Optimization of Convolutional Neural Networks via Distinctiveness Pruning

Siddharth Thakur
U6395848@anu.edu.au

Abstract –

Handwritten Digit Recognition has widely used to analyse various Deep learning models. Convolution Neural network is used to evaluate its performance which consist of 2 convolutional layers, 2 kernels, 2 pooling layers and 10 output label classes which are implemented using torch.nn, torch.Tensor, and tested with the help of MNIST dataset. In this paper, I compare the result of distinctiveness pruned convolutional neural network with the Handwritten Digit Recognition Using Deep Learning paper [1]. Using this I got 97% of accuracy for the pruned network in the comparison of 98.72%.

Keyword: Convolutional Neural Network, Distinctiveness Pruning, kernels, Full Connected layers, MNIST dataset.

1. Introduction

1.1. Motivation

Deep Neural Networks (DNN) have outperformed many shallow algorithms such as computer vision, automatic speech recognition etc. It has been established that strength of DNN comes from large parameter space and hierarchical structure. Hence reducing the complications in the DNN is going to be so beneficial when compare to computing cost and time [2]. The dataset is combination of huge number of examples in both test and training set and it is ideal dataset for the DNN which I can use to reduce the network size using pruning.

1.2. Dataset

I have used dataset from MNIST (Mixed National Institute of Standards and Technology) database. It is the database of hand written digits which are composed of training sets of 60000 examples and a test set of 10000 examples. It is the subset of the NSIT which is much larger database. These digits are size- normalized and centred in the fixed size image. First of 5000 examples of the test set in the MNIST database are taken from NIST training set and remaining is taken from NIST test set. More Importantly, first 5000 examples of test sets are better and more visible however last 5000 examples are not very clear. Both training set and test set consist of label values from 0 to 9. Their image files have organised pixel in row-wise with pixel value ranging from 0 to 255. The pixel value 0 represents white background and 255 represents black background. Each pixel is converted into number between 0 (black) and 1(white). It is saved in the vector and label of each image is saved in one hot encoding. [3] [4]

1.3. Problem Description

I implemented the convolution neural network (CNN) to classify the handwritten digits taken from the MNIST dataset and then I implemented the network pruning to optimize the CNN.

1.3.1. Investigation on the CNN Model

CNN is a feed forward Artificial Neural Network where the connectivity between neurons are inspired by visual cortex. They have learnable weights and biases. Each neuron receives inputs, execute dot product and follows it with non-linearity. The network is arranged in the form of width, height, and depth. It expresses a differentiable score function that is further allowed by ReLU function. [1]

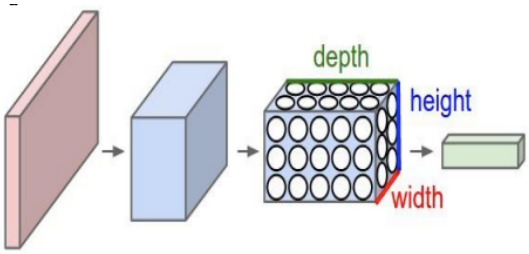

Figure 1: Layout of CNN [1]

5 layer that are connected in CNN are:

1.3.1.1. Input Layer

It includes the input which is pixel value in the case for our dataset. [1]

1.3.1.2. Convolutional Layer

This layer get the result from neurons that is connected to input layers and then each filter slides over the input matrix and get the max intensity pixel in the output. [1] This could be multiple layers.

1.3.1.3. Rectified Linear Unit Layer

It applies an element wise activation function on the image data. It is important that values are not changed by back propagation hence we apply Softmax function. [1]

1.3.1.4. Pooling Layer

It performs the down sampling operation to reduce the spatial dimensions. It helps to reduce the parameters, computation in the network and also control the overfitting of the network. [1] This is also could be multiple layers.

1.3.1.5. Fully Connected Layer

It is very similar to Artificial Neural Networks (ANN) when all the neurons are connected with each other i.e. all the activation of previous layers is connected in neural networks. [1]

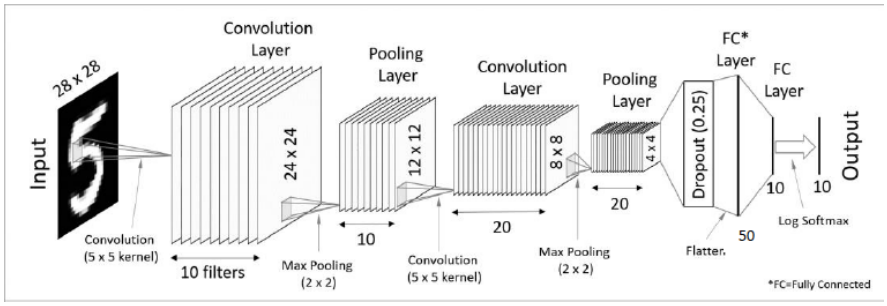

Figure 2: Experimental view of CNN

1.3.2. Investigation on Distinctiveness Pruning Measure

Deep Neural Networks are used in computer vision and speech recognition however, these algorithms are computationally expensive and use lots of memory. Therefore, it reduces the usage where there is hardware limitation exist. This is where pruning is used because it has ability to reduce the computation and the memory without reducing the accuracy further [6]. Aim of the pruning is to select the parameters for deletion and improve training and generalization. Pruning is achieved by particle filtering where configurations are weighted by misclassification rate. It actually drops the parameters with values below the threshold. It is effective for network compression and provide good performance in the intra-kernel pruning. It has used the distinctiveness [6] pruning measures to ensure optimal reduction of hidden neurons from fully connected layer.

2. Method

2.1. CNN Implementation

2.1.1. Input

The input is the MNIST data which is going to tested in 10000 examples which are 784 d-array of pixels. Hence the pre-processed inputs would be 28X28 matrix of pixels consist of 0s and 1s.

2.1.2. Network Architecture

CNN involves convolution layer, reLU functions and pooling layer. I have used 2 convolutional layers. First convolutional layer has 10 filters with dimensions of 24X24 and the second convolutional layer has 20 filters with the dimensions of 8X8 matrix size. I have used two kernel filters for two convolutional filter with the sliding window size of 5X5. Since CNN uses back propagation, ReLU reduces the probability of vanishing gradient and sparsity. It ensures that we don't lose important data and removes the duplication of data having pixel values of 0. [1] I have trained my network for 5 epochs for a batch size of 6400 training examples. Hyper parameter-learning rate is kept low (0.01) to make non-negative progress on the loss function [5]. Another hypermeter- momentum is used to build the parameter vectors in the direction of consistent gradient and also to achieve better converge rate [7].

2.1.3. Pooling Layers

It extracts the pixels from ReLU function and down sample it. I have used two such layers with 12X12 and 4X4. It actually pools the pixel of the image and form a new image with new matrix of the mentioned dimensions. The same functionality works in the second filter which actually reduces image further small size. I have used max pooling which will extract the maximum value in the matrix of 12X12 in the first pooling layer and same in the matrix of 4X4 in the second pooling layer [5]. Pooling layers' help make the representation more invariant, it helps in covering larger part of inputs by reducing the spatial dimensionality and it makes the optimization easier [6].

2.1.4. Full Connected Layers

There are 320 and 50 hidden neurons in fully connected convolutional layers with 320X50 connections and it is connected to Softmax classifier that returns the probability of 10 class labels of 10 handwritten images. Softmax classifier is generalization of binary form in logistic regression. It maps the output class label with simple dot product of weight and data [1].

2.2. Distinctiveness Pruning Implementation

2.2.1. Distinctiveness Pruning

The process of network reduction is taken from the Network Reduction Technique paper [8]. This has been taken from my previous research paper. I have used distinctive measure to verify how much the hidden neurons of fully connected layers are similar or different. It is tricky to analyse the minimum number of hidden neuron for CNN to work and provide similar amount of accuracy. It also important to rule of thumb in the DNN when network fails to learn to recognise. This means we can always add neurons till the point network generates the best accuracy and after that I can eliminate the neurons till the point accuracy is not affected dramatically [8]. This increase the efficiency of the network and less computation cost with minimum size of network [8].

2.2.2. Parameter Selection

I took the vectors of the neurons in the fully connected layers. I then evaluated their angles between them and compare every hidden neuron in the fully connected layer. I used Cos inverse function to evaluate the angle. There is a bit difference from the previous pruning implementation. Here, I used to eliminate the neurons for the angle greater than 130 and smaller than 55. After doing many experimental iteration, I observed that most of the angles lies between 60º and 120º. Hence, I decided to eliminate the neurons which are beyond this range because any neuron's vector angle less than 60º are similar thereby, I added the weight to the neuron from which it was compared and beyond 120º are different. Therefore, I able to eliminate 10 hidden neurons and also, I ran pruning process at the 3rd epoch training only because I wanted my network to learn something before I reduce the network size.

3. Results and Discussion

The loss function on training set and accuracy on the test set of the MNIST data is calculated. The accuracy is drawn for the test set by running through number of epoch. So, at this point I have total of 370 hidden neurons including both convolutional layers and out of that 6 neurons are pruned at the 1st epoch only and I was computing the accuracy with 364 neurons for 10000 test set and I got 97% accuracy which is remarkable when compare to 500 hidden neurons used in the paper [1] to achieve the accuracy of 98.72%. Figure 3, 4 and 5 are result summary when network is pruned at 1st epoch. I ran 50 epochs to achieve the result however, it depends on how much accuracy you want to receive and we can adjust learning rate, no of epochs, momentum and even number of hidden neurons.

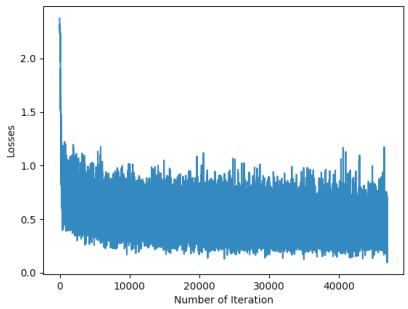

Figure 3: Loss Function

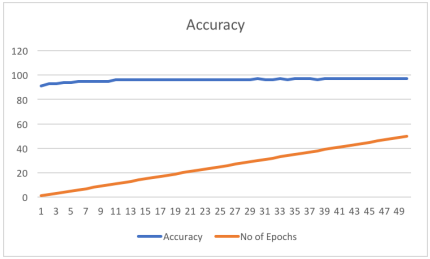

Figure 4: Test Accuracy

```
Test set: Average loss: 0.1175, Accuracy: 9666/10000 (97%)

Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5)
  (fc1): Linear(in_features=320, out_features=44, bias=True)
  (fc2): Linear(in_features=44, out_features=10, bias=True)
)
```

Figure 5: Output after Pruning at 3rd epoch

I ran some several tests when I pruned network at 30th epoch where I only left with 321 hidden neurons which further reduced the accuracy to 28% and this is not ideal. Figure 8 and 9 represent the number of hidden neurons left after pruning and loss values respectively. and then I decided to prune the network at 3rd epoch which reduces the hidden neurons to 325 and I recorded the accuracy of 88%. Figure 6 and 7 represent the number of hidden neurons left after pruning and loss values respectively. I observed that pruning does reduce the computation complexity however it also reduces the accuracy. This is because that I did one shot pruning. This is not iterative pruning and only done once. Hence when pruning is done at early stages when network already consist more than 90% of neurons, convergence (weights of neurons get merged) time decreases and accuracy increases and similarly, when pruning is done on later stages then convergence time increases and accuracy decreases [7]. This is shown in above result. After that the network performance regresses. It is for the best interest how much pruning we need to do to get the optimum result. Even after pruning, the accuracy of test set reduces when compare to original network. Hence pruning should be done at optimum level.

```
Test set: Average loss: 0.8391, Accuracy: 8807/10000 (88%)

Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5)
  (fc1): Linear(in_features=320, out_features=5, bias=True)
  (fc2): Linear(in_features=5, out_features=10, bias=True)
)
```

Figure 6: Output after Pruning at 3rd epoch

```
Test set: Average loss: 1.8915, Accuracy: 2803/10000 (28%)

Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5)
  (fc1): Linear(in_features=320, out_features=1, bias=True)
  (fc2): Linear(in_features=1, out_features=10, bias=True)
)
```

Figure 7: Output after pruning at 30th epoch

4. Conclusion and Future Work

I applied CNN on MNSIT data to recognise the handwritten digits. CNN model has 2 convolutional layers, 2 pooling layers operated with max pooling, 2 kernels and 10 output classes. I ran several iterations after applying distinctiveness pruning for the better result and then compared my result with the other research paper [1]. I achieved the best result of 97% which is bit less when compared to 98.72% of the research paper.

In addition to that, I found that with dense feature set it can be complex to extract the feature or even find the output label classes. Also, noise in the feature set reduces the accuracy as well. I think the use of genetic algorithm to determine optimal feature set to classify the digits is good application for this dataset in the future. I would like to work with genetic algorithm and using Simplifying Hand written digit recognition paper [9] for further exploration.

5. References

1. Dutt, A., 2017. Handwritten Digit Recognition Using Deep Learning 6, 8.

2. Tu, M., Berisha, V., Cao, Y., Seo, J., 2016. Reducing the Model Order of Deep Neural Networks Using Information Theory. ArXiv160504859 Cs.

3. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges [WWW Document], n.d. URL http://yann.lecun.com/exdb/mnist/ (accessed 5.30.18).

4. Zhang, B., Yin, M., Lin, X., Zhu, Z., n.d. An Exploratory Study on MNIST Dataset 6.

5. CS231n Convolutional Neural Networks for Visual Recognition [WWW Document], n.d. URL http://cs231n.github.io/neural-networks-3/ (accessed 5.30.18).

6. CS231n Convolutional Neural Networks for Visual Recognition [WWW Document], n.d. URL http://cs231n.github.io/convolutional-networks/#pool (accessed 5.30.18).
7. Frankle, J., Carbin, M., 2018. The Lottery Ticket Hypothesis: Finding Small, Trainable Neural Networks. ArXiv180303635 Cs

8. Gedeon, T.D. and Harris, D. (1991) "Network Reduction Techniques," Proceedings International Conference on Neural Networks Methodologies and Applications, AMSE, San Diego, vol. 1: 119-126.

9. Parkins, A.D., Nandi, A.K., n.d. Simplifying Hand Written Digit Recognition Using A Genetic Algorithm 4