Using Neural networks for Convolutional Neural network compression

Prathmesh Kale[1] and Kaushik Paranjape[2]

Abstract— **Convolutional Neural networks are deep neural networks which are complex than the simple multi layer perceptrons and help in achieving a far better accuracy for image recognition. The problem with CNN is that the overhead for such networks is huge which increases the execution time plus human intervention for compression**

In this paper the primary focus is to compress the Convolutional Neural network. We calculate the norm values of the filters in the first Convolutional layer and we remove the filters which have less impact on the accuracy.For the removal process we select these certain filters by plotting the norm values and simply remove those filters, with which the accuracy is not severely affected.We decrease the "overhead significantly reduce the human intervention" by removing these filters.

I. INTRODUCTION

The current state-of-the-art CNN models are characterized by a large number of weights that consumes considerable storage and memory resources which makes it difficult to deploy on small scale systems. To overcome these limitations the current solution is to prune the network connections whose corresponding weights are below some threshold. The main limitation of this part is due to the fact that, to identify the appropriate threshold parameter value, this approach has to re-iterate steps the same process many times, wasting a lot of computational resources. Moreover, since the threshold and the network weights are not jointly optimized during the training phase, this can produce a sub-optimal solution not able to achieve the maximum compression rate. Our proposal is to remove these filters runtime and make this process automated using a neural network which does this classification and removal of filters on the basis of norm values or any other mathematical calculations if required. The neural network will decide the best possible combination of filters to be removed and reaches a better trade-off between speed and accuracy.

II. CONVOLUTIONAL LAYERS AND NEURAL NETWORK COMPONENTS

These are the layers right next to the input layer, the convolutional layer is fed from the input layer and eventually pass the output to the next layer. This series of output is the result of one neuron. To classify images a very high number of neurons would be necessary because the input size for a CNN is very large when it comes to images.

The parameters provided to the convolutional layer consist of a set of learnable filters, these have small receptive fields but extended through the full depth of input volume. When the output is passed forward to the next layer, each filter is convolved across the width and height of input volume, where the dot product of filter entries and the input eventually producing a 2-dimensional map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

A. Pooling layers

Pooling layers are introduced in this structure because there are multiple outputs from the first convolutional layer and such set of multiple outputs can be sent to the next layer directly as input. Where the concept of pooling comes into picture, which combines the outputs of neuron clusters at one layer into single neuron as the input for the next layer. There are generally two pooling concepts:

- Max Pooling In Max pooling, the maximum value from each of a cluster of neurons in that specific layer or quadrant.
- Average Pooling In Average pooling the average value from each cluster of neurons in that specific layer or quadrant.

B. Fully connected layer

Fully connected layers are the layer of connection of every neuron in one layer to every neuron in another layer. (The same as multi-perceptron layer used in a traditional neural network)

C. Normalization layers

The normalization layers are incorporated in the design to implement inhibition schemes observed in the biological brain. Many CNN designs do not involve the normalization layer as the contribution is minimal in the prediction.

D. Weights

As mentioned in the convolutional layer, the CNNs are comprised of multi layer neurons, which share weights in these layers.

III. COMPARISON OF COMPRESSION METHODS

This literature review focuses towards compression of these CNNs where there is large number of data samples and multiple layers, which in turn increases the overhead of computation and the execution time for the neural network.

As demonstrated in the paper of Deep Compression by Han et al where using efficient pruning methods and quantization of the networks results in reducing the overhead for the VGG-16 by eliminating the unnecessary connections or filters and most importantly with out a loss in the accuracy. The test dataset used by Han et al was the ImageNet dataset and proved that the size of VGG16 can be reduced by a factor of 49.

In another source, the author has demonstrated compressing the MobileNet model which is 32 times smaller than VGG16. In compression, the trade off is network size vs accuracy.

The methods demonstrated by both the authors in compression of a neural network are straightforward

A. *Method 1 (Pruning using the L-1 relevance values)*

The overhead or the computation can be reduced by removal of output channels or removing unnecessary filters. So that the neural network does not need to go through all of the channels.

The next step to focus is the removal of certain filters so that the accuracy is not affected significantly but the overhead is reduced potentially.

The selection of the filters to be removed is based on a few metrics. One of them is the simple filter relevance by the L-1 norm of the filters weights. The L-1 norm value of the filters is based on the filters weights which make the filter significant.

Simply, the lower the L-1 norm value of the filter, the less relevant the filter is. The author proves that the model had multiple layers having filters with low L-1 norm values. Removing filters from a layer is definitely going to affect the next layer because they layers are interconnected and the output becomes smaller and smaller. As a result there comes a need to remove the corresponding input channels from that specific layer.

The concern in this removal process is retraining, removing filters means removing the segments of what the neural network has learned.

The author explains that compressing the convolutional layers by making the weights of the connections zero and by removing the filters physically are two different processes and will yield different results.

After the sequential pruning of multiple layers and the filters within them by calculating the L-1 norm values the conclusion of the experiment was that the network did become 25 percent smaller but the accuracy decreased but not significantly. The author ,in the end, claims that the L-1 norm method for pruning might not be the best method to prune the network as a substantial learned part of the network is removed which affects the network in some or the other way.

B. *Method 2(Trained Quantization and Weight Sharing)*

The Deep CNN compression paper by Han et al, where the author has demonstrated the compression technique by focusing on the weight sharing. The author compresses the network in a two stage process that is:

- Network Pruning:
  The connections with weights below a threshold value are removed from the network. Again the network is retrained to understand the final weights fro the remaining sparse connections. This method is the state of the art method for pruning as demonstrated in the paper by LeCun et al which is a bit analogous to the method 1 described above.
  The next step in compression comes after the network is pruned
- Weight sharing and trained Quantization:
  Network quantization and weight sharing compress the network even furthermore by reducing the number of bits in the weight. The number of effective weights is limited as there is a need to store multiple connections which have the same weight.
  The concept of weight sharing and quantization can be explained by this following diagram:
  Weight sharing is illustrated in the above figure. Suppose we have a layer that has 4 input neurons and 4 output neurons, the weight is a 44 matrix. On the top left is the 44 weight matrix, and on the bottom left is the 44 gradient matrix. The weights are quantized to 4 bins (denoted with 4 colors), all the weights in the same bin share the same value, thus for each weight, we then need to store only a small index into a table of shared weights. During the update, all the gradients are grouped by the color and summed together, multiplied by the learning rate and subtracted from the shared centroids from last iteration.
  The author has successfully compressed the network significantly and that too with out a loss in the accuracy. To calculate the compression rate, given k clusters, we only need log2(k) bits to encode the index. In general, for a network with n connections and each connection is represented with b bits, constraining the connections to have only k shared weights will result in a compression rate of:
  The author uses k-means clustering to identify the shared eights for each layer of a trained network so that the weights with near by same values will fall into the same cluster sharing the same weights. Weights are not shared across layers. The partition of n original weights and k clusters so as to minimize the within-cluster sum of the square by the formula:

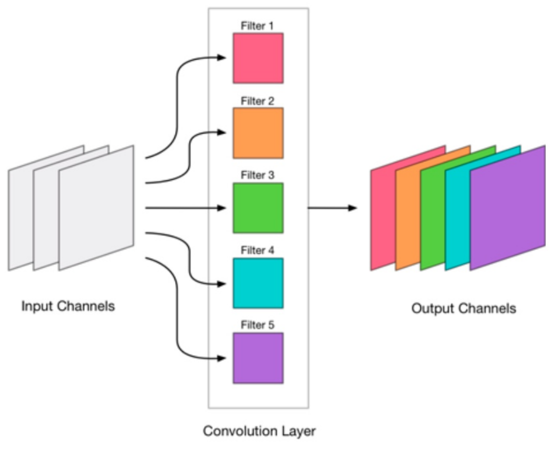


Fig. 1: L1 norm pruning

The different methods determined by the author for initialization of shared weights are:
Forgy (random) initialization randomly chooses k observations from the data set and uses these as the initial centroids. The initialized centroids are shown in yellow. Since there are two peaks in the bimodal distribution, Forgy method tends to concentrate around those two peaks.
Density-based initialization linearly spaces the CDF of the weights in the y-axis, then finds the horizontal intersection with the CDF, and finally finds the vertical intersection on the x-axis, which becomes a centroid, as shown in blue dots. This method makes the centroids denser around the two peaks, but more scatted than the Forgy method.
Linear initialization linearly spaces the centroids between the [min, max] of the original weights. This initialization method is invariant to the distribution of the weights and is the most scattered compared with the former two methods.

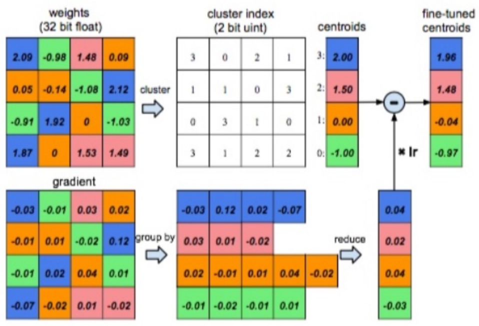


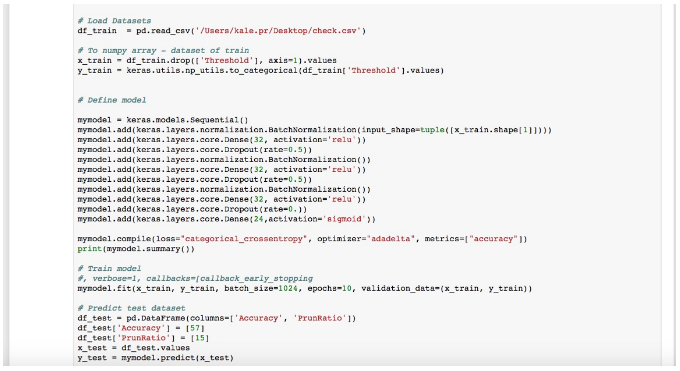
Fig. 2: Weight Quantization pruning

IV. Compressing using a experimental neural network

Our approach to solving this compression related issue by ruling out the manual intervention of pruning and automate the pruning process using a simple MLP network, which gives the optimum threshold values for pruning the network again.

For experimental purposes we remove a few of the filters (output channels) from the Convolutional layers. Naturally this makes the accuracy of the neural net worse, so we retrain for a few epochs to compensate. In the interest of saving time, we only train on a small sample (several thousand images) instead of on the entire ImageNet data set (1.28 million images). After pruning the network we plan on writing our own simple neural network to make the pruning process automated for one layer pruning, Our network will predict the optimum threshold value for pruning which indirectly gives maximum accuracy possible and minimum execution/testing time possible.

We start by compressing the layer with maximum number of parameters available for pruning and retrain the network based on the pruning, we pass the training accuracy to the simple MLP and again prune the original network's layer based on a different value given by the MLP. A comparison between the accuracy and testing time of before pruning,after manual pruning and automated pruning is provided in the notebook. You can find the full code and the results in this notebook.



```
# Load Datasets
df_train  = pd.read_csv('/Users/kale.pr/Desktop/check.csv')

# To numpy array - dataset of train
x_train = df_train.drop(['Threshold'], axis=1).values
y_train = keras.utils.np_utils.to_categorical(df_train['Threshold'].values)


# Define model

mymodel = keras.models.Sequential()
mymodel.add(keras.layers.normalization.BatchNormalization(input_shape=tuple([x_train.shape[1]])))
mymodel.add(keras.layers.core.Dense(32, activation='relu'))
mymodel.add(keras.layers.core.Dropout(rate=0.5))
mymodel.add(keras.layers.normalization.BatchNormalization())
mymodel.add(keras.layers.core.Dense(32, activation='relu'))
mymodel.add(keras.layers.core.Dropout(rate=0.5))
mymodel.add(keras.layers.normalization.BatchNormalization())
mymodel.add(keras.layers.core.Dense(32, activation='relu'))
mymodel.add(keras.layers.core.Dropout(rate=0.))
mymodel.add(keras.layers.core.Dense(24,activation='sigmoid'))

mymodel.compile(loss="categorical_crossentropy", optimizer="adadelta", metrics=["accuracy"])
print(mymodel.summary())

# Train model
#, verbose=1, callbacks=[callback_early_stopping
mymodel.fit(x_train, y_train, batch_size=1024, epochs=10, validation_data=(x_train, y_train))

# Predict test dataset
df_test = pd.DataFrame(columns=['Accuracy', 'PrunRatio'])
df_test['Accuracy'] = [57]
df_test['PrunRatio'] = [15]
x_test = df_test.values
y_test = mymodel.predict(x_test)
```

Fig. 3: MLP to predict optimum number of filters to be removed

V. Results

Pros:

We have achieved the same execution time of automated pruning and manual pruning which means there is no significant overhead of the network which was expected. The saved model is less in size as compared to the previous model

cons:

We faced a accuracy drop of 5 percent for one layer automated pruning and after retraining on complete dataset and training images we predict this accuracy loss to be somewhere between 1-3 percent We didn't consider a few of important factors such as batch size, pruning ratio, number of filters which would save the model's accuracy to a much better extent if considered in the MLP.

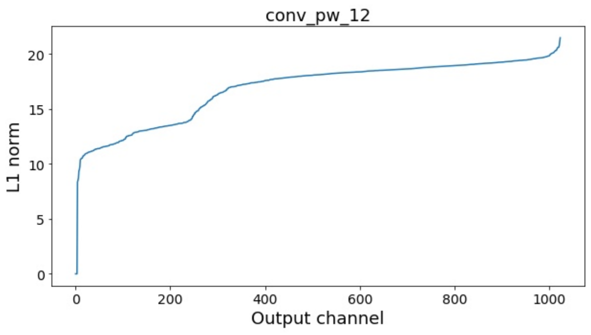


Fig. 4: pruned layer of the Convolutional neural network

VI. Conclusions

There are very few methods described for compressing the Deep CNNs. The above mentioned methods are both efficient in pruning a network and have their own vices and virtues. The first method is relatively simple in comparison to the second one but it does not reduce the network at its best in turn not providing the best accuracy. There is also a risk of losing the essential filters within the network which might affect the learning capability of the network. Whereas, the second method is efficient compared to the first one. But involves lots of mathematical calculations and pruning steps to go through which also might affect the execution time of the network but significantly reduces overhead as the pruning is a two stage process in this method. If the network is not big enough and the accuracy can face a nominal downfall then the first method for pruning fits the best. Similarly, every part of the network is important and has to be removed with care with achieving maximum accuracy possible, the second method of pruning is more likely to provide best results.

ACKNOWLEDGMENT

We sincerely thank Mr. Hollemans for providing a deeper insight to their compression and pruning code from which we have taken our inspiration of automated compression techniques. The code for our main frame of reference was :

https://gist.github.com/hollance/d15c0
cc6004a5479c00ac26bce61ac8d

http://machinethink.net/blog/compressing-deep-neural-nets/

The blog that explains the code and is built upon the above reference is:

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

REFERENCES

[1] Ji Lin, Yongming Rao, Jiwen Lu, Jie Zhou, Runtime Neural Pruning.31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 2017.

[2] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149, 2015.

[3] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In NIPS, pages 11351143, 2015.

[4] Stephen Jose Hanson and Lorien Y Pratt. Comparing biases for minimal network construction with back-propagation. In NIPS, pages 177185, 1989.

[5] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In NIPS, pages 598 605, 1990

[6] Nikko Strom. Phoneme probability estimation with dynamic sparsely connected artificial neural networks. The Free Speech Journal, 5:141, 1997.

[7] Franco Manessi, Alessandro Rozza, Simone Bianco, Paolo Napoletano, Raimondo Schettini, Automated Pruning for Deep Neural Network Compression, 2017

[8] Pavlo Molchanov,Stephen Tyree,Tero Karras,Timo Aila,Jan Kautz, PruningConvolutional Neural Networks For Resource Efficient Inference, 2017

[9] Jacob Gildenblat's, https://jacobgil.github.io/deeplearning/pruning-deep-learning

[10] Matthijs Hollemans. http://machinethink.net/blog/compressing-deep-neural-nets/, 2017.