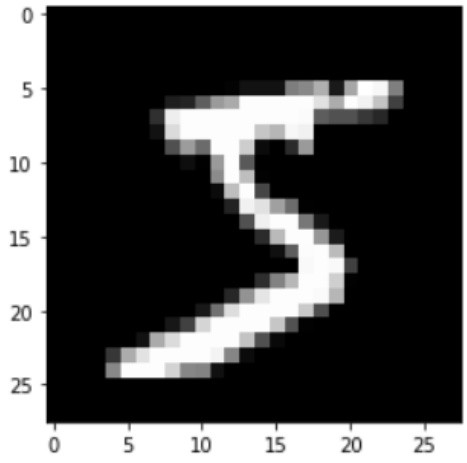
1.

(A). I see that the image matches with the label.

```
image = X_train[0]# plot the sample
fig = plt.figure
plt.imshow(image, cmap='gray')
plt.show()
```




```
Y_train[0]
```

```
5
```

(B). We know that the shape of X_train is 60000 * 784, the shape of X_test is 10000 * 784.

(C). By using one-hot encoding, we could analyze categorical variables much easier.

2.

(A). When k = 28 for KNN, I get the exact same test error.

```
knn = KNeighborsClassifier(n_neighbors = 28)
knn.fit(X_train, Y_train)

Y_pred = knn.predict(X_test)
```

```
Y_pred = knn.predict(X_test)
print(f'Accuracy for KNN classifier is: {metrics.accuracy_score(Y_test, Y_pred):0.3f}')
```

```
Accuracy for KNN classifier is: 0.950
```

When max depth equals 10 for decision tree, I get similar results

```
dt = DecisionTreeClassifier(max_depth = 10)
```

```
abc = AdaBoostClassifier(dt)
# Train Adaboost Classifer
model = abc.fit(X_train, Y_train_ori)

#Predict the response for test dataset
Y_pred = model.predict(X_test)
print(f'Accuracy for Adaboost classifier is: {metrics.accuracy_score(Y_test_ori, Y_pred):0.3f}')
```

```
Accuracy for Adaboost classifier is: 0.942
```

When using default values for SVM, I get similar results.

```
clf = svm.SVC()
clf.fit(X_train, Y_train_ori)
Y_pred = clf.predict(X_test)
print(f'Accuracy for Adaboost classifier is: {metrics.accuracy_score(Y_test_ori, Y_pred):0.3f}')
```

```
Accuracy for Adaboost classifier is: 0.979
```

(B).

```
from keras import Sequential
from keras.layers import Dense
nn = Sequential()
nn.add(Dense(800, activation = 'relu', input_dim = 784))
nn.add(Dense(10, activation = 'softmax'))
nn.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['acc'])
nn.fit(X_train, Y_train, epochs = 10)
```

```
Epoch 1/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.1897 - acc: 0.9445
Epoch 2/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0753 - acc: 0.9768
Epoch 3/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0491 - acc: 0.9845
Epoch 4/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0346 - acc: 0.9891
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0263 - acc: 0.9914
Epoch 6/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0194 - acc: 0.9937
Epoch 7/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0163 - acc: 0.9942
Epoch 8/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0164 - acc: 0.9945
Epoch 9/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0118 - acc: 0.9959
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0123 - acc: 0.9958
<keras.callbacks.History at 0x7ff8e6255690>
```

```
Y_pred = nn.predict(X_test)
Y_pred = (Y_pred > 0.5)
```

```
print(f'Accuracy for NN classifier is: {metrics.accuracy_score(Y_test, Y_pred):0.3f}')
```

```
Accuracy for NN classifier is: 0.974
```

I use the Neural Networks, Although I did not get a very satifactory result better than the score in the question, but I do get an output better than the previous three I gave above.

3. When I use simple forward ANN for modeling the MNIST dataset. For seed 1, 100, 1000, 10000 and 10086, under 200 epochs, the training loss and testing loss is generally going downward. This conclusion is the same for the misclassification rate.

(A).

```
torch.manual_seed(100)

class OurNet(torch.nn.Module):
    def __init__(self, n_feature, n_hidden, n_output):
        super(OurNet, self).__init__()
        self.hidden = torch.nn.Linear(n_feature, n_hidden)
        self.out = torch.nn.Linear(n_hidden, n_output)


    def forward(self, x):
        x = F.relu(self.hidden(x))
        x = self.out(x)
        return x

net2 = OurNet(n_feature=784, n_hidden=100, n_output=10)
optimizer = torch.optim.SGD(net2.parameters(), lr=0.1)
loss_func = torch.nn.CrossEntropyLoss()
```

```python
def train_single(model, epoch):
    train_out = []
    test_out = []
    for t in range(epoch):
        # forward passing
        out_tr = model(X_train_torch)
        train_out.append(out_tr)
        loss = loss_func(out_tr, Y_train_torch)
        loss_train.append(loss.detach().numpy())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        out_te = model(X_test_torch)
        test_out.append(out_te)
        loss1 = loss_func(out_te, Y_test_torch)
        loss_test.append(loss1.detach().numpy())
        if t%20 == 0:
        # printing the validation loss
            print('Epoch : ',t+1, '\t', 'train loss :', loss.detach().numpy())

    train_accuracy = []
    test_accuracy = []
    for t in range(epoch):
        train_prediction = torch.max(train_out[t],1)[1]   ##indice for maximum of each row
        test_prediction = torch.max(test_out[t],1)[1]
        pred_y_train = train_prediction.data.numpy()
        pred_y_test = test_prediction.data.numpy()
        target_y_train = Y_train_torch.data.numpy()
        target_y_test = Y_test_torch.data.numpy()
        train_accuracy.append(float((pred_y_train == target_y_train).astype(int).sum()) / float(target_y_train.size))
        test_accuracy.append(float((pred_y_test == target_y_test).astype(int).sum()) / float(target_y_test.size))

    for i in range(epoch):
        train_error.append(1-train_accuracy[i])
        test_error.append(1-test_accuracy[i])
    print("final test cross_entropy error loss is ", loss_test[epoch-1])
    print("final test mis-classification error is ", test_error[epoch-1])
```

detach: Any

seed = 1

```
Epoch :  1      train loss : 2.3179433
Epoch :  21     train loss : 1.9139081
Epoch :  41     train loss : 1.3017827
Epoch :  61     train loss : 0.89015293
Epoch :  81     train loss : 0.69705456
Epoch :  101    train loss : 0.5943162
Epoch :  121    train loss : 0.53120637
Epoch :  141    train loss : 0.4885208
Epoch :  161    train loss : 0.4577102
Epoch :  181    train loss : 0.43439847
final test cross_entropy error loss is  0.39722845
final test mis-classification error is  0.10550000000000004
```

Seed = 100

```
loss_train = []
loss_test = []
train_error = []
test_error = []
n_epochs = 200
train_single(net2, n_epochs)
```

```
Epoch :  1       train loss : 2.3200767
Epoch :  21      train loss : 1.911888
Epoch :  41      train loss : 1.2989203
Epoch :  61      train loss : 0.90516996
Epoch :  81      train loss : 0.7136704
Epoch :  101     train loss : 0.60797626
Epoch :  121     train loss : 0.5418562
Epoch :  141     train loss : 0.49685475
Epoch :  161     train loss : 0.46437457
Epoch :  181     train loss : 0.4398673
final test cross_entropy error loss is  0.40183088
final test mis-classification error is  0.1058
```

Seed = 1000

```
Epoch :  1       train loss : 2.296386
Epoch :  21      train loss : 1.8019158
Epoch :  41      train loss : 1.1913466
Epoch :  61      train loss : 0.8508828
Epoch :  81      train loss : 0.6827035
Epoch :  101     train loss : 0.5871998
Epoch :  121     train loss : 0.526611
Epoch :  141     train loss : 0.48513994
Epoch :  161     train loss : 0.45512223
Epoch :  181     train loss : 0.43241212
final test cross_entropy error loss is  0.39617485
final test mis-classification error is  0.10370000000000001
```

Seed = 10000

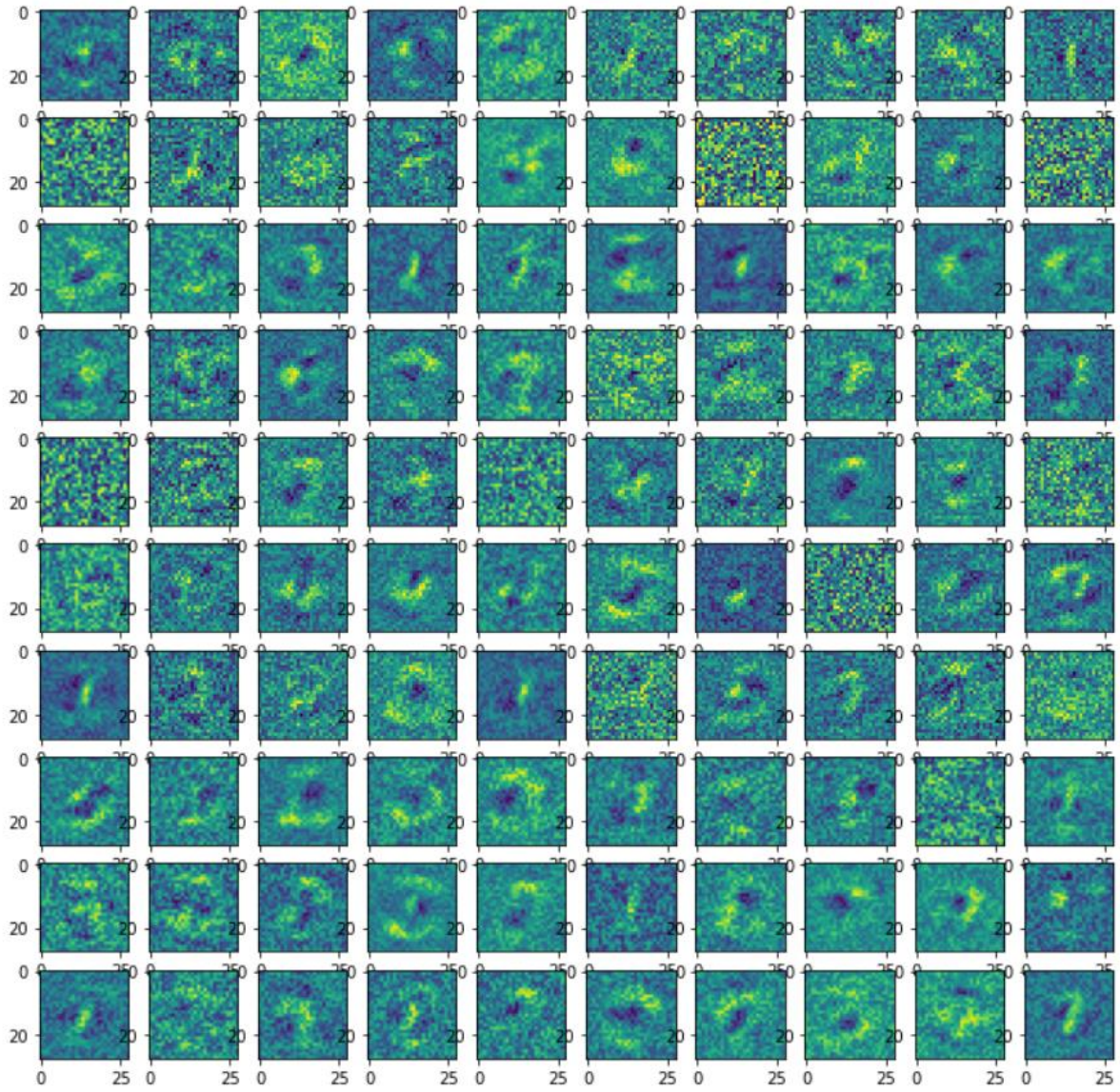
```
Epoch :  1       train loss : 2.3028762
Epoch :  21      train loss : 1.8523275
Epoch :  41      train loss : 1.2366866
Epoch :  61      train loss : 0.87856096
Epoch :  81      train loss : 0.70045006
Epoch :  101     train loss : 0.5999821
Epoch :  121     train loss : 0.5363248
Epoch :  141     train loss : 0.4926817
Epoch :  161     train loss : 0.4610526
Epoch :  181     train loss : 0.43713155
final test cross_entropy error loss is  0.39844418
final test mis-classification error is  0.10609999999999997
```

Seed = 10086

```
Epoch :  1       train loss : 2.3087256
Epoch :  21      train loss : 1.9234742
Epoch :  41      train loss : 1.2990752
Epoch :  61      train loss : 0.8862418
Epoch :  81      train loss : 0.69437075
Epoch :  101     train loss : 0.5928858
Epoch :  121     train loss : 0.5307699
Epoch :  141     train loss : 0.4888453
Epoch :  161     train loss : 0.45860127
Epoch :  181     train loss : 0.43571365
final test cross_entropy error loss is  0.39783475
final test mis-classification error is  0.10670000000000002
```

(B) Except for the numbers, the general pattern for Entropy Loss is the same as misclassification rate.

(C). I did not see any particular structure in this graph.



(d) From comparing the test loss of the above four seeds, I choose to continue with seed = 1000, which gives the smallest test loss.

Learning rate largely affects the convergence rate for simple ANN. For lr = 0.01, the model performs poorly after 200 epochs. While the speed for convergence increases as learning rate increases. When learning rate becomes 0.5, I get the best classification error and entropy loss among lr = 0.01, 0.1, 0.2, 0.5.

Momentum works similarly for simple forward ANN, when momentum increases, performance increases accordingly, I get best performance when momentum becomes 0.9.

Lr = 0.01

```
Epoch :  1       train loss : 2.296386
Epoch :  21      train loss : 2.2592006
Epoch :  41      train loss : 2.220688
Epoch :  61      train loss : 2.1789525
Epoch :  81      train loss : 2.1334338
Epoch :  101     train loss : 2.084217
Epoch :  121     train loss : 2.0315156
Epoch :  141     train loss : 1.9755327
Epoch :  161     train loss : 1.9164618
Epoch :  181     train loss : 1.8546257
final test cross_entropy error loss is  1.7806008
final test mis-classification error is  0.2995
```

Lr = 0.2

```
Epoch :  1       train loss : 2.3195474
Epoch :  21      train loss : 1.2630275
Epoch :  41      train loss : 0.6998918
Epoch :  61      train loss : 0.534946
Epoch :  81      train loss : 0.4603239
Epoch :  101     train loss : 0.4182121
Epoch :  121     train loss : 0.39098087
Epoch :  141     train loss : 0.37163755
Epoch :  161     train loss : 0.35692775
Epoch :  181     train loss : 0.34517875
final test cross_entropy error loss is  0.3211757
final test mis-classification error is  0.09019999999999995
```

Lr = 0.5

```
Epoch :  1       train loss : 2.3193634
Epoch :  21      train loss : 1.0737603
Epoch :  41      train loss : 0.5581579
Epoch :  61      train loss : 0.38928264
Epoch :  81      train loss : 0.33435163
Epoch :  101     train loss : 0.30928716
Epoch :  121     train loss : 0.29197738
Epoch :  141     train loss : 0.27691823
Epoch :  161     train loss : 0.26261207
Epoch :  181     train loss : 0.2504599
final test cross_entropy error loss is  0.23343064
final test mis-classification error is  0.06499999999999995
```

Momentum = 0.5

```
Epoch :  1       train loss : 2.3072257
Epoch :  21      train loss : 1.330273
Epoch :  41      train loss : 0.69894934
Epoch :  61      train loss : 0.53122586
Epoch :  81      train loss : 0.45808294
Epoch :  101     train loss : 0.41696918
Epoch :  121     train loss : 0.39022738
Epoch :  141     train loss : 0.3710389
Epoch :  161     train loss : 0.3562618
Epoch :  181     train loss : 0.3442924
final test cross_entropy error loss is  0.32080445
final test mis-classification error is  0.090099999999999996
```

Momentum = 0.9

```
Epoch :  1       train loss : 2.303581
Epoch :  21      train loss : 0.53810424
Epoch :  41      train loss : 0.38178256
Epoch :  61      train loss : 0.32105508
Epoch :  81      train loss : 0.29335892
Epoch :  101     train loss : 0.27416188
Epoch :  121     train loss : 0.2575563
Epoch :  141     train loss : 0.2425928
Epoch :  161     train loss : 0.22903956
Epoch :  181     train loss : 0.216793
final test cross_entropy error loss is  0.20503423
final test mis-classification error is  0.059699999999999975
```

4. When I use CNN for modeling the MNIST dataset. For seed 1, 100, 1000, 10000 and 10086, under 200 epochs, the training loss will become lower when epochs grow, but the testing loss will tend to maintain still or even become larger when epochs grow. I suppose it's because I trained the model to overfitting. For the misclassification rate, there is not much difference from the conclusion of entropy loss.

(a).

```python
torch.manual_seed(1)
class CNNNet(torch.nn.Module):
    def __init__(self):
        super(CNNNet, self).__init__()
        self.cnn_layers = torch.nn.Sequential(
            # Defining a 2D convolution layer
            torch.nn.Conv2d(1, 4, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(inplace=True),
            torch.nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.out = torch.nn.Linear(4 * 14 * 14, 10)
    # Defining the forward pass
    def forward(self, x):
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.out(x)
        return x
```

```python
def get_accuracy(logit, target, batch_size):
    ''' Obtain accuracy for training round '''
    corrects = (torch.max(logit, 1)[1].view(target.size()).data == target.data).sum()
    accuracy = corrects/batch_size
    return accuracy.item()
```

```python
def train_CNN(model, epoch):
    for t in range(epoch):
        train_running_loss = 0.0
        train_acc = 0.0
        for step, (batch_x, batch_y) in enumerate(trainloader):
            # forward passing
            out_tr = model(batch_x)
            loss = loss_func(out_tr, batch_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_running_loss += loss.detach().numpy()
            train_acc += get_accuracy(out_tr, batch_y, BATCH_SIZE)
        train_loss.append(train_running_loss / step)
        train_accuracy.append(train_acc/step)
        if t%20 == 0:
            print('Epoch:  %d | Loss: %.4f | Train Accuracy: %.2f'
                %(t, train_running_loss / step, train_acc/step))
        out_te = model(X_test_torch)
        test_prediction = torch.max(out_te,1)[1]
        pred_y_test = test_prediction.data.numpy()
        target_y_test = Y_test_torch.data.numpy()
        loss1 = loss_func(out_te, Y_test_torch)
        test_loss.append(loss1.detach().numpy())
        test_accuracy.append(float((pred_y_test == target_y_test).astype(int).sum()) / float(target_y_test.size))

    for i in range(epoch):
        train_error.append(1-train_accuracy[i])
        test_error.append(1-test_accuracy[i])
    print("final test cross_entropy error loss for CNN is ", test_loss[epoch-1])
    print("final test mis-classification error for CNN is ", test_error[epoch-1])
```

Seed = 1

```
CNNnet1 = CNNNet()
# defining the optimizer
optimizer = torch.optim.SGD(CNNnet1.parameters(), lr=0.1)
# defining the loss function
loss_func = torch.nn.CrossEntropyLoss()
if torch.cuda.is_available():
    CNNnet1 = CNNnet1.cuda()
    criterion = criterion.cuda()

n_epochs = 200
train_loss = []
test_loss = []
train_accuracy = []
test_accuracy = []
test_error = []
train_error = []
train_CNN(CNNnet1, n_epochs)
```

```
Epoch:  0 | Loss: 0.4452 | Train Accuracy: 0.87
Epoch:  20 | Loss: 0.0733 | Train Accuracy: 0.98
Epoch:  40 | Loss: 0.0638 | Train Accuracy: 0.98
Epoch:  60 | Loss: 0.0591 | Train Accuracy: 0.98
Epoch:  80 | Loss: 0.0559 | Train Accuracy: 0.98
Epoch:  100 | Loss: 0.0539 | Train Accuracy: 0.98
Epoch:  120 | Loss: 0.0517 | Train Accuracy: 0.98
Epoch:  140 | Loss: 0.0501 | Train Accuracy: 0.98
Epoch:  160 | Loss: 0.0497 | Train Accuracy: 0.98
Epoch:  180 | Loss: 0.0486 | Train Accuracy: 0.98
final test cross_entropy error loss for CNN is  0.12540391
final test mis-classification error for CNN is  0.030200000000000005
```

Seed = 100

```
Epoch:  0 | Loss: 0.4098 | Train Accuracy: 0.88
Epoch:  20 | Loss: 0.0499 | Train Accuracy: 0.99
Epoch:  40 | Loss: 0.0379 | Train Accuracy: 0.99
Epoch:  60 | Loss: 0.0310 | Train Accuracy: 0.99
Epoch:  80 | Loss: 0.0277 | Train Accuracy: 0.99
Epoch:  100 | Loss: 0.0239 | Train Accuracy: 0.99
Epoch:  120 | Loss: 0.0221 | Train Accuracy: 0.99
Epoch:  140 | Loss: 0.0201 | Train Accuracy: 0.99
Epoch:  160 | Loss: 0.0182 | Train Accuracy: 0.99
Epoch:  180 | Loss: 0.0168 | Train Accuracy: 1.00
final test cross_entropy error loss for CNN is  0.1742442
final test mis-classification error for CNN is  0.03390000000000004
```

Seed = 1000

```
Epoch:  0 | Loss: 0.4230 | Train Accuracy: 0.87
Epoch:  20 | Loss: 0.0763 | Train Accuracy: 0.98
Epoch:  40 | Loss: 0.0628 | Train Accuracy: 0.98
Epoch:  60 | Loss: 0.0571 | Train Accuracy: 0.98
Epoch:  80 | Loss: 0.0523 | Train Accuracy: 0.98
Epoch:  100 | Loss: 0.0498 | Train Accuracy: 0.98
Epoch:  120 | Loss: 0.0472 | Train Accuracy: 0.99
Epoch:  140 | Loss: 0.0462 | Train Accuracy: 0.99
Epoch:  160 | Loss: 0.0444 | Train Accuracy: 0.99
Epoch:  180 | Loss: 0.0434 | Train Accuracy: 0.99
final test cross_entropy error loss for CNN is  0.121508166
final test mis-classification error for CNN is  0.02739999999999998
```

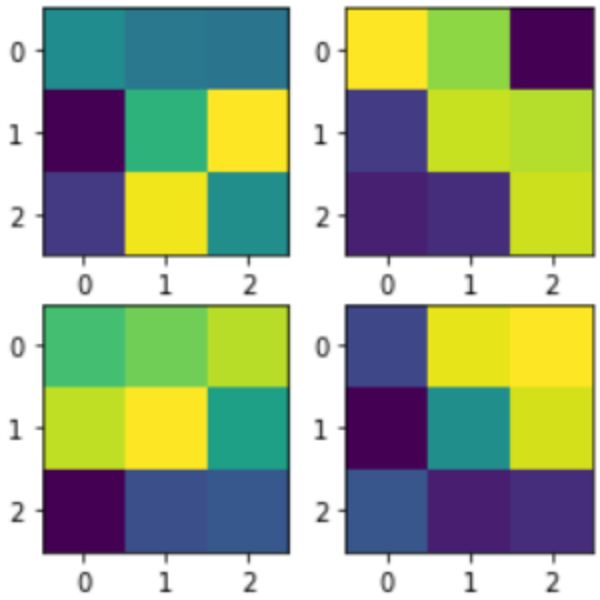
Seed = 10000

```
Epoch:  0 | Loss: 0.3797 | Train Accuracy: 0.89
Epoch:  20 | Loss: 0.0556 | Train Accuracy: 0.98
Epoch:  40 | Loss: 0.0446 | Train Accuracy: 0.99
Epoch:  60 | Loss: 0.0386 | Train Accuracy: 0.99
Epoch:  80 | Loss: 0.0341 | Train Accuracy: 0.99
Epoch:  100 | Loss: 0.0313 | Train Accuracy: 0.99
Epoch:  120 | Loss: 0.0290 | Train Accuracy: 0.99
Epoch:  140 | Loss: 0.0269 | Train Accuracy: 0.99
Epoch:  160 | Loss: 0.0251 | Train Accuracy: 0.99
Epoch:  180 | Loss: 0.0237 | Train Accuracy: 0.99
final test cross_entropy error loss for CNN is  0.15225099
final test mis-classification error for CNN is  0.02849999999999997
```

Seed = 10086

```
Epoch:  0 | Loss: 0.3542 | Train Accuracy: 0.89
Epoch:  20 | Loss: 0.0565 | Train Accuracy: 0.98
Epoch:  40 | Loss: 0.0437 | Train Accuracy: 0.99
Epoch:  60 | Loss: 0.0359 | Train Accuracy: 0.99
Epoch:  80 | Loss: 0.0309 | Train Accuracy: 0.99
Epoch:  100 | Loss: 0.0273 | Train Accuracy: 0.99
Epoch:  120 | Loss: 0.0253 | Train Accuracy: 0.99
Epoch:  140 | Loss: 0.0229 | Train Accuracy: 0.99
Epoch:  160 | Loss: 0.0208 | Train Accuracy: 0.99
Epoch:  180 | Loss: 0.0192 | Train Accuracy: 0.99
final test cross_entropy error loss for CNN is  0.1375785
final test mis-classification error for CNN is  0.025699999999999945
```

(b). The general pattern is more smooth than entropy loss.(C). Looking at the graph, I don't think it has any structure.



(D). From the above seeds, I choose to continue with seed = 10086.

For learning rate, generally lower learning rate seems to perform better. When lr = 0.01, the model performs best.

For momentum, it's the same, lower momentum gives better result. When momentum = 0.9, the result is poorest.

Lr = 0.01

```
Epoch:  0 | Loss: 0.7409 | Train Accuracy: 0.81
Epoch:  20 | Loss: 0.1396 | Train Accuracy: 0.96
Epoch:  40 | Loss: 0.0966 | Train Accuracy: 0.97
Epoch:  60 | Loss: 0.0782 | Train Accuracy: 0.98
Epoch:  80 | Loss: 0.0686 | Train Accuracy: 0.98
Epoch:  100 | Loss: 0.0623 | Train Accuracy: 0.98
Epoch:  120 | Loss: 0.0578 | Train Accuracy: 0.98
Epoch:  140 | Loss: 0.0540 | Train Accuracy: 0.98
Epoch:  160 | Loss: 0.0510 | Train Accuracy: 0.99
Epoch:  180 | Loss: 0.0485 | Train Accuracy: 0.99
final test cross_entropy error loss for CNN is  0.07042205
final test mis-classification error for CNN is  0.021399999999999975
```

Lr = 0.2

```
Epoch:  0 | Loss: 0.3110 | Train Accuracy: 0.91
Epoch:  20 | Loss: 0.0506 | Train Accuracy: 0.98
Epoch:  40 | Loss: 0.0407 | Train Accuracy: 0.99
Epoch:  60 | Loss: 0.0357 | Train Accuracy: 0.99
Epoch:  80 | Loss: 0.0321 | Train Accuracy: 0.99
Epoch:  100 | Loss: 0.0310 | Train Accuracy: 0.99
Epoch:  120 | Loss: 0.0286 | Train Accuracy: 0.99
Epoch:  140 | Loss: 0.0269 | Train Accuracy: 0.99
Epoch:  160 | Loss: 0.0243 | Train Accuracy: 0.99
Epoch:  180 | Loss: 0.0231 | Train Accuracy: 0.99
final test cross_entropy error loss for CNN is  0.18426909
final test mis-classification error for CNN is  0.030100000000000016
```

Lr = 0.5

```
Epoch:  0 | Loss: 2.3023 | Train Accuracy: 0.11
Epoch:  20 | Loss: 2.3055 | Train Accuracy: 0.11
Epoch:  40 | Loss: 2.3054 | Train Accuracy: 0.11
Epoch:  60 | Loss: 2.3053 | Train Accuracy: 0.11
Epoch:  80 | Loss: 2.3056 | Train Accuracy: 0.11
Epoch:  100 | Loss: 2.3054 | Train Accuracy: 0.11
Epoch:  120 | Loss: 2.3052 | Train Accuracy: 0.11
Epoch:  140 | Loss: 2.3055 | Train Accuracy: 0.11
Epoch:  160 | Loss: 2.3056 | Train Accuracy: 0.11
Epoch:  180 | Loss: 2.3054 | Train Accuracy: 0.11
final test cross_entropy error loss for CNN is  2.3049736
final test mis-classification error for CNN is  0.902
```

Momentum = 0.5

```
Epoch:  0 | Loss: 0.3033 | Train Accuracy: 0.91
Epoch:  20 | Loss: 0.0559 | Train Accuracy: 0.98
Epoch:  40 | Loss: 0.0438 | Train Accuracy: 0.99
Epoch:  60 | Loss: 0.0370 | Train Accuracy: 0.99
Epoch:  80 | Loss: 0.0331 | Train Accuracy: 0.99
Epoch:  100 | Loss: 0.0291 | Train Accuracy: 0.99
Epoch:  120 | Loss: 0.0269 | Train Accuracy: 0.99
Epoch:  140 | Loss: 0.0252 | Train Accuracy: 0.99
Epoch:  160 | Loss: 0.0248 | Train Accuracy: 0.99
Epoch:  180 | Loss: 0.0223 | Train Accuracy: 0.99
final test cross_entropy error loss for CNN is  0.17201342
final test mis-classification error for CNN is  0.02749999999999997
```

Momentum = 0.9

```
Epoch:  0 | Loss: 0.4163 | Train Accuracy: 0.88
Epoch:  20 | Loss: 0.3496 | Train Accuracy: 0.90
Epoch:  40 | Loss: 0.3391 | Train Accuracy: 0.91
Epoch:  60 | Loss: 0.3463 | Train Accuracy: 0.90
Epoch:  80 | Loss: 0.3435 | Train Accuracy: 0.90
Epoch:  100 | Loss: 0.3442 | Train Accuracy: 0.90
Epoch:  120 | Loss: 0.3508 | Train Accuracy: 0.90
Epoch:  140 | Loss: 0.3662 | Train Accuracy: 0.90
Epoch:  160 | Loss: 0.3407 | Train Accuracy: 0.90
Epoch:  180 | Loss: 0.3484 | Train Accuracy: 0.90
final test cross_entropy error loss for CNN is  0.34495714
final test mis-classification error for CNN is  0.09160000000000001
```

5. I use CNN with batch nomalization and drop out as my favorite model.

The general pattern for training loss goes downward. While for testing, although the pattern is downward, the points are more scattered out. This conclusion also applied to misclassification rate.

```
torch.manual_seed(1)
class CNNNet(torch.nn.Module):
    def __init__(self):
        super(CNNNet, self).__init__()
        self.cnn_layers1 = torch.nn.Sequential(
            # Defining a 2D convolution layer
            torch.nn.Conv2d(1, 4, kernel_size=3, stride=1, padding=1),
            torch.nn.BatchNorm2d(4),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            torch.nn.Dropout(p=0.2))
        self.out = torch.nn.Linear(4 * 14 * 14, 10)


    # Defining the forward pass
    def forward(self, x):
        x = self.cnn_layers1(x)
        x = x.view(x.size(0), -1)
        x = self.out(x)
        return x
```

```
favnet1 = CNNNet()
# defining the optimizer
optimizer = torch.optim.SGD(favnet1.parameters(), lr=0.1)
# defining the loss function
loss_func = torch.nn.CrossEntropyLoss()

n_epochs = 200
train_loss = []
test_loss = []
train_accuracy = []
test_accuracy = []
test_error = []
train_error = []

train_CNN(favnet1, n_epochs)
```

```python
def train_CNN(model, epoch):
    for t in range(epoch):
        train_running_loss = 0.0
        train_acc = 0.0
        for step, (batch_x, batch_y) in enumerate(trainloader):
            # forward passing
            out_tr = model(batch_x)
            loss = loss_func(out_tr, batch_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            train_running_loss += loss.detach().numpy()
            train_acc += get_accuracy(out_tr, batch_y, BATCH_SIZE)
        train_loss.append(train_running_loss / step)
        train_accuracy.append(train_acc/step)
        if t%20 == 0:
            print('Epoch:  %d | Loss: %.4f | Train Accuracy: %.2f'
                %(t, train_running_loss / step, train_acc/step))
        out_te = model(X_test_torch)
        test_prediction = torch.max(out_te,1)[1]
        pred_y_test = test_prediction.data.numpy()
        target_y_test = Y_test_torch.data.numpy()
        loss1 = loss_func(out_te, Y_test_torch)
        test_loss.append(loss1.detach().numpy())
        test_accuracy.append(float((pred_y_test == target_y_test).astype(int).sum()) / float(target_y_test.size))

    for i in range(epoch):
        train_error.append(1-train_accuracy[i])
        test_error.append(1-test_accuracy[i])
    print("final test cross_entropy error loss for CNN is ", test_loss[epoch-1])
    print("final test mis-classification error for CNN is ", test_error[epoch-1])
```

(A).

Seed = 1

```
Epoch:  0 | Loss: 0.2898 | Train Accuracy: 0.91
Epoch:  20 | Loss: 0.1197 | Train Accuracy: 0.96
Epoch:  40 | Loss: 0.1126 | Train Accuracy: 0.96
Epoch:  60 | Loss: 0.1074 | Train Accuracy: 0.97
Epoch:  80 | Loss: 0.1080 | Train Accuracy: 0.97
Epoch:  100 | Loss: 0.1063 | Train Accuracy: 0.97
Epoch:  120 | Loss: 0.1052 | Train Accuracy: 0.97
Epoch:  140 | Loss: 0.1048 | Train Accuracy: 0.97
Epoch:  160 | Loss: 0.1039 | Train Accuracy: 0.97
Epoch:  180 | Loss: 0.1006 | Train Accuracy: 0.97
final test cross_entropy error loss for CNN is  0.110939845
final test mis-classification error for CNN is  0.03359999999999996
```

seed = 100

```
Epoch:  0 | Loss: 0.2765 | Train Accuracy: 0.92
Epoch:  20 | Loss: 0.0946 | Train Accuracy: 0.97
Epoch:  40 | Loss: 0.0887 | Train Accuracy: 0.97
Epoch:  60 | Loss: 0.0843 | Train Accuracy: 0.97
Epoch:  80 | Loss: 0.0841 | Train Accuracy: 0.97
Epoch:  100 | Loss: 0.0813 | Train Accuracy: 0.97
Epoch:  120 | Loss: 0.0777 | Train Accuracy: 0.97
Epoch:  140 | Loss: 0.0794 | Train Accuracy: 0.98
Epoch:  160 | Loss: 0.0793 | Train Accuracy: 0.97
Epoch:  180 | Loss: 0.0779 | Train Accuracy: 0.98
final test cross_entropy error loss for CNN is  0.10308717
final test mis-classification error for CNN is  0.031000000000000028
```

Seed = 1000

```
Epoch:  0 | Loss: 0.3980 | Train Accuracy: 0.88
Epoch:  20 | Loss: 0.0889 | Train Accuracy: 0.97
Epoch:  40 | Loss: 0.0828 | Train Accuracy: 0.97
Epoch:  60 | Loss: 0.0801 | Train Accuracy: 0.97
Epoch:  80 | Loss: 0.0799 | Train Accuracy: 0.97
Epoch:  100 | Loss: 0.0782 | Train Accuracy: 0.98
Epoch:  120 | Loss: 0.0766 | Train Accuracy: 0.98
Epoch:  140 | Loss: 0.0725 | Train Accuracy: 0.98
Epoch:  160 | Loss: 0.0730 | Train Accuracy: 0.98
Epoch:  180 | Loss: 0.0726 | Train Accuracy: 0.98
final test cross_entropy error loss for CNN is  0.09350781
final test mis-classification error for CNN is  0.030299999999999994
```

Seed = 10000

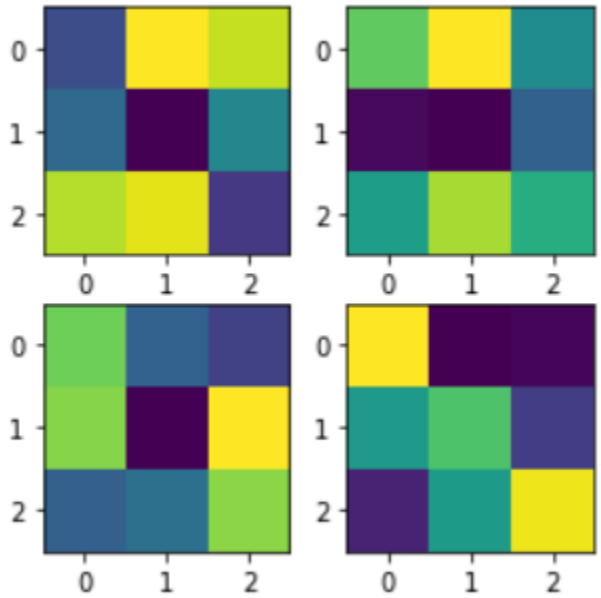
```
Epoch:  0 | Loss: 0.2840 | Train Accuracy: 0.91
Epoch:  20 | Loss: 0.0977 | Train Accuracy: 0.97
Epoch:  40 | Loss: 0.0919 | Train Accuracy: 0.97
Epoch:  60 | Loss: 0.0914 | Train Accuracy: 0.97
Epoch:  80 | Loss: 0.0862 | Train Accuracy: 0.97
Epoch:  100 | Loss: 0.0837 | Train Accuracy: 0.97
Epoch:  120 | Loss: 0.0813 | Train Accuracy: 0.97
Epoch:  140 | Loss: 0.0816 | Train Accuracy: 0.97
Epoch:  160 | Loss: 0.0822 | Train Accuracy: 0.97
Epoch:  180 | Loss: 0.0788 | Train Accuracy: 0.98
final test cross_entropy error loss for CNN is  0.10225743
final test mis-classification error for CNN is  0.03059999999999996
```

Seed = 10086

```
Epoch:  0 | Loss: 0.2695 | Train Accuracy: 0.92
Epoch:  20 | Loss: 0.1028 | Train Accuracy: 0.97
Epoch:  40 | Loss: 0.0986 | Train Accuracy: 0.97
Epoch:  60 | Loss: 0.0972 | Train Accuracy: 0.97
Epoch:  80 | Loss: 0.0939 | Train Accuracy: 0.97
Epoch:  100 | Loss: 0.0919 | Train Accuracy: 0.97
Epoch:  120 | Loss: 0.0915 | Train Accuracy: 0.97
Epoch:  140 | Loss: 0.0896 | Train Accuracy: 0.97
Epoch:  160 | Loss: 0.0875 | Train Accuracy: 0.97
Epoch:  180 | Loss: 0.0888 | Train Accuracy: 0.97
final test cross_entropy error loss for CNN is  0.111532375
final test mis-classification error for CNN is  0.03169999999999995
```

(B). Pattern is generally the same as entropy loss

(C). I did not see any particular structure in weights.



(D). I go along with seed = 1000, generally, smaller learning rate gives better performance, when learning rate = 0.5, the performance is worst.

This also applies to momentum, larger momentum generally not perform as well as smaller ones.

Lr = 0.01

```
Epoch:  0 | Loss: 0.6106 | Train Accuracy: 0.82
Epoch:  20 | Loss: 0.1110 | Train Accuracy: 0.97
Epoch:  40 | Loss: 0.0953 | Train Accuracy: 0.97
Epoch:  60 | Loss: 0.0896 | Train Accuracy: 0.97
Epoch:  80 | Loss: 0.0866 | Train Accuracy: 0.97
Epoch:  100 | Loss: 0.0843 | Train Accuracy: 0.97
Epoch:  120 | Loss: 0.0835 | Train Accuracy: 0.97
Epoch:  140 | Loss: 0.0784 | Train Accuracy: 0.98
Epoch:  160 | Loss: 0.0768 | Train Accuracy: 0.98
Epoch:  180 | Loss: 0.0774 | Train Accuracy: 0.98
final test cross_entropy error loss for CNN is  0.08785416
final test mis-classification error for CNN is  0.028200000000000003
```

Lr = 0.2

```
Epoch:  0 | Loss: 0.4104 | Train Accuracy: 0.87
Epoch:  20 | Loss: 0.2401 | Train Accuracy: 0.93
Epoch:  40 | Loss: 0.2175 | Train Accuracy: 0.93
Epoch:  60 | Loss: 0.2069 | Train Accuracy: 0.94
Epoch:  80 | Loss: 0.4381 | Train Accuracy: 0.86
Epoch:  100 | Loss: 0.4330 | Train Accuracy: 0.87
Epoch:  120 | Loss: 0.4334 | Train Accuracy: 0.87
Epoch:  140 | Loss: 0.4305 | Train Accuracy: 0.87
Epoch:  160 | Loss: 0.4267 | Train Accuracy: 0.87
Epoch:  180 | Loss: 0.4298 | Train Accuracy: 0.87
final test cross_entropy error loss for CNN is  0.44292855
final test mis-classification error for CNN is  0.12729999999999997
```

Lr = 0.5

```
Epoch:  0 | Loss: 2.3145 | Train Accuracy: 0.11
Epoch:  20 | Loss: 2.3056 | Train Accuracy: 0.11
Epoch:  40 | Loss: 2.3056 | Train Accuracy: 0.11
Epoch:  60 | Loss: 2.3054 | Train Accuracy: 0.11
Epoch:  80 | Loss: 2.3057 | Train Accuracy: 0.11
Epoch:  100 | Loss: 2.3054 | Train Accuracy: 0.11
Epoch:  120 | Loss: 2.3055 | Train Accuracy: 0.11
Epoch:  140 | Loss: 2.3055 | Train Accuracy: 0.11
Epoch:  160 | Loss: 2.3056 | Train Accuracy: 0.11
Epoch:  180 | Loss: 2.3055 | Train Accuracy: 0.11
final test cross_entropy error loss for CNN is  2.3019936
final test mis-classification error for CNN is  0.8865
```

Momentum = 0.5

```
Epoch:  0 | Loss: 0.5618 | Train Accuracy: 0.83
Epoch:  20 | Loss: 0.2025 | Train Accuracy: 0.94
Epoch:  40 | Loss: 0.1940 | Train Accuracy: 0.94
Epoch:  60 | Loss: 0.1936 | Train Accuracy: 0.94
Epoch:  80 | Loss: 0.1957 | Train Accuracy: 0.94
Epoch:  100 | Loss: 0.1925 | Train Accuracy: 0.94
Epoch:  120 | Loss: 0.1947 | Train Accuracy: 0.94
Epoch:  140 | Loss: 0.1909 | Train Accuracy: 0.94
Epoch:  160 | Loss: 0.1945 | Train Accuracy: 0.94
Epoch:  180 | Loss: 0.1913 | Train Accuracy: 0.94
final test cross_entropy error loss for CNN is  0.19057769
final test mis-classification error for CNN is  0.059699999999999975
```

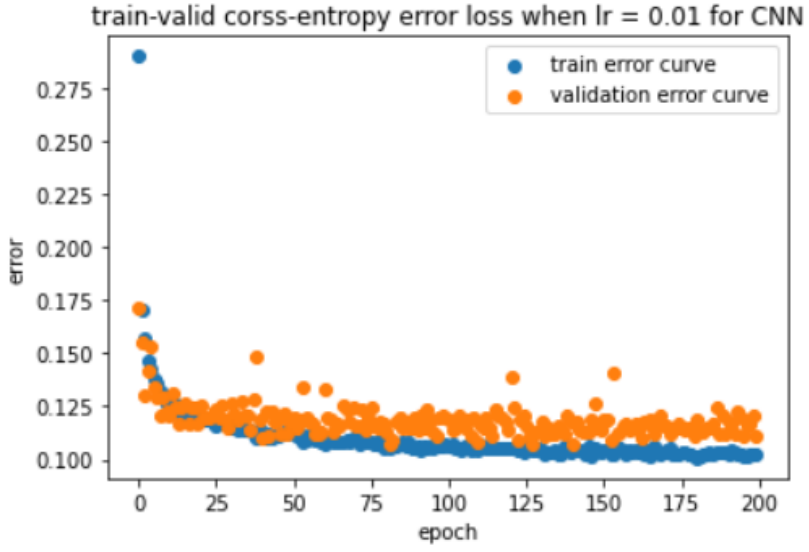
Momentum = 0.9

```
Epoch:  0 | Loss: 0.2532 | Train Accuracy: 0.93
Epoch:  20 | Loss: 0.1210 | Train Accuracy: 0.96
Epoch:  40 | Loss: 0.1528 | Train Accuracy: 0.95
Epoch:  60 | Loss: 0.1467 | Train Accuracy: 0.96
Epoch:  80 | Loss: 0.1470 | Train Accuracy: 0.96
Epoch:  100 | Loss: 0.1437 | Train Accuracy: 0.96
Epoch:  120 | Loss: 0.1447 | Train Accuracy: 0.96
Epoch:  140 | Loss: 0.1435 | Train Accuracy: 0.96
Epoch:  160 | Loss: 0.1411 | Train Accuracy: 0.96
Epoch:  180 | Loss: 0.1431 | Train Accuracy: 0.96
final test cross_entropy error loss for CNN is  0.15150012
final test mis-classification error for CNN is  0.04479999999999995
```

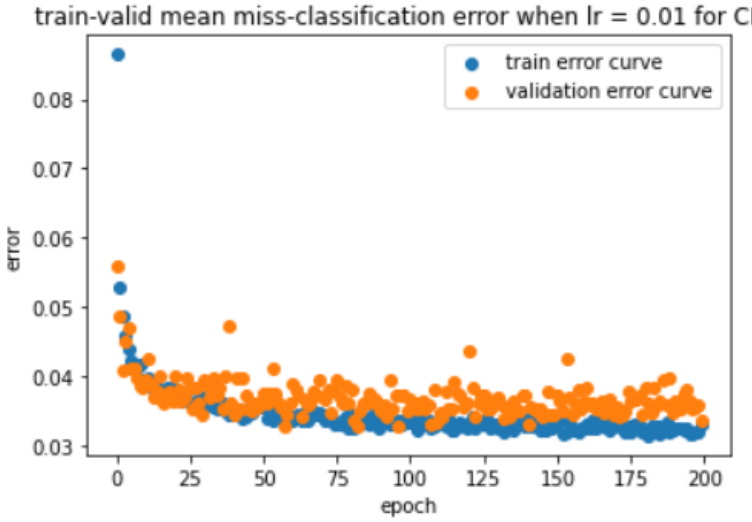
(E). The best of my favorite models cannot outperform the SVM model.

```
Epoch:  0 | Loss: 0.7409 | Train Accuracy: 0.81
Epoch:  20 | Loss: 0.1396 | Train Accuracy: 0.96
Epoch:  40 | Loss: 0.0966 | Train Accuracy: 0.97
Epoch:  60 | Loss: 0.0782 | Train Accuracy: 0.98
Epoch:  80 | Loss: 0.0686 | Train Accuracy: 0.98
Epoch:  100 | Loss: 0.0623 | Train Accuracy: 0.98
Epoch:  120 | Loss: 0.0578 | Train Accuracy: 0.98
Epoch:  140 | Loss: 0.0540 | Train Accuracy: 0.99
Epoch:  160 | Loss: 0.0510 | Train Accuracy: 0.99
Epoch:  180 | Loss: 0.0485 | Train Accuracy: 0.99
Epoch:  200 | Loss: 0.0463 | Train Accuracy: 0.99
Epoch:  220 | Loss: 0.0442 | Train Accuracy: 0.99
Epoch:  240 | Loss: 0.0427 | Train Accuracy: 0.99
Epoch:  260 | Loss: 0.0412 | Train Accuracy: 0.99
Epoch:  280 | Loss: 0.0396 | Train Accuracy: 0.99
Epoch:  300 | Loss: 0.0386 | Train Accuracy: 0.99
Epoch:  320 | Loss: 0.0375 | Train Accuracy: 0.99
Epoch:  340 | Loss: 0.0363 | Train Accuracy: 0.99
Epoch:  360 | Loss: 0.0353 | Train Accuracy: 0.99
Epoch:  380 | Loss: 0.0344 | Train Accuracy: 0.99
final test cross_entropy error loss for CNN is  0.07920393
final test mis-classification error for CNN is  0.023299999999999987
```

```
Text(0.5, 1.0, 'train-valid corss-entropy error loss when lr = 0.01 for CNN')
```



```
Text(0.5, 1.0, 'train-valid mean miss-classification error when lr = 0.01 for CNN')
```

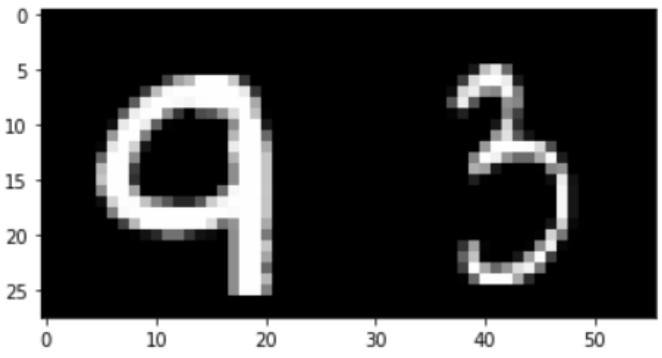


6.

The pixels are scaled in row major because python is row major.

The relationship between the two digits and the last coordinate is that the two digits add up to the last value.

```
image =train.loc[4][:1568].to_numpy().reshape(28,56)# plot the sample
fig = plt.figure
plt.imshow(image, cmap='gray')
plt.show()
```



+ Code + Text

```
[66] train.loc[4][1568]

     12.0
```

7. I use my CNN model in part 5 to predict the labels corresponding to the the images on the graphs. And then do a linear regression to predict the target based on the two predicted labels and true label.

```
train_a = train.iloc[:20000,:28]
for i in range(27):
  train_a = train_a.join(train.iloc[:20000,28*(i*2+2):28*(i*2+3)])
```

```
train_b = train.iloc[:20000,28:56]
for i in range(27):
  train_b = train_b.join(train.iloc[:20000,28*(i*2+3):28*(i*2+4)])
```

```
test_a = test.iloc[:5000,:28]
for i in range(27):
  test_a = test_a.join(test.iloc[:5000,28*(i*2+2):28*(i*2+3)])
```

```
test_b = test.iloc[:5000,28:56]
for i in range(27):
  test_b = test_b.join(test.iloc[:5000,28*(i*2+3):28*(i*2+4)])
```

```
x = df.iloc[:20000,:2]
y = df['Y']
reg = LinearRegression().fit(x, y)
reg.score(x, y)
```

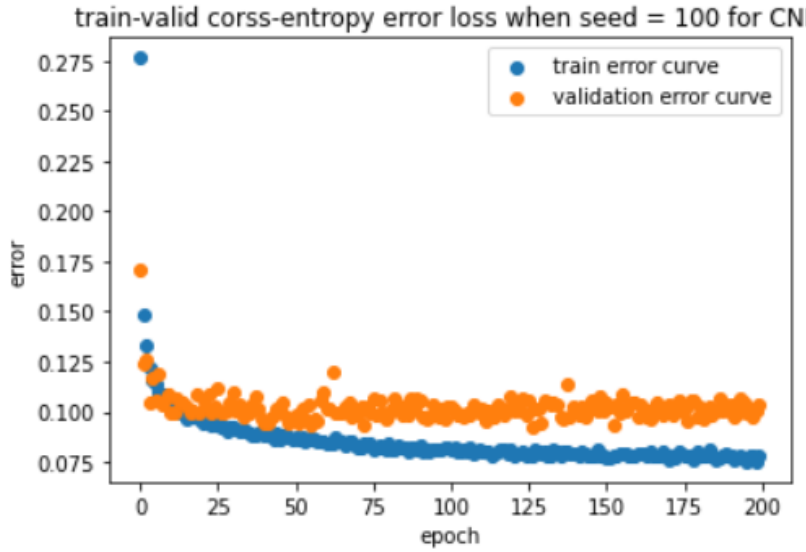
```
0.9350690965120676
```

```
x = df_val.iloc[:5000,:2]
y = df_val['Y']
reg.score(x,y)
```

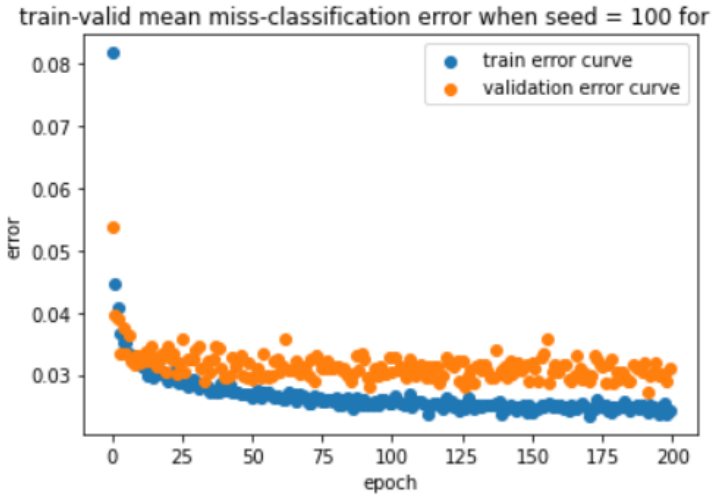
```
0.9437564129695519
```

(A). The general pattern still goes similarly as in question 5, testing loss tends to remain flat but sometimes gets scattered. And this conclusion also apply to misclassification rate.

```
Text(0.5, 1.0, 'train-valid corss-entropy error loss when seed = 100 for CNN')
```




```
Text(0.5, 1.0, 'train-valid mean miss-classification error when seed = 100 for CNN')
```



(B). The validation results are as follows.

When seed = 1, validation accuracy = 0.9437

When seed = 100, validation accuracy = 0.9495

When seed = 1000, validation accuracy = 0.9499

When seed = 10000, validation accuracy = 0.9461

When seed = 10086, validation accuracy = 0.9346

Apparently, when seed = 100, I can get the best model.

(C). The final test accuracy is 0.9303.

The reason I cannot get a test error smaller than 1% is because I am training on a set of targets that is predicted. So it will not give good performance from the beginning.