GR5241 Spring 2022

Project Milestone 2: Deep Learning Part

Name: Kangshuo Li UNI: kl3259

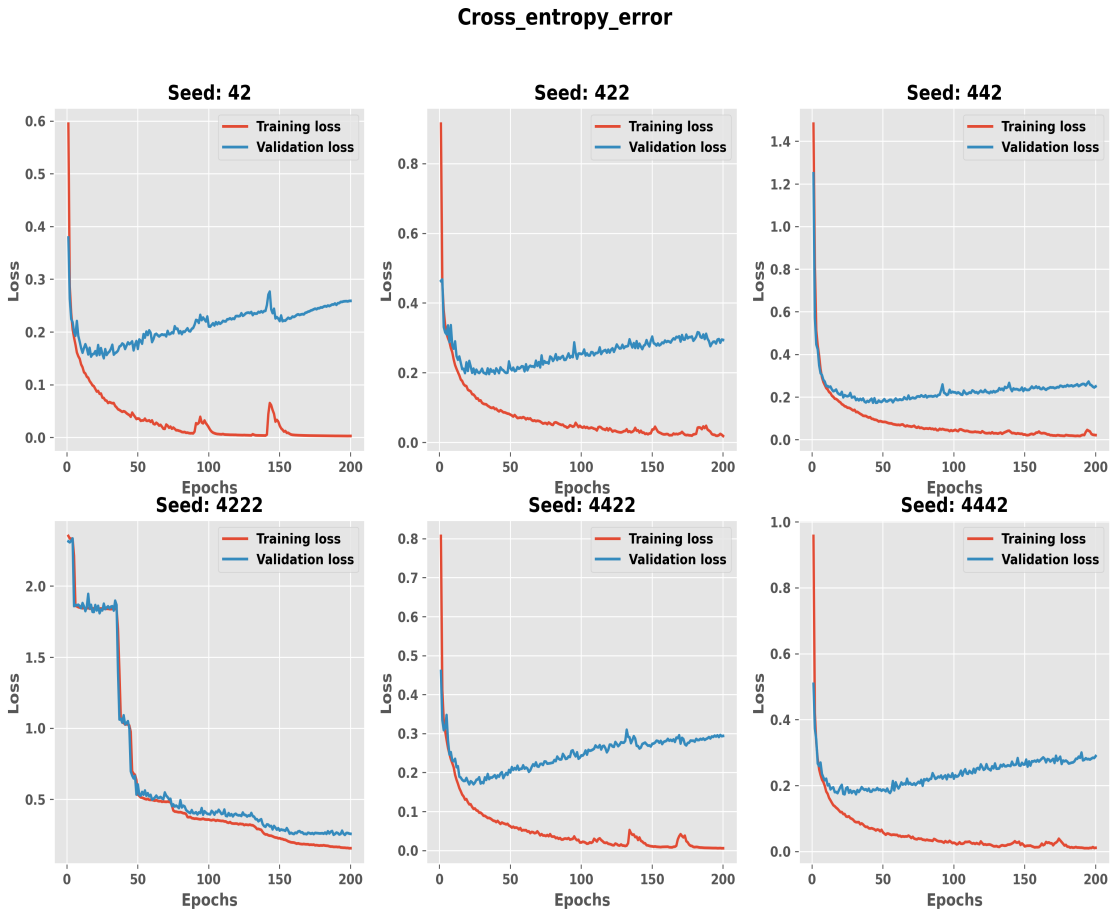
Part 4: Deep Learning

3. (5 points) Train a single layer neural network with 100 hidden units (e.g. with architecture: 784 → 100 → 10). You should use the initialization scheme discussed in class and choose a reasonable learning rate (i.e. 0.1). Train the network repeatedly (more than 5 times) using different random seeds, so that each time, you start with a slightly different initialization of the weights. Run the optimization for at least 150 epochs each time. If you observe underfitting, continue training the network for more epochs until you start seeing overfitting.

```
The structure of single layer neural network is:
single_layer_NN(
(flatten): Flatten(start_dim=1, end_dim=-1)
(single_layer_nn): Sequential(
(0): Linear(in_features=784, out_features=100, bias=True)
(1): Sigmoid()
(2): Linear(in_features=100, out_features=10, bias=True)
)
)
```

In this case we only construct one hidden layer with 784 original inputs, 100 neurons and the sigmoid function as activation function. The output size is 10 which corresponds to the number of classes in MNIST dataset.

(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 example) 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.



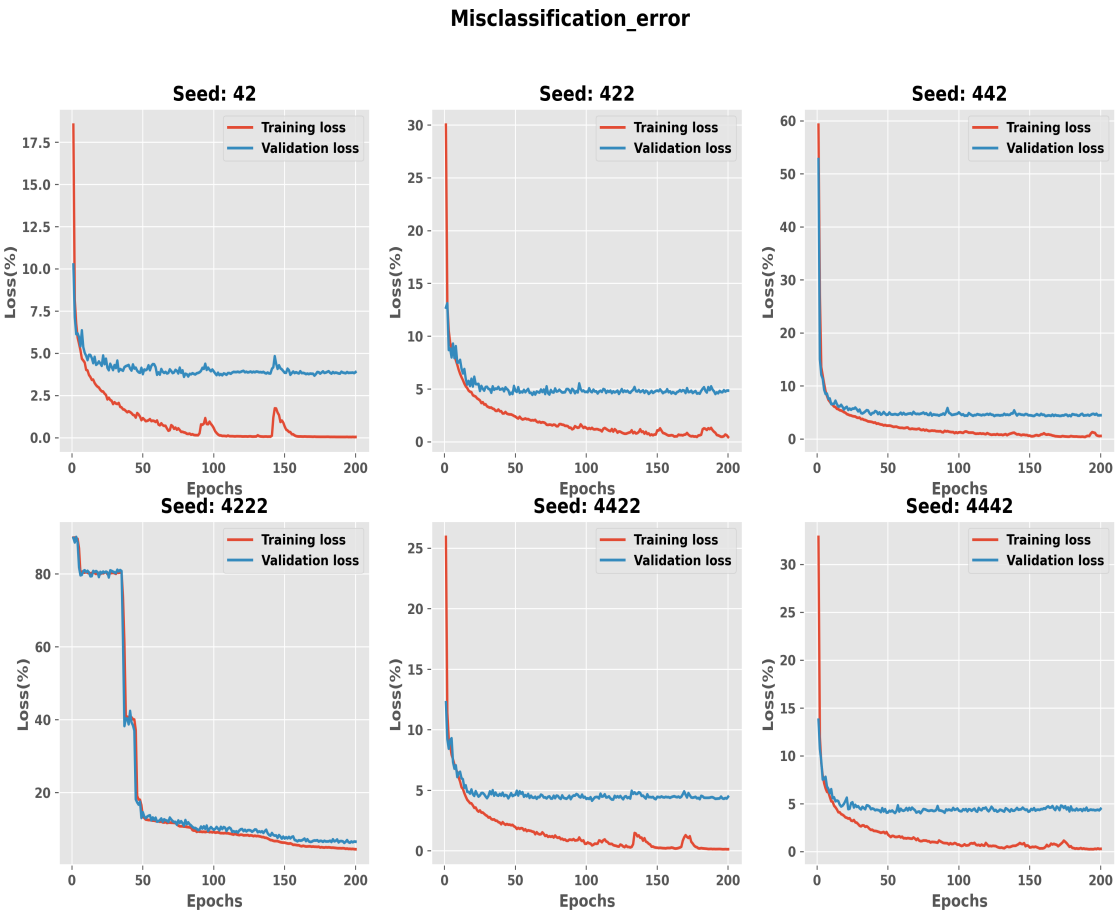
The leanring curves with respect to cross-entropy error of all 6 random initializations are shown above.

In this question, we set the learning rate $l = 0.1$, and $N_{epochs} = 200$ with batch size $N_{batch} = 64$. Since we used a setup that requires sufficient epochs to attain a status like overfitting, we can note that most of the training processes have a pattern of overfitting. The training errors are smaller than the test error. In axes of seed 42, 422, 442, 4422, and 4442, the cross-entropy loss started from different values with different initializations and decreased quickly in the first 25 epochs, then as the number of training epochs increased, the cross-entropy error slowly decreased to 0 with some fluctuations while the validation(test set here) cross-entropy error slightly increased from 0.2 to around 0.3, so these plots show that Their corresponding training process involves overfitting. In other words, the single neural networks began to become weaker for generalization while still working well on the training set.

Also note that the training process with seed 4222 got an unique learning curve, which includes a much higher initial cross-entropy loss and the learning process has several cliff drops and ends up with no significant overfitting pattern. This could be the result of a relatively unlucky initialization with parameters that were far from the local optimal point. Finally it also got the training error close to 0 and validation error close to 0.3, but the time cost to attained same performance is higher than the other random initializations.

(b) We could implement an alternative performance measure to the cross entropy, the mean miss-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?



The learning curves with respect to misclassification error of all 6 random initializations are shown above.

The learning curves of average misclassification error are pretty similar to learning curves based on cross-entropy error. They have the same quick decreasing sections in the first 25 epochs and overfitting sections corresponding to the remaining epochs, and all the fluctuations are at the same pace. Seed 4222 also got cliff drops in this learning curve and takes a longer time to earn the same performance as other neural networks.

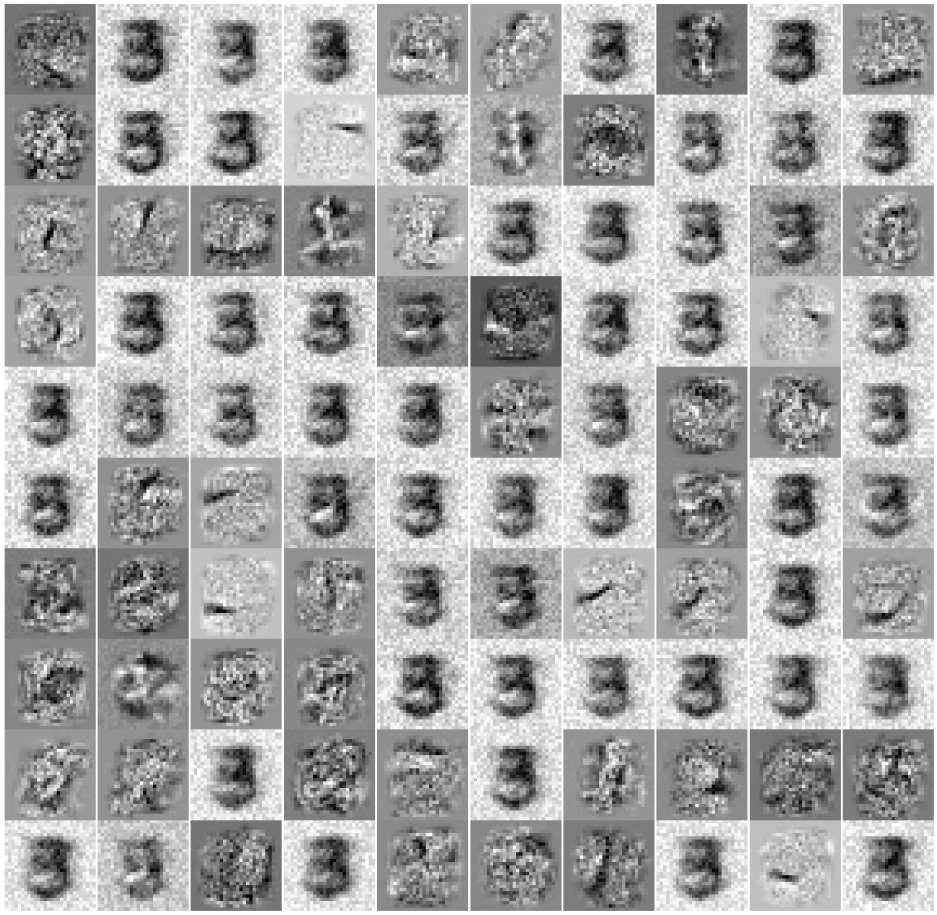
However, note that the overfitting parts of these plots are different from those in cross-entropy loss, the validation means misclassification error with seeds 42, 422, 442, 4422, and 4442 nearly maintaining the same value with some little noise in the overfitting phase, and the training loss of mean misclassification keep decreasing to 0. In the previous question, the cross-entropy error on test set was slowly increasing. If we measure the ability of generalization of the neural networks by using mean misclassification error, then this ability is not damaged by overfitting.

(c) Visualize your best results of the learned W as one hundred 28×28 images (plot all filters as one image, as we have seen in class). Do the learned features exhibit any structure?

	Cross_entropy_error_train	Misclassification_error_train	Cross_entropy_error_test	Misclassification_error_test	Accuracy_test
42	0.002937	0.045000	0.259231	0.0389	96.11
422	0.018808	0.451667	0.294355	0.0485	95.15
442	0.021964	0.583333	0.250466	0.0450	95.50
4222	0.156996	4.453333	0.258184	0.0653	93.47
4422	0.006148	0.131667	0.294285	0.0447	95.53
4442	0.011437	0.285000	0.289813	0.0447	95.53

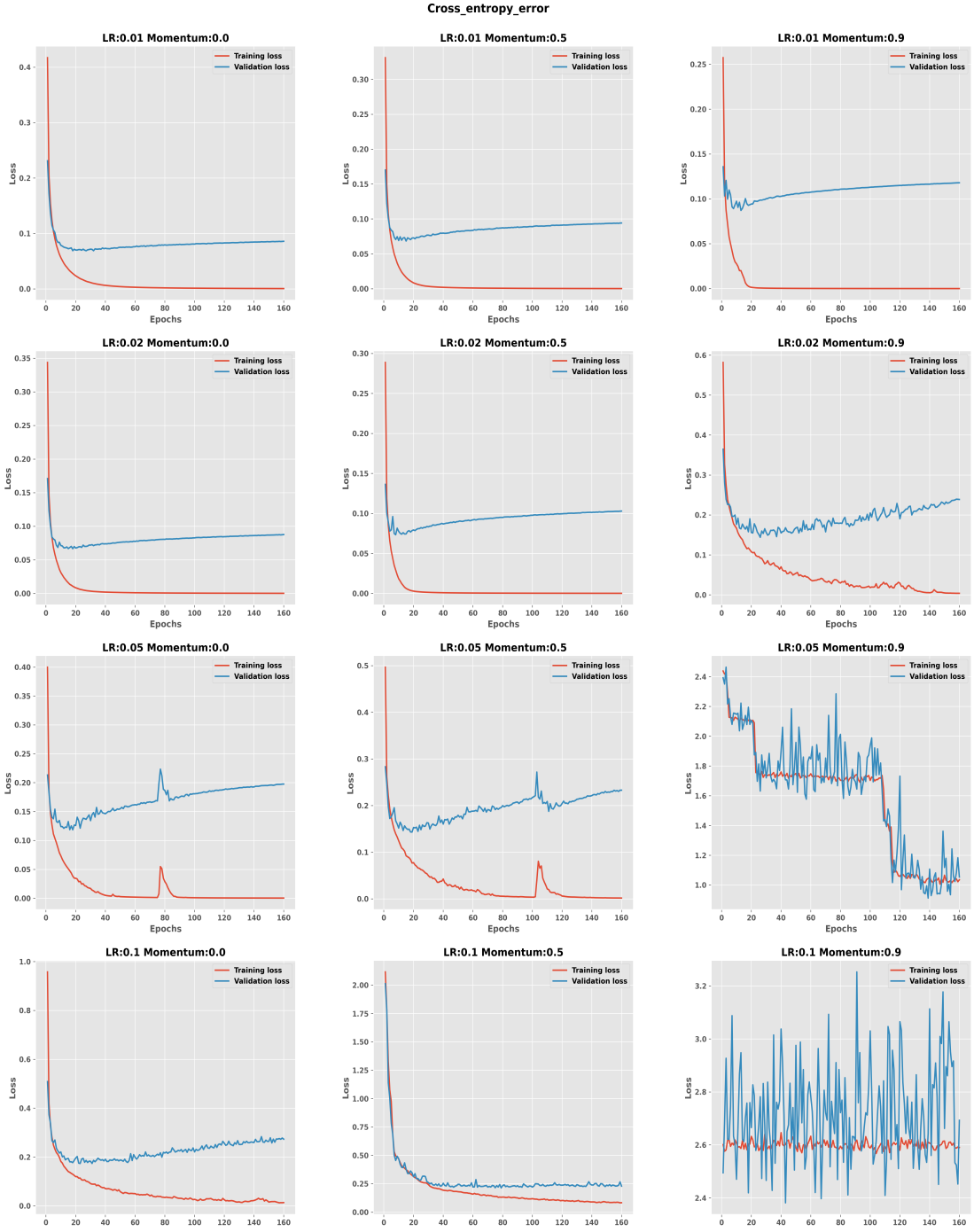
The table of evaluation metrics is shown above.

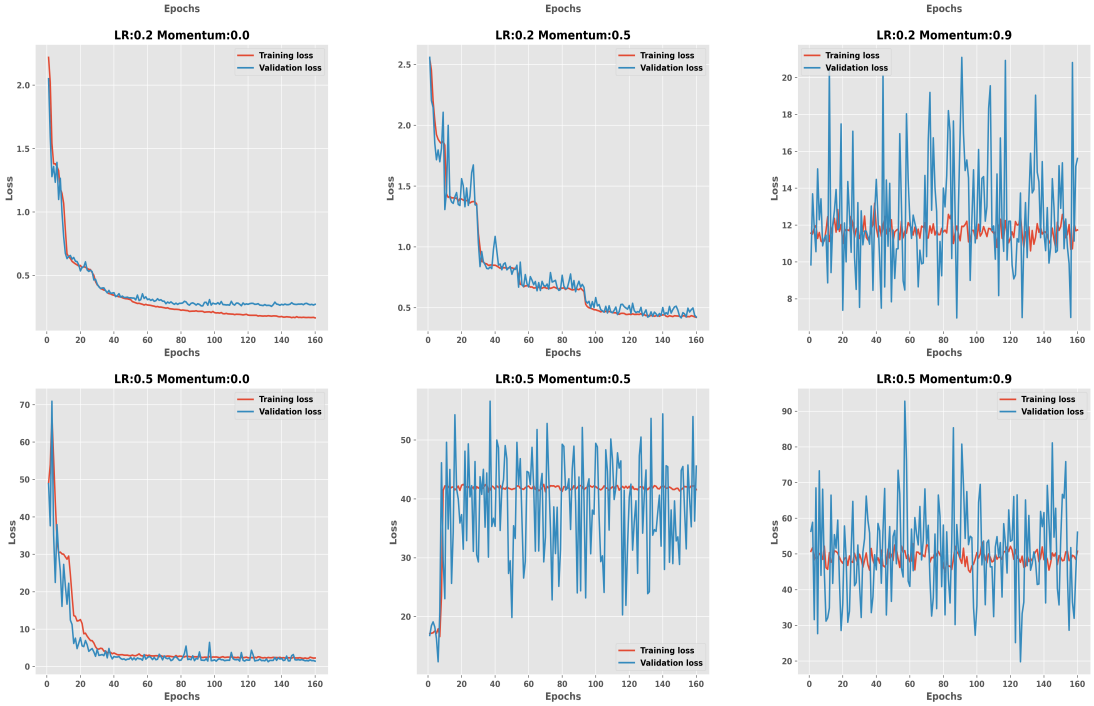
We select the best model based on mean misclassification error on test set standing for the test accuracy and the cross-entropy error on test set. Singel layer neural network with seed 42 has the lowest mean test misclassification error and relatively low test cross-entropy error, so we identify it as the best model.

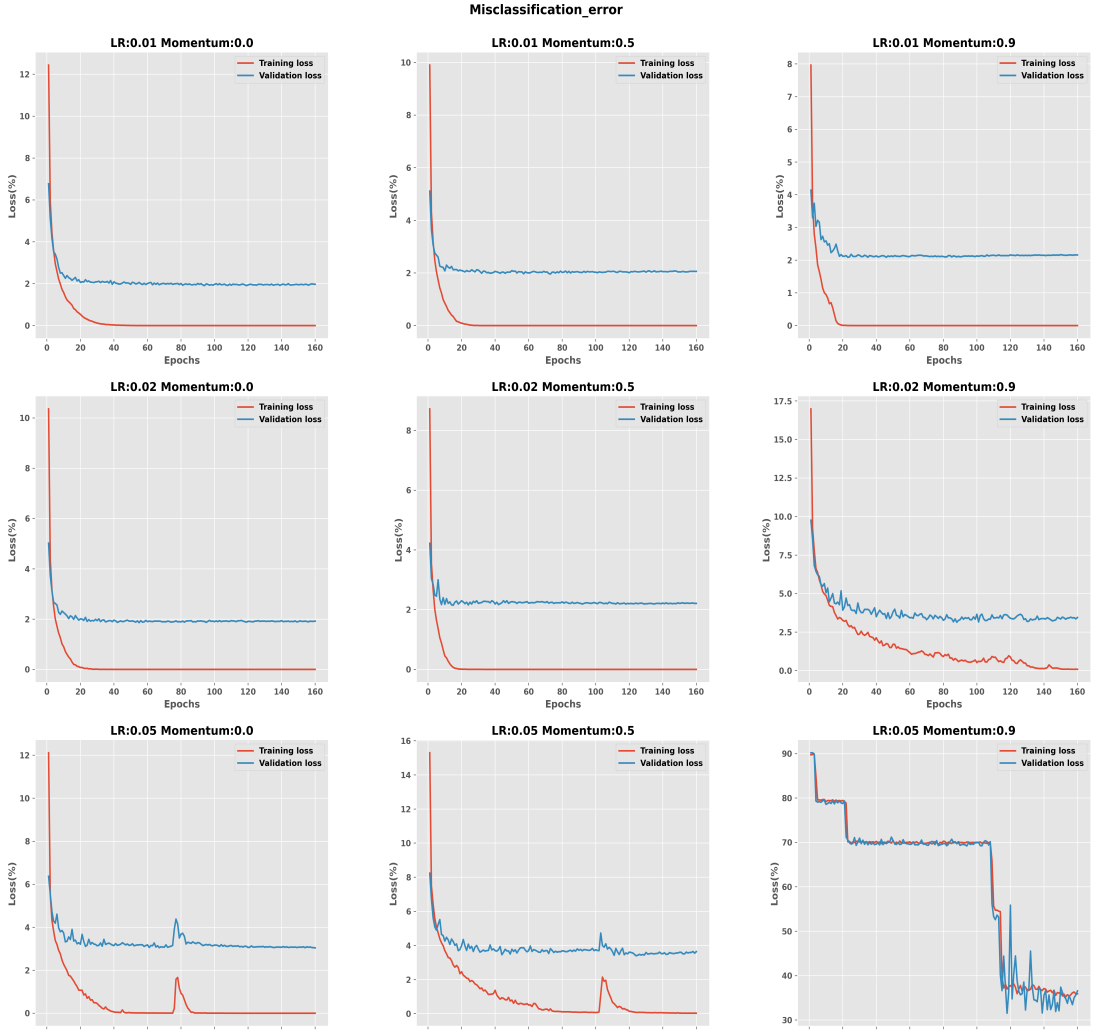


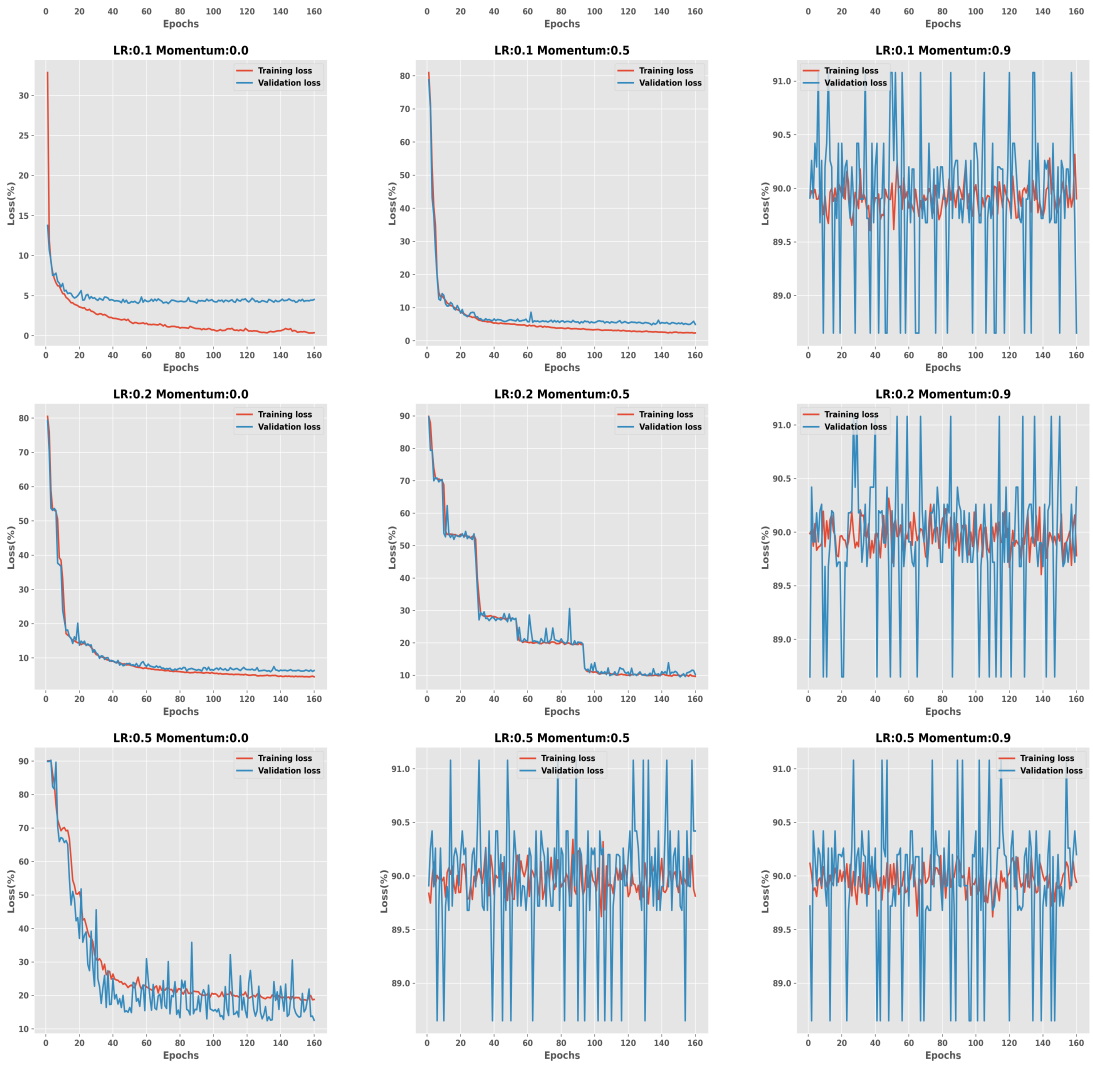
This is the visualization of the parameters learned from the best model with seed 42. It's clear that the most frequent pattern is the feature like a shape of "3" with a shade like a shape of "8". There are also some chaotic features and some features with only part of the number or strokes shown in the restored features. The patterns like "3" frequently appeared might be the result from severely overfitting, as the learning process led the weights to a local optimal that mainly recognizing "3" or "8".

(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?




Epochs
LR:0.2 Momentum:0.0
LR:0.2 Momentum:0.5
LR:0.2 Momentum:0.9
LR:0.5 Momentum:0.0
LR:0.5 Momentum:0.5
LR:0.5 Momentum:0.9
Training loss
Validation loss
Loss
Epochs


Misclassification_error
LR:0.01 Momentum:0.0
LR:0.01 Momentum:0.5
LR:0.01 Momentum:0.9
LR:0.02 Momentum:0.0
LR:0.02 Momentum:0.5
LR:0.02 Momentum:0.9
LR:0.05 Momentum:0.0
LR:0.05 Momentum:0.5
LR:0.05 Momentum:0.9
Training loss
Validation loss
Loss(%)
Epochs



We trained the best model with random seed 42 on the grid of learning rate $lr = {0.01, 0.02, 0.05, 0.1, 0.2, 0.5}$ and momentum $momemtum = {0.0, 0.5, 0.9}$.

According to these 2 figures, both cross-entropy error and mean misclassification error have the same learning pattern when the learning rate and momentum are the same. With fixed momentum, a larger learning rate would lead to higher error rate and higher gap between training and testing error in terms of both the cross-entropy error and misclassification error. With a fixed learning rate, as the momentum increases, we can see that the learning curves are becoming less stable and even cannot make optimization on the loss function or even diverge. One important thing here is that single layer neural networks with lower learning rates seem to have better convergence rates than those with large learning rates.

	Cross_entropy_error_train	Misclassification_error_train	Cross_entropy_error_test	Misclassification_error_test	Accuracy_test
0.01_0.0	0.000751	0.000000	0.086001	0.0197	98.03
0.01_0.5	0.000326	0.000000	0.094303	0.0206	97.94
0.01_0.9	0.000068	0.000000	0.117996	0.0216	97.84
0.02_0.0	0.000325	0.000000	0.087719	0.0192	98.08
0.02_0.5	0.000159	0.000000	0.103002	0.0221	97.79
0.02_0.9	0.004041	0.088333	0.238982	0.0345	96.55
0.05_0.0	0.000388	0.000000	0.197699	0.0304	96.96
0.05_0.5	0.001813	0.028333	0.233049	0.0364	96.36
0.05_0.9	1.035470	35.901667	1.054175	0.3657	63.43
0.1_0.0	0.013729	0.366667	0.273264	0.0452	95.48
0.1_0.5	0.083479	2.363333	0.229456	0.0496	95.04
0.1_0.9	2.590597	89.903333	2.693091	0.8865	11.35
0.2_0.0	0.165600	4.493333	0.272767	0.0631	93.69
0.2_0.5	0.421209	9.665000	0.422937	0.1024	89.76
0.2_0.9	11.736179	89.781667	15.614161	0.9042	9.58
0.5_0.0	2.295264	18.810000	1.450393	0.1258	87.42
0.5_0.5	41.569161	89.813333	45.569125	0.9042	9.58
0.5_0.9	50.769852	89.943333	56.105369	0.9020	9.80

This table shows the evaluation metrics of all single layer NNs trained on the learning rate and momentum grid with the best random seed 42 picked in 3(c). The best parameters should be chosen by the learning curve that have stable learning process and higher test accuracy and lower test cross-entropy error. Since the value of average cross-entropy error on test set is sensitive to overfitting, we can choose the best single layer neural network with parameters:

- $Seed = 42, lr = 0.01, momentum = 0.0$

since it got the lowest test average cross-entropy error and the highest test accuracy.

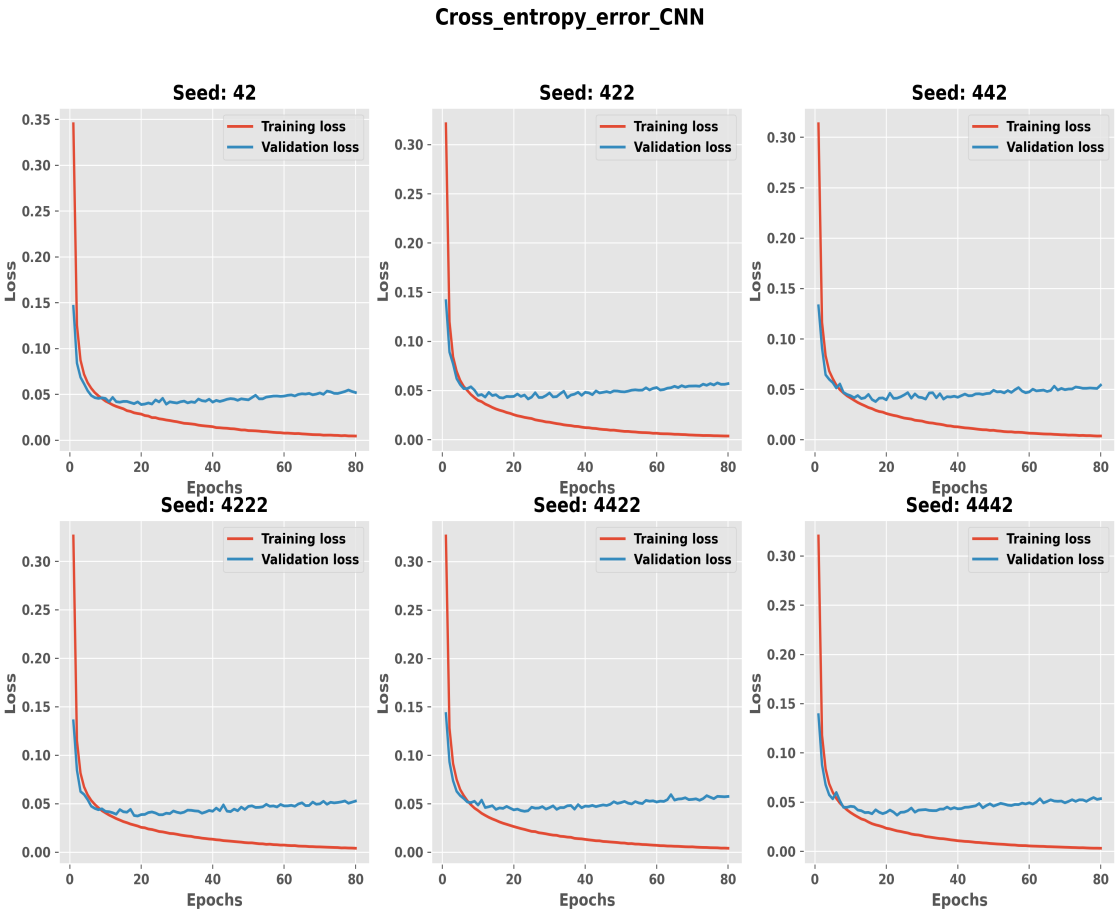
4. (5 points) Redo part 3(a) - 3(d) with a CNN i.e. with one 2-D convolutional layers → Relu activation → Maxpooling with appropriate hyperparameters. Compare the best result from the single layer neural network and the CNN, what could you conclude?

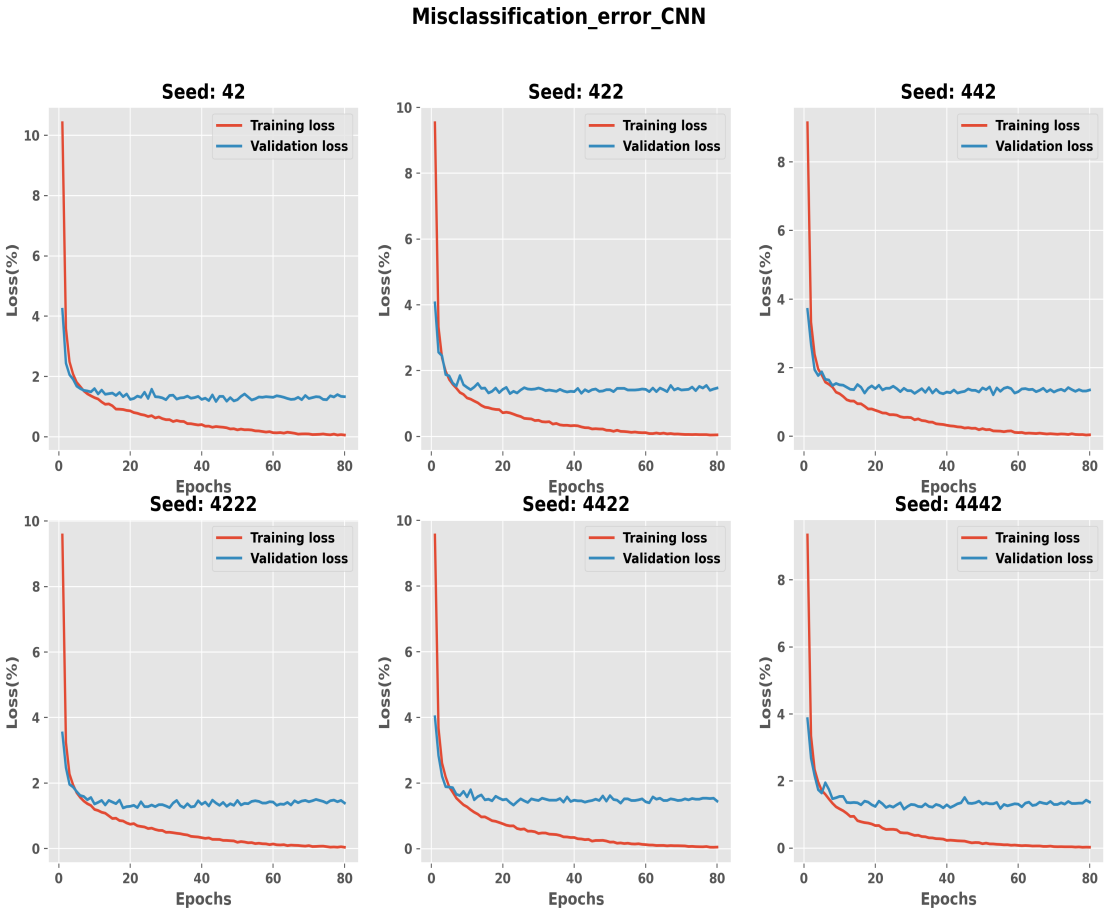
```
The structure of the single convolutional layer CNN is:
CNN(
(conv): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1,
ceil_mode=False)
)
(flatten): Flatten(start_dim=1, end_dim=-1)
(single_layer_nn): Sequential(
    (0): Linear(in_features=3136, out_features=10, bias=True)
```

```
    )
    )
```

We define the only convolutional layer by using 5×5 kernel size and a padding of 2 in both vertical and horizontal direction and moves the convolution kernel with stride 1. This conv2d layer would output the data in shape $(16, 28, 28)$. Then after ReLU activation, the features would be reduced to $(16, 14, 14)$ by the MaxPooling layer and flattened by the Flatten layer in to a fully connected layer. The number of output neurons is set to be the number of classes.

(a) & (b)





The learning curves of CNNs are listed above. The training parameters are: $lr = 0.001$, $N_{batch} = 100$, $N_{epoch} = 80$, $momentum = 0$. I reduced the number of epochs since the CNN is more suitable for computer vision problems, thus it would perform better than single layer NN, we can observe the overfitting under fewer epochs.

We can still see that the patterns in single layer NNs' learning curve also appear in CNNs' figure. The single convolutional layer CNN has the convergence rate as same as the single layer NN, both types can attain a status of overfitting at around the 20th epoch. Note that both two types of training errors and testing errors are lower than single layer NN when the training processes are finished, which conforms with the conclusion that CNN has better performance on image classification than single layer NN. The main reason is that its convolutional layer can work as a feature extractor and maxpooling layer can be interpreted as dimension reduction.

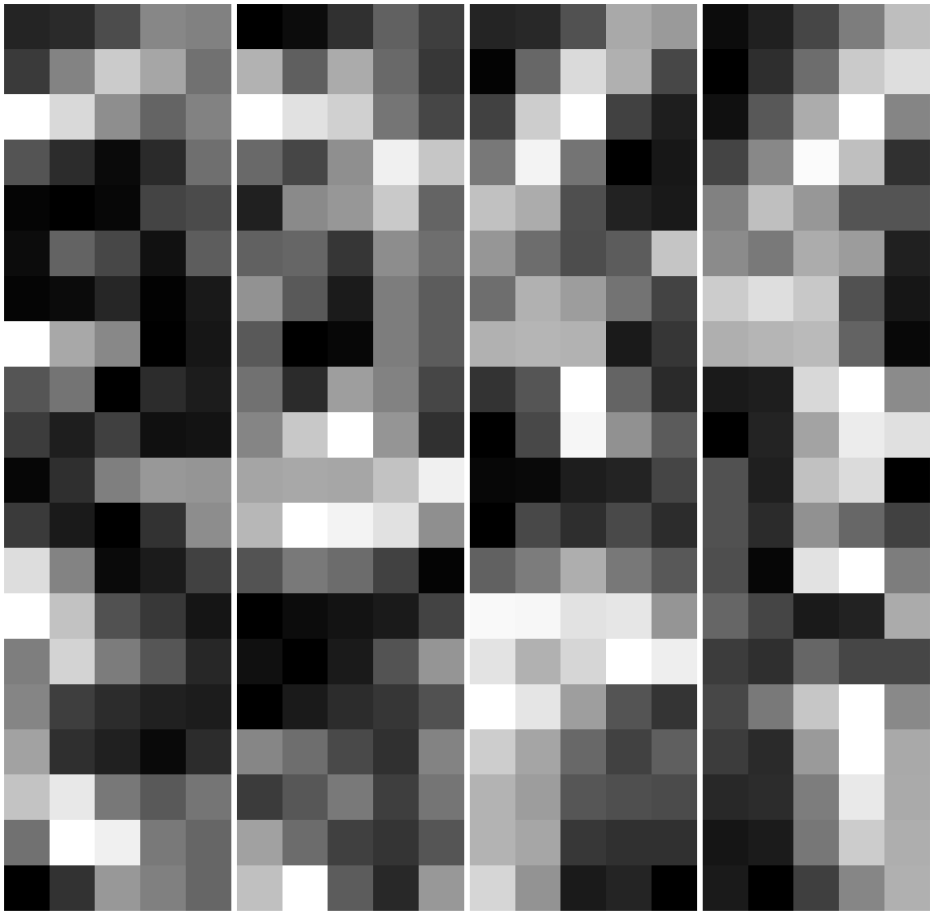
Also, there's less fluctuation during the training process in CNNs' learning curve.

(c)

	Cross_entropy_error_train	Misclassification_error_train	Cross_entropy_error_test	Misclassification_error_test	Accuracy_test
42	0.004573	0.058333	0.051995	0.0133	98.67
422	0.003798	0.046667	0.057091	0.0147	98.53
442	0.003827	0.043333	0.054304	0.0135	98.65
4222	0.004060	0.040000	0.052758	0.0139	98.61
4422	0.004143	0.051667	0.057638	0.0145	98.55
4442	0.003181	0.025000	0.053538	0.0137	98.63

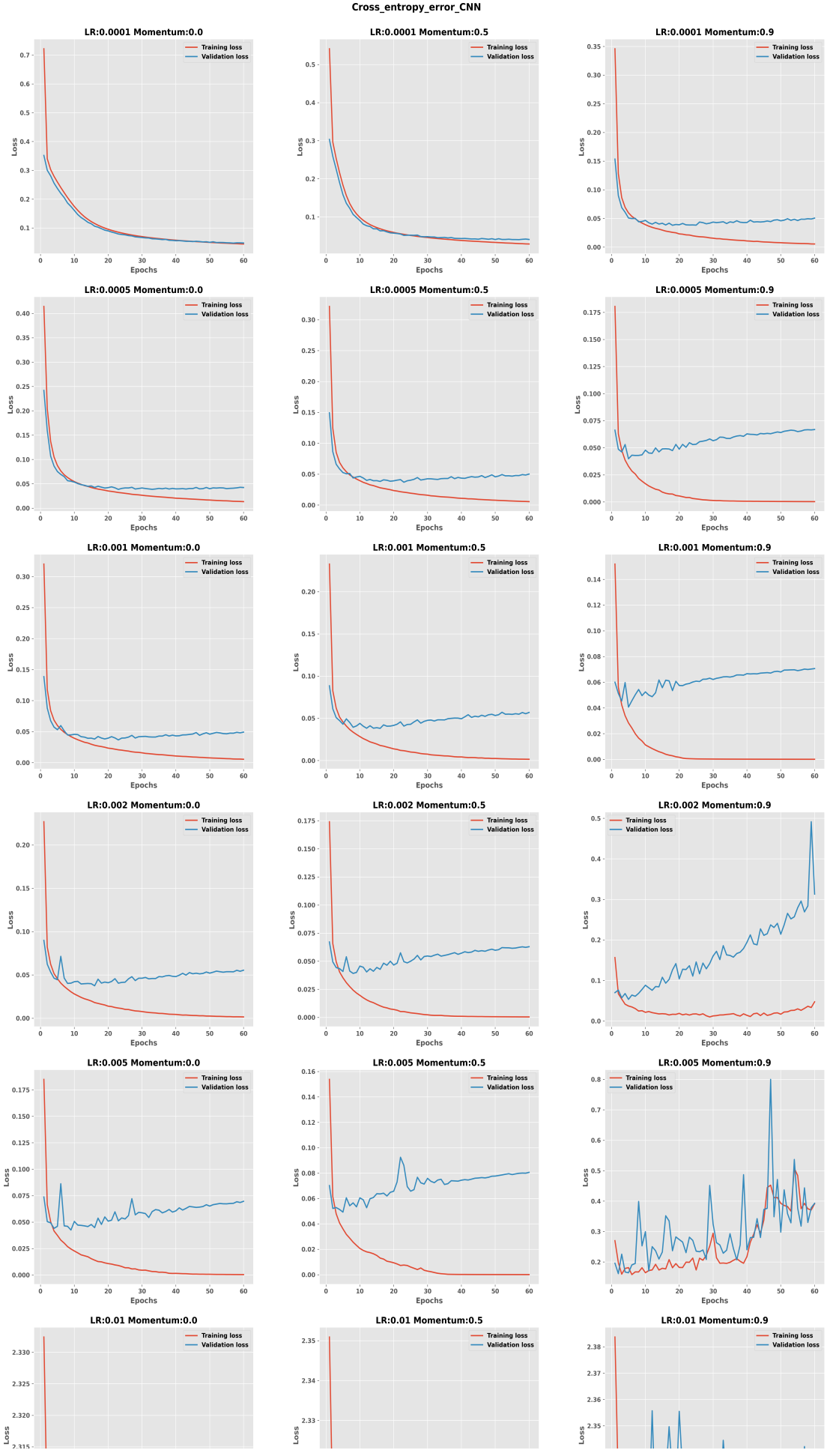
Model comparison between CNNs is based on this table.

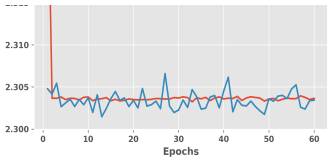
All CNNs have similar overfitting patterns starting from around the 10th epoch. The best model is CNN with seed 42 as it has the lowest test average cross-entropy error and lowest mean misclassification error.

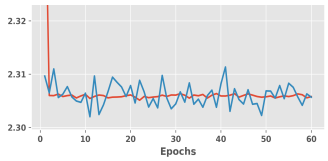


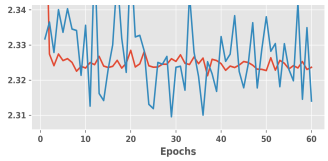
The visualization of the 16 convolutional kernel from the best single convolutional layer CNN shows some of the structures like the part of strokes with black and white pixels. The kernels are seem to be uncorrelated, which are reasonable.

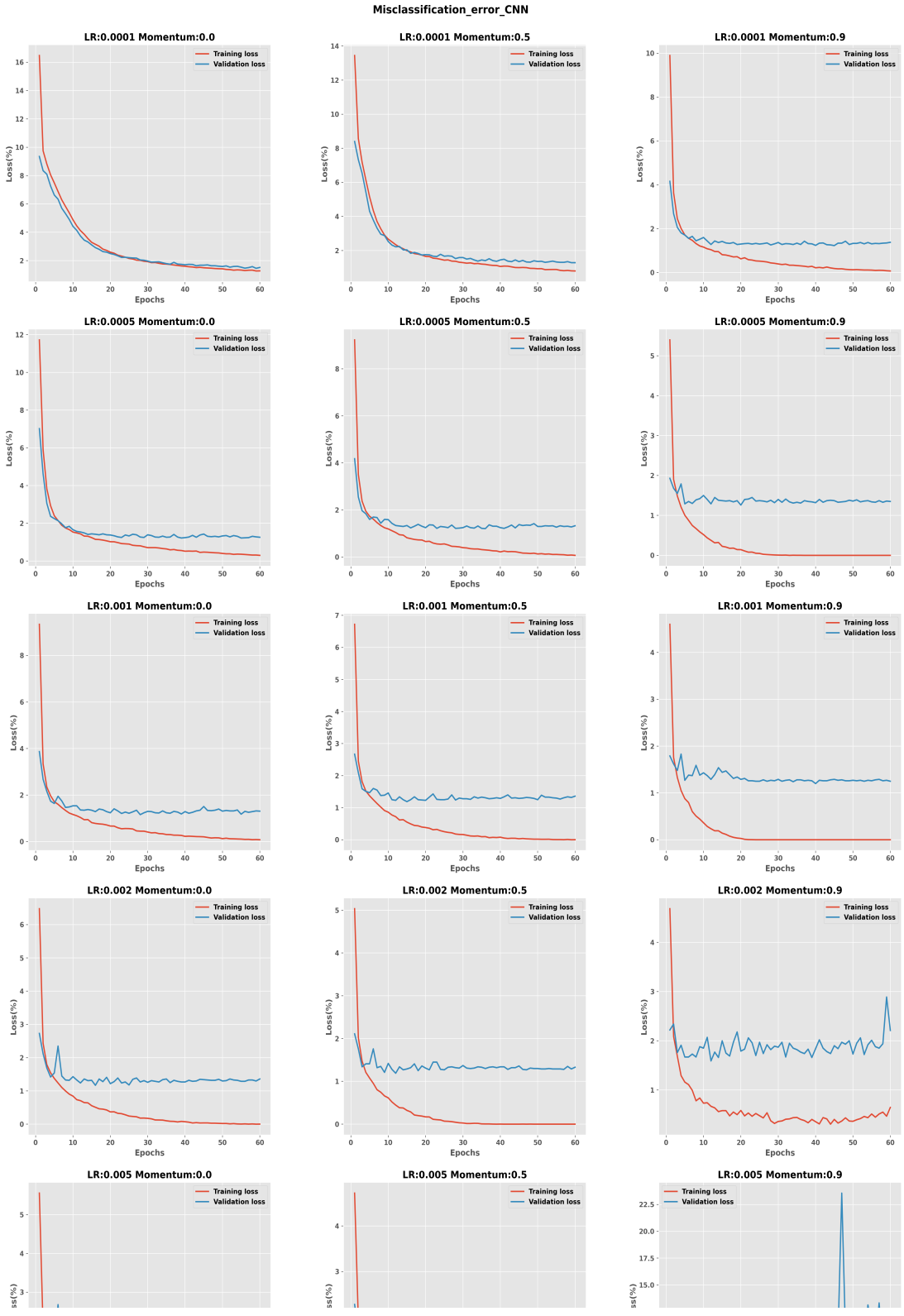
(d)

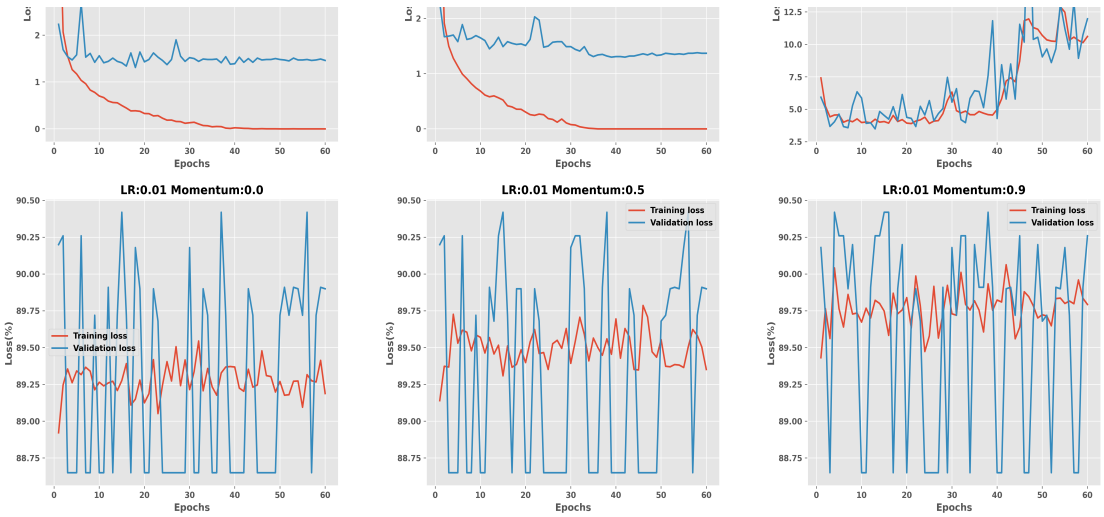

Cross_entropy_error_CNN
LR:0.0001 Momentum:0.0
LR:0.0001 Momentum:0.5
LR:0.0001 Momentum:0.9
LR:0.0005 Momentum:0.0
LR:0.0005 Momentum:0.5
LR:0.0005 Momentum:0.9
LR:0.001 Momentum:0.0
LR:0.001 Momentum:0.5
LR:0.001 Momentum:0.9
LR:0.002 Momentum:0.0
LR:0.002 Momentum:0.5
LR:0.002 Momentum:0.9
LR:0.005 Momentum:0.0
LR:0.005 Momentum:0.5
LR:0.005 Momentum:0.9
LR:0.01 Momentum:0.0
LR:0.01 Momentum:0.5
LR:0.01 Momentum:0.9
Training loss
Validation loss
Loss
Epochs


Epochs


Epochs


Epochs


Misclassification_error_CNN
LR:0.0001 Momentum:0.0
LR:0.0001 Momentum:0.5
LR:0.0001 Momentum:0.9
LR:0.0005 Momentum:0.0
LR:0.0005 Momentum:0.5
LR:0.0005 Momentum:0.9
LR:0.001 Momentum:0.0
LR:0.001 Momentum:0.5
LR:0.001 Momentum:0.9
LR:0.002 Momentum:0.0
LR:0.002 Momentum:0.5
LR:0.002 Momentum:0.9
LR:0.005 Momentum:0.0
LR:0.005 Momentum:0.5
LR:0.005 Momentum:0.9
Training loss
Validation loss
Loss(%)
Epochs



The learning curves with respect to two types of errors on the learning rate and momentum grid are shown above.

We trained the best single convolutional layer CNN with random seed 42 on the grid of learning rate $lr = {0.0001, 0.0005, 0.001, 0.002, 0.005, 0.01}$ and momentum $momemtum = {0.0, 0.5, 0.9}$. Since we already observed the overfitting around the 20th epoch in 4(a) & 4(b), here the number of epoch is reduced to 60 to speed up the training.

Comparing to the training result of single layer NN on the grid, we can see that CNNs are more sensitive to the momentum value, with sufficient epochs, the gap between training error and testing error is enlarged significantly as the momentum increases. The plots in the bottom of the two pictures demonstrate that CNN need to use smaller learning rate and momentum to converge. Suitable learning rate for single layer NN would result in remarkbale fluctuation when applying to CNNs.

	Cross_entropy_error_train	Misclassification_error_train	Cross_entropy_error_test	Misclassification_error_test	Accuracy_test
0.0001_0.0	0.044612	1.280000	0.047996	0.0152	98.48
0.0001_0.5	0.028999	0.803333	0.040825	0.0129	98.71
0.0001_0.9	0.005627	0.073333	0.050739	0.0138	98.62
0.0005_0.0	0.013537	0.300000	0.042523	0.0126	98.74
0.0005_0.5	0.005570	0.071667	0.050245	0.0133	98.67
0.0005_0.9	0.000352	0.000000	0.066899	0.0135	98.65
0.001_0.0	0.005535	0.080000	0.049360	0.0131	98.69
0.001_0.5	0.001532	0.001667	0.057008	0.0136	98.64
0.001_0.9	0.000122	0.000000	0.070673	0.0125	98.75
0.002_0.0	0.001580	0.003333	0.055514	0.0136	98.64
0.002_0.5	0.000479	0.000000	0.062956	0.0133	98.67
0.002_0.9	0.047486	0.645000	0.313150	0.0221	97.79
0.005_0.0	0.000482	0.000000	0.069713	0.0146	98.54
0.005_0.5	0.000133	0.000000	0.080638	0.0137	98.63
0.005_0.9	0.391485	10.610000	0.392926	0.1197	88.03
0.01_0.0	2.303650	89.188333	2.303438	0.8990	10.10
0.01_0.5	2.305807	89.351667	2.305698	0.8990	10.10
0.01_0.9	2.323657	89.793333	2.314059	0.9026	9.74

The final result after training is listed above. We can choose the best single convolutional layer CNN with:

- $Seed = 42, lr = 0.0001, momentum = 0.5$

since this model nearly has no overfitting in its learning curve, and has the lowest test average cross-entropy error and the 3rd highest test accuracy.

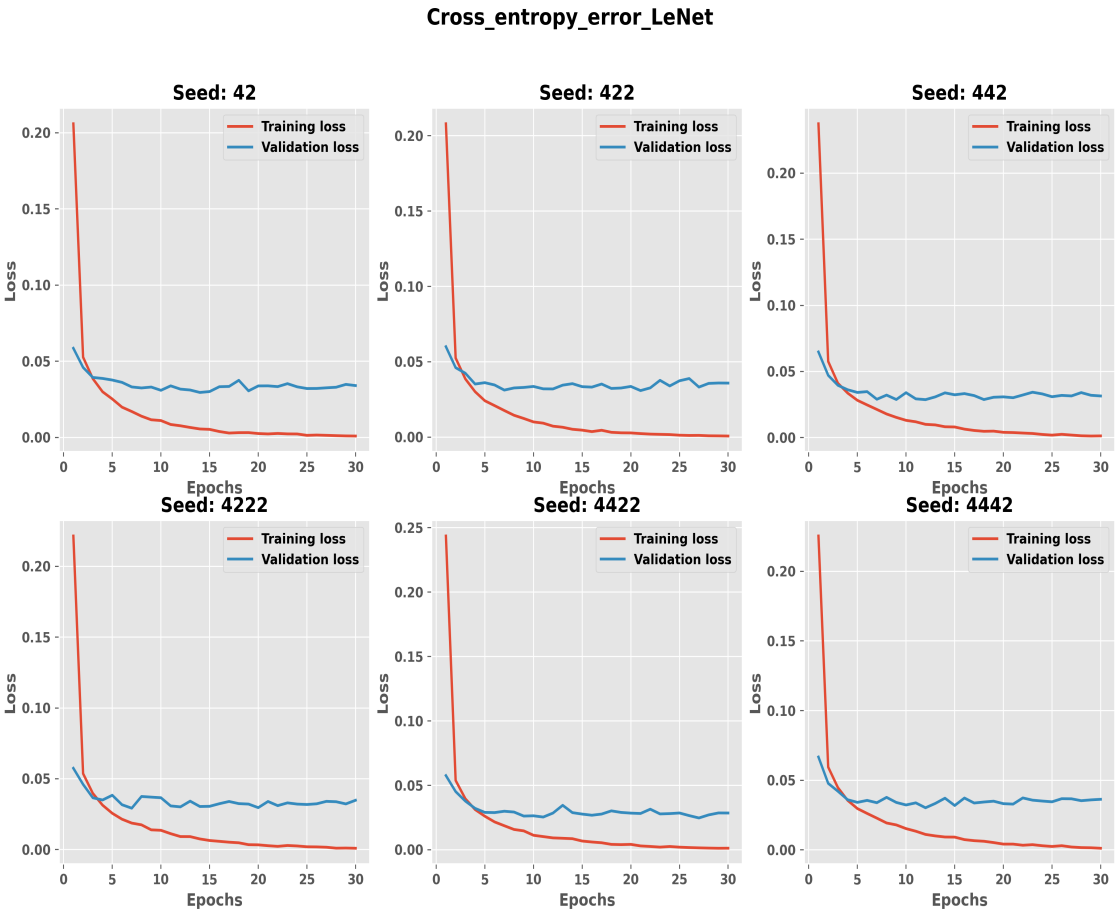
5. (5 points) Redo part 3(a) - 3(d) with your favorite deep learning architecture (e.g., introducing batch normalization, introducing dropout in training) to beat the performance of SVM with Gaussian Kernel, i.e., to have a test error rate lower than 1.4%.

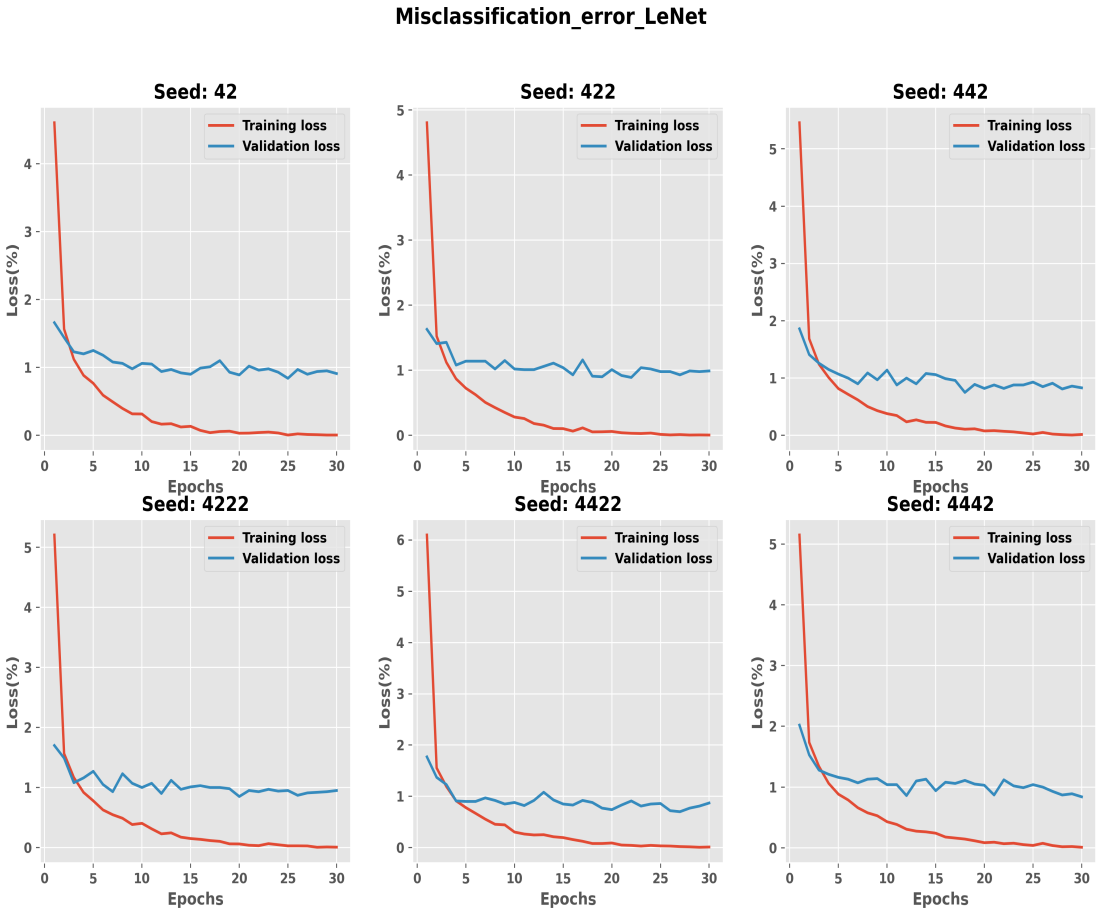
```
lenet(
(conv): Sequential(
    (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (4): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (5): ReLU()
    (6): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1,
ceil_mode=False)
)
(flatten): Flatten(start_dim=1, end_dim=-1)
```

```
  (fully_conncet): Sequential(
      (0): Linear(in_features=400, out_features=120, bias=True)
      (1): ReLU()
      (2): BatchNorm1d(120, eps=1e-05, momentum=0.1, affine=True,
  track_running_stats=True)
      (3): Linear(in_features=120, out_features=84, bias=True)
      (4): ReLU()
      (5): Linear(in_features=84, out_features=10, bias=True)
  )
  )
```

We use Lenet5-like CNN for this question, hereinafter called it lenet. I introduced batch-normalization in both of the convolutional and fully connected part. Evolved from the single convolutional layer CNN, the classic LeNet5 includes more convolutional, maxpooling and fully connected layers, which aims to improve the performance of feature extraction. More parameters makes this lenet more complex and thus would be suitable for image classification. The 28×28 image was shaped in a sequence: $(6,28,28)$ -> $(6,14,14)$ -> $(16,10,10)$ -> $(16,5,5)$ -> $16 \times 5 \times 5$ -> 120 -> 84 -> 10.

(a) & (b)





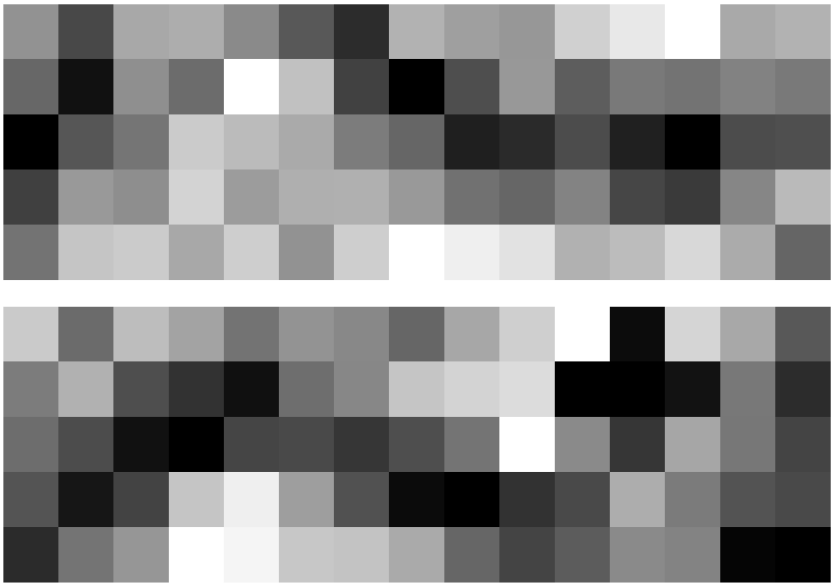
The learning curves of the LeNet5-like CNN are shown above. $N_{batch} = 100$, $lr = 0.0005$, $seeds = {42, 422, 442, 4422, 4442}$. I chose the $N_{epochs} = 30$ because the lenet is foreseeable more powerful than the single convolutional layer CNN. More parameters would make it overfitting easily and early. As we can see, lenets start overfitting at around the 5th epoch and easily attain a level of test accuracy of around 99%. The convergence can be attained instantly within 4 epochs. One difference is that the test average misclassification error seems to be more unstable than the average cross-entropy error. Other remaining patterns of overfitting are the same as we found in questions 3 and 4.

(c)

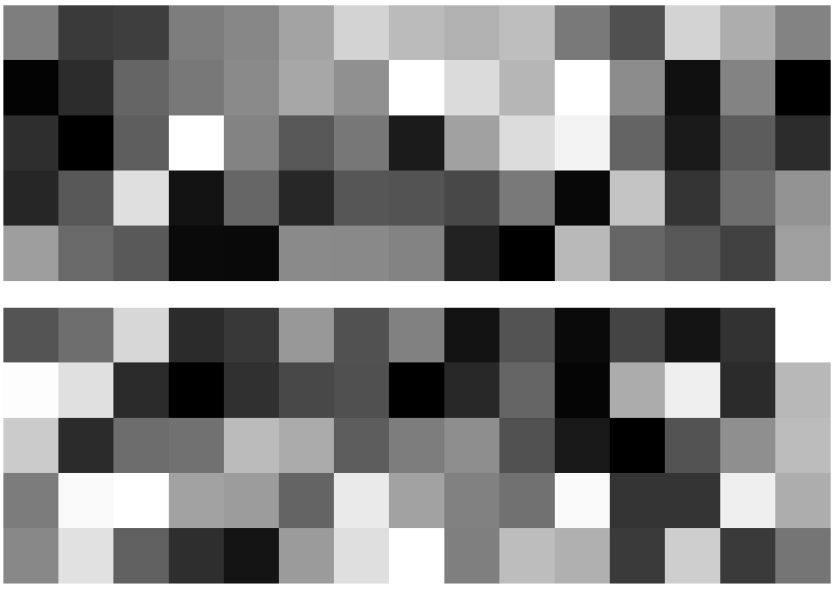
	Cross_entropy_error_train	Misclassification_error_train	Cross_entropy_error_test	Misclassification_error_test	Accuracy_test
42	0.000889	0.005000	0.034001	0.0091	99.09
422	0.000755	0.005000	0.035877	0.0099	99.01
442	0.001274	0.016667	0.031468	0.0083	99.17
4222	0.000909	0.006667	0.034819	0.0095	99.05
4422	0.001234	0.011667	0.028544	0.0087	99.13
4442	0.001198	0.010000	0.036334	0.0084	99.16

The table of evaluation metrics of lenets is shown above. Since all lenets are overfitting, so we pick the best model to be lenet with seed 4422, since it has nearly the highest test accuracy of 99.13% while with the lowest test cross-entropy error 0.028544, which is more robust against overfitting.

param_best_model_lenet_conv_kernel_0

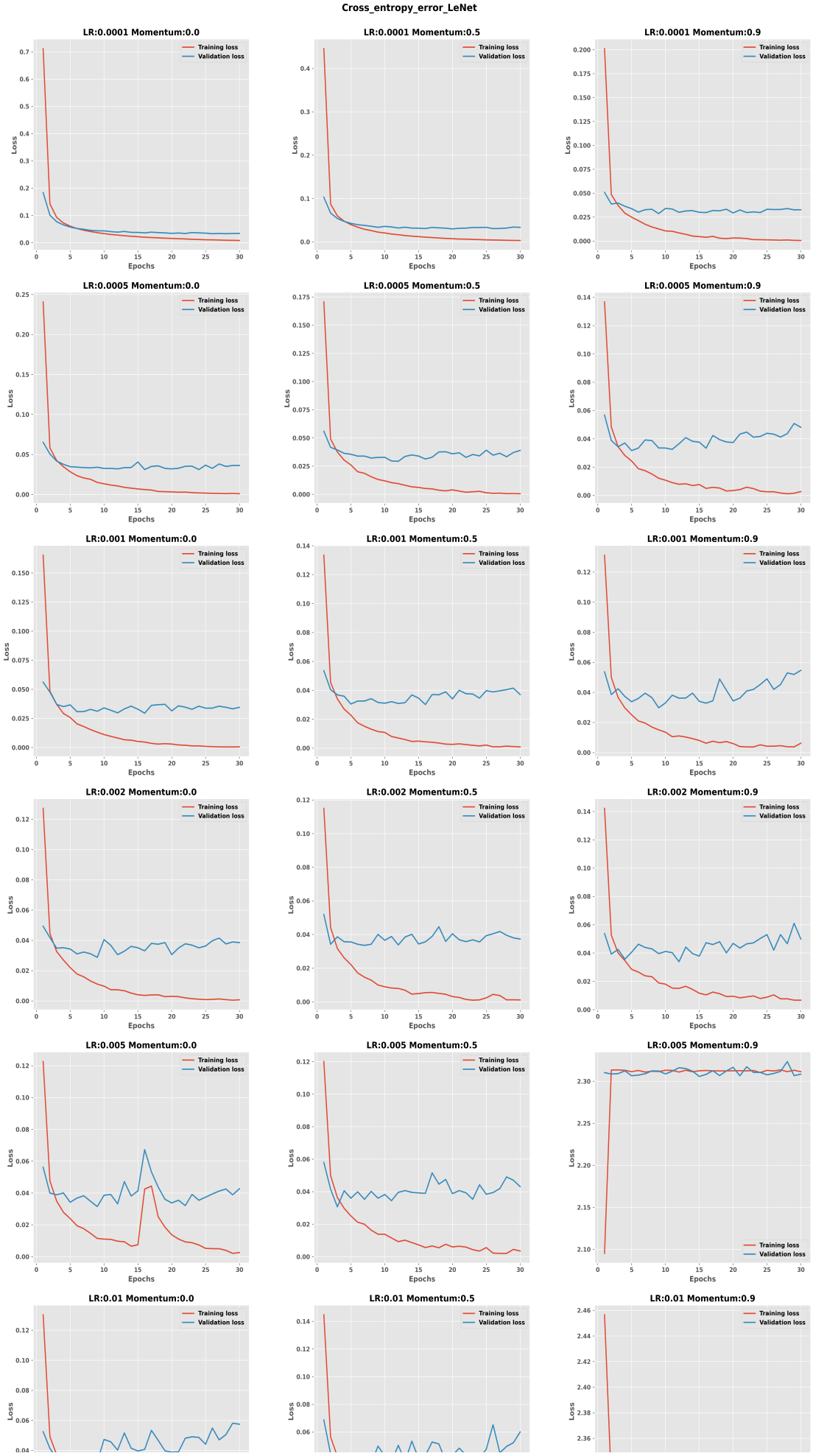


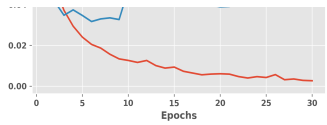
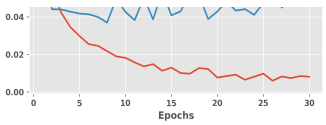
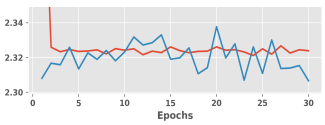
param_best_model_lenet_samples_conv_kernel_1

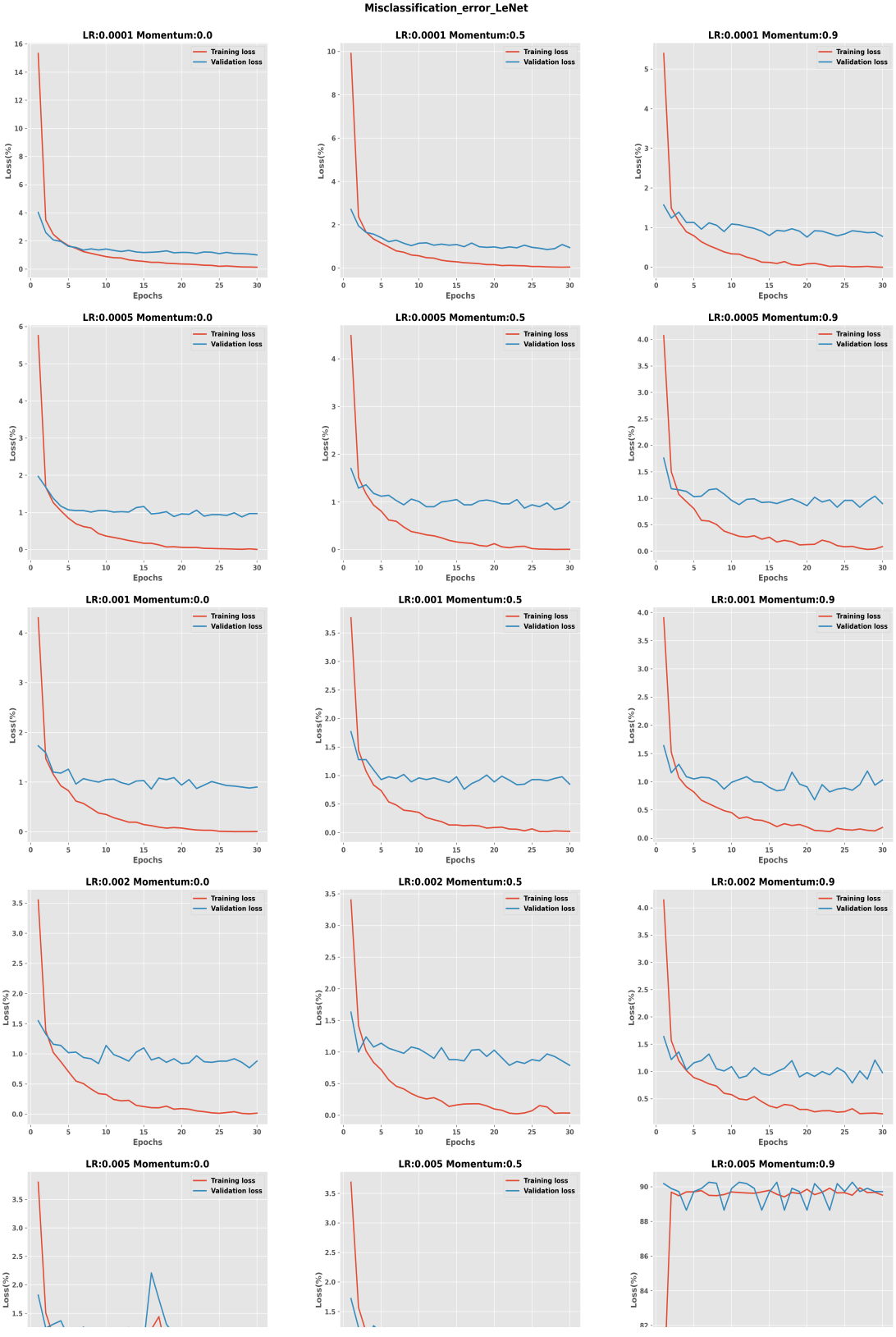


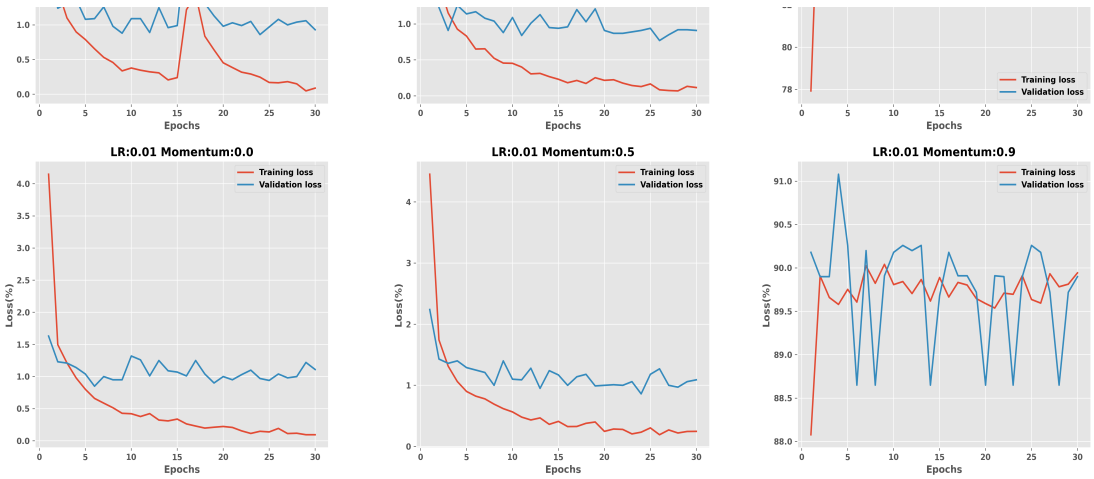
In this question I plotted the first and second convolutional kernels. The figure named kernel_0 shows all the 6 kernels learnt by lenet. For the second convolutional layer, I sampled 6 kernels to represent their structures. We can see that these kernels mainly contain the local structures still like part of strokes while remain uncorrelated.

(d)


Cross_entropy_error_LeNet
LR:0.0001 Momentum:0.0
LR:0.0001 Momentum:0.5
LR:0.0001 Momentum:0.9
LR:0.0005 Momentum:0.0
LR:0.0005 Momentum:0.5
LR:0.0005 Momentum:0.9
LR:0.001 Momentum:0.0
LR:0.001 Momentum:0.5
LR:0.001 Momentum:0.9
LR:0.002 Momentum:0.0
LR:0.002 Momentum:0.5
LR:0.002 Momentum:0.9
LR:0.005 Momentum:0.0
LR:0.005 Momentum:0.5
LR:0.005 Momentum:0.9
LR:0.01 Momentum:0.0
LR:0.01 Momentum:0.5
LR:0.01 Momentum:0.9
Training loss
Validation loss
Loss
Epochs


Misclassification_error_LeNet
LR:0.0001 Momentum:0.0
LR:0.0001 Momentum:0.5
LR:0.0001 Momentum:0.9
LR:0.0005 Momentum:0.0
LR:0.0005 Momentum:0.5
LR:0.0005 Momentum:0.9
LR:0.001 Momentum:0.0
LR:0.001 Momentum:0.5
LR:0.001 Momentum:0.9
LR:0.002 Momentum:0.0
LR:0.002 Momentum:0.5
LR:0.002 Momentum:0.9
LR:0.005 Momentum:0.0
LR:0.005 Momentum:0.5
LR:0.005 Momentum:0.9
Training loss
Validation loss
Loss(%)
Epochs



The learning curves of the best lenet with respect to two types of errors on the learning rate and momentum grid are shown above.

The parameters grid is :$lr = {0.0001, 0.0005, 0.001, 0.002, 0.005, 0.01}$, $momentum = {0.0, 0.5, 0.9}$, $seed = 4422$. Lenets have the similar property with CNNs(i.e. Learning rate should be small, otherwise there would be great fluctuations and even diverge. With fixed learning rate, larger momentum would be more unstable.) However, due to the model complexity a lenet has, we find that it can always reach the 99% test accuracy and nearly 0 training loss when it converges in 30 epochs, no matter what momentum and learning rate we use, this represents the advantage that lenet have a better structure over single layer CNNs.

	Cross_entropy_error_train	Misclassification_error_train	Cross_entropy_error_test	Misclassification_error_test	Accuracy_test
0.0001_0.0	0.008047	0.135000	0.034010	0.0101	98.99
0.0001_0.5	0.003356	0.050000	0.033743	0.0095	99.05
0.0001_0.9	0.000590	0.000000	0.032757	0.0078	99.22
0.0005_0.0	0.000995	0.005000	0.036328	0.0097	99.03
0.0005_0.5	0.000573	0.006667	0.038991	0.0100	99.00
0.0005_0.9	0.002699	0.086667	0.048184	0.0090	99.10
0.001_0.0	0.000613	0.005000	0.034571	0.0090	99.10
0.001_0.5	0.000865	0.023333	0.037119	0.0085	99.15
0.001_0.9	0.006197	0.190000	0.054711	0.0103	98.97
0.002_0.0	0.000751	0.016667	0.038558	0.0088	99.12
0.002_0.5	0.001136	0.035000	0.037330	0.0079	99.21
0.002_0.9	0.006732	0.225000	0.049922	0.0098	99.02
0.005_0.0	0.002586	0.088333	0.042676	0.0093	99.07
0.005_0.5	0.003506	0.116667	0.043133	0.0091	99.09
0.005_0.9	2.311442	89.516667	2.308489	0.8972	10.28
0.01_0.0	0.002691	0.095000	0.057508	0.0111	98.89
0.01_0.5	0.008190	0.246667	0.060003	0.0109	98.91
0.01_0.9	2.323921	89.943333	2.306656	0.8990	10.10

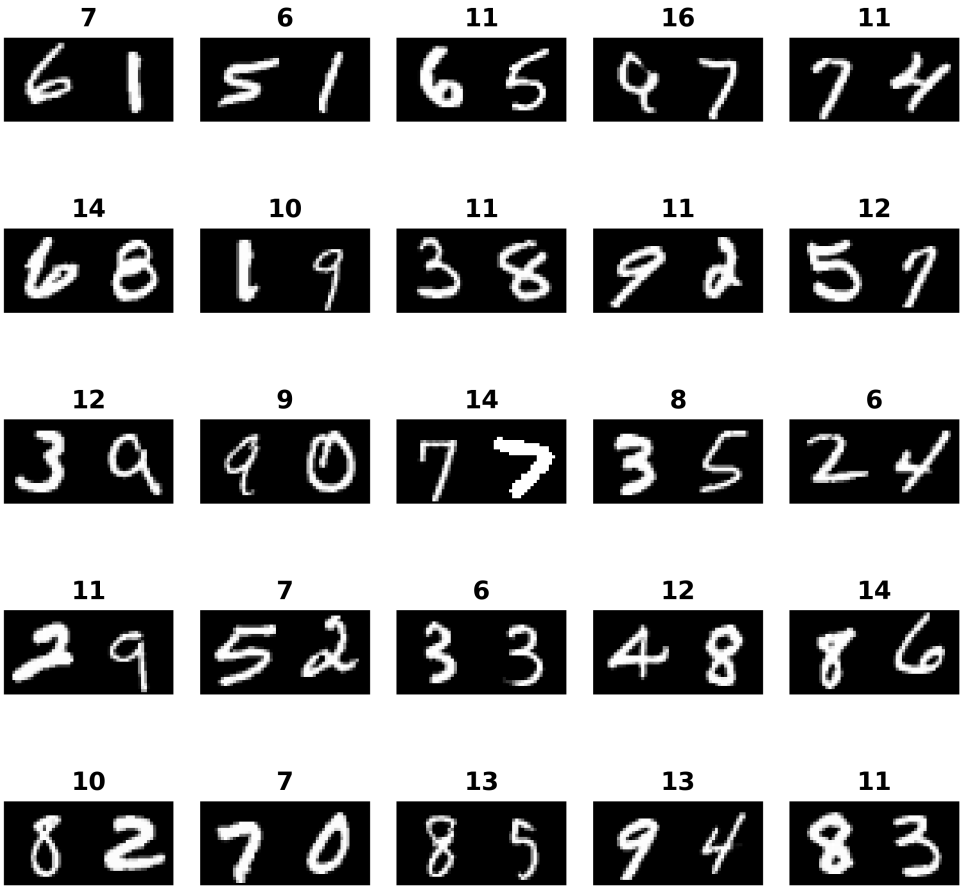
The final grid search result is shown in this table. **Most of the test results with a test accuracy over 99% already beat the SVM with Gaussian kernel.** We simply choose:

- $lr = 0.0001, momentum = 0.9, seed = 4422$

as our best lenet with lowest test cross-entropy error 0.032757 and highest test accuracy 99.22%.

Part 5: More About Deep Learning

6. (1 points) 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?



I split the last column as the labels for this new data. The plot is made by extracting each row of the first 1568 entries and using NumPy.reshape() function to reshape them into 28×56 array. Since the NumPy function is in a row-oriented order, so the new data is stored in row order. By observing the labels, I found that the labels are the sum of the two digits shown in one picture.

7. (8 points) Repeat part 3(a) - 3(d) with at least two of your favorite deep learning architecture (e.g., introducing batch normalization, introducing dropout in training) with respect to with train.txt, val.txt and test.txt. In particular,

> (a) Using train.txt to train your models.
>
> (b) Using the validation error (i.e., the performance on val.txt) to select the best model.
>
> (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%.

LeNet

```
lenet_alt(
(conv): Sequential(
    (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (4): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (5): ReLU()
    (6): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1,
ceil_mode=False)
)
(dropout): Dropout(p=0.25, inplace=False)
(flatten): Flatten(start_dim=1, end_dim=-1)
(fully_conncet): Sequential(
    (0): Linear(in_features=960, out_features=512, bias=True)
    (1): ReLU()
    (2): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (3): Linear(in_features=512, out_features=128, bias=True)
    (4): ReLU()
    (5): Linear(in_features=128, out_features=19, bias=True)
)
)
```

AlexNet

```
alexnet(
(conv): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU()
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (9): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
```

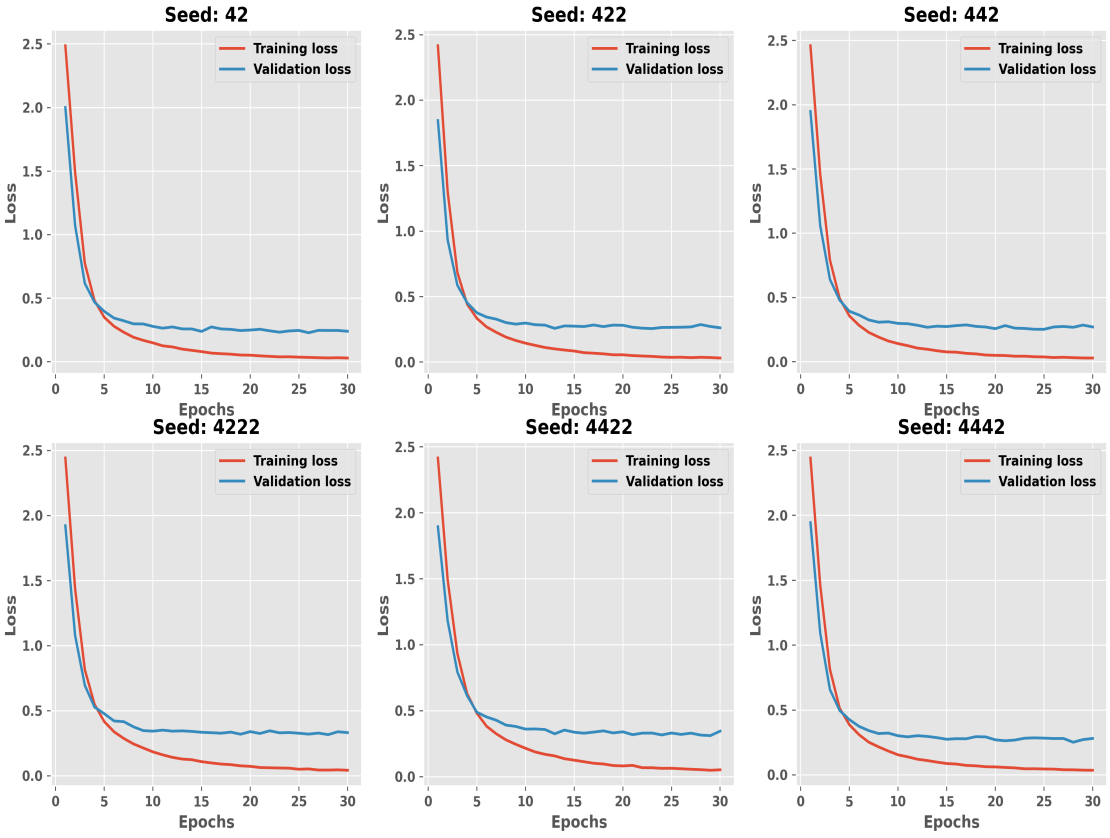
```
    (10): ReLU()
    (11): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
)
(flatten): Flatten(start_dim=1, end_dim=-1)
(fully_connect): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=4608, out_features=1024, bias=True)
    (2): ReLU()
    (3): Linear(in_features=1024, out_features=512, bias=True)
    (4): ReLU()
    (5): Linear(in_features=512, out_features=19, bias=True)
)
)
```

We use Lenet5-like CNN and AlexNet-like CNN for this question, hereinafter called them lenet and alexnet respectively. I also introduced dropout and batch-normalization in this question. The lenet is updated by adding a dropout layer before the flatten layer with a dropout probability 0.25 for slight regularization. Since this question has more features to extract, I implemented a modified AlexNet to try to achieve better results than lenet since it has more complex structure, which is designed for more complex hypotheses. To fit the alexnet for the MNIST dataset, I changed the kernel size of maxpooling layer from 3 to 2, and reduced the number of kernels in convolutional to $64, 128, 256$ rather than $96, 256, 384$. The fully connected layer is also simplified, their size is down to $1024, 512$. The output layer size is assigned to 19, which is the number of classes in this new dataset.

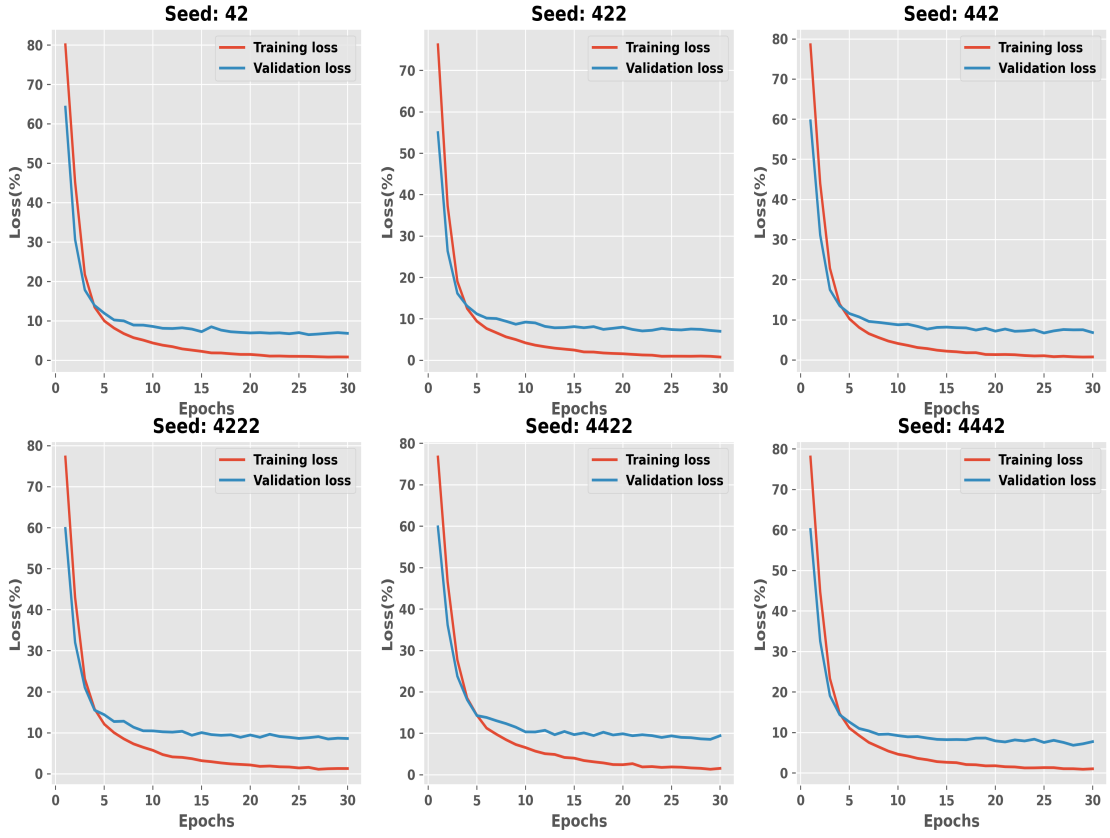
(a) & (b)

LeNet

Cross_entropy_error_LeNet_alt



Misclassification_error_LeNet_alt

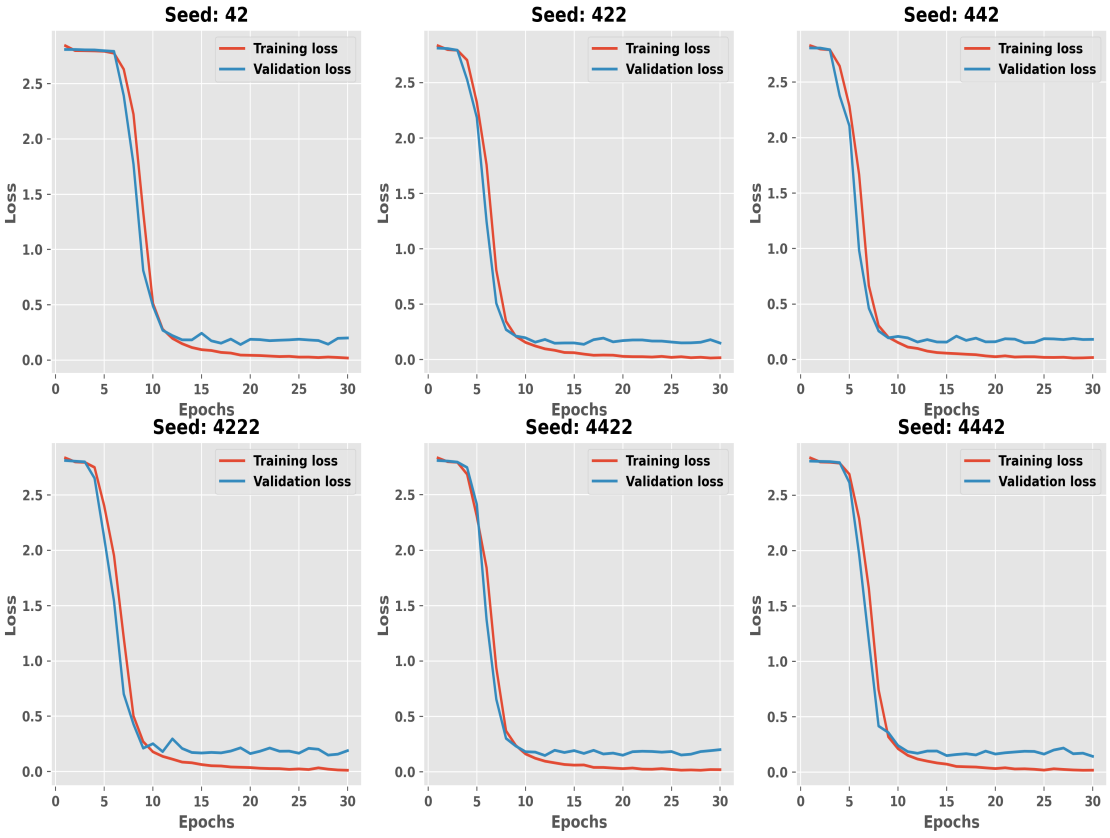


The learning curve of updated lenet is shown above.

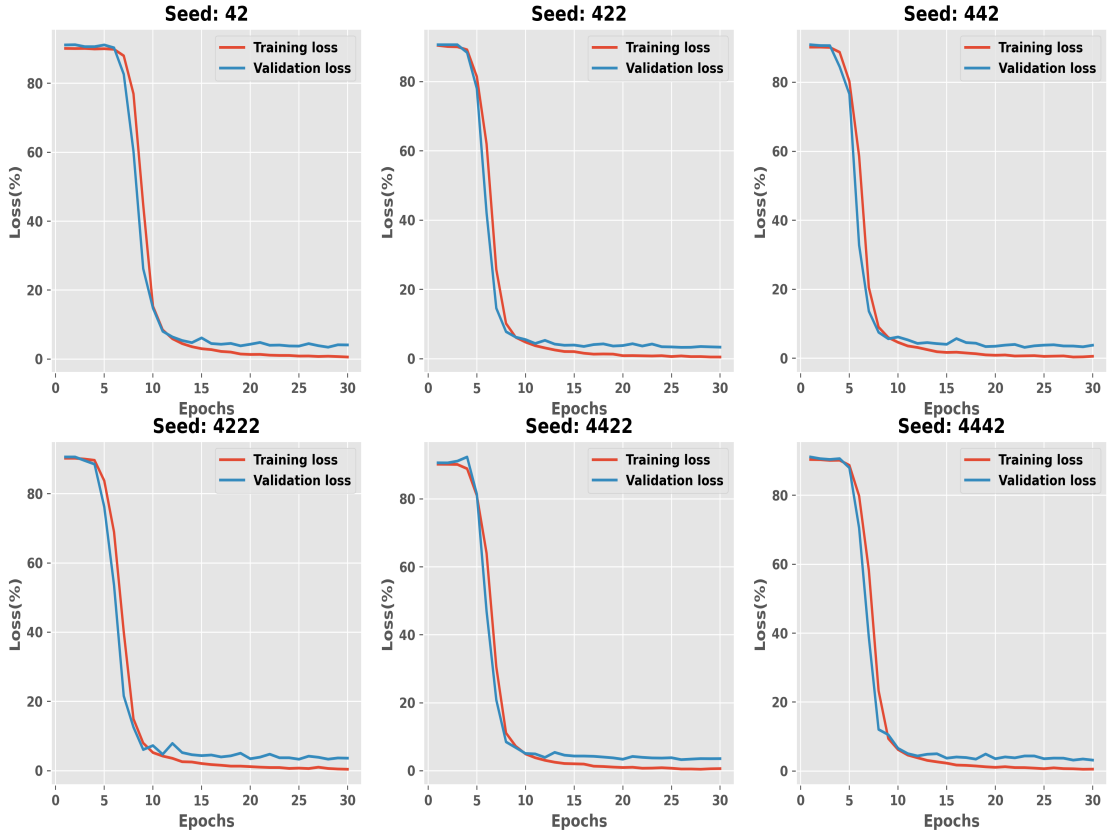
The parameters are: $lr = 0.0005$, $seeds = {42, 422, 442, 4222, 4422, 4442}$, $momentum = 0$, $N_{batch} = 100$, $N_{epochs} = 30$. We can see that the updated lenet is stable and the overfitting is limited, there'are no increment of validation cross-entropy error. All different random seeds yield similar learning curve. The validation loss functions of the udpated lenet is a little higher than working on original MNIST.

AlexNet

Cross_entropy_error_AlexNet



Misclassification_error_AlexNet



The learning curve of alexnet is shown above.

The parameters are: $lr = 0.001$, $seeds = {42, 422, 442, 4222, 4422, 4442}$, $momentum = 0$, $N_{batch} = 100$, $N_{epochs} = 30$. Alexnets have unique learning curves, as their loss function on both training set and the validation set remain high at the first 5 - 10 epochs, and then rapidly decrease to a level under 0.5 average cross-entropy error and 10% average misclassification error on both training and validation set. In the whole process, alexnets are extremely stable and faced no overfitting. This demonstrates the powerful learning ability and well-defined structure of alexnet.

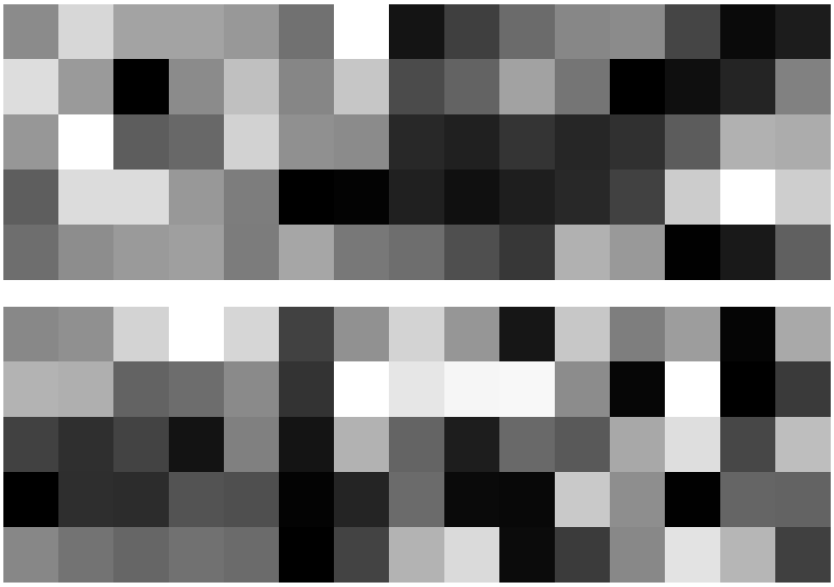
(c)

LeNet

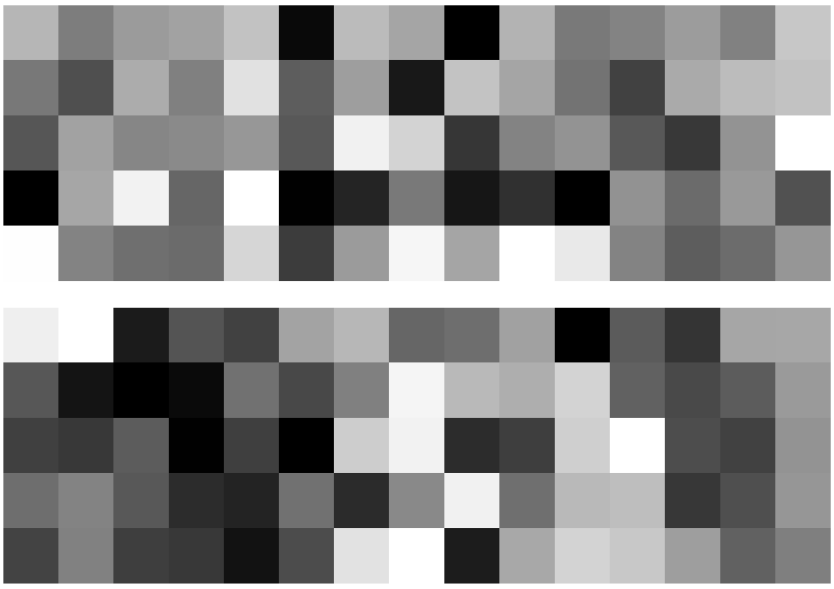
	Cross_entropy_error_train	Misclassification_error_train	Cross_entropy_error_test	Misclassification_error_test	Accuracy_test
42	0.029656	0.835	0.240117	0.0684	93.16
422	0.031457	0.800	0.262117	0.0702	92.98
442	0.028650	0.770	0.270597	0.0684	93.16
4222	0.042164	1.300	0.331506	0.0862	91.38
4422	0.052106	1.555	0.345363	0.0944	90.56
4442	0.035920	1.050	0.281117	0.0776	92.24

Best udpated lenet is the one with seed 42, it has the highest validation accuracy 93.16% and the lowest validation average cross-entropy error 0.240117.

param_best_model_lenet_alt_conv_kernel_0



param_best_model_lenet_alt_sample_conv_kernel_1



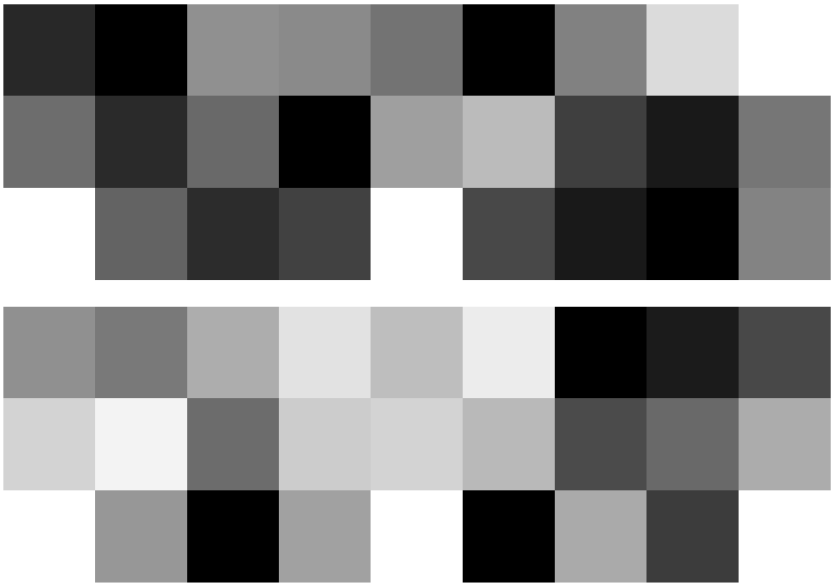
The visualization of two convolutional kernel in the best updated lenent are shown above. The second layer I choose only 6 samples to be displayed. They are similar with previous one in CNNs and lenets with some heterogeneous structures but also remain uncorrelated.

AlexNet

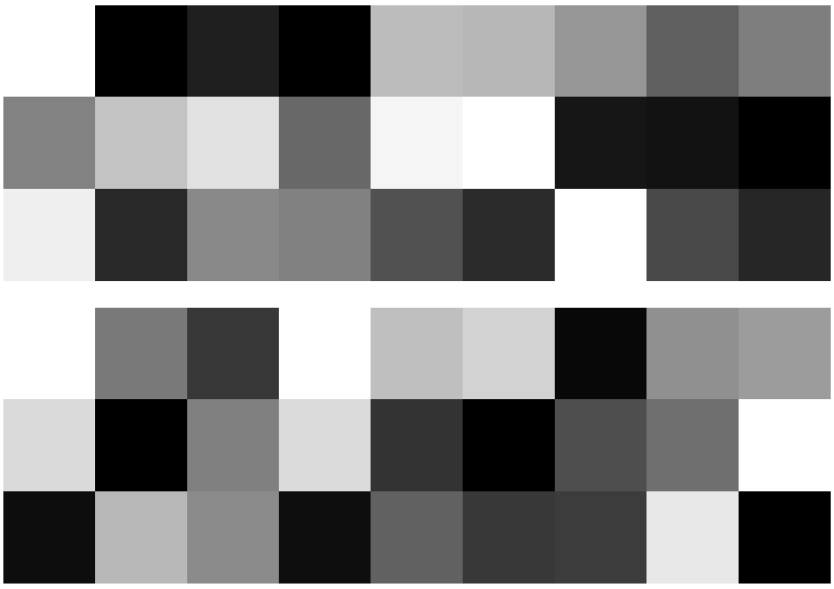
	Cross_entropy_error_train	Misclassification_error_train	Cross_entropy_error_test	Misclassification_error_test	Accuracy_test
42	0.018325	0.590	0.199624	0.0410	95.90
422	0.017128	0.485	0.149650	0.0334	96.66
442	0.017438	0.590	0.181479	0.0380	96.20
4222	0.012052	0.400	0.189282	0.0360	96.40
4422	0.020007	0.640	0.200628	0.0358	96.42
4442	0.017959	0.540	0.142802	0.0316	96.84

The best alexnet is from seed 4442 with the highest validation accuracy 96.84% and the lowest validation average cross-entropy error 0.142802. The difference between updated lenet and alexnet also confirms that alexnet is more powerful.

param_best_model_alexnet_samples_conv_kernel_0



param_best_model_alexnet_samples_conv_kernel_1

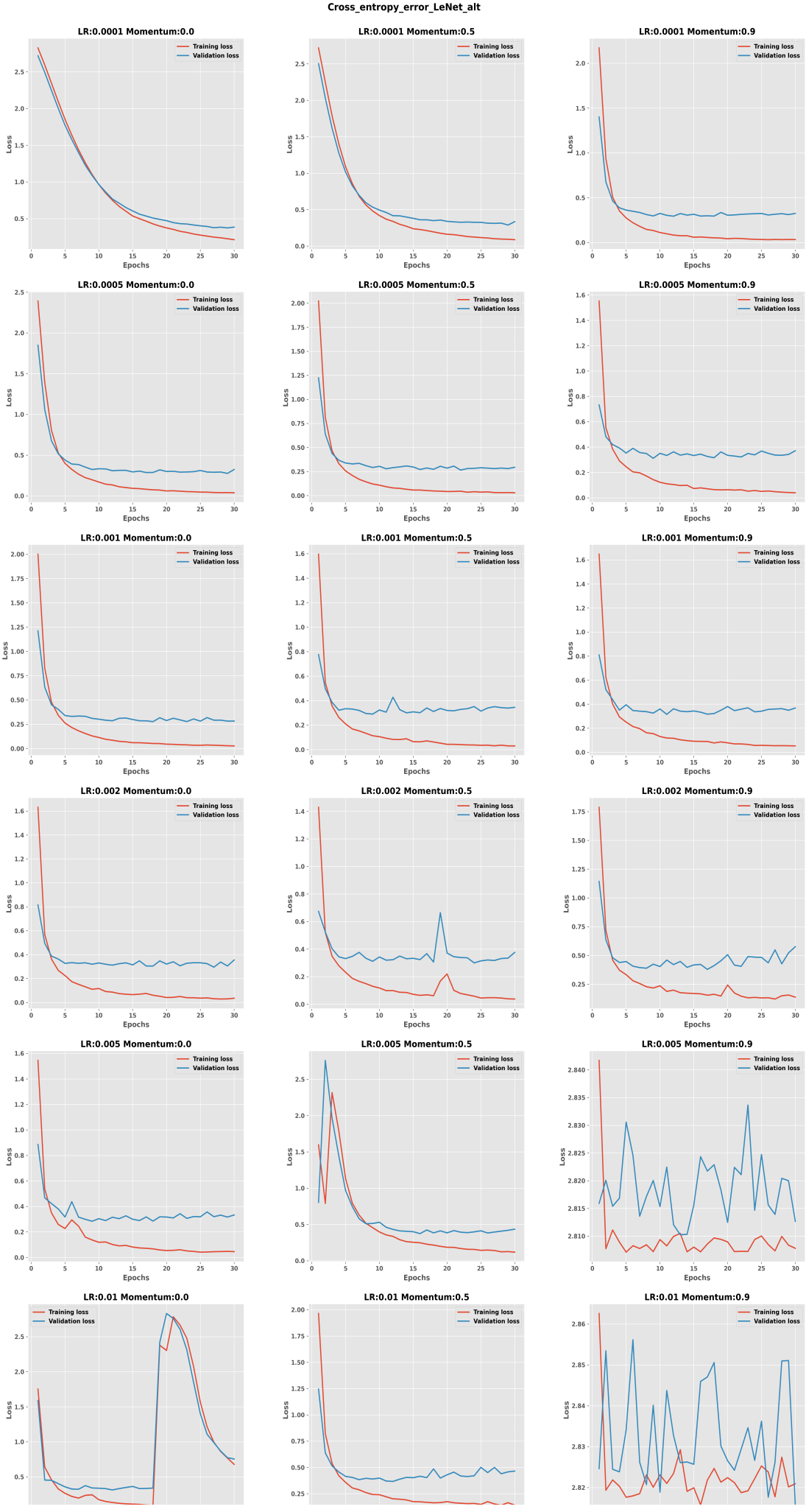


The samples of convolutional kernels of the best parameters learned in alexnet are shown above.

The 12 3×3 kernels also show different structures without any correlation, thus they are plausible.

(d)

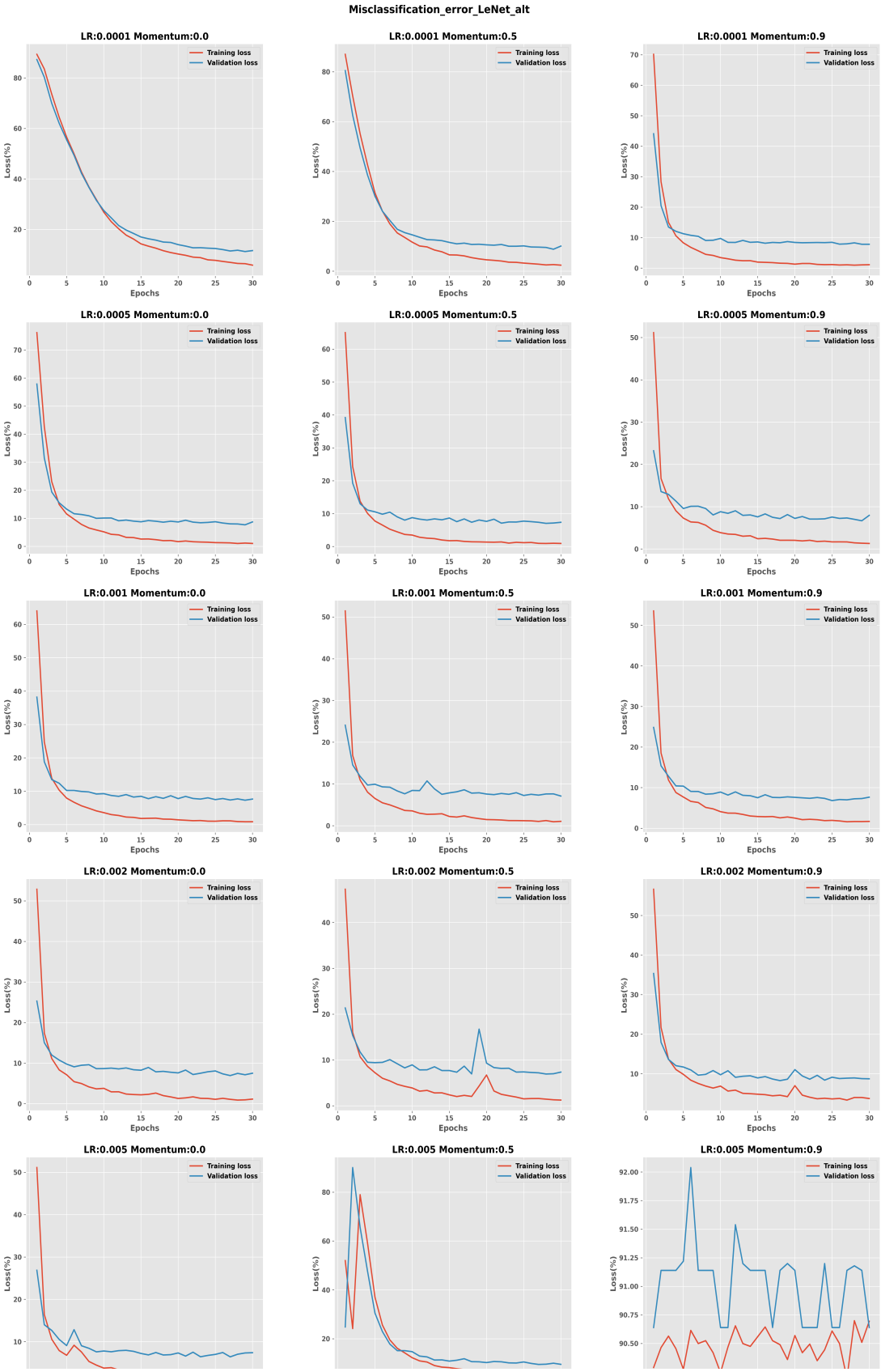
LeNet

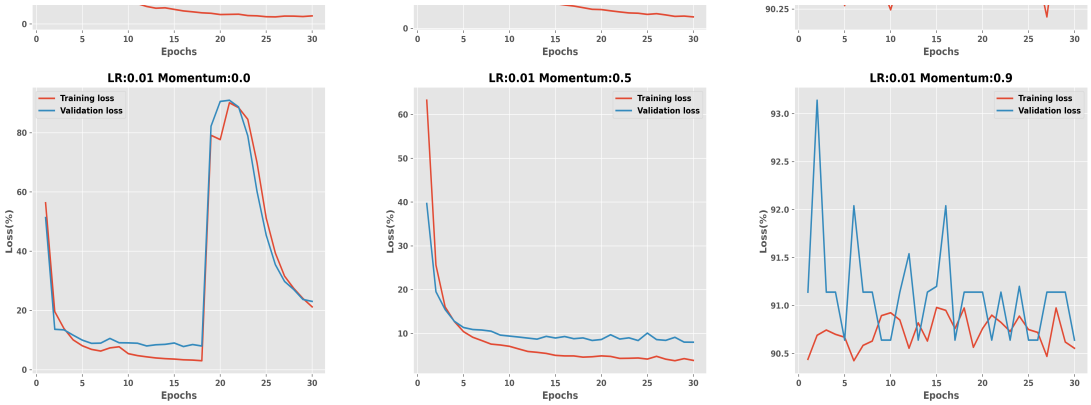

Cross_entropy_error_LeNet_alt
LR:0.0001 Momentum:0.0
LR:0.0001 Momentum:0.5
LR:0.0001 Momentum:0.9
LR:0.0005 Momentum:0.0
LR:0.0005 Momentum:0.5
LR:0.0005 Momentum:0.9
LR:0.001 Momentum:0.0
LR:0.001 Momentum:0.5
LR:0.001 Momentum:0.9
LR:0.002 Momentum:0.0
LR:0.002 Momentum:0.5
LR:0.002 Momentum:0.9
LR:0.005 Momentum:0.0
LR:0.005 Momentum:0.5
LR:0.005 Momentum:0.9
LR:0.01 Momentum:0.0
LR:0.01 Momentum:0.5
LR:0.01 Momentum:0.9
Training loss
Validation loss
Loss
Epochs


0.0
0 5 10 15 20 25 30
Epochs


0 5 10 15 20 25 30
Epochs


0 5 10 15 20 25 30
Epochs


Misclassification_error_LeNet_alt
LR:0.0001 Momentum:0.0
LR:0.0001 Momentum:0.5
LR:0.0001 Momentum:0.9
LR:0.0005 Momentum:0.0
LR:0.0005 Momentum:0.5
LR:0.0005 Momentum:0.9
LR:0.001 Momentum:0.0
LR:0.001 Momentum:0.5
LR:0.001 Momentum:0.9
LR:0.002 Momentum:0.0
LR:0.002 Momentum:0.5
LR:0.002 Momentum:0.9
LR:0.005 Momentum:0.0
LR:0.005 Momentum:0.5
LR:0.005 Momentum:0.9
Training loss
Validation loss
Loss(%)
Epochs



Above are the learning curves of updated lenet based on the best random seed 42 on the grid of learning rate and momentum.

The grid is $lr = {0.0001, 0.0005, 0.001, 0.002, 0.005, 0.01}$, $momentum = {0.0, 0.5, 0.9}$. We can see that the updated lenets are sensitive to learning rate. When learning rate is small(i.e. 0.0001) and without any acceleration of momentum, its learning curve would have a longer learning phase with gradient descent optimizations lasting about 30 epochs, which is clearly shown on the top left subplot. As it's a kind of CNN, it also has the property we discussed in the previous question. The convergence rate in this question is more strict on learning rate, if the learning rate is too large(i.e. 0.01), it even has a great fluctuation that makes it diverge after a temporary convergence.

While the training and validation metrics are pretty close in cases that would converge, a good parameter combination for lenet to learn is using small learning rate with high momentum to achieve stable and fast converge rate.

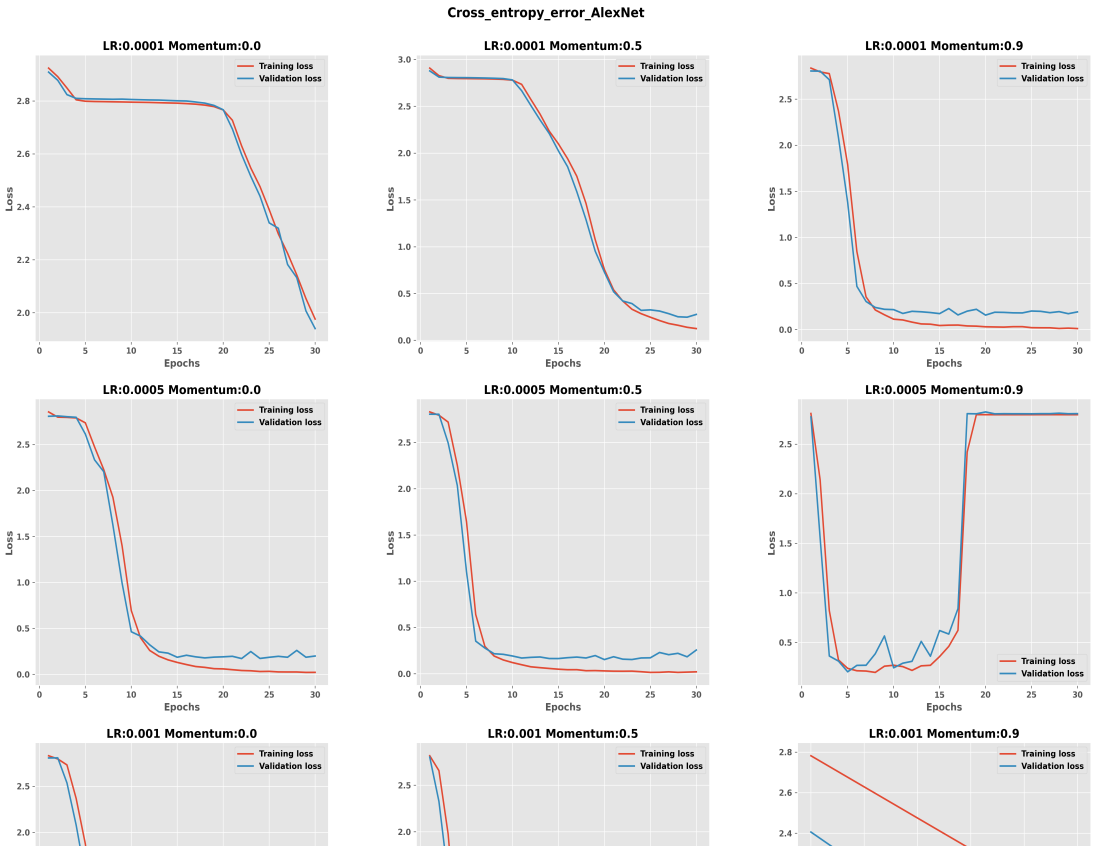
	Cross_entropy_error_train	Misclassification_error_train	Cross_entropy_error_test	Misclassification_error_test	Accuracy_test
0.0001_0.0	0.214056	5.890	0.385360	0.1166	88.34
0.0001_0.5	0.088439	2.430	0.335187	0.1008	89.92
0.0001_0.9	0.034580	1.130	0.325928	0.0784	92.16
0.0005_0.0	0.038561	1.060	0.324218	0.0874	91.26
0.0005_0.5	0.029855	0.950	0.294697	0.0738	92.62
0.0005_0.9	0.040189	1.310	0.371741	0.0796	92.04
0.001_0.0	0.027734	0.860	0.283284	0.0766	92.34
0.001_0.5	0.030850	1.065	0.345932	0.0712	92.88
0.001_0.9	0.053422	1.685	0.368555	0.0766	92.34
0.002_0.0	0.035921	1.145	0.355499	0.0752	92.48
0.002_0.5	0.038117	1.250	0.376363	0.0736	92.64
0.002_0.9	0.138103	3.755	0.576320	0.0874	91.26
0.005_0.0	0.046133	1.550	0.333250	0.0742	92.58
0.005_0.5	0.118998	3.810	0.435169	0.0956	90.44
0.005_0.9	2.807814	90.695	2.812674	0.9064	9.36
0.01_0.0	0.679548	21.225	0.755637	0.2306	76.94
0.01_0.5	0.142210	3.845	0.465800	0.0800	92.00
0.01_0.9	2.820941	90.555	2.815679	0.9064	9.36

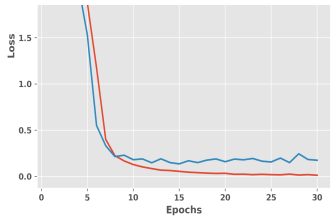
According to this grid search result, the best updated lenet should use:

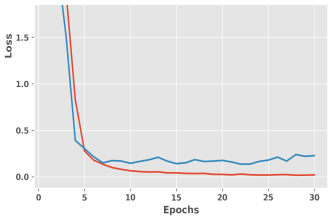
- $seed = 42, lr = 0.0005, momentum = 0.5$

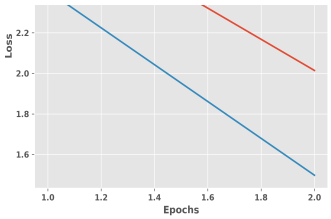
as its learning parameters to get the highest validation accuracy 92.62% and a low validation cross-entropy error.

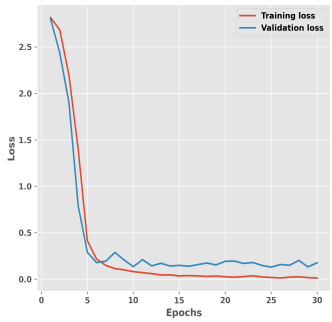
AlexNet

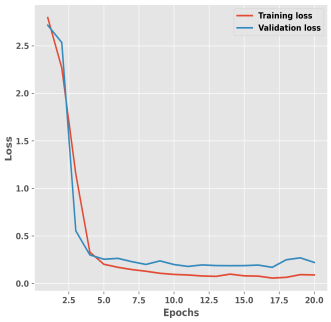


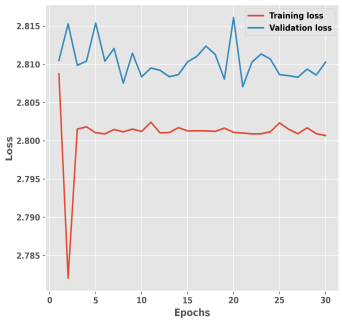


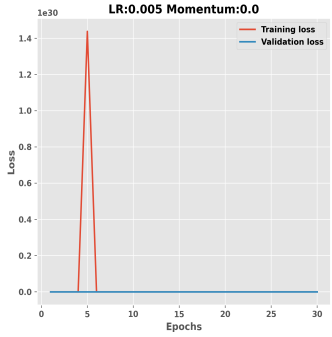


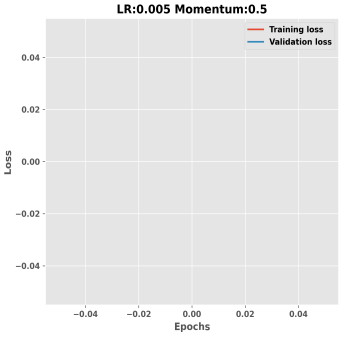


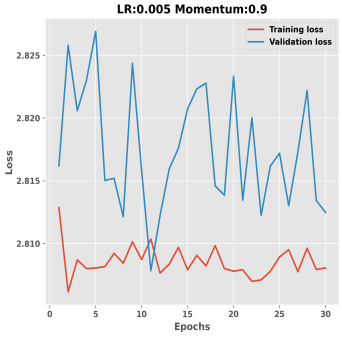


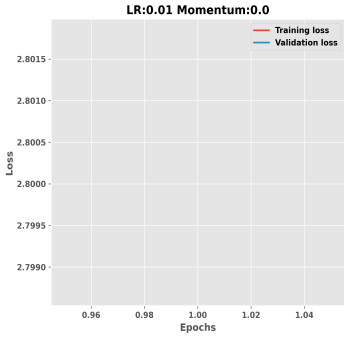


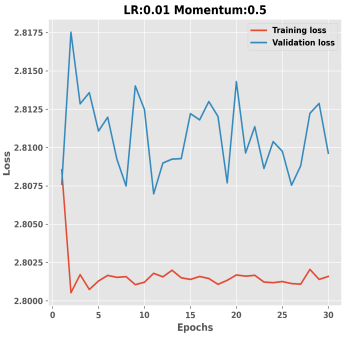


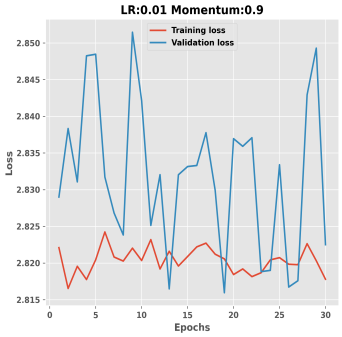




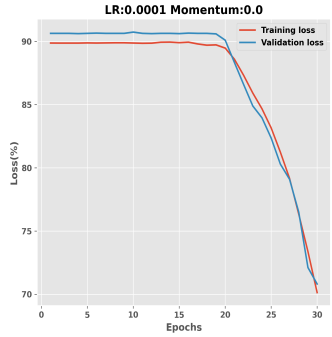


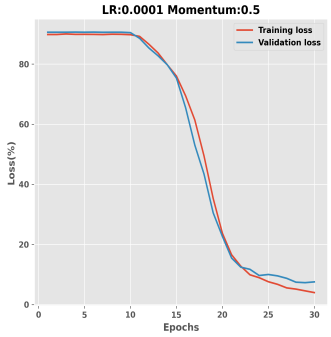


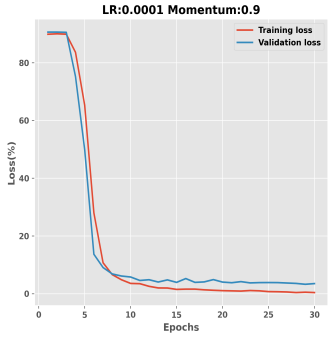


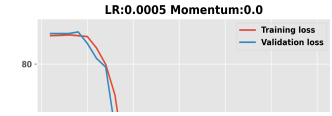


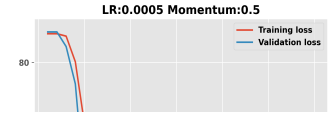
Misclassification_error_AlexNet

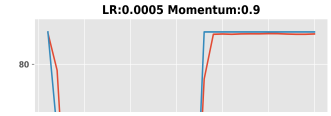


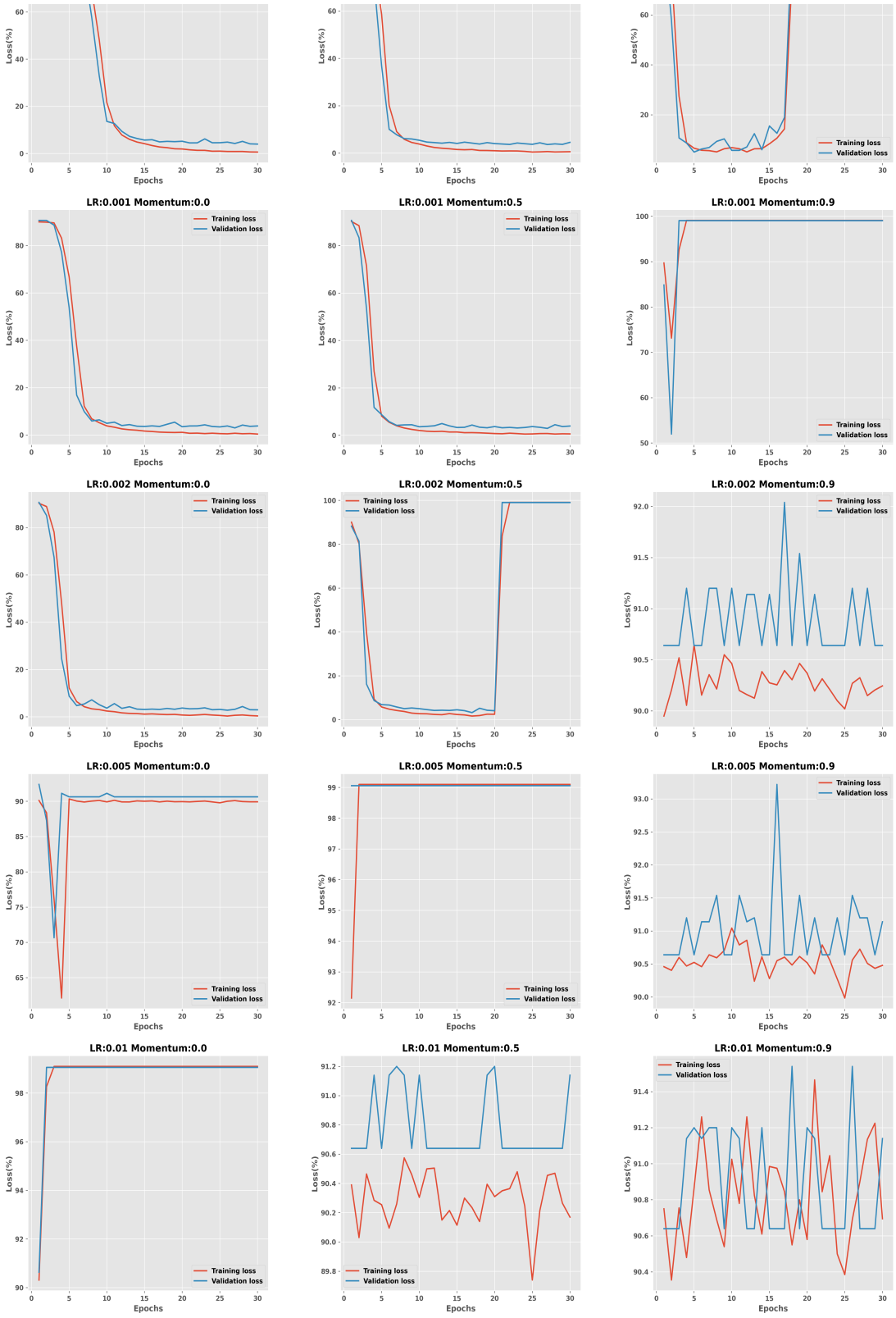












The learning curve of grid search is on $lr = {0.0001, 0.0005, 0.001, 0.002, 0.005, 0.01}$ and $momentum = {0.0, 0.5, 0.9}$. According to the subplots, alexnets are more sensitive to the learning rate even than the updated lenet in the previous question. Small learning rates like 0.0001 cannot let it converge in the first 30

epochs, while large learning rates like 0.005 and 0.01 cannot allow it to make optimization properly on this training dataset. The available range of learning rates for alexnet is the most narrow one we tried so far. Relatively large learning rates with large momentum would result in heavy fluctuations and divergence.

Some subplots show nothing or strange patterns, this might be the result of gradient explosion.

	Cross_entropy_error_train	Misclassification_error_train	Cross_entropy_error_test	Misclassification_error_test	Accuracy_test
0.0001_0.0	1.976305	70.140	1.940282	0.7082	29.18
0.0001_0.5	0.126353	4.000	0.277693	0.0756	92.44
0.0001_0.9	0.011991	0.405	0.192913	0.0352	96.48
0.0005_0.0	0.022062	0.640	0.199897	0.0402	95.98
0.0005_0.5	0.021864	0.600	0.257294	0.0456	95.44
0.0005_0.9	2.797876	89.980	2.808242	0.9064	9.36
0.001_0.0	0.013627	0.420	0.175670	0.0384	96.16
0.001_0.5	0.022155	0.615	0.229428	0.0394	96.06
0.001_0.9	NaN	99.100	NaN	0.9906	0.94
0.002_0.0	0.012248	0.395	0.176864	0.0298	97.02
0.002_0.5	NaN	99.100	NaN	0.9906	0.94
0.002_0.9	2.800701	90.245	2.810281	0.9064	9.36
0.005_0.0	2.797859	89.935	2.807767	0.9064	9.36
0.005_0.5	NaN	99.100	NaN	0.9906	0.94
0.005_0.9	2.808058	90.480	2.812489	0.9114	8.86
0.01_0.0	NaN	99.100	NaN	0.9906	0.94
0.01_0.5	2.801596	90.170	2.809637	0.9114	8.86
0.01_0.9	2.817800	90.695	2.822511	0.9114	8.86

The grid search result of alexnet

- $seed = 4442, lr = 0.002, momentum = 0.0$

finally attained the highest validation accuracy about 97.02% and beat the updated lenet.

Generalization Error

LeNet

```
Generalization result of lenet_alt:
Test Accuracy:  93.60%|Test Loss - cross entropy:    0.236467|Test Loss -
mis-clf:    0.064000
```

AlexNet

```
Generalization result of AlexNet:
Test Accuracy:  97.24%|Test Loss - cross entropy:    0.152874|Test Loss -
```

```
mis-clf:    0.027600
```

We finally use our best updated lenet and alexnet model picked in 7(d) to make predictions on the test set and give the generalization error. Note that updated lenet got validation cross-entropy error 0.294697 and validation accuracy 92.62%, alexnet got 0.176864 and 97.02% respectively. Both of the two best models outperform on the test data, and thus there are no overfitting problems in the training processes. However, both of the neural network structures got weaker performance than those working on original MNIST dataset. The best result on original MNIST dataset is the lowest test cross-entropy error of 0.032757 and the highest test accuracy of 99.22%

Explain why you cannot obtain a test error lower than 1%. The number of classes raised from 10 to 19, and the number of raw features are doubled in this question, which makes the real-world hypothesis more complex, thus it is natural that the classification performance got worse. If we assume that the features from a specific number are independent, as the class conditional assumption in naive Bayes, then the neural networks working on two digits would have the best test accuracy at $0.9922^2 \approx 0.9845$ to predict both digits correctly as two independent neural networks working on their own digit, which is less than 99%.