Redundant Feature Pruning for Accelerated Inference in Deep Neural Networks

Email addresses: babajide.ayinde@louisville.edu (Babajide O. Ayinde), t.inanc@louisville.edu (Tamer Inanc), jacek.zurada@louisville.edu (Jacek M. Zurada)

Redundant Feature Pruning for Accelerated Inference in Deep Neural Networks

Babajide O. Ayinde[a,*], Tamer Inanc[a], Jacek M. Zurada[a,b,**]

[a]*Electrical and Computer Engineering, University of Louisville, Louisville, KY, 40292 USA.*
[b]*Information Technology Institute, University of Social Science,Łódz 90-113, Poland*

Abstract

This paper presents an efficient technique to reduce the inference cost of deep and/or wide convolutional neural network models by pruning redundant features (or filters). Previous studies have shown that over-sized deep neural network models tend to produce a lot of redundant features that are either shifted version of one another or are very similar and show little or no variations; thus resulting in filtering redundancy. We propose to prune these redundant features along with their related feature maps according to their relative cosine distances in the feature space, thus leading to smaller networks with reduced post-training inference computational costs and competitive performance. We empirically show on select models (VGG-16, ResNet-56, ResNet-110, and ResNet-34) and dataset (MNIST Handwritten digits, CIFAR-10, and ImageNet) that inference costs (in FLOPS) can be significantly reduced while overall performance is still competitive with the state-of-the-art.

Keywords: Deep learning, feature correlation, filter pruning, cosine similarity, redundancy reduction, deep neural networks.

1. Introduction

Efficient architectural design of deep neural networks (DNNs) has shown superior performance in many supervised learning tasks ranging from computer vision (Simonyan and Zisserman, 2015; Krizhevsky et al., 2012; He et al., 2016) to speech recognition (Graves et al., 2013;

*Corresponding author
**Principal corresponding author
Email addresses: babajide.ayinde@louisville.edu (Babajide O. Ayinde), t.inanc@louisville.edu (Tamer Inanc), jacek.zurada@louisville.edu (Jacek M. Zurada)

Graves and Jaitly, 2014; Oord et al., 2016) and natural language processing (Jozefowicz et al., 2016; Shazeer et al., 2017). Over the past few years, the general trend has been that DNNs have grown deeper and wider, amounting to huge number of final parameters. Their flexibility and performance usually come with high computational and memory demands both during training and inference. However, a number of recent studies have shown that over-sized deep learning models typically result in largely over-determined (or over-complete) systems (Denil et al., 2013; Bengio and Bergstra, 2009; Changpinyo et al., 2017; Ayinde and Zurada, 2017; Han et al., 2015, 2017). For instance, such over-complete representation is evidently pronounced in features learned by the popular DNN of AlexNet (Krizhevsky et al., 2012) as emphasized by Rodríguez et al. (2016); Zeiler and Fergus (2014).

The resulting oversized architectures are by nature less computationally efficient due to their size, over-parameterization and their higher inference cost. To account for the scale, diversity and the difficulty of data these models learn from, the architectural complexity and the excessive number of weights are often deliberately built in into the initial design of DNNs (Bengio et al., 2007; Changpinyo et al., 2017). These over-sized models have expensive training and inference costs especially for applications with constrained computational and power resources such as web services, mobile and embedded devices. In addition to good accuracy, many resource-limited applications would greatly benefit from lower post-training inference cost (Li et al., 2017; Szegedy et al., 2016). While the demand for high computational inefficiency in the training phase has been alleviated with general-purpose computing engines otherwise known as Graphics Processing Units (GPUs) to accelerate computations but powerful GPUs are still unavailable in hand-held and wearable devices.

Additionally, flexibility of DNN models may hinder their scalability and practicality, and may result in extracting highly redundant parameters with risk of over-fitting (Yoon and Hwang, 2017). A symptom of learning replicated or similar features is that two or more processing units extract very similar and correlated information. From an information value standpoint, similar or shifted versions of features do not add extra information to the feature set, and could be possibly suppressed. In other words, the activation of one unit should not be predictable based on the activations of other units of the same layer. However, enforcing dissimilarity of features

in a traditional way can be generally involved, requiring computation of intractable joint probability table and batch statistics. To address this problem of over-representation, layers of deep and/or wide architectures have to be examined for possible redundancy and possible filter reduction after training.

Knowing the level of redundancy in models is useful mainly for two reasons: First, information about the level of redundancy in models can be used for feature diversification and improved performance (Ayinde et al., 2019; Rodríguez et al., 2016; Yoon and Hwang, 2017; Cogswell et al., 2016). Secondly, it can be used to build accurate inference-cost-efficient models via pruning for resource-limited applications that require lower inference cost and high accuracy (Li et al., 2017; Szegedy et al., 2016). This is important in practice because optimal architectures are unknown. However, pruning enables smaller model to preserve knowledge from a larger model. Since learning a complex function starting with a small initial architecture might result in low accuracy, it is therefore necessary to first learn a task with larger architecture and many parameters and later follow by pruning redundant and less important features (Anwar et al., 2017). In particular, model compression via pruning is important when transferring DNNs to resource-limited portable devices.

The contributions of this paper are:

- proposes a simple, intuitive, and optimized algorithms to localize and eliminate redundancy in DNNs without undermining their efficiency or introducing sparsity that would require specialized library and/or hardware
- leverages on redundant features of well-trained deep learning models for controlled network size reduction using feature agglomeration followed by *one-shot* redundant feature elimination and retraining
- in-depth layer-wise analysis of large-capacity DNNs for redundancy and pruning sensitivity
- empirically shows that proposed pruning technique improves inference cost over recently proposed techniques across a number of benchmark models and dataset without signifi-

cantly deteriorating the output performance or modifying existing hyperparameters.

The rest of the paper is structured as follows: Section 2 discusses the state-of-the-art. The proposed novel redundant feature detection and pruning using agglomerative hierarchical clustering is introduced in Section 3. Section 4 discusses the experimental designs and presents the results. Finally, conclusions are drawn in Section 5.

2. Related Work

Storage and computational cost reductions via model pruning techniques have a long history (LeCun et al., 1990; Hassibi and Stork, 1993; Mariet and Sra, 2016; Ioannou et al., 2016; Polyak and Wolf, 2015; Molchanov et al., 2017). For instance, Optimal Brain Damage (LeCun et al., 1990) and Optimal Brain Surgeon (Hassibi and Stork, 1993) use second-order derivative information of the loss function to prune redundant network parameters. Other related work include but not limited to Anwar et al. (2017) which prunes based on particle filtering, Mathieu et al. (2013) uses FFT to avoid overhead due to convolution operation, and Howard et al. (2017) uses depth multiplier method to scale down the number of filters in each convolutional layer. Taylor expansion of the network function with respect to activations was used in Molchanov et al. (2017) to remove both low-activation and low-gradient neurons.

Feature redundancy has also been explored to construct a low rank basis of features that are rank-1 in the spatial domain. However, this method involves additional cumbersome optimization procedures. As demonstrated in Denil et al. (2013), a fraction of the parameters is sufficient to reconstruct the entire network by simply training on low-rank decompositions of the weight matrices. HashedNets use a hash function to randomly group weights into hash buckets, so that all weights within the same hash bucket share a single parameter value for pruning purposes (Chen et al., 2015). Redundant feature maps are removed from a well trained network using particle filtering to select the best combination from a number of randomly generated masks (Anwar et al., 2017). With the assumptions that features are co-dependent within each layer, Ioannou et al. (2016) groups features in hierarchical order. Driven by feature map redundancy, Zhang et al. (2016) factorizes a layer into 3×3 and 1×1 combinations and prunes

redundant feature maps.

Another important and popular paradigm is network compression via knowledge distillation (KD) originally introduced in Bucilua et al. (2006) and formalized in Hinton et al. (2015). The main idea in KD is the transferability of "knowledge" from a model known as the teacher (usually of high capacity and performance) to another compact model known as the student (Ba and Caruana, 2014; Urban et al., 2017; Rusu et al., 2015). A standard practice to perform knowledge distillation is to make the student model reproduce the outputs of a trained teacher model, which is a two-stage training procedures containing pre-training stage on teacher and distilling stage on student. A key assumption is that the performance of the student cannot rival that of the teacher when trained directly on the data, but with KD the student is pushed closer to matching the predictive power of the teacher (Furlanello et al., 2018). In other words, the large-scale teacher trained on a certain task teaches a shallower student network to enhance its learning capability on identical task. In Bucilua et al. (2006), the information in an ensemble of neural networks is compressed into a single model and (Ba and Caruana, 2014) improves the performance of compact network by training it to mimic a larger teacher model and penalizing its cost function with the L2 norm of the difference between the student's and teacher's logits. A method called dark knowledge was demonstrated in Hinton et al. (2015), where the student model was trained with the objective of matching the full softmax distribution of the teacher model.

More recently, Huang et al. (2018) trains another neural network as pruning agent which takes filter weights of the model to be pruned as input and outputs binary decisions to remove or keep filters. Using the concept of tensor factorization and reconstruction, He et al. (2017) eliminates redundancy by pruning the feature maps instead of filter weights. The computational complexity of convolutional networks has been reduced by filter-group convolution with tiny accuracy loss while mostly preserving diversity in feature representation. Network reduction problem has also been formulated as a binary integer optimization with a closed-form solution based on final response importance Yu et al. (2018). Instead of localizing the redundant neurons in a fully-connected network, Mariet and Sra (2016) compresses a trained model by identifying a subset of diverse neurons using a determinantal point process.

The advantages of structured pruning of network parameters have been highlighted in Li et al. (2017), where filters are sorted and pruned based on the sum of their absolute weights. In their method, Li et al. (2017) uses a simple thresholding to prune all filters with low L1-norm, while our method prunes filters based on relative cosine similarity measure. We remark that using the approach in Li et al. (2017), all similar/identical filters with relatively high L1-norm will likely go unpruned. Closely related to our work, Han et al. (2015) prunes weights with magnitude below a set threshold. Since pruning is performed at weight level, this limits the number of computation that can be saved. Our method, on the other hand, performs filter-level pruning and thus leverage on eliminating many weights at once. In similar spirit with Han et al. (2015), the notion of pruning filters using similarity has been previously explored in RoyChowdhury et al. (2017), where a naive and suboptimal threshold of similarity is used to determine if filters are duplicates. On the flip side, our method use hierarchical clustering to minimize the average distance among intra-cluster filters and maximize inter-cluster distance; thus localizing redundancy better. However, the approach in RoyChowdhury et al. (2017) is faster because it does not go through many iterations of clustering.

3. Convolutional Feature Clustering and Pruning

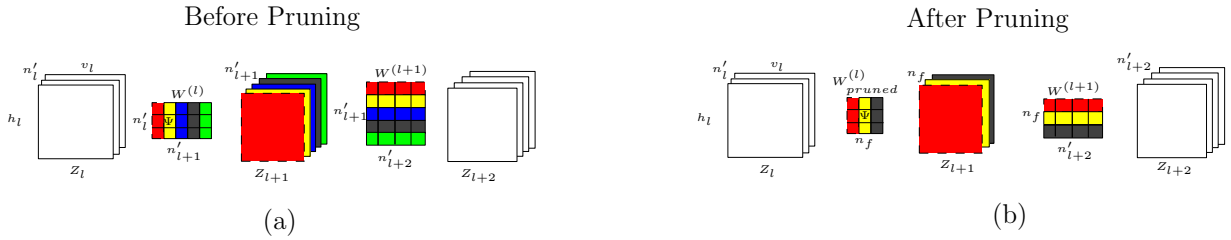


Figure 1: Pruning schema of l^{th} layer. (a) Assume filters red, blue, and green in the first, third, and fifth columns of $\mathbf{W}^{(l)}$, respectively, are very similar and are located in the same cluster (b) If filter red is sampled as the representative of the cluster, filters blue and green are redundant and their corresponding feature maps in Z_{1+1} and related weights in the next layer (third and fifth rows of $\mathbf{W}^{(l+1)}$) are all pruned. Figure is best viewed in color.

Typically, DNNs consist of input, output, and many intermediate processing layers. By letting the number of channels, height and width of input to the l^{th} layer be denoted as n'_l, h_l, and v_l, respectively. A layer (convolutional or fully-connected) in the network transforms input $Z_l \in \mathbb{R}^p$ into output $Z_{l+1} \in \mathbb{R}^q$, where Z_{l+1} serves as the input in layer $l+1$. For convolutional

neural network (CNN), p and q are given as $n'_l \times h_l \times v_l$ and $n'_{l+1} \times h_{l+1} \times v_{l+1}$, respectively. Whereas for a fully-connected network (FCN), p and q denote $n'_l h_l v_l \times 1$ and $n'_{l+1} \times 1$, respectively. A convolutional layer convolves Z_l with n'_{l+1} $3D$ filters $\chi \in \mathbb{R}^{n'_l \times k \times k}$, resulting in n'_{l+1} output feature maps (Z_{l+1}). Each $3D$ filter consists of n'_l $2D$ kernels $\zeta \in k \times k$. Unrolling and combining all features (or filters) in a single matrix results in kernel matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{m \times n'_{l+1}}$ where $m = k^2 n'_l$. In FCN, however, a layer operation involves only vector-matrix multiplication with kernel matrix $\mathbf{W} \in \mathbb{R}^{m \times n'_{l+1}}$, where $m = n'_l h_l v_l$. Additionally, $\mathbf{w}_i^{(l)}$, i=1,...n'_l, denotes ith feature in layer l, each $\mathbf{w}_i^{(l)} \in \mathbb{R}^m$ corresponds to the i-th column of the kernel matrix $\mathbf{W}^{(l)} = [\mathbf{w}_1^{(l)}, \ ...\mathbf{w}_{n'_l}^{(l)}] \in \mathbb{R}^{m \times n'_{l+1}}$.

In this section, two heuristics that aim at agglomerating and pruning convolutional features are introduced. The objective is to discover n_f features that are representative of n'_l original features using agglomerative hierarchical clustering approach. Achieving effective clustering of features requires choosing suitable similarity measures that express the inter-feature distances between features w_i that connect the feature map Z_{l-1} to feature maps of layer l. A number of suitable agglomerative similarity testing/clustering algorithms can be applied for localizing redundant features. Based on a comparative review, a clustering approach from Walter et al. (2008); Ding and He (2002) has been adapted and reformulated for this purpose.

By starting with each feature vector $\mathbf{w}_i$ as a potential cluster, agglomerative clustering is performed by merging the two most similar clusters C_a and C_b as long as the average similarity between their constituent feature vectors is above a chosen cluster similarity threshold denoted as τ Leibe et al. (2004); Manickam et al. (2000). The pair of clusters C_a and C_b exhibits average mutual similarities as follows:

$$\begin{aligned} \overline{SIM_C}(C_a, C_b) = \frac{\sum_{\phi_i \in C_a, \phi_j \in C_b} SIM_C(\phi_i, \phi_j)}{|C_a| \times |C_b|} > \tau \\ a, b = 1, ...n'_l; \ a \neq b; \ \ i = 1, ...|C_a|; \\ j = 1, ...|C_b|; \quad and \quad i \neq j \end{aligned} \tag{1}$$

where $\phi_i = {}^{w_i}/\sqrt{||w_i||^2}$, $SIM_C(\phi_1, \phi_2) = \frac{<\phi_1, \phi_2>}{\|\phi_1\| \|\phi_2\|}$ is the cosine similarity between two features and $< \phi_1, \phi_2 >$ is the inner product of arbitrary feature vectors ϕ_1 and ϕ_2, and τ is a set similarity threshold.

It is remarked that the above similarity definition uses the graph-based-group-average technique, which defines cluster proximity/similarity as the average of pairwise similarities (that is, the average length of edges of the graph) of all pairs of features from different clusters. In this work, other similarity definitions such as the single-link and complete-link were also experimented with. Single-link approach defines cluster similarity as the similarity between the two closest feature vectors that are in different clusters. On the other hand, complete-link assumes that cluster distance is the distance between the two farthest feature vectors of different clusters. Group average similarity definition empirically yielded better performance compared to the other two definitions and thus, we report experimental results using average similarity approach.

It is also important to note the importance of τ in (1). As an example, if we consider a model trained on CIAFR-10 dataset that has "automobiles" and "trucks" as two of its ten categories. If a particular lower-level feature describes the "wheel", then, it will not be out of place if two higher-level features describing automobile and truck share common feature that describes the wheel. The choice of τ determines the level of sharing allowed, that is, the degree of feature sharing across features of a particular layer. In other words, τ serves as a trade-off parameter that ensures a degree of feature sharing across multiple high-level features and at the same time ensuring features that are highly correlated are eliminated. Our core assumption rests on the premise that if two or more filters are similar and above allowable threshold τ, they extract feature maps that are nearly the same and can be eliminated with no significant information loss. From information theory standpoint, near-identical feature maps extracted by similar filters are not adding extra useful information to the following layer.

3.1. Method A: Pruning of Redundant Filters

The redundant convolutional feature-based pruning is detailed in Algorithm 1 with the objective of grouping filter vectors that are nearly identical in the weight space. The algorithm also aims at removing filters that are nearly identical to eliminate duplicative retrieval of feature maps. The detection and removal of redundant filters is generally tractable especially from practical standpoint since the pruning heuristic uses one-shot pruning and retraining mechanism. As highlighted in Algorithm 1, the proposed pruning heuristic assumes a fully-trained

model as input and filter grouping is performed at every layer of the model.

For a particular layer of the DNN model as in Figure 1, Algorithm 1 uses Algorithm 2

Algorithm 1 : Redundant Filter-based Pruning

for layer l in the trained model **do**
  get: convolutional filters of l^{th} layer $W^{(l)}$
  set: τ
  Extract: distinct filters in $\mathbf{W}^{(l)}$
  L_f, n_f = FILTERCLUSTERING$\left(\mathbf{W}^{(l)}, \tau\right)$
  initialize: $\mathbf{W}^{(l)}_{pruned}$ of the pruned model
  $k \leftarrow 0$
  for i in L_f **do**
    copy: i^{th} column of $\mathbf{W}^{(l)}$ into k^{th} column of $\mathbf{W}^{(l)}_{pruned}$
    $k \leftarrow k + 1$
  end for
end for
Construct the pruned model
Initialize the weights of l^{th} layer with $\mathbf{W}^{(l)}_{pruned}$
set: τ, *retrain_epoch*
for prescribed number of *retrain_epoch* **do**
  fine-tune the pruned model
end for

to group all the filters ϕ_i (columns of the kernel matrix $\mathbf{W}^{(l)}$) into n_f clusters whose average similarity among cluster members is above a set threshold τ while ensuring $n_f \leq n'$. One representative filter is randomly sampled from each of the n_f clusters. The output of Algorithm 2 is the list L_f of indices of clusters' representatives, which is equivalent to a subset of the indices of columns of $\mathbf{W}^{(l)}$. Algorithm 1 uses L_f to subset $\mathbf{W}^{(l)}$ and creates a smaller kernel matrix $\mathbf{W}^{(l)}_{pruned}$. After obtaining all the new kernel matrices $\mathbf{W}^{(l)}_{pruned}$, Algorithm 1 constructs a smaller model initialized with $\mathbf{W}^{(l)_{pruned}}$. For instance in Figure 1a, there are five filters 1,2,3,4,5 (n'=5) with color codes: red, yellow, blue, grey, and green. Each color code corresponds to a column of kernel matrix $\mathbf{W}^{(l)}$. The convolution of these filters with 3D input image Z_l yields five feature maps Z_{l+1}. Similarly, feature maps Z_{l+1} are connected to those in layer $l+2$ by kernel matrix $\mathbf{W}^{(l+1)}$. Assume the clustering of columns of $\mathbf{W}^{(l)}$ resulted in three clusters with members {1,3,5}, {2}, and {4}, resulting from the fact that filters 1, 3, and 5 have average pairwise similarity greater than τ. Thus they are grouped into the same cluster, in this case, cluster 1. Using algorithm 2

and assuming that filter 1 is the representative of cluster 1, the number of nonredundant filters $n_f = 3$ (corresponding to the number of clusters) and the list of indices of distinct filters L_f is then given as {1,2,4}. As a result, filters 3 (blue) and 4 (green) are automatically tagged as being redundant. Figure 1b depicts a compressed network where filters 3, 4, and their corresponding feature maps in Z_{l+1} have been pruned. The weights of the pruned feature maps in the next convolutional layer ($\mathbf{W}^{(l+1)}$) are also removed.

In general, pruning a large fraction of filters generally results in performance deterioration. In fact, it is observed that some convolutional layers are extremely sensitive to pruning than others and this must be taken into consideration when pruning such layers and/or models. In most cases, restoring the performance after pruning, the pruned model is fine-tuned for prescribed number of epochs. It must be noted that if two or more filters are grouped into the same cluster because they have cosine similarity (1) above τ, our approach in Algorithm 2 randomly chooses one out of them as the cluster representative. Another approach considered in this work uses cluster centroid to represent the cluster. However, it has been observed that the performance of this approach is similar to the random sampling of representative within each cluster as in Algorithm 2. This further suggests that the cluster centroid is very close to all filters in the cluster. In this paper, random selection is used in Algorithm 2 to inject some stochasticity in the selection process.

Algorithm 2 : Localization and Pruning of Redundant Filters (Method A)

function FILTERCLUSTERING():
    Input: {$\mathbf{W}$, τ}
        Scan for: cluster(s) of vectors in $\mathbf{W}$ with similarity $> \tau$
        Randomly sample and tag one representative filter from each of the n_f clusters as nonredundant
    Outputs: List of Indices L_f of distinct filters and n_f in $\mathbf{W}$;
        return L_f, n_f,
end function

3.2. Method B: Pruning of Random n_f Filters

Here, Algorithm 3 is used to detect the number of n_f distinct filters in kernel matrix $\mathbf{W}^{(l)}$ of a given layer l. It then randomly samples n_f out of n_l' filters to construct the kernel matrix

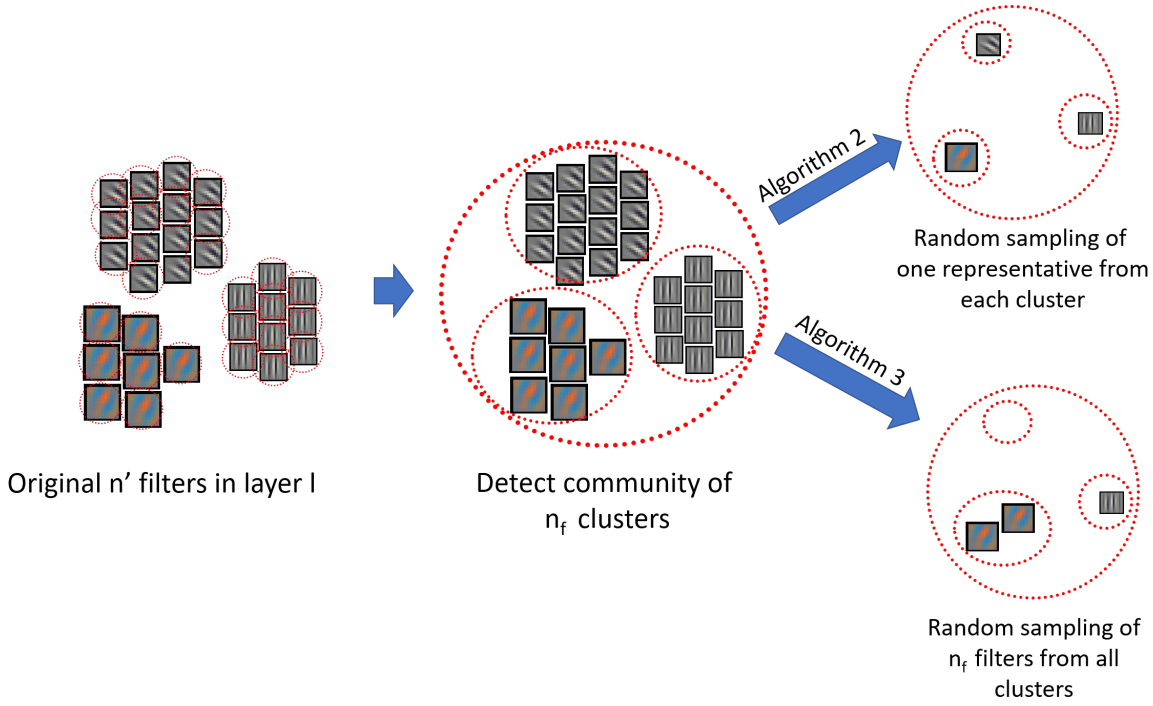


Figure 2: Illustration of how Algorithms 2 and 3 are independently used by Algorithm 1

$\mathbf{W}^{(l)}_{pruned}$ of the pruned model. It is worth mentioning that Algorithms 2 and 3 are alternative approaches used by Algorithm 1. As shown in Figure 2, the main difference is that Algorithm 2

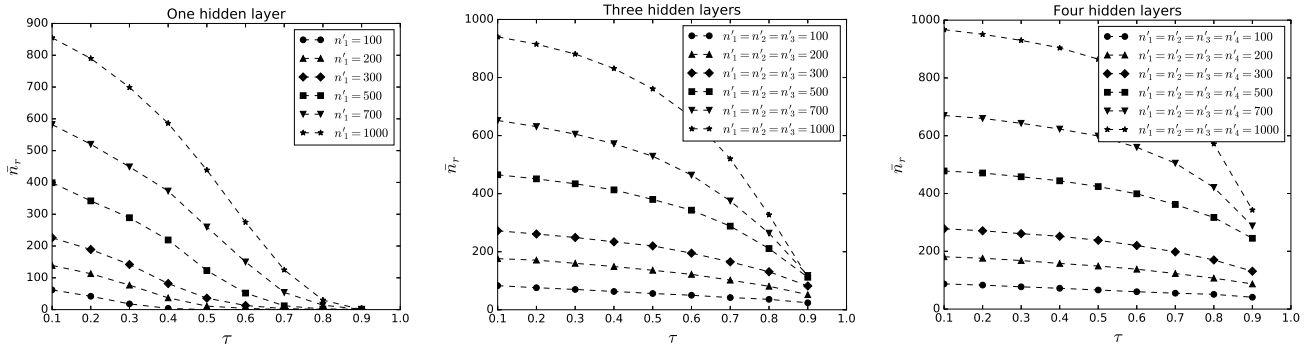
Algorithm 3 Estimation and Random Pruning of n_f filters (Method B)

function FILTERCLUSTERING():
  Input: {**W**, τ}
    Scan for: cluster(s) of vectors in **W** with similarity $> \tau$ to estimate n_f
    Randomly sample n_f filters
  Outputs: List of Indices L_f of randomly sampled filters and n_f in **W**;
    return $L_f, n_f,$
end function

randomly samples one filter out of every cluster of filters and prunes the remaining filters in all n_f clusters. Note that $n_f <= n_l$, where n_l is the total number of filters in layer l. Algorithm 3 on the other hand uses filter clustering algorithm only to estimate n_f (the number of distinct filter clusters) and randomly prunes n_f out of n_l filters. In other words, Algorithm 2 localizes and prunes precisely the redundant filters, while Algorithm 3 just estimates how many filters to randomly prune. As illustrated in Figure 2, Algorithm 2 is expected to be performing better

than Algorithm 3 because of its ability to sample from each distinct cluster and eliminate redundancy. Whereas in Algorithm 3, the probability of either sampling more than one filter from the same cluster or not sampling filters at all from some other clusters is higher. This phenomenon of over-representation of some clusters and non-representation of other clusters in Algorithm 3 could lead to redundancy, loss of vital information, and deterioration of post-pruning performance.

4. Experiments

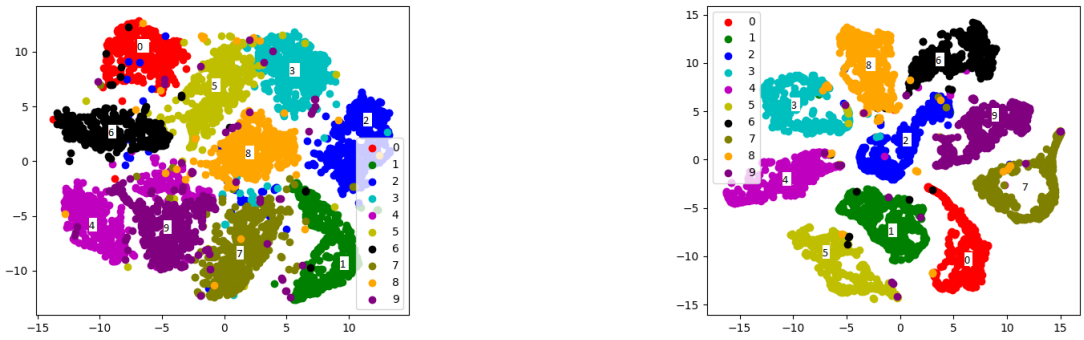


(a) (b) (c)

Figure 3: Average number of redundant features across all layers ($\bar{n}_r$) against threshold τ with (a) one (b) three, and (c) four hidden layers using MNIST dataset. Network width corresponds to the number of hidden units per layer and network depth corresponds to number of hidden layers. Networks with more than one hidden layer have equal number of hidden units in all layers.

All experiments were performed on Intel(r) Core(TM) i7-6700 CPU @ 3.40Ghz and a 64GB of RAM running a 64-bit Ubuntu 14.04 edition. The software implementation has been in Pytorch library [1] on two Titan X 12GB GPUs and the filter clustering was implemented in SciPy ecosystem (Jones, Oliphant, Peterson, et al., Jones et al.). The agglomeration of filters using hierarchical clustering is practical for very wide and deep networks even though the complexity of the agglomerative clustering algorithm itself is $O((n'_l)^2 \log(n'_l))$. In most network, $n'_l \leq 1000$ and number of layers is often less than 200. For instance, clustering VGG-16 feature vectors empirically takes 14.1 milliseconds and this is executed only once during training. This amounts

[1] http://pytorch.org/



(a) (b)

Figure 4: t-SNE projection (Maaten and Hinton, 2008) of the activation of last layer of network with (a) one and (b) four hidden layers using 5000 MNIST handwritten digits test samples. All networks have 1000 hidden units in all layers and use sigmoid activation function.

to a negligible computational overhead for most deep architectures.

The implementation of our filter pruning strategy is similar to that in Li et al. (2017) in the sense that when a particular filter of a convolutional layer is pruned, its corresponding feature map is also pruned and the weights of the pruned feature map in the filter of the next convolutional layer are equally pruned. It must be emphasized that after pruning the feature maps of last convolutional layer, the input to the fully-connected layer has changed and its weight matrix must be pruned accordingly.

In the preliminary experiment, a multilayer perceptron was trained using MNIST digits (LeCun, 1998). The standard MNIST dataset has 60000 training and 10000 testing examples. Each example is a grayscale image of an handwritten digit scaled and centered in a 28×28 pixel box. Adam optimizer (Kingma and Ba, 2014) with batch size of 128 was used to train the model for 400 epochs. The number of redundant feature was computed as $n_r = n_l' - n_f$ after the models have been fully trained. Figures 3 a,b, and c show the performance of multilayer perceptron with one, two, and four hidden layer(s), respectively. The average number of redundant features across all layers of the network is denoted as $\bar{n}_r$. It can be observed in Figure 3 that both width (number of hidden units per layer) and depth (number of layers in the network) increase $\bar{n}_r$. As the number of hidden units per layer increases, $\bar{n}_r$ grows almost linearly. Also, the higher the number of hidden layers in a network, the higher the average number of redundant features

extracted and the higher the average feature pairwise correlations.

For instance, the network with one hidden layer and 100 hidden units does not have any feature pair with similarity above 0.4. However, as the depth increases (for two or more hidden layers) more feature pairs have similarity above 0.4. This observation is similar for other hidden layer sizes (200, 300, 500, 700, and 1000) and depth. In particular, as can be observed in Figure 3c that many feature pairs in deep multilayer network (with four hidden layers) are almost perfectly correlated with cosine similarity of 0.9 even with just 100 hidden units per layer. Deep multilayer network was also evaluated based on the distribution of data in high level feature space. In this regard, t-distributed stochastic neighbor embedding (t-SNE) (Maaten and Hinton, 2008) was used to project the last hidden activations of a four-layer network and that of a single layer as shown in Figures 4a and b, respectively. The t-SNE projections show that network with four hidden layers has clustered activations compared to that of a single layer resulting in within class holes. This is observation is pronounced for activations of digit 7.

CIFAR-10 dataset (Krizhevsky and Hinton, 2009) was used in the second set of large-scale experiments to validate and retrain pruned models. The dataset contains a labeled set of 60,000 32x32 color images belonging to 10 classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is split into 50000 and 10000 training and testing examples, respectively. FLOP was used to compare the computational efficiency of the models because its evaluation is independent of any underlying software and hardware. In order to fairly compare our method with state-of-the-art, we also calculated the FLOP only for the convolution and fully connected layers. For CIFAR-10 dataset, we evaluated the proposed redundant-feature-based pruning on three deep networks, namely: VGG-16 (Simonyan and Zisserman, 2015) and two residual networks ResNet-56 and 110 (He et al., 2016). The baseline accuracy for residual networks were obtained by following the procedures highlighted in He et al. (2016). We have shared our pruning implementation and trained model for reproducibility of results [2].

[2] https://github.com/babajide07/Redundant-Feature-Pruning-Pytorch-Implementation

4.1. VGG-16 on CIFAR-10

In this set of experiments, we used a modified version of the popular convolutional neural network known as the VGG-16 (Simonyan and Zisserman, 2015), which has 13 convolutional layers and 2 fully connected layers. In the modified version of VGG-16, each layer of convolution is followed by a Batch Normalization layer (Ioffe and Szegedy, 2015). Our base model was trained for 350 epochs, with a batch-size of 128 and a learning rate 0.1. The learning rate was reduced by a factor of 10 at 150 and 250 epochs. After pruning we finetuned the pruned model with learning rate of 0.001 for 80 epochs to adjust the weights of the remaining connections to regain the accuracy.

Figure 5 shows the number of nonredundant filters per layer for different τ values. As

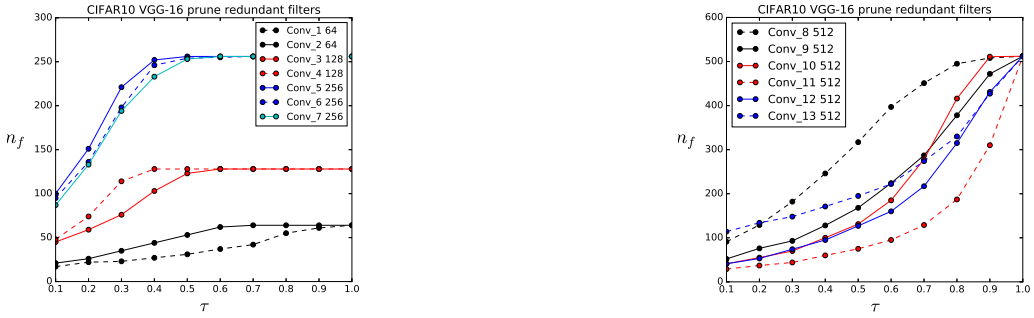
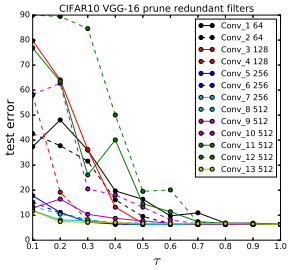


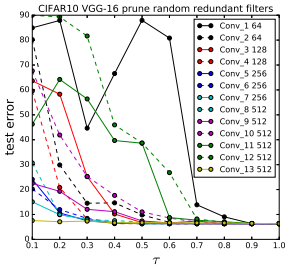
Figure 5: Number of nonredundant filters (n_f) vs. cluster similarity threshold (τ) for VGG-16 trained on the CIFAR-10 dataset. Initial number of filters for each layer is shown in the legend.

can be seen that some convolutional layers in VGG are prone to extracting features with very high correlation; examples of such as layer are layers 1, 11, 12, and 13. Another very important observation is that later layers of VGG are more susceptible to extracting redundant filters than earlier layers and can be heavily pruned. Figure 6(a) shows the sensitivity of VGG-16 layers to pruning and it can be observed that layers such as Conv 1, 3, 4, 9, 11, and 12 are very sensitive. However, as can be observed in Figure 6(c), accuracy can be restored after pruning filters in later layers (Conv 9, 11, and 12) compared to early ones (Conv 1, 3, and 4).

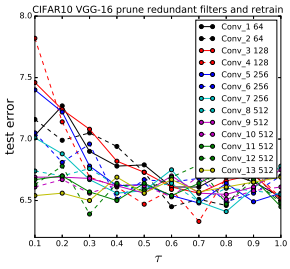
For our final test score, the pruned model is finetuned on the entire training set. In the pruning stage, we performed a grid search over τ values within 0.1 and 1.0, and found 0.54 gave the least test error. Table 1 reports the pruning performance for $\tau = 0.54$ and it can be easily



(a) Prune $n' - n_f$ redundant filters



(b) Prune $n' - n_f$ random filters



(c) Prune redundant filters and retrain

Figure 6: Sensitivity to pruning (a) redundant filters (b) random $n' - n_f$ filters, and (c) redundant filters and retraining for 30 epochs for VGG-16.

observed that more than 90% of most of the latter layer filters have been pruned and most of the sensitive earlier layers are minimally pruned. Figure 6(b) depicts the sensitivity of trained VGG-16 model to pruning using heuristic in Algorithm 3 that calculates the number of redundant filters ($n' - n_f$) and randomly prunes them.

As seen in Table 2, for $\tau = 0.54$ our approach in Algorithm 2 outperforms that Absolute

layer	$v_l \times h_l$	#Maps	FLOP	#Params	#Maps	FLOP%
Conv_1	32×32	64	1.8E+06	1.7E+03	32	50.0%
Conv_2	32×32	64	3.8E+07	3.7E+04	58	54.7%
Conv_3	16×16	128	1.9E+07	7.4E+04	125	11.5%
Conv_4	16×16	128	3.8E+07	1.5E+05	128	2.3%
Conv_5	8×8	256	1.9E+07	2.9E+05	256	0%
Conv_6	8×8	256	3.8E+07	5.9E+05	254	0.8%
Conv_7	8×8	256	3.8E+07	5.9E+05	252	2.3%
Conv_8	4×4	512	1.9E+07	1.2E+06	299	42.5%
Conv_9	4×4	512	3.8E+07	2.4E+06	164	81.3%
Conv_10	4×4	512	3.8E+07	2.4E+06	121	92.4%
Conv_11	2×2	512	9.4E+06	2.4E+06	59	97.3%
Conv_12	2×2	512	9.4E+06	2.4E+06	104	97.7%
Conv_13	2×2	512	9.4E+06	2.4E+06	129	94.9 %

Table 1: Pruning performance on CIFAR dataset using VGG-16 model at $\tau = 0.54$

filter sum approach (Li et al., 2017), Network Sliming (Liu et al., 2017), Try-and-learn (Huang et al., 2018) and are able to prune more than 78% of the parameters resulting in 40% FLOP reduction and a competitive classification accuracy. In addition, when τ was tuned to 0.46 we are able to achieve more than 65% FLOP reduction and outperform Variational method (Dai et al., 2018), which is one of the state-of-the-art. We suspect that our pruning approach outperforms other methods because it localizes and prunes similar or shifted versions of filters that do

VGG-16 Model	% Accuracy drop	% FLOP Pruned	% Parameters Pruned
Methods			
Li et al. (2017)	0.40	34.2	64.0
Liu et al. (2017)	-0.17	38.6	-
Huang et al. (2018)	0.60	34.2	-
Dai et al. (2018)	0.81	62.9	-
Ours-A ($\tau = 0.54$)	0.13	40.5	78.1
Ours-B ($\tau = 0.54$)	0.50	40.5	78.1
Ours-A ($\tau = 0.46$)	0.72	65.1	89.5

Table 2: Performance evaluation for three pruning techniques on CIFAR-10 dataset. Performance with the lowest test error is reported.

not add extra information to the feature hierarchy. This notion is reinforced from information theory standpoint that the activation of one unit should not be predictable based on the activations of other units of the same layer (Rodríguez et al., 2017). Another crucial observation is that heuristic A achieves a better accuracy than B because we suspect random pruning might remove dissimilar filters. We strongly believe that Algorithm 2 performs better than 3 because of its precise ability to remove redundancy. However, Algorithm 2 is a bit slower than 3 and that is the trade-off.

4.2. RESNET-56/110 on CIFAR-10

The architecture of residual networks is more complex than VGG and also the number of parameters in the fully connected layer is relatively smaller and this makes it a bit challenging to prune a large proportion of the parameters. Both ResNet-56 and ResNet-110 have three stages of residual blocks for feature maps of differing sizes. The size ($v_l \times h_l$) of feature maps in stages 1,2, and 3 are 32×32, 16×16, and 8×8, respectively. Each stage has 9 and 18 residual blocks for ResNet-56 and ResNet-110, respectively. A residual block consists of two convolutional layers each followed by a Batch Normalization layer. Preceding the first stage is a convolutional layer followed by a Batch Normalization layer[3]. Only the redundant filters in first convolution layer of each block are pruned since the mapping for selecting identity feature maps is unavailable.

[3]We used the Pytorch implementation of ResNet56/110 in https://github.com/D-X-Y/ResNeXt-DenseNet as baseline models

As can be observed in Figures 7 and 8 that convolutional layers in first stage are prone to

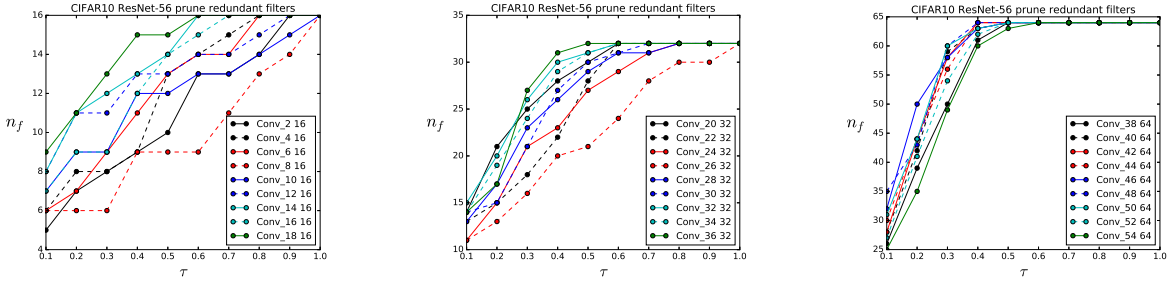


Figure 7: Number of nonredundant filters (n_f) vs. cluster similarity threshold (τ) for ResNet-56 trained on the CIFAR-10 dataset. Initial number of filters for each layer is shown in the legend.

Model	Error %	FLOP	Pruned %	# Parameters	Pruned %
ResNet-56	6.61	1.25 $\times 10^8$		8.5 $\times 10^5$	
Li et al. (2017)	6.94	9.09 $\times 10^7$	27.6%	7.3 $\times 10^5$	13.7%
Ours-A	**6.88**	**9.07 $\times 10^7$**	**27.9%**	**6.5 $\times 10^5$**	**23.7%**
Ours-B	6.94	9.07 $\times 10^7$	27.9 %	6.5 $\times 10^5$	23.7 %
ResNet-110	6.35	2.53 $\times 10^8$		1.72 $\times 10^6$	
Li et al. (2017)	**6.70**	1.55 $\times 10^8$	38.6%	1.16 $\times 10^6$	32.4%
Ours-A	6.73	**1.54 $\times 10^8$**	**39.1%**	**1.13 $\times 10^6$**	**34.2%**
Ours-B	7.41	1.54 $\times 10^8$	39.1%	1.13 $\times 10^5$	34.2%

Table 3: Performance evaluation of three pruning techniques for ResNet 56/110 trained on CIFAR-10 dataset. Performance with the lowest test error is reported.

extracting more redundant features than those of second stage, and the convolutional layers in the second stage are susceptible to extracting redundant filters than those of third block, which is contrary to the observations in experiments with VGG-16. In effect, more filters could be pruned from layers in first stage than latter stages without losing much to accuracy. More specifically, many layers in first stage of ResNet-56, such as Conv 2,8,10, and 26, have filters that are more than 80% correlated and could be easily pruned. Similarly, convolutional layers in the first stage of ResNet-110 exhibit similar tendency to produce more filters that are redundant. Due to differing redundancy tendencies at each stage, τ is customized for each of the stages. In pruning ResNet-56, we set τ to 0.253, 0.223, 0.20 as thresholds for stages 1,2, and 3, respectively. Similarly for ResNet-110 we used 0.18, 0.12, and 0.17.

Figure 9 shows the sensitivity of ResNet-56 layers to pruning and it can be observed that

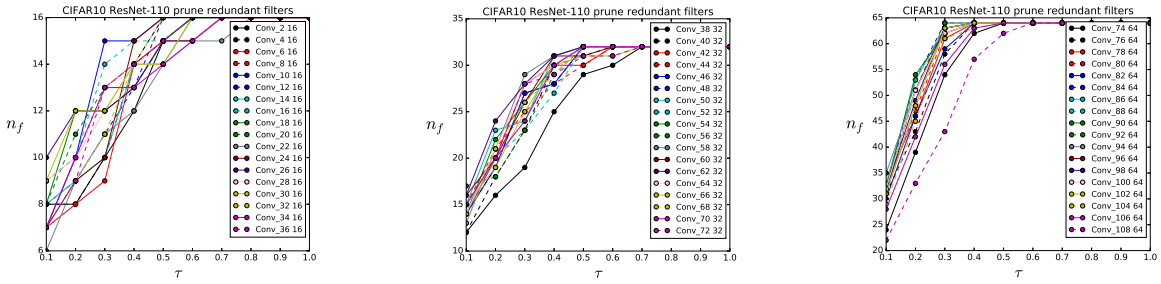


Figure 8: Number of nonredundant filters (n_f) vs. cluster similarity threshold (τ) for ResNet-110 trained on the CIFAR-10 dataset. Initial number of filters for each layer is shown in the legend.

layers such as Conv 10, 14, 16, 18, 20, 34, 36, 38, 52 and 54 are more sensitive to filter pruning than other convolutional layers. Likewise for ResNet-110, the layer sensitivity to pruning is depicted in Figure 10 and it can be observed that Conv 1, 2, 38, 78, and 108 are sensitive to pruning. In order to regain the accuracy by retraining the pruned model, we skip these sensitive layers while pruning.

As seen in Table 3 for ResNet-56, redundant-feature-based pruning methods A and B have

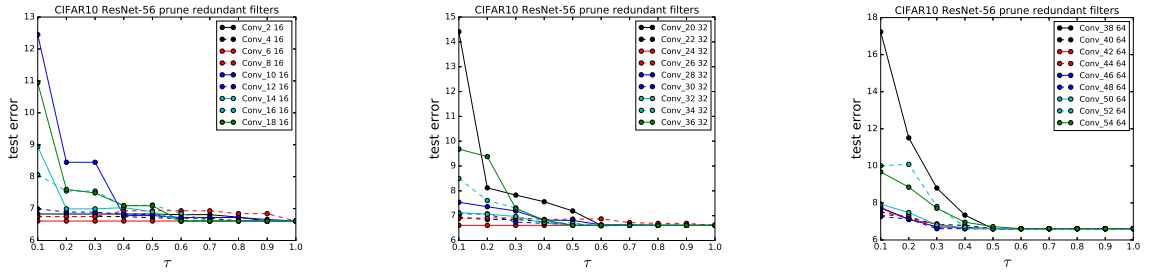


Figure 9: Sensitivity to pruning $n' - n_f$ redundant convolutional filters in ResNet-56

competitive performance in terms of FLOP reduction and outperform that in Li et al. (2017). Our approach reduces the number of effective parameters by 10% with relatively better classification accuracy after retraining. However, we marginally increase the effective number of parameters pruned in ResNet-110 from 38.6% to 39.1%, resulting in approximately 2% increase.

The inference times for original and pruned models are reported in Table 4. 10,000 test images of CIFAR-10 dataset used for the timing evaluation conducted in Pytorch version 0.2.0_3 with Titan X (Pascal) GPU and cuDNN v8.0.44, using a mini-batch of size 100. It can be observed that %FLOP reduction also translates almost directly into inference time savings.

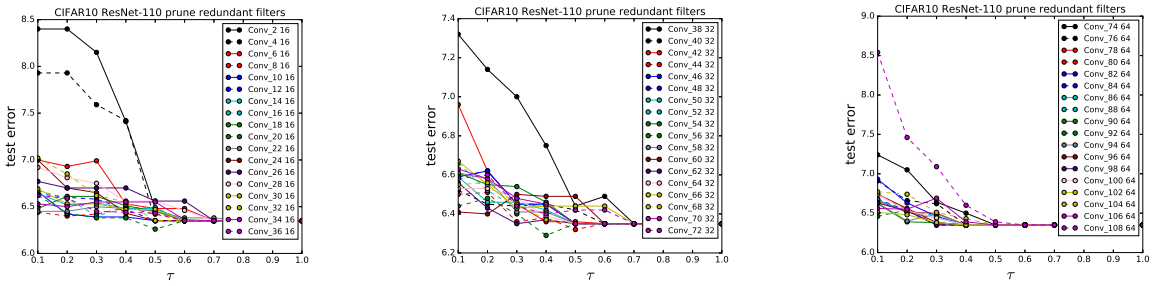


Figure 10: Sensitivity to pruning $n' - n_f$ redundant convolutional filters in ResNet-110

Model	FLOP	Pruned %	Time(s)	Saved %
VGG-16	3.13×10^8		1.47	
Ours-A	1.86×10^8	40.5%	0.94	34.01%
ResNet-56	1.25×10^8		1.16	
Ours-A	9.07×10^7	27.9%	0.96	17.2%
ResNet-110	2.53×10^8		2.22	
Ours-A	1.54×10^8	39.1%	1.80	18.9%

Table 4: FLOP and wall-clock time reduction for inference. Operations in convolutional and fully connected layer are considered for computing FLOP

4.3. ResNet-34 on ImageNet

ResNet-34 Model	% Accuracy drop	% FLOP Pruned	% Parameters Pruned
Methods			
Li et al. (2017)	1.06	24.20	10.80
Yu et al. (2018)	0.28	27.32	27.14
Ours-A ($\tau = 0.29$)	0.31	28.12	26.53

Table 5: Performance evaluation for three pruning techniques on ImageNet dataset. Performance with the lowest test error is reported.

The last set of large-scale experiments were performed on the 1000-class ImageNet 2012 dataset Russakovsky et al. (2015) which contains about 1.2 million training images, 50,000 validation images, and 100,000 test images (with no published labels). The results are measured by top-1 error rates Russakovsky et al. (2015). ImageNet dataset was used to train a residual network known as ResNet-34 He et al. (2016), which has four stages of residual blocks and uses the projection shortcut when the feature maps are down-sampled. The model was trained for 90 epochs, with a batch-size of 200 and a learning rate 0.1. It can be observed in Table 5 that our approach outperforms that in Li et al. (2017) and it is competitive with the state-of-the-art Yu

et al. (2018).

4.4. *Prune and Train from Scratch*

In order to see the effect of copying weights from the original (larger) model to a pruned (smaller) model, we pruned two models (VGG-16 and ResNet-56) as described above, re-initialized their weights, and trained them from scratch. Table 6 shows that fine-tuning a pruned model is almost always better than re-initializing and training a pruned model from scratch. We believe that already-trained filters may serve as good initialization for a smaller network which might on its own be difficult to train. Other observation from Table 6 is that redundant-feature-based pruning results in an architecture that attains a better performance than its counterpart in Li et al. (2017). This suggests that redundant-feature-based pruning has the potential to determine the architectural width of DNNs.

Model	Error %
VGG-16	
Pruned (Li et al., 2017)	6.60
Pruned (Ours-A)	6.33
Pruned-scratch-train (Li et al., 2017)	6.88
Pruned-A-scratch-train (Ours-A)	6.79
ResNet-56	
Pruned (Li et al., 2017)	6.94
Pruned (Ours-A)	6.88
Pruned-scratch-train (Li et al., 2017)	8.69
Pruned-scratch-train (Ours-A)	7.66

Table 6: Performance on CIFAR dataset

5. Conclusion

This work was motivated by the observations that widely and successfully used DNNs often have large number of overlapping features amounting to unnecessary filtering redundancy and high post-training inference cost. By grouping features at each layer according to a predefined measure in parameter space using agglomerative hierarchical clustering, we show that when redundancy can be reduced, inference cost (FLOPS) is reduced by 40% for VGG-16, 28%/39% for ResNet-56/110 trained on CIFAR-10, and 28% for ResNet-34 trained on ImageNet database

with minor loss of accuracy. To recover the accuracy after pruning, models were finetuned for a few iterations without the need to modify hyper-parameters.

Acknowledgement

This work was sponsored by the National Science Foundation under Grant 1641042.

References

Anwar, S., K. Hwang, and W. Sung (2017). Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC) 13*(3), pp. 32–42, DOI:10.1145/3005348.

Ayinde, B. O., T. Inanc, and J. M. Zurada (2019). Regularizing deep neural networks by enhancing diversity in feature extraction. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, DOI:10.1109/TNNLS.2018.2885972.

Ayinde, B. O. and J. M. Zurada (2017). Nonredundant sparse feature extraction using autoencoders with receptive fields clustering. *Neural Networks 93*, pp. 99–109.

Ba, J. and R. Caruana (2014). Do deep nets really need to be deep? In *Proc. of the Advances in Neural Information Processing Systems*, pp. 2654–2662.

Bengio, Y. and J. S. Bergstra (2009). Slow, decorrelated features for pretraining complex cell-like networks. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, pp. 99–107.

Bengio, Y., Y. LeCun, et al. (2007). Scaling learning algorithms towards ai. *Large-scale Kernel Machines 34*(5), pp. 1–41.

Bucilua, C., R. Caruana, and A. Niculescu-Mizil (2006). Model compression. In *Proc. of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 535–541. ACM.

Changpinyo, S., M. Sandler, and A. Zhmoginov (2017). The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*.

Chen, W., J. Wilson, S. Tyree, K. Weinberger, and Y. Chen (2015). Compressing neural networks with the hashing trick. In *Proc. of the International Conference on Machine Learning*, pp. 2285–2294. PMLR.

Cogswell, M., F. Ahmed, R. Girshick, L. Zitnick, and D. Batra (2016). Reducing overfitting in deep networks by decorrelating representations. In *Proc. of the International Conference on Learning Representations*, pp. 1–12.

Dai, B., C. Zhu, B. Guo, and D. Wipf (2018). Compressing neural networks using the variational information bottleneck. In *Proc. of the 35th International Conference on Machine Learning*, pp. 1135–1144. PMLR.

Denil, M., B. Shakibi, L. Dinh, N. de Freitas, et al. (2013). Predicting parameters in deep learning. In *Proc. of the Advances in Neural Information Processing Systems*, pp. 2148–2156.

Ding, C. and X. He (2002). Cluster merging and splitting in hierarchical clustering algorithms. In *Proc. of the IEEE International Conference on Data Mining*, pp. 139–146. IEEE.

Furlanello, T., Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar (2018). Born again neural networks. *arXiv preprint arXiv:1805.04770*.

Graves, A. and N. Jaitly (2014). Towards end-to-end speech recognition with recurrent neural networks. In *Proc. of the 31st International Conference on Machine Learning*, pp. 1764–1772.

Graves, A., A.-r. Mohamed, and G. Hinton (2013). Speech recognition with deep recurrent neural networks. In *Proc. of the International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649. IEEE.

Han, S., H. Mao, and W. J. Dally (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Han, S., J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, et al. (2017). Dsd: Dense-sparse-dense training for deep neural networks. In *Proc. of the International Conference on Learning Representations*, pp. 1–13.

Han, S., J. Pool, J. Tran, and W. Dally (2015). Learning both weights and connections for efficient neural network. In *Proc. of the Advances in Neural Information Processing Systems*, pp. 1135–1143.

Hassibi, B. and D. G. Stork (1993). Second order derivatives for network pruning: Optimal brain surgeon. In *Proc. of the Advances in Neural Information Processing Systems*, pp. 164–171.

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

He, Y., X. Zhang, and J. Sun (2017). Channel pruning for accelerating very deep neural networks. In *Proc. of the IEEE International Conference on Computer Vision*, pp. 1389–1397. Springer.

Hinton, G., O. Vinyals, and J. Dean (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

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

Huang, Q., K. Zhou, S. You, and U. Neumann (2018). Learning to prune filters in convolutional neural networks. In *Proc. of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 709–718. IEEE.

Ioannou, Y., D. Robertson, R. Cipolla, and A. Criminisi (2016). Deep roots: Improving cnn efficiency with hierarchical filter groups. *arXiv preprint arXiv:1605.06489*.

Ioannou, Y., D. Robertson, J. Shotton, R. Cipolla, and A. Criminisi (2016). Training cnns with low-rank filters for efficient image classification. In *Proc. of the International Conference on Learning Representations*, pp. 1–17.

Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. of the International Conference on Machine Learning*, pp. 448–456. PMLR.

Jones, E., T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python. Online accessed: 01-04-2018.

Jozefowicz, R., O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu (2016). Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Kingma, D. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A. and G. Hinton (2009). Learning multiple layers of features from tiny images. Technical report, http:/citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In *Proc. of the Advances in Neural Information Processing Systems*, pp. 1097–1105.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

LeCun, Y., J. S. Denker, and S. A. Solla (1990). Optimal brain damage. In D. S. Touretzky (Ed.), *Proc. of the Advances in Neural Information Processing Systems 2*, pp. 598–605. Morgan-Kaufmann.

Leibe, B., A. Leonardis, and B. Schiele (2004). Combined object categorization and segmentation with an implicit shape model. In *ECCV Workshop on Statistical Learning in Computer Vision*, pp. 1–16.

Li, H., A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf (2017). Pruning filters for efficient convnets. In *Proc. of the International Conference on Learning Representations*, pp. 1–12.

Liu, Z., J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang (2017). Learning efficient convolutional networks through network slimming. In *Proc. of the International Conference on Computer Vision (ICCV)*, pp. 2755–2763. IEEE.

Maaten, L. v. d. and G. Hinton (2008). Visualizing data using t-sne. *Journal of Machine Learning Research 9*(Nov), pp. 2579–2605.

Manickam, S., S. D. Roth, and T. Bushman (2000). Intelligent and optimal normalized correlation for high-speed pattern matching. *Datacube Technical Paper, link: http://locateobjects.com/vsfind-paper1.pdf*.

Mariet, Z. and S. Sra (2016). Diversity networks. In *Proc. of the International Conference on Learning Representations*, pp. 1–12.

Mathieu, M., M. Henaff, and Y. LeCun (2013). Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*.

Molchanov, P., S. Tyree, T. Karras, T. Aila, and J. Kautz (2017). Pruning convolutional neural networks for resource efficient transfer learning. In *Proc. of the International Conference on Learning Representations*, pp. 1–13.

Oord, A. v. d., S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Polyak, A. and L. Wolf (2015). Channel-level acceleration of deep face representations. *IEEE Access 3*, pp. 2163–2175.

Rodríguez, P., J. Gonzàlez, G. Cucurull, J. M. Gonfaus, and X. Roca (2016). Regularizing cnns with locally constrained decorrelations. *arXiv preprint arXiv:1611.01967*.

Rodríguez, P., J. Gonzàlez, G. Cucurull, J. M. Gonfaus, and X. Roca (2017). Regularizing cnns with locally constrained decorrelations. In *Proc. of the International Conference on Learning Representations*, pp. 1–11.

RoyChowdhury, A., P. Sharma, E. Learned-Miller, and A. Roy (2017). Reducing duplicate filters in deep neural networks. In *NIPS workshop on Deep Learning: Bridging Theory and Practice*, pp. 1–7.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla,

M. Bernstein, et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision 115*(3), pp. 211–252.

Rusu, A. A., S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*.

Shazeer, N., A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Simonyan, K. and A. Zisserman (2015). Very deep convolutional networks for large-scale image recognition. In *Proc. of the International Conference on Learning Representations*, pp. 1–11.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2016). Rethinking the inception architecture for computer vision. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826. IEEE.

Urban, G., K. J. Geras, S. E. Kahou, O. Aslan, S. Wang, R. Caruana, A. Mohamed, M. Philipose, and M. Richardson (2017). Do deep convolutional nets really need to be deep and convolutional? In *Proc. of the International Conference on Learning Representations*, pp. 1–13.

Walter, B., K. Bala, M. Kulkarni, and K. Pingali (2008). Fast agglomerative clustering for rendering. In *IEEE Symposium on Interactive Ray Tracing*, pp. 81–86. IEEE.

Yoon, J. and S. J. Hwang (2017). Combined group and exclusive sparsity for deep neural networks. In *Proc. of the International Conference on Machine Learning*, pp. 3958–3966. PMLR.

Yu, R., A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis (2018). Nisp: Pruning networks using neuron importance score propagation. In *Proc. of the Computer Vision and Pattern Recognition*, pp. 1–10. IEEE.

Zeiler, M. D. and R. Fergus (2014). Visualizing and understanding convolutional networks. In *Proc. of the European Conference on Computer Vision*, pp. 818–833. Springer.

Zhang, X., J. Zou, K. He, and J. Sun (2016). Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 38*(10), pp. 1943–1955.