On the Information of Feature Maps and Pruning of Deep Neural Networks

Mohammadreza Soltani*, Suya Wu*, Jie Ding†, Robert Ravier*, Vahid Tarokh*
Department of Electrical and Computer Engineering, School of Statistics†*
Duke University, University of Minnesota†*
Email: {mohammadreza.soltani, suya.wu, robert.ravier, vahid.tarokh}@duke.edu*, dingj@umn.edu†

***Abstract*—A technique for compressing deep neural models achieving competitive performance to state-of-the-art methods is proposed. The approach utilizes the mutual information between the feature maps and the output of the model in order to prune the redundant layers of the network. Extensive numerical experiments on both CIFAR-10, CIFAR-100, and Tiny ImageNet data sets demonstrate that the proposed method can be effective in compressing deep models, both in terms of the numbers of parameters and operations. For instance, by applying the proposed approach to DenseNet model with 0.77 million parameters and 293 million operations for classification of CIFAR-10 data set, a reduction of 62.66% and 41.00% in the number of parameters and the number of operations are respectively achieved, while increasing the test error only by less than 1%.**

***Index Terms*—Deep neural compression, mutual information, feature maps**

I. Introduction

Deep Neural Networks (DNNs) require intensive computational load and memory. This hinders their deployment in embedded systems with limited computational and memory resources (such as in some mobile devices and sensor networks). Many accurate award-winning deep convolutional neural networks (CNNs) in image competitions (e.g., *COCO* [37] and *ImageNet* [4]) have millions or even billions of parameters. For instance, AlexNet [32] and VGG-16 [53] Caffemodels require more than 200MB, and 500MB memory space, respectively [27]. Similarly, ResNet50 needs respectively more than 95MB memory for storage and 3.8×10^9 floating number multiplications for processing each image [63]. This combination of high memory storage requirements and high number of floating point operations per second (FLOPs) provides serious challenges towards incorporating deep neural models in hardware-constrained devices. Yet another obstacle is energy consumption; large neural networks require large bandwidths to fetch parameters and perform intensive algorithmic operations [12] . This has motivated a body of research in developing computationally efficient deep neural models.

A. Motivation

Recent empirical evidence has shown large redundancies in various deep neural models. It is well-known that the removal of many of the parameters of a trained network, various layers, features of convolutional networks, etc. does not cause much performance degradation [2], [22]. Intuitively, while many deep models are significantly over-parameterized, overfitting is typically not a matter of concern on the test data set [70]. Thus, dropping of some parameters should cause minimal performance degradation.

In order to remove the redundant weights and hence, compression the deep models, various approaches based on pruning, sparsification, and quantization of network parameters have been recently proposed. We will review some of these deep compression approaches in Section II.

B. Our contribution

Typically, the largest portion of memory storage requirements in DNNs stems from the dense layers, while the computational bottleneck is mainly dominated by operations from the convolutional layers. Our proposed approach for deep network compression is based on successive coarse pruning and re-training of deep neural models. In particular, we exploit two concepts introduced in deep learning literature: robustness of deep architectures with *skip-connection* against coarse pruning, and the *Information Bottleneck* (IB) framework [51] based on the mutual information (MI) between hidden feature maps and the output of the network. In addressing the first, we focus on deep architectures where the feature maps in previous layers directly contribute to forming the feature maps in the subsequent layers (i.e., skip-connection). Prominent examples of these architectures include residual networks (ResNet) [14] and densely connected convolutional networks (DenseNet) [23]. Many high performance deep models in image/video applications are based on these concepts. Our proposed approach has been inspired by the empirical observation about the ResNet family models [60], namely that they are robust to the removal of layers. In contrast, the removal of a single layer in non-residual networks (such as VGG) can cause significant performance loss. Our approach builds on this idea for deep compression. Somewhat surprisingly, we show that coarse pruning strategy—removing a residual unit in ResNet, or a dense unit in DenseNet (where the notion of 'unit' is defined later), instead of dropping of individual layers, does not have significant adverse effects on performance. We will also propose an algorithm to determine the removal or maintenance of a number of units in a trained deep model based on the mutual information between learned features and the output of the model. Our proposed method can be used for both classification and regression tasks; the only requirement

This work was supported in part by DARPA Grant No. HR00111890040.

of our algorithm is the existence of skip-connection among different layers. Our contributions are summarized below:

- We propose an iterative coarse pruning compression technique for deep neural models with skip-connection with competitive performance with state-of-the-art methods.
- We utilize the idea of information of feature maps in Information bottleneck framework by apply mutual information for quantifying the amount of redundant information in the learned feature maps.
- We illustrate efficacy of our approach in reducing the network complexity both in terms of memory usage and the computational complexity by providing extensive experimental results for classification on the CIFAR-10/100 [31], and Tiny ImageNet [4] data sets.

II. Prior Work

Deep compression has received a large amount of attention that has resulted in a comparatively large body of work. Please see the two recent surveys [2], [39] for more comprehensive review on deep compression methods. Here, we only briefly review those directly relevant to our paper.

We will first discuss prior work on pruning techniques [1], [3], [7], [10]–[12], [15]–[17], [19], [33], [35], [36], [40]–[42], [44], [47], [54], [64], [66], [68]. In general, pruning techniques are divided into structured and unstructured methods. Structured methods remove individual weights in a deep model without imposing any structure for the pruning process. Unstructured methods, on the other hand, prune a set of weights with a specific structure such as filters and layers. Structured methods typically result in higher compression ratio in the number of parameters, while they do not usually provide any gain in the FLOPs number. Also, these methods are hard to be deployed efficiently to parallel processors. However, structured methods such as the proposed one in this paper, are easier for palatalization and usually provide a significant reduction in the FLOPs count. Although, their memory improvement is not generally as high as the unstructured approaches.

In addition, pruning techniques are motivated by the empirical observation that DNNs are surprisingly robust to weight quantization, additive and multiplicative noise corrupting the weights, projection, etc. [42]. These approaches reduce both network complexity and overfitting by means of quantization and binarization, and are intimately related to common training methods for deep networks such as *drop-out* [55], *ReLU* activation function, etc. Though quantization is a promising compression approach, it may result in low accuracy when used in large networks such as *GoogleNet* [56]. Additionally, pruning based on ℓ_1-regularization usually requires a larger number of iterations to converge. Two more recent pruning approaches were proposed in [34] and [9]. [34] (an unstructured method) sparsifies the network based on the importance of the weights by defining a measure called connection sensitivity; less sensitive weights are removed, more important weights are kept, and the pruned network is subsequently retrained. In [9] (*Lottery Ticket Hypothesis*), authors empirically observed that various deep neural networks possess sub-networks (referred to as "winning tickets") that potentially achieve comparable performance to the full network if these individual subnetworks were extracted and re-trained with the same initialization of the full network.

Another approach is based on low rank approximation of the weight tensors of each layer, or the sparsification of weights in order to achieve a compressed version of the original model [5], [8], [24], [26], [48], among others. These methods (considered as structured) may be computationally expensive due to the cost of computing low-rank decompositions of large weight tensors. Yet another method is knowledge distillation (KD), where a student-teacher paradigm is employed. The knowledge of a larger network (teacher) is shifted to a smaller model (student) by learning the distribution of output classes produced by *softmax* function [18], [49]. A major disadvantage of KD methods is that they are only applicable for classification tasks using the cross-entropy loss function. Another class of methods is based on dynamic path selection, where a sub-graph of the original network is selected and is used as a proxy for inference tasks [21], [59], [62], [65]. Many papers in this class have focused on residual networks, with their goal being to dynamically activate/deactivate the residual units using learning-based approaches such as policy update in reinforcement learning [62], [65], which is a potentially expensive process. Finally, other efficient architectures were recently proposed [14], [20], [22], [23], [67], [71]. The methods of [23] and [71] achieve comparable performance to VGG for the task of classifying *ImageNet*, while reducing the amount of computation by factors of 10 and 25, respectively. Notably, Huang et al., [22] have achieved better accuracy with even higher reductions in the number of parameters and FLOPs. Our approach in this paper is an structured method and is close to those with dynamic path selection; albeit, with very different philosophy.

III. Background and Proposed Approach

Throughout this paper, we represent random variables and their realizations with respectively uppercase and lowercase letters. We let $P(.)$, $\mathbb{E}(.)$, and $\mathbb{1}(.)$ respectively denote probability, expectation operator and indicator function. We use $I(.)$ and $H(.)$ to denote mutual information and entropy, respectively. In addition, $[M] = \{1, .2, \ldots, M\}$ for any $M \in \mathbb{N}$. Finally, the cardinality of a set A is denoted by $|A|$.

A. Residual Networks (ResNet)

The idea of designing by-pass paths or skip-connection in the layers of neural networks is the key to the success of Residual Networks [14]. These networks have achieved record performance in ImageNet and *COCO* challenges for various computer vision tasks such as image classification, localization, and object detection. In its simplest form, the output of layer l, x_l is given by

$$x_l = \mathcal{F}(x_{l-1}) + x_{l-1},$$

where $\mathcal{F}$ is a nonlinear map from the output of a layer (such as $(l-1)^{th}$) layer to the next (e.g. l^{th}) layer, and the addition with

x_{l-1} models the skip-connection. Thus raw features of the previous layer are explicitly used to form the current features. The existence of this skip-connection eases the training of a very deep network by alleviating the gradient vanishing problem. Typically a Residual Network consists of a number of blocks consisting of multiple residual units. Each residual unit is made by stacking two or three convolutional layers together with *Batch Normalization* (BN) [25] layers, and a skip-connection from the output of the previous residual unit to the output of the current unit. This skip-connection may either go through another convolution or down-sampling operation, or may directly connect to the current unit (identity map).

B. Densely Connected Networks (DenseNet)

A similar idea to skip-connection is further investigated in DenseNet [23]. One of the characteristics of the ResNet-based models is that they only use the immediate previous layer in constructing the current feature map. Thus, these models ignore the effect of the feature maps in other prior layers. Moreover, the exploitation of the previous layer feature in ResNet is by the 'addition' operation. In contrast, the input of each DenseNet layer is a concatenation of all feature maps generated by all preceding layers (as opposed to the ResNet which adds them together). Similar to the ResNet family, DensNet consists of multiple dense blocks, each containing multiple dense units. The number of output features from each unit is called the growth rate of the network and is denoted by k. There are two sequences of *Conv-BN-ReLU* operations in each dense unit. The first convolutional layer (1×1 filters) reduces the number of channels, while the second one (3×3 filters) outputs k features which are further concatenated by all the features from the previous units. It has been shown in [23] that the number of parameters used in this architecture could be significantly less than the ResNet families.

C. Information Bottleneck Framework

In this section, we review the Information Bottleneck (IB), a theoretical framework proposed by Tishby [51], [57] for quantifying the notion of *meaningful* information, i.e., extracting information from one variable which is as relevant as possible for prediction of the other variable. In the context of deep neural networks, Tishby [58] showed that mutual information (MI) between the layers and the input and output of any DNN is a strong quantitative measure. Shannon's mutual information, $I(X;Y)$ is considered as a measure of relevant information provided by random variable Y about X. Intuitively, the best representation of X is the one that captures the most relevant features in X and removes the irrelevant features by compressing X. The IB framework looks for the most concise representation of input X using another random variable T (hidden feature map) that is the most relevant to the output Y of the network [51]. Consider a multi-classification scenario given by a set of input-label pairs of length n denoted by $(x_i, y_i)_{i=1}^n \in \mathcal{X} \times \{0, 1, \ldots, l-1\}$ drawn from the joint distribution $P(x, y)$. Also consider an auxiliary variable T (representing the hidden feature map in the network). In this context, the Information Bottleneck Theory looks for a feature map T (representing a compressed representation of X) that is as informative as possible about output Y. The overarching goal is to classify a new input point x' (with label y') sampled according to the $P(t|y')$. One way to choose the most probable label can be found by minimizing the Kullback-Leibler (KL) divergence as: $y^* = \min_{y \in \{0,1,\ldots,l\}} D_{KL}(P(t)||P(t|Y = y'))$, where $D_{KL}(.)$ denotes the KL-divergence. The relation of this minimization problem with MI between a hidden layer T and the output Y has been established in [57], which makes the MI a relevant measure between hidden layers and the output of a deep model. In this paper, we look at the MI between units and the output as a measure to determine the important units.

D. Estimating the Mutual Information

Our pruning approach is based on computing Shannon's mutual information between hidden layers and the output of a DNN. To this end, we need to estimate $I(T;Y)$ efficiently. Computing the mutual information between the output and the hidden layers of DNNs has been a research topic with a long history in unsupervised feature learning [38], and variable ranking or screening [46]. Mutual information is an appealing mathematical measure because of its invariance with respect to smooth invertible transformations. However, estimating the mutual information is a challenging problem for high dimensional data. Various non-parametric methods, including k-nearest neighbors (KNN) [30], kernel density estimation [13], and histogram-based [52] methods have been proposed for estimating the mutual information. Parametric approaches based on Gaussian mixture models have also been studied in [28]. In the DNN literature, most previous work have used the histogram approach by binning the samples and feature maps. These methods become more accurate if the bins are not very coarse. On the other hand, they are not computationally efficient for large number of bins. Another issue with binning is that different binning schemes may give significantly different estimates of mutual information. Here, we use the parametric approach of [28] (later applied in the information bottleneck framework by [50]) for estimating the mutual information in our proposed method. This method is based on estimating the entropy of mixture models. We briefly review this method (Please see [28] for more details). Consider the differential entropy of a continuous random vector $T \in \mathbb{R}^d$ defined as $H(T) = -\int P(t) \ln P(t) dt$, where $P(t)$ is given by a Gaussian mixture model (GMM) as $P(t) = \sum_{i=1}^N c_i P_i(t)$, where $P_i(t)$ is the Gaussian density with mean μ_i and covariance matrix $\sigma^2 \mathbf{I}_d$ for some $\sigma > 0$ and $\sum_{i=1}^N c_i = 1$. Also, coefficients c_i's form probability values for a discrete random variable C with N possible outcomes such that $P(C = i) = c_i$; hence, $H(T|C) = \sum_{i=1}^N H(P_i) c_i$. Modeling the distribution of T can using Gaussian mixture models has been discussed in Appendix C of [50], and also in Section 2 of [29]. Saxe and others [50] derived both an upper and a lower bound for the differential entropy using the Gaussian mixture model. We can apply their results and derive

the following upper bound on the mutual information, $I(T;Y)$ (Please see the appendix.):

$$
\begin{aligned}
I(T;Y) &= H(T) - H(T|Y) \\
&\leq -\frac{1}{n}\sum_{i=1}^{n} \ln \frac{1}{n}\sum_{j=1}^{n} \exp\Big(-\frac{||\mu_j - \mu_i||_2^2}{2\sigma^2}\Big) \\
&- \sum_{k=0}^{l-1} p_k \Big(-\frac{1}{n_k} \sum_{\substack{i=1 \\ y_i=k}}^{n} \ln \frac{1}{n_k} \sum_{\substack{j=1 \\ y_j=k}}^{n_k} \exp\Big(-\frac{1}{4}\frac{||\mu_j - \mu_i||_2^2}{2\sigma^2}\Big)\Big),
\end{aligned}
$$

where $P_k = \frac{n_k}{n}$ denotes the probability of class k and $n_k = \sum_{i=1}^{n} \mathbb{1}(y_i = k)$.

We next present our proposed pruning approach.

E. Proposed Approach

The work [60] empirically observed that the pre-trained ResNet architectures are very robust to removal of layers, while in trained non-residual networks (i.e., there is no skip-connection) such as VGG, removing even a single layer causes a significant drop in performance. One possible explanation is that in ResNet-type architectures, there are alternative paths for the flow of gradient from input to the output of the network through skip-connection. We investigate this phenomenon for a broad range of deep models. Instead of removing just the individual layers, we show that even dropping a whole unit (defined below) in the models that have skip-connection (such as ResNet, or DenseNet) does not cause significant performance degradation. This pruning strategy results in significant reduction in the size of network parameters, and inference running time.

Before discussing the details of our approach, we will first introduce some notation. Consider a DNN with L units. In our terminology, a unit is a set of consecutive operations such as Convolution (*Conv*), Batch Normalization (*BN*), *ReLU*, Pooling, Dropout, etc., such that its output is combined in some manner with the feature maps from the previous unit(s). Let U_{l-1} denote the input of l-th unit. Also the result of applying the above operations consecutively on U_{l-1} inside of a unit, and before receiving the feature maps from the previous unit(s), is denoted by $T_l = f_l(U_{l-1})$, where $f_l(.)$ denotes the composition of above operations. We refer to T_l as the feature maps of the unit l. Mathematically, each unit is represented by the equation

$$U_l = \Psi(T_l, U_{<l}, \alpha_l), \quad l = 1, 2, \ldots, L,$$

where $\alpha_l \in \{0, 1\}$, $U_{\{<l\}} = U_{\{0,1,\ldots,l-1\}}$ denotes the feature maps from first unit up to $l-1^{th}$ unit, and Ψ denotes an operation applied to T_l and U_{l-1}. Thus, the ResNet Ψ_{res} and DenseNet Ψ_{den} architectures are respectively defined by:

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

$$U_l = \Psi_{den}(T_l, U_{<l}, \alpha_l) = Concat(\alpha_l T_l,\ U_{<l}), \tag{2}$$

where $Concat$ denotes the concatenation operation, and $\mathcal{A}_{l-1}$ is an identity, a convolution, down-sampling, or combination of these operators. Moreover, U_0, the input for the first unit is typically provided by a convolution operator (e.g., in ResNet, and DenseNet). The pseudo-code of our approach (referred to as the *Multi-Stage Pruning with Information Clustering* (MSPIC)) is given in Algorithm 1. In each stage t of MSPIC, the algorithm keeps more important units as active units for which $\alpha_l = 1$, and removes the less important ones, i.e., $\alpha_l = 0$. To determine the units whose feature maps T_l are more informative about the output of model given the input, MSPIC first computes $I(T_l, Y)$ and then selects active units by clustering $I(T_l, Y)$ values. Units inside each cluster have the same information about the output of the model. Hence, the algorithm keeps only one unit (centroid) in each cluster, and removes the other units from the model (graph of the network)[1]. (Please see the appendix for the formal presentation of the clustering scheme). The cluster centroid is selected as the unit which is closer to the input layer; in fact, our extensive experiments indicate that selecting units closer to the input layer as centroids tends to produce better accuracy when compared with other schemes such as random selection. Thus, by the end of stage t, the compression ratios is given by the total weights of removed units. The algorithm proceeds with re-train the new sub-graph with weights initialized with their values from the previous stage, $t-1$, and continue this multi-stage coarse pruning and re-training process until the pre-determined size of the compressed network is met. It is worthwhile to mention that if there were no "skip-connections", it would make sense to select only largest $I(T_l, Y)$ values. However, the existence of skip-connections violate the Markov chain requirements, and hence the information processing inequality does not hold. As a result, the information $I(T_l, Y)$ do not decrease monotonically. This motivates the proposed approach in our algorithm. As a result, the clustering approach may select larger or smaller values.

As a sample illustration, we now show the values of $I(T_l; Y)$ for all the active units across different stages. Figure 1 illustrates the histogram of the mutual information for the DenseNet100-k12 model. The top row in this figure shows the distribution of mutual information values for the CIFAR-10 data set, where Plot (a) denotes the frequency of the MI values for the full model and without any pruning. Plot (b) shows these values after 7 stages pruning. Plot (c) gives the distribution of mutual information of the active units after 13 stages. The bottom row illustrates the frequency of mutual information values of the active units for the CIFAR-100 data set. It can be observed from Figure 1 that the distribution of MI values are more widely spread in the later stages of the algorithm. This observation implies a rule of thumb for selecting the number of clusters in each stage of the algorithm. We may start with more aggressive pruning in the earlier stages (i.e., using the larger number of clusters), and gradually switch to finer pruning by selecting a smaller number of clusters in latter stages. Thus the larger number of pruning stages, the more important individual units are, and in the latter stages

[1]For clustering, we can use any clustering approach. Here, we cluster units for different bin resolutions (The set of all resolutions is called resolution vector).

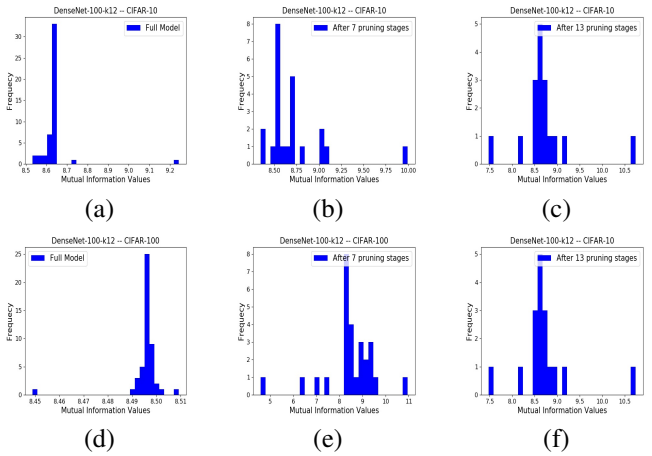

Fig. 1. Histogram for the Mutual Information of units in DenseNet100-k12 model. **Top Row-** Mutual Information frequency for CIFAR-10 data set. (a) for stage $N = 0$ (full model), (b) After $N = 7$ pruning stages, and (c) After $N = 13$ pruning stages. **Bottom Row-** Mutual Information frequency for CIFAR-100 data set. (d) for stage $N = 0$ (full model), (e) After $N = 7$ pruning stages, and (f) After $N = 13$ pruning stages.

of pruning, the active units (units that have not been pruned) have more information about the output of the model.

IV. Experimental Results

In this section, we provide extensive experimental results demonstrating the performance of our algorithm (MSPIC) on CIFAR-10, CIFAR-100, and Tiny-ImagNet data sets. Both CIFAR-10 and CIFAR-100 consist of 50000 and 10000 training and testing images with size $32 \times 32 \times 3$. The Tiny ImageNet data set is a subset of the full ImageNet data set. Each class includes 500 and 50 images with size $64 \times 64 \times 3$ for training and validation, respectively. All these data sets are used for the classification task respectively with 10, 100, and 200 classes. Comparisons of our approach with those of the state-of-the-art methods in deep compression are given respectively in Table I for CIFAR-10, Table II for CIFAR-100, and Table III for Tiny ImageNet. The column labeled 'R' in all the tables represents the percentage of reduction in the number of parameters. Also, the values are shown in columns labeled as "Par" (Number of Parameters) and "FLOPs" are in million and rounded by two-decimal digits. Since MSPIC involves re-training the pruned model in each stage, spending more time on fine-tuning the hyper-parameters for re-training may improve the reported test results. In some of the experiments, we have also used the Cutout augmentation technique [6] for the training of the deep models in each stage, including the training of the original full model. The key point here is 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 [22], [40] also use data augmentations techniques such as zero-padding, random cropping, shifting and mirroring only to training data set. Also, different papers use different normalizations for the training set. For instance, CondenseNet and the one given in [34] use different normalization values. Our experiments show that

Algorithm 1 Multi-Stage Pruning with Information Clustering (MSPIC)

INPUT:
$\mathbf{DNN}^0$: Pre-trained Deep Neural Network
S^0: The index set of units
T_l: Feature maps, $l = 1, 2, \ldots, |S^0|$
N: Number of stages
R^t: Resolution vector, $t = 0, 1, \ldots, N-1$
ϵ^t: Error threshold, $t = 0, 1, \ldots, N-1$
for $t = 0, 1, \ldots, N-1$ **do**
  Compute $I(T_l^t, Y)$, $l = 1, \ldots, |S^t|$ using $\mathbf{DNN}^t$
  Construct $\mathbf{I^t} = [I(T_1^t; Y), I(T_2^t; Y), \ldots, I(T_{|S|}^t; Y)]$
  $\{Cluster_1^{policy_1}, \ldots, Cluster_{M_{policy_1}^t}^{policy_1},$
  $\ldots, \quad Cluster_{M_{policy_{|R^t|}}^t}^{policy_{|R^t|}} \} = \text{Cluster}(R^t, \mathbf{I^t}, S^t)$
  for policy **in** R^t **do**
    for $j = 1, 2, \ldots, M_{policy}^t$ **do**
      $a_l = 1, \quad l = \min_{k \in Cluster_j^{policy}} k$
      $a_u = 0, \quad \forall u \in Cluster_j^{policy} \setminus l$
    end for
    Compute $Test_{error}^t$ for given policy
  end for
  Select one policy with $Test_{error}^t < \epsilon^t$
  $S^{t+1} \leftarrow S^t \setminus \{v : a_v = 0, v \in S\}$
  $\mathbf{DNN}^{t+1} \leftarrow$ Re-train the resulted sub-network with the trained weights in stage t
end for
Return pruned model with $|S^{N-1}|$ units

for CIFAR-10/100 data sets[2], we obtain better compressed model using DenseNet architecture which outperforms all the other method and is competitive with CondenseNet model. In addition, while the test accuracy of the compressed ResNet-based models are typically less than the other methods, but their FLOPs and number of parameters are significantly less than the others. This implies that if we run our proposed algorithm for smaller number of stages, the accuracy of the resulted models are as good as the other ones of course with larger FLOPs count and number of parameters. Finally, we have to mention that as we expect (please see section II), the structured pruning methods such as SNIP [34], GraSP [61], and DSR [45] achieve higher accuracy and compression ratio (e.g., 90%) in terms of number of parameters; however, they do not report any gain in FLOPs count.

We next summarize the setup of each experiment (Due to the lack of space, we defer the details of intermediate stages of compressed models to the appendix).

[2]For the purpose of (re)training pruned models in sections IV-A and IV-B, we use SGD optimization algorithm with momentum equals to 0.9. For the values of hyperparameters, we set epoch number to 120 for all stages, weight decay to 1e-4, and learning rate to 0.1 which is reduced by a factor of 0.1 every 25 epochs. The Cutout parameter is set to 16.

A. Compressing ResNet-based Models for CIFAR10/100

In this section, we have tested MSPIC on three different ResNet models including ResNet56, and ResNet164.

1) ResNet56: The ResNet56 model consists of 56 layers with 27 units arranged in 3 blocks, with one convolutional layer in the input of network, and a fully connected one in the last layer of the model (after the 27^{th} unit). Each unit has two convolutional layers with batch normalization. We first train this model (without any pruning) without cutout augmentation technique on the CIFAR-10 data set. This achieves 93.54% test accuracy. Full ResNet56 model has 853018 training parameters, and it consists of 126538378 FLOPs including *ReLU* operation. The results of applying MSPIC on this model with $N = 3$ stages is shown in Table II tagged by ResNet56.

2) ResNet164: We also compress ResNet164 under different conditions for both CIFAR-10 (Table I) and CIFAR-100 (Table II). This model has 164 layers grouped in 3 blocks each with 18 units. Similar to previous cases, we first train this model on CIFAR-10 data set with 0.9503 test accuracy, 1703258 as the number of parameters, and 254941706 FLOPs. By running MSPIC for $N = 2$ stages, we can achieve 0.9197 test accuracy with 715482 number of parameters and 153758218 FLOPs. Next, we train ResNet164 with cutout (i.e., ResNet164-c) pre-processing and compress the resulted model with accuracy 0.9569 to a model with test accuracy 0.9207 (that is obtained by applying 2 stages of MSPIC). We show this compressed model by ResNet164-ca in Table I. We also repeat the same experience with higher number of stages of MSPIC (N=7). This corresponds to a finer way of pruning (e.g., smaller number of units are dropped in each stage). We denote this model by ResNet164-cb in Table I. As we can see, ResNet164-cb can achieve almost the same test accuracy as ResNet164-ca; albeit, with significantly more pruned number of parameters and number of FLOPs. Finally, in Table II, we present the compressed model of ResNet164-c on CIFAR-100 data set with original; top-1 test error of 25.41% by running MSPIC for $N = 7$ as the number of stages. This gives a model compression rate of 45.30% in terms of number of parameters.

B. Compressing DenseNet-based Models for CIFAR10/100

In this section, we present compression results for the DenseNet model. We focus on the DenseNet100-k12 with 100 layers and growth-rate, $k = 12$ (i.e., the number of channels in T_l). This model has 3 blocks within each there are 16 units (48 units in total). There are two-transition layers between the blocks, as well as a convolutional layer and a fully connected layer at the beginning (before the first unit) and end of the network (after the last unit). Each dense unit has a *Bottleneck* architecture, consisting of *BN-ReLU-Conv* operations. In addition, each transition-layer is constructed by a batch normalization operation followed by one convolution layer.

[3]The accuracy reported in [15], [36] and [34] is given by 0.9359 ± 0.58 and 0.9259 ± 0.10, respectively.

[4]The Flops number reported in [65] is 508 million, while in [62] and [35] is give by 253 million.

TABLE I

COMPARISON OF CLASSIFICATION TEST ACCURACY (ACC) ON CIFAR-10, NUMBER OF PARAMETERS (PAR) IN MILLION, AND FLOPS BETWEEN MSPIC AND THOSE OF THE STATE-OF-THE-ART METHODS. "–" MEANS NO REPORTED VALUE. COLUMN R(%) DENOTES THE REDUCTION IN THE NUMBER OF PARAMETERS IN PERCENTAGE. THE BEST NUMERICAL RESULTS HAVE BEEN WRITTEN IN BOLD TEXT.

Model	Acc	Par(M)	FLOPs(M)	R(%)
VGG16 (PFEC-A) [35]	0.9340	5.40	206.00	-
VGG19 [40]	0.9380	2.30	195.00	-
VGG19 (SNIP) [34]	0.9363	2.00	-	90.00
ResNet32 (SNIP) [34][3]	0.9259	0.19	-	90.00
ResNet56 (PFEC-B) [35]	0.9306	0.73	90.90	13.70
ResNet56 (PP-1) [54]	0.9309	-	39.99	-
ResNet56 (NISP) [69]	0.9301	-	67.57	-
ResNet56 (AFP-G) [7]	0.9294	-	55.50	-
ResNet56 (SFP) [15][3]	0.9359	-	59.40	-
ResNet110 (BlkDrop) [65][4]	0.9360	-	173.00	-
ResNet110 (SkipNet) [62]	0.9330	-	126.00	-
ResNet110 (PFEC-B) [35]	0.9330	1.16	155.00	32.40
ResNet164B [40]	0.9473	1.21	124.00	-
Wide ResNet16-8 [34]	0.9337	0.55	-	95.00
DenseNet40 [40]	0.9481	0.66	190.00	-
CondenseNet$^{\text{light}}$-94 [22]	0.9496	0.33	122.00	-
CondenseNet86 [22]	0.9496	0.52	65.00	-
ResNet56 **(ours)**	0.9128	0.23	42.30	72.72
ResNet164 **(ours)**	0.9190	0.71	153.70	57.99
ResNet164-ca **(ours)**	0.9207	0.99	143.90	41.53
ResNet164-cb **(ours)**	0.9173	0.47	112.50	72.02
DenseNet100-k12 **(ours)**	0.9264	0.32	213.20	57.69
DenseNet100-k12-ca **(ours)**	0.9352	0.34	224.90	56.27
DenseNet100-k12-cb **(ours)**	**0.9437**	**0.29**	**173.20**	**62.66**

TABLE II

COMPARISON OF CLASSIFICATION **TOP-1** TEST ACCURACY (ACC) ON CIFAR-100, NUMBER OF PARAMETERS (PAR) IN MILLION, AND FLOPS BETWEEN MSPIC AND THOSE OF THE STATE-OF-THE-ART METHODS. "–" MEANS NO REPORTED VALUE. "$\sim$" MEANS APPROXIMATE VALUE. THE ACCURACY REPORTED IN [61] FOR VGG-19 AND RESNET32 IS GIVEN BY 0.7195 ± 0.18 AND 0.6924 ± 0.24, RESPECTIVELY. R(%) DENOTES THE REDUCTION IN THE NUMBER OF PARAMETERS IN PERCENTAGE. THE BEST NUMERICAL RESULTS HAVE BEEN WRITTEN IN BOLD TEXT.

Model	Acc	Par(M)	FLOPs(M)	R(%)
VGG19 (GraSP) [61]	0.7195	2.00	-	90.00
ResNet32 (SET) [43]	0.6966	0.19	-	90.00
ResNet32 (GraSP) [61]	0.6924	0.19	-	90.00
ResNet32 (DSR) [45]	0.6963	0.19	-	90.00
Resnet110 (BlkDrop) [65]	0.7370	-	$\sim$284.00	-
ResNet110 (SkipNet) [62]	0.7250	-	-	-
ResNet164B [40]	0.7609	1.21	124.00	-
DenseNet40 [40]	0.7472	0.66	190.00	-
CondenseNet$^{\text{light}}$-94 [22]	0.7592	0.33	122.00	-
CondenseNet86 [22]	0.7636	0.52	65.00	-
ResNet164-c **(ours)**	**0.7459**	0.94	130.97	45.30
DenseNet100-k12-c **(ours)**	0.7408	**0.38**	**203.35**	**52.37**

Since the input of the transition-layer in DenseNet models expects to receive $k \times 16$ channels to apply convolution operation, one needs to make sure that k channels added to the immediate previous active unit when a dense unit is removed from the architecture. To address this, we simply pad the output of the immediate previous active unit with k zero channels (i.e., channels with zero values). We note that this zero-padding does not increase the number of parameters in the pruned model, and also with implementing sparse con-

TABLE III
COMPARISON OF CLASSIFICATION TEST ACCURACY (ACC) ON TINY IMAGENET, NUMBER OF PARAMETERS (PAR) IN MILLION, AND FLOPS BETWEEN MSPIC AND THOSE OF THE STATE-OF-THE-ART METHODS. "–" MEANS NO REPORTED VALUE. ↓ INDICATES THE DROPPING IN PERCENTAGE OF THE TEST ACCURACY WITH RESPECT TO THE BASE (FULL) MODEL.

Model (ResNet32)	Acc	Par(M)	FLOPs	R(%)
(Deep-R) [1]	0.5329 (↓ 9.60%)	0.28	-	85.00
(SNIP) [34]	0.5633 (↓ 6.56%)	0.28	-	85.00
ours	0.5432 (↓ 7.24%)	0.35	2.46	81.34

volution, we can ignore the number of addition/multiplication with these zero channels. Now, we train a DenseNet100-k12 with and without Cutout pre-processing on CIFAR-10 data set and only with Cutout for CIFAR-100. The trained full model with 769162 number of parameters, and 293546820 number of FLOPS achieves respectively test results of 0.9531 and 0.9590 for CIFAR-10 with and without Cutout. Also, its test accuracy for CIFAR-100 based on top-1 error is given by 0.7793 (with 800032 number of parameters, and 304104360 number of FLOPS). Table I demonstrates that by applying MSPIC for $N = 13$ stages on DenseNet100-k12 for classification of CIFAR-10 images, we can achieve compressed model of DenseNet100-k12-cb which has a competitive compressed performance with the state-of-the-art method of CondenseNet$^{\text{light}}$ [22]. Also, DenseNet100-k12-ca denotes the compressed model by running MSPIC for $N = 3$ stages (i.e., coarser pruning in each stage). Finally, Table II gives the results of compression on CIFAR-100 with Cutout. This result shows that MSPIC can achieve 52.37% reduction in the number of parameters.

C. Compressing ResNet32 for Tiny ImageNet

In this section, we present the result of applying our pruning technique on ResNet32 and for Tiny ImageNet data set. In general, the Tiny ImageNet classification task is considered more difficult than classifying CIFAR-10 data set. In ResNet32 model, there are 32 layers and 3 blocks each with 5 residual units. This model can achieve 61.60% top-1 test accuracy with 1885032 parameters and 1078912200 FLOPs. Our trained full ResNet32 achieves 61.56% top-1 test accuracy, while the top-1 test test accuracy of full ResNet32 for the other methods reported in Table III is 62.89% [61] (we could not reproduce the test accuracy of [61]). 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. We have used Cutout for retraining the pruned models with parameter 16 for all the stages. Table III shows the result of applying MSPIC for 9 pruning stages for classifying Tiny ImageNet data set. As illustrated in Table III, MSPIC has better performance than Deep_R, and competitive to the structured method such as SNIP without any reported any gain in FLOPs count.

V. CONCLUSIONS

In this paper, we proposed a new method for the deep neural model compression. In particular, we considered models with skip-connection such as ResNet and DenseNet in which the feature maps of the previous layer(s) are used for constructing those of the next layers. Our approach is based on coarse pruning of the units (collection of some layers) based on the information they provide about the output of the network. Motivated by Information Bottleneck theory, we use the mutual information as a measure to determine the units that are pruned. Extensive simulation results demonstrate that the proposed approach can achieve competitive performance with those of the state-of-the-art methods in deep model compression. Future research can include the investigation of more theoretical aspects of the proposed method.

REFERENCES

[1] G. B., D. K., W. M., and R. L., "Deep rewiring: Training very sparse deep networks," in *Int. Conf. Learning. Rep.*, 2018.

[2] 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.

[3] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Adv. Neural Inf. Process. Sys.*, 2015, pp. 3123–3131.

[4] 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.*, 2009, pp. 248–255.

[5] 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.*, 2014, pp. 1269–1277.

[6] T. DeVries and G. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[7] X. Ding, G. Ding, J. Han, and S. Tang, "Auto-balanced filter pruning for efficient convolutional neural networks," in *AAAI Conf. Artificial Intelligence*, 2018.

[8] 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.*, 2017, pp. 1039–1048.

[9] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Int. Conf. Learning. Rep.*, 2019.

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

[11] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Adv. Neural Inf. Process. Sys.*, 2016, pp. 1379–1387.

[12] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2016.

[13] Y. Han, J. Jiao, T. Weissman, and Y. Wu, "Optimal rates of entropy estimation over Lipschitz balls," *arXiv preprint arXiv:1711.02141*, 2017.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. Comp. Vision. Patt. Recog.*, 2016, pp. 770–778.

[15] 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*, 2018, pp. 2234–2240.

[16] 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.*, 2019, pp. 4340–4349.

[17] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *IEEE Int. Conf. Comp. Vision*, 2017, pp. 1389–1397.

[18] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[19] L. Hou, Q. Yao, and J. Kwok, "Loss-aware binarization of deep networks," *Int. Conf. Learning. Rep.*, 2017.

[20] 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.

[21] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *IEEE Conf. Comp. Vision. Patt. Recog.*, 2018, pp. 7132–7141.

[22] 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.*, 2018, pp. 2752–2761.

[23] G. Huang, Z. Liu, L. Van Der Maaten, and K. Weinberger, "Densely connected convolutional networks," in *IEEE Conf. Comp. Vision. Patt. Recog.*, 2017, pp. 4700–4708.

[24] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Europ. Conf. Comp. Vision*, 2018, pp. 304–320.

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

[26] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *British Machine Vision Confs.*, 2014.

[27] Y. Jia, D. L., and E. Shelhamer, "Caffe model zoo," 2016. [Online]. Available: http://caffe.berkeleyvision.org/model_zoo

[28] A. Kolchinsky and B. Tracey, "Estimating mixture entropy with pairwise distances," *Entropy*, vol. 19, no. 7, p. 361, 2017.

[29] A. Kolchinsky, B. Tracey, and D. Wolpert, "Nonlinear information bottleneck," *arXiv preprint arXiv:1705.02436*, 2017.

[30] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical review E*, vol. 69, no. 6, p. 066138, 2004.

[31] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[32] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Adv. Neural Inf. Process. Sys.*, 2012, pp. 1097–1105.

[33] Z. L., M. S., T. Z., G. H., and T. D., "Rethinking the value of network pruning," in *Int. Conf. Learning. Rep.*, 2019.

[34] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single shot network pruning based on connection sensitivity," in *Int. Conf. Learning. Rep.*, 2019.

[35] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf, "Pruning filters for efficient convnets," *Int. Conf. Learning. Rep.*, 2017.

[36] 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.*, 2019, pp. 3977–3986.

[37] 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*, 2014, pp. 740–755.

[38] R. Linsker, "Self-organization in a perceptual network. computer," pp. 105–117, 1988.

[39] 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.

[40] 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*, 2017, pp. 2736–2744.

[41] 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.*, vol. 41, no. 10, pp. 2525–2538, 2018.

[42] 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.

[43] 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*, vol. 9, no. 1, pp. 1–12, 2018.

[44] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *Int. Conf. Learning. Rep..*, 2017.

[45] H. Mostafa and X. Wang, "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization," in *Int. Conf. Machine. Learning.*, 2019, pp. 4646–4655.

[46] K. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[47] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Europ. Conf. Comp. Vision.* Springer, 2016, pp. 525–542.

[48] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, "Sbnet: Sparse blocks network for fast inference," in *IEEE Conf. Comp. Vision. Patt. Recog.*, 2018, pp. 8711–8720.

[49] 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.

[50] A. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. Tracey, and D. D., "On the information bottleneck theory of deep learning," in *Int. Conf. Learning. Rep.*, 2018.

[51] O. Shamir, S. Sabato, and N. Tishby, "Learning and generalization with the information bottleneck," *Theoretical Comp. Science*, vol. 411, no. 29-30, pp. 2696–2711, 2010.

[52] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," *arXiv preprint arXiv:1703.00810*, 2017.

[53] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[54] 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*, 2019, pp. 3460–3466.

[55] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhuinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal. Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[56] 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.*, 2015, pp. 1–9.

[57] N. Tishby, F. Pereira, and W. Bialek, "The information bottleneck method," *arXiv preprint physics/0004057*, 2000.

[58] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *IEEE Inf. Theory. Workshop. (ITW)*, 2015, pp. 1–5.

[59] A. Veit and S. Belongie, "Convolutional networks with adaptive computation graphs," *arXiv preprint arXiv:1711.11503*, 2017.

[60] A. Veit, M. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *Adv. Neural Inf. Process. Sys.*, 2016, pp. 550–558.

[61] C. W., G. Z., and R. G., "Picking winning tickets before training by preserving gradient flow," in *Int. Conf. Learning. Rep.*, 2020.

[62] X. Wang, F. Yu, Z. Dou, T. Darrell, and J. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *Europ. Conf. Comp. Vision*, 2018, pp. 409–424.

[63] Y. Wang, C. Xu, C. Xu, and D. Tao, "Packing convolutional neural networks in the frequency domain," *IEEE Trans. Patt. Analysis. Machine Intelligence*, 2018.

[64] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Adv. Neural Inf. Process. Sys.*, 2016, pp. 2074–2082.

[65] 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.*, 2018, pp. 8817–8826.

[66] X. Xiao, Z. Wang, and S. Rajasekaran, "Autoprune: Automatic network pruning by regularizing auxiliary parameters," in *Adv. Neural Inf. Process. Sys.*, 2019, pp. 13 681–13 691.

[67] 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.*, 2018, pp. 8847–8856.

[68] 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.

[69] R. Yu, A. Li, C. Chen, J. Lai, V. Morariu, X. Han, M. Gao, C. Lin, and L. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *IEEE Conf. Comp. Vision. Patt. Recog.*, 2018, pp. 9194–9203.

[70] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *Int. Conf. Learning. Rep.*, 2017.

[71] 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.*, 2018, pp. 6848–6856.

VI. Appendix

A. Clustering Algorithm

Here, we present the pseudocode for the clustering algorithm given in Algorithm 2. The clustering procedure starts with computing the maximum and minimum I ($\mathcal{L} = \min_l I(T_l;Y)$, and $\mathcal{U} = \max_l I(T_l;Y)$). Then, a grid is constructed for the interval ranging between $\mathcal{U}$ and $\mathcal{L}$ according to each *policy* in the resolution vector (we refer to entries of the resolution vector as policies since clustering according to their values leads to various compressed architectures). Next, units within each grid are grouped as a cluster. The cluster centroid is then selected as the unit which is closer to the input layer; in fact, our extensive experiments indicate that selecting units closer to the input layer as centroids tends to produce better accuracy when compared with a random selection of centroids.

Algorithm 2 Cluster(R, **I**, S)

INPUT:
R: Resolution vector
$\mathbf{I} = [I(T_1;Y), I(T_2;Y), \ldots, I(T_{|S|};Y)]$
S: The index set of current active units
$\mathcal{U} = \max_{l\in S} I(T_l;Y)$, $\quad b = \text{argmax}_{l\in S} I(T_l;Y)$
$\mathcal{L} = \min_{l\in S} I(T_l;Y)$
for policy in R **do**
  $\delta = policy \times (\mathcal{U} - \mathcal{L})$
  for $j = 1, 2, \ldots, M_{policy} = \frac{1}{policy}$ **do**
    $Cluster_j^{policy} = \{Unit_q : \mathcal{L} + (j-1)\delta \le I(T_q;Y) < \mathcal{L} + j\delta,\ q \in S\}$
  end for
  if $\nexists j \in [M_{policy}]$ $s.t.$ $Unit_b \in Cluster_j^{policy}$ **then**
    $Cluster_{M+1}^{policy} = Unit_b$
  end if
end for
Return $\cup_{policy} \cup_j Cluster_j^{policy}$

B. Upper Bound on Mutual Information

In this section, we will give the details of our upper bound on the mutual information. This calculation is based on the upper and lower bounds on the differential entropy in [28], where the authors gave an estimate of the differential entropy. To this end, recall our notation of Section III-D. Then,

$$\hat{H}_D(X) = H(X|C) - \sum_i c_i \ln \sum_j c_j \exp(-D(P_i||P_j)),$$

where P_i and P_j denote the i^{th} and j^{th} densities in the mixture model and D is a distance or divergence function. Now, by instantiating the above estimator for Gaussian mixture model (GMM) with N components, and selecting D as KL-divergence, we can derive an upper bound for the entropy of T using the KL-divergence formula between two Gaussian distributions, and the entropy of Gaussian probability distribution:

$$H(T) \le \hat{H}_{KL}(T) = \frac{d}{2} - \sum_{i=1}^{N} c_i \ln \sum_{j=1}^{N} c_j P_j(\mu_i),$$

where $P_j(.)$ denotes the j^{th} Gaussian component with mean μ_j and variance σ^2, d is the dimension of the random vector X, and c_i's denote the coefficients of the GMM. The above upper bound is the same as the non-parametric KDE estimation for entropy with an extra dimension correct term, i.e., $H(T) \le \frac{d}{2} - \sum_{i=1}^{N} c_i \ln \sum_{j=1}^{N} c_j p_j(\mu_i)$ [29]. Moreover, [28] shows that by instantiating the entropy estimator with the Chernoff α-divergence[5] with $\alpha = 0.5$ and with the same GMM, a lower bound on the entropy can be derived given by:

$$H(T) \ge \hat{H}_{c_{0.5}}(T) = \frac{d}{2} + \frac{d}{2}\ln\frac{1}{4} - \sum_{i=1}^{N} c_i \ln \sum_{j=1}^{N} c_j \tilde{P}_{j,0.5}(\mu_i),$$

where $\tilde{P}_{j,0.5} \sim \mathcal{N}(\mu_j, 4\sigma^2\mathbf{I}_d)$. Putting these two bounds together, we can find an upper bound for the mutual information as follows:

$$\begin{aligned}
I(T;Y) &= H(T) - H(T|Y) \\
&\le -\sum_{i=1}^{N} c_i \ln \sum_{j=1}^{N} c_j P_j(\mu_i) \\
&\quad - \sum_{k=0}^{l-1} P_k\Big(-\sum_{\substack{i=1\\ y_i=k}}^{N} c_i \ln \sum_{\substack{j=1\\ y_j=k}}^{N} c_j P_j(\mu_i)\Big) \\
&= -\frac{1}{n}\sum_{i=1}^{n} \ln \frac{1}{n}\sum_{j=1}^{n} \exp\Big(-\frac{||\mu_j - \mu_i||_2^2}{2\sigma^2}\Big) \\
&\quad - \sum_{k=0}^{l-1} P_k\Big(-\frac{1}{n_k}\sum_{\substack{i=1\\ y_i=k}}^{n} \ln \frac{1}{n_k}\sum_{\substack{j=1\\ y_j=k}}^{n_k} \exp\Big(-\frac{1}{4}\frac{||\mu_j - \mu_i||_2^2}{2\sigma^2}\Big)\Big).
\end{aligned}$$

In the above inequalities, we have used the fact that the conditional entropy is give by $H(T|Y) = \sum_{k=0}^{l-1} P_k H(T|Y = y_k)$.

C. More Details on Experimental Results

In this section, we present the details of intermediate stages for some of the pruned model by MSPIC. In all the following results, the column of Reduction Percentage has been rounded to 2 decimal digits. Table IV shows the intermediate results of pruning for ResNet56 model and CIFAR-10 data sets. Also, Table V shows the details of pruned DenseNet100-k12 in classification task on CIFAR-10 data set (without using Cutout) for $N = 6$ stages. Here, DenseNet100-k12-A denotes the pruned model in which MSPIC has selected 24 units to be removed, while DenseNet100-k12-B corresponds to the pruned model with 27 dense units. Table VI shows the pruned details in the intermediate stages of MSPIC applied on

[5] Chernoff α-divergence is defined as $C_\alpha(p_i||p_j) = -\ln \int p_i^\alpha(x) p_j^{1-\alpha}(x) dx$.

TABLE IV
PRUNING OF RESNET56 IN CLASSIFICATION OF CIFAR-10 DATA SET (NO CUTOUT) IN INTERMEDIATE STAGES.

Model (No Cutout)	Test Accuracy	Reduction	Parameters	Flops
ResNet56 (full)	93.34	0.0000	0.85	126.55
ResNet56 (t=1)	93.59	0.5044	0.42	83.84
ResNet56 (t=2)	92.47	0.6837	0.27	55.31
ResNet56 (t=3)	91.92	0.7218	0.24	46.99
ResNet56 (t=4)	92.17	0.7272	0.23	42.21
ResNet56 (t=5)	91.15	0.8140	0.16	37.47

TABLE V
PRUNING OF DENSENET100-K12 IN CLASSIFICATION OF CIFAR-10 DATA SET (NO CUTOUT) IN INTERMEDIATE STAGES. DENSENET100-K12-A DENOTES THE PRUNED MODEL IN WHICH MSPIC HAS SELECTED 24 UNITS TO BE REMOVED, WHILE DENSENET100-K12-B CORRESPONDS TO 27 REMOVED DENSE UNITS.

Model (No Cutout)	Test Accuracy	Reduction	Parameters	Flops
DenseNet100-k12 (full)	0.9531	0%	769162	293546820
DenseNet100-k12 (t=1)	0.9490	24.34%	581962	267113028
DenseNet100-k12 (t=2)	0.9458	34.40%	504562	258479940
DenseNet100-k12 (t=3)	0.9392	42.33%	443542	245534916
DenseNet100-k12 (t=4)	0.9356	47.32%	405202	238716228
DenseNet100-k12 (t=5)	0.9362	49.92%	385222	237542340
DenseNet100-k12-A (t=6)	0.9314	51.89%	370042	236669508
DenseNet100-k12-B (t=7)	0.9264	57.69%	325402	213187140

TABLE VI
PRUNING OF DENSENET100-K12 IN CLASSIFICATION OF CIFAR-10 DATA SET (USING CUTOUT) IN INTERMEDIATE STAGES. DENSENET100-K12-A DENOTES THE PRUNED MODEL IN WHICH MSPIC HAS SELECTED 20 UNITS TO BE REMOVED, WHILE DENSENET100-K12-B CORRESPONDS TO 26 REMOVED DENSE UNITS.

Model (Cutout-Coarse prune)	Test Accuracy	Reduction	Parameters	Flops
DenseNet100-k12 (full)	0.9573	0%	769162	293546820
DenseNet100-k12 (t=1)	0.9484	37.84%	478102	254783172
DenseNet100-k12-A (t=2)	0.9477	44.15%	429562	245172804
DenseNet100-k12-B (t=3)	0.9352	56.27%	336382	224876484

TABLE VII
PRUNING OF DENSENET100-K12 IN CLASSIFICATION OF CIFAR-10 DATA SET (USING CUTOUT) IN INTERMEDIATE STAGES. HERE, THE PRUNED UNITS ARE DONE IN A FINER WAY COMPARED TO TABLE VI.

Model (Cutout-Fine prune)	Test Accuracy	Reduction	Parameters	Flops
DenseNet100-k12 (full)	0.9590	0%	769162	293546820
DenseNet100-k12 (t=1)	0.955	24.52%	580582	274015428
DenseNet100-k12 (t=2)	0.9456	45.58%	418582	245120964
DenseNet100-k12 (t=3)	0.9479	47.67%	402502	241072068
DenseNet100-k12 (t=4)	0.9479	49.06%	391822	238046148
DenseNet100-k12 (t=5)	0.9457	50.68%	379342	235232196
DenseNet100-k12 (t=6)	0.9436	52.62%	364462	231484356
DenseNet100-k12 (t=7)	0.9452	54.47%	350182	215768004
DenseNet100-k12 (t=8)	0.9424	55.94%	338902	213255108
DenseNet100-k12 (t=9)	0.9425	57.25%	328822	201753540
DenseNet100-k12 (t=10)	0.9425	59.03%	315142	187966404
DenseNet100-k12 (t=11)	0.943	61.51%	296062	183496644
DenseNet100-k12 (t=12)	0.9437	62.66%	287182	173199300
DenseNet100-k12 (t=13)	0.9354	64.56%	272602	172364100

DenseNet100-k12 using the Cutout pre-processing technique. In Table VI, DenseNet100-k12-A denotes the pruned model in which MSPIC has selected 20 units to be removed, while DenseNet100-k12-B corresponds to the pruned model with 26 dense units. Next, we have Table VII which shows the intermediate pruned models in the classification of CIFAR-10 using Cutout but in a finer way compared to the stages presented in Table VI. Also, Table VIII presents the results for pruning the DenseNet100-k12 for classifying the CIFAR-100 data set using Cutout. Finally, Table IX shows the intermediate results for pruning the ResNet32 for classifying the Tiny ImageNet data set.

TABLE VIII
PRUNING OF DENSENET100-K12 IN CLASSIFICATION OF CIFAR-100 (WITH USING CUTOUT) DATA SET IN INTERMEDIATE STAGES.

Model (Cutout)	Test Accuracy	Reduction	Parameters	Flops
DenseNet100-k12 (full)	0.7793	0%	800032	304104360
DenseNet100-k12 (t=1)	0.7695	16.47%	668272	291888552
DenseNet100-k12 (t=2)	0.7646	25.88%	592972	280689192
DenseNet100-k12 (t=3)	0.7644	34.07%	527452	257898024
DenseNet100-k12 (t=4)	0.7552	36.64%	506872	256686504
DenseNet100-k12 (t=5)	0.7501	39.91%	480712	238997928
DenseNet100-k12(t=6)	0.7516	41.40%	468832	235670952
DenseNet100-k12 (t=7)	0.7479	43.22%	454252	234835752
DenseNet100-k12 (t=8)	0.7436	45.68%	434572	229754664
DenseNet100-k12 (t=9)	0.7447	47.09%	423292	226909992
DenseNet100-k12 (t=10)	0.7414	48.65%	410812	223764264
DenseNet100-k12 (t=11)	0.7406	50.43%	396532	220166952
DenseNet100-k12 (t=12)	0.7408	52.37%	381052	203246376
DenseNet100-k12 (t=13)	0.7316	54.64%	362872	202185384

TABLE IX
PRUNING OF RESNET32 IN CLASSIFICATION OF TINY IMAGENET DATA SET IN INTERMEDIATE STAGES.

Model (No Cutout)	Test Accuracy	Reduction	Parameters	Flops
ResNet32 (full)	0.6156	0.0000	1.89	1078.91
ResNet32 (t=1)	0.6248	15.6721	1.59	1003.32
ResNet32 (t=2)	0.6094	36.2536	1.20	776.14
ResNet32 (t=3)	0.5953	51.9257	0.91	700.54
ResNet32 (t=4)	0.5905	55.8505	0.83	624.85
ResNet32 (t=5)	0.5861	59.7753	0.76	549.15
ResNet32 (t=6)	0.5756	63.7001	0.68	473.46
ResNet32 (t=7)	0.5459	79.3722	0.39	397.86
ResNet32 (t=8)	0.5447	80.3568	0.37	321.97
ResNet32 (t=9)	0.5432	81.3414	0.35	246.08