EE569 Introduction to Digital Image Processing

Homework Report #5

Name: Boyang Xiao **USC ID:** 3326730274 **Email:** boyangxi@usc.edu

I. Part(a) CNN architecture

1.1 Non-programming questions

Q1: Describe CNN components in your own words: 1) the fully connected layer, 2) the convolutional layer, 3) the max pooling layer, 4) the activation function, and 5) the softmax function. What are the functions of these components?

Answer:

1) The fully connected layer is the **CLASSIFIER** of the whole CNN. After the convolutional layers and pooling layers extract the features from the images and project the images to the feature spaces, the fully connected layers do the tensor multiplication operations and classify the original images with the extracted features.
2) The main function of convolutional layers is extracting the features from the images and project the image data from its original space to the feature spaces to do the classification.
3) The max pooling layers can reduce the size of the images and feature matrices to accelerate the computation of networks and to avoid over-fitting.
4) The activation functions can bring some non-linear operations to the CNN to improve network's classification ability. Since the main the classifier of CNN is the fully connected layer and the operations in the fully connected layer are all linear which has limited classification ability, the activation functions can reinforce its classification ability.
5) The softmax function can mapping all the output values to the range of [0,1] and make the sum of all the values to be 1, which conforms to the probability distribution. We can choose the node with the highest value to be the final output result.

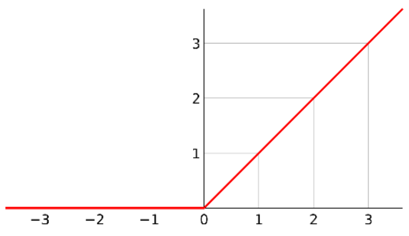
Q2: What is the over-fitting issue in model learning? Explain any technique that has been used in CNN training to avoid the over-fitting.

Answer: The over-fitting issue means the CNN model can perform very well on the training dataset but can perform worse on the testing dataset. This usually occurs when the model is too complicated and the scale of training dataset is limited. Some typical techniques to avoid over-fitting are: data augmentation, early stopping and dropout.

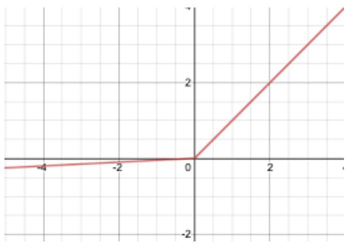
For example, technique of data augmentation means expanding the number of training dataset using some image modification methods such as rotation, flipping and translation. It can generate more distinguished training data using a limited scale of dataset, with which the CNN model can be better trained.

Q3: Explain the difference among different activation functions including ReLU, LeakyReLU and ELU.

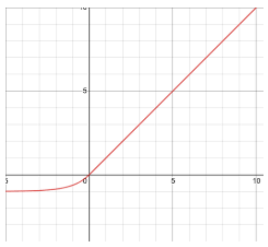
Answer: The shapes of these three functions are displayed below in Figure 1.1. We can tell from the graphs that **ReLU** has zero-value before x == 0 and has a linear tilting part after x == 0. **Leaky ReLU** is much similar to ReLU but has a slightly negative linear part before x == 0. **ELU** is similar to ReLU and Leaky ReLU but has a negative non-linear part before x ==0.



(a) ReLU



(b) Leaky ReLU



(c) ELU

Figure 1.1 Three common activation functions

When used in the CNN model, **ReLU** can greatly accelerate the convergence process of gradient descent. **Leaky ReLU** has the same advantage as ReLU and it can also preserve the information from its negative part to avoid some neuron to be stuck/dead if it falls in the negative part of ReLU. **ELU** is a combination of ReLU and sigmoid. The right part of ELU can accelerate the gradient descent and also avoid gradient vanishing issue and the left part of ELU can preserve more information and make the CNN model more adaptive to the noise in the training dataset.

Q4: Read official documents of different loss functions including L1Loss, MSELoss and BCELoss. List applications where those losses are used, and state why do you think they are used in those specific cases?

Answer: L1Loss function and MSELoss function are commonly used in the regression situations, such as stock price prediction and temperature prediction. They are used in regression cases because they are both typical regression loss functions and are more robust and more accurate.

BCELoss function is commonly used in binary classification cases because it only has binary outputs of 0 and 1 and it makes this function more efficient.

II. Compare classification performance on different datasets

2.1. Abstract and motivation

In this part of problem, a very classical and typical Convolutional Neural Network(CNN) will be introduced and tuned, that is LeNet-5 which was one of the earliest pre-trained models proposed by Yann LeCun and others in the year 1998, in the research paper *Gradient-Based Learning Applied to Document Recognition.* They used this architecture for recognizing the handwritten and machine-printed characters. The main reason behind the popularity of this model was its simple and straightforward architecture. It is a multi-layer convolution neural network for image classification.

In the following sections, LeNet-5 will be applied to classify some very popular and commonly-used image datasets: MNIST, fashion-MNIST and CIFAR-10, which are some earliest dataset designed for machine learning and model training. LeNet-5 model will then be tuned by trying different parameters to get higher accuracy both on training set and testing set.

2.2. Approaches and procedures

As shown in Figure 2.1 below, LeNet-5 has a structure of 2 convolutional layers (followed by 2 max-pooling layers respectively) and three fully connected layers. The convolutional kernel window size is 5×5 and all the max-pooling window size is 2×2 (which can reduce the size to be half). The first convolutional layer has 6 filter kernels and the second convolutional layer has 16 filter kernels. The neuron numbers for the three fully connected layers are 120, 84 and 10, respectively.

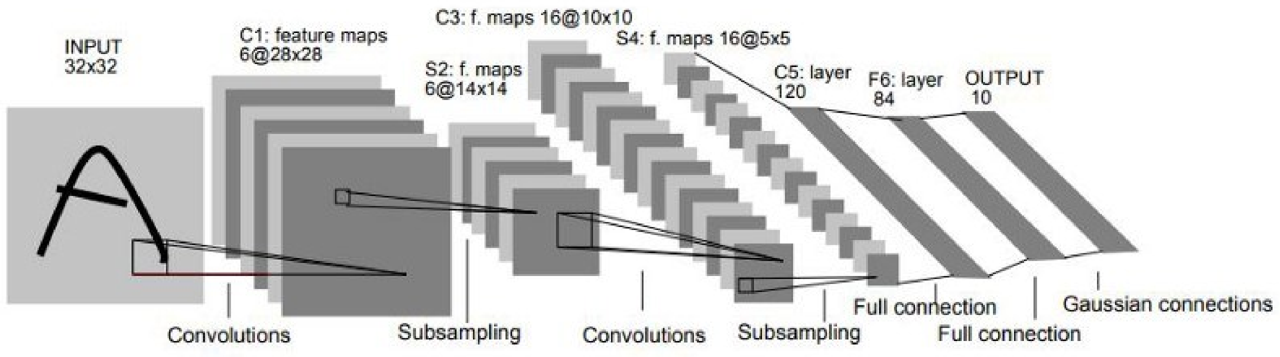


Figure 2.1 LeNet-5 network structure

The datasets MNIST, fashion-MNIST and CIFAR-10 are downloaded and separated into two groups: training set and testing set. The training set is used to train the CNN model and to adjust the weights of the neurons with its labels. The testing set does NOT involve in the training process of the model but only tests the accuracy of the model after each epoch.

The main hyper-parameters that will be tuned are the **learning-rate**, the **optimizer** and the **momentum**. Training results with different hyper-parameter settings will be shown and compared.

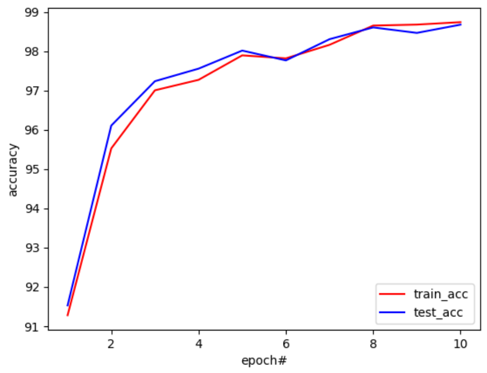
2.3. Experiment results on dataset MNIST

The three parameter settings for experiments on MNIST are shown in Table 2.1 below. The final training accuracy and the testing accuracy (averaged results from 5 runs) are also shown in Table 2.1. The max-epoch numbers are all 10.

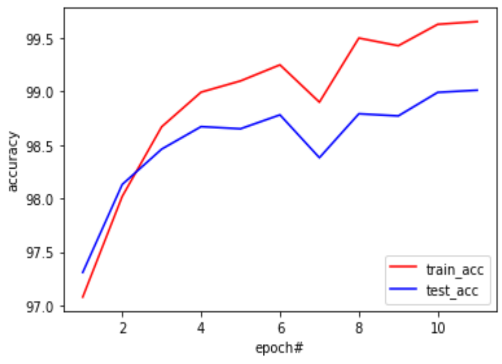
Table 2.1 Parameters settings and corresponding averaged results for MNIST dataset

NO.	Learning rate	Momentum	Weights decay	Training accuracy	Testing accuracy
1	0.001	0.9	0	98.743%	98.680%
2	0.003	0.9	0	99.650%	99.010%
3	0.001	0.7	0	95.655%	96.170%
4	0.001	0.9	0.05	95.917%	96.200%

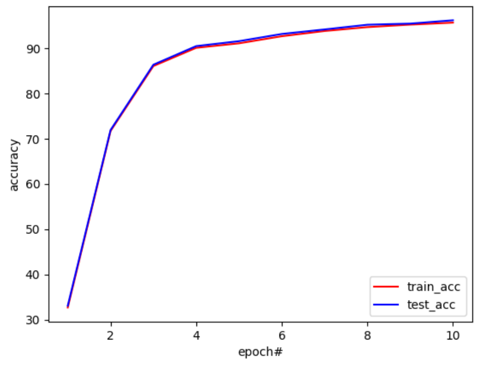
The training performance curves for the parameter settings above are shown in Figure 2.2 below, where each subplot's x-axis represents the epoch number and the y-axis represents the accuracy for both the training set and the testing set.



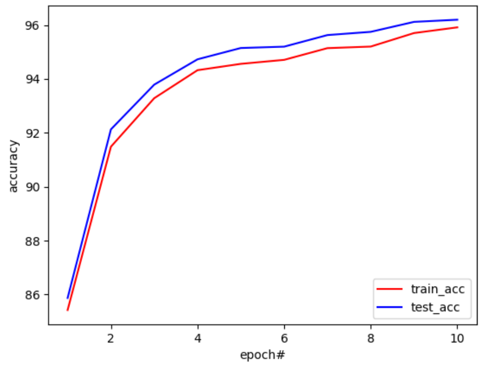
(a) Setting #1



(b) Setting #2



(c) Setting #3



(d) Setting #4

Figure 2.2 Performance curves for the 4 different parameter settings

From the results above, parameter setting#1 is the most common setting with all the default parameters setting and this is the standard setting. Compared to setting#1, setting#2 uses a higher learning rate and the CNN model's accuracy can rise much faster. However, the training process for setting#2 is not stable and has a phase of decreasing (but increases anyway after that). This is because the training rate is too high and when the accuracy reaches some level, the training is not able to get more accurate to reach the optimized point. But this high learning rate can also make the final accuracy higher and make the training process faster than the others anyway.

Setting#3 uses a lower momentum value compared to setting#1, which mainly influences the initialization of the whole CNN model. As a result, the starting accuracy (the first several epoch) is quite low because the bad choice of the momentum value, but it can still reach a higher accuracy after the training process and the testing accuracy is closer to the training accuracy, which means that the model is more robust to the unfamiliar data. However, the final accuracy is still lower than the previous settings.

Setting#4 uses a non-zero weights-decay value compared to setting#1 which uses the default zero-value decay, and this weights-decay parameter mainly influences the L2 regularization of the model. With this non-zero decay value, the training process is faster but the final accuracy is lower.

2.4. Experiment results on dataset fashion-MNIST

The three parameter settings for experiments on fashion-MNIST are shown in Table 2.2 below. The final training accuracy and the testing accuracy (averaged results from 5 runs) are also shown in Table 2.2. The max-epoch is set to be 30.

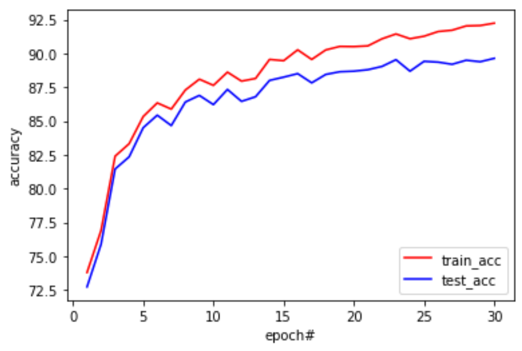
Table 2.2 Parameters settings and corresponding averaged results for fashion-MNIST dataset

NO.	Learning rate	Momentum	Weights decay	Training accuracy	Testing accuracy
1	0.001	0.9	0	92.230%	89.630%
2	0.005	0.9	0	95.790%	89.740%
3	0.001	0.6	0	86.748%	85.550%
4	0.004	0.78	0	93.583%	90.080%

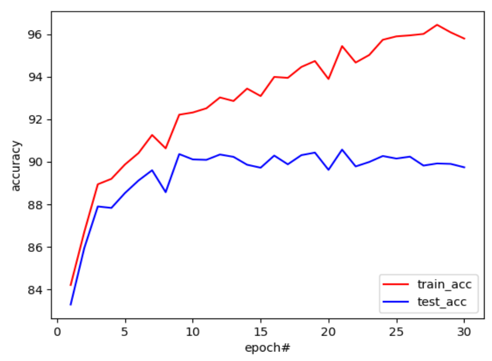
The training performance curves for the parameter settings above are shown in Figure 2.3 below, where each subplot's x-axis represents the epoch number and the y-axis represents the accuracy for both the training set and the testing set.

From the results above, the classification accuracies for fashion-MNIST dataset are lower than the results for MNIST dataset. For the standard parameters setting#1, the training accuracy can reach 92.230% and the testing accuracy can reach 89.630% after 30 epochs. Although the accuracies have an increase trend and can get higher if more epochs are given to the CNN model, the experiment only sets 30 epochs to compare different parameter settings.

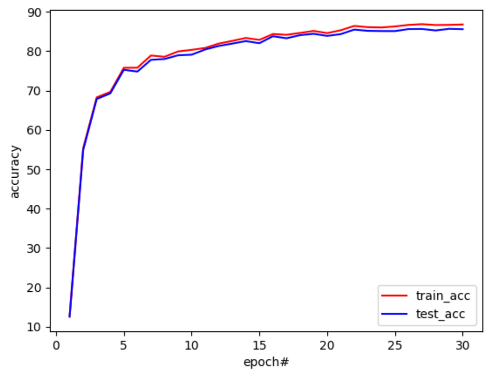
Setting#2 uses a higher learning rate and the training accuracy increases very fast to reach over 95%. But the testing accuracy is still low, which means that the CNN model is less robust to unfamiliar data. Also, the training process has some fluctuation phases due to the high learning rate.



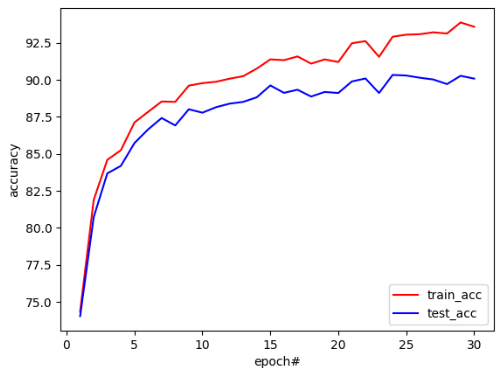
(a) Setting #1



(b) Setting #2



(c) Setting #3



(d) Setting #4

Figure 2.3 Performance curves for the 4 different parameter settings

Setting#3 uses a higher momentum which initialize the whole model in a different way and the testing accuracy is closer to the training accuracy, which means that the model is more robust and more adaptive to the unfamiliar data. However, the overall accuracy is lower if it goes through the same number of epochs.

Setting#4 combines the advantages of setting#2 and setting#3. The CNN model under parameter setting#4 has both a higher convergence speed and robustness. And the final testing accuracy can reach 90.080%.

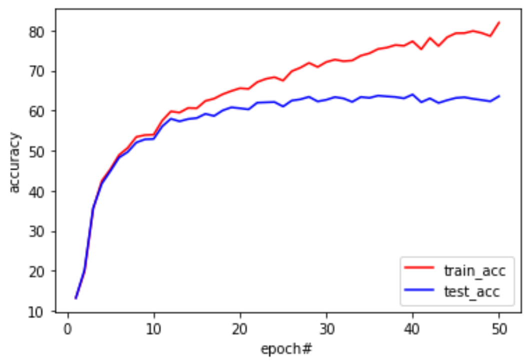
2.5. Experiment results on dataset CIFAR-10

The three parameter settings for experiments on CIFAR-10 dataset are shown in Table 2.3 below. The final training accuracy and the testing accuracy (averaged results from 5 runs) are also shown in Table 2.3. The max-epoch is set to be 50.

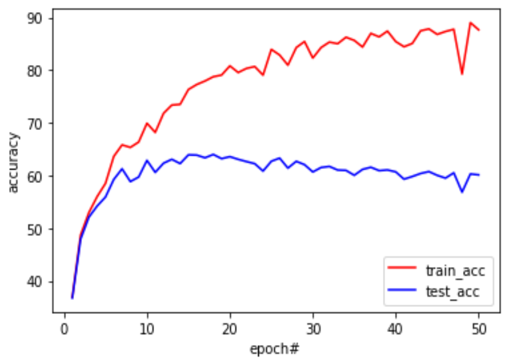
Table 2.3 Parameters settings and corresponding averaged results for the CIFAR-10 dataset

NO.	Learning rate	Momentum	Weights decay	Training accuracy	Testing accuracy
1	0.001	0.9	0	81.928%	65.550%
2	0.005	0.9	0	87.668%	60.130%
3	0.0003	0.9	0	63.944%	59.480%
4	0.0003	0.8	0	51.406%	50.950%

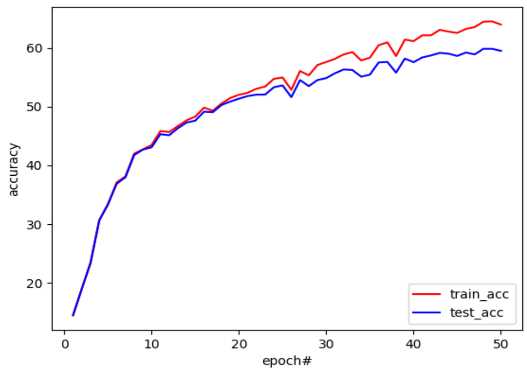
The training performance curves for the parameter settings above are shown in Figure 2.4 below, where each subplot's x-axis represents the epoch number and the y-axis represents the accuracy for both the training set and the testing set.



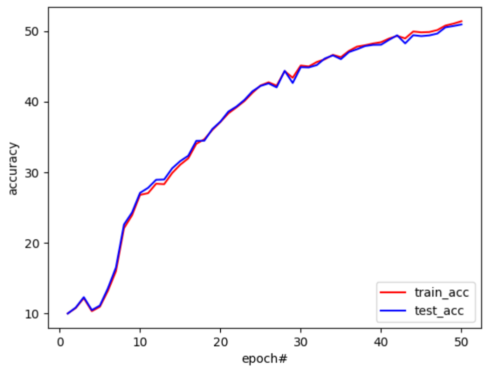
(a) Setting #1



(b) Setting #2



(c) Setting #3



(d) Setting #4

Figure 2.4 Performance curves for the 4 different parameter settings

Since the accuracies for CIFAR-10 dataset are much lower than the previous datasets we used, only the learning rate and momentum settings are considered in this section. Setting#1 uses the same setting as we used previously, and the testing accuracy can reach more than 65%.

Setting#2 and setting#3 use a higher learning rate and a lower learning rate respectively. With a higher learning rate, the training accuracy can get higher in a shorter range of epochs, but the testing accuracy gets even lower, which means the model has some over-fitting issue and is not adaptive to the unfamiliar data. Otherwise, with a lower learning rate, the training process gets smoother and the testing accuracy is closer to the training accuracy. However, we can find from the performance curve that the model does not converge yet within the 50 epochs and the accuracies can still get higher if more epochs are given to the model. Technically, this parameter setting can reach higher accuracy if trained further, which will be examined in the next section.

Setting#4 uses a lower momentum value, compared to setting#3, to improve the robustness of the model and we can find that the training process is much similar to the setting#4 but the testing accuracy is closer to the training accuracy, which can satisfy my expectations. The final accuracies are lower in this parameter setting because the lower momentum setting makes the model start from a lower accuracy and the model still need more epochs to converge.

2.6. Discussions

From the experiments results above on three different datasets, the CNN model shows different performance and results. Although different parameter settings can make some difference on a certain dataset, the overall accuracies on a certain dataset are quite different from each other.

The accuracies on MNIST dataset are the highest and the CNN model converges very fast on the MNIST training, because the MNIST images are simple and intuitive. It only contains the handwritten numbers from 0 to 9 and the patterns of these images are quite easy to analyze.

The fashion-MNIST dataset is actually similar to MNIST with the same image size and they are both gray-scaled. However, the fashion-MNIST dataset's images have much more complicated patterns and features, which only two convolution layers may not be able to extract and it's more difficult for the fully connected layer to do the classification. Therefore, the results on fashion-MNIST dataset are worse than the MNIST.

The CIFAR-10 dataset is the most difficult one among these three datasets we use in this section, because images in it have three channels and the size of images are larger, too. The CNN model structure is also modified to accept the three-channel images. The final accuracies are quite low in this experiment and the main reason for it can be that more epochs are needed to train the networks. Only 50 epochs are clearly not enough for training. Training with more epochs will be performed in the next part of problem.

2.7. Non-programming questions

All the questions are answered above.

III. Part (c) Analysis on confusion classes and hard samples

3.1 Abstract and motivations

In Section II., LeNet-5 model has already shown some extraordinary performance on classification, especially for the simple dataset such as MNIST and fashion-MNIST. The model can converge after a very short range of epochs and reach a quite high accuracy on the testing dataset. The performance curves and the final accuracies are used to analyze the performance and results of the model in the last section, but this is not enough.

In this part, the confusion matrix will be introduced and be used to analyze the results of the LeNet-5 model. Analysis will be performed on the three datasets we used before and the model will adopt the optimized parameter settings discovered before.

3.2 Approaches and procedures

Confusion matrix is a commonly-used visualization tool in the realm of machine learning, especially the supervised learning. Given the machine learning model will classify the samples into n classes, and the confusion matrix will be an n×n matrix. Every row represents a predicted class from the model's results and every column represents an actual labelled class from the dataset. Every node in the confusion matrix (say the node in position i×j) represents the proportion of samples that from the class ***j*** that are classified as class ***i***. The numbers on the diagonal lines are the real accuracies for each class and all the other nodes of the matrix represents the confused results.

To get the confusion matrix in our experiment, the CNN model is firstly trained with the training dataset. After we get the trained model, the testing dataset will be input to the model and all the classification results for all the samples will be recorded. We use these results to count the numbers of samples belonging to each predicted-actual pair and calculated the normalized confusion matrix.

To visualize the results, the confusion matrix is usually be plotted as a heatmap. The results of this experiment will be shown in the next several sections.

3.3 Experiment results for MNIST dataset

The parameter setting for this experiment is listed in the table below. And the final accuracies after 10 epochs are also shown in the table.

Learning rate	Momentum	Weights decay	Training accuracy	Testing accuracy
0.002	0.75	0	98.513%	98.350%

The confusion matrix obtained by this trained model for the testing dataset is shown below in Figure 3.1.

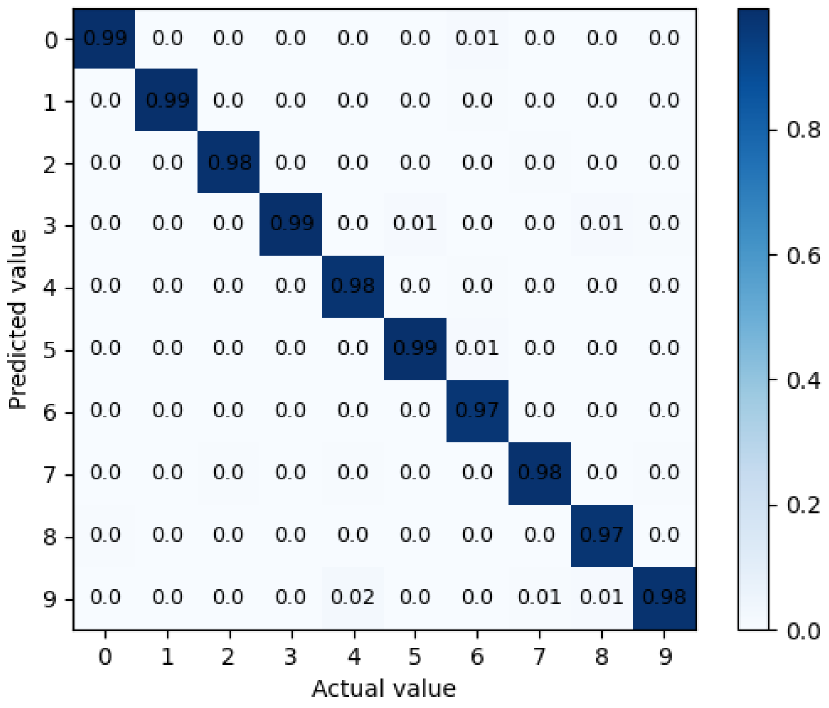
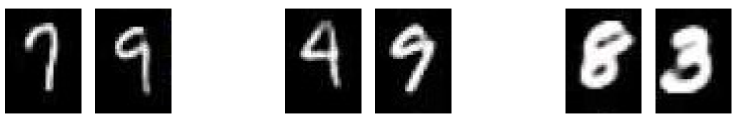


Figure 3.1 Confusion matrix for testing set from MNIST dataset

Since the MNIST dataset is quite simple and the accuracy can reach over 98%, the confusion matrix in this experiment has just several minor confused pair. The top three confused pairs are (notated in the format of [Actual class, Predicted class]):

- **[4, 9]**
- **[7, 9]**
- **[8, 3]**

This result is actually very intuitive because if we observe some sample images, we can find that the handwritten number “9” and handwritten number “4” are much similar. So are the pair of “7” & “9” and the pair of “8” & “3”, as shown in Figure 3.2 below. Therefore, the well-trained CNN model can make some minor mistakes on these confused pairs.



(a) pair of 7 and 9 (b) pair of 4 and 9 (c) pair of 8 and 3

Figure 3.2 Three pairs of confused classes samples from MNIST dataset

3.4 Experiment result on fashion-MNIST

The parameter setting for this experiment is listed in the table below. And the final accuracies after 30 epochs are also shown in the table.

Learning rate	**Momentum**	**Weights decay**	**Training accuracy**	**Testing accuracy**
0.004	0.78	0	93.810%	90.270%

The confusion matrix obtained by this trained model for the testing dataset is shown below in Figure 3.3.

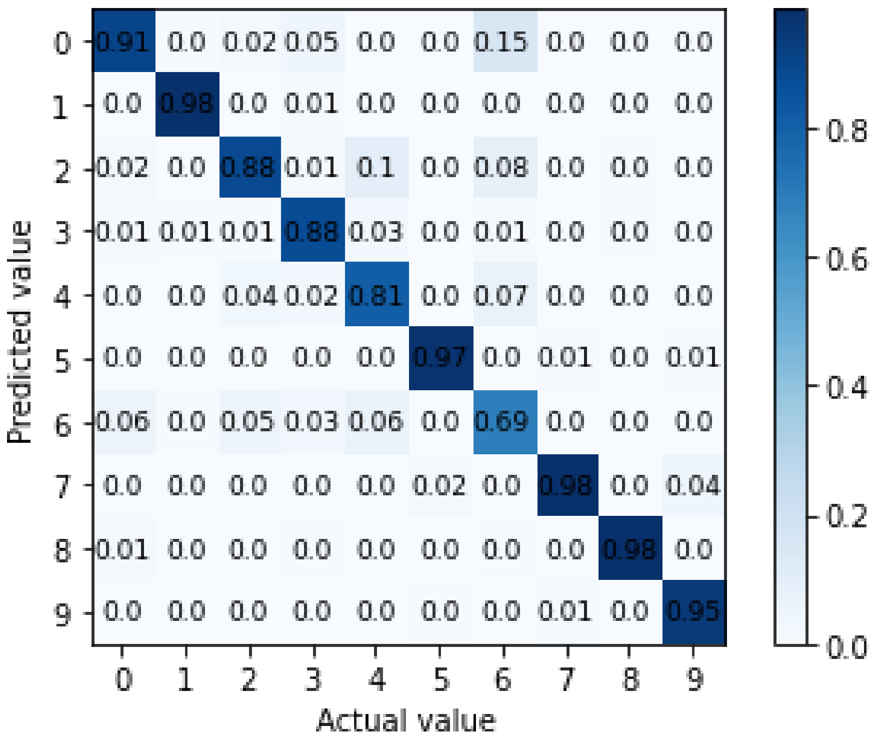


Figure 3.3 Confusion matrix for testing set from fashion-MNIST dataset

From the confusion matrix shown above, we can find that LeNet-5 model can perform well on more than half of the classes, especially for some distinctive classes, such as 1-trousers, 7-sneakers and 8-bag, because these three items are quite different from the other items, which makes it easier for the model to classify them.

However, some classes' results are much worse and the top three confused pairs are (notated in the format of [Actual class, Predicted class]):

- **[6-shirt, 0-T-shirt]**
- **[6-shirt, 2-Pullover]**
- **[6-shirt, 4-Coat]**

Some samples images of these four classes: 6-shirt, 0-T-shirt, 2-Pullover and 4-Coat, are shown in the Figure 3.4 below. As we can observe from the sample images, these four classes are all tops and they are very similar to each other with almost the same edge shapes, size and patterns. Therefore, the trained CNN model can still get confused about some samples from these classes.

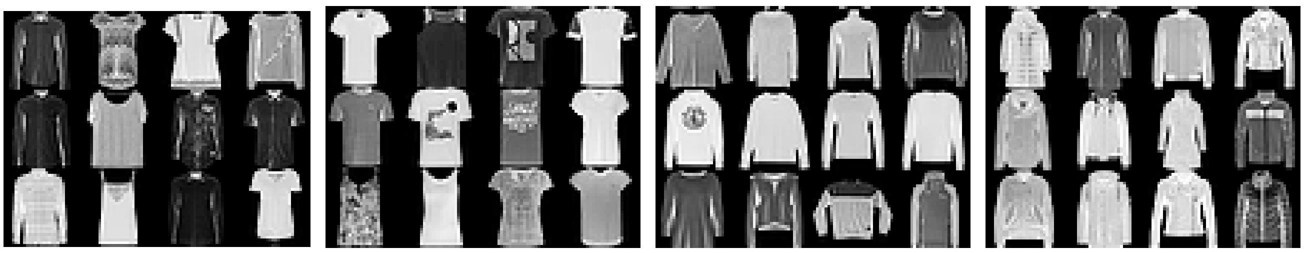

(a) 6-Shirts (b) 0-Tshirts (c) 2-Pullovers (d) 4-Coats

Figure 3.4 Four confused pairs sample images from fashion-MNIST dataset

3.5 Experiment results on CIFAR-10 dataset

The parameter setting for this experiment is listed in the table below. And the final accuracies after 80 epochs are also shown in the table.

Learning rate	Momentum	Weights decay	Training accuracy	Testing accuracy
0.0004	0.85	0	70.474%	63.020%

The confusion matrix obtained by this trained model for the testing dataset is shown below in Figure 3.5.

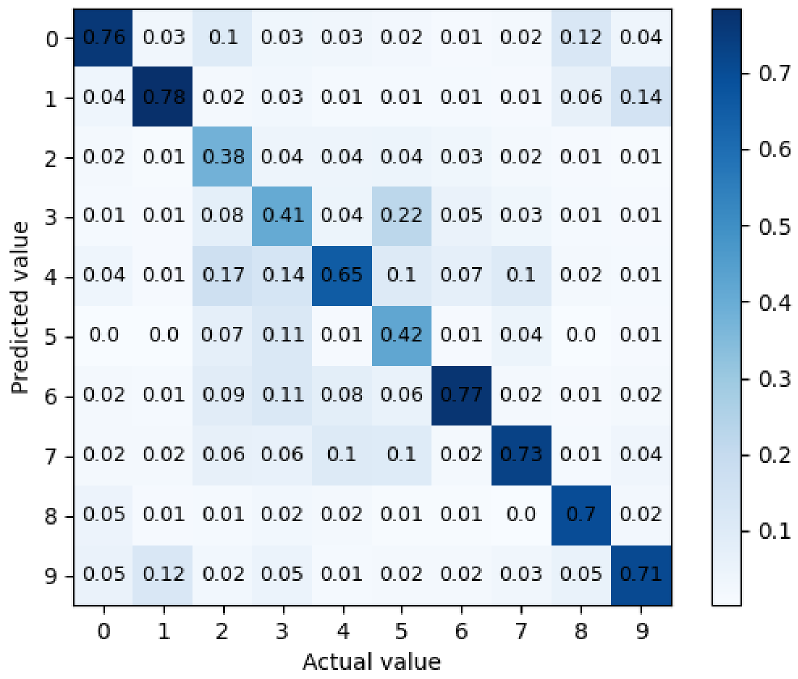
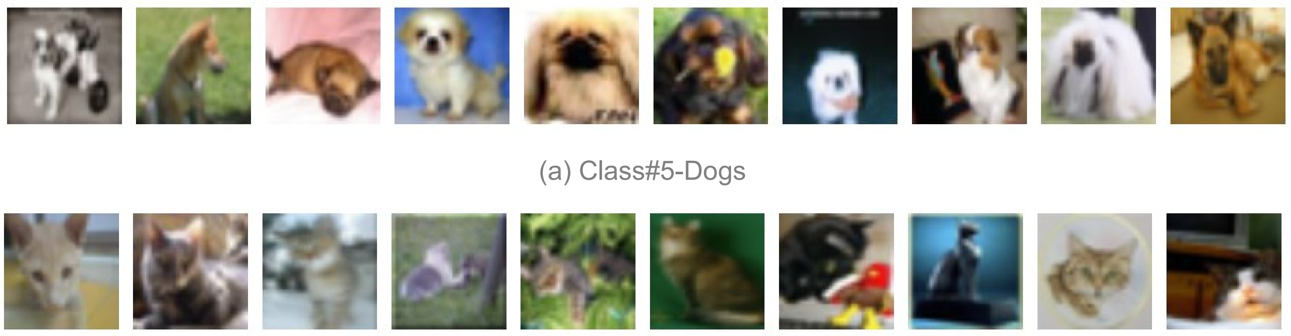


Figure 3.5 Confusion matrix for testing set from CIFAR-10 dataset

As we have tested in Section 2.5, LeNet-5's performance on CIFAR-10 is worse than the other two datasets because the CIFAR-10 dataset is much more complicated and contains color images. There are some classes are distinctive and they get some satisfying results, such as 1-automobil and 6-frog. But some classes are much similar to each other and more than half of each classes are confused with other classes. The top three confused pairs are (notated in the format of [Actual class, Predicted class]):

- **[5-Dog, 3-Cat]**
- **[2-Bird, 4-Deer]**
- **[9-Truck, 1-Automobile]**

Sample images from the first pair of two classes are shown in Figure 3.6, which are classes of 5-Dogs and 3-Cats. Dogs and cats are both small sized animals in the real world and they have very similar shape patterns and color patterns. Also, the images cannot show the real size of these animals, so this is much difficult for the CNN model to distinguish these two classes.



(a) Class#5-Dogs

(b) Class#3-Cats

Figure 3.6 Samples images from one pair of confusion classes of CIFAR-10

3.6 Discussions

All the discussions and explanations are included in each dataset's experiment results sections.

IV. Part (d) Classification with noisy data

4.1. Abstract and motivations

In the real-world situations, the datasets collected by companies and industries are usually not that pure as the distinguished datasets we previously used, such as MNIST and CIFAR-10. There are usually noises in the datasets, which means that some of the labels do not match the images samples, and this kind of situations is very common in the real projects of Deep learning.

In this part of problem, the influences brought by different levels of noises will be studied. MNIST dataset will be considered as a noise-free dataset and different levels of noises will be added to the MNIST dataset manually using a technique named Symmetric Label Noise (SLN). The testing accuracies of the networks trained by different noised datasets will be displayed.

4.2. Approaches and procedures

In this part of problem, the main method to add noises to MNIST dataset is called Symmetric Label Noise (SLN). SLN defines a parameter ϵ which can control the error rate in the noisy dataset, that is, after adding the noises, $1-\epsilon$ of the data has the correct labels and the rest ϵ of the data will get the rest wrong labels evenly. One thing that we should pay attention to is that noises should only be added to the training set but not the testing set.

My approach to implement this algorithm is as following:

- Firstly, I numerate all the labels in the training set. For each label, I generate a random number in the range of [0,1] and compare it to the parameter ϵ. If the random number is smaller, the label value will be kept, otherwise, the label value should be modified.
- To modify the label value, I define a list: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] and I remove the original label value from the list. Say the original label value is 4 and the new list should be [0, 1, 2, 3, 5, 6, 7, 8, 9] without number 4. Then I generate a random int in the range of [0, 8] and use it as the index to get the list member at this index. I assign this member's value as the label's new value. In this way, I can make all the modified labels be assigned with the rest of the wrong label values evenly.

In the Figure 4.1 below, the confusion matrix of the actual labels and modified labels is shown when ϵ is set to be 40%. From the confusion matrix, we can see that on the diagonal line, which hold all the correct labels, the correct rates are all 60%. And all the other nodes in the confusion matrix have a rate of around 4.44%, which can satisfy the requirements of SLN algorithm.

After apply the SLN to the MNIST dataset, CNN models will be trained 5 times and the testing accuracies' average values and the standard deviation values will be calculated. The results will be shown and discussed in the next section.

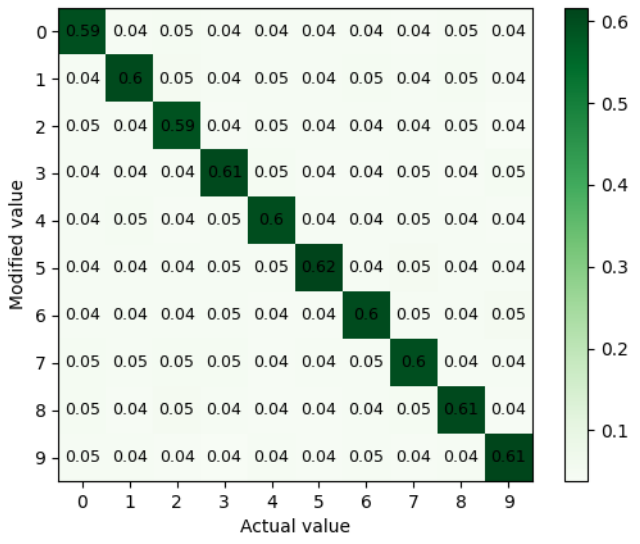


Figure 4.1 confusion matrix of the actual labels and modified labels

4.3. Experiment results

To train the CNN model, the parameter settings are as same as the settings in Section 3.3. And the testing accuracies results under different ϵ values are shown in the Table 4.1 below.

Table 4.1 Testing accuracies results under different ϵ values

ϵ value	Testing accuracy mean value	Testing accuracy standard deviation
0%	98.354%	0.152%
20%	97.870%	0.119%
40%	97.348%	0.165 %
60%	95.756%	0.609%
80%	85.708%	2.642%

The curves of [testing accuracy mean value - ϵ value] and [testing accuracy standard deviation - ϵ value] are plotted and shown in Figure 4.2 on the next page.

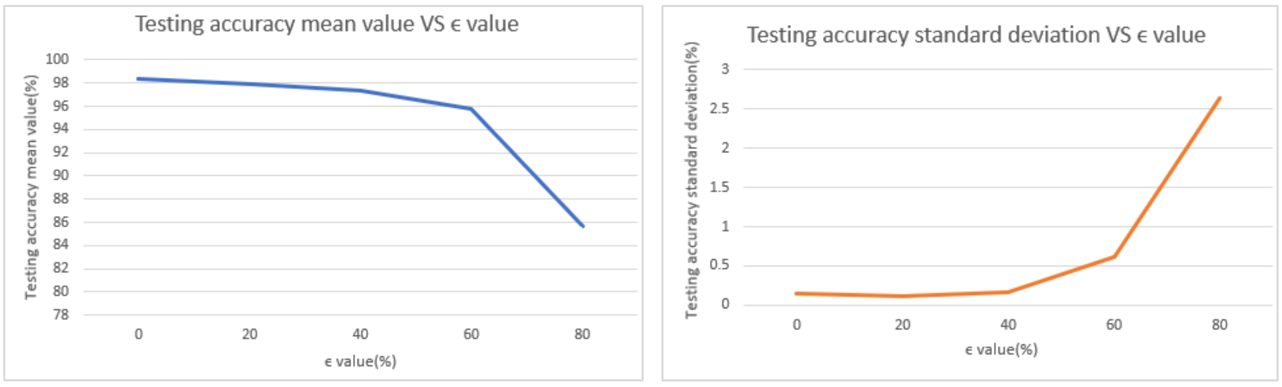
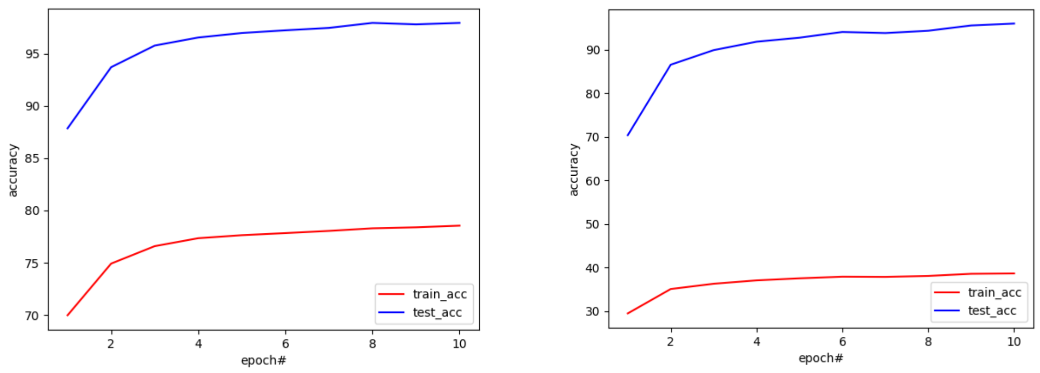


Figure 4.2 Testing accuracy - ϵ value curves

4.4. Discussions

From the experimental results shown in Section 4.3 above, adding different noises to the MNIST dataset can actually make very limited difference to the testing accuracy, and the testing accuracy will decay massively only when the noise level is high enough. If the noises are not too large, they can bring little influence to the CNN model training results. But with the ϵ value getting higher, the standard deviation of the testing accuracies among several parallel experiments is also getting higher, which indicates that high level of noises will make the training results of CNN models become less stable.

However, if we take a look at the training performance curves in Figure 4.3, we will find that although the testing accuracy is still very high with the noisy datasets, the training accuracy is quite low. The low accuracy of training dataset is caused by the wrong labels, which means that the accuracy can still get high if the labels are correctly marked because the classification ability of the CNN model is quite strong and can hardly be influenced by some wrong labels.



(a) ϵ = 20% (b) ϵ = 60%

Figure 4.3 Training performance curves of two different noisy datasets

To conclude, in real-world situations, datasets with a limited level of noises (e.g. some little error labels) will actually not influence the training results too much. We can control the noises level to prevent it from getting too high, and in this way we can guarantee the training performance.

4.5. Non-programming questions

All the questions are answered above.