Distinctiveness Reduction Technique using Different Threshold Angles on Convolutional Neural Network

Zhiheng Zhou

Australian National University

Abstract. This paper is a report of the implementation of the **Distinctiveness** Reduction Technique [1] using different *similar threshold angles* on convolutional neural network's hidden fully connected layer. The *similar threshold angles* I considered in this research are 15°, 18°, 22.5°, 30°, 45°. For the evaluation of reduction (pruning) process, I use the testing accuracy and pruned hidden neurons number first. Then I construct two novel evaluation methods which are LPN and LPNS to evaluate the pruning results. Base on my analysis and the LPNS evaluation results, 30° is the best *similar threshold angle,* and 45° is the second preferred. Besides, comparing my results of this assignment and last assignment, when the *similar threshold angle* is same, **Distinctiveness** Reduction Technique tend to prune more hidden neurons on a fully connected neural. And by comparing the *k-means* method results on the same dataset implemented by other researchers [9], a *cnn* can work better than *k-means* for image classification problem.

Keywords: Reduction, Pruning, Distinctiveness, Convolutional Neural Network, Back-propagation, Sigmoid.

1 Introduction

Neural Network is a powerful and hot tool to solve some real work problem nowadays. Thus more and more researches are focused on a neural work. However, designing a neural network is not an easy thing. For example, there are countless choices for the designed number of hidden neurons in a fully connected neural network.

In some hand-crafted neuron network construction, to get a good performance for a special classification problem, many researchers usually will construct a network with a great number of hidden neurons. Because normally more hidden units can have more decomposition for the data. However, the number of hidden units usually will much more than the real number which needed to solve the problem. Therefore, in most situation, there are hidden units in those large hand-craft which take no real function for the final goal. In other words, those hidden units can be removed from the network.

Meanwhile, time complexity is always a significant factor for training a network. The solution which takes a shorter time is always be preferred than those longer solutions if they have same outcomes for a specific problem. Therefore, for a given problem, the optimal network model for it is the smallest size one which also has a great performance on the problem.

In this research, I use a *cnn* (Convolutional Neural Network) to work out hand write number classification problem. Then, I use a reduction (pruning) technique which is called "**Distinctiveness**" [1] (constructed by Gedeon, T.D. and Harris, D. 1991) on *cnn*'s hidden layer of the fully connected layer and observe the pruned *cnn* performance. Because the **Distinctiveness** reduction technique is based on the *threshold angle* of hidden neurons output vector (will be introduced in detail at next section), I use different some *threshold angles* for **Distinctiveness** to prune the original *cnn* and compare the performance of them. I also create two novel evaluation methods to evaluate the performance of a pruned *cnn*. Then, based on those evaluation results, try to find the most appropriate angle among those different *threshold angles* for **Distinctiveness** technique.
Besides, I will compare the results of this assignment and last assignment. And comparing the cnn results in this research to *k-means* method results implemented by other researchers [9].

1.1 Data Set - MNIST

The MNIST database [7] (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems [5]. The MNIST database contains 60,000 training images and 10,000 testing images [6]. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset [8]. For each instance, the attributes are an image byte of a human's handwriting **Arabic numerals**, and the label is what the **numerals** they have written (from 0 to 9). Thus there are 10 classes for labels. Figure 1 is an example of this dataset:

Figure 1. Example of dataset.
Source: By Josef Steppan - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=64810040

2 Methods

In this section, I will introduce the neural network model (cnn) first and explain the hyperparameters I used. Then introduce the network reduction(pruning) method which is **Distinctiveness**. Last, introduce the novel evaluation methods I construct for pruning process and the motivation of them.

2.1 Model Selections

Since the data is an image, I use *cnn* in this research. *Cnn* is a kind of powerful neural network which can deal with spatially related data like the image. It has multiple convolutional layers and ends with a fully connected layer. In each convolutional layer, there is one convolutions operation and one subsampling (pooling) operations. Figure 4 is an example structure of a *cnn*.

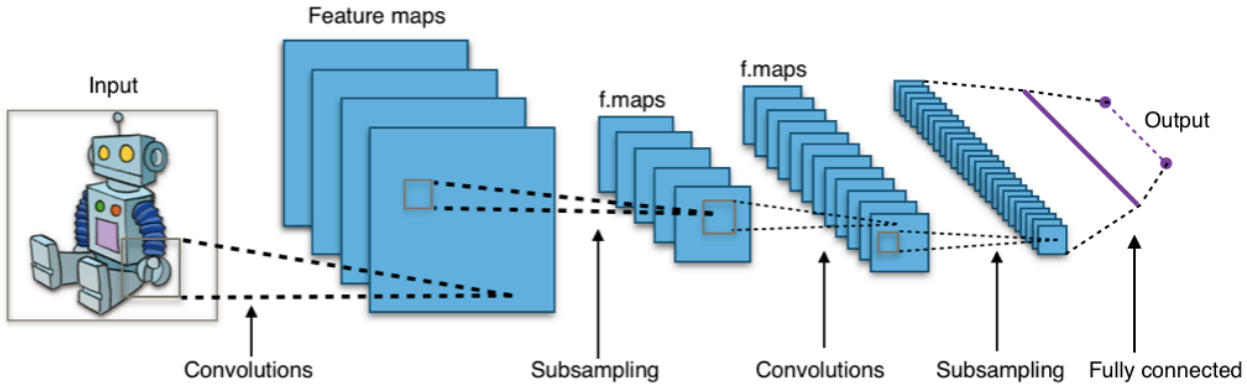

Figure 4. *Cnn* example.
Source: By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

For the structure of *cnn* in this research, I use the structure in **lab 3 task 1** since it is a recommended structure for the dataset. Because the goal of this research is to find a good *threshold angle* for **Distinctiveness** technique. Thus we do not have to spend much time on tuning the hyperparameters of the neural network. We just need to focus on not bad structure (like the recommendation structure) and using different *threshold angles* for **Distinctiveness** on it.

The recommendation *cnn* structure has two convolutional layers. The first convolutional layers have 10 out channels while the second has 20 out channels. Both two convolutional layers using 5x5 filter, 2x2 max pooling and a ReLU function on the outputs.

For the fully connected layer in *cnn*, the hidden layer has 50 neurons and the output layer have 10 neurons (since there are 10 classes). Since the fully connected layer only has one hidden layer (not very deep), I choose sigmoid function as the activation function. The advantage of this function over ReLU activation function is it can bond the output from 0 to 1 while ReLU activation function cannot. Therefore, choosing sigmoid do not need to do something more to bond the output.

For training part, I will use back-propagation (gradient descent) of cross-entropy error measures. Because the back-propagation method has shown a great success in some earlier research [2,3].
For the back-propagation method in training part of the network. I use Rprop (resilient back-propagation) method but not SGD (stochastic gradient descent). Because in SGD, the learning rate is fixed overall train epoch, so it needs to be set to a very proper value to make it work better. If the learning is too small, it will take a long time for the train part and still cannot get a really good result. However, if learning is too big, it does train network fast, however, when the loss is near

the minimal value, it just needs a small step to reach it, but the learning rate is too big, it will go through the minimal value and stay around the minimal value but never reach it.
And Rprop method solves problems elegantly, it gives each weight an individual learning rate. And it will update each learning rates base on the direction of corresponding weights update value. If current update value direction is same as the previous update value direction, the learning rates of this weight will multiple 1.2, if they are opposite direction, the learning rate multiple 0.5. And in practice, Rprop method performance is much better than SGD. So I choose Rprop as the optimizer function.

2.2 Reduction Technique – Distinctiveness

Reduction (pruning) technique is removing those no real function hidden neurons from the neural network. Different techniques have a different method to detect the no real function hidden neurons. In this research, I choose **Distinctiveness** reduction technique for the pruning process.

In this method, we are trying to use the output from all hidden neurons over the train set to determine the distinctiveness of hidden neurons. Note that the output is the output after a sigmoid function. These outputs can construct a matrix which x-axis is the hidden neurons and y-axis is the training instances. In the outputs matrix, each column is an outputs vector for a hidden neuron. The vector dimension has corresponded to the number of train instance and the element inside is the activation output of every instance in training dataset. Those vectors represent the functionality of corresponding hidden neurons in training data space. Therefore, we call the vector as functionality vector to the corresponding hidden neuron. And each functionality vectors represent the functionality of corresponding hidden neuron. The table 1. below shows an example of these vectors:

	Hidden Neurons Outputs		
Instance	1	2	3
i01	0.3	1.0	0.9
i02	0.1	0.9	0.3
i03	0.1	0.4	0.8

Table 1.

In above table, vector (0.3, 0.1, 0.1) is the functionality vector of hidden neuron 1 over the training instances i01, i02, i03. Similarly, vector (1.0, 0.9, 0.4) is the functionality vector of hidden neuron 2 and vector (0.9, 0.3, 0.8) is the functionality vector of hidden neuron 3.

Base on above, the pruning (reduction) process of this method is all base on the functionality vectors of hidden neurons. As the normal vector comparison method. We use the cosine similarity on the functionality vector to determine the relationship between two functionality vectors. The cosine similarity of two vectors can represent the angle between these two vectors.

However, before the pruning process, some pre-processing operation needs to be done on those functionality vectors. Since all outputs in the vector are after the activation function (sigmoid), so they are constrained to range 0 to 1. Consequently, the vectors are bonded to the first quadrant which all the variables are at the positive axis. This situation makes the angle between two functionality vectors is bonded in 0° to 90°. However, this means all functionality vectors are pointing in the same direction (positive) at all axis (dimensions), or at most vertical to others. Which is not a good situation for vector comparison, because it ignores the situation that two functionality vectors are opposite to each other, which is a significant situation in vector comparison. Therefore, the pre-processing operation normalized the origin point from (0, 0) to (0.5, 0.5). Then the opposite situation of two functionality vectors will happen. The following two figs shows that how this operation work:

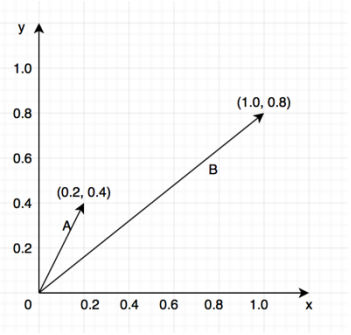

Figure 2.

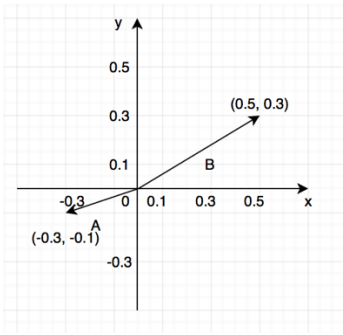

Figure 3.

Base on above figs, after normalization to (0.5, 0.5), vector A from (0.2, 0.4) to (-0.3, -0.1) and vector B from (1.0, 0.8) to (0.5, 0.3). And these two vectors are on the opposite direction for both x and y axis. And for the high dimension space, the original point just need to be changed from (0, 0, … 0) to (0.5, 0.5, … 0.5).

After normalization pre-processing on functionality vectors. Now we can start the similarity comparison part. As mentioned above, the angle between two vectors can be used as similarity comparison.
If the angle between two vectors is 0°, then these two vectors are exact as same as each other and one of them can be replaced. However, the situation that two vectors are totally same is rare, so a *similar threshold angle* (originally set to 15° in [1]) can be used here to identify those similar vectors. If the angle between two vectors is less than the *similar threshold angle*, then two vectors can be considered as similar vectors. This means the corresponding neurons of these two vectors have similar functionality in the network. Therefore, one of the neurons can replace another and the replaced one can be removed from the network to prune the network size. For the replacement step, the weight vector (not the functionality vector) and bias of removed neuron will be added to the similar functionality neuron's weight vector and bias. Then the other neuron can be removed from the network without losing significant functionality.

On the other situation, if the angle between two functionality vectors is 180°. It means the neurons corresponding to those vectors have the opposite functionality in the network. Their functionality will counteract each other functionality in the network. For example, the functionality of one of them is **f** and the another is **-f**, when they sum up, **f + (-f) = 0**. Which means the total functionality of them is zero in this network. Therefore, both can be removed from the network. Similarity, an *opposite threshold angle* (originally set to 165° in [1]) can be used here to extend the opposite range. For two neurons hiving opposite functionality in the network, both of them can be removed from the network without a significant side effect.

In conclusion, this reduction (pruning) technique will combine those similar functionality hidden neurons and remove all the opposite functionality neuron pair in the network without a significant side effect in above analysis. Besides, *similar threshold angle* + *opposite threshold angle* = 180°.

2.3 Evaluation Methods for Pruning

Base on above section, *similar threshold angle* + *opposite threshold angle* = 180°, thus 180° - *similar threshold angle* = *opposite threshold angle*. Therefore, we just need to pick up some different *similar threshold angle,* then we can use the equation above to get the *opposite threshold angle*.

There are 5 *similar threshold angles* will be implemented in this research which are 15°, 18°, 22.5°, 30°, 45° which respectively are, pi/12, pi/10, pi/8, pi/6, pi/4.

For each *threshold angle*, first I will show the number of hidden neurons being pruned by **Distinctiveness** using this *threshold angle*. Then is the testing set accuracy of pruned *cnn* for each *threshold angle*.

Intuitively, it is easy for us to get the increase of the pruned hidden neurons number, the testing set accuracy of pruned *cnn* will decrease. And the smaller *similar threshold angle* will have less similar and opposite functionality vectors which means the smaller *similar threshold angle* will make the pruned hidden neurons number smaller. Therefore, the smaller *similar threshold angle* will have smaller pruned hidden neurons number and higher testing set accuracy of pruned *cnn*.

And the pruning purpose here is to prune as many hidden neurons as it can without losing much accuracy (I consider less than 2% intuitively) in testing.

Base on above, I construct some novel evaluation methods which can balance the pruned hidden neurons number and the testing set accuracy of pruned *cnn* to evaluate the pruning results for different *threshold angles*.
The first evaluation method called *accuracy **L**oss **P**er pruned **N**euron (LPN)*. The formula of this method is below:

$$LPN = \frac{original\ cnn\ accuracy - pruned\ cnn\ accuracy}{pruned\ hidden\ neuron\ number + 1}$$

To avoid the zero denominators, since sometimes **Distinctiveness** technique with a small *threshold angle* cannot prune any hidden neuron from *cnn*, I use one plus the pruned hidden neuron number as the denominator. *LPN* use the accuracy gap between original *cnn* and pruned *cnn* to divide by one plus the hidden neurons number pruned by **Distinctiveness** with a specific *threshold angle*. Therefore, the *LPN* results are the average loss of testing set accuracy for a pruned hidden neuron. For this evaluation method, the smaller value is preferred since we smaller average loss of testing set accuracy for a pruned hidden neuron is preferred.

Base on first evaluation method, since I want to increase the pruned hidden neuron number influence of the evaluation final score. Thus I construct the second evaluation method called *accuracy **L**oss **P**er pruned **N**euron **S**quared (LPNS)*. The formula is below:

$$LPNS = \frac{original\ cnn\ accuracy - pruned\ cnn\ accuracy}{(pruned\ hidden\ neuron\ number)^2 + 1}$$

I add one to the denominator for the same reason of above method. The difference between this method and above method is that this method using the squared of pruned hidden neuron number as denominator while the above just use pruned hidden neuron number. Similar to above method, the smaller value of this method is preferred. Since *LPNS* using the squared of pruned hidden neuron number as the denominator, a pruned *cnn* with a large pruned hidden neuron number will have a much smaller *LPNS* value than its *LPN* value while a pruned *cnn* with a small pruned hidden neuron number will just have a slightly smaller *LPNS* value than its *LPN* value. Therefore, *LPNS* focus more on pruned hidden neuron number than *LPN*.

3 Results and Discussion

In this section, first I will show all the evaluation method results for five different threshold angles in **Distinctiveness** technique. Then compare the results of my first assignment and this assignment. Last, compare my *cnn* accuracy to an accuracy get from the k-means method using the same dataset constructed by other researchers [9].

3.1 Results for Different Threshold Angles

For the experiment, I train the cnn using 60000 training images. And test the cnn using 10000 testing images. Below are the results figures:

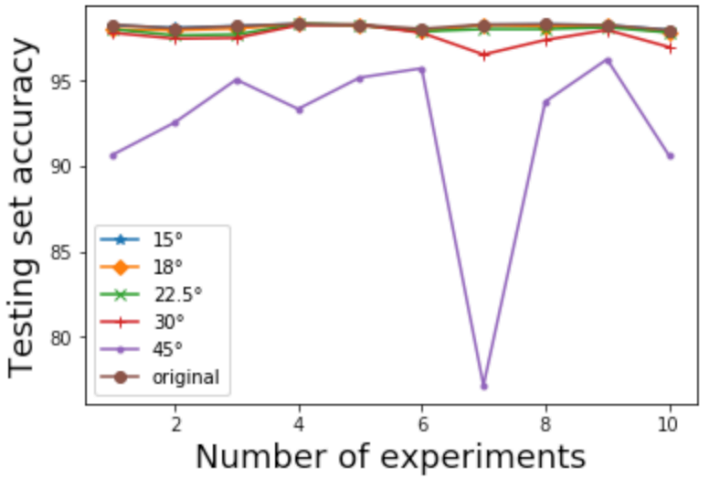

Figure 5. Testing accuracy results.

Figure 5 shows the line chart results of testing accuracy for original cnn and five pruned *cnns* pruned by **Distinctiveness** techniques using five different *threshold angles*. The x-axis is the experiment number (which in this research I do the experiment ten times). The y-axis is the testing accuracy. And the left-bottom of this figure is the label of each line chart in the figure. For example, 45° means the *similar threshold angle* of this **Distinctiveness** technique is 45°.
In above figure, when *similar threshold angles* are 15°, 18°, and 22.5°, we can observe that the testing accuracy of them are very close to the original one which is around 98% accuracy, which means the pruned *cnns* using these *similar threshold angles* will not lose much testing accuracy for the final results. When *similar threshold angle* is 30°, we can get that the testing accuracy decreases about 2% from original testing accuracy for the seventh experiment and decrease about 1% for the eighth and tenth experiment. However, when *similar threshold angle* is 45°, the overall testing accuracy is much less than the original and the others pruned cnns. And the testing accuracy of it is quite unstable, the highest testing accuracy can reach around 95%-96% while the lowest one is less than 80%.
As my analysis in Section 2.3, as the increase of the *similar threshold angle*, the testing accuracy will decrease, and the pruned hidden neurons number will increase. From above figure, we can confirm the former analysis. And for the latter analysis, figure 6 shows the results of the pruned hidden neurons number.

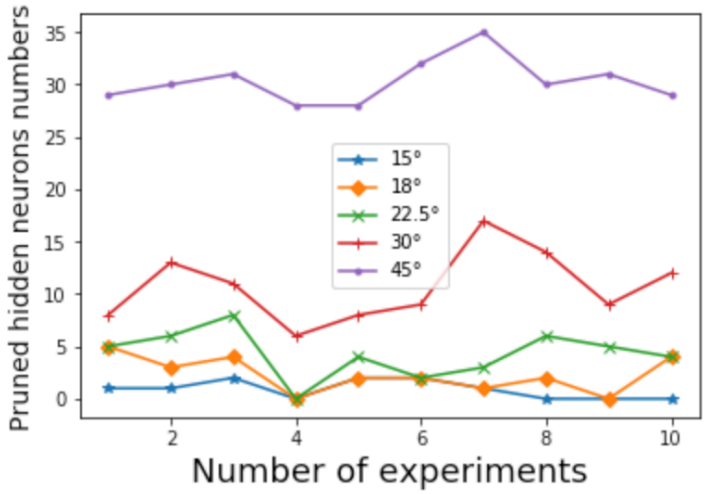

Figure 6. pruned hidden neurons number results.

The description of x-axis and labels of figure 6 are as same as figure while y-axis is the number of pruned hidden neurons. Because the pruned hidden neurons for original *cnn* is always zero, thus I am not showing the line chart of it in figure 6. From above figure, the average pruned hidden neurons number for 45° *similar threshold angle* is around thirty and as expected, it is much more than the others *threshold angles*. And the average pruned hidden neurons number for 30° *similar threshold angle* is around ten and it is also more than the others *threshold angles* excepted 45°. The average pruned hidden neurons number for 22.5° *similar threshold angle* is around five and the average pruned hidden neurons number for 18° and 15° *similar threshold angle*s are less than five. In conclusion, based on figure 6, we can observe that the bigger *similar threshold angle* can prune more hidden neurons from *cnn*. Therefore, we can confirm the latter analysis mentioned above.

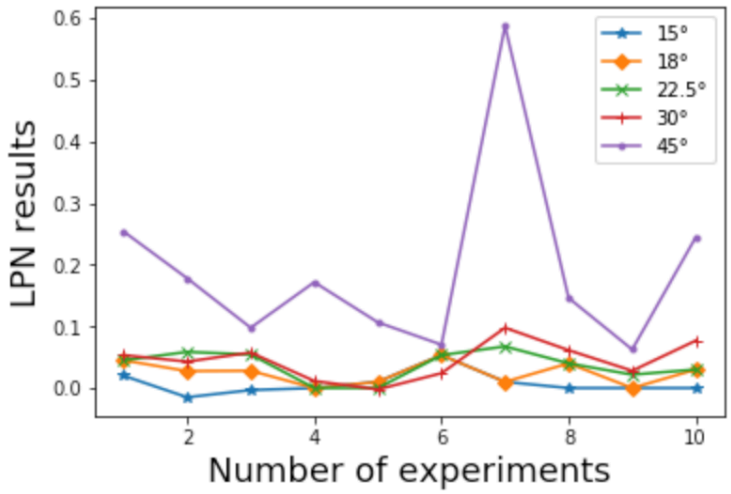

Figure 7. LPN results.

Figure 7 shows the LPN results for different *threshold angles*. In this figure, the LPN results for 15° *similar threshold angle* have the most smallest value among all experiments. However, among all the results for 15° *similar threshold angle*, a great amount of them did not prune any hidden neuron from the original *cnn*, thus the testing accuracy of this pruned *cnn* is equal to the original one and the numerator of LPN formula is zero. Because of this, a great number of its LPN results is zero which is a quite small LPN result.

What is more, form this figure, we can get an outlier result for 45° *similar threshold angle* at the seventh experiment. From figure 5, the testing accuracy for 45° *similar threshold angle* also reaches its lowest at the seventh experiment. Besides, the LPN results for 30° *similar threshold angle* reach its highest (worst) at the seventh experiment and its lowest testing accuracy in figure 5 is also appear at the seventh experiment.

Combine figure 5 and figure 7, we can observe that figure 7 is very similar to the vertical opposite version of figure 5. And based on above analysis, we can get that the LPN results focus much more on testing accuracy than the pruned hidden neurons number which is not we expected. Therefore, there come the LPNS results which will focus more on pruned hidden neurons number.

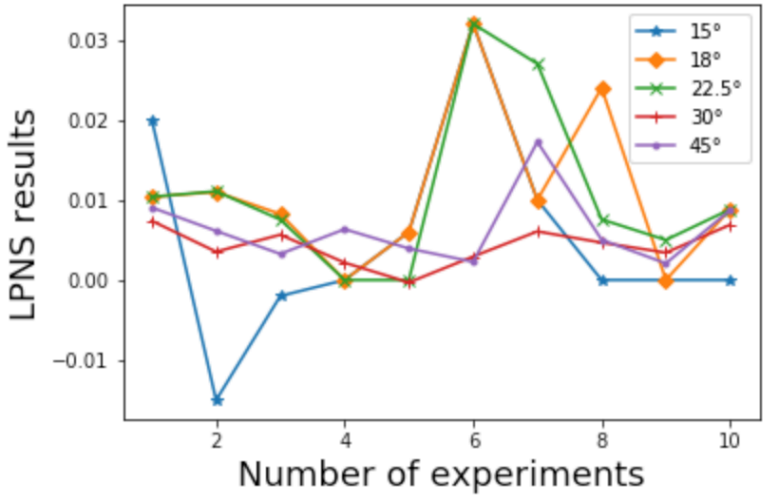

Figure 8. LPNS results.

Form figure 8, we can observe that 15°, 18°, 22.5° *similar threshold angles* have unstable LPNS results compare to 30° and 45°. And since most LPNS values of 30° and 45° are less than 18°, 22.5°, therefore, based on LPNS value, 30° and 45° are more suitable for **Distinctiveness** technique than 18° and 22.5°. Then compare 30° to 45°, we can observe that most LPNS values of 30° are less than 45° LPNS values.
For 15°, the results of it at second experiment are less than 0 because the testing accuracy of its pruned cnn is greater than the original one, however, from figure 6 we can know that it only prunes one hidden neuron from the original *cnn* at the second experiment. And since most of its prune hidden neurons number are zero, therefore, and LPNS value are meaningless when its numerator is zero. Therefore, I ignore 15° in the comparison.
In conclusion, LPNS value focus more on pruned hidden neurons number meanwhile it not losing the testing accuracy. Therefore, 30° have the best average LPNS value in figure 8. And then is the 45°.
And from all figure in this section, we can get 30° *similar threshold angle* can prune a good number of hidden neurons (average around ten), but not losing much testing accuracy (average losing 0.6%) while others smaller angles can only average prune less than or equal to five hidden neurons number form original *cnn*. If we want to prune more hidden neurons, 45° *similar threshold angle* is also a good choice because the average pruned hidden neurons number can reach thirty, and the best accuracy only have about 2% testing accuracy gap between the original *cnn* and its pruned *cnn*.
Base on above and the pruning purpose in Section 2, 30° *similar threshold angle* do is the most suitable angle here, then is the 45° *similar threshold angle*. And LPNS value can represent that. Therefore, I consider LPNS evaluation method is better than LPN evaluation method.

3.2 Results Comparison for Assignment 1 and Assignment 2

In assignment 1, I used **Distinctiveness** technique on a three-layer fully connected neural network with one input layer, one hidden layer, and one output layer while this assignment using *cnn*. The optimizer for both assignments is same (which is Rprop). And I choose 20 and 50 as the initial hidden neurons number.
Since I only use 50 original hidden neurons in this assignment (I use 20 and 50 original hidden neurons in assignment 1). I will only compare the results for 50 original hidden neurons here.
And since I only use 15° for **Distinctiveness** technique in assignment 1 (I use 15°, 18°, 22.5°, 30° and 45° in this assignment). I will only compare the results for **Distinctiveness** technique using 15° as *similar threshold angle*.
Base on above, my comparison results are based on:
1. 50 original hidden neurons.
2. Using 15° as *similar threshold angle* in **Distinctiveness** technique.

The comparison results I showed below are the accuracy gap between the pruned neural network and original neural network and the pruned hidden neurons number:

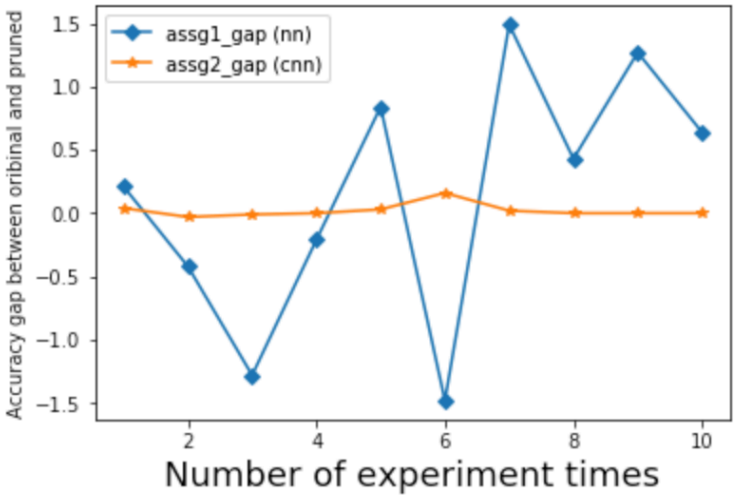

Figure 9. Accuracy gap result.

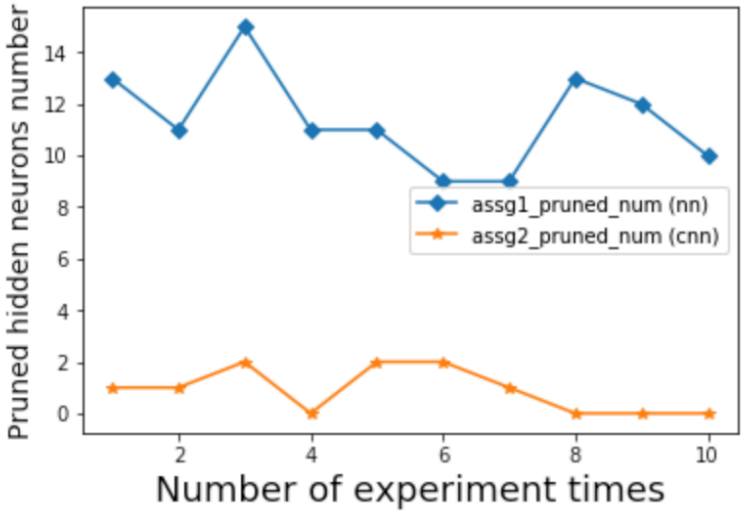

Figure 10. Pruned hidden neurons number.

In figure 9, the y-axis value is (original *nn* (assg1) or *cnn* (assg2) testing accuracy – pruned *nn* or *cnn* testing accuracy). From this figure, we can observe that the gap for assignment 2 is very stable while the gap for assignment 1 is very unstable. The reason for this situation I consider is because **Distinctiveness** technique in assignment 2 only can prune little hidden neurons, thus the pruned *cnn* is quite similar to original *cnn* in the assignment, then the gap between them is quite small and stable (close to zero).
From figure 10, we can observe the pruned hidden neurons number for each assignment clearly. As I analysis above, the pruned hidden neurons number in assignment 2 to is less than or equal to two. And since the average pruned hidden neurons number in assignment 1 is about twelve, the accuracy gap in assignment 1 is unstable. However, despite the accuracy gap in assignment 1 is quite unstable, there are situations that the pruned *nn* has a higher testing accuracy than the original *nn* in assignment 1 (for those gaps less than 0 in figure 9).

3.3 Results Comparison for Cnn and K-means

In the paper "A Large Scale Clustering Scheme for Kernel K-Means" [9], the authors use *k-means* method on the MNIST dataset. The best accuracy for the *k-means* method they can get is about 96%. While the original *cnn* in this research can have around 98% testing accuracy even sometimes reach 99%. Besides, all the pruned *cnns* for different *threshold angles* except 45° can have better results than k-means method (The worst testing accuracy for 30° is greater than 96%).

From above, we can obtain that a convolutional neural network can work better than *k-means* methods for image classification problem. The reason I consider for why a convolutional neural network can work better than *k-means* method is that *k-means* are an unsupervised method. In *k-means* processing, it does not use the label information of an image but only using the pixel RGB value and location in the image to clustering image. However, for *cnn* training process, we use image information as input and using training set image labels to adjust *cnn's* weights, thus we use both image information and image labels for the training process. Therefore, a *cnn* can work better than *k-means* methods because *cnn* using image labels to train while *k-means* not.

What is more, in training process, *cnn's* convolutional layer can automatically find good features among these images and extract them to classify. While *k-means* can only base on image pixel RGB value and location to classify. This is another reason I consider why a *cnn* can work better than *k-means* method for image classification problem.

4 Conclusion and Future Works

In this section, first I conclude the report above. Then talk about the future works of this research.

4.1 Conclusion

From Section 3.1, we know that 30° *similar threshold angle* can average prune ten hidden neurons from original *cnn* and the best testing accuracy of its pruned *cnn* is as same as the original one (fourth and fifth experiment and figure 5). 45° *similar threshold angle* can average pruned thirty hidden neurons from original *cnn* and the best testing accuracy of its pruned *cnn* have less than 2% gap between original *cnn* and its pruned *cnn*. And others *threshold angles* have a tiny gap between its pruned cnn and original cnn while their average pruned hidden neurons number is less than or equal to five. According to the pruning purpose, we want to prune as many hidden neurons as it can without losing much accuracy in testing. We can know that 30° *similar threshold angle* can prune about one fifth hidden neurons from original cnn which is much more than those smaller *similar threshold angle* (15°, 18°, 22.5°) and the accuracy gap between its (30°) pruned *cnn* and original *cnn* is really close to those smaller *similar threshold angles'* gap. Therefore, 30° is preferred compared to hose smaller *similar threshold angle* (15°, 18°, 22.5°). And for 30° and 45°, though 45° can prune more hidden neurons than 30°, the testing accuracy loss for 45° is larger than testing accuracy loss for 30°. Balance the testing accuracy loss and prune more hidden neurons number. I consider 30° is more preferred than 45°. However, if we want to prune more hidden neurons, 45° is also a good choice. Therefore, I consider 45° as second preferred *similar threshold angle*.
Besides, according to LPNS results in figure 8, we can also know the best LPNS results is for 30° and the second best is 45° which is identical to the analysis results above. Therefore, I consider LPNS is a good evaluation method for pruning.

From Section 3.2, we know that when the *threshold angle* is same, the **Distinctiveness** reduction technique may tend to prune more hidden neurons from a *nn* than *cnn*. And a *cnn* may need larger *threshold angle* to prune more hidden neurons from *cnn's* fully connected layer.

From section 3.3, we know that a *cnn* can work better than *k-means* method for an image classification problem.

4.2 Future Works

For the future works of this research. First, we can use others image dataset on research, and we may find others most suitable *similar threshold angle* (not 30°) for others dataset. If the most suitable *similar threshold angle* for others dataset is still 30°, then we have a high chance to say the most suitable *similar threshold angle* for pruning a same structure *cnn's* hidden fully connected layer is 30°.
Second, in this research, I use five *similar threshold angles* which are 15°, 18°, 22.5°, 30° and 45°. Therefore, in future work, we could use more *similar threshold angles* for research. And we may find a more suitable *similar threshold angle* than 30° between 30° and 45° or between 22.5° and 30°.
Third, I only use a structure for *cnn* in this research. Thus, in future work, we could try different *cnn* structure and we may find different most suitable *similar threshold angle* for other cnn structure.

References

1. 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.
2. Dutta, S, Shekhar,S, "Bond rating: A non-conservative application of neural networks," IEEE Int Conf on Neural Networks, vol. II, pp. 443-450, 1988.
3. Sejnowski, TJ, Rosenberg, CR, "Parallel networks that learn to pronounce English text," Complex Systems, vol. 1, pp. 145-168, 1987.
4. K.A.J. Doherty, R.G. Adams and N. Davey, "Non-Euclidean Norms and Data Normalisation". Bruges (Belgium), 28-30 April 2004, d-side publi., ISBN 2-930307-04-8, pp. 181-186.
5. Gangaputra, Sachin. "Handwritten digit database". Retrieved 17 August 2013.
6. Kussul, Ernst; Tatiana Baidyk (2004). "Improved method of handwritten digit recognition tested on MNIST database". Image and Vision Computing. 22 (12): 971–981. doi:10.1016/j.imavis.2004.03.008.
7. https://en.wikipedia.org/wiki/MNIST_database.
8. Zhang, Bin; Sargur N. Srihari (2004). "Fast k -Nearest Neighbor Classification Using Cluster-Based Trees" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence. 26 (4): 525–528. doi:10.1109/TPAMI.2004.1265868. PMID 15382657.
9. Zhang, R., & Rudnicky, A. I. (2002). A large scale clustering scheme for kernel k-means. In Pattern Recognition, 2002. Proceedings. 16th International Conference on (Vol. 4, pp. 289-292). IEEE.