacity usually have redundancy among different filters. Therefore, recently Ki *et al.* and Chen *et al.* propose the methods of optimizing the model architecture by removing the entire convolution filter at a time according to the sparsity, which is defined differently in these two works [10], [11]. Based on the definition of filter sparsity [11], our previous work [12] defines the reducing factor component and analyzes the sensitivity of a network. By pruning the less sensitive part, we can obtain lower performance drop with the same number of parameters left.

Nonetheless, there are still no clear and systematic methodology for probing the sensitivity of a CNN network [12]. In this paper, inspired by rate-distortion optimization (RDO) technique widely employed in video and image coding, we define the sensitivity of each convolutional layer and propose a new computation-performance optimization (CPO) algorithm to successively choose the proper layers to reduce the computation by filter removal when given some performance constraints. The layer with the lowest sensitivity will be pruned first, and by monitoring the performance sensitivity globally, we can derive the number of filters to remove for each layer. Therefore, by pruning the model with filter removal method and the CPO algorithm, we can find out the optimal number of channels in each layer of a deep well-trained but redundant CNN network. In order to prove the effectiveness of the proposed method, a deep CNN model for image super resolution (SR) and three models for image classification are employed. Compared with the previous work [12], our method achieves

larger reduction of computation and parameters under the same performance constraint. Furthermore, we also compare the experimental results with the state-of-the-art filter pruning method [10] to show that the proposed method can reduce more parameters. Specifically, our contributions are:

1) Exploring the redundancy of convolutional layers with their sensitivity and filter sparsity,
2) Proposing the Computation-Performance Optimization (CPO) method for systematically reducing computation operations under given performance drop constraint, and
3) Applying CPO on SR and image classification tasks, which leads to significant improvement of computation reduction.

II. Related Works

Several methods have been proposed in order to compress networks. Pruning is shown to be effective in reducing the network complexity and over-fitting. By eliminating weight connections, it sometimes even leads to performance gain. Early works such as Optimal Brain Damage (OBD) [7] and Optimal Brain Surgeon (OBS) [13] compute the saliency of each individual parameter through second order derivatives and remove those with lower saliencies. However, with the growing scale of modern network architectures, it becomes unrealistic to compute the saliency of every parameter. Therefore, Han *et al.* remove weights whose magnitudes are smaller than a certain threshold [8]. In addition to pruning, they incorporate weight sharing and Huffman coding to further boost the compression rate while still being able to retain the original performance at the same time.

Aside form pruning, fixed-point quantization is also one common method towards compressing and accelerating networks. Hung *et al.* show that sometimes the quantized network can outperform the original full-precision network [14]. Recent research include XNOR-net [15], in which both inputs and weights are binarized. They introduce a scaling factor, which is the *L1-norm* of the original inputs and weights. By multiplying binarized layers by the scaling factor, the accuracy drop is significantly decreased compared with other binary networks. Incremental Network Quantization [16] partitions weights in a single layer into different groups and incrementally quantizes the weights of the entire network to powers of two and zeros, resulting in better accuracy than the full-precision one. Besides finding the optimum quantization step size, Anwar *et al.* give the insight into layer-wise sensitivity by conducting experiments on quantizing one layer when keeping others in high precision [17].

Vector Quantization for quantizing the parameter values has also been used in compression, Gong *et al.* exploit multiple methods like scalar and product quantization with K-Means clustering for the weight parameters [18]. A unified Quantized Convolutional Neural Network (Q-CNN) framework [19] is proposed. They not only quantize the parameters in both convolutional layers and fully-connected layers, but minimize the estimation error of each layer's response. With the error correction mechanism, only minor accuracy drop was sustained.

From the perspective of filters within convolutional layers, Jaderberg *et al.* present an approximation of full rank filter banks as a combination of rank-1 filter basis and reduces the inference time [20]. Group-wise Brain Damage [21] revisits the concept of OBD and leverages the fact that convolutions are in practice matrix multiplications. They group together entries of the convolution filters and reduce them to zeros in a coordinated way. Anwar *et al.* introduce three levels of structured sparsity, which are channel wise, filter wise and intra filters strided sparsity when it comes to pruning weights and filters [22]. They also point out that other compressing techniques (e.g. quantization) are orthogonal to pruning, and will enable greater computation and storage savings.

Meanwhile, some works are dedicated to filter-wise removal [11], [10], [12], they point out that removing redundant filters to alter the network architecture can dramatically save the computation. Among those network compression and optimization techniques described above, filter removal is one of the methods with high potential since it can be used not only for model size compression but also for computation reduction for general computing platforms. In the well-performed work conducted by Li *et al.* [10], they use the *L1-norm* of every filter to rank the removing order. In addition, the number of filters to be removed in each layer is decided by observations and empiricism. Different from their *L1-norm* calculation and based on the sparsity definition in our previous work [12], we provide a well defined metric, performance sensitivity (PS), to measure the layer's sensitivity for filter pruning. With the guide of PS and the constraint of a given expected performance drop, we can layer-wisely remove sparse filters and fine-tune the model to find the suitable number of filters to remove for each layer. Finally, experiments on image classification and image super resolution are conducted to prove the effectiveness of the proposed method and our method can be comparable or even better than the state-of-the-art [10].

III. Proposed Method

The operations of the i-th CNN layer involve convolving a 3-D tensor (input, $\mathbf{x_i} \in \mathbb{R}^{C_i \times Y_i \times X_i}$) with N_i different 3-D tensors (filters, $F_{i,n} \in \mathbb{R}^{C_i \times H_i \times W_i}$) to extract different features and then generating a 3-D feature map tensor (output, $\mathbf{y_i} \in \mathbb{R}^{N_i \times Y_i' \times X_i'}$), where C_i, Y_i, X_i are channel, height and width of the i-th input tensor, H_i, W_i are height and width of one filter, and N_i is the number of convolution filters in the i-th layers, which is equal to C_{i+1}, the number of channels of the next layer. Y_i', X_i' are slightly different from Y_i, X_i owing to the boundary of convolution, and the output tensor is also the input tensor of the next layer. The filter pruning procedure is based on removing one complete filter of the i-th layer at a time, which reduces $C_i H_i W_i X_i' Y_i'$ operations. And furthermore, it will also eliminate one feature map channel at the next layer, so it will concurrently reduce $N_{i+1} H_{i+1} W_{i+1} X_{i+1} Y_{i+1}$ operations.

Given a well-pretrained CNN model with the common state-of-the-art architecture, in the network optimization process with filter removal, the number of filters and which filters to be removed among each layer are two important parameters

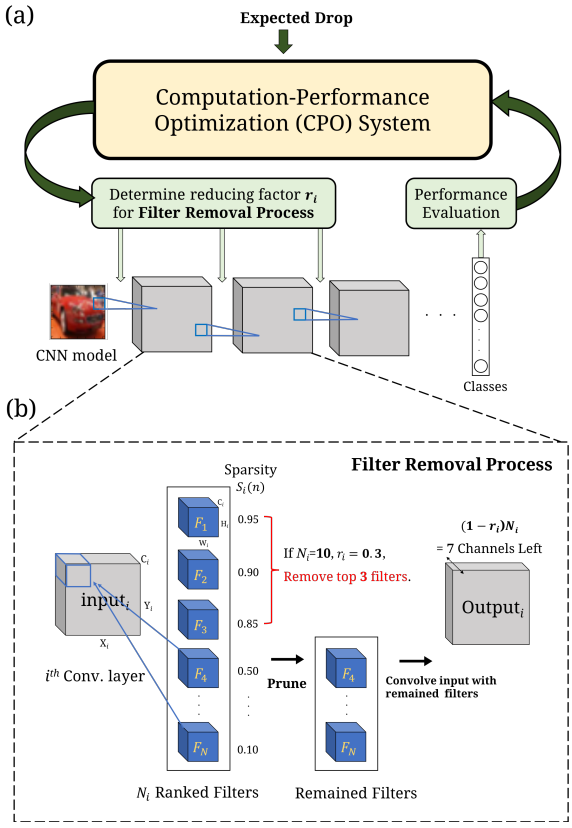

Fig. 1: **(a) Flow of Conducting CPO System. (b) Intra-layer Filter Removal Process.** Figure (a) illustrates the whole CPO pruning algorithm. Given an expected drop by the user, the system will iteratively prune the well-trained CNN model by determining the layer-wise reducing factors, and evaluate the model performance to start the next iteration. Figure (b) demonstrates the intra-layer filter removal process with a given reducing factor. We first rank the filters in the i-th layer by sparsity and remove the first $N_i r_i$ filters. When $N_i = 10$,$r_i = 0.3$, after pruning, 7 filters will exist and the output feature map will remain 7 channels, too.

we need to determine. Within one layer, we define the filter sparsity and rank the possible candidates to be removed; between layers, we propose a Computation-Performance Optimization (CPO) algorithm to take sparsity and performance sensitivity, which will be defined later, into consideration to iteratively determine the sequence of layer-wise reducing factors (the ratio of removed filters). Specifically, owing to different tasks or applications, the users may have an expected acceptable performance drop after model pruning. Therefore, we can adaptively prune the CNN network according to any expected drop. Fig. 1 (a) illustrates our CPO system. Given a trained CNN network and an expected performance drop by the user, the CPO system will iteratively determine the reducing factor of every convolutional layer for the "Filter Removal Process". After pruning and retraining, we will do "Performance Evaluation" and start the next CPO iteration to generate the next reducing factor. In the following subsections, we will introduce the proposed method from the intra-layer to the inter-layer parts.

A. Definition of Sparsity

The criterion of redundancy is defined layer-by-layer according to their weight distribution. For a specified layer i, we use M_i to represent the mean value of all absolute filter weights:

$$M_i = \frac{\sum_{n,c,h,w} |F_{i,n,c,h,w}|}{N_i \times C_i \times W_i \times H_i}, \tag{1}$$

where n, c, h, w are the indices of the filter tensor F. Then the Sparsity $S_i(n)$ of the n-th filter at layer i can be written as:

$$S_i(n) = \frac{\sum_{c,h,w} \sigma(F_{i,n,c,h,w})}{C_i \times W_i \times H_i}, \qquad \sigma(x) = \begin{cases} 1, \text{if } |x| < M_i \\ 0, \text{otherwise} \end{cases} \tag{2}$$

In other words, for a specific layer i, if a filter has several coefficients which are less than the mean value, $S_i(n)$ is close to 1, which means this filter is more redundant than others. We then rank the filters in the i-th layer in descending order according to their sparsity values. When we conduct CPO at the i-th layer afterwards, the filters ranked higher will be removed first.

B. Definition of Reducing Factor

Considering that the numb:qers of convolution filters vary from layer to layer, it is not convenient for us to compute the exact number of redundant filters when conducting CPO algorithm. We thus define the reducing factor r_i, $0 \leq r_i \leq 1$ for the i-th layer. The value of r_i is the ratio of the numbers of removed filters to all filters at the layer i. Fig. 1 (b) demonstrates an example of intra-layer filter removal process. For the i-th layer, there are $N_i = 10$ filters, and originally the output feature map will contain 10 channels. We first calculate the filter sparsity $S_i(n)$, and construct the ranked sparsity list. If we set $r_i = 0.3$, $N_i r_i = 3$ filters on the top of the ranked list will be removed and 3 channels of the output feature map will be removed as well.

C. Concept of Computation-Performance Optimization

The next step is to determine the reducing factors while considering the global effects of filter removal across layers. Inspired by the rate-distortion optimization (RDO) technique in video and image coding [23], [24], we propose the concept of computation-performance optimization (CPO) for CNN optimization. In video and image coding systems, RDO is the method to determine the optimal bit allocation to achieve the minimized distortion δ^* under a given bit-rate constraint R_c:

$$\delta > \delta^* \ \forall \ R < R_c \tag{3}$$

To solve this problem, in [24], a post-compression rate-distortion optimization (PCRDO) is proposed for image coding. An image is decomposed into several small coding unit

CU_i, and $\Delta\delta_i/\Delta R_i$ is calculated, where $\Delta\delta_i$ is the global distortion reduction when coding unit CU_i in included, and ΔR_i is the required bitrate for this coding unit. Given any λ, the set of coding units

$$\{CU_i \quad |\Delta\delta_i/\Delta R_i > \lambda\} \tag{4}$$

is an RDO solution under the total bit-rate

$$R_c = \sum\{R_i \quad |\Delta\delta_i/\Delta R_i > \lambda\}. \tag{5}$$

Inspired by PCRDO, we model the filter removal process as a computation-performance optimization (CPO) problem, that is, we would like to achieve the minimized performance drop D^* under a given computation budget ζ_c. A group of filters is then employed as the small unit, and the associated $\Delta D/\Delta\zeta$ is derived for selecting the filter to be removed. Similarly, the units with larger $\Delta D/\Delta\zeta$ are kept, that is, the units with smaller $\Delta D/\Delta\zeta$ are removed to achieve computation-performance optimization. Note that $\Delta D/\Delta\zeta$ can be viewed as a kind of performance sensitivity.

D. Definition of Performance Sensitivity

For one specific convolutional layer, we can deduce the potential redundancy of filters by calculating the filter sparsity with (2), and consequently we can first remove the filters with high sparsity. However, for the entire CNN model, we have no clear criterion for determining which layer we can conduct filter pruning first. Based on the concept of CPO, we define the Performance Sensitivity (PS) of the i-th convolutional layer with the change of reducing factor (r_i) and performance drop (D):

$$\mathrm{PS}_i(\Delta r_i, \Delta D) = \frac{\Delta D}{\Delta\zeta} = \frac{\Delta D}{\Delta r_i \times W_i \times H_i \times C_i}, \tag{6}$$

where D is the performance drop, which is a positive value, and ΔD is the change of drop between two pruning steps. Δr_i is the change of reducing factor, and the whole denominator part approximates the computation change $\Delta\zeta$ for removing a portion of filters in the i-th layer.

The Performance Sensitivity (PS) represents the aptness of being pruned for a layer. When conducting our Computation-Performance Optimization (CPO) with reducing factor, it is more likely to assign higher reducing factors to the layers with lower PS values.

E. Computation-Performance Optimization

The problem left now is to determine the exact number of filters allowed to be removed for each single layer without apparent performance drop. Based on the concept of CPO, the Performance Sensitivity (PS) is employed to balance the trade-off between "Computation Reduction" and "Performance Drop".

First, we need to find out the PS value of each layer. By emulating the method in finding layer-wise sensitivity [17], we iteratively set $r = 0.5$ to halve the number of filters in one layer and meanwhile keep the rest untouched. With those remained parameters, we retrain the model for few epochs to fine-tune the model until convergence. After obtaining the performance drop D_i, PS_i can be calculated with $\Delta r_i = 0.5 - 0 = 0.5$ and $\Delta D_i = D_i - 0 = D_i$ in (6), where the initial reducing factor and the initial drop are both zero. Finally, with this pruning test for each layer respectively, we can construct the sensitivity list for the subsequent steps. The reason why we choose half over other fractions to probe the sensitivity is two-fold. For one, if a too small portion of filters is pruned away, the performance will be quickly restored through the retraining process, resulting in unstable PS values. For the other, it inherently fits the property of Binary Search (BS) to find out the appropriate reducing factor for the consecutive procedures in the proposed CPO algorithm.

Second, with the PS list obtained by setting $r_i = 0.5$ for each layer i respectively, we sort it in ascending order and start removing filters from the least sensitive layer, which is called "the current layer" in following descriptions. Following the Binary Search order, we increase the reducing factor from $r_i = 0.5$ to $0.75, 0.875...$ for the current layer, unless one of the following conditions is met:

1) When the performance drop becomes intolerable.
2) When the updated PS value of the current layer becomes larger than that of the runner-up.
3) When the layer run out of filters to remove.

The performance drop is intolerable when it exceeds the expected drop D_{exp}, which is decided by the user of our CPO system. When it happens, we will take a step back by following the binary search order. That is to say, instead of removing $N_i r_{i,k}$ filters which causes unexpected drop for the k-th step, we remove $N_i \frac{(r_{i,k}+r_{i,k-1})}{2}$ filters. If it is still intolerable, we will search the suitable reducing factor for the current layer until the drop becomes smaller than D_{exp}. Next, we move on to removing the filters at the next least sensitive layer.

The second condition appears during the sensitivity updating within the current layer. The goal of updating the PS value is to evaluate the performance change owing to the computation reduction, or we can say the incrementally increased reducing factor. That is, for example, if $r_{i,k} = 0.75$ and $r_{i,k-1} = 0.5$ in the k-th and $(k-1)$-th steps, respectively, Δr_i is set as $0.75 - 0.5 = 0.25$ in (6). Note that PS will become higher and higher when we continue removing filters from one single layer, which is a kind of diminishing marginal utility. Therefore, when we detect that the PS value of the current layer has grown larger than the runner-up layer i' in the sensitivity list, it strongly suggests that this reducing factor influences the whole performance too much, and we will then switch to the next layer i'.

In addition, once we decide the number of filters to remove in the i-th layer, the PS value of the $(i+1)$-th layer in the original sensitivity list is also updated because of channel reduction. As mentioned in Sec. III-B, if we remove n_i filters in the current layer i, every filter in the $(i+1)$-th layer will also be reduced by n_i channels, which causes the reduction of computation at that layer. Therefore, we need to modify the denominator term C_{i+1} of (6), and it will simultaneously increase the PS_{i+1} value.

The procedure described above will iterate through all of the

layers that are available for filter pruning. For the experiments in SR, we will not prune the last convolutional layer because the number of filters at that layer is only one. Meanwhile for the experiments in image classification, the available layers for filter pruning in our CPO algorithm differ from model to model. It depends on the original architecture (VGGnet, Resnet [3] and so on) or whether the hidden layers exist between the last convolutional layer and the output layer. If there is only one linear layer which maps the feature map of the last convolutional layer to the class prediction, we will not remove the filters in the last convolutional layer because it will correspondingly remove some of the parameters in the next linear layer and meanwhile greatly influence the prediction performance. We will discuss the details in Sec. IV. The overall algorithm flow is shown as follows.

Alogorithm *Flow of CPO*

Given a trained CNN model.
Determine an expected performance drop D_{exp}.
Start probing the Performance Sensitivity :
repeat through every convolutional layer
    Prune the i-th layer with $r_i = 0.5$ and obtain the corresponding drop (D_i) after retraining few epochs.
    Calculate the Performance Sensitivity (PS$_i$) according to (6) and add it to PS list.
    Recover the i-th layer by setting $r_i = 0$
until all available layers are iterated through.
Start determining the actual r_i for every layer:
repeat through the ranked PS list
    Start pruning the i-th layer :
    loop with the BS order for $r_{i,k}$ starting from 0.5
        Prune the layer with $r_{i,k}$ for the k-th iteration and retrain the model for few epochs.
        Update the PS value and check the termination conditions.
        if it meets the conditions 1) or 2). **then**
            repeat step back to $r_i = \frac{(r_{i,k}+r_{i,k-1})}{2}$
            until the drop is acceptable.
            break
        else if it meets conditions 3). **then break**
        else
            Go to next iteration with larger reducing factor.
        end if
    end loop
    Update the PS list owing to channel reduction.
until All layers in PS list are iterated through

F. FLOP and Parameters Calculation

After conducting CPO, we can significantly remove a large number of parameters, which will result in smaller storage size and less floating point operation (FLOP). To quantify the operations remained in all convolutional layers, we follow the equations:

$$FLOP = \sum_i N_i \times (W_i \times H_i \times C_i) \times (X_i \times Y_i), \qquad (7)$$

where W_i, H_i, and C_i are the width, height, and number of channels of N_i filters in the i-th layer respectively, while X_i and Y_i are the width and height of the convolved input.

To calculate the parameters in the meantime, we just remove the $(X_i \times Y_i)$ terms of the shifting window operations of convolution in (7), and thus we can obtain the parameter size inside all convolutional layers.

IV. Experimental Results

We conduct experiments on two tasks to demonstrate our CPO algorithm, which are image super-resolution and image classification. For image super-resolution, we employ the residual CNN model in Very Deep Super Resolution [6] (VDSR). This model is constructed only with convolutional layers; therefore, the model size and the computation time will not be influenced by the fully-connected layers. As for image classification, we construct a modified VGG-19 model [2], a self-designed Resnet-32 model and a self-designed Mobilenet-22 model, which are modified from Resnet-34 [3] and Mobilenet-v1 [25], respectively. These three networks are dedicated models for training and testing on Cifar-10 dataset [26]. Cifar-10 is a small dataset including 10 categories, and all of which are composed of 3-channel RGB images with the resolution of 32 $\times$ 32. The following experiments will be conducted on the four models.

a) ***Performance and Computation Comparison with baseline method:*** For the tasks mentioned above, we first construct a baseline method as what was done in our previous work [12], called **Uniform Removal (UR)**. UR will remove filters with a fixed reducing factor r_i across all available layers. After that, we retrain for few epochs to recover the performance. The number of retraining epochs may be slightly larger than that in [12] because we find that the performance drop will be stable if we make the retraining stage converge. To prove the effectiveness of our method, we perform CPO on the original model and set the expected drop D_{exp} as the validation drop obtained from UR. We then compare the remaining parameters and FLOP between CPO and UR. We also conduct some experiments with D_{exp} set as other values to observe the trade-off between performance and computation. The expected drop and the drop monitored in CPO algorithm are all tested with the validation set, which is seen but not trained during the procedure. After our CPO algorithm, we will test the model with a totally unseen testing set to evaluate our model performance. It is worth noticing that once one filter is removed, the number of channels in every filter of the next layer will consequently decrease by one. This is the reason why the actual parameters removed will be more than the percentage of filters we attempt to remove in UR. In addition, we conduct CPO experiments from lower D_{exp} to higher one, and we will use the model pruned by lower D_{exp}

TABLE I: **SCALE-sim Hardware Configuration.** This table shows the configurations of our systolic array simulator. IFmap and OFmap means Input and Output Feature map.

Array Height	Array Width	Data Flow
14	12	Output Stationary
IFmap SRAM Size	**OFmap SRAM Size**	**Filter SRAM Size**
70 KB	40 KB	108 KB

as the base model to continue the next experiment of higher D_{exp}.

*b) **Hardware Simulation**:* Our proposed method is trying to reduce the entire convolutional filters that have less contribution to the network. Therefore, it genuinely reduces not only the parameters but also the FLOP when performing on any hardware platform. We simulate the operations on the Systolic CNN AcceLErator Simulator (SCALE-sim) proposed by ARM [27] to prove the reduction of computation. Table I shows the simulation configurations of the systolic CNN array. This tool can help generate the computation cycles and the DRAM read/write bandwidth. In the following sections, we will only show the total cycle count of network inference stage for simplicity.

*c) **Shallow and Deep Model Comparison**:* Aside from comparing the results between UR and CPO, we also design a shallow model with comparable parameters to the deep model pruned by our CPO system. The results will show the trade-off between training time and the performance of the model.

A. Fully Convolutional VDSR network

We obtain the well-trained VDSR model from the official website [28]. Because we have no knowledge of which validation set the model was originally validated on, we choose Set5 [29] as our validation set and Set14 [30] as the final testing set, both with $\times 2$ scale. These two datasets are commonly used in image super-resolution tasks.

The model structure is a 20-layer residual CNN as illustrated in Fig. 2. The input is an interpolated low-resolution (ILR) image with one channel (Y channel), and the output is the derived high-resolution (HR) one. During training and validation, for convenience, we will use input images with sizes of 41×41 that are randomly cropped from the dataset, which is same as the settings in VDSR. When performing testing phase, the input and output sizes can be arbitrary depending on the testing image. Among the convolutional layers, each of the first 19 layers has 64 filters, but only one filter exists in the last layer to generate the residual part, which is added by the low resolution image to finally generate the high-resolution one. Since the last layer of VDSR has only one filter, we perform filter-pruning on the rest of the layers. There are two main reasons for conducting our experiments on VDSR. First, the fully-convolutional model can help us clearly evaluate the performance of our filter removal method. Second, since the task of SR is difficult in computer vision, we are interested in finding the redundancy of an SR model.

TABLE II: **Experimental results of VDSR.** This table shows the settings and performance results of the original model, the models pruned after UR and our proposed CPO. The number of retraining epochs after pruning is 5. Set14 is our unseen testing set and the last column is the cycle count after our SCALE-sim CNN hardware simulator.

Original Model				
Reducing Factor	Val PSNR (dB)	Params / FLOP	Set14 PSNR (dB)	Latency (Cycle)
0	40.26	6.7×10^5/ 1.1×10^9	33.08	7.6×10^6
Uniform Removal (UR) [12]				
Reducing Factor	Val Drop (dB)	Params / FLOP Remained (%)	Set14 Drop (dB)	Latency (Cycle)
0.0625	0.10	87.91 / 87.91	0.11	5.9×10^6
0.1250	0.13	76.60 / 76.60	0.19	5.5×10^6
0.2500	0.26	56.31 / 56.41	0.29	3.8×10^6
CPO (Ours)				
D_{exp} (dB)	Final Val Drop (dB)	Params / FLOP Remained (%)	Set14 Drop (dB)	Latency (Cycles)
0.10*	0.08	**72.31 / 72.31**	0.12	$\mathbf{5.2 \times 10^6}$
0.13*	0.14	**64.68 / 64.68**	**0.15**	$\mathbf{4.7 \times 10^6}$
0.18	0.18	54.25 / 54.25	0.2	4.0×10^6
0.26*	0.26	**51.52 / 51.25**	**0.28**	$\mathbf{3.8 \times 10^6}$
0.32	0.32	45.85 / 45.85	0.34	3.4×10^6

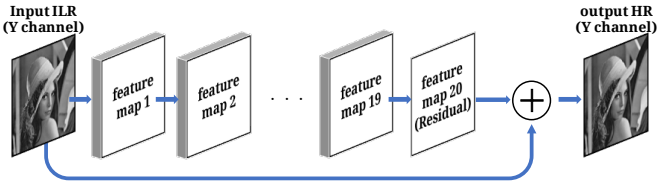

Fig. 2: **Network structure of VDSR.** The 20-layer residual network is composed of 20 times convolutions and nonlinear operations. There is no pooling layer, so the output of the residual part has the same size as the input. ILR means Interpolated Low-Resolution and HR represents High-Resolution.

Table II shows the experimental results of pruning VDSR network. We use PSNR (dB) to evaluate the performance. The first four columns are the model settings and performance results. The last column is the summed computation latency calculated in cycle count after performing the network inference on one input image, which has the size of 41×41 in this case, with the SCALE-sim neural network simulator. In all tables, "Val" means validation and "Params" represents parameters. First, the upper part is the performance of the original trained model, which achieves $40.26dB$ on the validation set and $33.08dB$ on the Set14 testing set. The baseline UR results are in the middle. Owing to the unchanged size of the feature maps among the VDSR network, the percentage of parameters and FLOP remained, which are calculated by

(7), are the same. Note that the PSNR drop of the results are slightly different from those in [12] because we conduct 5 retraining epochs instead of 3 as mentioned above. We show three results with different reducing factors. When we increase the reducing factor, the remaining parameters decrease but the performance will also drop accordingly. At the bottom part of Table II, it shows the results with our proposed CPO. The first column is the user expected drop (D_{exp}), and the second column is the final validation drop after the last pruning and retraining iteration. Note that the three settings whose expected drops are marked with ($*$) correspond to the three baseline UR methods. It can be seen that CPO can achieve results superior to UR for every reducing factor. With comparable PSNR drop on unseen testing set, CPO is able to achieve more computation reduction. It can save about 50% of parameter storage and computation with minor performance drop (Val:0.26dB/Test:0.28dB). In addition, it also shows that about 30% of the computations are redundant given only a negligible drop (Val:0.05dB/Test:0.12dB). For a model purely containing convolutional layers, our method will surely alleviate the computational burden on the hardware. Furthermore, we also conduct two other expected drop settings to observe the trend between performance drop and computation reduction. We find that we only get marginally additional computation reduction when we have already removed the majority of redundancy in the model. The reason we speculate is that the difficult SR task is a regression task rather than a classification task, so there might be not so much redundancy in the VDSR model. However, we can still eliminate 15% more of the parameters than UR method did when there is a negligible drop.

Designing a shallow model can also reduce the computation cost. However, a non-deep network may sometimes fail to perform well. Table III shows our experimental results. The shallow VDSR model is composed of 10 convolutional layers, and the remained parameters are comparable to the model pruned after CPO with the settings of $D_{exp} = 0.32$. We can see that our method can perform well on the validation and testing sets. We use the number of training epochs to evaluate the training time. Although our method can perform better, the trade-off is that it may take more time to retrain the model with conducting CPO than simply train the shallow model from scratch. In addition, the retraining time is also higher than that in UR (5 epochs) which is mentioned in Sec. IV. we think that increased retraining time is not the main consideration. The purpose of our algorithm is to reduce the burden when we deploy the model on some hardware devices at last. Therefore, we can use a powerful GPU first to eliminate the redundancy as possible as we can by reasonable offline training, and then deploy the model on those hardware devices.

B. VGG-19 on Cifar-10 Image Classification

The following three sections are the experiments on Cifar-10 image classification task with three dedicated neural network models. The purpose of all our experiments is to observe the impact of removing filters in convolutional layers. Hence, there are no hidden fully-connected (FC) layers in our self-designed models. We only use one linear layer to map the flattened feature map after the last pooling layer to the classes prediction.

TABLE III: **Comparison of Shallow Model and Pruned Deep Model (VDSR).** This table shows the trade-off between training a shallow model and pruning a given well-trained deep model. We choose the CPO results with $D_{exp} = 0.32$ to do the comparison. It shows that we can get better performance with CPO but it takes more retraining time because of the iterative pruning.

	Val PSNR (dB)	Params / FLOP Remained (%)	Set14 PSNR (dB)	Training time(epochs)
Original	40.26	100 / 100	33.08	100
Shallow	39.82	44.54 / 44.54	32.62	**80**
CPO(0.32)	**39.94**	45.85 / 45.85	**32.74**	305

TABLE IV: **Architecture of our modified VGG-19 network.** There are five parts and 16 convolutional layers in our model. The number of filters for each convolution are shown inside the parentheses at the first row. Note that there is only one linear layer instead of three to map the feature vector to the output prediction.

Conv1(64)	**Conv2**(128)	**Conv3**(256)	**Conv4**(512)	**Conv5**(512)
conv1-1	conv2-1	conv3-1	conv4-1	conv5-1
conv1-2	conv2-2	conv3-2	conv4-2	conv5-2
maxpool(2)	maxpool(2)	conv3-3	conv4-3	conv5-3
		conv3-4	conv4-4	conv5-4
		maxpool(2)	maxpool(2)	maxpool(2)
				output(10)

The first self-designed model is a modified VGG-19 network, which contains 16 convolutional layers and one linear layer. Furthermore, we preserve all the filters in the last convolutional layer in order not to accordingly influence the last output layer, which contains less parameters but plays a role in the feature-to-class transformation. In summary, filter-pruning will only be performed on the first 15 layers in our modified VGG-19 network to gain undistracted insight into how the removal of filters in convolutional layers affects the performance. Table IV briefly illustrates the modified VGG-19 network. There are five convolution parts, and the settings in each part such as the number and sizes of convolution filters are all the same as the original network. The final output layer which contains 512×10 parameters will map the feature vector to the 10 classes output. Moreover, we implemented Batch-Normalization (BN) [31] after every convolution layer, which are not shown in the table. BN layers store additional statistical information of the preceding feature maps and also contain the parameters of linear shifting operation; therefore, the removal of filters will result in corresponding reduction of the variables in BN layers, too.

The experimental results of VGG-19 on Cifar-10 are shown in Table V. We randomly choose ten percent of the training data as our validation set. Then, we train the unpruned VGG-19 network by ourselves. It achieves 93.16% validation accuracy and 92.98% testing accuracy. Based on the model, we conduct both UR and our CPO algorithm. Same as the

TABLE V: **Experimental results of VGG-19.** This table shows the results of unpruned and pruned VGG-19 models tested on Cifar-10. The number of retraining epochs after every pruning iteration is 8. The last column is the cycle count after our SCALE-sim CNN hardware simulator.

Original Model				
Reducing Factor	Val Acc (%)	Params / FLOP	Testing Acc (%)	Latency (Cycle)
0	93.16	2.0×10^7/ 3.9×10^8	92.98	3.7×10^6
Uniform Removal (UR) [12]				
Reducing Factor	Val Drop (%)	Params / FLOP Remained (%)	Testing Drop (%)	Latency (Cycle)
0.250	1.60	58.47 / 56.78	1.25	2.0×10^6
0.375	2.34	41.84 / 39.72	1.89	1.4×10^6
0.500	3.72	27.96 / 25.70	3.52	9.5×10^5
0.625	5.32	16.84 / 14.72	4.01	5.1×10^5
CPO (Ours)				
D_{exp} (%)	Final Val Drop (%)	Params / FLOP Remained (%)	Testing Drop (%)	Latency (Cycles)
0.25	0.14	28.57 / 54.09	0.50 (0.40)	1.6×10^6
0.58	0.44	15.68 / 42.26	0.83 (0.76)	1.2×10^6
1.60*	1.58	**10.23 / 28.11**	1.83 (1.34)	$\mathbf{7.9 \times 10^5}$
2.34*	2.16	**5.75 / 20.18**	2.49 (**1.73**)	$\mathbf{5.5 \times 10^5}$
3.72*	3.58	**2.49 / 16.88**	4.44 (**2.72**)	$\mathbf{4.5 \times 10^5}$
5.32*	5.18	**2.07 / 13.74**	6.29 (**3.91**)	$\mathbf{3.5 \times 10^5}$

notation in Table. II, we use $(*)$ to represent the value of D_{exp} which is set as the "Val Drop" evaluated after performing UR.

It can be seen in Table V that the remaining parameters of the new architecture derived from our CPO algorithm are radically less than that of UR. Notice that the reduction in FLOP of the models derived by CPO are not as much as the reduction in parameters. The explanation is that owing to the low PS values of the last few layers in VGG-19, as Fig 3 shows, we prefer to remove the filters at those layers. However, the X_i, Y_i of those feature maps are small because we've gone through many pooling layers. Therefore, the overall reduction in FLOP will not be as much as that in parameters. Next, for the testing drop in the CPO part, the original performance after retraining for 8 epochs in the final iteration is a little less than the results of UR. Our conjecture is that the models suffer much more parameter reduction after CPO than after UR; therefore, we need more retraining epochs for the last iteration in CPO to recover the performance. Hence, the number in the parentheses on the right side is the drop after retraining for 40 epochs. It shows that the model can recover to the comparable performance as expected. We also conduct two CPO experiments to observe the redundancy when given negligible drops. We find that almost 70% parameters can be removed in the redundant VGG-19 network.

Tabel VI also shows the performance of training a shallow model. The model is constructed with 5 convolutional layers, where each layer contains 64, 128, 256, 512 and 512 filters. Cifar-10 is less complicated in comparison with other large image classification dataset, for which VGG-19 is originally

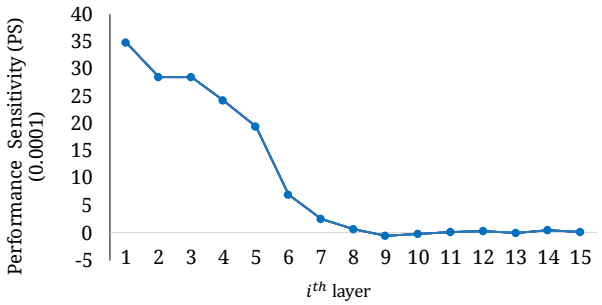

Fig. 3: **VGG-19 Performance Sensitivity List.**

designed. Therefore, the shallow model with only 5 convolutional layers can even achieve 89.56% testing accuracy. This time we choose the CPO results with comparable performance (90.26%) to compare the remaining computation. We can claim that our CPO is able to detect a great amount of redundancy in VGG-19 and remove almost 97% of parameters. However, same as the VDSR experiment stated above, we need to spend more training time to achieve this performance. In details, we use one Geforce 1080Ti GPU to train the Cifar-10 dataset, and it only takes less than fifteen seconds to train for an epoch. Therefore, in practice, our CPO method takes reasonable training time for some small dataset.

C. Resnet-32 on Cifar-10 Image Classification

It has been a well-known fact that VGG-19 contains great amount of redundancy. Therefore, it can be easily pruned without apparent performance drop. To prove the effectiveness of our CPO, we modify the original architecture of Resnet-34 [3] and design a dedicated Resnet-32 to train on Cifar-10. Resnet [3] is recently one of the most powerful networks, and meanwhile it contains less parameters than the VGG network. Our Resnet-32 possesses 31 convolutional layers and 1 FC layer to map to 10 classes. Among the convolutional layers, except for the first layer, every 2 layers constitute a residual block. There are three stages presented in the model, and each contains 10 convolutional layers. The sizes of output feature maps at the end of each stages are 32×32, 16×16, and 8×8. We utilize 1×1 convolutions to deal with the conflict when the input and output of a residual shortcut have different channel dimensions, which is also the design adopted by the

TABLE VI: **Comparison of Shallow Model and Pruned Deep Model (VGG-19).** This table also shows the trade-off between shallow and pruned deep model. We choose one CPO results with comparable performance to compare the computation remained.

	Params / FLOP Remained (%)	Testing Acc (%)	Training time (epochs)
Original	100 / 100	92.98	300
Shallow	19.51 / 17.17	89.56	**200**
CPO(3.72)	**2.49 / 16.88**	90.26	620

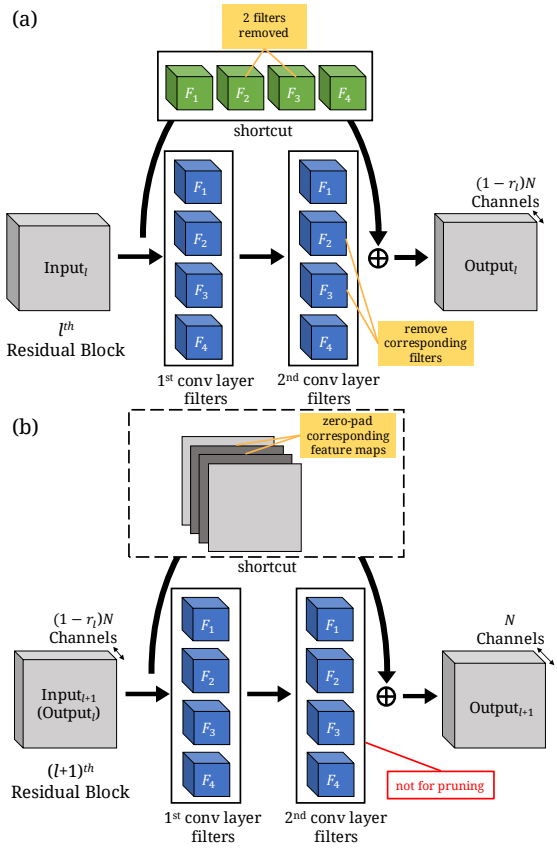

Fig. 4: **(a)Shortcut Convolution Pruning. (b)Shortcut Identity Padding.** For the residual block with convolution shortcut (a), we first remove the filters in the shortcut and then remove the corresponding filters. For the identity mapping after the shortcut convolution (b), we will do zero-padding to make the dimension of feature maps consistent.

original Resnet model. In details, our Resnet-32 only contains approximately 9% parameters compared to the above VGG-19 network.

It is worth noticing that not every convolutional layer is free to be pruned. On one hand, for a residual block with input and output of identical dimensions, the shortcut is an identity mapping. Hence we do not prune the second layer of the residual block with an eye to maintaining the number of channels of the output feature map, which will be added with the identity mapping. On the other hand, same as the method proposed in [10], when a residual block has input and output of different dimensions, we remove filters in the 1×1 shortcut convolution with a given reducing factor, and then remove the filters in the second layer of the residual block according to the indices removed in the shortcut convolution. Fig. 4 (a) demonstrates this pruning process, our system will first determine the filter be removed in the shortcut convolution, like the second and third filters, and subsequently remove the corresponding filters in the second convolution layer to avoid dimension conflict. Additionally, when filters are removed in a residual block with convolution shortcut, the identity mapping of the next residual block will cause a conflict. Fig. 4 (b) shows the situation. The channel of the input feature map is

TABLE VII: **Experimental results of Resnet-32.** This table shows the results of unpruned and pruned Resnet-32 model tested on Cifar-10. The number of retraining epochs after every pruning iteration is 8. The last column is the cycle count after our SCALE-sim CNN hardware simulator.

Original Model				
Reducing Factor	Val Acc (%)	Params / FLOP	Test Acc (%)	Latency (Cycle)
0	94.66	1.9×10^6/ 2.8×10^8	94.58	1.8×10^6
Uniform Removal (UR) [12]				
Reducing Factor	Val Drop (%)	Params / FLOP Remained (%)	Test Drop (%)	Latency (Cycle)
0.125	0.64	85.34 / 76.60	0.86	1.5×10^6
0.250	1.44	71.30 / 56.31	1.25	9.6×10^5
0.375	1.88	57.89 / 39.14	1.89	7.7×10^5
0.500	2.84	45.09 / 25.08	3.52	5.3×10^5
CPO (Ours)				
D_{exp} (%)	Val Final Drop (%)	Params / FLOP Remained (%)	Test Drop (%)	Latency (Cycles)
0.25	0.32	74.92 / 81.88	0.61 (0.30)	1.5×10^6
0.64*	0.48	**60.27 / 60.88**	0.89 (**0.56**)	$\mathbf{1.1\times10^6}$
1.44*	1.28	**38.11 / 42.63**	2.16 (**1.47**)	$\mathbf{8.2\times10^5}$
1.88*	1.68	**30.62 / 36.81**	2.61 (**1.84**)	$\mathbf{7.1\times10^5}$
2.84*	2.78	**23.98** / 26.31	3.54 (**3.06**)	5.4×10^5

reduced owing to the filter removal of the preceding layer. Because we will not prune the second convolution layer as mentioned above, there will be a conflict on the identity mapping. Therefore, we will zero-pad the feature map to match the dimension when performing identity addition. Considering the situations about dimension consistency above, the first layer and the FC layer are intact as well. Therefore, there is a limitation on the maximum portion of parameters that can be pruned.

The experimental results can be found in Table. VII. All the settings are the same as VGG-19, except that the testing drop in the parentheses are the drop retraining for 100 epochs after the final CPO iteration. We speculate that Resnet-32 model involves more complicated operations (shortcut convolution and zero-padding); therefore, it needs more epochs to recover the performance. Still, we can see that all the computation reduction done by CPO performs better than that by UR, except one point where the remaining parameters are less than UR but FLOP is a little bit more. This is also caused by removing most of the parameters at the last convolution stage. In summary, given a well-performed Resnet model (Test Acc:94.58%) with less parameters, although we can not remove as much parameters as we did for VGG-19 network, we still can eliminate 30% of parameters with negligible drop (0.30%) and remove almost 80% of weights with an accuracy that is still higher than 90%.

The comparisons of the performance with shallow networks in Sec. IV-A, IV-B show that a deep but "thin" network, which

TABLE VIII: **Comparison of fine-tuning and random initialization of pruned Resnet-32 on Cifar-10.** This table shows that fine-tuning the model after CPO can perform better and more effective than training the model from scratch of the same model architecture with random weight initialization.

	Extra epochs	Val Acc (%)	Test Acc (%)
CPO (1.44)	100	93.76	93.11
Random Initialization	100	92.14	91.60
	260	93.70	93.20

means the number of filters in each layer is less than those in the original model, perform better than a shallow network with comparable parameters. We now conduct another experiment on Resnet-32 to discuss the importance of first conducting CPO on the original deep but "fat" model and then fine-tune on it. Without CPO, one can exhaustively try different light-weight architectures, randomly initialize the weight and train the model from scratch to meet the expected performance metrics. However, it is not efficient for optimization and the performance may not be as expected. Table. VIII shows the results. The first row is the performance of our CPO with $D_{exp} = 1.44$. As the experiments conducted above, the pruned architecture found by CPO will be fine-tuned for extra 100 epochs to recover the performance. 93.76% and 93.11% are the validation and testing accuracy after fine-tuning. The next two rows are the experiments performed on the same architecture of that in **CPO (1.44)** but with random initialization on the weights. It shows that if we train from scratch with the same number of epochs, the performance is still far from the result after CPO. Not until the 260-th epochs does it reach the comparable performance to ours. Therefore, initializing the model with the original weights helps us speed up the training process and at the same time retain excellent performance.

D. Mobilenet-22 on Cifar10 Image Classification

Last year, a new architecture called Mobilenet [25] has been proposed. Mobilenet is distinguished for its ability to cut down on the number of parameters in convolutional layers in comparison with conventional networks. By substituting standard convolutions with depth-wise separable convolutions, which can be done with depth-wise filters following point-wise filters (1×1 convolution), the number of parameters are dramatically reduced. To further inspect the efficacy of our proposed method performing on those special convolution operations, we also apply it to a Mobilenet that we designed for Cifar-10 called Mobilenet-22. Our Mobilenet-22 has 21 convolutional layers and 1 FC layer. Except for the first layer in the convolutional layers, every 2 layers compose a depth-wise separable filter. Mobilenet-22 is a light-weight model with approximately 7% of parameters compared with VGG-19. We will try to explain the pruning mechanism in the following discussion. The filters that perform point-wise convolutions are the major targets of our pruning process. However, the removal of these filters will result in channel reduction in the output feature map and induce additional parameter reductions of the next layer in two ways. For one, the following depth-wise filters with corresponding indices will be removed. For the other, because of a shallower input feature map at the next layer, the following point-wise filters will become shallow, too. We show the experimental results in Table. IX. There are no hardware simulation latency for the Mobilenet-22 because the SCALE-sim simulator does not support special depth-wise convolution. Therefore, we only use FLOP to represent the computation complexity. Because our Mobilenet-22 is a light-weight model without residual connections, the unpruned well-trained model can not perform as well as Resnet-32, and it only achieves 90.78% testing accuracy. We can see that in order not to perform worse than 90% testing accuracy, we can only remove about 60% parameters with the UR baseline method. By conducting the proposed CPO, we can surprisingly reduce 88% parameters and 70% operations. Note that the negative sign in the testing drop means that the performance even becomes better than the original one, and the drop in parentheses represents the performance with 40 additional training epochs after the final CPO iteration. We find that we can approximately reduce half of the FLOP in our Mobilenet-22 network and the performance ($D_{exp} = 0.2$) is still maintained.

TABLE IX: **Experimental results of Mobilenet-22.** This table shows the results of unpruned and pruned Mobilenet-22 model tested on Cifar-10. The number of retraining epochs after every pruning iteration is 8. The negative sign in the testing drop column means the rise of the performance.

Original Model			
Reducing Factor	Val Acc (%)	Params / FLOP	Testing Acc (%)
0	92.22	1.3×10^6/ 2.9×10^7	90.78
Uniform Removal (UR) [12]			
Reducing Factor	Val Drop (%)	Params / FLOP Remained (%)	Testing Drop(%)
0.125	0.20	76.83 / 77.33	0.13
0.250	0.22	56.70 / 57.57	0.12
0.375	0.68	39.63 / 40.71	0.79
CPO (Ours)			
D_{exp} (%)	Final Val Drop (%)	Params / FLOP Remained (%)	Testing Drop (%)
0.10	0.08	67.40 / 88.25	0.01 (-0.01)
0.20*	0.06	**40.53 / 50.57**	**-0.01 (-0.01)**
0.22*	0.12	**15.54 / 34.21**	**0.10 (-0.05)**
0.68*	0.64	**11.56 / 28.62**	**0.54 (0.25)**

After conducting experiments on the three self-designed networks for Cifar-10 image classification, we can claim that no matter what kind of architectures, such as residual connection or 1×1 point-wise convolution, our CPO can alter the structure effectively according to the complexity of the task and the expected acceptable performance drop D_{exp} given by the user. In addition, unlike some works resulting in sparse weight matrix, our CPO can truly alleviate the computation

TABLE X: **VGG-16 Comparison between CPO ($D_{exp} = 0.1$) and [10].** Both experiments use VGG-16 trained on Cifar-10 as the targeted model and retrain 40 epochs afterwards. CPO achieves more reduction in parameters and FLOP and meanwhile maintains the performance.

Model	Error	FLOP Remained	Params Remained
VGG-16 [10]	6.75%	3.13×10^8 (100%)	1.5×10^7 (100%)
Pruned [10]	6.60%	2.06×10^8 (65.8%)	5.4×10^6 (36.0%)
VGG-16 (ours)	6.48%	3.13×10^8 (100%)	1.5×10^7 (100%)
Pruned (CPO)	6.66%	$\mathbf{1.97 \times 10^8}$ **(53.0%)**	$\mathbf{3.3 \times 10^6}$ **(22.1%)**

burden on the CNN accelerator.

For image classification, we use Cifar-10 dataset to quickly demonstrate the efficacy of our proposed CPO method. The next section we will compare with the state-of-the-art filter pruning method [10] with two standard models proposed in their experiments, which are a VGG-16 model that is also trained on Cifar-10 dataset and a Resnet-34 model trained on Imagenet dataset.

E. Comparison with the State-of-the-Art

Recently, the outstanding work [10] utilized similar concept of filter-pruning and achieved impressive compression rate. They determine which filters to be pruned within a single layer L_i by calculating the *L1-norm* of each filter $F_{i,j}$ in that layer, i.e. the *L1-norm* of filter j in i-th layer is $s_j = \sum |F_{i,j}|$. And they remove those with smaller s_j first. In addition, they decide the number of filters to be pruned for each layer based on observations and empiricism. They independently prune each layer by different numbers of filters and respectively inspect their performance on the validation set. According to the layer's resilience to filter pruning, they empirically assign the suitable number of filters to be pruned for each stage of convolutional layers, where the convolutional layers in the same stage have the same size of feature maps. Therefore, layers in the same stage will have the same number of filters left afterwards.

We choose two networks which are commonly used and also in their experiments, VGG-16 and Resnet-34, to demonstrate our CPO method and show the performance and computation comparisons. Their VGG-16 for Cifar-10 dataset possesses 13 convolutional layers and 2 FC layers, which are composed of a hidden layer and an output layer. When performing CPO on this model, there are two slight differences between the settings of our modified VGG-19 network discussed in Section IV-B and theirs. First, the last convolutional layer of VGG-16 is available for filter pruning because the next layer is not the output layer. The other one is that the first FC layer will participate in the retraining stage because of the corresponding reduction of parameters by removing filters at the last convolutional layer.

Table X shows the performance of the original and pruned models. The upper part is the results in [10], and the lower part is ours. In order to have the same representation in their experiment, we use "**Error**" to represent miss classification rate (MCR). Note that their unpruned VGG-16 model [10] is unreleased, so we train a model from scratch with exactly identical architecture, which results in a slight difference of the performance error. In addition, there is no expected performance drop (D_{exp}) in their method; therefore, we perform our CPO method with D_{exp} set as $\mathbf{0.1}\%$. The result shows that CPO achieves more reduction in parameters and FLOP. Moreover, the error performance of the CPO-pruned model (6.66%) is also comparable to theirs (6.60%). In Table XI, we present the final architecture of VGG-16 derived from CPO. We list the original structure on the left side and the required FLOP for reference. On the right side are the new architecture and the FLOP pruned of every layer by our proposed CPO. It can be seen that the last few layers are more inclined to be removed, which is consistent with the previous VGG-19 experiments and the VGG-16 PS list shown in Fig 5.

Next, the Resnet-34 network trained on famous Imagenet dataset is the other pruned target. Imagenet contains one million training images with one thousand label classes. There-

TABLE XI: **Model structure of VGG-16 derived from CPO ($D_{exp} = 0.1$).** The unpruned VGG-16 model structure and the pruned one are listed along. The last few layers are more inclined to be pruned because of low PS values.

layer type	#Filters (unpruned)	#FLOP (unpruned)	#Filters Remained	FLOP pruned (%)
Conv1	64	1.8×10^6	48	25.0%
Conv2	64	3.8×10^7	60	30.0%
Conv3	128	1.9×10^7	128	0%
Conv4	128	3.8×10^7	124	3.2%
Conv5	256	1.9×10^7	256	4.2%
Conv6	256	3.8×10^7	256	0%
Conv7	256	3.8×10^7	256	0%
Conv8	512	1.9×10^7	256	50%
Conv9	512	3.8×10^7	192	81.2%
Conv10	512	3.8×10^7	56	95.9%
Conv11	512	9.4×10^6	192	95.9 %
Conv12	512	9.4×10^6	192	86.0%
Conv13	512	9.4×10^6	24	98.2%
FC1	512	2.6×10^5	512	95.3%
FC2	10	5.1×10^3	10	0%
Total		3.1×10^8		47.0%

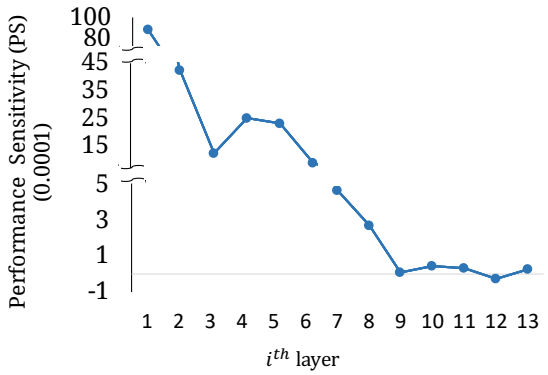

Fig. 5: **VGG-16 Performance Sensitivity List.**

fore, it is considered to be hard to discover the redundancy within the model that is trained on this large dataset. We perform our proposed CPO with the pruning method elaborated in Sec IV-C. Because of the great amount of training time required for one training epoch, we only conduct three epochs for every retraining stage and for every epoch we only randomly sample $1/3$ images for training. Table XII shows the experimental results of the two pruning methods. Notice that the performance of the two unpruned models may not be identical because of the different released sources, and our pretrained Resnet-34 model is obtained from Pytorch [32]. Obviously, the computation reduction of the two methods are not as much as that in VGG-16. By performing our CPO with $D_{exp} = 1.3\%$, we can discover 14.4% of redundancy for parameters and obtain less than 1% drop of the performance, which is better than the other. Nonetheless, when it comes to FLOP, our method only removes 15% redundancy. The reason is same that it tends to eliminate the filters at the last few layers because removing the filters in those layers harm the performance less. Compared with [10], they empirically decide the number of filters to be removed in every convolutional stage according to the layer sensitivity. Therefore, by removing the filters in the first few layers, the FLOP can be pruned more (75.8% remained) but at the same time, the classification performance can not be controlled. Conducting the CPO to Resnet-34 model on Imagenet dataset takes almost a week with two Geforce 1080Ti GPUs. Therefore, we are not able to conduct many experiments to progressively increase the expected drop to improve the performance. We believe that if we progressively increase the D_{exp} to a higher number and conduct the retraining stage for more than three epochs, like the 8 epochs in previous experiments but at the same time increasing the training time for the corresponding proportion, or conduct the experiments with paralleled and high bandwidth GPU devices in order to reduce the training time, we can obtain a model with less computation by our proposed CPO. In addition, for this kind of model trained on large dataset, besides conducting filter pruning to specifically alter the model architecture, we can still combine other methods like quantization to help deploy the model on edge devices.

Last, we try to analyze the hardware energy efficiency with the state-of-the-art [10]. Because there is no actual architecture of the model in [10], we utilize FLOP to represent the cycle count by observing that FLOP has positive correlation to the latency in previous experiments. We can deduce from the energy formula ($E = P \times T = CV^2 f \times T$) that the energy is proportional to the number of cycles ($f \times T$). If we can change the working frequency (f), under the same execution time and lower number of cycles, we can reduce the working frequency and supply voltage to get the quadratic saving. Therefore, if we have less cycle count for the model inference stage, we can design a more energy efficient hardware. Compared to [10], we have better energy efficiency on the VGG-16 model, but with inferior energy efficiency on the Resnet-34 model because of the remained FLOP. We believe that increasing the retraining time to 8 epochs can help make the model converge and then push our CPO to remove more filters among layers.

TABLE XII: **Resnet-34 Comparison between CPO ($D_{exp} = 1.3\%$) and [10].** Both experiments use Resnet-34 as the targeted model. CPO achieves more reduction in parameters and FLOP and meanwhile maintains the performance.

Model	Error	FLOP Remained	Params Remained
Resnet-34 [10]	26.77%	3.64×10^9 (100%)	2.16×10^7 (100%)
Pruned [10]	27.83%	$\mathbf{2.76 \times 10^9}$ **(75.8%)**	1.93×10^7 (89.2%)
Resnet-34 (ours)	26.68%	3.64×10^9 (100%)	2.16×10^7 (100%)
Pruned (CPO)	**27.66%**	3.11×10^9 (85.4%)	$\mathbf{1.85 \times 10^7}$ **(85.6%)**

V. Conclusion

In this paper, we present a Computation-Performance Optimization (CPO) method by removing filters in convolutional layers of a neural network. It utilizes Sparsity, Reducing Factor, and Performance Sensitivity to determine which and how many filters to prune in one layer. With an expected drop given by the user, CPO can effectively alter the model structure according to the complexity of the task. For super-resolution, it reduces more than 50% of parameters in VDSR but only causes about 0.28dB in performance drop. Furthermore, CPO is also proved to be efficient when applied to image classification. We conduct VGG-19, Resnet-32 and Mobilenet-22 (on Cifar-10) to demonstrate that it can eliminate great amount of parameters and FLOP without significant drop in accuracy. Compared with previous works, CPO provides a solution to determining the layer-wise hyperparameters of filter pruning, and achieves superior results.

VI. Future Works

In this work, we design an algorithm to remove the redundancy of deep neural networks by conducting it with strong GPUs offline. This method is proved to be helpful for alleviating the burden of hardware computation. In the future, we will deploy the pruned model on a real hardware system. We will design and compare the impact of the proposed method for different hardware architectures. Furthermore, online learning on the hardware devices is a new trend. No matter if it is a supervised or unsupervised learning scheme, it may need our pruning algorithm to further reduce some of the redundancy when learning on the new data. Therefore, we will design a pruning methodology conducted on the online learning devices. The relation of the performance between validation set and testing set is another issue we can discuss. Because our proposed CPO method will monitor the performance on the validation set to determine the reducing factor of each layer, the integrity of the validation set plays a role in the entire pruning process. We will put the experiment of observing the influence when the validation set is corrupted in our future work. Last, it is noted that the hardware design of CNN will be influenced by the scaling issues (ie. dropouts, batchnorm, number of layers in dense networks and number of filters). We focus on the number of filters in this paper and ~~we~~ will consider other issues in the future works.

ACKNOWLEDGMENT

This research was supported in part by the Ministry of Science and Technology of Taiwan (MOST 107-2633-E-002-001), National Taiwan University, Intel Corporation, and Delta Electronics.

REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proc. 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[5] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. 2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2980–2988.

[6] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proc. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 1646–1654.

[7] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage." in *Proc. Advances in Neural Information Processing Systems (NIPS)*, vol. 2, 1989, pp. 598–605.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[9] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[10] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[11] C.-F. Chen, G. G. Lee, V. Sritapan, and C.-Y. Lin, "Deep convolutional neural network on ios mobile devices," in *Proc. 2016 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2016, pp. 130–135.

[12] C.-T. Liu, Y.-H. Wu, Y.-S. Lin, and S.-Y. Chien, "Computation-performance optimization of convolutional neural networks with redundant kernel removal," in *Proc. 2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5.

[13] B. Hassibi, D. G. Stork *et al.*, "Second order derivatives for network pruning: Optimal brain surgeon," *Proc. Advances in Neural Information Processing Systems (NIPS)*, pp. 164–164, 1993.

[14] P.-H. Hung, C.-H. Lee, S.-W. Yang, V. S. Somayazulu, Y.-K. Chen, and S.-Y. Chien, "Bridge deep learning to the physical world: An efficient method to quantize network," in *Proc. 2015 IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2015, pp. 1–6.

[15] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. 2016 European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 525–542.

[16] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[17] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Proc. 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2015, pp. 1131–1135.

[18] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[19] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4820–4828.

[20] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[21] V. Lebedev and V. Lempitsky, "Fast convnets using group-wise brain damage," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2554–2564.

[22] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *Proc. ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[23] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 74–90, Nov 1998.

[24] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, Jul 2000.

[25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[26] A. Krizhevsky, "Convolutional deep belief networks on Cifar-10."

[27] ARM-Software, "Systolic cnn accelerator simulator (scale sim)," https://github.com/ARM-software/SCALE-Sim, 2018.

[28] J. K. Lee, "A caffe-based implementation of very deep convolution network for image super-resolution," https://github.com/huangzehao/caffe-vdsr, 2016.

[29] M. Bevilacqua, A. Roumy, C. Guillemot, and M. L. Alberi-Morel, "Low-complexity single-image super-resolution based on nonnegative neighbor embedding," 2012.

[30] R. Zeyde, M. Elad, and M. Protter, "On single image scale-up using sparse-representations," in *Proc. International conference on curves and surfaces*. Springer, 2010, pp. 711–730.

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[32] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.