Deep Neural Compression with Information of Feature Map

Anonymous CVPR submission

Paper ID 9857

Abstract

We propose a novel technique for compressing deep neural models which achieves a competitive performance to state-of-the-art methods. Our approach utilizes the mutual information between the feature maps and the output of the model to prune the redundant layers of the network. Extensive numerical experiments on both CIFAR-10 and CIFAR-100 data sets show that the proposed method can be an effective approach for compressing deep models both in terms of the numbers of parameters and operations. For instance, by applying our pruning approach on DenseNet model with 0.77 *million parameters and* 293 *million operations for classifying images from CIFAR-10 data set, we can achieve respectively* 62.66% *and* 41.00% *reduction in the number of parameters and the number of operations, while increasing the test error less than* 1%.

1. Introduction

Deep Neural Networks (DNNs) are known for their intensive computational load and memory usage, hindering their deployment on embedded systems with limited computational resources such as mobile devices and sensor networks. Many accurate award-winning deep convolutional neural networks (CNNs) in image competitions (e.g., *COCO* [1] and *ImageNet* [2]) have millions or even billions of parameters. For instance, AlexNet [3] and VGG-16 [4] Caffemodels occupy respectively over 200MB, and 500MB memory space [5]. Similarly, ResNet-50 needs over 95MB memory for storage and over 3.8×10^9 floating number multiplications for calculation each image [6]. Thus, the large memory storage and the large number of floating point operations per second (FLOPs) prevents deep neural models to be incorporated into hardware constraint devices. Another issue in resource-limited embedded systems is energy consumption. Running large neural networks requires large bandwidths to fetch the parameters and intensive algorithmic operations, which consume a significant amount of energy [7]. In recent years, the above issues have motivated active research for developing computationally efficient deep neural models.

1.1. Motivation

Recent empirical evidence has shown that there is a lot of redundancy in deep neural models. That is, if one removes many of the learned parameters of a trained network, including feature maps in CNNs, or even some layers, the performance of the trained network will remain around the same level [8, 9]. This observation is also consistent with the success of deep learning models in achieving a low generalization error for many real-world applications. In particular, while many deep models are extremely over-parameterized (i.e., the number of parameters is far more than the number of training samples), overfitting is typically not a matter of concern on test data [10]. Motivated by these observations and the need for deployment of the deep models in resource-limited systems, the following question naturally arises. Is it possible to reduce memory and computation usages in the training or testing of a neural network without sacrificing its original predictive performance? To that end, many approaches based on pruning, sparsifying, and quantizing of the network parameters have been proposed within the last few years. We will review the deep compression approaches in Section 2.

1.2. Our contribution

In general, the bulk of memory storage in DNNs is due to the dense layers, while the computation time bottleneck is mostly related to the convolutional layers. Our proposed approach for compressing deep networks is based on the successively coarse pruning and re-training of a deep neural model. 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 [11] that uses mutual information (MI) between hidden feature maps and output labels. Regarding the first concept, we focus on the deep architectures in which the feature maps in the previous layer(s) have a direct contribution in forming feature maps in the next layers (i.e., skip-connection). Prominent examples of these architectures include residual net-

works (ResNet) [12] and densely connected convolutional networks (DenseNet) [13]. Many successful deep models in image/video applications are based on these concepts. Our proposed approach has been inspired by an interesting observation about ResNet family models [14]. It was empirically observed that the trained ResNet architectures are robust against the removing of layers, while removing even a single layer in non-residual networks such as VGG can drop the performance significantly. Our approach investigates this phenomenon through the lens of deep compression. Instead of removing just the individual layers, we show that dropping a residual unit in ResNet, or a dense unit in DenseNet (where the 'unit' is defined later) does not hurt the performance significantly. We will also propose a novel approach to remove/keep several complete units in a trained deep model using the mutual information between learned features and the output of the model. Our contributions can be summarized as follows:

- Proposing an iterative coarse pruning compression technique for deep neural models with skip-connection with much less complexity and competitive performance when compared with state-of-the-art methods,
- Using mutual information for quantifying the amount of redundant information in the learned feature maps, and
- Illustrating the efficacy of our approach in reducing the network complexity both in memory usage and computational complexity through extensive experimental results for the classification tasks of CIFAR-10 and CIFAR-100 data sets [15].

The rest of this paper is organized as follows. In Section 2, we review some existing and related works in deep compression. Section 3 provides our proposed approach. In Section 4, we present extensive experimental results. In Section 5, we make our conclusions and suggest some future directions for future research.

2. Prior Work

There is a large body of work in deep compression in the last few years. Here, we only present the most relevant ones to our paper. A more detailed discussion could be found in a recent survey on deep compression [8]. The first line of research is concerned with pruning techniques [16, 17, 7, 18, 19, 20, 21, 22, 23, 24] which reduce the network complexity and addressing the overfitting problem through quantization and binarization. These methods have been motivated by an empirical observation that DNNs are surprisingly robust to various types of weight distortion such as quantization, additive corruption, multiplicative noise, projection, etc. [22]. In addition, these approaches can be seen as a continuation of some commonly used techniques in training of deep learning such as the *drop-out* [25], *ReLU* activation function, etc. A drawback regarding the quantization based methods is that they may result in low accuracy in large networks such as *GoogleNet* [26]. Moreover, pruning based on ℓ_1-regularization usually requires require more iterations to converge. The second method is based on imposing some structures on the weights of a neural network, e.g., approximating the weight tensors of each layer based on low-rank approximation, or sparsifying the weights in order to achieve a compressed version of the original model [27, 28, 29, 30, 31]. These methods may be computationally expensive as the low-rank decomposition of large weight tensors is usually cumbersome. Another method is knowledge distillation (KD), which employs a student-teacher paradigm, where the knowledge of a larger network (teacher) is shifted to a smaller model (student) through learning the distribution of output classes produced by *sofmax* function [32, 33]. A disadvantage of KD methods is that they can only be applied for classification tasks using the cross-entropy loss function. Other methods are based on dynamic path selection which selects a sub-graph of the original network and use that as the proxy for the inference tasks [34, 35, 36, 37]. Many of the works in this category have focused on residual networks, and their goal is to dynamically deactivate/activate the residual units using learning-based approaches such as policy update in reinforcement learning [37, 36], a potentially time-consuming process. The proposed method in this paper is conceptually related to this category but uses a very different approach. Finally, more recently some other efficient network architectures have been proposed [12, 38, 13, 9, 39]. For instance, methods of [13] and [39] achieve comparable performance as VGG for the task of classification on *ImageNet* while reducing the amount of computation by a factor $10\times$ and $25\times$, respectively. Also, the work of Huang et al. [9] can achieve better accuracy with even higher reductions in the number of parameters and FLOPs, and has been be considered as the state-of-the-art compressed architecture for classification on CIFAR-10, CIFAR-100, and *ImageNet* data sets.

3. Background and Proposed Approach

In this section, we provide some relevant background and then our proposed method. Throughout this paper, we represent random variables and their realizations with respectively uppercase and lowercase letters. Also, $P(.)$, $\mathbb{E}(.)$, and $\mathbb{1}(.)$ respectively denote probability expectation operator and indicator function. Let $I(.)$ and $H(.)$ respectively denote mutual information and entropy, and $[M] = \{1, .2, \ldots, M\}$.

3.1. Residual Networks (ResNet)

The idea of designing by-pass paths or skip-connection in the layers of neural networks has recently emerged due to the success of ResNet [12]. This has achieved record-breaking performance in ImageNet and *COCO* challenges for various computer vision tasks such as image classification, localization, and object detection.

Let $\mathcal{F}$ be a transform that maps the output of a layer (such as $(l-1)^{th}$) layer to the next (e.g. l^{th}) layer. Then the output of layer l, x_l is given by

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

Thus the previous ResNet feature maps are explicitly used to form the current features. It has been observed that this eases the training of a very deep network by alleviating the gradient vanishing problem. Typically, the ResNet consists of a number blocks consisting of multiple residual units. Each residual unit is made by stacking two or three convolutional layers together with *Batch Normalization* (BN) [40] layers, and a skip-connection from the output of the previous residual unit to the output of the current unit. This skip-connection either goes through another convolution operation, or directly connects to the current unit (identity map).

3.2. Densely Connected Networks (DenseNet)

A similar idea to skip-connection is further investigated in DenseNet [13]. 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, they 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 [13] that the number of parameters used in this architecture could be significantly less than the ResNet families.

3.3. Information Bottleneck Framework

In this section, we review the Information Bottleneck (IB) which is a theoretical framework proposed by [41, 11] 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, the authors of[42] showed that mutual information between the layers and the input and output of any DNN is a strong quantitative measure. Let us consider a supervised learning scenario, and denote the joint distribution of input and output of a DNN by $P(x, y)$. 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 that removes the irrelevant ones by compressing X. The IB framework looks for the most concise representation of input X through another random variable T (hidden feature map), which is simultaneously most relevant to the output Y of the network (a trade-off between minimal sufficiency of X and maximal information on Y) [11]. To better understand the relation between $I(T;Y)$ and the risk minimization in DNN for a supervised task, we describe a multi-classification scenario. Consider 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 (e.g. a hidden feature map in the network). In this context, IB looks for a feature map T as a compressed representation of X, which 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 and $p(t) = \sum_{i=1}^n p(t|x_i)P(x_i)$. On the other hand, Chernoff-Stein's Lemma [43] provides an asymptotic result for the misclassification probability. Specifically, denote by y' the label of a test input x', then according to this lemma as $n \to \infty$, the misclassification probability (i.e., probability of classifying x' in class y for some $y \neq y'$) is upper bounded by:

$$P(y \neq y') \leq \exp\big(-nD_{KL}(p(t|Y = y)||p(t|Y = y'))\big).$$

For the multi-classes case, using union bound, we have:

$$\begin{aligned}
P(y \neq y') &\leq \sum_{y,y'\in\{0,\ldots,l-1\}} \exp\big(-nD_{KL}(p(t|Y = y)||p(t|Y = y'))\big) \\
&\leq l^2 \exp\big(-\frac{n}{l^2}\mathbb{E}_{Y,Y'} D_{KL}(p(t|Y = y)||p(t|Y = y'))\big) \\
&\leq l^2 \exp\big(-\frac{n}{l^2}\mathbb{E}_{Y} D_{KL}(p(t|y)||\mathbb{E}_{Y'} p(t|y'))\big) \\
&= l^2 \exp\big(-\frac{n}{l^2}\mathbb{E}_{Y} D_{KL}(p(t|y)||p(t))\big) \\
&= l^2 \exp\big(-\frac{n}{l^2}\mathbb{E}_{T,Y} \log\big(\frac{p(t,y)}{p(t)P(y)}\big)\big) \\
&= l^2 \exp\big(-\frac{n}{l^2} I(T;Y)\big). \qquad (1)
\end{aligned}$$

Equation (1) shows that if a hidden feature map T has a higher value (i.e., it is more confident about the output), the classification error will be decreased.

3.4. Estimating the Mutual Information

Our 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 MI between input or output and the hidden layers of DNNs has been a research topic with a long history in unsupervised feature learning [44]. Mutual information (MI) is an appealing mathematical measure because of its invariance with respect to smooth invertible transformations. However, estimating MI is a challenging problem in for high dimensional data. Various non-parametric methods, including k-nearest neighbors (KNN) [45], kernel density estimation [46], and histogram-based [47] methods have been proposed for estimating the MI. Parametric approaches based on Gaussian mixture models have also been studied [48]. 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 too coarse; on the other hand, they are not computationally efficient if the number of bins is too large. Another issue with binning is that different binning schemes result in various discrete variables; hence, different MI estimation. We use the parametric approach of [48] (later applied in the IB framework by [49]) for estimating the MI in our proposed method. This method is based on estimating the entropy of mixture models. We briefly review this method (Please see [48] 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

$$P(t) = \sum_{i=1}^{N} c_i P_i(t),$$

with $P_i(t)$ being the density of Gaussian 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 $P(C = i) = ci$; hence, $H(T|C) = \sum_{i=1}^{N} H(P_i) c_i$. Following the discussions in Appendix C of [49], and Section 2 of [50], the mutual information between hidden layer T and the input can be potentially infinite as T is a continuous and deterministic map from the input of the network. This means that the conditional probability $P(T|X = x)$ is a delta function at feature map T; as a result, the conditionally differential entropy, $H(T|x = x)$ is $-\infty$. In order to have a finite and computationally tractable MI, a solution is to add a standard Gaussian noise with some variance σ^2 to T. Thus, the distribution of T can be modeled using GMM with components centered on each training sample (i.e., $N = n$). The authors of [49] demonstrated that using the above Gaussian mixture model, both an upper and a lower bound for the differential entropy can be derived. Using these bounds, we can have the following upper bound on the mutual information, $I(T;Y)$. A detailed derivation is included in 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 $n_k = \sum_{i=1}^{n} \mathbb{1}(y_i = k)$.

Having reviewed all the required backgrounds and tools, we are at a position to present our proposed approach for compression of a DNN.

3.5. Proposed Approach

In the aforementioned work [14], the authors empirically observed that the pre-trained ResNet architectures are very robust against the removing 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. 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 much drop in the performance of the original model. 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 *Conv*, *BN*, *ReLU*, Pooling, Dropout, etc., such that its output is combined in some way 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 $T_l = f_l(U_{l-1})$. We call T_l as the feature maps of the unit l. Mathematically, each unit is represented by the equation

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

where $\alpha_l \in \{0, 1\}$ and Ψ denotes an operation applied to T_l and U_{l-1}. The ResNet architecture Ψ_{res} and DenseNet architecture Ψ_{den} are respectively defined by:

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

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

where $Concat$ denotes the concatenation operation, and $\mathcal{A}_{l-1}$ is an identity or convolution operator. Moreover, U_0,

the input for the first unit is typically provided by a convolution operator (e.g., in ResNet, and DenseNet). The pseudocode 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, we need to determine the units whose feature maps T_l are more informative about the class label of the given input. As discussed in section 3.3, we use $I(T_l, Y)$ to select active units (corresponding to $\alpha_l = 1$) by clustering/grouping their $I(T_l, Y)$ values. We cluster the units for different bin resolutions (The set of all resolutions is called resolution vector). Those units inside each cluster have the same information about the class label up to a resolution value. Hence, the algorithm keeps only one unit (centroid) in each cluster, and removes the other units from the model (graph of the network). Thus $\alpha = 1$ for centroids. We set $\alpha = 0$ for other units.

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, we make a grid in the interval 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, we group the units within each grid as one cluster. 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 tend to produce better accuracy when compared with a random selection of centroids. The above procedure is outlined in Algorithm 2. Next, we 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.

We note that there are 2^L possible binary outcomes for α_l, $l = 1, 2, \cdots, L$. An exhaustive search over this set is computationally intractable in most cases (e.g. ResNet and DensNet architectures with $L > 40$). On the other hand, randomly dropping a whole unit increases the error of the model. We show this by an experiment illustrated in Figure 1 for the classification task on the CIFAR-10 data set. The goal of this experiment is to compare the performance of compressing a ResNet model according to the MSPIC approach with that of random removal of the units. For this experiment, we focus on the ResNet-56 model which has 56 layers with 27 units. Without any pruning, a trained ResNet-56 model can achieve 93.54% test accuracy on CIFAR-10 data set. Each unit is constructed by two convolutional layers with batch normalization operation. Also, there is a convolutional layer in the input of the network and a fully connected last layer (after the 27^{th} unit).

In Figure 1, MSPIC is run for 4 stages ($N = 4$). The bars with colors blue and red correspond to the MI policy pruning, while those with black and green colors are obtained

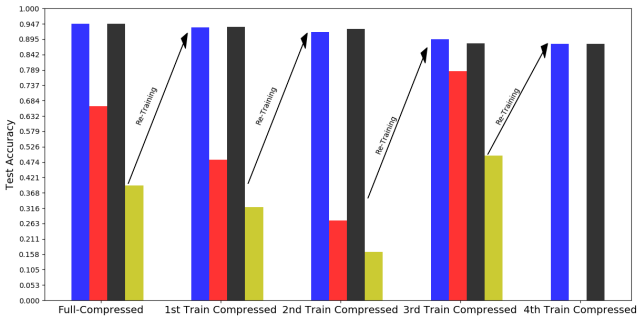

Figure 1. Test accuracy performance for classification of CIFAR-10 data set according to two policies for dropping the units in the ResNet-56 model with 27 units. Pair Blue-Red is calculated based on MI, and pair Black-Green is simulated based on random dropping policy. Blue and Black bars denote the test accuracy after re-training. Red and Green bars denote the test accuracy of the compressed network after dropping new units.

by random removal of the units and re-training the resulted model. Moreover, blue and black bars denote the test accuracy of the compressed model after re-training. Red and green bars denote the compressed network test accuracy after dropping new units. In the beginning (labeled by "full-compressed"), both blue and black bars show the test accuracy of the full ResNet-56 model trained on the CIFAR-10 data set which is pre-processed by *Cutout* technique [51]. Then, MSPIC runs on this model and determines 6 units to drop. The test accuracy is denoted by the red bar in "full-compressed". We also drop 6 units out of 27 units uniform randomly and measure the test accuracy by the green bar. Next, the second stage is started by re-training the compressed models (e.g., red and green models) which lead to the label "1st Train and compressed".

By repeating this process, we drop 9, 15 and, 17 units respectively in the second, third, and fourth stage. Table 1 shows the details of this process in terms of the number of pruned parameters, and the reduced number of operations in all the stages for both MSPIC and random approaches. We measure the number of operations as the total number of multiplication, addition, and *ReLU* operations. From Figure 1 and Table 1, we see that the error incurred by MSPIC is less than the one achieved by the random strategy, specifically when more number of units are removed. Hence, we can achieve a more pruned network by removing more parameters using MSPIC.

4. Experimental Results

In this section, we provide extensive experimental results to show the performance of MSPIC on both CIFAR-10 and CIFAR-100 data sets which consist of 50000 and 10000 training and testing images with size $32 \times 32 \times 3$.

	Approach	Test Accuracy	Reduction Percentage	Number of Parameters	Number of Operations
	Full Model	0.9479	0%	853018	126538378
$t=1$	MSPIC	0.9347	32.55%	575386	98116234
	Random	0.9366	29.29%	603162	98067082
$t=2$	MSPIC	0.9197	49.91%	427290	85090954
	Random	0.9307	39.05%	519834	83800714
$t=3$	MSPIC	0.8949	72.70%	232858	56607370
	Random	0.8808	67.81%	274586	56509066
$t=4$	MSPIC	0.8788	75.42%	209626	47096458
	Random	0.868	72.161%	237466	47022730

Table 1. Comparison of MSPIC with the random dropping the units in ResNet-56 for classification task of CIFAR-10 data set. Reduction Percentage shows the ratio of the reduced number of parameters after each stage.

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

INPUT:
S: The index of current active units
T_l: Feature maps, $l = 1, 2, \ldots, |S|$
N: Number of stages
R: Resolution vector
ϵ^t: Test Error threshold, $t = 1, 2, \ldots, N$
for $t = 1, 2, \ldots, N$ **do**
Compute $I(T_l, Y)$, $l = 1, 2, \ldots, |S|$
Construct $\mathbf{I} = [I(T_1;Y), I(T_2;Y), \ldots, I(T_{|S|};Y)]$
$\{Cluster_1^{policy_1}, \ldots, Cluster_{M^t_{policy_1}}^{policy_1}, \ldots,$
$Cluster_{M^t_{policy_{|R|}}}^{policy_{|R|}}\} = \text{Clustering}(R, \mathbf{I}, S)$
for policy in R **do**
for $j = 1, 2, \ldots, M^t_{policy}$ **do**
Keep a unit with the smallest index as the cluster centroid
$a_l = 1, \quad l = \min_{k \in Cluster_j^{policy}} k$
Remove the rest of units in the $Cluster_j$
$a_u = 0, \quad \forall u \in Cluster_j^{policy} \setminus l$
end for
Compute $Test_{error}$ for given policy
end for
Select one policy with $Test_{error} > \epsilon^t$
$S \leftarrow S \setminus \{k : a_k = 0, k \in S\}$
Re-train the resulted sub-network with the trained weights in stage t
end for
Return pruned model with S units

Algorithm 2 Clustering(R, $\mathbf{I}$, S)

INPUT:
R: Resolution vector
$\mathbf{I} = [I(T_1;Y), I(T_2;Y), \ldots, I(T_{|S|};Y)]$
$\mathcal{U} = \max_{l \in S} I(T_l;Y), \quad b = \text{argmax}_{l \in S} I(T_l;Y)$
$\mathcal{L} = \min_{l \in S} I(T_l;Y)$
for policy in R **do**
$\delta = policy \times (\mathcal{U} - \mathcal{L})$
for $j = 1, 2, \ldots, M_{policy} = \frac{1}{policy}$ **do**
$Cluster_j^{policy} = \{Unit_q : \mathcal{L} + (j-1)\delta \leq I(T_q;Y) < \mathcal{L} + j\delta, \; q \in S\}$
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 $\cup_{policy} \cup_j Cluster_j^{policy}$

Both of these data sets are used for the classification task respectively with 10 and 100 classes. The comparison of our approach with those of the state-of-the-art methods in deep compression is given in Table 2 for CIFAR-10 and Table 3 for CIFAR-100. The column labeled 'Reduction' in all the tables represents the percentage of reduction in the number of parameters. Also, the values are shown in columns labeled as "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. 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 in the supplementary file).

4.1. Compressing ResNet-based models

We have tested MSPIC on three different ResNet models including ResNet-56, ResNet-152, and ResNet-164.

4.1.1 ResNet-56

The ResNet-56 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) with and without cutout augmentation technique on the CIFAR-10 data set. These achieve respectively 94.79% and 93.54% test accuracy. Cutout is a common technique for improving the accuracy of the deep models in many image tasks. Full ResNet-56 model has 853018 training parameters, and it consists of 126538378 FLOPs including *ReLU* operation. The results of applying MSPIC on both of these models with $N = 3$ stages are shown in Table 3 tagged by ResNet-56 and ResNet-56 (Cutout). While the test accuracy of both compressed models is less than the other methods, but their FLOPs and number of operations are significantly less than the others.

4.1.2 ResNet-152

Next, we consider the ResNet-152 model with 152 layers consisting of 50 units arranged in 4 block respectively consisting of 3, 8, 36, and 3 units. Each unit has 3 convolutional layers with batch normalization. Also, there are convolution operation plus BN in some of the skip-connection paths (e.g., $\mathcal{A}_l = Conv + BN$ in equation 3)[1]. We train ResNet-152 without cutout which achieves 0.9549 test accuracy with 58156618 number of parameters and 3737432074 Flops. By applying MSPIC with $N = 6$, we can compress the model by 69.71% and sacrifice only 2% in the test accuracy.

4.1.3 ResNet-164

We also compress ResNet-164 under different conditions for both CIFAR-10 (Table 2) and CIFAR-100 (Table 3). 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 ResNet-164 with cutout 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 ResNet-164-a (Cutout) in Table 2. 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 ResNet-164-b (Cutout) in Table 2. As we can see, ResNet-164-b can achieve almost the same test accuracy as ResNet-164-a; albeit, with significantly more pruned number of parameters and number of FLOPs. Finally, in Table 3, we present the compressed model of ResNet-164 on CIFAR-100 data set with original; top-1 test error of 22.01% by running MSPIC for $N = 7$ as the number of stages. This gives a model compression rate of 45.30%.

[1]This architecture is used in PyTorch and it is contrary to the original ResNet architecture which uses zero-padding for dimension consistency between the units. Introducing these extra operations increases the number of parameters and FLOPs significantly.

4.2. Compressing DenseNet-based models

In this section, we present compression results for the DenseNet model. We focus on the DenseNet-100-k12 with 100 and growth-rate, $k = 12$ (i.e., the number of channels in T_l). This model has 48 dense units each consisting of 3 blocks, and there are two-transition layers between the blocks. Also, there are a convolutional layer and a fully connected layer in the beginning (before the first unit) and also at the 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.

When we remove a dense unit from this architecture, we need to make sure that $k = 12$ channels added to the first previous active unit as the input of transition-layer expects $k\times16$ channel as its input channels for the convolution operation. To address this, we simply zero pad the output of the first previous active unit with k channels. We note that this zero padding does not increase the number of permeates in the pruned model, and also with implementing sparse convolution, we can ignore the number of addition with these zero channels. Now, we train a DenseNet-100-k12 with and without Cutout pre-processing on CIFAR-10 data set and only with Cutout for CIFAR-100. The trained 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 2 demonstrates that by applying MSPIC for $N = 13$ stages on DenseNet-100-k12 for classification of CIFAR-10 images, we can achieve compressed model of DenseNet-100-k12-b which has a competitive compressed performance with the state-of-the-art method of CondenseNet^{light} [9]. Also, DenseNet-100-k12-a denotes the compressed model by running MSPIC for $N = 2$ stages (i.e., coarser pruning in each stage). Finally, Table 3 gives the results of compression on CIFAR-100 with Cutout. These results show that MSPIC can achieve 52.37% reduction in the number of parameters.

5. Conclusion

We proposed a new method for deep neural model compression. In particular, we considered models such

Model	Test Accuracy	Number of Parameters (M)	FLOPs (M)	Reduction
VGG-16-pruned [20]	0.9340	5.40	206	-
VGG-19-pruned [21]	0.9380	2.30	195	-
ResNet-56-pruned [20]	0.9306	0.73	90	-
ResNet-110-pruned [20]	0.9365	1.68	213	-
ResNet-164-B-pruned [21]	0.9473	1.21	124	-
DenseNet-40-pruned [21]	0.9481	0.66	190	-
CondenseNet-182* [9]	0.9624	4.2	513	-
CondenseNet^{light}-94 [9]	0.9496	0.33	122	-
CondenseNet-86 [9]	0.9496	0.52	65	-
ResNet-56 **(ours)**	0.9128	0.23	42.3	72.72%
ResNet-56 (Coutout) **(ours)**	0.8949	0.23	56.6	72.70%
ResNet-152 **(ours)**	0.9340	17.6	771.6	69.71%
ResNet-164 **(ours)**	0.0.919	0.71	153.7	57.99%
ResNet-164-a (Coutout) **(ours)**	0.9207	0.99	143.9	41.53%
ResNet-164-b (Coutout) **(ours)**	0.9173	0.47	112.5	72.02%
DenseNet-100-k12 **(ours)**	0.9264	0.32	213.2	57.69%
DenseNet-100-k12-a (Cutout) **(ours)**	0.9352	0.34	224.9	56.27%
DenseNet-100-k12-b (Cutout) **(ours)**	**0.9437**	**0.29**	**173.2**	**62.66%**

Table 2. Comparison of classification accuracy on CIFAR-10, number of parameters, and number of FLOPs between the proposed approach and those of the state-of-the-art deep compression methods.

Model	Test Accuracy	Number of Parameters (M)	FLOPs (M)	Reduction
VGG-16-pruned [20]	0.7472	5.40	206	-
ResNet-164-B-pruned [21]	0.7609	1.21	124	-
DenseNet-40-pruned [21]	0.7472	0.66	190	-
CondenseNet-182* [9]	0.8153	4.2	513	-
CondenseNet^{light}-94 [9]	0.7592	0.33	122	-
CondenseNet-86 [9]	0.7636	0.52	65	-
ResNet-164 (Coutout) **(ours)**	0.7459	0.94	130.97	45.30%
DenseNet-100-k12 (Cutout) **(ours)**	**0.7408**	**0.38**	**203.35**	**52.37%**

Table 3. Comparison of classification accuracy on CIFAR-100, number of parameters, and number of FLOPs of the proposed approach and those of the state-of-the-art deep compression methods.

as ResNet and DenseNet in which the feature maps of the previous layers 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 have provide about the output of the network. Motivated by Information Bottleneck theory, we use the mutual information to decide which units should be 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] 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*, pages 740–755, 2014. 1

[2] 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 Pattern Recog.*, pages 248–255, 2009. 1

[3] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advance. Neural Inf. Process. Sys.*, pages 1097–1105, 2012. 1

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[5] Y. Jia, Developer L., and E. Shelhamer. Caffe model zoo, 2016. 1

[6] Y. Wang, C. Xu, C. Xu, and D. Tao. Packing convolutional neural networks in the frequency domain. *IEEE Trans. Patt. Analysis. Machine. Intell.*, 2018. 1

[7] S. Han, H. Mao, and W. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding, 2016. 1, 2

[8] 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. 1, 2

[9] G. Huang, S. Liu, L. Van der Maaten, and K. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *IEEE Conf. Comp. Vision Pattern Recog.*, pages 2752–2761, 2018. 1, 2, 7, 8

[10] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *Int. Conf. Learning Representation.*, 2017. 1

[11] O. Shamir, S. Sabato, and N. Tishby. Learning and generalization with the information bottleneck. *Theoretical Comp. Science*, 411(29-30):2696–2711, 2010. 1, 3

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conf. Comp. Vision Pattern Recog.*, pages 770–778, 2016. 2, 3

[13] G. Huang, Z. Liu, L. Van Der Maaten, and K. Weinberger. Densely connected convolutional networks. In *IEEE Conf. Comp. Vision Pattern Recog.*, pages 4700–4708, 2017. 2, 3

[14] A. Veit, M. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advance. Neural Inf. Process. Sys.*, pages 550–558, 2016. 2, 4

[15] A. Krizhevsky and Ge. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 2

[16] M.u Courbariaux, Y. Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advance. Neural Inf. Process. Sys.*, pages 3123–3131, 2015. 2

[17] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 2

[18] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *IEEE Int. Conf. Comp. Vision.*, pages 1389–1397, 2017. 2

[19] L. Hou, Q. Yao, and J. Kwok. Loss-aware binarization of deep networks. *Int. Conf. Learning Representation*, 2017. 2

[20] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 2, 8

[21] 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*, pages 2736–2744, 2017. 2, 8

[22] 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. 2

[23] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Europ. Conf. Comp. Vision.*, pages 525–542. Springer, 2016. 2

[24] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advance. Neural Inf. Process. Sys.*, pages 2074–2082, 2016. 2

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014. 2

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 2

[27] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advance. Neural Inf. Process. Sys.*, pages 1269–1277, 2014. 2

[28] 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 Pattern Recog.*, pages 1039–1048, 2017. 2

[29] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. In *Europ. Conf. Comp. Vision.*, pages 304–320, 2018. 2

[30] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Confs.*, 2014. 2

[31] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun. Sbnet: Sparse blocks network for fast inference. In *IEEE Conf. Comp. Vision Pattern Recog.*, pages 8711–8720, 2018. 2

[32] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2

[33] 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. 2

[34] A. Veit and S. Belongie. Convolutional networks with adaptive computation graphs. *arXiv preprint arXiv:1711.11503*, 2017. 2

[35] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *IEEE conf. Comp. Vision Pattern Recog.*, pages 7132–7141, 2018. 2

[36] 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 Pattern Recog.*, pages 8817–8826, 2018. 2

[37] X. Wang, F. Yu, Z. Dou, T. Darrell, and J. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Europ. Conf. Comp. Vision.*, pages 409–424, 2018. 2

[38] A. G Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2

[39] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *IEEE Conf. Comp. Vision Pattern Recog.*, pages 6848–6856, 2018. 2

[40] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Int. Conf. Machine Learning*, pages 448–456, 2015. 3

[41] N. Tishby, F. Pereira, and W. Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000. 3

[42] N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *IEEE Inf. Theory. Workshop. (ITW)*, pages 1–5. IEEE, 2015. 3

[43] T. Cover and J. Thomas. *Elements of information theory*. John Wiley & Sons, 2012. 3

[44] R Linsker. Self-organization in a perceptual network. computer. pages 105–117, 1988. 4

[45] A. Kraskov, H. Stögbauer, and P. Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004. 4

[46] Y. Han, J. Jiao, T. Weissman, and Y. Wu. Optimal rates of entropy estimation over Lipschitz balls. *arXiv preprint arXiv:1711.02141*, 2017. 4

[47] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017. 4

[48] A. Kolchinsky and B. Tracey. Estimating mixture entropy with pairwise distances. *Entropy*, 19(7):361, 2017. 4, 10, 11

[49] A. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. Tracey, and David D. On the information bottleneck theory of deep learning. In *Int. Conf. Learning Representation.*, 2018. 4

[50] A Kolchinsky, B. Tracey, and D. Wolpert. Nonlinear information bottleneck. *arXiv preprint arXiv:1705.02436*, 2017. 4, 11

[51] T. DeVries and G. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 5

6. Appendix

6.1. 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 [48], where the authors gave an estimate of the differential entropy. To this end, recall our notation of Section 3.4. 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 (No Cutout)	Test Accuracy	Reduction Percentage	Number of Parameters	Number of Operations
DenseNet-100-k12 (full)	0.9531	0%	769162	293546820
DenseNet-100-k12 (t=1)	0.9490	24.34%	581962	267113028
DenseNet-100-k12 (t=2)	0.9458	34.40%	504562	258479940
DenseNet-100-k12 (t=3)	0.9392	42.33%	443542	245534916
DenseNet-100-k12 (t=4)	0.9356	47.32%	405202	238716228
DenseNet-100-k12 (t=5)	0.9362	49.92%	385222	237542340
DenseNet-100-k12-a (t=6)	0.9314	51.89%	370042	236669508
DenseNet-100-k12-b (t=6)	0.9264	57.69%	325402	213187140

Table 4. Pruning of DenseNet-100-k12 in classification of CIFAR-10 data set (using Cutout) during different stages. DenseNet-100-k12-a denotes the pruned model in which MSPIC has selected 24 units to be removed, while DenseNet-100-k12-b corresponds to 27 removed dense units.

Model (Cutout-Coarse prune)	Test Accuracy	Reduction Percentage	Number of Parameters	Number of Operations
DenseNet-100-k12 (full)	0.9573	0%	769162	293546820
DenseNet-100-k12 (t=1)	0.9484	37.84%	478102	254783172
DenseNet-100-k12-a (t=2)	0.9477	44.15%	429562	245172804
DenseNet-100-k12-b (t=2)	0.9352	56.27%	336382	224876484

Table 5. Pruning of DenseNet-100-k12 in classification of CIFAR-10 data set (using Cutout) during different stages. DenseNet-100-k12-a denotes the pruned model in which MSPIC has selected 20 units to be removed, while DenseNet-100-k12-b corresponds to 26 removed dense units.

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:

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

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., $H(T) \leq \frac{d}{2} - \sum_{i=1}^{N} c_i \ln \sum_{j=1}^{N} c_j p_j(\mu_i)$ [50]. Moreover, [48] shows that by instantiating the entropy estimator with the Chernoff α-divergence (i.e., $C_\alpha(p_i||p_j) = -\ln \int p_i^\alpha(x) p_j^{1-\alpha}(x) dx$) with $\alpha = 0.5$ and with the same GMM, a lower bound on the entropy can be derived given by:

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

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:

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

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

6.2. 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 4 shows the details of pruned DenseNet-100-k12 in classification task on CIFAR-10 data set (without using Cutout) for $N = 6$ stages. Here, DenseNet-100-k12-a denotes the pruned model in which MSPIC has selected 24 units

Model (Cutout-Fine prune)	Test Accuracy	Reduction Percentage	Number of Parameters	Number of Operations
DenseNet-100-k12 (full)	0.9590	0%	769162	293546820
DenseNet-100-k12 (t=1)	0.955	24.52%	580582	274015428
DenseNet-100-k12 (t=2)	0.9456	45.58%	418582	245120964
DenseNet-100-k12 (t=3)	0.9479	47.67%	402502	241072068
DenseNet-100-k12 (t=4)	0.9479	49.06%	391822	238046148
DenseNet-100-k12 (t=5)	0.9457	50.68%	379342	235232196
DenseNet-100-k12 (t=6)	0.9436	52.62%	364462	231484356
DenseNet-100-k12 (t=7)	0.9452	54.47%	350182	215768004
DenseNet-100-k12 (t=8)	0.9424	55.94%	338902	213255108
DenseNet-100-k12 (t=9)	0.9425	57.25%	328822	201753540
DenseNet-100-k12 (t=10)	0.9425	59.03%	315142	187966404
DenseNet-100-k12 (t=11)	0.943	61.51%	296062	183496644
DenseNet-100-k12 (t=12)	0.9437	62.66%	287182	173199300
DenseNet-100-k12 (t=13)	0.9354	64.56%	272602	172364100

Table 6. Pruning of DenseNet-100-k12 in classification of CIFAR-10 data set (using Cutout) during different stages. Here, the pruned units are done in a finer way compared to Table 5.

Model (Cutout (C-100))	Test Accuracy	Reduction Percentage	Number of Parameters	Number of Operations
DenseNet-100-k12 (full)	0.7793	0%	800032	304104360
DenseNet-100-k12 (t=1)	0.7695	16.47%	668272	291888552
DenseNet-100-k12 (t=2)	0.7646	25.88%	592972	280689192
DenseNet-100-k12 (t=3)	0.7644	34.07%	527452	257898024
DenseNet-100-k12 (t=4)	0.7552	36.64%	506872	256686504
DenseNet-100-k12 (t=5)	0.7501	39.91%	480712	238997928
DenseNet-100-k12(t=6)	0.7516	41.40%	468832	235670952
DenseNet-100-k12 (t=7)	0.7479	43.22%	454252	234835752
DenseNet-100-k12 (t=8)	0.7436	45.68%	434572	229754664
DenseNet-100-k12 (t=9)	0.7447	47.09%	423292	226909992
DenseNet-100-k12 (t=10)	0.7414	48.65%	410812	223764264
DenseNet-100-k12 (t=11)	0.7406	50.43%	396532	220166952
DenseNet-100-k12 (t=12)	0.7408	52.37%	381052	203246376
DenseNet-100-k12 (t=13)	0.7316	54.64%	362872	202185384

Table 7. Pruning of DenseNet-100-k12 in classification of CIFAR-100 (with using Cutout) data set during different stages.

to be removed, while DenseNet-100-k12-b corresponds to the pruned model with 27 dense units. Table 5 shows the pruned details in the intermediate stages of MSPIC applied on DenseNet-100-k12 using the Cutout pre-processing technique. In Table 5, DenseNet-100-k12-a denotes the pruned model in which MSPIC has selected 20 units to be removed, while DenseNet-100-k12-b corresponds to the pruned model with 26 dense units.

Next, we have Table 6 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 5. Finally, Table 7 presents the results for pruning the DenseNet-100-k12 for classifying the CIFAR-100 data set using Cutout.