CMPE 544 Final Project Report
Fall 2020-2021

Mustafa Burak Otlu 2019700099
İlkay Sıkdokur 2019705069

1. Introduction

Deep neural networks are widely used for machine learning problems because of their great learning capacity. However, the goal of achieving higher classification accuracies comes with a cost of high amount of computational complexity and a complicated neural network. This complexity problem is more challenging for weak devices with limited computing and power resources. Therefore, deep neural networks have to be simplified for poor devices without deteriorating the classification performance of the network.

Pruning redundant components of the layers in a neural net is one technique to reduce the amount of computations during training or inference. [1] proposes filter pruning to make the deep neural networks thinner. In this study, the pruning procedure is applied to the trained models. The simplified models are fine-tuned with several epochs of training. The authors also proposed a group convolutional technique with shuffling to further reduce the model size. In this project, we only focused on the filter pruning method because this is the part where the great amount of the model compression is done.

2. Methodology of the Paper

The authors created the preserved filter set at the ith layer of a network if it is possible to approximate the input of the (i+2)'th layer. Then, they scaled the corresponding preserved channel set with scales determined in the filter selection part and deleted other channels at the (i+1)'th layer and deleted the corresponding filters at the ith layer. The pruning strategy is illustrated in Figure 1.

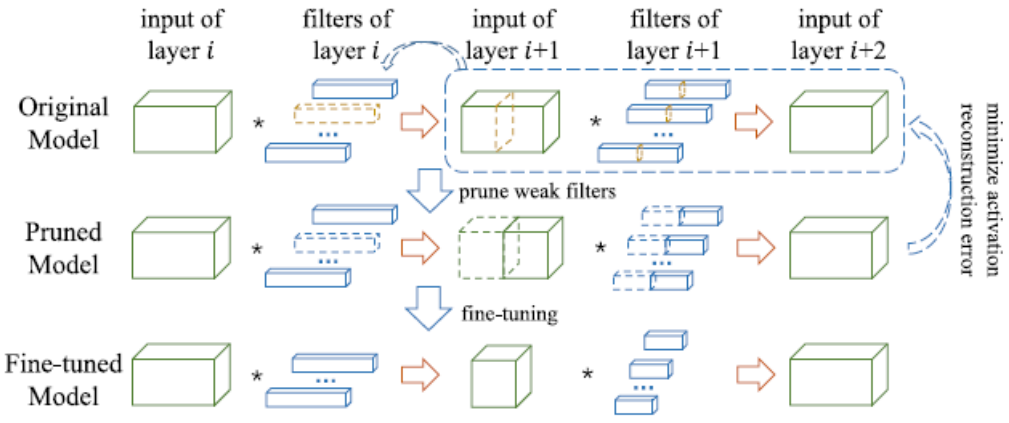

Figure 1: The filter pruning strategy in the paper[1]

The filter selection problem is converted to a linear regression problem to minimize difference between the ground truth values at the input of the (i+2)'th layer and the output at the (i+1)'th layer calculated by the preserved channels. It is another machine learning problem to determine the preserved channels and filters. Therefore, training data and labels are necessary to correctly determine the preserved filter set. The authors created a training matrix X by feeding input to the i'th layer and by taking samples from the input of the (i+1)'th layer. The dimension of X is the sample size m times number of channels at the (i+1)'th layer. The label data corresponds to the y vector that consists of the true values at the input of the (i+2)'th layer.

The authors calculated the weight values for each channel in X and calculated the loss value iteratively. The amount of preserved channels are determined by multiplying the amount of channels with the compression ratio. Then, the channels of minimum loss values with the size of preserved set length are selected. The resulting weight vector is used to scale the remaining filters at i'th layer. The filter pruning strategy is shown in Figure 2.

Algorithm 1. A Greedy Algorithm for Minimizing Eq. (8)

Input: Training set $\mathbf{X}$, $\mathbf{y}$, and compression rate r
Output: The subset of preserved channels: S and a corresponding scaling vector $\mathbf{w}$

$S \leftarrow \emptyset$; $I \leftarrow \{1, 2, \ldots, C\}$; $\hat{\mathbf{y}} \leftarrow \mathbf{y}$;
while $|S| < C \times r$ **do**
  $min\_value \leftarrow +\infty$;
  for each item $i \in I$ and $i \notin S$ **do**
    $\hat{\mathbf{w}} \leftarrow (\mathbf{X}_{:,i}^{\mathrm{T}}\mathbf{X}_{:,i})^{-1}\mathbf{X}_{:,i}^{\mathrm{T}}\hat{\mathbf{y}}$;
    $value \leftarrow \|\hat{\mathbf{y}} - \mathbf{X}_{:,i}\hat{\mathbf{w}}\|_2^2$;
    if $value < min\_value$ **then**
      $min\_value \leftarrow value$; $min\_i \leftarrow i$;
    end if
  end for
  move min_i into S;
  $\mathbf{w} \leftarrow (\mathbf{X}_{:,S}^{\mathrm{T}}\mathbf{X}_{:,S})^{-1}\mathbf{X}_{:,S}^{\mathrm{T}}\mathbf{y}$;
  $\hat{\mathbf{y}} \leftarrow \mathbf{y} - \mathbf{X}_{:,S}\mathbf{w}$;
end while

Figure 2: The filter pruning strategy in the paper[1]

3. Downsides

There are several downsides of the use of the method proposed in the paper as mentioned in the preceding section.

First of all, the proposed method in the paper requires a completely trained model for the pruning operation. The idea of the pruning is to decrease the high amount of parameters by detecting the relatively unnecessary ones and removing them from the model. One aim of this operation is to decrease computational time taken during the work of the model. However, the method proposed in the paper requires an unpruned model that is trained for many epochs. The training of this unpruned complete model takes quite a large time itself. Our method aims to prune the model while the model is in the training phase in the first place so the training time decreases in the pruning process as well.

Another downside of the method proposed in the paper is that it contains calculation of inverse matrix since it computes the least squared error to detect the filters that are to be pruned. In their method, the matrix that is to be used in inverse calculation is generated as a random sample from the input of the next convolutional layer of the layer that is to be pruned. This random sampling process may end up with a matrix that is not positive semi-definite hence not invertible. In our experiments, we dealt with this problem by adding a small amount (1e-8) to the matrix to save it from non positive semi-definiteness. However, this is not an optimal solution and not mentioned in their paper as a problem.

Final downside of their method is that it does not prune the last convolutional layer because their method requires the convolutional layer that is to be pruned has to be preceded by another convolutional layer. As the last convolutional layer is preceded by mostly a dense layer, their method does not prune the last convolutional layer. From the experiments, the last convolutional layer generally contains the most amount of parameters among the all convolutional layers. Moreover, the connections of the filters to the next dense layer also bring a vast amount of parameters in the dense layer side. These parameters could be removed as well for making the model even more thinner.

4. Our Method

Our method tackles these three downsides as making the pruning of the model on-the-go during training without calculating any inverse and pruning the last convolutional layer as well.

The filters to be pruned in a convolutional layer need to be selected based on a metric. By this metric, their relevances are sought to be detected. Since we aim to detect the relevance of the filters during the training phase, we thought that the gradients calculated during the backpropagation phase of the filters may deliver some useful information about their relevances. Gradients of the weights are calculated based on the loss that is generated by comparing the ground-truth value and predicted value that is computed by use of the weights. Therefore, if the loss value is large then the gradients to be used in weight update is large. Thus, the bigger the gradient is, the more erroneous the filter is to produce an accurate output. With this intuition, we aimed to detect the irrelevant filters with large gradient values.

Detection of the filters with this method requires the model to be mildly trained just for making a base so the filters producing greater gradients can be discriminated from the others that are considerably better fit than them. It should be noted that while applying our method, the gradients are not applied back in the backpropagation phase.

After calculating the gradients of the filters, the metric can be chosen in order to rank them as relatively relevant or irrelevant. In this work:

- We calculated the norms of the gradient matrices of each filter for every mini-batch step. These gradient matrices are of the same size of the kernels of these filters such as (3x3), (5x5) etc. After this process, there is produced an array of norms of the gradients by mini-batch, layer, filter and channel in the order.

- After then, we take the mean of the norms by channels in order to simplify the array in just one channel. This process yields an array by mini-batch, layer and filter. In the method, we do not take regarding calculated values for every mini-batch but a randomized sample of them in order to simplify the calculations.
- In the next step, each filter is scored in every layer per mini-batches. This scoring idea is built to detect outliers in the distribution of average of norm of gradients. The filters are scored per each mini-batch with +1 if the averaged values of the filters are higher than the mean of the averaged values plus the standard variation of the averaged values times a standard variation factor that is given as parameter. In the end of this process, an array by layer and filters.
- After then, the same approach is applied with the scores on the layers. If the scores of the filters are higher than the threshold calculated with the same approach, then the regarding filters are selected to be pruned from the layer.

The pseudo code of this method can be seen in Figure 3.

```
PruneModel(model, gradients, s, r, epsilon):

    for batch in gradients:
        for layer in batch:
            for filter in layer:
                avgGradBatchLayerFilter =
                mean(gradients[batch][layer][filter])


    for batch in gradients:
        for layer in batch:
            for filter in layer:
                if avgGradBatchLayerFilter[batch][layer][filter] >
                mean(avgGradBatchLayerFilter[batch][layer]) + s *
                std(avgGradBatchLayerFilter[batch][layer]):
                    scores[layer][filter] += 1

    for layer in scores:
        for filter in layer:
            if scores[layer][filter] > mean(scores[layer]) + s *
            std(scores[layer]) and r >= [remaining parameters] / [initial
            parameters]:
                prune_filters[layer].add(filter)

    for layer in scores:
        for filter in layer:
            if filter in prune_filters:
                model[layer].delete(filter)
            if filter not in prune_filters:
                model[layer][filter] += epsilon

    return model
```

Figure 3: Pseudo algorithm for pruning the model during training

5. Experiments

After implementing the pruning strategy of the paper and our pruning technique, we conducted 5 tests with different compression ratios. We used the Cifar-10 dataset and selected the base model as VGG-16. We fixed the total amount of epochs to 20. We created two identical base models. One of the models is pruned according to our method during training. After each 5 epochs we pruned the network until the model is compressed with the given ratio. On the other hand, the other base model is firstly trained for 15 epochs. Afterwards, the trained model is pruned according to the method of [1]. The remaining 5 epochs are used for fine-tuning. The test accuracy results after applying both methods are presented in Table 1.

Table 1: Test accuracy results for different compression ratios after filter pruning with our approach and [1]

Compression Rate (r)	Test Accuracy	
	Our Approach	[1]
0.8	0.7636	0.7569
0.6	0.7452	0.7292
0.4	0.6275	0.6097
0.2	0.6266	0.5871

6. Conclusion

Filter pruning is a good technique to simplify complex neural nets. Although filter selection on the trained models is safe and may not deteriorate the performance of the network dramatically, filter pruning during training is also possible and efficient. In this project, we tried to tackle the downsides of [1] and applied filter pruning during training not after training. Since we do not have any ground truth labels during training, we created a metric from the gradients at the backpropagation to get information about the importance of the filters. After a set of experiments , we could achieve better test accuracies compared to the pruning approach in [1].

REFERENCES

[1] Luo, J. H., Zhang, H., Zhou, H. Y., Xie, C. W., Wu, J., & Lin, W. (2018). Thinet: pruning cnn filters for a thinner net. *IEEE transactions on pattern analysis and machine intelligence*, *41*(10), 2525-2538.