Proceedings of Machine Learning Research 129:385–400, 2020 ACML 2020

FIREPruning: Learning-based Filter Pruning for Convolutional Neural Network Compression

Yuchu Fang FANGYUCHU@SMAIL.NJU.EDU.CN
Wenzhong Li LWZ@NJU.EDU.CN
Yao Zeng ZENGYAO@SMAIL.NJU.EDU.CN
Sanglu Lu SANGLU@NJU.EDU.CN
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210093, China

Editors: Sinno Jialin Pan and Masashi Sugiyama

Abstract

Despite their great success in various fields, modern convolutional neural networks (CNNs) require huge amount of computation in inference due to their deeper network structure, which prevents them from being used in resource-limited devices such as mobile phones and embedded sensors. Recently, filter pruning had been introduced as a promising model compression method to reduce computation cost and storage overhead. However, existing filter pruning approaches are mainly model-based, which rely on empirical model to evaluate the importance of filters and set parameters manually to guide model compression. In this paper, we observe that CNNs commonly consist of large amount of inactive filters, and introduce Filter Inactive RatE (FIRE), a novel metric to evaluate the importance of filters in a neural network. Based on FIRE, we develop a learning based filter pruning strategy called FIREPruning for fast model compression. It adopts a regression model to predict the FIRE value and uses a three stage pipeline (FIRE prediction, pruning, and fine-tuning) to compress the neural network efficiently. Extensive experiments based on widely-used CNN models and well-known datasets show that FIREPruning reduces overall computation cost up to 86.9% without sacrificing too much accuracy, which significantly outperforms the state-of-the-art model compression methods.

1. Introduction

Recent years have witnessed an explosive development of convolutional neural networks (CNNs). Nowadays, CNNs have been widely used in various fields such as facial recognition, object detection, autonomous vehicle, and so on. However, nearly all stat-of-the-art CNNs require huge amount of computation in inference due to their deeper network structure, which forms a barrier in their applications in light-weight devices like mobile phones and embedded sensors. Thus, there is a growing demand in community to find an efficient solution to compress deep CNNs without harming model accuracy.

Numerous of efforts had been done on neural network model compression (Han et al. (2016); Guo et al. (2016); Liu et al. (2015)). Han et al. proposed a three stage (pruning, trained quantization, and Huffman coding) parameter compression method (Han et al. (2016)) that can reduce the storage requirement of neural networks by 35× to 49× without

. The corresponding authors are Wenzhong Li and Sanglu Lu.

© 2020 Y. Fang, W. Li, Y. Zeng & S. Lu.

affecting their accuracy. Louizos et al. further proposed to learn sparse networks through ℓ_0-*norm* regularization based on stochastic gate (Louizos et al. (2018)). However, parameter compression mainly focused on removing parameters from fully connected layers to reduce storage requirement, which does not necessarily reduce the computation time since fully connected layers occupy majority number of the total parameters but only contribute less than 1% of the overall computations in modern CNNs (Li et al. (2017)).

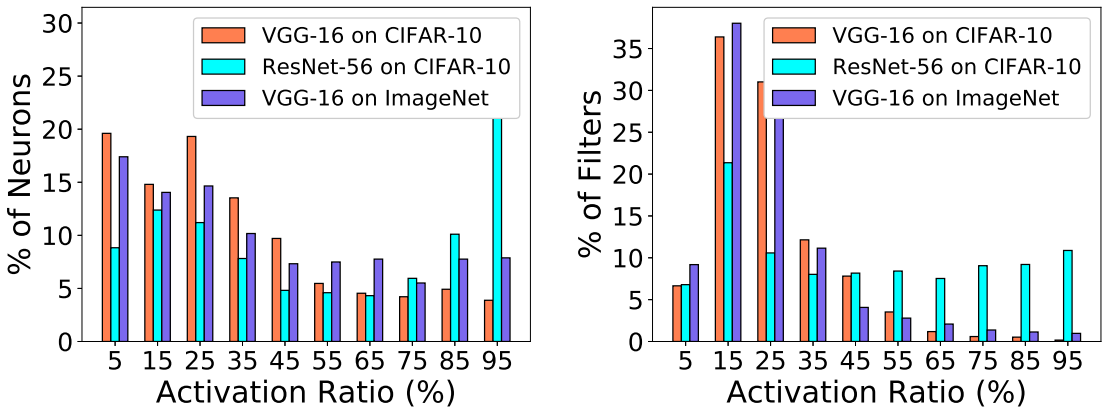
Recently, filter-wise network pruning had been introduced as a promising model compression method to reduce computation cost (Lebedev and Lempitsky (2016); Li et al. (2017)). The key idea of filter pruning is to measure the importance of filters and discard the unimportant ones. Most existing works adopted ℓ_1-*norm* and ℓ_2-*norm* to select unimportant filters (Li et al. (2017); Zhong et al. (2018); Singh et al. (2019)). However, there is lack of theoretical basis to conjecture that small norm contributes less in model inference. Besides, norm-based pruning results in the difficulty of cross-layer comparisons for filters due to their different sizes. Some works used *uniqueness* as metric to remove redundant/similar filters (Xavier et al. (2018); Wang et al. (2019)). Yet such method didn't take the effect of filters into account: a filter can be unique but has little effect on model inference.

Modern convolutional neural networks typically use Rectified Linear Unit (ReLU) as the activation function of neurons (He et al. (2016); Szegedy et al. (2015)). It can not only reduce likelihood of so-called gradient vanishing but also bring the benefit of sparsity due to its constant output of zero in the negative axis. Despite the goodness of ReLU in model training and inference, it was reported to cause the "dying ReLU problem" (He et al. (2015)). Dying ReLU problem refers to the state where neurons remain inactive for essentially all inputs. And since the derivative of ReLU is zero in negative axis, the weights of these neurons can hardly be updated afterwards, leading to stuck in a position where they can hardly output useful information.

Dying ReLU problem is very common in CNNs, and we speculate that there exist large amount of "inactive neurons" due to dying ReLU. For example, Fig. 1(a) shows our observations to the neuron activation rates of some CNN models on different datasets. It is observed that neuron activation rates are concentrated on the left part of the axis: up to 20% neurons are activated by 5% samples, up to 15% neurons are activated by 15% samples, up to 20% neurons are activated by 25% samples, and only very few neurons are activated by more than 35% samples. Such inactive neurons are rarely activated by the input, and they have very limited effect on neural networks.

In CNNs, a filter consists of a set of neurons sharing the same weights. We further observe the average neuron activation rates in filter-wise, whose results are illustrated in Fig.1(b). It is shown that most filters (up to 75%) are activated by less than 25% input samples. We call them "inactive filters", which mainly consist of inactive neurons and will produce very sparse feature maps with little useful information for CNN inference.

Inspired by the observation of inactive filters, we develop a novel measurement to guide filter pruning. Specifically, we propose *Filter Inactive RatE (FIRE)*, a novel metric to evaluate the importance of filters in a neural network. Based on FIRE, we develop a general filter pruning framework with three stage pipeline: FIRE measurement, pruning, and fine-tuning. The three stages run repeatedly to remove unimportant filters from the CNN model, until the accuracy drop exceeds a predefined tolerance threshold.



(a) Distribution of neuron activation ratio. (b) Distribution of filter activation ratio.

Figure 1: Illustration of inactive neurons and filters.

To improve the efficiency of model compression, we further propose a *learning based FIREPruning strategy* which adopts regression models to predict FIRE values without exhaustive measurements. It uses input samples to compute the FIRE value of filters in the original neural network to form a training dataset. Then, it extracts features from individual filter, and trains a *FIRE detector* to predict the FIRE value. It uses the predicted FIRE to prune the filters, and fine-tunes the CNN model round by round. We develop extensive experiments on widely-used deep CNN models (VGG and ResNet) based on a number of well-known datasets (CIFAR-10, CIFAR-100, and ImageNet). It shows that FIRE detector achieves mean absolute error (MAE) as low as 0.1, and predicts the FIRE ranking with high accuracy. The proposed FIREPruning reduces the overall floating point operations (FLOPs) by 86.9% for VGG-16 and 57.3% for ResNet on CIFAR-10 dataset with accuracy drop less than 1%. And it can reduce the FLOPs up to 62.89% with tolerable accuracy drop for ResNet-50 on ImageNet. These results significantly outperform the state-of-the-art model compression methods.

The contributions of this paper are summarized as follows:

- We propose a novel metric called FIRE to measure the importance of filters in a neural network. We empirically observed that inactive neurons/filters are very common in deep CNN models, and show that discarding inactive filters from CNN leads to great reduction in computation cost without sacrificing too much accuracy. To the best of our knowledge, we are the first one to introduce dying ReLU problem into CNN model compression to guide filter pruning.

- We propose an efficient learning based filter pruning strategy called FIREPruning for fast model compression. It extracts features from individual filter and trains a regression model to predict the its importance without exhaustive measurements, which substantially reduces the computation cost of FIRE measurement for large dataset. The idea of adopting a learning-based method to choose specific filters to prune is novel and its efficiency is verified in our work.

- We conduct extensive experiments on cross combinations of widely-used CNN models and well-known datasets. Numerical results show that FIREPruning reduces overall FLOPs computation up to 86.9% without sacrificing too much accuracy, which significantly outperforms the state-of-the-art model compression methods.

2. Related Work

We summarize the related work of CNN model compression as two categories: parameter pruning methods and filter pruning methods.

2.1. Parameter Pruning Methods

Early efforts on neural network compression mainly focused on pruning parameters from fully connected layers to reduce the model size. For example, the Optimal Brain Surgeon method (Hassibi and Stork (1993)) calculated the importance of parameters by second-order Taylor expansion, which is time-consuming. The Deep Compression method (Han et al. (2016)) regarded parameters with small absolute value to be less important and pruned the network by setting the unimportant parameters to zero to build a sparse network.

However, parameter compression methods mainly reduced the storage requirement of CNNs, and it did not necessarily reduce the computation time since the computation cost for fully connected layers only occupies a small proportion of the overall computation in modern CNNs. What's more, specially designed hardware may be needed for their implementation.

2.2. Filter Pruning Methods

Filter pruning is the most popular model compression method in the recent years. The main idea of filter-pruning is to prune "unimportant" filters in convolutional layers. Li et al. argued that filters with small norm are less important and pruned filters layer-wise iteratively with a given pruning rate (Li et al. (2017)). Luo et al. proposed to use a subset of channels in the next layer's input to approximate the original output, and remove the other channels safely (Luo et al. (2017)). Yang et al. proposed a Soft Filter Pruning method to enable the pruned filters being updated in fine-tuning stage (Yang et al. (2018)). Zhong et al. argued that the pruning order is very significant for model pruning and trained an LSTM model to generate a pruning decision (Zhong et al. (2018)). Singh et al. used an adaptive filter pruning module to minimize the number of filters and a pruning rate controller module to maximize the accuracy during pruning (Singh et al. (2019)). Wang et al. proposed correlation-based pruning method which can find redundancies among filters and prune the filters in a cross-layer manner (Wang et al. (2019)).

Different from their works, we introduce a novel metric called FIRE to measure the importance of filters in a neural network and extracts features from individual filter and trains a regression model to predict its FIRE value without exhaustive measurements. To the best of our knowledge, we are the first one to introduce dying ReLU problem into CNN model compression and train a learning-based model to choose filters to prune and guide model compression.

3. FIREPruning Approach

In this section, we introduce the FIREPruning approach in detail.

3.1. Notations and Definitions

We first introduce the notations and definitions that will be used for analysis throughout the paper.

We use $\mathcal{N}$ to denote a convolutional neural network (CNN) model. Assume it contains L convolutional layers, and the *i-th* convolutional layer in the model is denoted by L_i. We use $F_i = \{f_{(i,1)}, f_{(i,2)}, f_{(i,3)}, \cdots\}$ to represent the set of filters (i.e., convolutional kernels) in L_i, where $f_{(i,j)}$ $(j = 1, 2, \cdots)$ is the *j-th* filter in L_i. Each filter consists of a set of neurons represented by $f_{(i,j)} = \{e_{(i,j,1)}, e_{(i,j,2)}, e_{(i,j,3)}, \cdots\}$, where $e_{(i,j,k)}$ $(k = 1, 2, \cdots)$ is the *k-th* neuron of the filter.

A CNN model $\mathcal{N}$ can be represented by sets of filters $\mathcal{N} = \{F_1, F_2, \cdots, F_L\}$. Noted that a CNN model may contains other type of layers, but this paper only focuses on pruning filters in the convolutional layers.

The *CNN model compression problem* can be described as: given a CNN model $\mathcal{N} = \{F_1, F_2, \cdots, F_L\}$, we want to find an algorithm to prune filters in each layer and return a neural network $\mathcal{N}' = \{F'_1, F'_2, \cdots, F'_L\}$, where $F'_i \subset F_i$ $(i = 1, 2, \cdots, L)$, and the pruned CNN model $\mathcal{N}'$ has comparable accuracy against $\mathcal{N}$.

In a CNN model, ReLU[2] function is commonly used as the activation function of neurons. If a neuron is activated by an input sample, it will output the original value, otherwise it outputs 0. If a neuron output 0 for most input samples, it is called an inactive neuron. An inactive neuron can be the result of an aggressive update when the learning rate is too large. Yet, what makes matter worse is that once a neuron steps into this state, it can hardly be recovered. Because the derivative of ReLU is also 0 in negative axis and the gradient flow will be cut down and can't reach the inactive neuron. This means gradient descent learning will not alter the neuron's weight and the weight can hardly be updated afterwards. Since inactive neurons contribute very tiny to model inference, they can be removed from the CNN model without harming the accuracy of the model. Based on such intuition, we design a filter pruning strategy based on evaluating the inactive degree of filters.

Firstly, we formulate the Neuron Inactive Rate (NIRE) for individual neuron, which is defined as follows.

Definition 1 (Neuron Inactive Rate (NIRE)) *Given a neuron e in a deep neural network $\mathcal{N}$, a dataset X, the $NIRE$ of e is defined as the percentage of samples in X that could not activate e, which is written as*

$$NIRE_e = \frac{\sum_{\forall x \in X} g(e \circ x)}{|X|}, \tag{1}$$

where g is an indicator function defined as

$$g(e \circ x) = \begin{cases} 1, & if\ ReLU(e \circ x) = 0 \\ 0, & otherwise \end{cases} \tag{2}$$

2. https://en.wikipedia.org/wiki/Activation_function

In the above definition, x is a sample of the dataset X, and $e \circ x$ means matrix multiplication between neuron e and sample x. In convolutional neural networks, x represents the specific part of the input feature map that e is in charge of. The function $g(e \circ x)$ will be 1 if the result of $ReLU$ is 0, which means that the neuron e is not activated by the input sample x. NIRE is used to measure the inactiveness of the neuron. The larger NIRE is, the more inactive the neuron will be. If a neuron's NIRE equals to 1, then it is dead and completely useless in the neural network.

A filter consists of a set of neurons. These neurons share the same weights all over the input space. A filter will be inactive if it contains a large portion of inactive neurons. In that case, the filter will only produce a very sparse feature map which includes a lot of zeros. And it can hardly be updated due to the inactive neurons it contains. It has little effect on the forward and back propagation. Thus pruning an inactive filter will only result in little performance drop for neural network. We use Filter Inactive Rate (FIRE) to measure the inactive degree of a filter, which is defined as follows.

Definition 2 (Filter Inactive Rate (FIRE)) *Given a filter f in a deep neural network $\mathcal{N}$, a dataset X, the $FIRE$ of f is defined as the mean NIRE of all its neurons, which is computed by*

$$FIRE_f = \frac{\sum_{\forall e \in f} NIRE_e}{|f|}. \tag{3}$$

The value of FIRE ranges in [0,1]. The larger FIRE is, the more inactive the filter will be. FIRE not only measures the activeness of filters but more importantly quantifies the proportion of useless outputs in the feature map. A CNN model typically consists of a number of inactive filters as illustrated in the introduction. Pruning these filters can be beneficial to get a slimmer and computational-efficient neural network.

3.2. General FIREPruning Framework

Based on the definition of FIRE, we can design a general filter pruning framework, which is illustrated in Fig. 2. It works as follows. (1) Given a trained deep neural network model (e.g., VGG) and a dataset (e.g., CIFAR-10), it takes the dataset as input to test the model and compute the FIRE value of each filter. (2) It prunes the model by removing the top $p\%$ filters with highest FIRE values, where p is a predefined pruning ratio in the framework. (3) After pruning, it updates the deep neural network parameters by using the dataset to fine-tune the model. (4) If the accuracy of the updated model is within a predefined accuracy drop tolerance $r\%$ compared to the original model, it keeps pruning the model by repeating step (1)-(3); otherwise it outputs the updated model as the compression result.

In the proposed framework, the deep neural network model is compressed by removing a small proportion of filters ($p\%$) round by round. It does not adopt large pruning ratio for single round due to the following reason. If large amount of filters are removed, it will cause dramatic change in the structure of neural network, which makes it hard to fine-tune the pruned model to recover its performance to the original level.

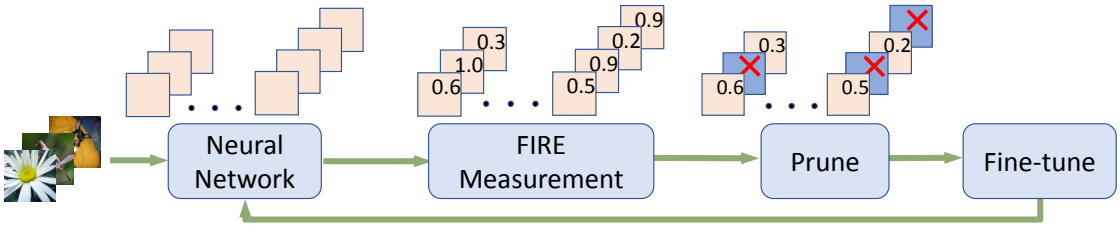


Figure 2: The general FIREPruning framework.

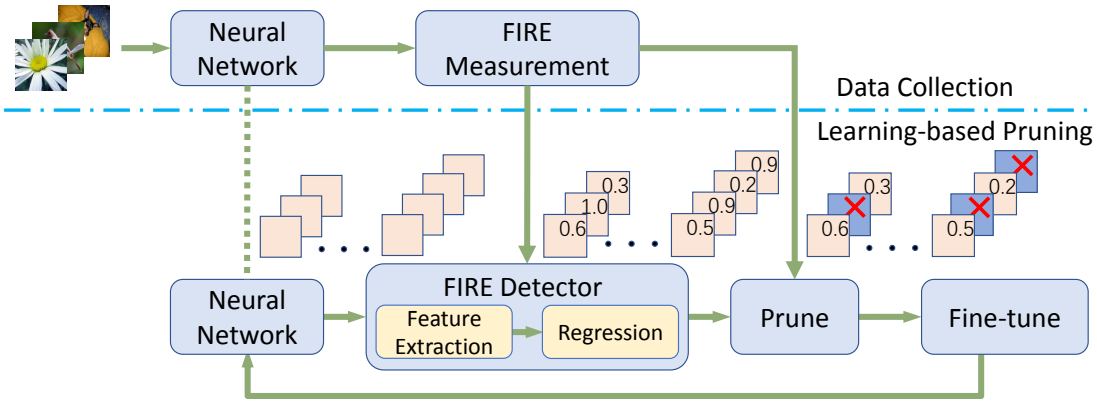


Figure 3: The learning based FIREPruning strategy.

3.3. Learning based FIREPruning Strategy

The major drawback of the general FIREPruning framework is that it needs to compute the FIRE value for each filter in every round. If the input dataset size is large (e.g., CIFAR-10 dataset contains 50000 training samples with size 32×32), the computation could be highly time-consuming. To achieve efficient model compression, we propose an improved FIREPruning strategy based on machine learning. The intuition is that FIRE values may correlated to a number of filter's features, which can be exploited to build a regression model to predict the FIRE value for individual filter without the need of input samples.

The learning based FIREPruning strategy is illustrated in Fig. 3. It also compresses the model round by round. In the first round, it works similar to the general FIREPruning framework that takes the dataset as input to compute the FIRE values of filters, which forms a training set for machine learning. Based on the training set, we extract features from individual filter, and train a *FIRE Detector* to map from features to FIRE values. Next, it runs round by round. In each round, it adopts the FIRE detector to estimate the FIRE value of each filter, removes the top $p\%$ inactive filters, and fine-tunes the model. The pruning process stops if the accuracy of the fine-tuned model exceeds the predefined accuracy drop tolerance $r\%$.

The details of learning based FIREPruning strategy are discussed as follows.

No.	Feature	Meaning
1	Channel Num	number of channels in the filter
2	Layer	layer of the filter
3	Sum	sum of filter weights
4	ℓ_1-norm	ℓ_1-norm of the filter
5-6	Max/Min	maximun and minimum of filter weights
7-8	Mean/Median	mean and median of filter weights
9	Midrange	(Max+Min) / 2
10	Range	Max - Min
11	Standard Deviation	standard deviation of filter weights
12-13	First/Third Quartile	lowest 25% and 75% of filter weights
14	Quartile Deviation	Third Quartile - First Quartile

Table 1: Extracted features for individual filter.

3.3.1. FIRE Detector

The FIRE detector extracts features from filters, and trains a regression model to predict the FIRE value of individual filter. For each filter, we extract 14 features as listed in Table 1. The extracted features are simple statistics of the raw data which are commonly used in the machine learning literature.

With the extracted features and the FIRE values computed by the input samples, we can build a regression model for FIRE prediction. Common machine learning regressors such as Linear Regressor (LR) and Support Vector Regressor (SVR) can be adopted to fulfill this task.

3.3.2. Pruning Strategy

The output of FIRE detector is used to guide the pruning process. The model is pruned round by round. In the first round, it uses the FIRE values computed from the input samples to rank the filters and form a training set for FIRE detector. In the next rounds, it uses the trained FIRE detector to predict the FIRE value of each filter. Based on the prediction, the top $p\%$ filters are removed in each round, and the parameters of the model is fine-tuned. The pruning process is repeated for several rounds, until the accuracy drop of the fine-tuned model is below the predefined accuracy drop tolerance $r\%$.

In the learning based FIREPruning strategy, neural network compression is based on the prediction from FIRE detector. Compared to the general FIREPruning framework, learning based FIREPruning does not need to use the dataset to calculate FIRE values in every round, which saves a lot of computation cost. Besides, learning based FIREPruning is more flexible and even independent of dataset: we can use random samples or synthetic input to train the FIRE detector for pruning guidance.

Since FIRE detector directly weighs the importance of filters among all layers, the algorithm can automatically discover the structure of the network and decide which filters to be pruned in each layer. A minor concern is that some layers may be pruned more aggressively than the others. With slight modification, we can manually choose the preferred

structure. For example, if we prefer the structure of neural network to be more balanced, we can set up a threshold $p_{l_i}\%$ for each layer to guarantee that at most $p_{l_i}\%$ filters are pruned in that layer.

The efficiency of learning based FIREPruning strategy is verified by empirical experiments in the following section.

4. Performance Evaluation

In this section, we conduct extensive experiments to evaluate the performance of the proposed neural network compression method.

4.1. Experimental Environment

We implement the proposed pruning strategy in PyTorch v1.0. The experiments are conducted on a GPU-equipped server (NVIDIA Tesla V-100). The default pruning ratio for each round is set to $p\% = 5\%$, and the default accuracy drop tolerance is set to $r\% = 1\%$. We use 20% of the datasets to calculate FIRE values in the first round and train the detector.

We apply the proposed compression method on two widely used deep neural networks: VGG (Simonyan and Zisserman (2015)) and ResNet (He et al. (2016)). VGG was proposed by the Visual Geometry Group at Oxford University and ranked 2nd in the classification task and 1st in the localization task of ImageNet Large Scale Visual Recognition Competition (ILSVRC) 2014. ResNet introduced a new neural network layer (the residual block) to the CNN architecture, and became the winner of ILSVRC 2015 in image classification, detection, and localization, as well as the winner of MS COCO 2015 in detection, and segmentation.

The performance are evaluated based on three datasets:

- **CIFAR-10** (Krizhevsky et al. (2009)): a dataset with 10 categories. Each category has 5000 training images and 1000 validation images. All images' resolution is 32×32.
- **CIFAR-100** (Krizhevsky et al. (2009)): a dataset with 100 categories. Each category has 500 training images and 100 validation images. All images are of size 32×32.
- **ImageNet** (Russakovsky et al. (2015)): a large-scale image classification dataset with 1.27M training images and 50K validating images in 1000 classes. All images are of size 224 × 224.

4.2. Performance of FIRE Detector

We first study the performance of FIRE detector. We implement the detector with different regression models including Multilayer Perceptron (MLP), Support Vector Regressor (SVR), Random Forest (RF), and Gradient Boosting (GB).

We use four metrics to evaluate the performance of FIRE detector: *Precision, Mean Absolute Error (MAE), and Normalized Discounted Cumulative Gain (NDCG)*, which are well-known in ranking systems (Buckley and Voorhees (2004)). The NDCG metric measures how well the predicted value preserves the top-k neurons and their ranking orders. The more NDCG close to 1, the better the prediction performance.

CNN model	Algorithm	*MAE*	*NDCG*	*Prec.*@10%	*Prec.*@40%
VGG-16	MLP	**0.0625**	0.8265	0.422	0.84
	SVR	0.0631	**0.8448**	**0.452**	0.864
	RF	0.0638	0.8435	0.43	**0.866**
	GB	0.0639	0.8372	0.412	0.832
ResNet-56	MLP	0.1831	0.8583	0.3	0.89
	SVR	0.1464	0.8932	0.43	0.95
	RF	0.1226	0.9231	**0.47**	**0.99**
	GB	**0.1078**	**0.9334**	0.45	**0.99**

Table 2: Performance of FIRE Detector on CIFAR-10 for different CNN models.

The results are listed in Table 2. According to the table, the MAE of the detector is as low as 0.1 for most cases. The NDCG of both models are higher than 0.8, some of them approaches to 0.99, which means the detector can preserve the ranking of FIRE values very well. The precision @10% and @40% show that about 45% of the predicted top 10% filters are lain in the actual top 10%, and more than 86% predicted filters are lain in the actual top 40%, which means the top filters are well predicted if the model is pruned more than 40%.

Random forest achieves the best comprehensive performance among all regression models, which is used in our experiments thereafter.

Method	FLOPs ($\times 10^7$)	Pruned FLOPs (%)	Accuracy (%)
Li-pruned (Li et al. (2017))	20.6	34	**93.4**
AFP (Ding et al. (2018))	5.83	81.39	92.87
COP (Wang et al. (2019))	8.31	73.5	93.31
RBP (Zhou et al. (2019))	8.96	71.43	91.0
FIREPruning	**4.11**	**86.90**	93.37

Table 3: Performance of different model compression methods for VGG-16 on CIFAR-10.

4.3. Comparison with State-of-the-Arts

Next, we compare the proposed approach with the state-of-the-art neural network compression methods. Similar to the literatures, we use the following metrics to evaluate the performance of different methods: (1) **Accuracy**: the accuracy of the CNN model after pruned and fine-tuned. (2) **Floating Point Operations (FLOPs)**: a widely used metric to measure the computation cost of convolutional operations (Li et al. (2017)). For a CNN, its FLOPs is calculated by $D_K \times D_K \times M \times N \times D_F \times D_F$, where M and N are number of input and output channels; $D_K \times D_K$ is kernel size; and $D_F \times D_F$ is the size of feature map. (3) **Pruned FLOPs**: the percentage of FLOPs being pruned by the algorithm.

We compare the results of our method against the reported results in other literatures. The numerical results are discussed in the following.

Method	FLOPs ($\times 10^7$)	Pruned FLOPs (%)	Accuracy (%)
Li-pruned (Li et al. (2017))	9.09	27.6	93.06
SFP (Yang et al. (2018))	5.94	52.6	93.35
AMC (He et al. (2018))	6.27	50	91.9
WtP (Zhong et al. (2018))	6.56	47.5	92.93
Rethinking (Liu et al. (2019))	6.27	50	93.05
FIREPruning	**5.36**	57.30	**93.36**

Table 4: Performance of different model compression methods for ResNet-56 on CIFAR-10.

4.3.1. VGG on CIFAR-10

VGG-16 contains 13 convolutional layers with a total number of 4224 filters. We train it from scratch following the same parameter settings and training schedule in Li et al. (2017). In the fine-tuning stage, we use SGD as the optimizer with 0.01 as initial learning rate. Learning rate will descend by a factor of 0.5 at epoches 50, 100, 150, and 250 accordingly.

The comparison of different compression methods is illustrated in Table 3. As shown in the figure, the proposed FIREPruning strategy achieves the lowest FLOPs, which is only 1/5 of the Li-pruned method. It reduces 86.90% FLOPs computation cost, which is much better than the other methods. For accuracy, FIREPruning is comparable to other methods. It shows that FIREPruning achieves highest compression rate and lowest computation cost, which significantly outperforms the other methods.

4.3.2. ResNet on CIFAR-10

ResNet-56 for CIFAR-10 has 3 stages of residual blocks, each of which contains 8 basic blocks. There are 2 convolutional layers in every basic block, and the last convolutional layer needs to produce the same number of feature maps as the input of the basic block due to the projection mapping. The whole model contains 55 convolutional layers and 2032 filters in total. We train the network from scratch following the same parameter settings and training schedule in Li et al. (2017). The learning parameters such as optimizer and learning rate are configured as same as those in the experiment of VGG on CIFAR-10.

The results of model compression are listed in Table 4. The proposed FIREPruning achieves 5.36×10^7 FLOPs with an accuracy of 93.36%. Both figures are the best among 6 methods, which means we are able to get a compact model with the best performance.

4.3.3. VGG on CIFAR-100

The CIFAR-100 dataset is similar to CIFAR-10 that has 500 training images for each class. Compared to CIFAR-10, which has 5000 images per class, CIFAR-100 is much more difficult to be used to train the model due to its fewer training data and more categories. The results of pruning VGG-16 on CIFAR-100 are compared in Table 5.

As can be seen from the table, with more categories and less training data in the dataset, the accuracy of all models is lower than that on other datasets both before and after pruning. The compression ratio is not as high as that on CIFAR-10 dataset. But compared with other

Method	FLOPs ($\times 10^8$)	Pruned FLOPs (%)	Accuracy (%)
SFP (Yang et al. (2018))	1.92	42.2	71.52
PFA (Xavier et al. (2018))	1.90	42.9	71.19
COP (Wang et al. (2019))	1.89	43.1	71.77
FIREPruning	**1.78**	**46.6**	**71.89**

Table 5: Performance of different model compression methods for VGG-16 on CIFAR-100.

pruning methods, as illustrated in Table 5, the proposed FIREPruning method is still able to get a smaller model with higher accuracy.

4.3.4. ResNet on ImageNet

We further develop experiments on a large-scale dataset ImageNet using the ResNet-50, a less redundant CNN model. We use the pretrained ResNet-50 from PyTorch which has a Top-5 accuracy of 92.86%. In the fine-tune stage, we use SGD as optimizer and fine tune the pruned model up to 60 epochs. The learning rate starting from 0.01 will be divided by 10 every 15 epochs. The results are illustrated in Table 6. It is shown that FIREPruning is able to get a compressed model with highest accuracy: 91.19%. Besides, the FLOPs of the pruned network is 1.534×10^9, which dramatically declines compared with other methods. These all confirm the efficiency of the proposed learning-based approach on real-world large-scale dataset.

Table 6: Performance of different model compression methods for ResNet-50 on ImageNet

Method	FLOPs ($\times 10^9$)	Pruned FLOPs (%)	Accuracy (%)
CP (He et al. (2017))	2.04	50.0	90.8
ThiNet-50 (Luo et al. (2017))	1.71	58.18	90.02
RRBP (Zhou et al. (2019))	1.86	54.5	91.00
GAL (Lin et al. (2019))	1.84	55.0	90.82
FIREPruning	**1.534**	**62.89**	**91.19**

4.4. Parameter Analysis

Since the per round pruning ratio $p\%$ and accuracy drop tolerance $r\%$ are two predefined system parameters, we analysis the performance of the proposed strategy with different system settings.

4.4.1. Influence of Pruning Ratio

Pruning ratio is an important system parameter that influences the efficiency of filter pruning. Unlike most prior works that set specific pruning rates for each layer, we use one global pruning ratio for model compression. The pruning ratio $p\%$ represents the aggressiveness of pruning the neural network. The larger it is, the more filters will be pruned. This may cause a huge information loss for neural network that can't be restored in fine-tuning. But a smaller pruning rate usually means more rounds of pruning as well as time budget.

Table 7 shows the performance of FIREPruning for VGG-16 on CIFAR-10 under different pruning ratio given a fixed time budget. It is shown that large pruning ratio (i.e., $p\% = 30\%$) may lead to unguaranteed accuracy drop due to dramatic change in network structure. Smaller pruning ratio can yields lower FLOPs, but it also increases the computation time for model compression. A suitable pruning ratio is 5% in our system.

Pruning Ratio (%)	2	5	10	15	20	25	30
FLOPs ($\times 10^7$)	7.52	7.09	7.10	7.13	8.84	8.39	6.08
Pruned FLOPs (%)	76.02	77.40	77.37	77.27	71.82	73.26	80.63
Accuracy (%)	93.07	93	93.04	93.12	93.57	93.67	-

Table 7: Performance of FIREPruning under different pruning ratio (VGG-16 on CIFAR-10).

4.4.2. Influence of Accuracy Drop Tolerance

A tolerance of accuracy drop $r\%$ will be given prior to pruning. The pruning procedure will stop if the model's accuracy drop after pruning-and-fine-tuning is beyond the tolerance. However, accuracy drop is nearly inevitable if we want to accelerate the network by pruning more FLOPs. To find out the relation between tolerance accuracy drop and FLOPs pruned, we conduct experiments on varying r values, whose results are illustrated in Table 8.

According to the table, smaller accuracy drop tolerance may lead to less model compression and less acceleration. Increasing the accuracy drop tolerance can reduce the FLOPs dramatically. There is a trade-off between the tolerance accuracy drop and the FLOPs reduction, and a suitable value is 1% in our system.

Tolerance (%)	0.1	0.2	0.5	1	2
FLOPs ($\times 10^7$)	28.40	25.50	15.19	4.11	3.18
Pruned FLOPs (%)	9.49	18.72	51.59	86.90	89.85
Accuracy%	94.03	93.94	93.64	93.37	92.06

Table 8: Performance of FIREPruning under different accuracy drop tolerance (VGG-16 on CIFAR-10).

5. Conclusion

Filter pruning had been introduced as a promising method to compress CNNs to reduce computation cost and storage overhead. In this paper, we introduced the dying ReLU problem into CNN model compression to guide filter pruning. Specifically, we proposed Filter Inactive Rate (FIRE), a novel indicator to measure the degree of inactive filters in a neural network. Based on FIRE, we developed a learning based filter pruning strategy for fast

model compression. It extracted features from individual filter, trained a FIRE detector to predict the FIRE value and used a three stage pipeline (FIRE prediction, pruning, and fine-tuning) to compress CNN models. The three stages run repeatedly to remove unimportant filters from the CNN model, until the accuracy drop exceeds a predefined tolerance threshold. Using FIRE to "burn" the inactive filters, our extensive experiments show that the proposed FIREPruning strategy outperforms the other state-of-the-art pruning methods and achieves up to 86.9% computation cost saving for widely-used CNN models (VGG and ResNet) on a number of well-known datasets (CIFAR-10, CIFAR-100, and ImageNet).

Acknowledgements

This work was partially supported by the National Key R&D Program of China (Grant No. 2018YFB1004704), the National Natural Science Foundation of China (Grant Nos. 61972196, 61672278, 61832008, 61832005), the Key R&D Program of Jiangsu Province, China (Grant No. BE2018116), the open Project from the State Key Laboratory of Smart Grid Protection and Operation Control "Research on Smart Integration of Terminal-Edge-Cloud Techniques for Pervasive Internet of Things", the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Sino-German Institutes of Social Computing.

References

Chris Buckley and Ellen M Voorhees. Retrieval evaluation with incomplete information. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 25–32. ACM, 2004.

Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'18)*, pages 6797—6804, 2018.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *Adavances in the neural information processing systems (NeurIPS'16)*, pages 1379–1387, 2016.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of the InInternational Conference on Learning Representations (ICLR'16)*, pages 1–14, 2016.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Adavances in the neural information processing systems (NeurIPS'93)*, 5: 164–171, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'15)*, pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778, 2016.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'17)*, pages 1389–1397, 2017.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*, pages 784–800, 2018.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical Report 4, University of Toronto, 2009.

Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 2554–2564, June 2016.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, pages 1–11, 2017.

Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR'19)*, June 2019.

Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*, pages 806–814, June 2015.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *Proceedings of the International Conference on Learning Representations (ICLR'19)*, pages 1–29, 2019.

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *Proceedings of the InInternational Conference on Learning Representations (ICLR'18)*, pages 1–13, 2018.

Jian Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'17)*, pages 5058–5066, 2017.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *Adavances in international journal of computer vision*, 115(3):211–252, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR'14)*, pages 1–14, 2015.

Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay P. Namboodiri. Play and prune: Adaptive filter pruning for deep model compression. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'19)*, pages 1–7, 2019.

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 (CVPR'15)*, pages 1–9, 2015.

Wenxiao Wang, Cong Fu, Jishun Guo, Deng Cai, and Xiaofei He. Cop: Customized deep model compression via regularized correlation-based filter-level pruning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'19)*, pages 1–7, 2019.

Suau Xavier, Zappella Luca, and Apostoloff Nicholas. Network compression using correlation analysis of layer responses. *arXiv*, 1807.10585:1–13, 2018.

He Yang, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yang Yi. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'18)*, pages 2234—-2240, 2018.

Jing Zhong, Guiguang Ding, Yuchen Guo, Jungong Han, and Bin Wang. Where to prune: Using lstm to guide end-to-end pruning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'18)*, pages 3205–3211, 2018.

Yuefu Zhou, Ya Zhang, Yanfeng Wang, and Qi Tian. Accelerate cnn via recursive bayesian pruning. In *The IEEE International Conference on Computer Vision (ICCV'19)*, pages 3306–3315, October 2019.