Pruning Depthwise Separable Convolutions for MobileNet Compression

Cheng-Hao Tu
Institute of Information Science, Academia Sinica
Taipei, Taiwan
andytu28@iis.sinica.edu.tw

Jia-Hong Lee
Institute of Information Science, Academia Sinica
Taipei, Taiwan
honghenry.lee@iis.sinica.edu.tw

Yi-Ming Chan
Institute of Information Science, Academia Sinica
Taipei, Taiwan
yiming@iis.sinica.edu.tw

Chu-Song Chen
Institute of Information Science, Academia Sinica
MOST Joint Research Center for AI Technology and All Vista Healthcare
Taipei, Taiwan
song@iis.sinica.edu.tw

***Abstract*—Deep convolutional neural networks are good at accuracy while bad at efficiency. To improve the inference speed, two directions have been explored in the past, lightweight model designing and network weight pruning. Lightweight models have been proposed to improve the speed with good enough accuracy. It is, however, not trivial if we can further speed up these "compact" models by weight pruning. In this paper, we present a technique to gradually prune the depthwise separable convolution networks, such as MobileNet, for improving the speed of this kind of "dense" network. When pruning depthwise separable convolutions, we need to consider more structural constraints to ensure the speedup of inference. Instead of pruning the model with the desired ratio in one stage, the proposed multi-stage gradual pruning approach can stably prune the filters with a finer pruning ratio. Our method achieves satisfiable speedup with little accuracy drop for MobileNets. Code is available at https://github.com/ivclab/Multistage_Pruning.**

***Index Terms*—Network Pruning, Lightweight CNN**

I. Introduction

Convolution neural networks (CNNs) have become an effective and well-developed technique that achieves state-of-the-art performance on many tasks. However, to gain better accuracy, the architectures of CNNs tend to be larger or deeper, which makes the CNN models infeasible to be realized on resource-limited devices such as home robots and mobile phones.

To overcome this problem, a significant direction in recent studies is to design new lightweight architectures that consume less floating-point operations. Among these researches, depthwise separable convolution [1] becomes a promising design strategy for reducing the model size and complexity of CNNs, where the standard convolution operation of CNN is decomposed into a depthwise convolution layer followed by several 1×1 convolution layers to simplify the CNN structure. The resulted CNN models such as MobileNetsV1 [1] and MobileNetV2 [2] have been widely used in real applications. The depthwise separable convolution structure has also become a popular choice in recent neural architecture search (NAS) studies such as Liu et al. [3], Cai et al. [4], and Tan et al. [5].

From another viewpoint, a CNN model often contains much redundancy among their weights after training [6], [7]. Hence, a useful strategy to compress CNN models is to remove the superfluous weights or filters of their convolutional kernels [8]–[11], so that the model size can be reduced and the inference speed can be increased. Once the unnecessary weights or filters are pruned, the CNN models can be fine-tuned or distilled [12] for restoring the performance via re-training.

Although the technique of pruning redundant filters has been shown highly effective for compressing standard convolution operations of CNNs, it has not been well applied to depthwise separable convolutions yet. The reasons could be as follows. First, depthwise separable convolution is often performed once per layer. As it is the only operation regarding the spatial context of the input feature map in the layer, reducing it would cause considerable performance degradation. Second, unlike standard convolutional kernels, pruning the depthwise separable convolution involves more sophisticated constraints on both the input and output feature maps.

To address these issues, we solve the structural constraint over the depthwise separable convolutions. Based on the constraints derived for pruning, we then introduce a method to remove the redundancy of lightweight structures built up with depthwise separable convolution layers. Our approach applies to the tasks with the architectures factorized using depthwise separable convolutions (such as MobileNets) for the deployment of embedded devices; this is helpful to earn extra gains of the inference efficiency of CNN models. The characteristics are summarized as follows:

- We derive the constraints required on the layer-wised I/O when deleting filters in a channel-wise group convolution since depthwise separable operations are realized as group convolutions [13]–[15].
- A multi-stage gradual pruning approach is introduced to compact the CNN models and adapt the weights progressively.

II. Related work

In this section, we first give a review of the depthwise-separable structure in Section II-A. Then, we survey the

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

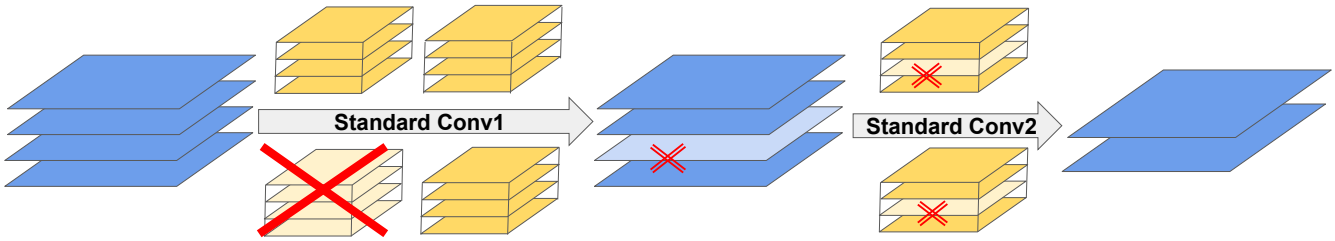

Fig. 1. Illustration of the filter pruning of a standard convolution. Suppose that there are 4 filters in Conv1, pruning the third filter will remove the third channel of output feature maps. After that, the third channel in all filters in the next layer (Conv2) should also be removed.

neural network pruning techniques for eliminating redundant branches in Section II-B.

A. Deep CNNs with Depthwise Separable Convolutions

The principle of depthwise separable convolution is derived from Flattened Networks [16] and Factorized Networks [17]. The architecture of Flattened Networks [16] is composed of flattened convolutions that separate the standard 3D convolutional filter into a set of 1D convolutional filters over channels, along with vertical and horizontal directions. The flattened convolutions can reduce numerous parameters of a CNN and enhance its accuracy slightly. Wang et al. [17] treat the standard 3D convolutional filter as a combination of 2D spatial convolutional filters inside each channel (intra-channel convolution) and a linear projection (1×1 standard convolutional layer) applied to all the channels. This combined with a residual connection is named as a single intra-channel convolutional layer. It also can compress the CNN models and enhance the accuracy of CNN slightly.

Inspired by the above researches, Howard et al. [1] separate the standard convolutional layer into a combination of one channel-wise group convolutional layer (depthwise convolution) and one standard convolution layer (pointwise convolution) with the kernel size of 1×1. ReLU [18] and batch-normalization [19] are applied after each convolution. To make the CNN model able to be deployed on mobile devices, they construct a novel structure of a lightweight network with 13 depthwise separable convolutions, referred to as MobileNetV1 [1]. It can save 85% of model size with only 1.1% accuracy drops. To further improve the information flow of MobileNetV1, Sandler et al. [2] expand the feature map channels by inserting an extra pointwise convolution before depthwise separable convolution. Combining with shortcut connections, it forms a novel building block called bottleneck depth-separable convolution with residuals that can better avoid information loss than depthwise separable convolution. By stacking 17 of such blocks, Sandler et al. [2] propose MobieNetV2 that achieves accuracy improvement as well as model size reduction. Besides, MobileNetV2 is also applied as backbones of SSD [20] and DeepLabV3 [21] to speed up object detection and semantic segmentation tasks.

B. Neural Network Pruning

In the field of deep model compression, filter pruning becomes one of the most popular directions because the pruned models can gain acceleration directly using the existing deep learning frameworks, such as Caffe [22] and PyTorch [23]. In filter pruning, the main challenge is to recognize unimportant filters so that removing them will not cause unrecoverable catastrophic performance drops. Li et al. [9] recognize unimportant filters by computing the $l1$ norm of filters in the pre-trained large model given. He et al. [8] utilize LASSO regression-based channel selection to remove the less useful channels. Li et al. [9] determine the importance of filters using the scale factors in batch normalization layers. After pruning the unimportant filters, these methods require retraining to compensate for the losses of performance. On the other hand, Ding et al. [11] follow another path to recognize correlated filters by K-means clustering and propose C-SGD to force filters in the same cluster to converge to the same values. Therefore, the redundant filters can be removed without the need for retraining.

Apart from filter pruning, several works also consider pruning finer structure like intra-kernel pruning that prunes spatial locations within convolution filters. Anwar et al. [24] use evolutionary algorithms to prune intra-kernel structures that have the least accuracy drops. Yang et al. [25] generate a series of intra-kernel patterns and prune them with a minimal absolute summation. Furthermore, unstructured pruning treats each weight as a pruning candidate and thus becomes the finest pruning strategy. Under this category, Zhu et al. [10] iteratively prune a handful of weights and retrain the remaining weights to recover the performance immediately. Although they can maintain accuracy while increasing the sparsity, special libraries or hardware are required to gain acceleration for the model inference, which imposes the development overhead in real applications.

III. Methodology

The standard convolution of CNN is formulated as follows. Denote $\mathbf{F}^{(i)} \in \mathbb{R}^{c_{i+1} \times c_i \times k \times k}$ as the convolution kernel of the i-th layer with k as the spatial size of the kernel. This layer takes an input feature map $\mathbf{X}^{(i)} \in \mathbb{R}^{c_i \times h_i \times w_i}$ and computes the output as

$$\mathbf{Y}^{(i)}_{c,:,:} = \sum_{j=1}^{c_i} \mathbf{X}^{(i)}_{j,:,:} * \mathbf{F}^{(i)}_{c,j,:,:}, \qquad c \in \{1 \cdots c_{i+1}\}, \tag{1}$$

where "$*$" denotes the common 2D convolution of size $k \times k$. Pruning the standard convolution (e.g., removing one kernel)

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

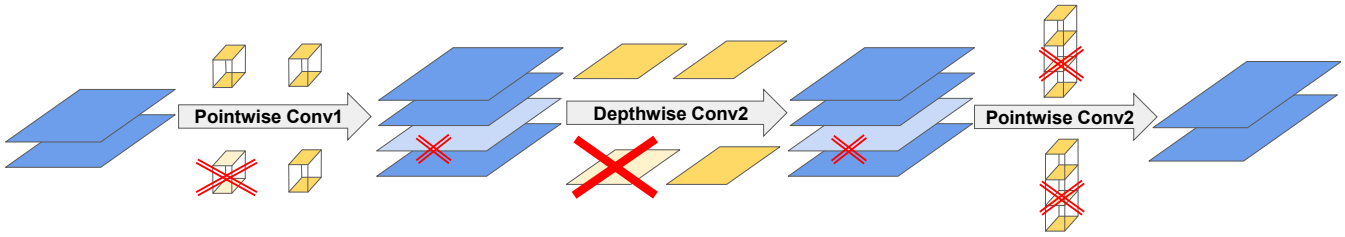

Fig. 2. Pruning a depthwise separable convolution. Suppose that there are 4 filters in Depthwise Conv2, pruning the third one, as depicted in the red cross, will remove the third channel of both input and output feature maps due to the parallelism in depthwise convolution. Thus, the corresponding kernels in Pointwise Conv1 and Depthwise Conv2 are also removed, as shown in double-crosses.

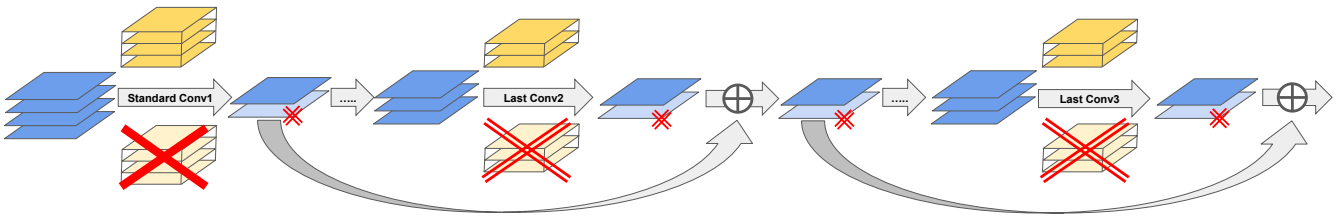

Fig. 3. Illustration of pruning residual blocks following the strategy in Ding et al. [11]. Supposed that there are 2 filters in Standard Conv1 which is followed by 2 residual blocks. Let Last Conv2 and Last Conv3 be the last layers of the two residual blocks, respectively. Pruning the second filter in Conv1 will force the second filter in the Last Conv2 and Last Conv3 to be removed.

only affects the input depth of the next layer. As illustrated in Figure 1, when a kernel (filter) is deleted, the output depth (that is equal to the input depth of the next layer) is also decreased, and thus the kernel depth in the next layer is shortened.

The convolution can be divided into groups in a layer, resulting in the operation known as group convolution [13], [15] that is originally used in AlexNet [26]. Pruning group convolutions is a structural pruning problem where the pruned parameters should follow patterns of their positions on the input and output channels. Therefore, more structural constraints have to be considered when pruning group convolutions.

A. Depthwise Separable Convolution & MobileNet Architecture

Assume that the input depth (number of channels) is c_i in the i-th layer. Depthwise convolution divides the input as c_i groups with each group containing a single channel. The output tensor $\mathbf{M}^{(i)} = DW_{conv}(\mathbf{X}^{(i)}, \mathbf{D}^{(i)})$ is given by

$$\mathbf{M}^{(i)}_{c,:,:} = \mathbf{X}^{(i)}_{c,:,:} * \mathbf{D}^{(i)}_{c,1,:,:}, \qquad c \in \{1 \cdots c_i\}. \tag{2}$$

In practical frameworks like Pytorch [23], this layer is implemented using grouped convolution by setting the number of groups equaling to the number of input channels.

MobileNetV1 [1] is proposed to utilize the depthwise convolution followed by the standard convolutions of spatial size 1×1 (referred to as pointwise convolutions). Let $Std_{conv}(\mathbf{X}, \mathbf{F})$ denote the standard convolution of Eq. 1. Supposed that $\mathbf{P}^{(i)} \in \mathbb{R}^{c_{i+1}\times c_i\times1\times1}$ are c_{i+1} pointwise kernels applied, MobileNetV1 then evaluates the output of the layer as follows

$$\mathbf{Y}^{(i)} = Std_{conv}(\mathbf{M}^{(i)}, \mathbf{P}^{(i)}). \tag{3}$$

The parameters in the layer are thus $(\mathbf{D}^{(i)}, \mathbf{P}^{(i)})$, where $\mathbf{D}^{(i)} \in \mathbb{R}^{c_i\times1\times k\times k}$ and $\mathbf{P}^{(i)} \in \mathbb{R}^{c_{i+1}\times c_i\times1\times1}$ are the kernels of depthwise and pointwise convolution layers, respectively.

To further improve MobileNetV1, Sandler et al. [2] propose to use a novel building block called bottleneck depth-separable convolution with residuals. This building block uses a shortcut connection and inserts an extra pointwise convolution that expands the input depth by a factor t before the depthwise convolution. Therefore, it contains three convolutions and computes the output as

$$\mathbf{Y}^{(i)} = Std_{conv}(DW_{conv}(Std_{conv}(\mathbf{X}^{(i)}, \mathbf{E}^{(i)}), \mathbf{D}^{(i)}), \mathbf{P}^{(i)}) + \mathbf{X}^{(i)}, \tag{4}$$

where $\mathbf{E}^{(i)} \in \mathbb{R}^{tc_i\times c_i\times1\times1}$, $\mathbf{D}^{(i)} \in \mathbb{R}^{tc_i\times1\times k\times k}$ and $\mathbf{P}^{(i)} \in \mathbb{R}^{c_{i+1}\times tc_i\times1\times1}$ are the three respective kernels, and thus its parameters are $(\mathbf{E}^{(i)}, \mathbf{D}^{(i)}, \mathbf{P}^{(i)})$.

B. Constrained Filter Pruning

Filter pruning is popular for structural reduction because it directly produces a thinner network that can easily fit into existing deep-learning frameworks for acceleration without the needs of self-defined operations. In the following, we introduce the approach that maintains the structural consistency when pruning depthwise separable convolutions with shortcut connections.

1) Depthwise Separable Convolutions: Unlike the case of standard convolutions, pruning depthwise convolutions influences both the input and output of a layer. For the i-th depthwise separable convolution with parameters $(\mathbf{D}^{(i)}, \mathbf{P}^{(i)})$, we denote the index sets of the depthwise and pointwise convolutions as $\mathbb{D}^{(i)} \subset \{0, 1, ..., c_i-1\}$ and $\mathbb{P}^{(i)} \subset \{0, 1, ..., c_{i+1}-1\}$. According to Eq. 2, each filter $\mathbf{D}^{(i)}_{c,1,:,:}$ operates on the input feature map $\mathbf{X}^{(i)}_{c,:,:}$. Besides, $\mathbf{X}^{(i)}_{c,:,:}$ is generated by the pointwise

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

convolution of the $(i-1)$-th layer with the filter $\mathbf{P}^{(i-1)}_{c,:,:,:}$. Thus, we can conclude that the filter pruning pattern of the i-th depthwise convolution should be the same as that of the $(i-1)$-th pointwise convolution (i.e., $\mathbb{P}^{(i-1)} = \mathbb{D}^{(i)}$), as illustrated in Figure 2. After pruning the filters in $\mathbf{D}^{(i)}$ and $\mathbf{P}^{(i)}$, similar to the pruning of standard convolutions, the associated kernels in the next layer $(i+1)$ have to be removed too. [1]

The structural constraints are extended to the depth-separable convolution blocks in MobileNetV2 [2] with the parameters $(\mathbf{E}^{(i)}, \mathbf{D}^{(i)}, \mathbf{P}^{(i)})$ as defined in Eq. 4. Similarly, we denote the index sets to be $\mathbb{E}^{(i)}$, $\mathbb{D}^{(i)}$ and $\mathbb{P}^{(i)}$. Because the input feature maps of the depthwise convolution are from the extra 1×1 convolution layer, their filter pruning indices should be the same. In other words, we have $\mathbb{E}^{(i)} = \mathbb{D}^{(i)}$ for all such blocks in MobileNetV2 [2].

2) Shortcut Connections: A shortcut connection that sums the learned residuals and the stem features introduce dependencies between the two layers linked. We call the layer producing the stem features the *pacesetter*, and the last layer of the residual block the *follower*. To maintain the validity of the information flow after filter pruning, the pacesetter and the follower must have the same pruning patterns. Li et al. [9] skip the pruning of pacesetters and followers and only prune other internal layers in the residual blocks. Accordingly, the input and output feature maps remain to have the same number of channels as the original model, and this limits the amount of prunable model size. Liu et al. [7] and He et al. [8] insert a sampler layer before the first internal layer in the residual block to reduce the channels of input feature maps. Though they can remove the corresponding kernels in the first internal layer, the filters of pacesetters and followers are un-pruned, which still limits the possibility of redundancy removal.

In our approach, to fully boost efficiency, we follow the strategy in Ding et al. [11] that allows pruning the whole network without sidestepping these troublesome layers. Starting from the beginning of MobileNetV2 [2], let m be the layer index of the non-residual block just before a sequence of n bottleneck depth-separable convolution residual blocks, as shown in Figure 3. Without loss of generality, we assume that the m-th layer is a standard convolution layer, which is usually the case in MobileNetV2 [2]. Each of the following n residual blocks has a pointwise convolution as its last layer within the block. The m-th layer produces the stem feature maps, and thus is called the pacesetter; the following blocks' pointwise convolutions are then the followers. Therefore, for all such structures starting with a standard convolution layer and followed by residual blocks, we can derive the constraints as shown below:

$$\mathbb{F}^{(m)} = \mathbb{P}^{(m+1)} = \mathbb{P}^{(m+2)} = ... = \mathbb{P}^{(m+n)}. \tag{5}$$

[1]Particularly, in MobileNetV1 [1], RGB images are converted into 32-channel feature maps via standard convolutions in the first layer. Thus we have $\mathbb{F}^{(1)} = \mathbb{D}^{(2)}$ as the starting point for pruning.

Algorithm 1: Multistage Gradual Pruning

Input: Kernel groups $\{\mathbb{W}^1, \mathbb{W}^2, ..., \mathbb{W}^G\}$, final pruning ratio r_f, number of stages S, pruning epochs E_p and finetune epochs E_f
Output: Filter index sets $\{\mathbb{I}^1, \mathbb{I}^2, ..., \mathbb{I}^G\}$ after pruning
Data: Training set $\mathcal{D}$

for s *in* $0, 1, ..., S-1$ **do**
  Compute the begin pruning ratio $r_b = s\frac{r_f}{S}$ and end pruning ratio $r_e = (s+1)\frac{r_f}{S}$.
  Determine the number of iterations in a pruning epoch $T = E_p \times \#iters\_per\_epoch$.
  for t *in* $0, 1, ..., T-1$ **do**
    if t *is multiple of* Δt **then**
      `// Prune a small amount of the model`
      Determine pruning ratio r_t by Eq. 6.
      Let the set of unpruned weights $\mathbb{T} = \{\}$.
      for *g in 1, 2, ..., G* **do**
        For each index i in $\mathbb{I}^g$, compute its score $\sum_{\mathbf{W}\in\mathbb{W}^g} \text{AbsSum}(\mathbf{W}_{i,:,:,:})$.
        Remove the indices with the smallest score from $\mathbb{I}^g$ until reaching the overall pruning ratio r_t in the g-th group.
        Add $\{\mathbf{W}_{i,:,:,:} | \forall i \in \mathbb{I}^g, \mathbf{W} \in \mathbb{W}^g\}$ into $\mathbb{T}$.
      Set all weights to 0.0 except for those in $\mathbb{T}$.
    Train the weights in $\mathbb{T}$ with the t-th batch of $\mathcal{D}$.
  `// Finetune the model to regain performance`
  Train the weights in $\mathbb{T}$ for E_f epochs on $\mathcal{D}$.

C. Gradual Pruning with Multiple Stages

In the above, we have derived the filter pruning constraints implied by the network structures. In this section, we describe our pruning approach for MobileNets. Gradual pruning [10] is a simple but effective technique that iteratively prunes small portions of weights with low absolute values until the target sparsity is reached. The original approach of Zhu et al. [10] is used for unstructured pruning that removes weights without following a regular network structure. Therefore, no direct speedup can be attained when implementing the pruned model to existing deep learning frameworks. In this work, we apply the gradual pruning principle to filter removing following the derived constraints. Besides, we extend the principle to multi-stage gradual pruning. The original gradual pruning approach is a special case of our approach of a single stage, and our multi-stage gradual pruning method provides a smoother track of pruning for handling the depthwise separable structure.

Given a pre-trained model, we first investigate all the filter pruning constraints and group the kernels having to share the same pruning patterns. Supposed that we get G groups of kernels, let $\mathbb{W}^g$ be the set containing all kernels in the g-th group. Because kernels in the g-th group have the same pruning patterns, they share the same index set $\mathbb{I}^g$. Filter pruning removes indices from $\mathbb{I}^g$ to reduce the network widths so that inference speedups can be acquired. To prune a portion,

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

denoted as r_f, of filters from a pre-trained model (pruning ratio 0.0), we split this process into S stages and apply gradual pruning to each stage.

Gradual pruning aims to prune a little number of weights every Δt iterations as the model is trained to maintain the performance while increasing the pruned ratio. The number of weights to be removed in each pruning iteration is computed by a cubic function to interpolate the beginning pruning ratio and end pruning ratio in a stage. Therefore, the pruning ratio after every Δt iterations is

$$r_t = r_e + (r_b - r_e)(1 - \frac{t}{n\Delta t})^3, \text{ for } t \in \{0, \Delta t, ..., n\Delta t\}, \quad (6)$$

where r_b and r_e are the begin and end pruning ratio, and n is the number of pruning iterations. In each pruning iteration, we compute importance scores for all candidate indices in $\mathbb{I}^g$ by summing the absolute values of corresponding filter elements and remove the ones with smaller scores. We further denote AbsSum($\mathbf{W}$) as the summation of absolute values of all elements in $\mathbf{W}$ and illustrate our approach in Algorithm 1.

IV. Experimental Results and Analyses

In this section, we show the experimental results and analyses to verify our approach.

A. Experimental Settings

To evaluate the performance on pruning lightweight models, we conduct experiments by applying the proposed multistage gradual pruning on MobileNetV1 and MobileNetV2 using the following three benchmarks:

CIFAR10 [27] is a standard benchmark for image classification. It contains 60,000 32×32 color images labeled with 10 categories of animals and vehicles, and each category has 6,000 images. We split them into 50,000 images as the training set and 10,000 images as the test set.

SVHN [28] is a benchmark for recognizing digits in natural scene images. It consists of 99,289 32×32 images, which are split into 73,257 for training and 26,032 for testing.

ImageNet [29] is a large-scale benchmark for image classification. It contains 13.3 million color images with 10,000 categories. We following the common protocol to use 12.8 million images for training and 50,000 images for validation.

On CIFAR10 and SVHN datasets, we train baseline models from scratch for 300 epochs using SGD optimizer with learning rates 0.1 and 0.01, respectively; for both datasets, the momentum is 0.9 weight-decay is 10^{-4}, and the learning rates decay 0.1 every 80 epochs. On ImageNet, we train MobileNetV1 using SGD for 100 epochs with learning rate 0.1, momentum 0.9, weight-decay 4×10^{-5}, and the learning rates decay exponentially with gamma 0.96. The pre-trained MobileNetV2 is officially provided in Pytorch [23]. In each pruning stage, we set Δt to 2000 for ImageNet and 200 for others. Our evaluation metrics include relative accuracy drops, reduced FLOPs between the baseline and pruned models. Our method is implemented on PyTorch [23] and evaluated on an Nvidia Geforce GTX 1080Ti GPU.

TABLE I
THE PERFORMANCE OF MOBILENETV1 WITH $1/4$ OF FILTERS PRUNED ON CIFAR10, SVHN, AND IMAGENET DATASETS.

MobileNetV1	Baseline Top1 Acc.	Pruned Top1 Acc.	Rel. ↓ %	FLOPs ↓ %
CIFAR10	86.28	86.15	0.15	42.5
SVHN	91.53	91.36	0.19	
ImageNet	70.69	68.84	2.61	

TABLE II
THE PERFORMANCE OF MOBILENETV2 WITH $1/4$ OF FILTERS PRUNED ON CIFAR10, SVHN, AND IMAGENET DATASETS.

MobileNetV2	Baseline Top1 Acc.	Pruned Top1 Acc.	Rel. ↓ %	FLOPs ↓ %
CIFAR10	86.31	85.61	0.81	41.0
SVHN	92.25	91.91	0.37	
ImageNet	71.88	67.25	6.44	

B. Results on CIFAR10 and SVHN

We compress baseline MobileNets on CIFAR10 and SVHN using 16-stage pruning, i.e. pruning $1/16$ of filters in each stage and report the performance on the models with $1/4$ filters pruned, as shown in Table I and Table II. On both datasets, multistage gradual pruning maintains the accuracy with less than 1% relative drops. This result indicates that even in lightweight models, redundant weights still exist and can be removed with negligible accuracy drops. As for the computation resources, the FLOPs can be reduced around 42.5% and 41.0% after pruning $1/4$ filters from baseline MobileNetV1 and V2.

Because our method further extends gradual pruning into multistage gradual pruning, we conduct experiments on how the number of stages affects the performance. When the number of stages increases, the pruning ratio in each stage decreases and thus results in a finer filter pruning procedure. We compare multistage gradual pruning with 8 and 16 stages, which prunes $1/8$ and $1/16$ of filters in one stage, respectively. In Figure 4, the performance of 8 and 16 stages on CIFAR10 are similar and note that 8-stage pruning already performs well on maintaining the accuracy when pruning ratio increases. However, the 16-stage pruning still maintains accuracy better on around 0.8 of filters pruned than the 8-stage setting. In Figure 5 (on the SVHN dataset), we find that the 8-stage pruning suffers from accuracy drops when the pruning ratio increases, but the 16-stage pruning manages to maintain the performance significantly better. We attribute this to that it is easier to recover performance by fine-tuning when performing a finer pruning procedure. Compared with coarse-grain pruning, fine-grain pruning produces recoverable accuracy drops but is more time-consuming because it needs more training epochs. By adjusting the number of pruning stages, multistage gradual pruning provides flexibility to trade accuracy maintenance off for faster pruning.

C. Results on ImageNet

In the above, we have shown that compact models like MobileNets can be further compressed to gain efficiency

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

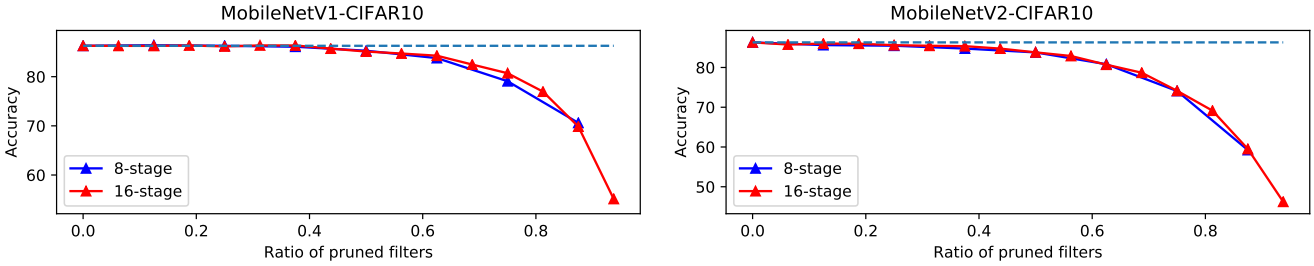

Fig. 4. The Accuracy of the pruned MobileNets on CIFAR10 dataset with 8 and 16 stages of gradual pruning.

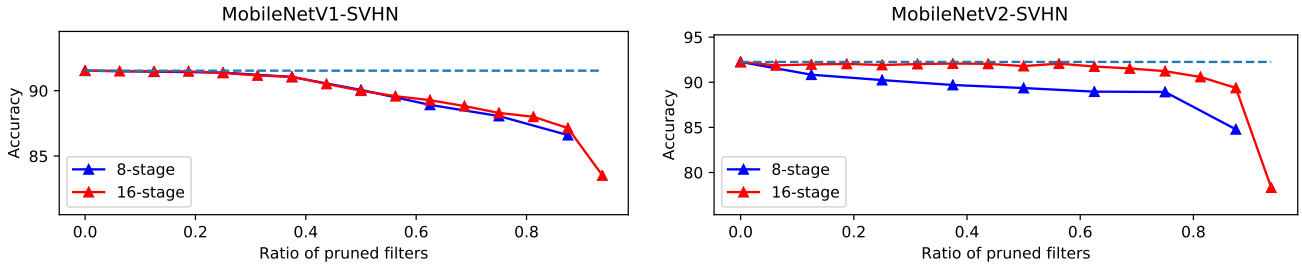

Fig. 5. The Accuracy of the pruned MobileNets on the SVHN dataset with 8 and 16 stages of gradual pruning.

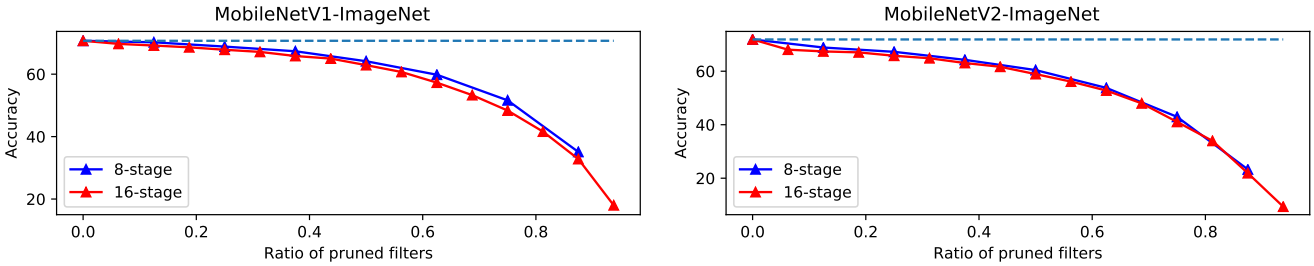

Fig. 6. The Accuracy of the pruned MobileNets on ImageNet dataset with 8 and 16 stages of gradual pruning.

TABLE III
THE COMPARISON OF MOBILENETV1 WITH 1.0, 0.75, 0.5 AND 0.25 WIDTH MULTIPLIERS ON IMAGENET BY OUR MULTISTAGE PRUNING AND TRAINED FROM SCRATCH.

Width Multipliers	Ours Acc.	Scratch [1] Acc.	FLOPs ↓ %	Params ↓ %
1.0×	70.69	70.6	-	-
0.75×	68.84	68.4	42.44%	38.90%
0.5×	64.15	63.7	73.26%	68.53%
0.25×	51.62	50.6	92.45%	88.89%

improvements. However, CIFAR10 and SVHN are small-scale datasets. To evaluate the accuracy drops on large-scale datasets, we perform multistage gradual pruning on ImageNet for both MobileNetV1 and V2, as shown in Table I and Table II. MobileNetV1 attains less than 3% relative accuracy drops while MobileNetV2's relative accuracy drop is 6.4%. Both pruned models have less than 60% FLOPs of their non-pruned counterparts. We also perform 8 and 16 stages of pruning in this experiment. As shown in Figure 6, on both pruning settings, MobileNetV1 with around 40% filters pruned can maintain its accuracy close to the baseline.

As the pruning ratio increases, the accuracy gradually decreases. In Table III, we show accuracy on various pruning ratios and compare them with the corresponding width multipliers of MobileNetV1 trained from scratch as reported in [1]. Models with 0.25, 0.5 and 0.75 of filters pruned are equivalent to those with 0.75, 0.5 and 0.25 width multipliers, respectively. As can be seen, MobileNetV1 pruned from baseline with our approach manages to outperform its trained from scratch counterpart by around 0.4% to 1.0%. These results demonstrate that our multistage gradual pruning is still effective to further compress compact models trained on large-scale datasets to a certain extent. However, as an improvement from MobileNetV1, MobileNetV2 struggles to maintain its accuracy in the early pruning stages. We owe the reason as follows. Compared with MobileNetV1, MobileNetV2 is a more compact model attaining higher baseline accuracy with smaller model size. Thus, it contains less redundant weights, particularly on large-scale datasets. Nevertheless, for smaller datasets (such as CIFAR10 and SVHN), MobileNetV2 is still redundant and our method can help remove them.

Therefore, as can be seen, some models are already too compact that is hard to be compressed. An issue becomes interesting is to identify such models so that we can decide to stop pruning or switch to finer pruning for better accuracy maintenance. To achieve this goal, we analyze the pruned models using their weight distributions and discuss the results in the next section.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

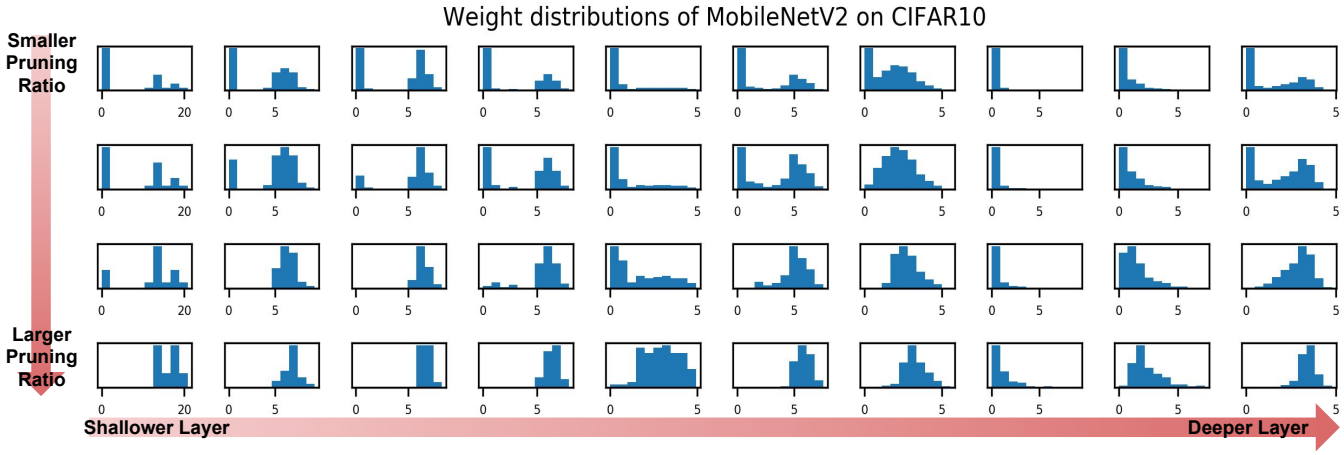

Fig. 7. Illustration of weight distributions on different kernel groups during the 8-stage pruning of MobileNetV2 on CIFAR10. From top to bottom, the first to the fourth rows of subplots represent models with pruned filter ratios of 0, 1/4, 2/4, 3/4, respectively. The columns of subplots represent 10 out of 25 kernel groups, and their depths become deeper as the plots locate from left to right columns. Within each subplot, the x-axis represents the absolute weight values and the y-axis indicates the number of weights of that value.

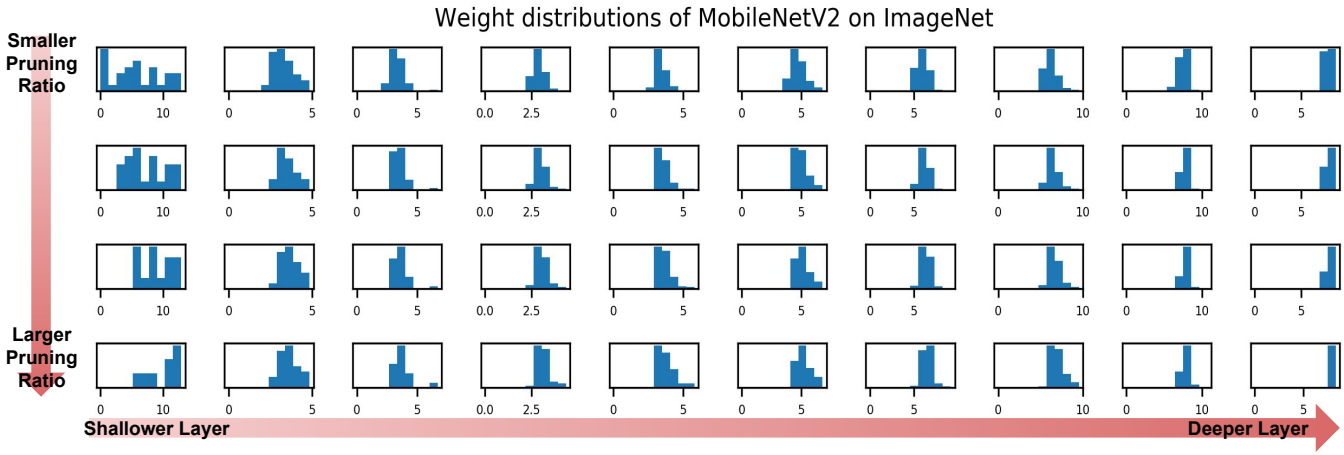

Fig. 8. Illustration of weight distributions on different kernel groups during the 8-stage pruning of MobileNetV2 on ImageNet. From top to bottom, the first to the fourth rows of subplots represent models with pruned filter ratios of 0, 1/4, 2/4, 3/4, respectively. The columns of subplots represent 10 out of 25 kernel groups, and their depths become deeper as the plots locate from left to right columns. Within each subplot, the x-axis represents the absolute weight values and the y-axis indicates the number of weights with that value.

D. Results discussion: analysis on pruning weights

When proceeding to a certain pruning ratio, the accuracy of the pruned models will start decreasing considerably. To take a closer look at this phenomenon, we provide an insightful discussion from analyzing the weight distributions in different pruning stages at different kernel groups for MobileNetV2 on CIFAR10. In Figure 7, subplots from the first to fourth rows show the weight distributions of models with 0, 1/4, 2/4, 3/4 of filters pruned. Subplots from left to right represent the weight distributions of shallow to deep layers.

As can be seen, at the beginning of MobileNetV2 on CIFAR10, there exist a portion of weights that are close to zero, and the performance decreases negligibly because when pruning these near-zero weights, the performance is easy to be recovered by fine-tuning. Hence, starting from the weight distribution shown in the first row of Figure 7, MobileNetV2 still achieves an accuracy of 85.47% that is close to the non-pruned one, 86.31%, when 25% filters are pruned.

The weight distribution after 25% filters pruned is then shown in the second row of Figure 7. As can be seen, the portion of zero weights becomes smaller and pruning such model results in an accuracy of 83.79% when the pruning ratio reaches 50%. Then, on the third row, the situation becomes severer as the pruning proceeds because some kernel groups have no or less zero weights. Being pruned to the ratio of 75%, the model drastically degrades to accuracy 74.08% afterward. Therefore, we conclude that the amount of zero weights in the kernel groups is related to the performance degradation of pruning, which provides useful clues to predict whether a model is compressible using our approach.

Similar phenomenons exhibit in the other experiments. For further discussion, we investigate the accuracy drops of MobileNetV2 on ImageNet versus its weight distribution. In Figure 8, we find that MobileNetV2 has little amount of zero weights at the beginning, which means that MobileNetV2 is already a highly compact model on handling the large-scale

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

task, ImageNet. This causes the accuracy degradation from 71.88% to 67.25% after pruning only 25% filters. As the pruning proceeds, we can only prune the weights of larger absolute values since no zero weights are left. Pruning such weights introduces accuracy drops that are unrecoverable via fine-tuning, and results in the accuracy of 42.94% when 75% filters are pruned. This explains why the performance drops in early stages when pruning MobileNetV2 on ImageNet.

Observing that the weight distributions can be employed to predict the performance of pruned models, we can judge whether the model is apt for pruning in our approach quickly when providing a new task.

V. Conclusions and Future Work

In this paper, a multistage gradual pruning approach for depthwise separable convolution networks is introduced. We show that redundancy of the lightweight models, MobileNets, can be further exploited in a multi-stage manner for a smooth model pruning and compression for proper tasks. In our methods, we prune the filters considering structural constraints so that the efficiency can be boosted. Our experiments reveal that we can maintain accuracy better when using finer pruning ratios and more stages. An analysis of weight distribution is also given to forecasting the multi-stage gradual pruning performance in advance. In the future, we plan to investigate more on the weight distributions for dynamically adjusting the pruning ratio settings in each stage and kernel groups. Besides, we will also apply our approach to the depthwise separable convolution structures found by NAS to improve the inference efficiency.

References

[1] 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.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[3] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.

[4] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HylVB3AqYm

[5] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[6] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[7] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[8] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[9] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=rJqFGTslg

[10] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," in *International Conference on Learning Representations Workshops*, 2018. [Online]. Available: https://openreview.net/pdf?id=Sy1iIDkPM

[11] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal sgd for pruning very deep convolutional networks with complicated structure," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4943–4953.

[12] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[13] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[14] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014. [Online]. Available: https://docs.nvidia.com/deeplearning/sdk/cudnn-developer-guide/index.html

[15] Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi, "Deep roots: Improving cnn efficiency with hierarchical filter groups," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1231–1240.

[16] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," in *International Conference on Learning Representations Workshops*, 2015. [Online]. Available: https://arxiv.org/abs/1412.5474

[17] M. Wang, B. Liu, and H. Foroosh, "Factorized convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 545–553.

[18] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[21] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[22] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

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

[25] M. Yang, M. Faraj, A. Hussein, and V. Gaudet, "Efficient hardware realization of convolutional neural networks using intra-kernel regular pruning," in *2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2018, pp. 180–185.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[27] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[28] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

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

978-1-7281-6926-2/20/$31.00 ©2020 IEEE