

# MODEL-FREE ENERGY DISTANCE FOR PRUNING DNNs

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

We propose a novel method for compressing Deep Neural Networks (DNNs) with competitive performance to state-of-the-art methods. We measure a new model-free information between the feature maps and the output of the network. Model-freeness of our information measure guarantees that no parametric assumptions on the feature distribution are required. The new model-free information is subsequently used to prune a collection of redundant layers in the networks with skip-connections. Numerical experiments on CIFAR-10/100, SVHN, Tiny ImageNet, and ImageNet data sets show the efficacy of the proposed approach in compressing deep models. For instance, in classifying CIFAR-10 images our method achieves respectively 64.50% and 60.31% reduction in the number of parameters and FLOPs for a full DenseNet model with 0.77 million parameters while dropping only 1% in the test accuracy. Our code is available at <https://github.com/suuyawu/PEDmodelcompression>.

## 1 INTRODUCTION

Modern deep convolutional neural networks (CNNs) with competitive performance in image understanding competitions (e.g. *ImageNet* Deng et al. (2009) and *COCO* Lin et al. (2014)) are over-parameterized with millions of parameters. As some examples, Caffe models of AlexNet (Krizhevsky et al., 2012) and VGG-16 of Simonyan & Zisserman (2014) need over 200MB, and 500MB memory space, respectively. In addition, pre-trained ResNet152 in PyTorch requires 228MB memory, and has more than 58 million parameters<sup>1</sup>. Hence, using deep neural models in devices with limited hardware is a challenge, due to both memory requirements and the large number of floating point operations per second (FLOPs). Recently, the aforementioned issues have fueled research activity in developing computationally efficient deep neural models. As mentioned, most remarkable performance of deep models have been achieved by very large models both in depth and width. Also, empirical observations demonstrate that the test accuracy of these models will not be affected drastically by distorting their trained weights (Cheng et al., 2017; Huang et al., 2018). These observations may be interpreted as the existence of a lot of redundancy in a trained deep neural model. This motivates network compression algorithms where the goal is to maintain the performance quality while reducing the computation and memory requirements by removing redundant parameters of the network.

**Our Contributions:** In this paper, we focus on pruning networks with skip-connections by quantifying how informative of the units within the skip-connections about the output of the model. To this end, we first introduce a new model-free measure of distance between a set of layers referred to here as *skip-units* and the output of the network. This measure enjoys some of the properties of popular Shannon’s mutual information, but has computational advantages and is model-free. Next based on the model-free measure, we propose a pruning algorithm for architectures with skip-connections e.g., ReseNet (He et al., 2016) and DenseNet (Huang et al., 2017) to remove iteratively less important units from the graph of the network. In other words, our proposed pruning method is a structured technique that aims to prune a network by removing redundant units instead of individual weights. This gives competitive compression ratio with the existing methods. We provide extensive experimental results on the classification tasks of CIFAR-10, CIFAR-100, SVHN, Tiny ImageNet, and ImageNet data sets, demonstrating the efficacy of the proposed technique.

<sup>1</sup>Typically, the largest portion of memory utilization stems from the dense layers while most computational load is dominated by convolutional layers.

## 2 PRIOR ART

Due to the lack of space, here we review the most relevant pruning techniques to ours, and defer review of the other compression methods to the appendix. Among the existing approaches in the network compression, pruning techniques (Courbariaux et al., 2015; Gong et al., 2014; Han et al., 2016; He et al., 2017; Hou et al., 2017; Li et al., 2017; Liu et al., 2017; Merolla et al., 2016; Rastegari et al., 2016; Wen et al., 2016; Singh et al., 2019; Ding et al., 2018; He et al., 2018; Molchanov et al., 2017; Guo et al., 2016; Li et al., 2019; Luo et al., 2018; He et al., 2019; Ye et al., 2018; Bellec et al., 2018; Xiao et al., 2019; Liu et al., 2019; Frankle & Carbin, 2019; Malach et al., 2020; Lin et al., 2018; 2020; Zhuang et al., 2018) are quite popular. These methods reduce the network complexity through weights/kernels removal. These are in turn motivated by experimental observations that DNNs are robust to various types of weight distortion including quantization, additive and multiplicative noise injection, projection, etc (Merolla et al., 2016). Additionally, the well-known *drop-out* technique (Srivastava et al., 2014) and regularization technique of drop-block (Ghiasi et al., 2018) in residual networks may provide another intuitive inspiration for the pruning-based approaches. In general, pruning techniques are divided into two categories: Structured and Unstructured methods. Unstructured methods directly try to remove individual weights in a deep model and do not impose any structure in the pruning process. On the other hand, structured methods apply pruning in the level of filters and layers by imposing some structure on the topology of weights. While unstructured methods provide the most flexibility for the model and generally achieve higher compression ratio and accuracy in terms of memory requirement, they do not typically provide any gain in the FLOPs number (many of unstructured methods do not report any FLOPs reduction). Furthermore, it is harder to map unstructured methods efficiently to parallel processors, and the deep learning software packages have typically limited support for these methods. In contrast, structured methods are much easier to be deployed in parallel processors, and usually provide a significant reduction in the FLOPs count; hence, speeding up the inference running time. However, their compression ratio in terms of parameters count may be less than the structured methods as some of our experiments have illustrated this issue. In summary, there is a trade-off curve between achievable compression levels for a given model and the level of structure imposed on the model weights<sup>2</sup>. We invite the readers to keep this important difference between the structured and the unstructured methods in their mind when comparing the pruning results with each other.

The most relevant pruning technique with our approach is the structured method of dynamic path selection. In this method, a sub-graph or a group of weights with a specific structure is selected from the original network such that the chosen graph has a comparable inference (test) performance with the original model (Veit & Belongie, 2017; Hu et al., 2018; Wu et al., 2018; Wang et al., 2018; Huang et al., 2016; Frankle & Carbin, 2019; Lee et al., 2019). In particular, approaches proposed by Wang et al. (2018) and Wu et al. (2018) have mostly focused on the residual networks, and their goal is to dynamically select a set of layers in the original graph using learning-based approaches such as policy update in reinforcement learning. While it is an interesting observation, it seems that it is less effective with deeper neural networks such as deep ResNet families. In this paper, we propose a novel pruning method, delivering comparable/better performance to those of state-of-the-art deep compression models/techniques. Our method is a structured method based on removing a set of feature maps with less “information” about the output of the model, where the information is defined by a non-parametric energy distance. In addition, we compare our results with some of the state-of-the-art unstructured pruning methods.

## 3 BACKGROUND AND PROPOSED METHOD

### 3.1 DEEP NEURAL MODELS WITH SKIP-CONNECTION

It is well-known that network architectures with skip-connection are more robust to weight distortion (such as a random drop of the layers) than the feed-forward networks (Veit et al., 2016). In this paper, we take this phenomena one step further and develop a structured pruning technique based on removing a collection of redundant layers instead of individual random layers. First, we define a *skip-unit* as a set of layers where its output is a function of sequential application of operations in the unit including *Conv*, *Pooling*, *ReLU*, *BN*, *Dropout*, etc and feature maps of previous unit(s). Specifically,

<sup>2</sup>We refer the readers to an excellent paper by Gale et al. (2019) for more discussion.

consider a DNN with  $L$  skip-units. We denote the input of  $l^{th}$  unit as  $U_{l-1}$ . Let  $T_l = f_l(U_{l-1})$  denote the output of sequential application of the aforementioned operations in a skip-unit, summarized by  $f_l$ . We call  $T_l$  as the feature map of the unit  $l$ . Hence, each skip-unit is mathematically given by:

$$U_l = \Psi(T_l, U_{1:l-1}, \alpha_l), \quad l = 2, 3, \dots, L, \quad (1)$$

where  $\alpha_l \in \{0, 1\}$  is referred to as *policy* and  $\Psi$  denotes an operation that combines  $T_l$  and  $U_{1:l-1}$ . Also,  $U_{1:l-1}$  denotes all the skip-units from unit 1 to the  $(l-1)^{th}$  unit. In ResNet and DenseNet models,  $\Psi_{res}$  and  $\Psi_{den}$  are respectively given by (see the appendix for reviewing these models):

$$U_l = \Psi_{res}(T_l, U_{1:l-1}, \alpha_l) = \alpha_l T_l + \mathcal{A}_{l-1} U_{l-1}, \quad (2)$$

$$U_l = \Psi_{dense}(T_l, U_{1:l-1}, \alpha_l) = \text{Concat}(\alpha_l T_l, U_{1:l-1}), \quad (3)$$

where  $\text{Concat}$  is the concatenation operation, and  $\mathcal{A}_{l-1}$  is an identity, down-sampling, or some operation such as convolution operator(s). Furthermore,  $U_0$ , the input for the first unit is usually given by a convolution operator (e.g., in ResNet, and DenseNet). In the above expressions,  $\alpha_l$  indicates that which skip-unit should be removed or retained from the architecture ; hence, the name policy.

### 3.2 MODEL-FREE MEASUREMENT OF DEPENDENCY

Our main idea for compression of deep models with skip-units is based on measuring the information between each skip-unit and the output of the model, and then removing the less important units. A popular information measure is given by Shannon’s *Mutual Information* (MI). Computing the mutual information between the output and the features of a machine learning model has been a research topic with a long history in unsupervised feature learning (Linsker, 1988), and variable ranking or screening (Murphy, 2012). In the context of deep learning, MI between the hidden layers and the output of a network has been investigated by the *Information Bottleneck* theory (Shamir et al., 2010; Tishby & Zaslavsky, 2015). While Shannon MI is an appealing mathematical measure, its estimation may be a difficult task. The estimation of MI for high-dimensional data needs a model assumption for the underlying probability distribution. The selected model may be statistically unrealistic (even when a large number of samples are available (Paninski, 2003)). For example, one method for computing Shannon MI is the histogram method. This becomes more accurate if the histogram bins are not too coarse, but will be computationally more cumbersome to handle. Another option is to use a more complex parametric model such as Gaussian mixture models (Kolchinsky & Tracey, 2017) which is computationally prohibitive for high-dimensional feature maps. The above issues for the computation of MI motivate new model-free measures with similar properties of MI that are not computationally intense (Bottou et al., 2018). We introduce a measure that quantifies the dependency between two random vectors  $T$  and  $Y$ , network features and output labels, respectively. Our proposed measure is based on the following Energy Distance (ED) between two continuously-valued random vectors.

**Definition 1** (Energy Distance (ED) Székely & Rizzo (2013)). *Suppose that the characteristic functions and distribution functions of two continuous-valued random vectors  $T_i \in \mathbb{R}^d$  are respectively given by  $\phi_i(\cdot)$ ,  $F_i(\cdot)$  ( $i = 1, 2$ ). The ED between  $F_1$  and  $F_2$  is given by*

$$\mathcal{E}(F_1, F_2) = \frac{1}{c_d} \int_{\mathbb{R}^d} \frac{|\phi_1(s) - \phi_2(s)|^2}{\|s\|^{d+1}} ds, \quad (4)$$

where  $c_d = \pi^{(1+d)/2} / \Gamma((1+d)/2)$  is a constant that only depends on  $d$ .

We have the following result due to Székely & Rizzo (2013)<sup>3</sup>.

**Theorem 1.** *The energy distance in equation 4 can be written as*

$$\mathcal{E}(F_1, F_2) = 2E\|T_1 - T_2\| - E\|T_1 - T'_1\| - E\|T_2 - T'_2\|, \quad (5)$$

where  $T'_1$  and  $T'_2$  are i.i.d copies of  $T_1$  and  $T_2$ , respectively, and  $\|\cdot\|$  denotes the Euclidean norm.

Now suppose that we have  $n_i$  observations of  $T_i$ ,  $i = 1, 2$ , where  $t_{i,j}$  denotes the  $j$ -th observation of  $T_i$ . Then an unbiased estimator of ED in (5) is given by:

$$\begin{aligned} \hat{\mathcal{E}}(F_1, F_2) = & \frac{2}{n_1 n_2} \sum_{1 \leq j_1 \leq n_1, 1 \leq j_2 \leq n_2} \|t_{1,j_1} - t_{2,j_2}\| - \\ & \frac{1}{n_1^2} \sum_{1 \leq j_1, j'_1 \leq n_1} \|t_{1,j_1} - t_{1,j'_1}\| - \frac{1}{n_2^2} \sum_{1 \leq j_2, j'_2 \leq n_2} \|t_{2,j_2} - t_{2,j'_2}\|. \end{aligned} \quad (6)$$

<sup>3</sup>Székely & Rizzo (2013) have shown that energy distance given in equation 5 is always non-negative.

Applying the theory of V-statistics (Lee, 2019), it can be proved under mild assumptions that the above  $\hat{\mathcal{E}}(F_1, F_2)$  is a consistent estimator of  $\mathcal{E}(F_1, F_2)$ , that is  $\hat{\mathcal{E}}(F_1, F_2) \rightarrow \mathcal{E}(F_1, F_2)$  in probability, as  $n = \min\{n_1, n_2\} \rightarrow \infty$ . This implies that with sufficiently large data size,  $\hat{\mathcal{E}}(F_1, F_2)$  vanishes for  $T_1$  independent from  $T_2$ , and is bounded away from zero otherwise, without the need of specifying any parametric model. Based on the above distance, we define the following quantity of dependency.

**Definition 2.** Consider random vector/variable  $T \in \mathbb{R}^d$  and  $Y \in \mathcal{Y}$ , where  $\mathcal{Y}$  has  $p$  elements, say  $\{1, 2, \dots, p\}$  for notational convenience. The energy dependence between  $T$  and  $Y$  is defined by  $D(T, Y) = \max_{1 \leq i, j \leq p} \mathcal{E}(F_i, F_j)$ , where  $F_i$  denotes the distribution of  $T$  conditional on  $Y = i$ .

**Theorem 2.** For random vector/variable  $T \in \mathbb{R}^d$  and  $Y \in \mathcal{Y}$  with a finite alphabet  $\mathcal{Y}$ ,  $D(T, Y) = 0$  iff  $T$  and  $Y$  are independent.

*Proof.* The independence of  $T$  and  $Y$  is equivalent to  $F_{T|Y=j} = F_{T|Y=j'}$  for any  $1 \leq j, j' \leq p$ . This is further equivalent to  $D(T, Y) = 0$ , according to Definition 2 and Theorem 1.  $\square$

From its definition, the distance in equation 4 between each pair of conditional distributions  $T | Y$  can be interpreted as a weighted  $L_2$  distance between their characteristic functions. The larger the distance is, the larger is the dissimilarity. As a consequence, the energy dependence in Definition 2 may be interpreted as a quantification of the dependency between  $T$  and  $Y$ . Based on (6), we define a consistent estimator of  $D(T, Y)$  by  $\hat{D}(T, Y) = \max_{1 \leq i, j \leq p} \hat{\mathcal{E}}(F_i, F_j)$ , where  $\hat{\mathcal{E}}(F_i, F_j)$  is given by equation 6. We use this to measure the dependency between  $T$  and  $Y$ . The larger its value is, the more information  $T$  reveals about  $Y$ . Similar to the consistency of (6),  $\hat{D}(T, Y)$  is a consistent estimator of  $D(T, Y)$  according to the standard theory of V-statistics.

### 3.3 PROPOSED ALGORITHM

We now present our pruning algorithm using energy dependence. Our pruning method is applicable to any network with skip-connections among different layers. The pseudocode of our approach referred to as *Pruning with Energy Dependence* (PED) is given in Algorithm 1. PED is run in  $N$  stages where  $N$  is a hyper-parameter. In this algorithm, we define the cluster vector  $K^t$  as a set of candidates for the number of cluster in stage  $t$ . We denote the  $i^{th}$  entry of  $K^t$  by  $K_i^t$ . Also,  $|A|$  is the size of a set  $A$  and  $S^t$  is the index set of skip-units which maintains the index of active skip-units. Finally,  $\text{DNN}^t$  denotes a trained pruned model with weights initialized from stage  $t - 1$ . In each stage  $t$ , PED determines the units whose feature maps  $T_l$  are more informative about the output of the model for the given input. To quantify the informativeness of units, PED computes the estimate of energy dependence between  $T_l$  and the output of the model,  $\hat{D}(T_l, Y)^4$ ,  $l = 1, 2, \dots, L$ , defined in Definition 2. Next PED uses  $\hat{D}(T_l, Y)$  to select active units for which  $\alpha_l = 1$  by clustering the  $\hat{D}(T_l, Y)$  values. Please see the appendix for the pseudocode of the clustering algorithm<sup>5</sup>. Out of all the number of cluster candidates, the one with minimum training error is selected ( $k_*^t$ ). Next PED only keeps  $k_*^t$  cluster centroids and remove all the other units<sup>6</sup>. In the appendix, we have illustrated the pattern of dropping units across the different stages for various models and data sets. At the end of stage  $t$ , PED re-trains the new compressed model with weights in the active units and initialized with their values from the previous stage. This multi-stage coarse pruning and re-training process continues until the pre-determined size of the compressed network is met.

We now illustrate what  $\hat{D}(T_l, Y)$ 's represent across different stages through an example depicted in Figure 1, where we have plotted the histogram of the energy values. Plots (a) and (b) in Figure 1 and also plots (c) and (d) illustrate the frequency of the energy values (distribution of energy dependence) for the full model (before any pruning) and after 9 stages pruning for the CIFAR-10 and CIFAR-100 data sets, respectively. These plots reveal some interesting observations. First, the values of energy

<sup>4</sup>For the notation simplicity, we use  $D(\cdot)$  instead of  $\hat{D}$  in the pseudocode.

<sup>5</sup>For clustering, we have used an optimal  $k$ -means algorithm based on dynamic programming (Wang & Song, 2011). Other clustering algorithm can be used here; however, since the clustering is applied on 1-D data, using the approach by Wang & Song (2011) guarantees optimality of clustering which is not generally the case for  $k$ -means type algorithms.

<sup>6</sup>Empirically, we have observed that choosing a centroid as the one either with the smallest index, or the largest index in all the cluster results in less reduction in the training accuracy compared to the training accuracy of the previous stage. This implies the importance of units closer to the input and the output of the network.

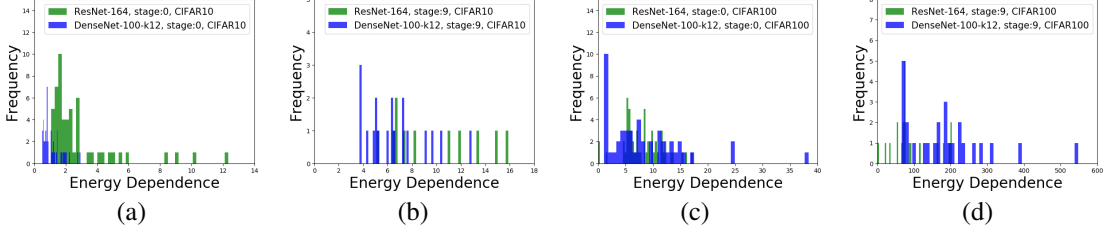


Figure 1: Histogram of the ED of skip-units in ResNet164 and DenseNet100-k12. **(a) and (b)** Energy Dependence frequency for CIFAR-10 data set respectively for stage  $N = 0$  (full model), and for the last stage,  $N = 9$ . **(c) and (d)** Energy Dependence frequency for CIFAR-100 data set respectively for stage  $N = 0$  (full model), and for the last stage ( $N=9$ ).

dependence (x-axis in the plots) increase with the number of pruning stages, and the rate of increase is more significant in the CIFAR-100 data set (from the plot (c) to (d)). This is intuitively appealing since as we drop more skip-units, the information of the remaining units about the output layer is getting more important. The second observation is that the distribution of energy dependence values has more variation in the later stages of the algorithm (plots (b) and (d)). The latter observation motivates starting with more aggressive pruning in the earlier stages (i.e., using the smaller number of clusters), and gradually switch to finer pruning (by selecting a larger number of clusters) in later stages. In addition, this conclusion can be used as a rule of thumb for choosing the number of clusters (cluster vector  $K^t$ ) in each stage. For selecting the entries of  $K^t$  in each stage, while simply trying all the possible values from 1 to  $|S^t| - 1$  is an acceptable approach (In ResNet or DenseNet,  $|S^t|$  is at most 60, and it decreases after each stage), we choose entries of  $K^t$  based on the observation illustrated in Figure 1: In the early stages,  $K^t$  has a few smaller values than  $|S^t|$  (e.g.,  $|S^t| - i$  for  $2 \leq i \leq 4$ ), while in the later stages,  $K^t$  has only one large value, e.g.,  $K^t = \{|S^t| - 1\}$ .

Another important question here is why not choosing the units with the largest energy values, or selecting them randomly instead of clustering approach. To answer this question, we first note that model-free measure of information is a random variable which depends on the data samples. When these values are close to each other, a test of hypothesis fails to reject if an information value is larger than the other. Thus, we cluster the values that cannot be distinguished from each other statistically. On the other hand, due to the fact that the existence of the skip-units violates the Markov Chain property among the layers, the mutual information between the skip-units and the output does not decrease monotonically according to the information processing inequality. The latter argument implies that simply selecting units randomly or with the largest energy values (information) may remove some important skip-units. To support the preference of clustering over two other schemes, we have compared pruning by clustering with the two above schemes in plots (a) and (b) of Figure 2 for pruning ResNet56 and CIFAR-10 data set. To be fair in comparison with the clustering scheme used in PED, we have used the same number of units used in the clustering for removing units randomly and with the smallest energy. As we can see in plots (a) and (b) of Figure 2, the test accuracy with respect to the percentage of remaining parameters, and the percentage of remaining FLOPs by pruning through the largest values and random selection are worse than the clustering approach specifically for the later stages (corresponding to more pruning). This observation favors the clustering approach rather than simply dropping the units randomly or with the smallest energy; hence, clustering scheme avoids removing potentially important units. Please see section A.6.2 in the appendix for the details of experiments (pruning performance in the intermediate stages) in Figure 2. Finally, we have to mention that our proposed technique can be combined with the unstructured methods, which remove the less important individual weights. This means that after we finished pruning a model with PED, we may apply unstructured state-of-the-art methods such as SNIP (Lee et al., 2019), or GraSP (W. et al., 2020) on the pruned model to remove individual weights; hence, achieving more compression rate. We leave this for more investigation in the future.

## 4 EXPERIMENTAL RESULTS

In this section, we present the performance of our proposed algorithm, PED on CIFAR-10, CIFAR-100, SVHN, and ImageNet data sets in terms of the test accuracy, number of parameters, and

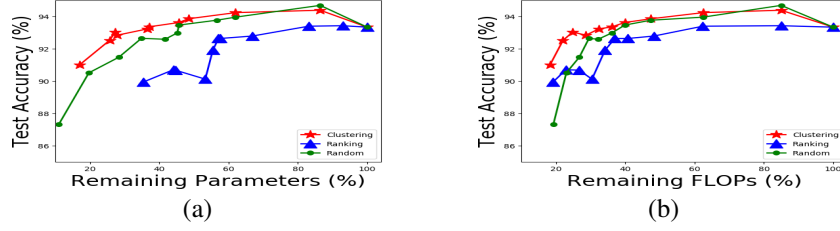


Figure 2: Comparison of clustering scheme in PED ( $N = 10$ ) against pruning using units with the largest energy values, and random selection. The experiments are based on ResNet56 and CIFAR-10.

---

**Algorithm 1** Pruning with Energy Dependence (PED)

---

**INPUT:**

$\text{DNN}^0$ : Pre-trained Deep Neural Network  
 $S^0$ : The index set of skip-units in  $\text{DNN}^0$   
 $T_l$ : Feature maps,  $l = 1, 2, \dots, |S^0|$   
 $K^t$ : Cluster vector,  $t = 0, 1, \dots, N - 1$   
 $N$ : Number of stages  
**for**  $t = 0, 1, \dots, N - 1$  **do**  
  Compute  $D(T_l^t, Y)$ ,  $l = 1, \dots, |S^t|$  using  $\text{DNN}^t$   
  Construct  $\mathbf{D}^t = [D(T_1^t, Y), D(T_2^t, Y), \dots, D(T_{|S^t|}^t, Y)]$   
   $\{Cluster_1^{K^t}, \dots, Cluster_{K_1^t}^{K^t}, \dots, Cluster_1^{K_{|K^t|}^t}, \dots, Cluster_{K_{|K^t|}^t}^{K_{|K^t|}^t}\} = \text{Clustering}(K^t, \mathbf{D}^t, S^t)$   
  **for**  $k$  in  $K^t$  **do**  
    **for**  $j = 1, 2, \dots, k$  **do**  
       $a_c = 1$ ,  $c$  = cluster centroid index  
       $a_u = 0$ ,  $\forall u \in Cluster_j^k \setminus c$   
    **end for**  
    Compute  $Train_{err}^k$  for given  $k$   
  **end for**  
  Select  $k_*^t = \arg \min_{k \in K^t} Train_{err}^k$   
  Update  $S^t$  by keeping only  $k_*^t$  units and remove the rest of units  
  Update  $\text{DNN}^t$  by re-training the model with  $k_*^t = |S^t|$  units with weights initialized in stage  $t$   
**end for**  
Return pruned model with  $|S^{N-1}|$  active skip-units

---

FLOPs of the pruned model<sup>7</sup>. Among all the existing pruning methods, two structured methods of SkipNet (Wang et al., 2018) and BlockDrop (Wu et al., 2018) are most relevant ones to PED as they try to prune dynamically residual units in ResNet families based on reinforcement learning. For computing FLOPs, we add up the number of multiplications, additions, and ReLU comparisons of the pruned model. The CIFAR-10 and CIFAR-100 data sets used extensively for image classification consist of 50000 and 10000 training and testing images with size  $32 \times 32 \times 3$  which are categorized in 10 and 100 classes, respectively. Moreover, the ImageNet data set is consist of 1000 labels with 1.2 million training images and 50000 validation images. We use the common data-augmentation scheme at training time (He et al., 2016), and perform a rescaling to  $256 \times 256$  followed by a  $224 \times 224$  center crop at test time before giving the input images to the models. We also use Cutout augmentation technique (DeVries & Taylor, 2017) for CIFAR data sets in the training of the deep models in each stage including the original full model. We note that this augmentation technique has only been applied to the training data sets and not the test ones. Other compression techniques including those presented in (Huang et al., 2018; Liu et al., 2017) also use data augmentations techniques such as zero-padding, random cropping, shifting and mirroring, and with different normalization only to the training data set. We compare the performance of PED with the state-of-the-art deep neural compression methods in Table 1 for CIFAR-10, in Table 2 for CIFAR-100, for SVHN in Table 3, and for ImageNet in, Table 4. The third/forth and fifth/sixth columns labeled 'Red' in all tables represent

<sup>7</sup>Due to the lack of space, we defer the details of experiments and architectures, results of intermediate stages, and experiments on Tiny ImageNet to the appendix.

Models	Acc.	Par.	Red.(%)	FLOPs	Red.(%)
ResNet32 (SNIP) Lee et al. (2019) <sup>U</sup>	0.9259	0.19	90.00	-	-
ResNet56 (PFEC-B) Li et al. (2017) <sup>S</sup>	0.9306	0.73	13.70	90.90	27.60
ResNet56 (PP-1) Singh et al. (2019) <sup>S</sup>	0.9309	-	-	39.99	68.40
ResNet56 (NISP) Yu et al. (2018) <sup>U</sup>	0.9301	-	-	67.57	46.60
ResNet56 (AFP-G) Ding et al. (2018) <sup>S</sup>	0.9294	-	-	55.50	60.86
ResNet56 (SFP) He et al. (2018) <sup>S</sup>	0.9359	-	-	59.40	52.60
ResNet56 (CNN-FCF) Li et al. (2019) <sup>S</sup>	0.9338	-	43.09	72.40	42.78
ResNet56 (FPGM-mix 40%) He et al. (2019) <sup>S</sup>	0.9359	-	-	59.40	52.60
ResNet56 (HRank) Lin et al. (2020) <sup>S</sup>	0.9317	0.49	42.40	62.72	50.00
ResNet110 (BlockDrop) Wu et al. (2018) <sup>S</sup>	0.9360	-	-	173.00	65.00
ResNet110 (SkipNet) Wang et al. (2018) <sup>S</sup>	0.9330	-	-	126.00	50.47
ResNet110-pruned (PFEC-B) Li et al. (2017) <sup>S</sup>	0.9330	1.16	32.40	155.00	38.60
ResNet164-pruned Liu et al. (2017) <sup>S</sup>	0.9492	1.44	14.90	381.00	23.70
DenseNet40-pruned Liu et al. (2017) <sup>S</sup>	0.9481	0.66	35.70	381.00	28.40
IGC-V2*C416 Xie et al. (2018) <sup>S</sup>	0.9451	0.65	-	-	-
CondenseNet86 Huang et al. (2018) <sup>S</sup>	0.9496	0.52	-	65.00	-
ResNet56 (pruned) ( <b>ours</b> )	0.9336	0.32	62.96	45.73	63.86
ResNet164-a (pruned) ( <b>ours</b> )	0.9519	0.65	61.78	71.06	72.13
ResNet164-b (pruned) ( <b>ours</b> )	0.9426	0.48	71.57	48.93	80.81
DenseNet100-k12-a (pruned) ( <b>ours</b> )	0.9432	0.29	61.98	117.73	59.89
DenseNet100-k12-b (pruned) ( <b>ours</b> )	0.9425	0.27	64.50	116.51	60.31

Table 1: Classification test accuracy, number of parameters (Par), and FLOPs on CIFAR-10 between PED and those of the state-of-the-art methods. Par and FLOPs are in million.

Models	Acc.	Par.	Red.(%)	FLOPs	Red.(%)
ResNet32 (SET) Mocanu et al. (2018) <sup>U</sup>	0.6966	0.19	90.00	-	-
ResNet32 (GraSP) W. et al. (2020) <sup>U</sup>	0.6924	0.19	90.00	-	-
ResNet32 (DSR) Mostafa & Wang (2019) <sup>U</sup>	0.6963	0.19	90.00	-	-
Resnet110 (BlockDrop) Wu et al. (2018) <sup>S</sup>	0.7370	-	-	~284.00	~56.00
ResNet110 (SkipNet) Wang et al. (2018) <sup>S</sup>	0.7250	-	-	-	37.00
ResNet164-pruned Liu et al. (2017) <sup>S</sup>	0.7713	1.46	15.50	333.00	33.30
DenseNet40-pruned Liu et al. (2017) <sup>S</sup>	0.7472	0.66	37.50	371.00	30.30
CondenseNet86 Huang et al. (2018) <sup>S</sup>	0.7636	0.52	-	65.00	-
ResNet164-a (pruned) ( <b>ours</b> )	0.7499	0.58	67.70	100.09	60.74
ResNet164-b (pruned) ( <b>ours</b> )	0.7402	0.47	72.80	91.01	64.30
DenseNet100-k12-a (pruned) ( <b>ours</b> )	0.7526	0.51	36.64	239.27	21.32
DenseNet100-k12-b (pruned) ( <b>ours</b> )	0.7488	0.47	40.78	221.89	27.04

Table 2: The top-1 test accuracy, number of parameters (Par), and FLOPs on CIFAR-100 between PED and those of the state-of-the-art methods. Par and FLOPs are in million. “~” means approximate value. The accuracy reported in W. et al. (2020) ResNet32 is given  $0.6924 \pm 0.24$ .

the percentage of reduction in the number of parameters and FLOPs, respectively. Moreover, the numbers given in columns labeled as “Par.” (i.e., Parameters) and “FLOPs” are in million (Only FLOPs in Table 4 is in Billion) and rounded by two-decimal digits. In all tables, “-” means no reported value. In general, FLOPs count is not an accurate measure for comparing the performance of pruning methods as different techniques have used various operations in counting total FLOPs; hence different reported values<sup>8</sup>. Also, many structured pruning methods do not provide any FLOPs reduction. Next we present the result of applying PED on ResNet and DenseNet models. In all the tables, the superscript “S” and “U” denote the structured or unstructured methods, respectively.

**ResNet56:** The ResNet56 architecture consists of 56 layers with 27 residual units and the total number of 0.85 (M) training parameters and 126.54 (M) FLOPs. In order to run PED on ResNet56,

<sup>8</sup>(Wu et al., 2018) has reported 508 million FLOPs for the full ResNet110, while it is given 253 million by (Wang et al., 2018; Li et al., 2017).

Model (ResNet164)	Acc.	Par.	Red.(%)	FLOPs	Red.(%)
(40% Pruned) Liu et al. (2017) <sup>S</sup>	0.9815	1.46	14.50	344.00	31.10
(60% Pruned) Liu et al. (2017) <sup>S</sup>	0.9819	1.12	34.30	225.00	54.90
PED (pruned) ( <b>ours</b> )	0.9815	0.83	51.23	101.84	60.00

Table 3: Classification test accuracy, number of parameters (Par), and FLOPs on SVHN data set between PED and those of the state-of-the-art methods. Par and FLOPs are in million.

Model (ResNet50)	Top1 Acc.	Top5 Acc.	Par.	Red.(%)	FLOPs	Red.(%)
SNIP Lee et al. (2019) <sup>U</sup>	0.7395	0.9197	10.22	60.00	-	-
GraSP W. et al. (2020) <sup>U</sup>	0.7402	0.9186	10.22	60.00	-	-
ThiNet-50 Luo et al. (2017) <sup>S</sup>	0.6842	0.8830	8.66	66.04	1.10	73.10
SkipNet Wang et al. (2018) <sup>S</sup>	0.7200	-	-	-	-	12.00
SSS Huang & Wang (2018) <sup>S</sup>	0.7182	0.9079	15.60	38.82	2.33	43.32
GAL-0.5 Lin et al. (2019) <sup>S</sup>	0.7180	0.9082	19.31	24.74	1.84	55.01
HRank Lin et al. (2020) <sup>S</sup>	0.71.98	0.9101	13.77	46.95	1.55	62.10
PED (pruned) ( <b>ours</b> )	0.7280	0.9094	12.40	51.49	2.03	50.53

Table 4: The top-1 and top-5 test accuracy, number of parameters (Par), and FLOPs on ImageNet data set between PED and those of the state-of-the-art methods. FLOPs is in billion.

we first train it on CIFAR-10 to achieve 0.9334 test accuracy. The results of applying PED on ResNet56 with  $N = 5$  stages are shown in Table 1 tagged by ResNet56. As we can see, without almost any dropping in the test accuracy, we can reduce the number of trainable parameters and FLOPs by 62.96% and 63.86%, respectively. Also, the effect of pruning in the inference running time (in mS) on one GPU (GeForce RTX 2080 Ti) of the ResNet56 for stages from 0 to 6 is given by 67, 45, 32, 32, 26, 22, 21, respectively (we have averaged over batch-test images with size 20). Please see the appendix for the running time of the other models and data sets.

**ResNet164:** Next we evaluate the performance of PED on ResNet164 model. We apply our algorithm to prune ResNet164 for CIFAR-10 (Table 1), CIFAR-100 (Table 2), and SVHN (Table 3). ResNet164 consists of 18 units with total number of 1.70 (M) trainable parameters and 254.94 (M) FLOPs. Similar to ResNet56, we first train this model on CIFAR-10 and achieve 0.9569 test accuracy. In Table 1, we have listed two ResNet164 pruned models. The first one, ResNet164-a is a compressed version of ResNet164 by running PED for  $N = 6$  stages, while ResNet164-b corresponds to  $N = 9$  stages. As we can see, ResNet164-a achieves 0.9519 test accuracy with 0.65 (M) parameters and 71.06 (M) FLOPs. For the CIFAR-100, a full ResNet164 model has top-1 test accuracy equals to 77.93%. Table 2 presents two compressed models of ResNet164 model, ResNet164-a and ResNet164-b corresponding to running PED respectively with  $N = 9$  and  $N = 11$  stages on CIFAR-100. Finally, Table 3 shows the pruning (with  $N = 2$  stages) result of a full ResNet164 model with 98.20% test accuracy trained for the classification of SVHN data set. No Cutout is used for SVHN experiment.

**DenseNet:** Next we focus on the DenseNet100-k12 with 100 layers,  $k = 12$  growth-rate (i.e., the number of output channels in each unit), and 48 skip-units. We apply PED to DenseNet100-k12 by first training DenseNet100-k12 on both CIFAR-10 with 0.9531 test accuracy, 0.77 (M) number of parameters, and 293.55 (M) FLOPs (Table 1) and CIFAR-100 with 0.7793 as the top-1 test accuracy, 0.80 (M) number of parameters, and 304.10 (M) FLOPs (Table 2). Similar to the ResNet experiments, we have presented two pruned DenseNet models in Table 1, DenseNet100-k12-a and DenseNet100-k12 corresponding to running PED with  $N = 10$  and  $N = 11$  stages, respectively. Also we have two models of DenseNet100-k12-a and DenseNet100-k12 in Table 2 corresponding to running PED with  $N = 5$  and  $N = 6$  stages, respectively. These experiments suggest that our proposed algorithm is competitive to the state-of-the-art methods of in one or more criteria of accuracy/parameters/FLOPs.

**ResNet50:** Finally, we present the experiment on ImageNet data set using ResNet50 in Table 4. ResNet50 consists of 4 blocks with 3, 4, 6, and 3 skip-units in each block. This model has 25.56 (M) trainable parameters and 4.11 (B) FLOPs. As we expected, two structured methods of SNIP and GraSP have better compression ratio, but they have not reported FLOPs. Compared to the structured methods of ThiNet, SkipNet, SSS, GAL-0.5, and HRank, our method, PED has better performance.



## REFERENCES

- G. Bellec, D. Kappel, W. Maass, and R. Legenstein. Deep rewiring: Training very sparse deep networks. In *Int. Conf. Learning. Rep.*, 2018.
- L. Bottou, M. Arjovsky, D. Lopez-Paz, and M. Oquab. Geometrical insights for implicit generative modeling. In *Braverman Readings in Machine Learning. Key Ideas from Inception to Current State*, pp. 229–268. Springer, 2018.
- Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.
- M. Courbariaux, Y. Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Adv. Neural Inf. Process. Sys.*, pp. 3123–3131, 2015.
- J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 248–255, 2009.
- E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Adv. Neural Inf. Process. Sys.*, pp. 1269–1277, 2014.
- T. DeVries and G. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.
- X. Ding, G. Ding, J. Han, and S. Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *AAAI Conf. Artificial Intelligence*, 2018.
- M. Figurnov, M. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 1039–1048, 2017.
- J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Int. Conf. Learning. Rep.*, 2019.
- T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.
- G. Ghiasi, T. Lin, and Q. Le. Dropblock: A regularization method for convolutional networks. In *Adv. Neural Inf. Process. Sys.*, pp. 10727–10737, 2018.
- Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.
- Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Adv. Neural Inf. Process. Sys.*, pp. 1379–1387, 2016.
- S. Han, H. Mao, and W. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding, 2016.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 770–778, 2016.
- Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *IEEE Int. Conf. Comp. Vision*, pp. 1389–1397, 2017.
- Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Int. Joint Conf. Artificial Intelligence*, pp. 2234–2240, 2018.
- Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 4340–4349, 2019.
- G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

- L. Hou, Q. Yao, and J. Kwok. Loss-aware binarization of deep networks. *Int. Conf. Learning. Rep.*, 2017.
- A. 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.
- J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 7132–7141, 2018.
- G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. In *European Conf. Comp. vision*, pp. 646–661, 2016.
- G. Huang, Z. Liu, L. Van Der Maaten, and K. Weinberger. Densely connected convolutional networks. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 4700–4708, 2017.
- G. Huang, S. Liu, L. Van der Maaten, and K. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 2752–2761, 2018.
- Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. In *Europ. Conf. Comp. Vision*, pp. 304–320, 2018.
- S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Int. Conf. Machine Learning*, pp. 448–456, 2015.
- M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Confs.*, 2014.
- A. Kolchinsky and B. Tracey. Estimating mixture entropy with pairwise distances. *Entropy*, 19(7): 361, 2017.
- A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Adv. Neural Inf. Process. Sys.*, pp. 1097–1105, 2012.
- A. Lee. *U-statistics: Theory and Practice*. Routledge, 2019.
- N. Lee, T. Ajanthan, and P. Torr. SNIP: Single shot network pruning based on connection sensitivity. In *Int. Conf. Learning. Rep.*, 2019.
- H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf. Pruning filters for efficient convnets. In *Int. Conf. Learning. Rep.*, 2017.
- T. Li, B. Wu, Y. Yang, Y. Fan, Y. Zhang, and W. Liu. Compressing convolutional neural networks via factorized convolutional filters. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 3977–3986, 2019.
- M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao. Hrank: Filter pruning using high-rank feature map. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 1529–1538, 2020.
- S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, pp. 2425–2432, 2018.
- S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 2790–2799, 2019.
- T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Zitnick. Microsoft coco: Common objects in context. In *European Conf. Comp. vision*, pp. 740–755, 2014.
- R. Linsker. Self-organization in a perceptual network. *computer*. pp. 105–117, 1988.
- J. Liu, S. Tripathi, U. Kurup, and M. Shah. Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey. *arXiv preprint arXiv:2005.04275*, 2020.
- Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *IEEE Int. Conf. Comp. Vision*, pp. 2736–2744, 2017.

- Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *Int. Conf. Learning. Rep.*, 2019.
- J. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *PIEEE Int. Conf. Comp. Vision*, pp. 5058–5066, 2017.
- J. Luo, H. Zhang, H. Zhou, C. Xie, J. Wu, and W. Lin. Thinet: pruning cnn filters for a thinner net. *IEEE Trans. Patt. Anal. Machine Intell.*, 41(10):2525–2538, 2018.
- E. Malach, G. Yehudai, S. Shalev-Shwartz, and O. Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. *arXiv preprint arXiv:2002.00585*, 2020.
- P. Merolla, R. Appuswamy, J. Arthur, S. K Esser, and D. Modha. Deep neural networks are robust to weight binarization and other non-linear distortions. *arXiv preprint arXiv:1606.01981*, 2016.
- D. Mocanu, E. Mocanu, P. Stone, P. Nguyen, M. Gibescu, and A. Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.
- P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *Int. Conf. Learning. Rep.*, 2017.
- H. Mostafa and X. Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Int. Conf. Machine. Learning.*, pp. 4646–4655, 2019.
- K. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learn. Unsup. Feature Learn.*, 2011.
- L. Paninski. Estimation of entropy and mutual information. *Neural computation*, 15(6):1191–1253, 2003.
- V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. What’s hidden in a randomly weighted neural network? In *Conf. Comp. Vision Patt. Recog.*, pp. 11893–11902, 2020.
- M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Europ. Conf. Comp. Vision.*, pp. 525–542. Springer, 2016.
- M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun. Sbnnet: Sparse blocks network for fast inference. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 8711–8720, 2018.
- A. Romero, N. Ballas, S. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.
- O. Shamir, S. Sabato, and N. Tishby. Learning and generalization with the information bottleneck. *Theoretical Comp. Science*, 411(29-30):2696–2711, 2010.
- N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. A sparse-group lasso. *Journal of Comp. Graph. Statistics*, 22(2):231–245, 2013.
- K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- P. Singh, V. Verma, P. Rai, and V. Namboodiri. Play and prune: adaptive filter pruning for deep model compression. In *Int. Joint Conf. Artificial Intelligence*, pp. 3460–3466, 2019.
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal. Machine Learning Research*, 15(1): 1929–1958, 2014.
- K. Sun, M. Li, D. Liu, and J. Wang. Igcv3: Interleaved low-rank group convolutions for efficient deep neural networks. *arXiv preprint arXiv:1806.00178*, 2018.

- C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 1–9, 2015.
- G. Székely and M. Rizzo. Energy statistics: A class of statistics based on distances. *Journal of stat. Plan. Inference*, 143(8):1249–1272, 2013.
- N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *IEEE Inf. Theory. Workshop. (ITW)*, pp. 1–5, 2015.
- A. Veit and S. Belongie. Convolutional networks with adaptive computation graphs. *arXiv preprint arXiv:1711.11503*, 2017.
- A. Veit, M. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Adv. Neural Inf. Process. Sys.*, pp. 550–558, 2016.
- Chaoqi W., Guodong Z., and Roger G. Picking winning tickets before training by preserving gradient flow. In *Int. Conf. Learning. Rep.*, 2020.
- H. Wang and M. Song. Ckmeans. 1d. dp: Optimal k-means clustering in one dimension by dynamic programming. volume 3, pp. 29, 2011.
- X. Wang, F. Yu, Z. Dou, T. Darrell, and J. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Europ. Conf. Comp. Vision*, pp. 409–424, 2018.
- W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Adv. Neural Inf. Process. Sys.*, pp. 2074–2082, 2016.
- Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. Davis, K. Grauman, and R. Feris. Blockdrop: Dynamic inference paths in residual networks. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 8817–8826, 2018.
- X. Xiao, Z. Wang, and S. Rajasekaran. Autoprune: Automatic network pruning by regularizing auxiliary parameters. In *Adv. Neural Inf. Process. Sys.*, pp. 13681–13691, 2019.
- G. Xie, J. Wang, T. Zhang, J. Lai, R. Hong, and G. Qi. Interleaved structured sparse convolutional neural networks. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 8847–8856, 2018.
- J. Ye, X. Lu, Z. Lin, and J. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *Int. Conf. Learning. Rep.*, 2018.
- R. Yu, A. Li, C. Chen, J. Lai, V. Morariu, X. Han, M. Gao, C.g Lin, and L. Davis. Nisp: Pruning networks using neuron importance score propagation. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 9194–9203, 2018.
- X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *IEEE Conf. Comp. Vision. Patt. Recog.*, pp. 6848–6856, 2018.
- H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Adv. Neural Inf. Process. Sys.*, pp. 3597–3607, 2019.
- Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In *Adv. Neural Inf. Process. Sys.*, pp. 875–886, 2018.

## A APPENDIX

The Appendix is organized as follows. We first present the result of applying PED on Tiny ImageNet data set. Next, we review other deep neural compression techniques which have been omitted from Section 2, and then we give some overview about the ResNet and DenseNet architectures. The pseudocode for the clustering sub-routine used by PED is next presented. We then provide more examples of inference running time for the other models and data sets. The details of the intermediate stages of PED for all the experiments are given in the subsequent sections.

Model (ResNet32)	Acc. ( $\Delta$ )	Par.	Red(%)	FLOPs	Red(%)
Deep-R Bellec et al. (2018)	0.5329 ( $\downarrow$ 9.60% )	0.28	85.00	-	-
GraSP W. et al. (2020)	0.5725 ( $\downarrow$ 5.64% )	0.28	85.00	-	-
SNIP Lee et al. (2019)	0.5633 ( $\downarrow$ 6.56% )	0.28	85.00	-	-
PED (pruned) ( <b>ours</b> )	0.5564 ( $\downarrow$ 5.92% )	0.48	74.50	278.91	74.15

Table 5: Comparison of classification test accuracy on Tiny-ImageNet data set, number of parameters, and FLOPs between PED and those of the state-of-the-art unstructured methods. “-” means no reported value.  $\Delta$  and  $\downarrow$  indicate the change and dropping in percentage of the test accuracy with respect to the base (full) model, respectively.

### A.1 RESNET32 ON TINY IMAGENET

In this section, we present the result of applying our pruning technique, PED on ResNet32 and for the Tiny-ImageNet data set Deng et al. (2009). This data set is a subset of the full ImageNet data set with 200 labels. Each class includes 500 and 50 images with size  $64 \times 64 \times 3$  for training and validation sets, respectively. In general, the Tiny-ImageNet classification task is more difficult than CIFAR-10/CIFAR-100 data set.

ResNet32 model is a collection of 32 layers and 3 blocks each with 5 residual units. We first train a ResNet32 model from (W. et al., 2020) with 1885032 parameters and 1078912200 FLOPs. However, we could not reproduce the test accuracy of full ResNet32 reported in W. et al. (2020) with 62.89% top-1 test accuracy. Our trained ResNet32 achieves 61.56% top-1 test accuracy and we used this model as the input for PED. For the purpose of (re)training, we use SGD optimization algorithm with momentum equals to 0.9. For the values of hyperparameters, we set epoch number to 160 for all stages, batch size to 128, and learning rate to 0.1 which is reduced by a factor of 0.1 in epochs 80 and 120. No Cutout is used for training of the full model. For retraining the pruned models, we have used Cutout with parameter 16 for stages from 1 to 8 and 32 for the rest of stages.

Table 5 presents the performance of PED in comparison with some structured pruning methods. Although the accuracy and the number of pruned parameters for SNIP and GraSP are better than PED, the SNIP and GraSP algorithms do not report any gain in the FLOPs number. In addition, they have started pruning a base model with 1.39% higher the test accuracy than the one is used for the full model in PED. This means that the accuracy of PED would be increased by 1 to 1.5% if the former base model could be used in the experiment.

Moreover, the experiment presented in Table 5 verifies our discussion about the structured and unstructured pruning methods. In summary, PED provides a competitive performance (or even better in CIFAR-10 and CIFAR-100 data set) than the structured methods such as GraSP and SNIP in terms of accuracy and parameter count, and in the same time it significantly reduces the FLOPs count in the base model (e.g., 74.15% reduction). Finally, Table 6 shows the result of applying PED for 11 pruning stages for classifying the Tiny-ImageNet data set.

### A.2 REVIEW OF PRIOR ART

Another recent paper, *Lottery Ticket Hypothesis* (Frankle & Carbin, 2019) has empirically observed that various deep neural networks possess sub-networks (referred to as “winning tickets”) such that if they are trained with the same initialization values used in the training of the original full network, they potentially achieve comparable performance to that of the full model (Winning the lottery” means that this goal is achieved). Other subsequent papers (Zhou et al., 2019; Ramanujan et al., 2020; Malach et al., 2020) tried to give more theoretical intuition about the Lottery Ticket Hypothesis. On the other hand, pruning through the removal of weights is accomplished by determining and removing less important weights in the final prediction of the model. However, finding the less important weights is a prohibitively costly problem due to its combinatorial nature. To get around this issue, various criteria such as the minimum norm of weights, the activation of the feature maps, information gain, and etc have been proposed in literature (Molchanov et al., 2017). Another recent paper (Lee et al., 2019) prunes the weights of a deep network by defining a measure called connection sensitivity and removes the less sensitive weights. Once the important weights are discovered, the pruned network is

Stage No.	Test Accuracy	Parameters	Parameters Red.(%)	FLOPs	Flops Red.(%)
0	0.6156	1885032	0.0000	1078912200	0.0000
1	0.6345	1866472	0.9846	1003021512	7.0340
2	0.6225	1847912	1.9692	927130824	14.0680
3	0.6170	1829352	2.9538	851240136	21.1020
4	0.6217	1755368	6.8786	775546056	28.1178
5	0.6131	1459944	22.5507	699950280	35.1244
6	0.6118	1441384	23.5353	624059592	42.1584
7	0.5940	1219432	35.3097	581493960	46.1037
8	0.5916	1145448	39.2346	505799880	53.1195
9	0.5920	1071464	43.1594	430105800	60.1352
10	0.5759	776040	58.8315	354510024	67.1419
11	0.5564	480616	74.5036	278914248	74.1486

Table 6: The results of pruning in intermediate stages for ResNet32 and Tiny-ImageNet data set.

retrained. Other network compression techniques are based on quantizing or binarizing the weights of a deep model which have also shown some promising results in practice; however, these methods potentially have low accuracy in large networks such as GoogleNet (Szegedy et al., 2015).

Another technique is based on low-rank approximation of the weight matrices/tensors, or sparsifying the weights (these may be considered as other forms of weight pruning) (Denton et al., 2014; Figurnov et al., 2017; Huang & Wang, 2018; Jaderberg et al., 2014; Ren et al., 2018; Sun et al., 2018). However, low-rank approximation of large weight tensors may be computationally expensive. Also, sparsification based on  $\ell_1$ -regularization usually requires a large number of iterations for convergence, and may not necessarily give a structurally sparse network. To resolve these issues, Wen et al. (2016) have introduced structured sparsity method similar to group lasso (Simon et al., 2013) on AlexNet. This is an interesting approach that has been only applied to the convolution layers, where the number of parameters are typically far less than those of the fully connected layers. In this light, it achieves 3.1% reduction in the number of parameters.

Knowledge Distillation (KD) is another network compression approach, which uses a student-teacher paradigm, where the knowledge of a teacher model is transferred to the student network by learning the distribution of output labels resulted by *softmax* function (Hinton et al., 2015; Romero et al., 2014). However, use of KD methods is limited to the classification tasks, and the cross-entropy loss function.

In recent years, a number of other innovative and efficient network architectures such as MobileNet V1/V2, CondenseNet, and IGC-V2\* C416 respectively proposed by Howard et al. (2017), Huang et al. (2018), and Xie et al. (2018) (CondenseNet and IGC-V2\* C416 are also considered as the state-of-the-art compressed architectures for the classification task) and other architectures given by He et al. (2016); Huang et al. (2017); Zhang et al. (2018) achieve the high test accuracy while giving even higher reductions in the number of parameters and FLOPs. For example, architectures in Huang et al. (2017); Zhang et al. (2018) achieve comparable performance as VGG for classification task on ImageNet, while reducing the computation load  $10\times$  and  $25\times$ , respectively. MobileNet v2 (Howard et al., 2017) can respectively achieve 11.15% test error and 31.1% top-1 error on CIFAR-10 and CIFAR-100 data sets with 1.4 million parameters. However, PED has far better performance both in terms of accuracy and parameters count. Please see Cheng et al. (2017); Liu et al. (2020) for recent comprehensive surveys on the deep compression techniques.

### A.3 OVERVIEW OF RESNET AND DENSENET ARCHITECTURES

We next briefly discuss two prominent examples of deep architecture with skip-connection, namely ResNet and DenseNet.

**Residual Networks (ResNet).** The idea of incorporating skip-connection (by-pass paths) in the layers of neural networks has been popularized by ResNet (He et al., 2016) family. This new architecture idea has achieved remarkable results in image competitions such as ImageNet, COCO challenge, etc. As discussed in Equation (1), in ResNet, the feature maps from the immediate previous

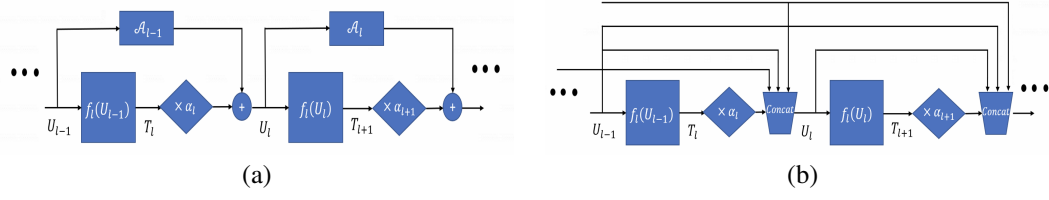


Figure 3: (a) Illustration of two consecutive skip-units in a ResNet family. (b) Illustration of two consecutive skip-units in a DenseNet family.

skip-unit (i.e., residual-unit) are added directly to the features in the next unit. Please see plot (a) in Figure 3. There exists various ResNet architecture, but all of them consist of a number of blocks consisting of multiple skip-units. Each unit is made by stacking two or three convolutional layers together with Batch Normalization (BN) (Ioffe & Szegedy, 2015) layers, and a skip-connection from the output of the previous unit to the output of the current unit. Depending on design, the skip-connection can either be passed through a convolution operation, or directly connected to the current unit (identity map).

**Densely Connected Networks (DenseNet).** Construction of the feature maps of units in ResNet-based models are based only on the immediate previous layer (as described above). In contrast, in DenseNet Huang et al. (2017) architecture, the input of each skip-unit is a concatenation of feature maps from all preceding units (Equation (2)). Similar to the ResNet family, DenseNet-based models consist of multiple dense blocks, each containing multiple skip-units (i.e., dense-unit). Also, the growth rate of the network,  $k$  is defined as the number of output features from each unit. In each unit, two sequences of *Conv-BN-ReLU* operations are applied to the input (of that unit). The convolutional layer in the first sequence ( $1 \times 1$  filters) reduces the number of channels, while the second one ( $3 \times 3$  filters) outputs  $k$  features which are further concatenated by all the features from the preceding units. The number of parameters in DenseNet architecture Huang et al. (2017) could be significantly less than the ResNet families. Please see plot (b) in Figure 3.

#### A.4 CLUSTERING ALGORITHM

The clustering algorithm in Algorithm 1 is given by Algorithm 2. In this algorithm,  $\text{Opt\_K\_Means}(k, \mathbf{D})$  refers to the optimal  $k$ -means algorithm based on dynamic programming due to Wang & Song (2011).

---

##### Algorithm 2 Clustering( $K, \mathbf{D}, S$ )

---

**INPUT:**

$A = \emptyset$

$K$ : Cluster vector

$\mathbf{D} = [D(T_1, Y), D(T_2, Y), \dots, D(T_{|S|}, Y)]$

$S$ : The Index set of current active skip-units

$\text{Opt\_K\_Means}(\cdot, \cdot)$  Wang & Song (2011)

**for**  $k$  in  $K$  **do**

$\{Cluster_1^k, \dots, Cluster_k^k\} = \text{Opt\_K\_Means}(k, \mathbf{D})$

$A = A \cup \{Cluster_1^k, \dots, Cluster_k^k\}$

**end for**

**Return**  $A$

---

#### A.5 INFERENCE RUNNING TIME OF THE PRUNED MODELS

In this section, we show the effect of pruning in clock-time (in millisecond (mS)) of ResNet models. In particular, we report the average clock-time on one GPU (GeForce RTX 2080 Ti) takes to pass batch test images with size 20 in each stage. We have round the numbers.

**ResNet56 on CIFAR-10.** The clock-time (in mS) for stages from 0 to 6 is respectively given by 67, 45, 32, 32, 26, 22, 21.

**ResNet164 on CIFAR-10.** The clock-time (in mS) for stages from 0 to 8 is respectively given by 31, 24, 18, 17, 15, 12, 11, 9.

**ResNet164 on CIFAR-100.** The clock-time (in mS) for stages from 0 to 10 is respectively given by 32, 28, 25, 21, 19, 18, 17, 17, 16, 16, 15.

**ResNet164 on SVHN.** The clock-time (in mS) for stages from 0 to 2 is respectively given by 100, 69, 64.

**ResNet50 on ImageNet.** The clock-time (in mS) for stages from 0 to 7 is respectively given by 27, 26, 23, 21, 20, 18, 17, 16.

## A.6 DETAILS OF EXPERIMENTS

For re-training in each stage and for both DenseNet and ResNet architectures (for the experiments on CIFAR-10, CIFAR-100, and SVHN data sets), we use SGD optimization algorithm with momentum equals to 0.9. The general setup of hyperparameters is given as follows: The learning rate is set to 0.1 and weight decay to  $1e-4$ . Also, batch size is set to 512 for SVHN and 128 for CIFAR-10 and CIFAR-100 data sets. The epoch number is set to 100 for CIFAR-10 and CIFAR-100 data sets and 120 for SVHN data set. Cutout() parameter is selected 16 for CIFAR-10 and 8 for CIFAR-100 data sets. No Cutout is used for SVHN data set. For the learning rate schedulers, we use *StepLR* PyTorch scheme for CIFAR-10/CIFAR-100 data sets and *MultiStepLR* PyTorch scheme for SVHN data set. To improve the test accuracy of the pruned model in different stages (i.e., re-training part), we have used different epoch numbers for diminishing (by a factor of 0.1) the learning rate (milestones) in MultiStepLR scheme from [40, 60] to [50, 80, 100], and we also have set the weight decay in a few stages to  $5e-4$ .

### A.6.1 RESNET56 FOR CIFAR-10 DATA SET

The ResNet56 architecture consists of 56 layers with 27 residual units arranged in 3 blocks, with one convolutional layer in the input of the network, and a fully connected one in the last layer of the model (after the 27<sup>th</sup> unit). Each unit has two convolutional layers with batch normalization. This model has 853018 training parameters and consists of 126550720 FLOPs.

Pruning of ResNet56 model within 10 stages for the classification of CIFAR-10 data set is given in Table 7. Furthermore, plots (a) in Figure 4 demonstrates the pattern of dropping of skip-units in 10 stages in ResNet56 model for CIFAR-10 data sets. In Figure 4, blue, orange, and green bars respectively correspond to the first, second, and third blocks. The horizontal axis gives the stage number, and the vertical axis corresponds to the number of active (i.e., the remainder of skip-units) units in each stage.

### A.6.2 PRUNING WITH RANDOM SELECTION OF UNITS AND WITH THE LARGEST ENERGY VALUES

As we mentioned in section 3.3, choosing units randomly or simply the units with the largest energy values is not a good strategy for pruning less important units. This is mainly because of the existence of skip-paths in the ResNet and DenseNet-type architectures, and violating the Markov chain property existing in the non-residual models such as VGG. As a result, the information between units and the output will not decrease monotonically according to the information processing inequality. This implies that by choosing randomly or only by the units with the largest energy values, we may remove important units; hence, degrading the performance. We have illustrated pruning performance affecting by the random selection in Table 8 and by selecting using the largest values in Table 9. To be fair in comparison with the clustering scheme used in PED (illustrated in Table 7), we have used the same number of units used in the clustering scheme for removing units randomly and with the smallest energy. That is, in stages from 0 to 10, we have removed 0, 4, 10, 14, 16, 17, 18, 19, 20, 21, 22 units, respectively. As we can see from tables 7, 8 and 9, the test accuracy, parameters reduction rate, and



Stage No.	Test Accuracy	Parameters	Parameters Red.(%)	FLOPs	Flops Red.(%)
0	0.9334	853018	0.00000	126550720	0.00000
1	0.9439	737242	13.5725	107528896	15.0310
2	0.9424	528922	37.9940	79020736	37.5581
3	0.9386	413146	51.5666	59998912	52.5890
4	0.9363	389914	54.2901	50463424	60.1240
5	0.9336	315930	62.9633	45728448	63.8655
6	0.9323	311258	63.5110	40944320	67.6459
7	0.9284	237274	72.1842	36209344	71.3875
8	0.9304	232602	72.7319	31425216	75.1679
9	0.9252	218650	74.3675	27853504	77.9902
10	0.9100	144666	83.0407	23118528	81.7318

Table 7: The results of pruning in intermediate stages for ResNet56 and CIFAR-10 data set.

Stage No.	Test Accuracy	Parameters	Parameters Red.(%)	FLOPs	Flops Red.(%)
0	0.93.34	853018	0.00000	126550720	0.00000
1	0.94.69	737242	13.5725	107528896	15.0310
2	0.93.96	528922	37.9940	79020736	37.5581
3	0.93.77	482458	43.4411	59949760	52.6279
4	0.93.47	389914	54.2901	50463424	60.1240
5	0.92.98	385242	54.8378	45679296	63.9044
6	0.93.05	311258	63.5110	40944320	67.6459
7	0.92.65	297306	65.1466	37372608	70.4683
8	0.91.48	241754	71.6590	33817280	73.2777
9	0.90.51	167770	80.3322	29082304	77.0193
10	0.87.32	93786	89.0054	24347328	80.7608

Table 8: The results of pruning using random selection of units in intermediate stages for ResNet56 and CIFAR-10 data set.

Stage No.	Test Accuracy	Parameters	Parameters Red.(%)	FLOPs	Flops Red.(%)
0	0.9334	853018	0.00000	126550720	0.00000
1	0.9343	792666	7.0751	107512512	15.0439
2	0.9340	709082	16.8737	78938816	37.6228
3	0.9279	570202	33.1548	61080256	51.7346
4	0.9264	491546	42.3757	51561152	59.2565
5	0.9264	486874	42.9234	46777024	63.0369
6	0.9191	472922	44.5590	43205312	65.8593
7	0.9012	454362	46.7348	38453952	69.6138
8	0.9068	380378	55.4080	33718976	73.3554
9	0.9070	375706	55.9557	28934848	77.1358
10	0.8994	301722	64.6289	24199872	80.8773

Table 9: The results of pruning using units with the largest energy values in intermediate stages for ResNet56 and CIFAR-10 data set.

FLOPs reduction rate in pruning using random selection and using the largest values are worse than the clustering approach, which favors the clustering approach rather than simply dropping the units randomly or with smallest energy values.

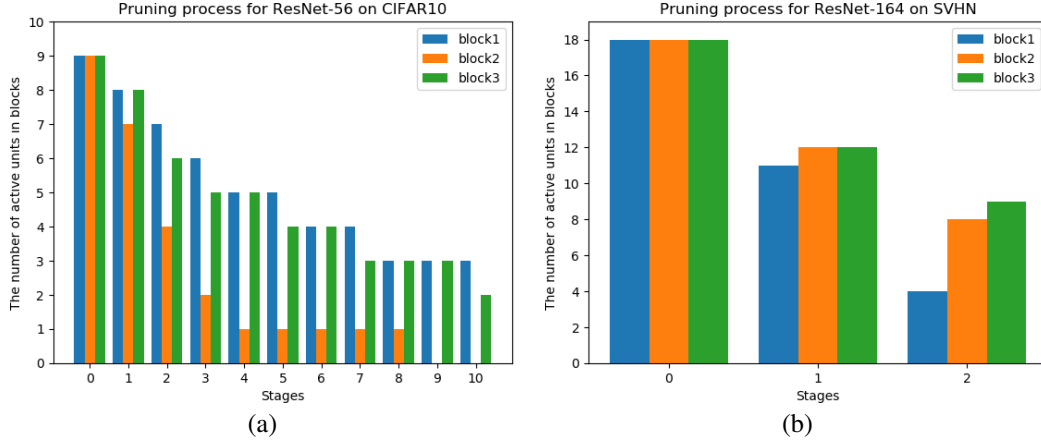


Figure 4: The pattern of removing skip-units across various pruning stages by PED. (a) ResNet56 model and CIFAR-10 data set. (b) ResNet56 model and SVHN data set.

#### A.6.3 RESNET164 AND CIFAR-10 DATA SET

ResNet164 is a collection of 164 layers grouped in 3 blocks each with 18 units. This model has 1703258 trainable parameters and 254941706 FLOPs.

Table 10 shows the results of running PED for 9 stages on ResNet164 model for the classification of CIFAR-10 data set. In each stage, we have reported the number of parameters and FLOPs in the pruned model together with the rate of reduction. As we can see, with  $N = 6$  stages, we can achieve 61.78% and 72.13% reduction on these quantities compared to the original ResNet164 model without any pruning. Running PED for extra 3 stages (i.e., 9 pruning stages) leads to the ResNet164-b architecture with less number of parameters (484154) and FLOPs (48929290). The test accuracy of (with  $N = 9$  stages) only drops by 1.4%. Also, we have shown the pattern of redundant skip-units removal in the 9 stages (Please see Plot (a) of Figure 5).

Stage No.	Test Accuracy	Parameters	Parameters Red.(%)	FLOPs	FLOPs Red.(%)
0	0.9569	1703258	0.000000	254941706	0.000000
1	0.9575	1398554	17.889480	213858826	16.114617
2	0.9578	1155162	32.179271	167918090	34.134712
3	0.9549	942810	44.646671	126433802	50.406780
4	0.9559	911386	46.491606	107805194	57.713787
5	0.9533	879962	48.336541	89176586	65.020793
6	0.9519	650970	61.780893	71064074	72.125363
7	0.9456	628634	63.092262	61807114	75.756374
8	0.9455	624954	63.308319	57956874	77.266617
9	0.9426	484154	71.574829	48929290	80.807656

Table 10: The results of pruning in intermediate stages for ResNet164 and CIFAR-10 data set.

As was discussed before, ResNet164 consists of three blocks each with 18 skip-units (i.e., residual-units). In Plot (a) of Figure 5, blue, orange, and green bars respectively correspond to the first, second, and third blocks. The horizontal and the vertical axes represent the stage number, and the the number of active units (i.e., the remainder of skip-units) in each stage, respectively. The pruning strategy demonstrates that the units in the second block are first dropped. Towards the end of the process, it can be seen that the first block is subject to more drops than the third block.

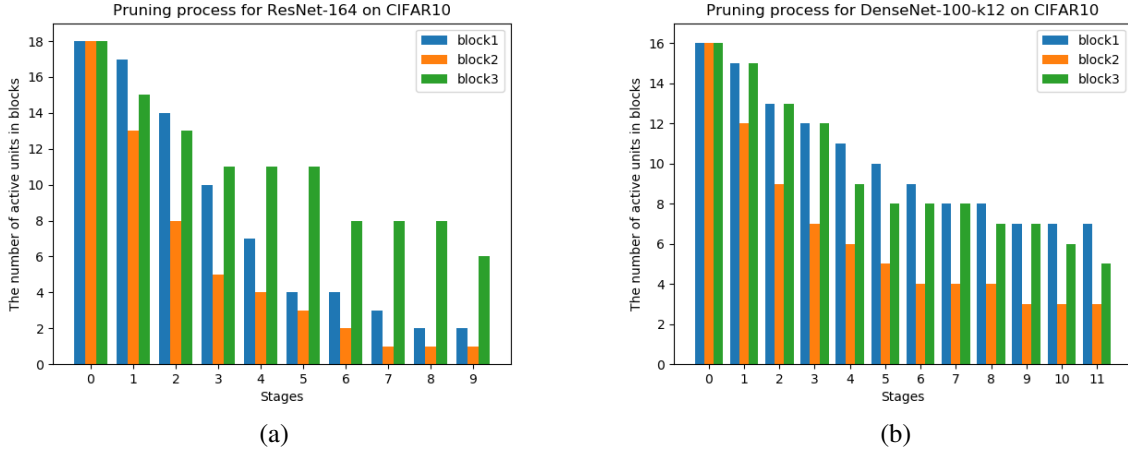


Figure 5: The pattern of removing skip-units across various pruning stages by PED. (a) ResNet164 model and CIFAR-10 data set. (b) DenseNet100-k12 model and CIFAR-10 data set.

Stage No.	Test Accuracy	Parameters	Parameters Red.(%)	FLOPs	FLOPs Red.(%)
0	0.9531	769162	0.000000	293546820	0.000000
1	0.9574	688582	10.476337	267992772	8.705272
2	0.9539	582022	24.330375	224525508	23.512880
3	0.9532	515602	32.965747	199341636	32.092047
4	0.9502	440302	42.755622	180976836	38.348221
5	0.9468	395362	48.598345	161959236	44.826779
6	0.9445	364402	52.623505	144605508	50.738520
7	0.9457	353722	54.012029	132501828	54.861774
8	0.9461	333142	56.687668	131124420	55.331003
9	0.9463	308782	59.854751	118679748	59.570419
10	0.9432	292402	61.984341	117731652	59.893399
11	0.9425	273022	64.503967	116512452	60.308733

Table 11: The results of pruning in intermediate stages for DenseNet100-k12 and CIFAR-10 data set.

#### A.6.4 DENSENET100-K12 FOR CIFAR-10 DATA SET

In this section, we provide the intermediate results in pruning DenseNet100-k12 model for the classification of CIFAR-10 data set. A full trained DenseNet100-k12 on CIFAR-10 has 769162 trainable parameters, 293546820 FLOPS, and test accuracy of 0.9531. Table 11 gives the results of running trainable parameters and the number of FLOPs by PED for 11 stages.

DenseNet100-k12 consists of 3 blocks each with 16 skip-units (i.e., dense-units). This model has 48 skip-units grouped in 3 blocks (16 units in each block). In addition, there are two-transition layers between the blocks. There is also a convolutional layer in the beginning and before the first unit in the first block, and a fully connected layer at the end of the network after the last unit in the third block. Each dense unit consists of *BN-ReLU-Conv* operations that form the *Bottleneck* architecture. Each transition-layer between the blocks is a sequence of a batch normalization followed by a convolution layer. Since the number of output channels in each unit is  $k = 12$ , we need to make sure that 12 channels are added to the previous active unit. This is because the input of the transition-layer expects  $k \times 16$  channel for the convolution operation. One possible remedy for the dimension inconsistency when removing a unit is to pad the output of the previous active unit with  $k$  zero channels. We note that this zero padding does not increase the number of parameters in the pruned model. Also by implementing sparse convolution, it can be assumed that these zero channels do not increase the FLOPs operation.

Stage No.	Test Accuracy	Parameters	Parameters Red.(%)	FLOPs	FLOPs Red.(%)
0	0.7799	1726388	0.000000	254964836	0.000000
1	0.7629	1509876	12.541329	222966884	12.549947
2	0.7611	1254004	27.362563	190911588	25.122385
3	0.7641	1196084	30.717544	172512356	32.338765
4	0.7549	953460	44.771396	140342372	44.956185
5	0.7516	808116	53.190360	126628964	50.334734
6	0.7480	667316	61.346117	117601380	53.875451
7	0.7477	579124	66.454586	108516452	57.438660
8	0.7462	561332	67.485177	103945316	59.231509
9	0.7499	557652	67.698339	100095076	60.741615
10	0.7480	539860	68.728930	95523940	62.534465
11	0.7402	469460	72.806808	91010148	64.304824

Table 12: The results of pruning in intermediate stages for ResNet164 and CIFAR-100 Data Set.

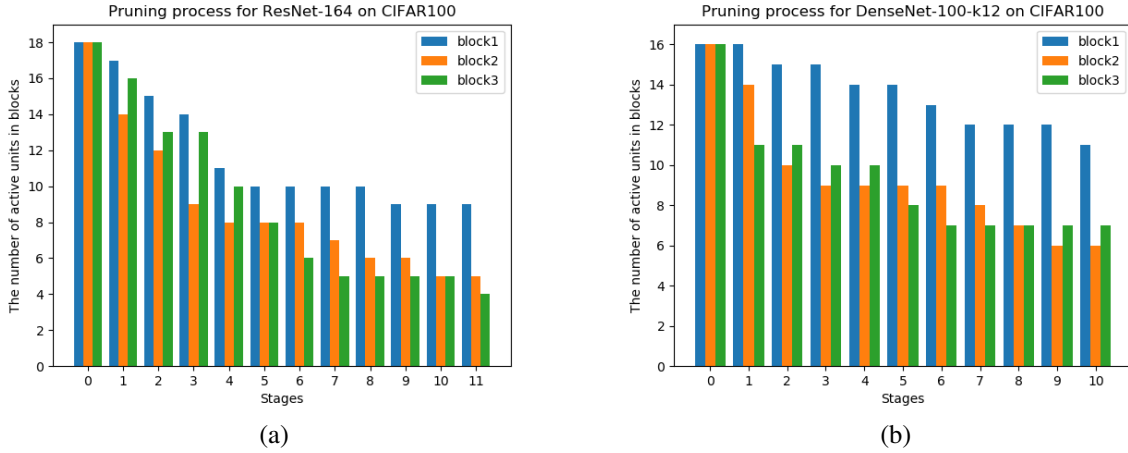


Figure 6: The pattern of removing skip-units for various pruning stages by PED. (a) ResNet164 model and CIFAR-100 data set. (b) DenseNet100-k12 model and CIFAR-100 data set.

In Plot (b) of Figure 5, blue, orange, and green bars respectively correspond to the first, second, and third blocks. The pruning strategy indicates that the units in the second block are first dropped. However, in latter steps, the third block is subject to slightly more unit droppings than the first block.

#### A.6.5 RESNET164 AND DENSENET100-K12 FOR CIFAR-100 DATA SET

Table 12 and Plot (a) in Figure 6 respectively demonstrate the intermediate pruning results of ResNet164 model for the classification of CIFAR-100 data set and intermediate pruning results for 11 stages of the applications of our algorithm.

Finally, Table 13 and Plot (b) in Figure 6 respectively demonstrate the intermediate pruning results of DenseNet100-k12 model for the classification of CIFAR-100 data set and intermediate pruning results for 10 stages of the applications of our algorithm.

Figure 6 illustrates an analogous pruning pattern compared to the previous case. The second block has largest number of dropped skip-units, while blocks 3 and 1 are respectively ranked as the second and third.

Stage No.	Test Accuracy	Parameters	Parameters Red.(%)	FLOPs	FLOPs Red.(%)
0	0.7793	800032	0.000000	304104360	0.000000
1	0.7634	665572	16.806828	288290856	5.200025
2	0.7617	585772	26.781429	256985640	15.494260
3	0.7573	548512	31.438742	251710632	17.228864
4	0.7563	539632	32.548698	241413288	20.614986
5	0.7526	506872	36.643534	239268264	21.320344
6	0.7488	473812	40.775869	221888808	27.035309
7	0.7436	444652	44.420723	203843880	32.969103
8	0.7422	429172	46.355646	199945512	34.251021
9	0.7413	418492	47.690592	196919592	35.246048
10	0.7414	412012	48.500560	189030696	37.840189

Table 13: The results of pruning in intermediate stages of execution of our algorithm for DenseNet100-k12 and CIFAR-100 data set.

Stage No.	Test Accuracy	Parameters	Parameters Red.(%)	FLOPs	FLOPs Red.(%)
0	0.9821	1704154	0.00000	254176000	0.00000
1	0.9828	1145242	32.7970	166144768	34.6340
2	0.9815	831066	51.2329	101837568	59.9342

Table 14: Comparison of classification test accuracy using ResNet164 on SVHN data set, number of parameters, and number of FLOPs between the various stages of proposed pruning method and those of the state-of-the-art deep compression methods.

#### A.6.6 RESNET164 FOR SVHN DATA SET

Pruning of ResNet164 model within 2 stages for the classification of SVHN data set is given in Table 14. SVHN data set (Netzer et al., 2011) consists of  $32 \times 32$  colored digit images with 73257 original training images, 531131 extra training images (604,388 total training examples), and 26032 test images. The full model ResNet164 has been trained on the SVHN data set to achieve 98.20% test accuracy. This model has 1704154 parameters and 254176000 FLOPs. There is no usage of Cutout in the SVHN data set. Finally, plot (b) in Figure 4 illustrates the pattern of dropping units inside of each block in 2 stages pruning of ResNet56.

#### A.6.7 RESNET50 FOR IMAGENET DATA SET

We have provided the details of intermediate stages of running PED for pruning ResNet50 and for classification of ImageNet data set in Table 15. Moreover, the pattern of removing skip-units from the architecture of ResNet50 has been illustrated in Figure 7. In this Figure, blue, orange, green, and red bars respectively correspond to the first, second, third, and fourth blocks. The pruning strategy indicates that in the early stages, intermediate blocks (block 2 and 3) are more subject to be pruned, while in the later stages first and last block will drop more units.

For running the ImageNet experiment, we have used Cutout with parameter 56 only for the retraining stages and not for the full model. The batch size is set to 256 for all the stages. The total number of epochs is set to 150. In addition, we have set the weight decay hyperparameter to  $1e-4$ . Initial learning rate is set to 0.1 and is decreased by a factor of 0.1 every 35 epochs. Finally, SGD optimizer with momentum equals to 0.9 is used for all the stages.

Stage No.	Top-1 Acc.	Top-5 Acc.	Parameters	Red.(%)	FLOPs	Red.(%)
0	0.7613	92.862	25557032	0.00	4111514624	0.00
1	0.7674	0.9332	24570920	3.86	3840865280	6.58
2	0.7570	0.9263	18711080	26.79	3183660032	22.57
3	0.7525	0.9236	17523496	31.43	3183660032	22.57
4	0.7463	0.9202	16406312	35.81	2743917568	33.26
5	0.7437	0.9174	16348200	36.03	2524949504	38.59
6	0.7280	0.9094	12397608	51.49	2033977344	50.53
7	0.7095	0.8968	11280424	55.86	1815009280	55.86

Table 15: The results of pruning in intermediate stages of execution of our algorithm for ResNet50 and ImageNet data set.

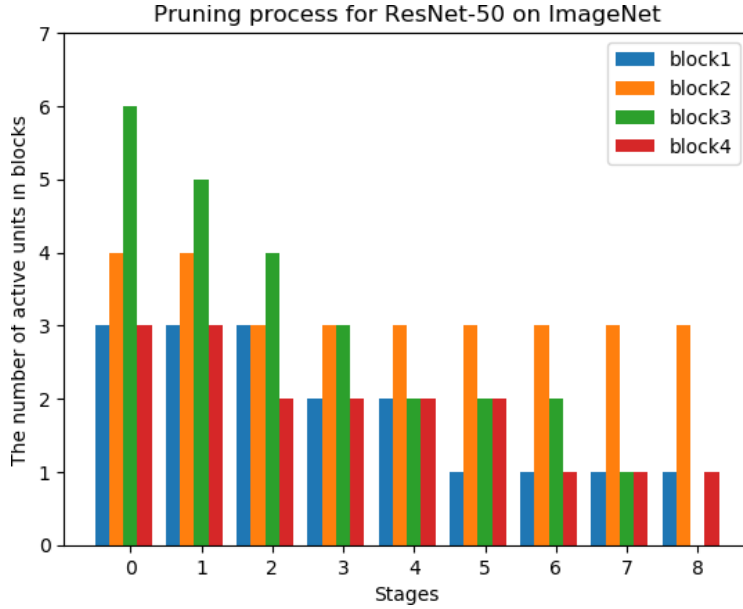


Figure 7: The pattern of removing skip-units for various pruning stages by PED for ResNet50 and ImageNet data set.