5241 Project Report

Yuqi Zhang (yz4091@columbia.edu)

Table of Content

Question 3 Single Layer Model

```
def savemodel(seed_range,epoch):
    for i in range(0, seed_range):
      set_random_seed(i)
      sgd = SGD(learning_rate= 0.1)
      model = Sequential()
      model.add(Dense(100, activation='relu', input_shape=(784,)))
      model.add(Dense(10, activation='softmax'))
      model.compile(optimizer=sgd,loss=SparseCategoricalCrossentropy(),
                    metrics=[SparseCategoricalAccuracy()])
      model.fit(train_x,Y_train,epochs=150,verbose = 0,
                validation_split=0.2, batch_size=128)
      model.save('model%d.h5'%(i))
```

In this question, I reshape the dataset so that my input shape is 784. Then I normalize the training dataset to perform a single layer neural network with 100 hidden units and ReLU activation function. I compile the model using SGD with 0.1 learning rate as the optimization function, cross entropy as the loss function, accuracy as the matrics. I fit the model setting the validation split as 0.2 and batch size as 128. Since we are required to repeat more than 5 times with different random seeds, I train this model six times and each time with a different random seed from range (0, 6) and 150 epochs each time as required.

3 (a)

Plot the average training cross-entropy error (sum of the cross-entropy error terms over the training dataset divided by the total number of training examples) on the y-axis vs. the epoch number (x-axis). On the same figure, plot the average validation cross- entropy error function. Examine the plots of training error and validation/test error (generalization). How does the network's performance differ on the training set versus the validation set during learning? Use the plot of training and testing error curves to support your argument.

Answer to 3(a)

Below (Fig.1) are the six plots of average training cross-entropy error (y-axis) vs. the epoch number (x-axis). From my observation, the trends of all six figures with different seeds are significantly similar. The vertical purple line in the graph represents the best epoch, which is determined by the minimum validation loss, and from the plots, we can easily observe that the best epoch for all plots are between 30 and 60. From the best epoch line, we can observe that it becomes overfitted after epochs around 60.

In general, validation loss is smaller than training loss at the beginning. However, training errors keep decreasing, but validation errors decrease to a minimum and start to become steady and slightly increase. So, the training errors are smaller than the validation errors after epochs around 10.

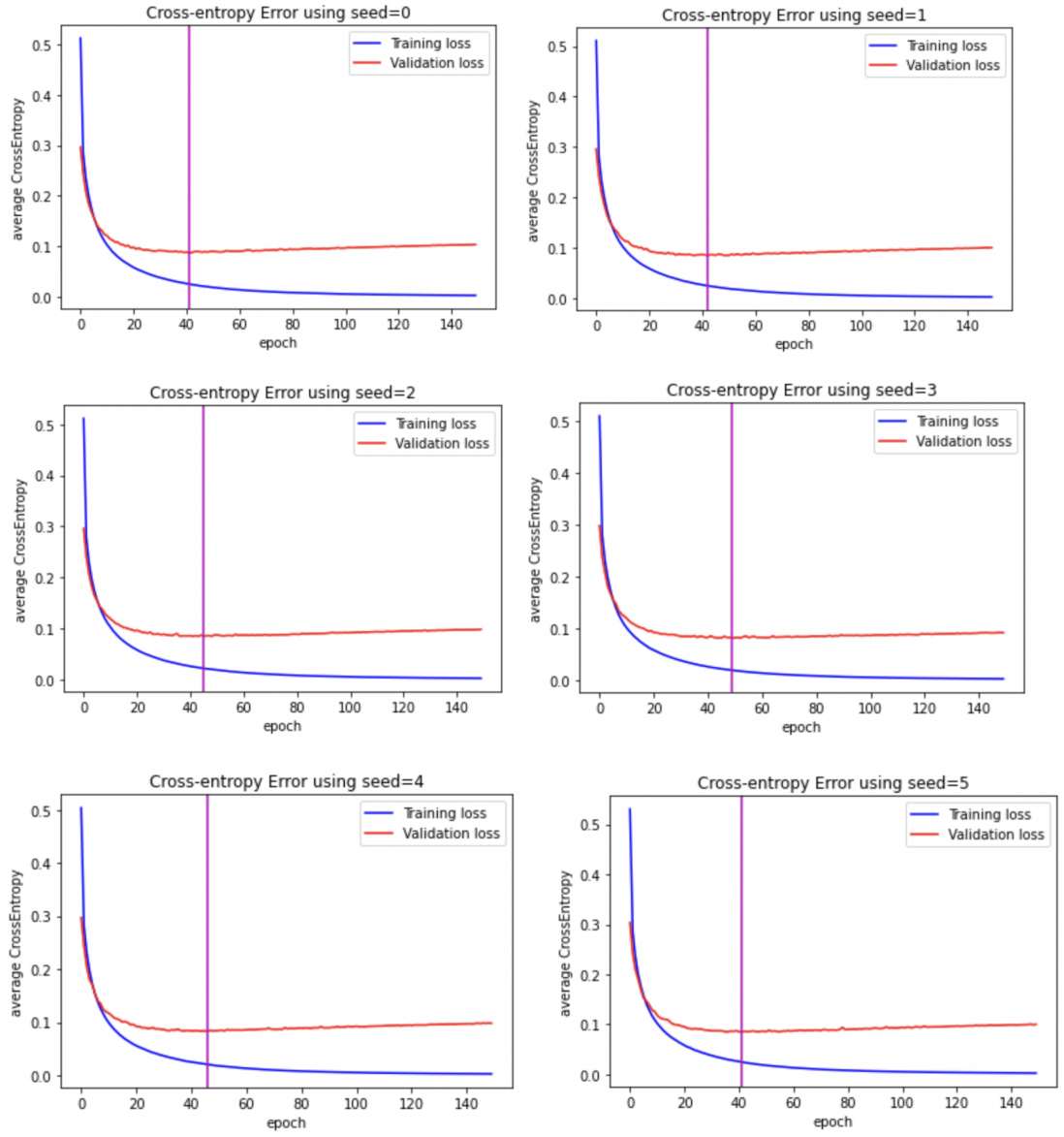


Figure 1: Average Cross-entropy Error v.s. Epoch for different seeds

3 (b)

We could implement an alternative performance measure to the cross entropy, the mean mis-classification error. We can consider the output correct if the correct label is given a higher probability than the incorrect label, then count up the total number of examples that are classified incorrectly (divided by the total number of examples) according to this criterion for training and validation respectively, and maintain this statistic at the end of each epoch. Plot the classification error (in percentage) vs. number of epochs, for both training and testing. Do you observe a different behavior compared to the behavior of the cross-entropy error function?

Answer to 3(b)

The mis-classification error is one minus the accuracy. I make the mean mis-classification plots with y represents the classification error in percentage and x represents the number of epochs, for both training and testing. In Figure 2, the best epoch is measured by the maximum of accuracy, which is the same as the minimum of misclassification error.

From the plots, it seems there is not too much difference between Figure 1 and Figure 2. The trends in Figure 2are very similar to that in Figure 1 except the range of y since y represents the error in percentage. Moreover, the best epoch is a little different from these two kinds of error plots.

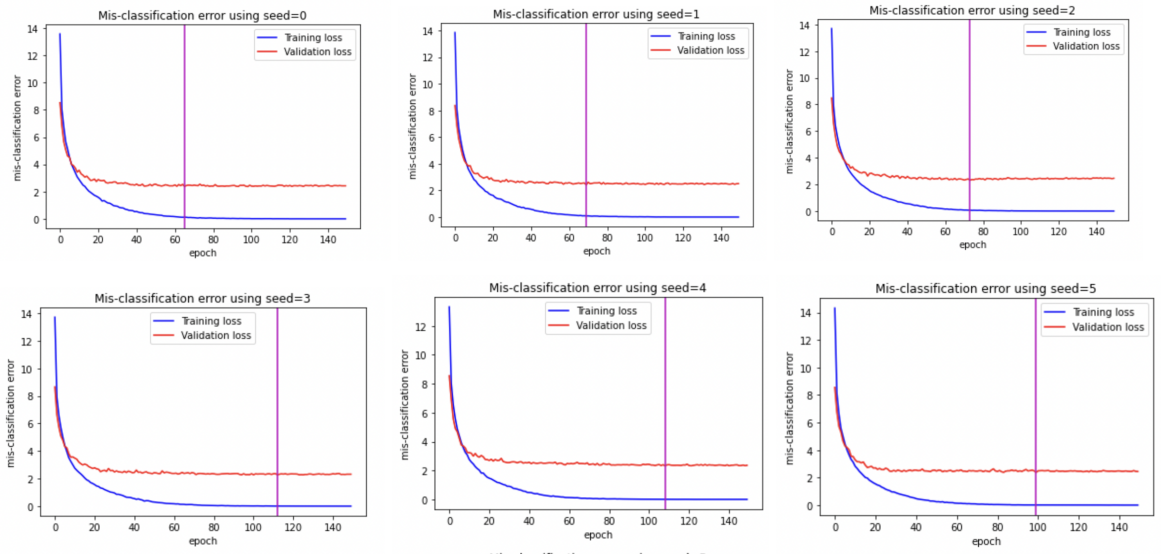


Figure 2: Mis-classification error (%) v.s. Epochs for different seeds

3 (c)

Visualize your best results of the learned W as one hundred 2828 images (plot all filters as one image, as we have seen in class). Do the learned features exhibit any structure?

Answer to 3(c)

Since it is difficult to conclude the best results from the plots, I check the accuracy to help me compare different models. According to the accuracy (Table 1), I choose the 4th model with random seed 4, which has the highest accuracy 0.9773, as my best model.

```
[0.9763333201408386,
 0.9759166836738586,
 0.9769166707992554,
 0.9773333072662354,
 0.9768333435058594,
 0.9762499928474426]
```

Table 1: Accuracy of different models

My best results of learned W as 100 28X28 images are shown below (Fig.3). This image contains some abstract features for which we have no words or mental concept. The display of feature visualizations along with training data can help. The learned features show some blurry contours of digits but are still not interpretable.

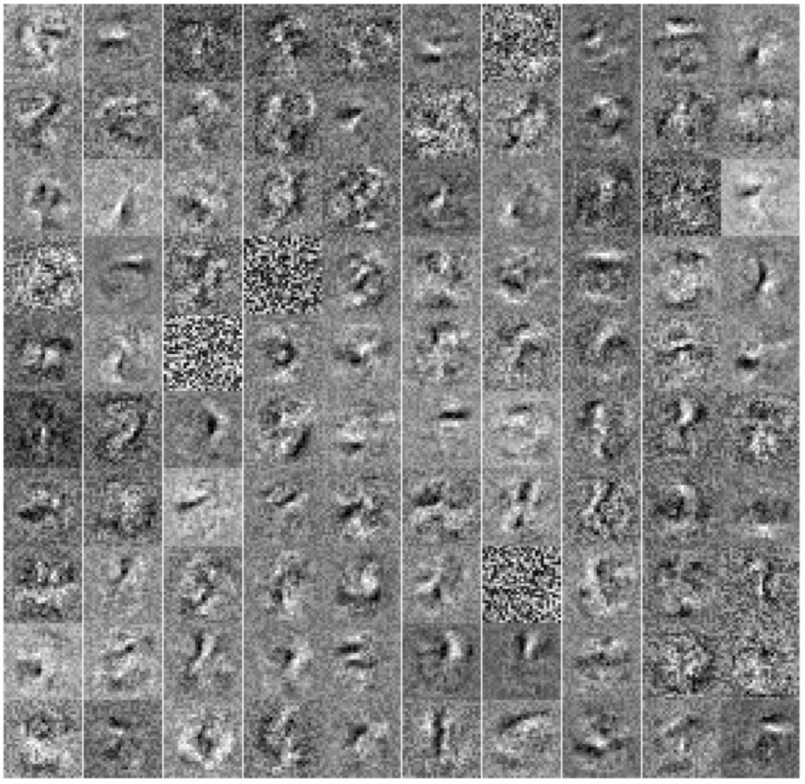


Figure 3: Feature map

3 (d)

Try different values of the learning rate. You should start with a learning rate of 0.1. You should then reduce it to .01, and increase it to 0.2 and 0.5. What happens to the convergence properties of the algorithm (looking at both average cross entropy and % incorrect)? Try momentum of 0.0, 0.5, 0.9. How does momentum affect convergence rate? How would you choose the best value of these parameters?

Answer to 3(d)

I tried a total of 12 different combinations of learning rate in (0.1, 0.01, 0.2, 0.5) and momentum in (0.0, 0.5, 0.9). To observe the convergence properties, I draw twelve plots to visualize the results with the best epoch line measured by the minimum test loss. Two Loss lines represent the cross-entropy error and two accuracy lines represent misclassification error rate.

We mainly focus on the error in the test set, which are the yellow and blue lines. From blow plots (Figure 4), with the momentum increase with the same learning rate, the best epoch line moves toward y axis. The move of the best epoch line indicates that with the momentum increase, the test error reaches the minimum in less epoch, so that we can get the best result in less epoch.

Similarly, we can Figure 4 vertically to observe at the same level of momentum. The best epoch line can help us observe the converge rate better. From the plots, we can conclude the best epoch moving to the left with the increase of learning rate. From Figure 4.1 and 4.2 we can see that if the learning rate changes from 0.1 to 0.01, we have to run a lot more epochs to get the minimum test error. Also, if the learning rate changes from 0.01 to 0.2, we only need to run less than 40 epochs to get the minimum test error.

So, we can conclude, with the increase of either learning rate or momentum, we can get the minimum error in less epoch. However, the plot of learning rate is 0.5v.s. momentum is 0.9 is weird. The minimum test loss is more than 0.2 and the test loss keeps increasing a lot after reaching the minimum. So, I may conclude that when learning rate and momentum are both large, the value of minimum test error may increase. My choice of best parameter would be learning rate is 0.2 and momentum is 0.5. I think in that case I can get the small test error in reasonable epoch number and would not be overfitting a lot with the increase of the epoch.

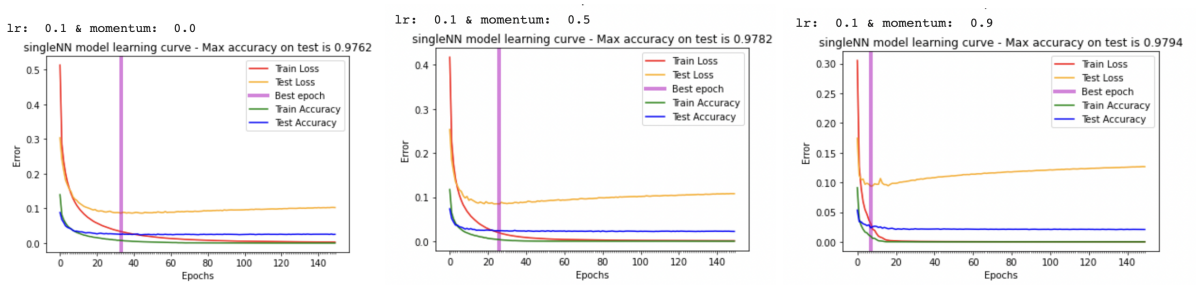


Figure 4.1: learning rate 0.1 v.s momentum in (0.0, 0.5, 0,9)

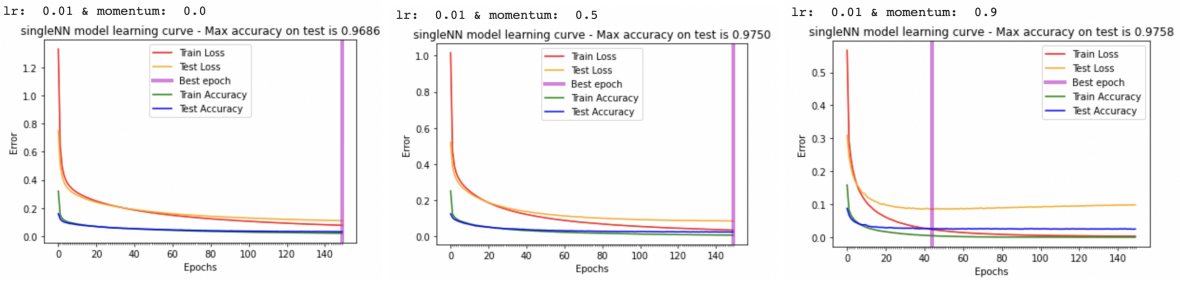


Figure 4.2: learning rate 0.01 v.s. momentum in (0.0, 0.5, 0.9)

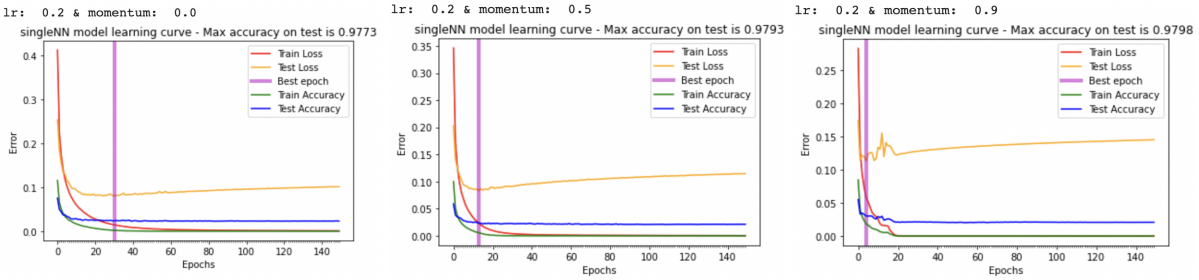


Figure 4.3: learning rate 0.2 v.s. momentum in (0.0, 0.5, 0.9)

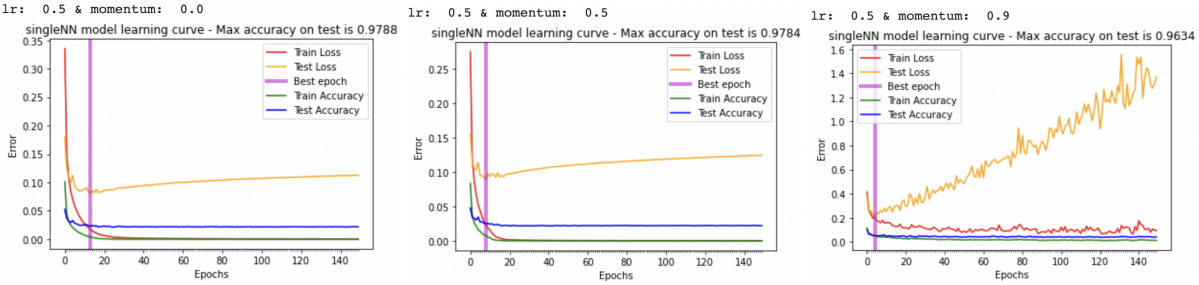


Figure 4.4: learning rate 0.5 v.s. momentum in (0.0, 0.5, 0.9)

Question 4 CNN

```
def cnn(seed):
  model = Sequential()
  model.add(Conv2D(filters=32, kernel_size=(3,3),
                   activation='relu',
                   input_shape=(28, 28, 1)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(100, activation='relu'))
  model.add(Dense(10, activation='softmax'))
  # compile model
  np.random.seed(seed)
  opt = SGD(learning_rate=0.01)
  model.compile(optimizer=opt,
                loss=losses.CategoricalCrossentropy(),
                metrics=['accuracy'])
  return model
```

In this question, I reshape the dataset so that my input shape is (28, 28, 1). Then I normalize the training dataset to perform a CNN model. I add one 2-D convolutional layer with ReLU activation function and kernel size (3 ,3). Then add Maxpooling with (2, 2). I compile the model using SGD with 0.1 learning rate as the optimization function, cross entropy as the loss function, accuracy as the matrics. I fit the model setting the validation split as 0.2 and batch size as 128. Since we are required to repeat more than 5 times with different random seeds, I train this model six times and each time with a different random seed from range (0, 6). I trained the model 80 epochs each time since from the results of Question 3, I think the model can stop underfitting in less epochs.

Answer to 4(a) : Redo 3(a) with CNN one layer

Below (Figure 5) are the six plots of average training cross-entropy error (y-axis) vs. the epoch number (x-axis). From my observation, the trends of all six figures with different seeds are significantly similar. The vertical purple line in the graph represents the best epoch, which is determined by the minimum validation loss, and from the plots, we can easily observe that the best epoch for all plots are between 70 and 80. From the best epoch line, we can observe that it becomes overfitted after epochs around 75.

The conclusion is similar as in 3(a). The reason these plots look a little different from the plots in 3a is that the range of the y labels has changed. In general, validation loss is smaller than training loss at the beginning. However, training errors keep decreasing, but validation errors decrease to a minimum and start to become steady and slightly increase. So, the training errors are smaller than the validation errors after epochs around 10. However, for this CNN model, we need more epoch to get the best epoch than the single layer model and the beginning training loss is very large.

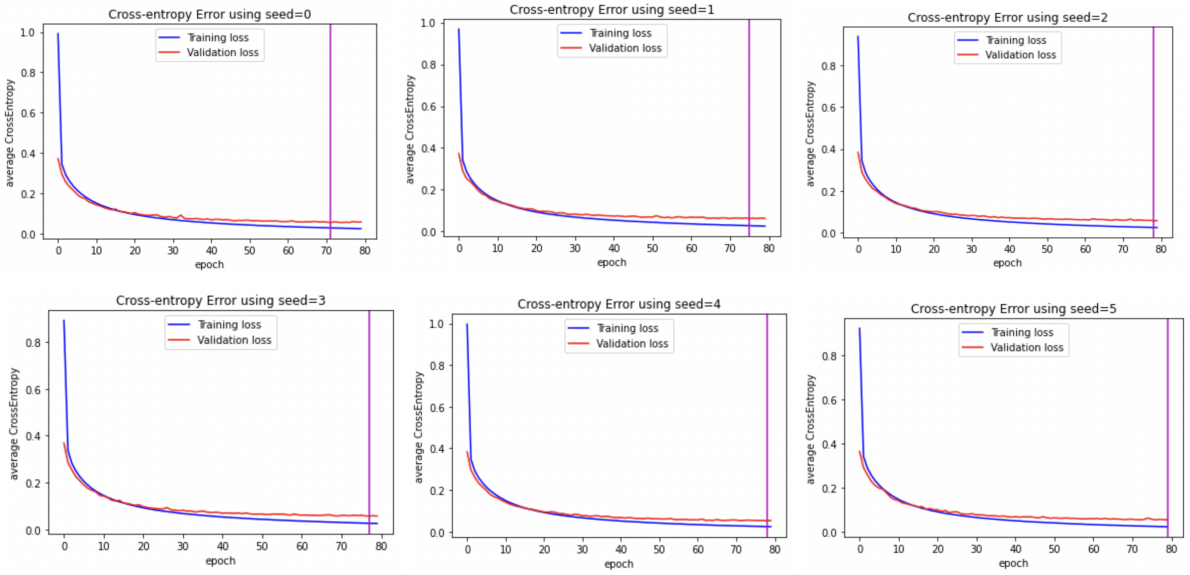


Figure 5: Average Cross-entropy Error v.s. Epoch for different seeds

Answer to 4(b) : Redo 3(b) with CNN one layer

I make the mean mis-classification plots as I did in 3(b).The best epoch is measured by the maximum of accuracy, which is the same as the minimum of misclassification error.

The plots (Figure 6) are similar to the plots in 4(a). The trends in Figure 6 are very similar to that in Figure 5 except the range of y since y represents the error in percentage. Moreover, the best epoch is a little different from these two kinds of error plots. We conclude the same result as 3(b).

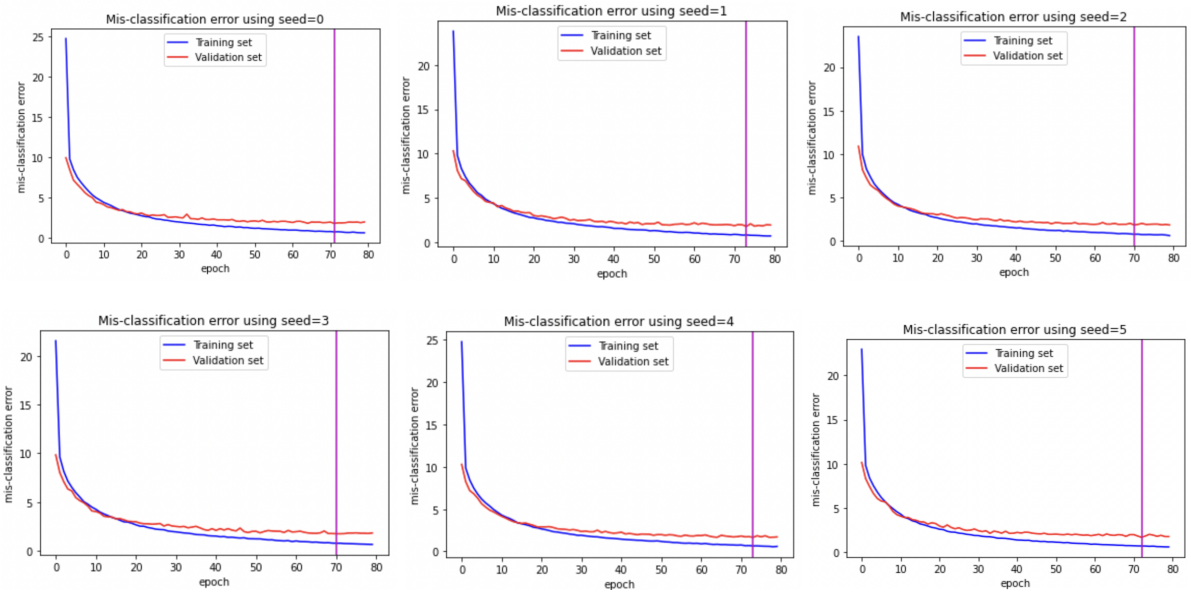


Figure 6: Mis-classification error (%) v.s. Epochs for different seeds

Answer to 4(c) : Redo 3(c) with CNN one layer

Just as in 3(c), I generate the accuracy number and choose the seed with the highest accuracy as my best results. According to the accuracy, I choose the 5th model with random seed 4, which has the highest accuracy 0.9833, as my best model.

```
[0.9819999933242798,
 0.9824000000953674,
 0.9817000031471252,
 0.9825000166893005,
 0.983299970626831,
 0.9830999970436096]
```

Table 2: Accuracy of different models

Since I set my kernel size as (3, 3) and filters as 32, I visualize my best results of the learned W as thirty-two 3×3 images. I set the color as black and white. Compared to 3(c), our plot (Fig.7) becomes more abstract and it is not interpretable from human understanding. This image contains some abstract features for which we have no words or mental concept.

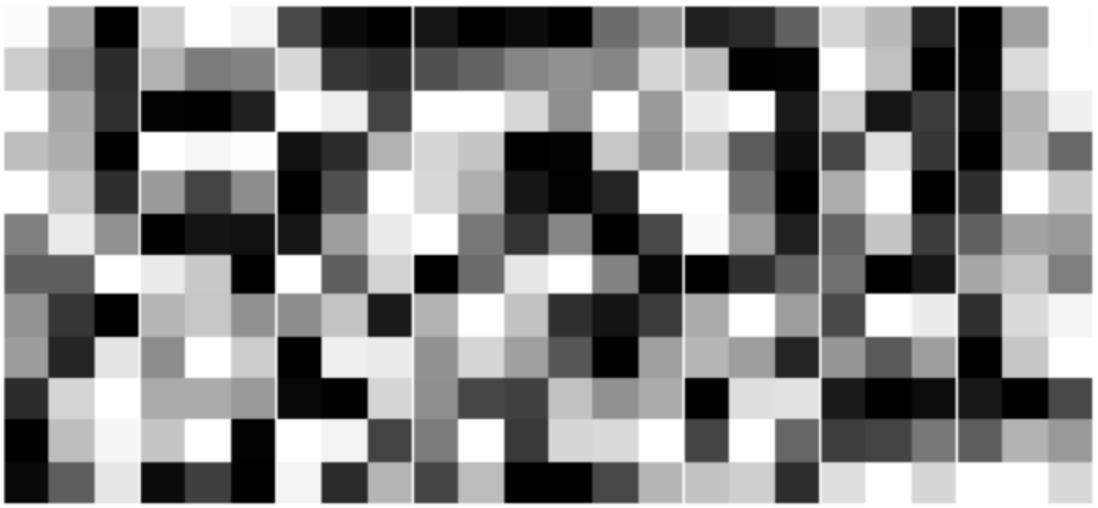
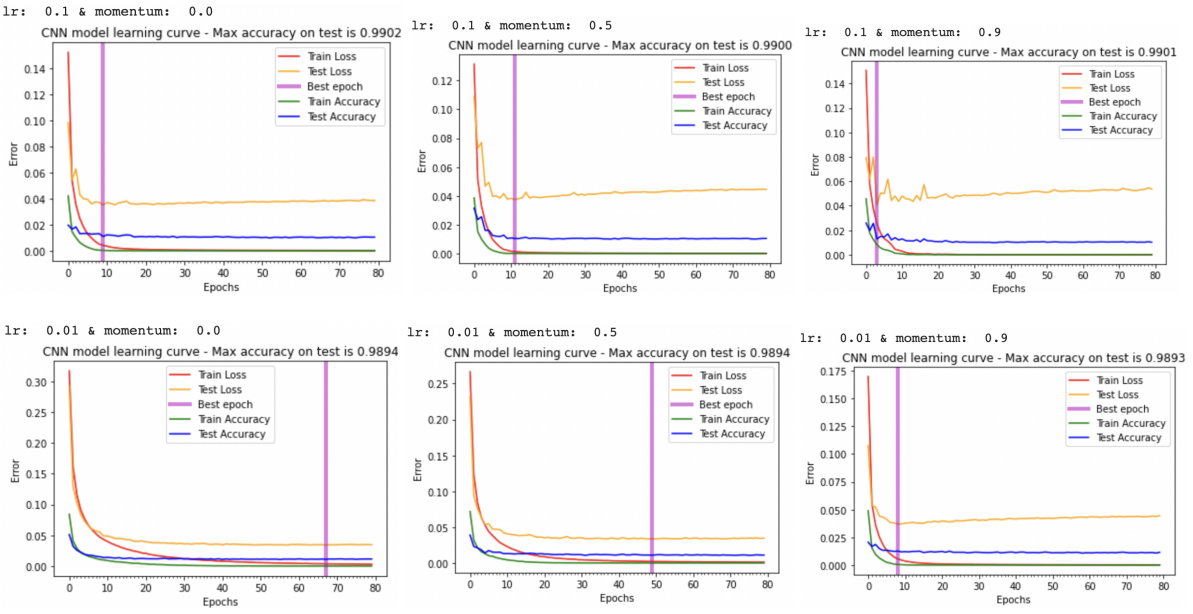


Figure 7: Feature map

Answer to 4(d) : Redo 3(d) with CNN one layer

As I did in 3(d), I draw a total of 12 different plots to observe the convergence properties and visualize the results with the best epoch line measured by the minimum test loss. I still focus on the error in the test set, which are the yellow and blue lines.

By observing the plots (Fig.8), we can conclude the similar argument as 3(d). The best epoch line moves toward y axis if the momentum increases at fixed learning rate or the learning rate increases at fixed momentum. Also, the plot of learning rate is 0.5v.s. momentum is 0.9 is still weird. So, we can conclude, whether for a single layer model or simple CNN model, the learning rate and momentum affect the convergence rate. My choice of best parameter would still be learning rate is 0.2 and momentum is 0.5. I think in that case I can get the small test error in reasonable epoch number and would not be overfitting a lot with the increase of the epoch.



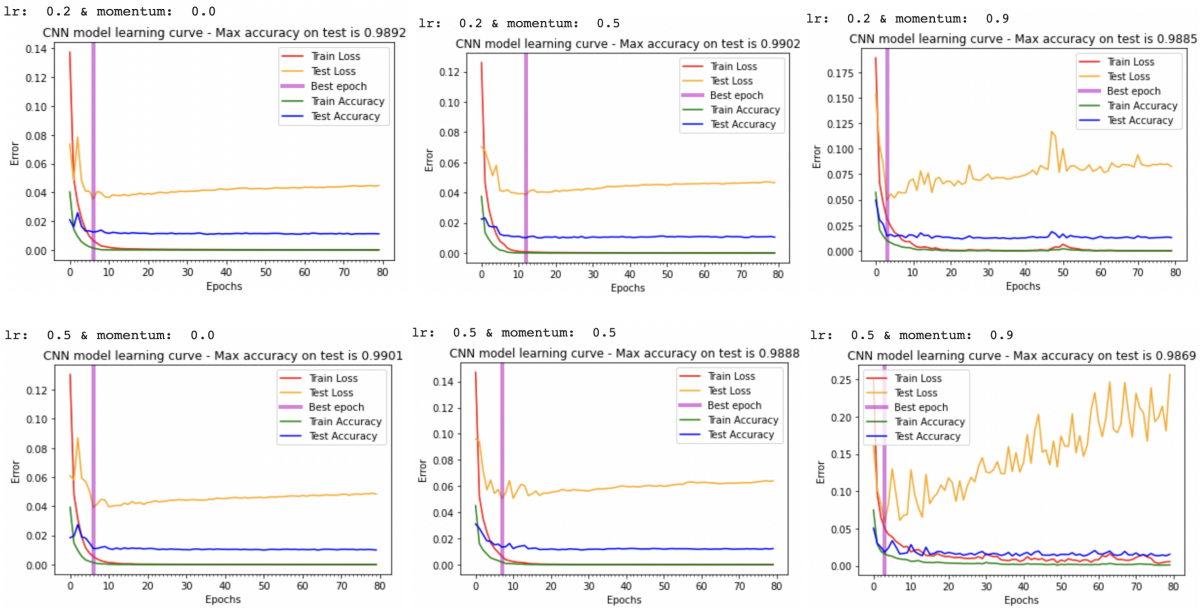


Figure 8: Plots of different combinations of learning rate and momentum

Question 5 Favorite CNN Model

```
def bestcnn(seed):
  model = Sequential()
  model.add(Conv2D(filters=32, kernel_size=(3,3),
                   activation='relu', input_shape=(28, 28, 1)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Flatten())
  model.add(Dense(100, activation='relu'))
  model.add(BatchNormalization())
  model.add(Dense(10, activation='softmax'))
  # compile model
  np.random.seed(seed)
  opt = SGD(learning_rate=0.01, momentum=0.9)
  model.compile(optimizer=opt,
                loss=losses.CategoricalCrossentropy(),
                metrics=['accuracy'])
  return model
```

For this part, I add more layers than the model in Question 4. I added one more 2-D convolutional layer with ReLU activation function, kernel size (3 ,3) and 64 filters. Then add Maxpooling with (2, 2). I also added the batch normalization layer to help me accelerate training. I compile the model using SGD with 0.1 learning rate and 0.9 momentum as the optimization function, cross entropy as the loss function, accuracy as the matrics. I fit the model setting the validation split as 0.2 and batch size as 128. Since we are required to repeat more than 5 times with different random seeds, I train this model six times and each time with a different random seed from range (0, 6). I train this model 10 epochs each time since I set momentum to 0.9. From previous observation, I predict the model can achieve good test error in small epochs.

Answer to 5(a) : Redo 3(a) with CNN more layers

Just as I did in 3(a) and 4(a), the trends of all six figures with different seeds are significantly similar. The vertical purple line in the graph represents the best epoch, which is determined by the minimum validation loss. By observing the best epoch line, we can see 2 plots have the best epoch in less than 10 epochs. So, even though the other 4 plots may not reach the minimum test error in 10 epochs because of the initial seed difference, I predict the model is going to stop underfitting just after a few more epochs. The minimum value of the test error

among all 6 models does not have too much difference just because of the initial seed difference. So, I believe 10 epochs is a reasonable choice. Even though we only have 10 epochs, it is easy to see that the performance of this model on training and validation sets are similar as in previous models. The average cross-entropy error on the training set is high in the beginning and keeps decreasing with the increased number of epochs. The test error is lower than the training error at the beginning but it starts to become steady and increases after reaching a minimum value.

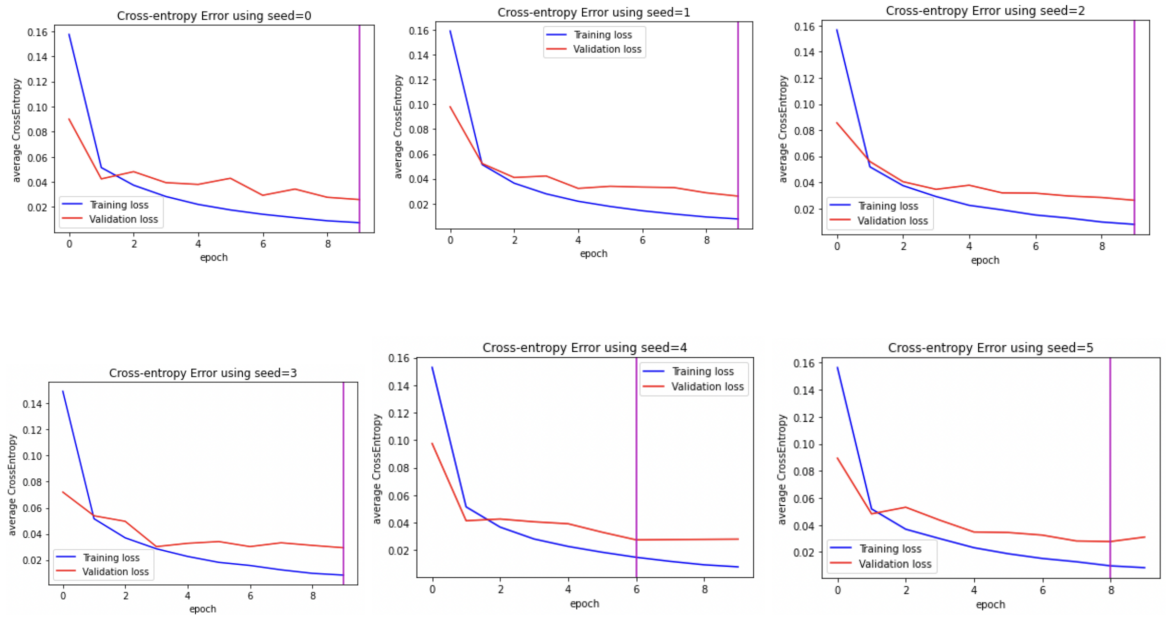


Figure 9: Average Cross-entropy Error v.s. Epoch for different seeds

Answer to 5(b) : Redo 3(b) with CNN more layer

I make the mean mis-classification plots as I did previously. The best epoch is measured by the maximum of accuracy, which is the same as the minimum of misclassification error.

The plots (Figure 10) are similar to the plots in 4(a). The trends in misclassification error plots are very similar to that in cross-entropy error plots except the range of y since y represents the error in percentage.

There is one noteworthy thing that both train and test error at the beginning, which is also the maximum error, is smaller than previous models. So, I predict that the mean error in this model must be smaller than previous models.

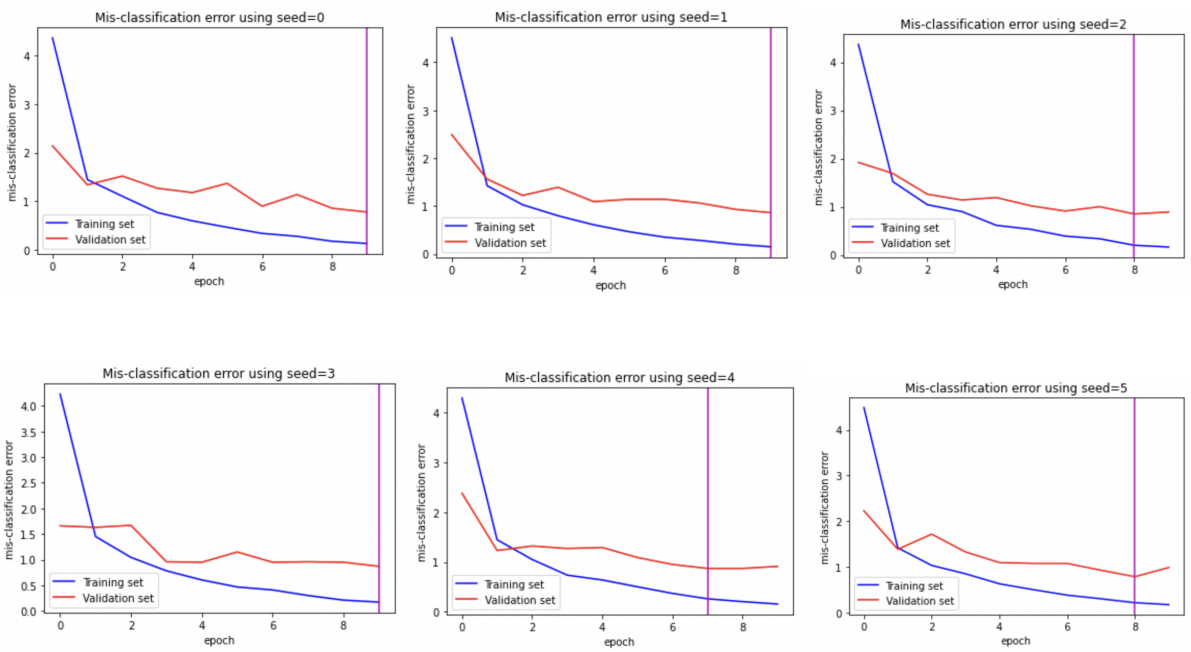


Figure 10: Mis-classification error (%) v.s. Epochs for different seeds

Answer to 5(c) : Redo 3(c) with CNN more layer

As before, I generate the accuracy number and choose the seed with the highest accuracy as my best results. According to the accuracy, I choose the 1st model with random seed 0, which has the highest accuracy 0.9922, as my best model.

```
[0.9922000169754028,
 0.9914000034332275,
 0.9915000200271606,
 0.9912999868392944,
 0.9912999868392944,
 0.9921000003814697]
```

Table 3: Accuracy of different models

Since I set my kernel size as (3, 3) and filters as 32, I visualize my best results of the learned W as thirty-two 3×3 images. I set the color as black and white. Compared to 3(c), our plot (Fig.7) becomes more abstract. This image includes some abstract features for which we cannot give explicit explanations. This image is a little different from images in 4(c), but just as images in 4(c), it is not interpretable from human understanding.

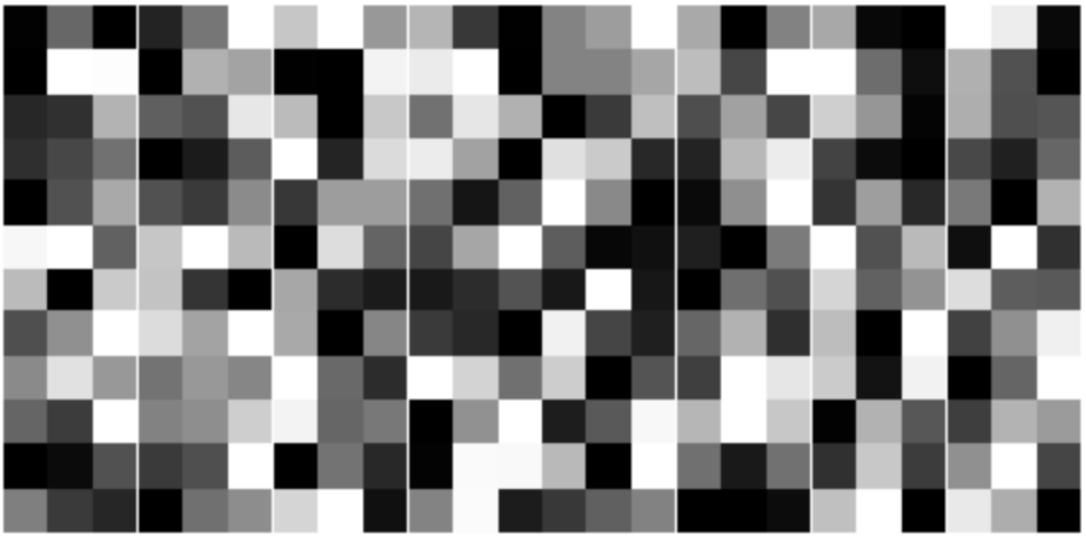


Figure 11: Feature map

Answer to 5(d) : Redo 3(d) with CNN more layer

As I did in 3(d), I draw a total of 12 different plots to observe the convergence properties and visualize the results with the best epoch line measured by the minimum test loss, which is the yellow line in the plot. I still focus on the error in the test set, which are the yellow and blue lines.

I assume the statement we conclude in 3(d) adn 4(d) is still valid in this model. By observing the plots (Fig.12), we can see that some of them have the best epoch at less than 10. To see the exact number of the errors, I made a table (Table 4) to help me compare the results. Combined with both tables and plots, I would choose learning rate 0.01 and momentum 0.9 as my favorite parameters. Since in that case, the model has the lowest mean misclassification error rate and relatively small min misclassification error. From the table we can see, this model can absolutely beat the performance of SVM with Gaussian Kernel. Choosing a reasonable learning rate and momentum combination would make our model have a test error rate lower than 1.4%.

Min Misclassification error in percentage(%)

Momentum / Learning Rate	0.0	0.5	0.9
0.1	0.84000	0.83000	1.10000
0.01	1.36000	1.17000	0.86000
0.2	0.82000	0.91000	1.03000
0.5	0.77000	1.10000	1.29000

Mean Misclassification error in percentage(%)

Momentum / Learning Rate	0.0	0.5	0.9
0.1	1.47300	1.33500	1.68900
0.01	2.06400	1.60700	1.31600
0.2	1.33500	1.33600	1.90800
0.5	1.43900	1.52600	2.32800

Table 4: Mean and Min Errors(%) for different models

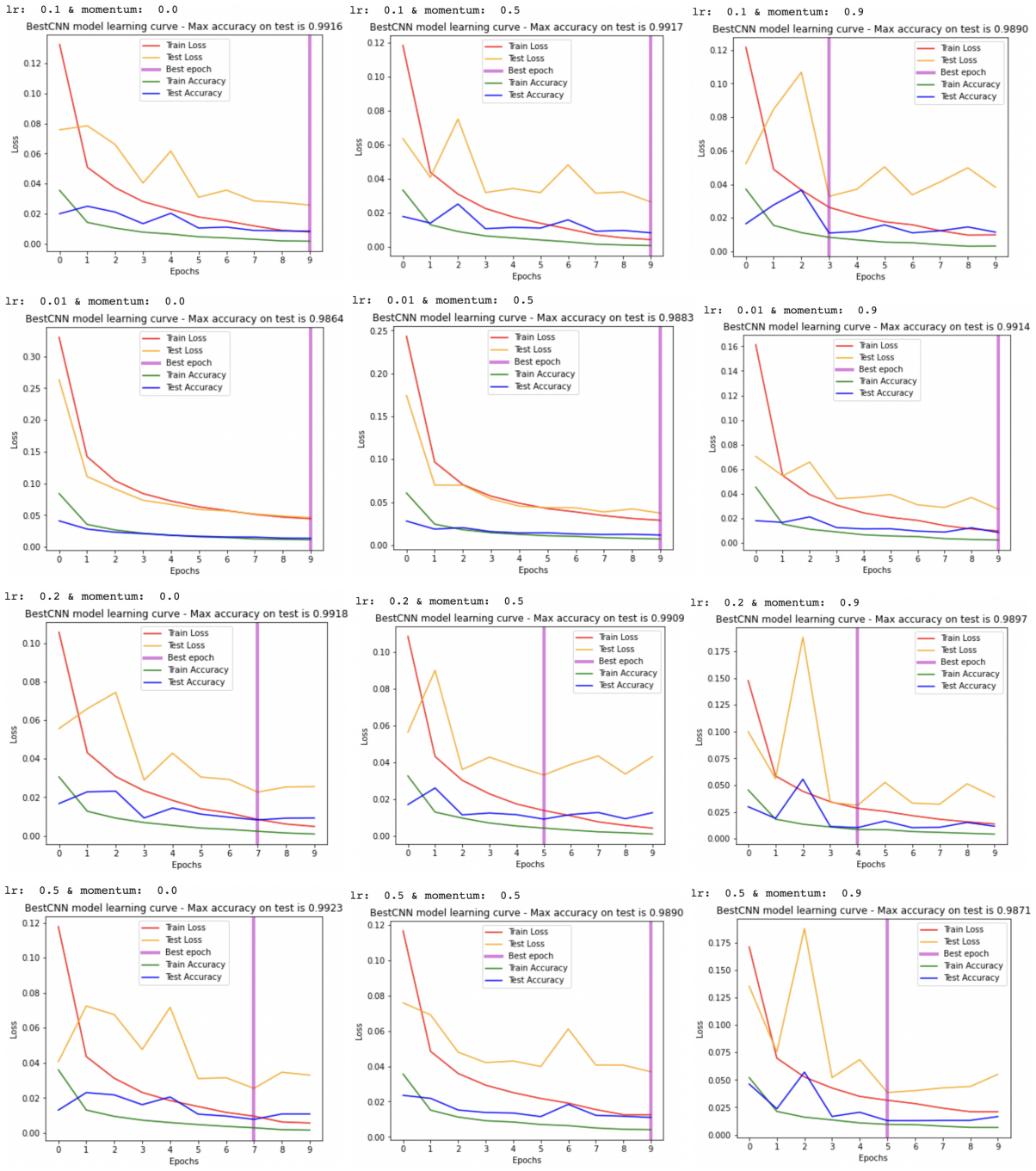


Figure 12: Plots of different combinations of learning rate and momentum

Question 6

As a warm up question, load the data and plot a few examples. Decide if the pixels were scanned out in row-major or column-major order. What is the relationship between the 2 digits and the last coordinate of each line?

Answer to 6

I plot the first five examples from the train dataset as shown in Fig.13. From this plot, we can see that when reading each line, the numbers from the digits row wise. So, I believe that the pixels were scanned out in row-major. What's more, the two digits are located side by side. The last coordinate of each line should be the sum of these digits since the 5 last coordinates of each of these 5 plots are: 5, 5, 4, 16, 12. So, I conclude that the last coordinate of each line is the sum of two digits.

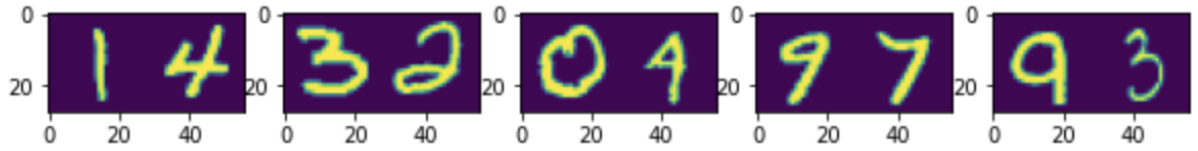


Figure 13: Plots of digits in train

Question 7

```
def model_7(seed):
  model = Sequential()
  model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(28,56,1)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.25))
  model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.25))
  model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.25))
  model.add(Flatten())
  model.add(Dense(100, activation='relu'))
  model.add(Dropout(0.25))
  model.add(Dense(19, activation='softmax'))
  # compile model
  np.random.seed(seed)
  opt = SGD(learning_rate=0.01,momentum = 0.9)
  model.compile(optimizer=opt, loss=losses.CategoricalCrossentropy(), metrics=['accuracy'])

  return model
```

For this part, I use train.txt to train my model. I added a 2-D convolutional layer with ReLU activation function, kernel size (3 ,3) with different filters, Maxpooling with (2, 2), and dropout layers to help me accelerate training. I compile the model using SGD with 0.01 learning rate and 0.9 momentum as the optimization function, which is the best parameter I found in Question 5.I train the model using val.txt. Since we are required to repeat more than 5 times with different random seeds, I train this model six times and each time with a different random seed from range (0, 6). I train this model 20 epochs each time since I set momentum to 0.9. From previous observation, I predict the model can achieve good test error in small epochs.

Answer to 7a and 7b

To save time, I draw both cross-entropy and mis-classification errors in the same plots.

For 7a: Fig14 is different from previous plots, the test errors here keep decreasing and do not reach to a minimum. What's more, both errors on the test set are always smaller than those on the training set.

For 7b: From Fig.14, we can see that the trends of both errors are similar. So, I do not observe a different behavior compared to the behavior of the cross-entropy error function except the numerical range in the y axis is different.

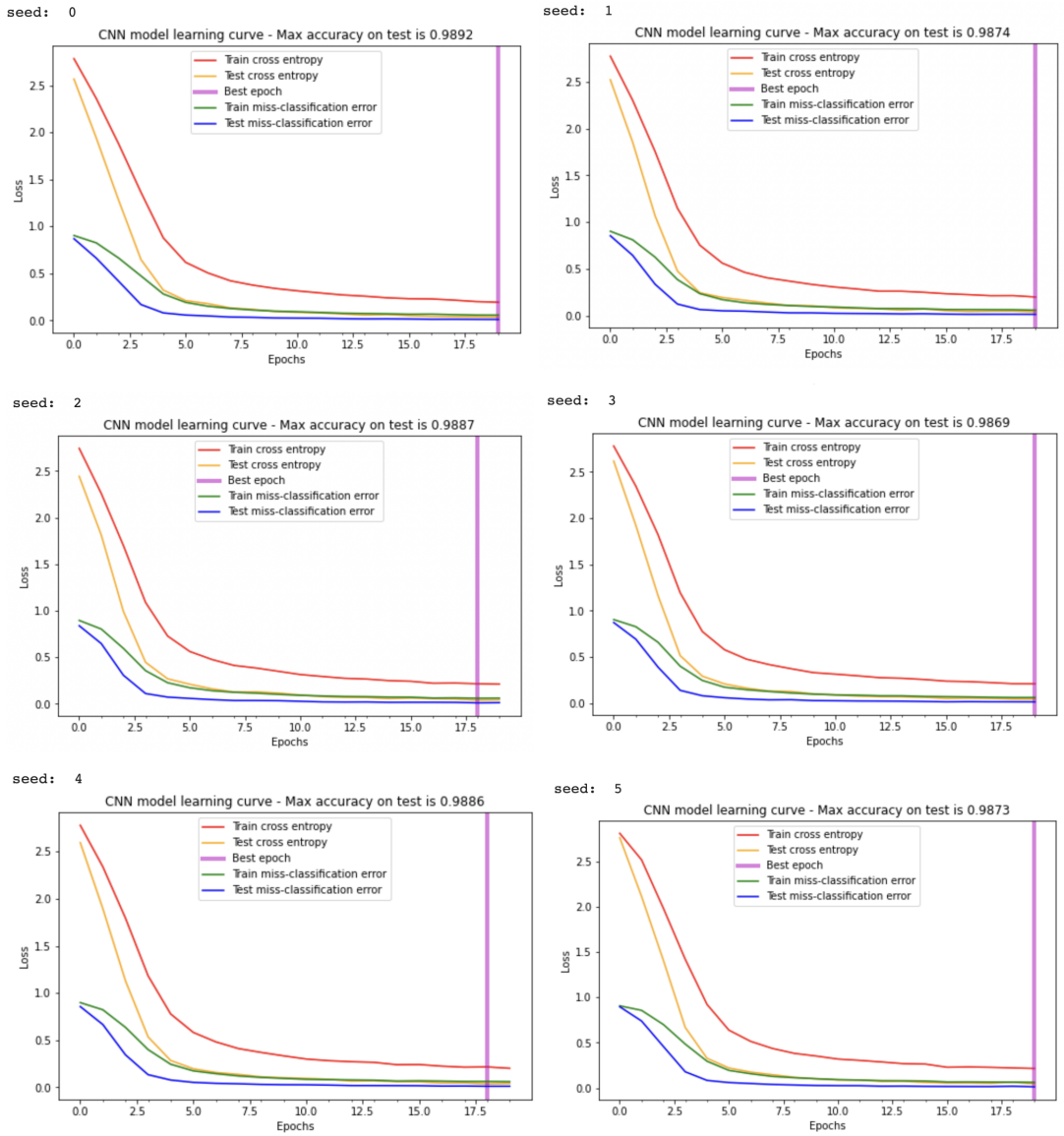


Figure 14: Plots of Errors

Answer to 7c

From Fig.14, we can see when seed is 0, the model has the highest max accuracy. So, I think a model with seed 0 will have the best results of the learned W. I am going to use seed 0 for the following parts of this question. Fig.15 is similar to the image in 4 (c) and5(c), which is not readable for human beings but contains abstract information.

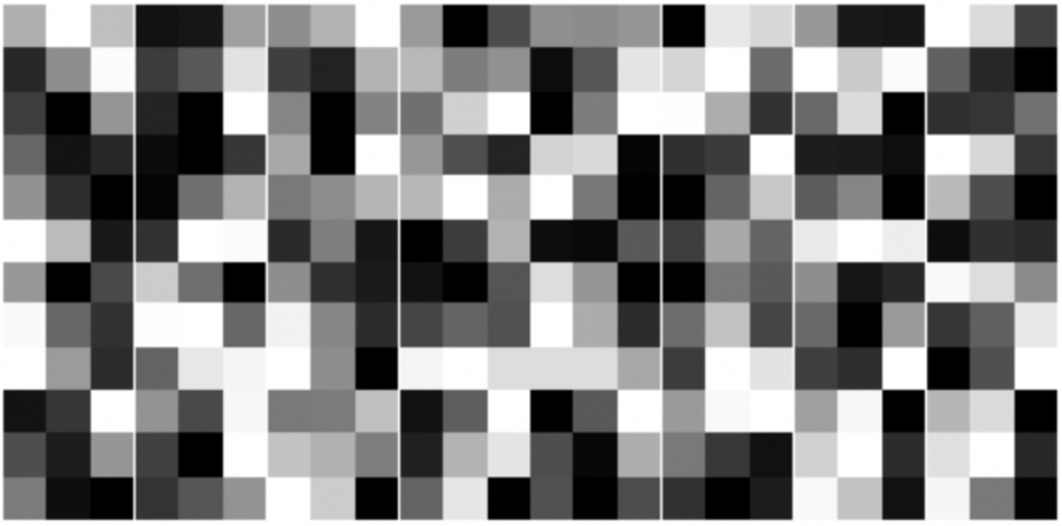


Figure 15: Feature map

Answer to 7d

I generate below three plots (Fig.16) after running all 12 combinations of learning rate and momentum.

As the learning rate increases with momentum fixed, the onset loss based on cross-entropy and the onset mis-classification error both decrease. The best epoch with highest validation accuracy appears earlier, which means my model takes fewer and fewer epochs to reach overfitting. However, since I only run 20 epochs, the models may still be underfitting, so we cannot make a conclusion here. From the accuracy table, we can see that the mean misclassification error is around twelve percent but the minimum of the misclassification error is small. For this model, different learning rate and momentum does not affect the performance or the converge rate a lot.

Min Misclassification error in percentage(%)

Learning Rate \ Momentum	0.0	0.5	0.9
0.1	1.22000	1.31000	1.17500
0.01	1.05000	1.37000	1.15500
0.2	1.26000	1.19000	1.08000
0.5	1.19500	1.26000	1.17500

Mean Misclassification error in percentage(%)

Learning Rate \ Momentum	0.0	0.5	0.9
0.1	12.52200	12.93375	12.10050
0.01	12.50250	13.80450	12.96800
0.2	11.04425	13.51600	12.61275
0.5	13.09500	12.82450	11.60550

Table 5: Mean and Min Errors(%) for different models

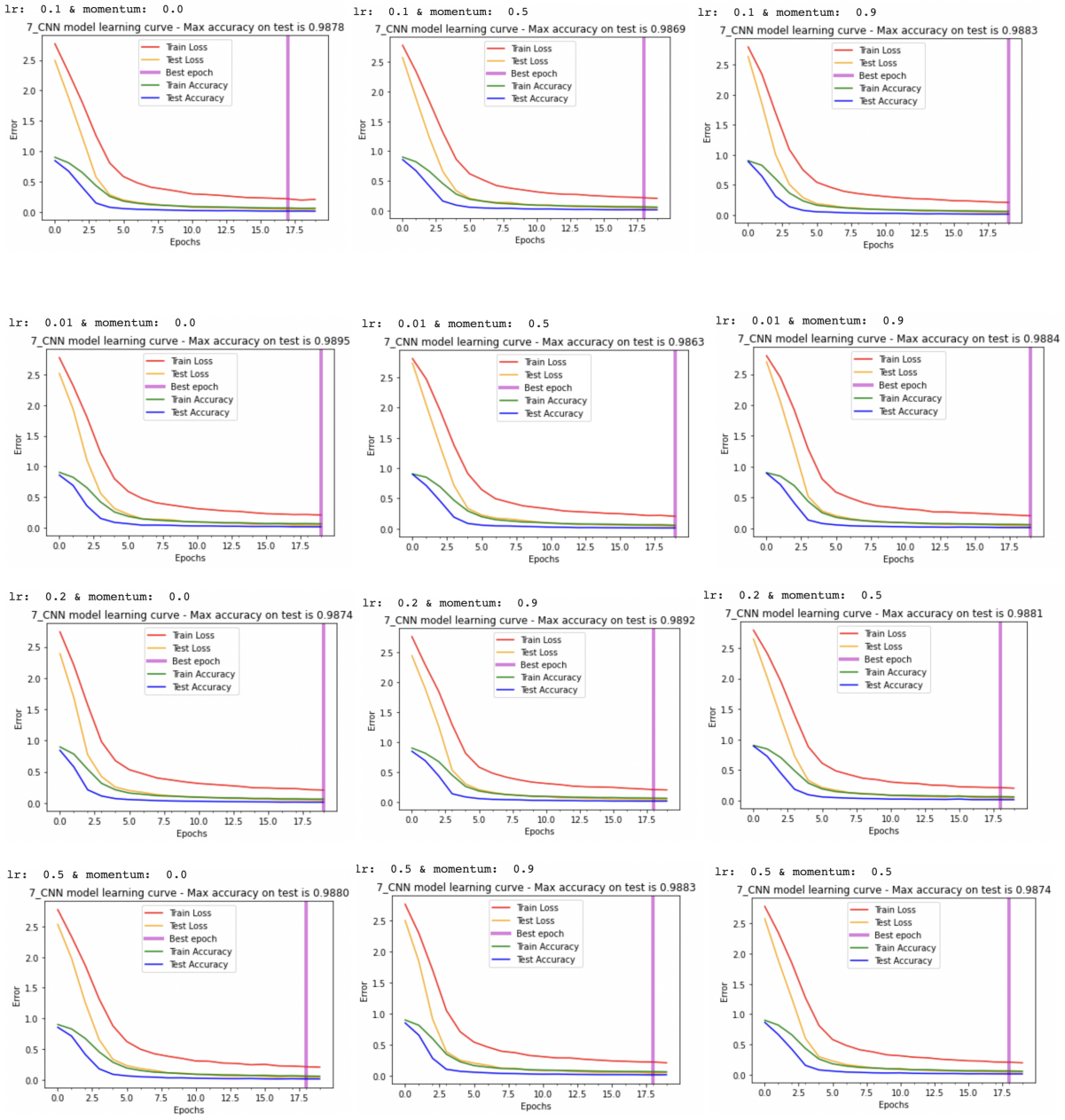


Figure 16: Plots of different combinations of learning rate and momentum

(c)

Report the generalization error (i.e., the performance on test.txt) for the model you picked.How would you compare the test errors you obtained with respect to the original MNIST data? Explain why you cannot obtain a test error lower than 1%.

I use a model with the random seed 0, learning rate 0.01 and momentum 0 to evaluate the test dataset. Table 6 shows my result about the test error in my model. The accuracy is 0.8228, which means the model I picked may not perform very well. This accuracy is much lower than the one I obtained from the original MNIST data. I think the reason behind this may be that the epoch is too small in my model. The dataset itself may have some problems. The starting misclassification error is too high to achieve a low average error.

```
model.evaluate(testX, testY)
```

```
625/625 [==============================] - 7s 11ms/step - loss: 0.6848 - accuracy: 0.8228
```

Table 6: Test Error

I also tried to fit 100 epochs for my model, Figure 17 shows that the validation test error is always lower than the train set, which is weird. So I think either the dataset has some problem or I need to try more epochs to see if the model can overfit.

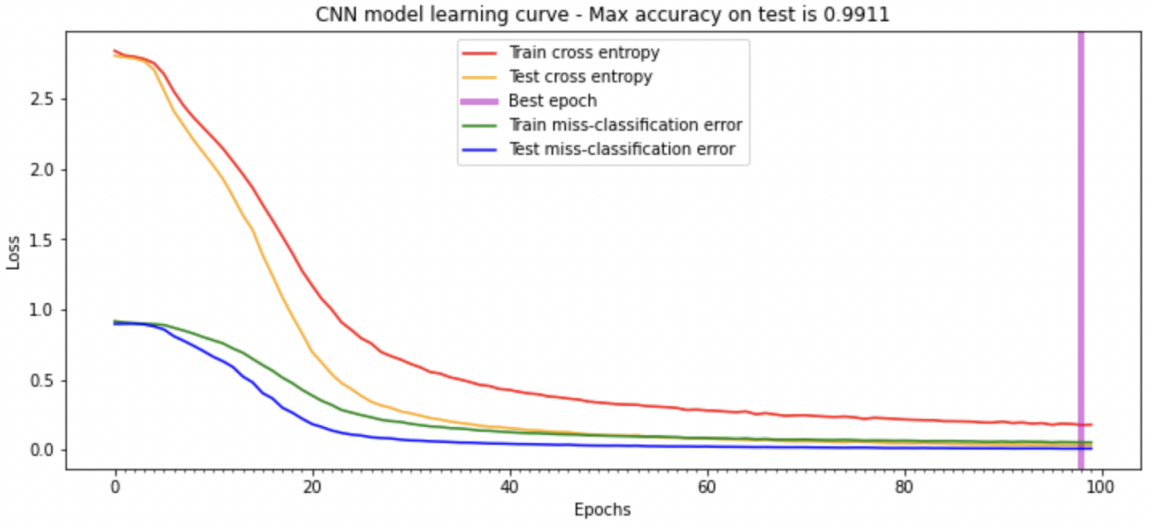


Figure 17: Plots of 100 epochs