- First 5 pages are Project milestone 1 contents
- Begin at 6th page, it is other parts of the project
- After Page 14, it is the Appendix: attachment of codes

GR5241_Project1_Lili_Tan

April 5, 2022

[1]:
```
import torchvision
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import sklearn.preprocessing
import pandas as pd

sns.set_style('darkgrid')
%matplotlib inline

# Data download and preprocessing
DOWNLOAD_MNIST = True # If already download, set as False
train_data = torchvision.datasets.MNIST(
    root = './mnist/',
    train = True, # this is training data
    #transform=torchvision.transforms.ToTensor(),
    download = DOWNLOAD_MNIST ,
)
test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)

# change the features to numpy
X_train = train_data.train_data.numpy()
X_test = test_data.test_data.numpy()

# change the labels to numpy
Y_train = train_data.train_labels.numpy()
Y_test = test_data.test_labels.numpy()
```

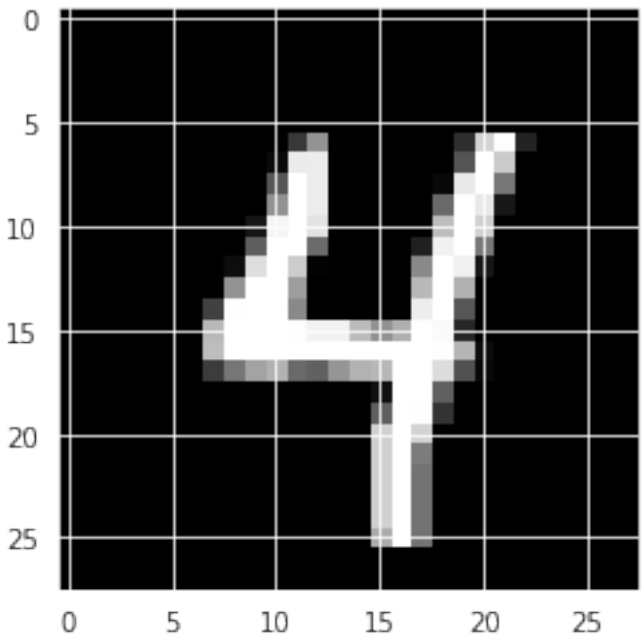
```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
./mnist/MNIST/raw/train-images-idx3-ubyte.gz
```

```
  0%|          | 0/9912422 [00:00<?, ?it/s]
```

```
Extracting ./mnist/MNIST/raw/train-images-idx3-ubyte.gz to ./mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
./mnist/MNIST/raw/train-labels-idx1-ubyte.gz
```

```
  0%|          | 0/28881 [00:00<?, ?it/s]

Extracting ./mnist/MNIST/raw/train-labels-idx1-ubyte.gz to ./mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
./mnist/MNIST/raw/t10k-images-idx3-ubyte.gz

  0%|          | 0/1648877 [00:00<?, ?it/s]

Extracting ./mnist/MNIST/raw/t10k-images-idx3-ubyte.gz to ./mnist/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
./mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz

  0%|          | 0/4542 [00:00<?, ?it/s]

Extracting ./mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./mnist/MNIST/raw


/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:62:
UserWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:67:
UserWarning: test_data has been renamed data
  warnings.warn("test_data has been renamed data")
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:52:
UserWarning: train_labels has been renamed targets
  warnings.warn("train_labels has been renamed targets")
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:57:
UserWarning: test_labels has been renamed targets
  warnings.warn("test_labels has been renamed targets")
```

[2]:
```
#Q1.(a)
import random
i = random.randint(0, X_train.shape[0])
# change i to a fix number or fix a seed to get fix result everytime
plt.imshow(X_train[i], cmap = plt.get_cmap('gray'))
plt.show()
print(f'Y_train is {Y_train[i]}')
print('It matches the label in Y_train')
```




```
Y_train is 4
It matches the label in Y_train
```

[3]:
```
#Q1.(b)
#dimension of X_train
print(f'X_train dimension: {X_train.shape}')

#dimension of X test
print(f'X_test dimension: {X_test.shape}')

#normalize X_train
norm = np.linalg.norm(X_train)
X_train_norm = X_train/norm

#normalize X_test
norm = np.linalg.norm(X_test)
X_test_norm = X_test/norm
```

```
X_train dimension: (60000, 28, 28)
X_test dimension: (10000, 28, 28)
```

[4]:
```
#Q1.(c)
#one-hot embedding Y train
Y_train_onehot = np.array(pd.get_dummies(Y_train))
#one-hot embedding Y test
Y_test_onehot = np.array(pd.get_dummies(Y_test))
```

Q1(c) benefit of such transformation: By one-hot embedding method, we vectorize the value in Y_test and Y_train. It is simple and fast to create and update the vectorization, just add a new entry in the vector with a one for each new category which means it can be rescaled easily. Moreover, the one-hot implementation is known for being the fastest one, allowing a state machine to run at a faster clock rate than any other encoding of that state machine.

[5]:
```
#Q2.(a)
#reshape x_test and x_train
nsamples, nx, ny = X_train.shape
X_train_reshape = X_train.reshape((nsamples,nx*ny))
nsamples, nx, ny = X_test.shape
X_test_reshape = X_test.reshape((nsamples,nx*ny))
```

[6]:
```
#KNN
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=27).fit(X_train_reshape,Y_train_onehot)
print(f'knn test set error: {(1-knn.score(X_test_reshape,Y_test_onehot))*100:0.
 ↪2f}%')
```

```
knn test set error: 4.80%
```

[7]:
```
#AdaBoost
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(DecisionTreeClassifier(max_depth=30), random_state=0).
 ↪fit(X_train_reshape,Y_train)
print(f'AdaBoost test set error: {(1-ada.score(X_test_reshape,Y_test))*100:0.
 ↪2f}%')
```

```
AdaBoost test set error: 3.19%
```

[8]:
```
#SVM
from sklearn import svm
clf = svm.SVC(C=4, kernel='rbf',random_state=123)
clf.fit(X_train_reshape, Y_train)

print(f'SVM with Gaussian Kernel test set error: {(1-clf.score(X_test_reshape,␣
 ↪Y_test))*100:0.2f}%')
```

```
SVM with Gaussian Kernel test set error: 1.57%
```

[9]:
```
#Q2.(b)
#Pick favorite classifier
#reshape data to fit model
X_train = X_train.reshape(60000,28,28,1)
X_test = X_test.reshape(10000,28,28,1)
```

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
#create model
model = Sequential()
#add model layers
model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
#train the model
model.fit(X_train, Y_train_onehot, validation_data=(X_test, Y_test_onehot),␣
 ↪epochs=3)
```

```
Epoch 1/3
1875/1875 [==============================] - 205s 109ms/step - loss: 0.2237 -
accuracy: 0.9492 - val_loss: 0.0877 - val_accuracy: 0.9720
Epoch 2/3
1875/1875 [==============================] - 203s 108ms/step - loss: 0.0701 -
accuracy: 0.9784 - val_loss: 0.0810 - val_accuracy: 0.9774
Epoch 3/3
1875/1875 [==============================] - 204s 109ms/step - loss: 0.0461 -
accuracy: 0.9858 - val_loss: 0.1033 - val_accuracy: 0.9729
```

[9]: <keras.callbacks.History at 0x7f30a1f9ae50>

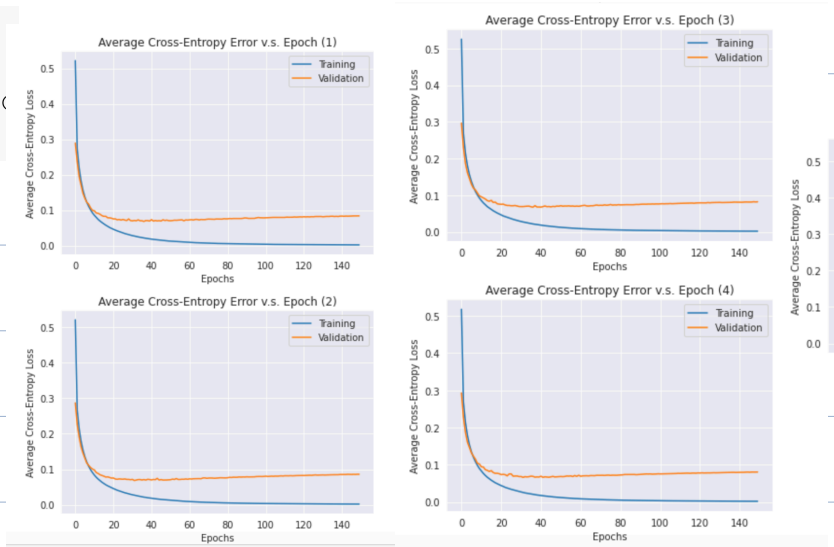
Accuracy for CNN method is 98.58%

GR5241 Project 2 Lili Tan lt2846 Since the code is too long, I will attache the code at the end of the document.

4. Deep Learning

Q3(a) model:

```
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(784, 100),
            nn.ReLU(),
            nn.Linear(100, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

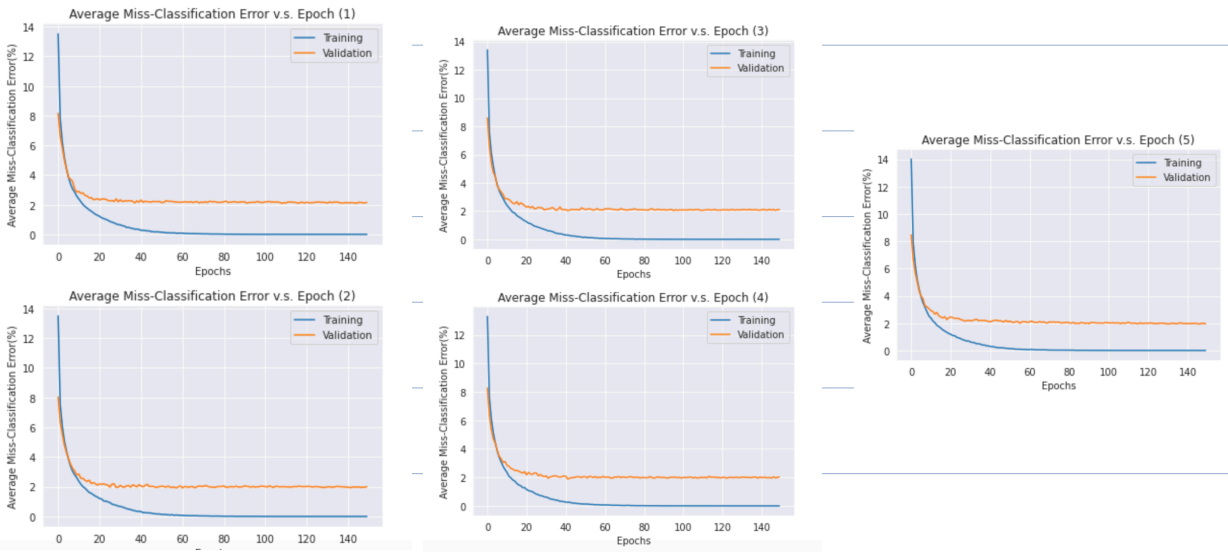


From this model, with epoch = 150, lr = 0.1, mmt = 0.

our training error infinitely close to 0. Testing Error is approximately 0.075.

From the graphs we can see that the error of training set is continuous decreasing, but for test error, it always decreas to a point and then increasing. For example, in above graphs, test error decreases until about 30th epoch. After about 30 epoch, the test error begin to increase.

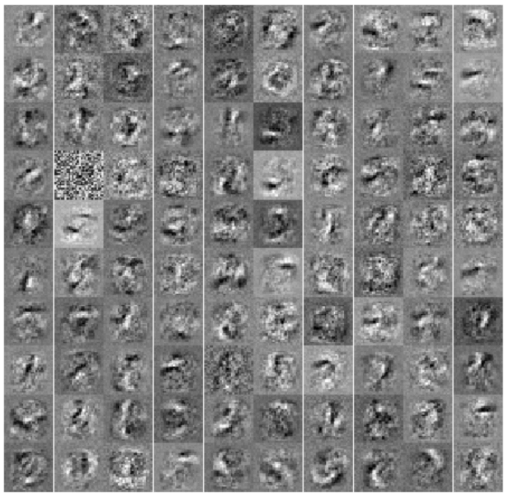
Q3(b)



our training error infinitely close to 0%. Testing Error is approximately 2%

For mean miss-classification error, the test error trend to converge to some value. Unlike, the cross-entropy: the test error decreas to a point and then increasing.

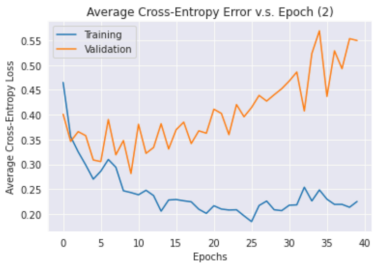
Q3(c)

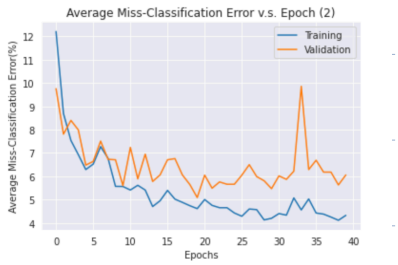


From the graph, it seems like the learned feature exhibit a edge structure, it is clear and not seems like too correlated with each other.

Q3(d) In this section, I use epoch=40 to do the experiment ! From (a) part, test error change little after epoch=40.

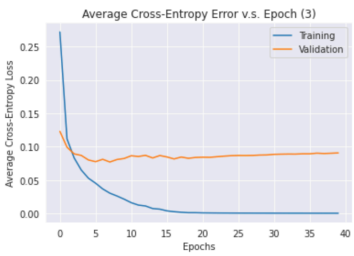
(For learning rate = 0.5 momentum = 0.9)





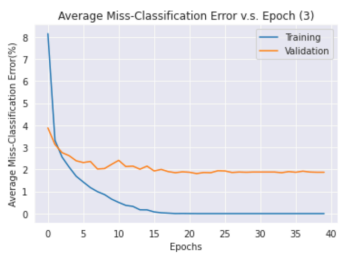
After experiments, when learning rate = 0.5 and momentum = 0.9, the model is unstable and hard to converge. Like the graphs show above. But when we have learning rate = 0.01, it take very long time to learn, too time consuming. Hence, we prefer to use learning rate = 0.1. The hyperparameter momentum can accelerate training, then the model convergence faster when momentum bigger.

(@lr = 0.1 mmt = 0.9)



train error: ≈ 0%

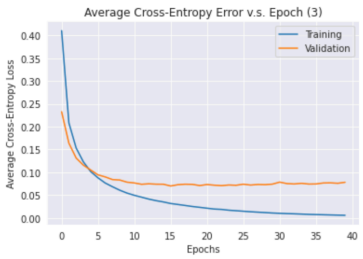
test error: 0.076



train error: ≈ 0%

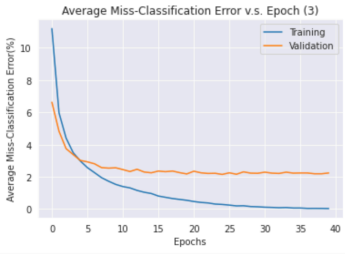
test error: 1.9%

(@lr = 0.1 mmt = 0.5)



train error: ≈ 0

test error: 0.064



train error: ≈ 0%

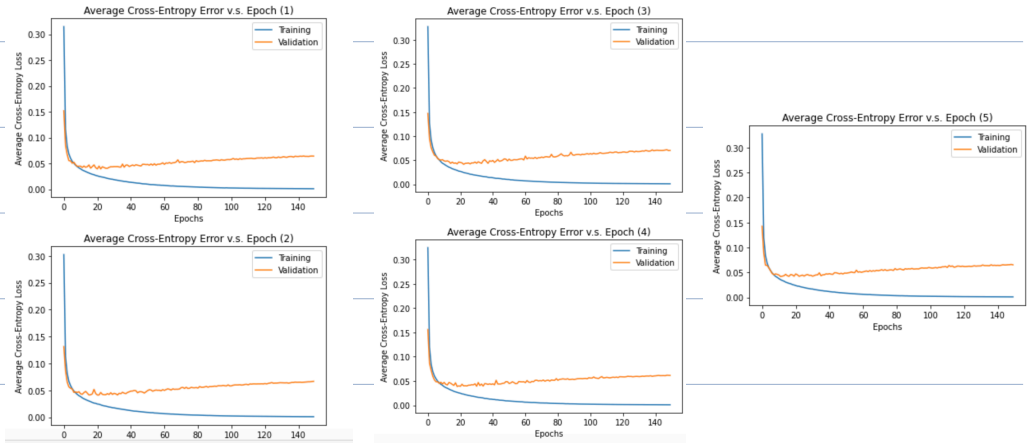
test error: 2.2%

We can see when mmt = 0.9 convergence faster than mmt = 0.5, and when mmt = 0.9 we contain smaller Cross-Entropy Error and Average Miss-classification Error on training set.

Hence, the best value I choose is lr = 0.1, mmt = 0.9

Q4(a) model:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=5,
                stride=1,
                padding=2
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        # fully connected layer, output 10 classes
        self.out = nn.Linear(3136, 10)
    def forward(self, x):
        x = self.conv1(x)
        # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output, x    # return x for visualization
```

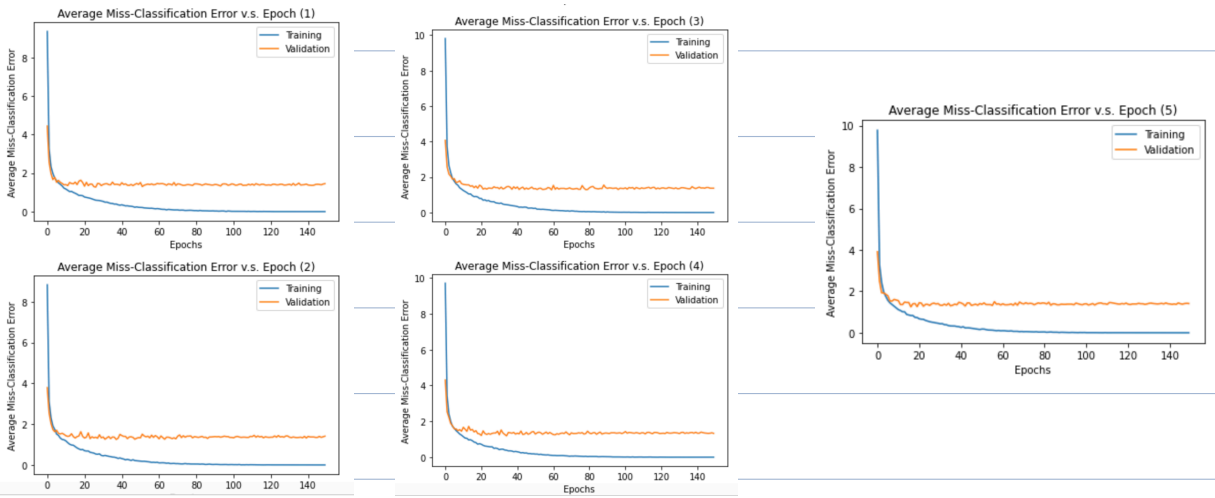


In this CNN model, with epoch = 150, lr = 0.1, mmt = 0.

our training error infinitely close to 0. Testing Error is approximately 0.038. It is much smaller than the first model we used.

From the graphs we can see that the error of training set is continuous decreasing, but for test error, it always decreas to a point and then increasing. For example, in above graphs, test error decreases until about 40th epoch. After about 40 epoch, the test error begin to increase.

Q4(b)



our training error infinitely close to 0%. Testing Error is approximately 1.3%. It is smaller than the first model we used.

For mean miss-classification error, the test error trend to converge to some value. Unlike, the cross-entropy: the test error decreas to a point and then increasing.

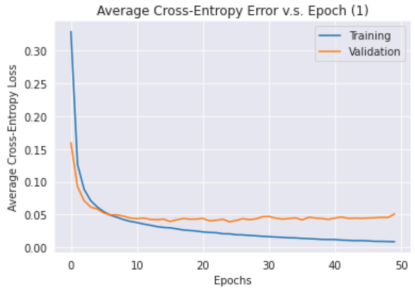
Q4(c)  Because of my setting of CNN, the "cnn.parameters" I get is in shape (16, 1, 5, 5). Hence, I only have

16 picture here. we can see the learned features exhibit a clear edge structure, and not seems like too correlated with each other.

Q4(d) In this section, I use epoch = 50 to do the experiment ! From (a) part, test error change little after epoch = 40. Here, we choose lr=0.01 Since when we have relative small learning rate, we get higher accuracy but more time consume. mmt we choose 0.9. The hyperparameter momentum can accelerate training, then the model convergence faster when momentum bigger.

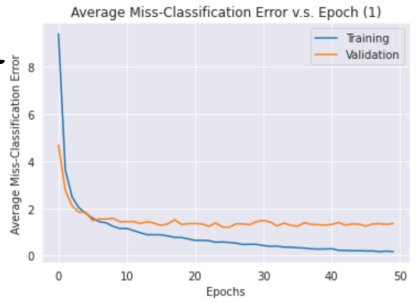
lr=0.01

mmt=0.9



train error: 0.02

test error: 0.041



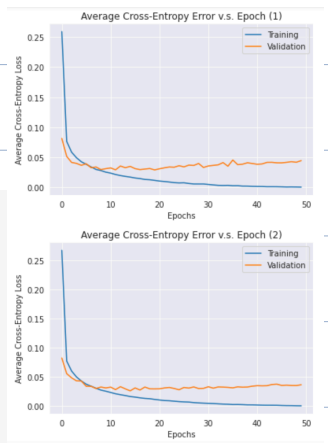
train error: 0.14%

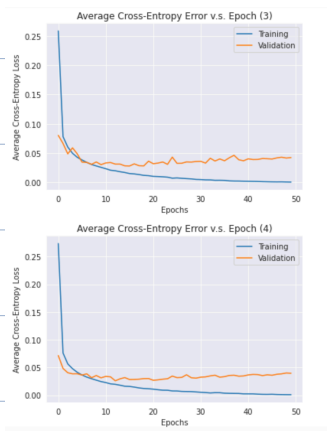
test error: 1.27%

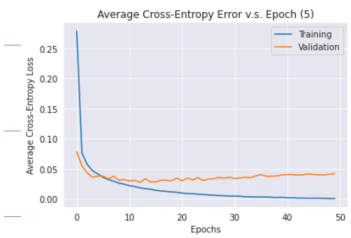
Q4. The best result we get from single layer NN is around error rate 2% but for CNN we improved, the error rate reduce to 1.3%. We can see when we more complex model, we can have a higher accuracy. And CNN is a great method to deal with picture classification task.

Q5(a) model: 2 layer CNN

```
class FavNN1(nn.Module):
    def __init__(self):
        super(FavNN1, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=5,
                stride=1,
                padding=2
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )

        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        # fully connected layer, output 10 classes
        self.out = nn.Linear(32 * 7 * 7, 10)
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output, x    # return x for visualization
```



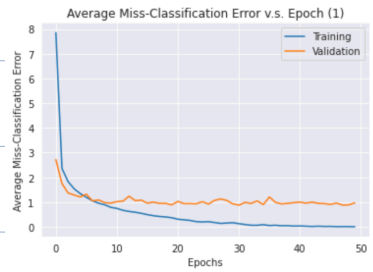


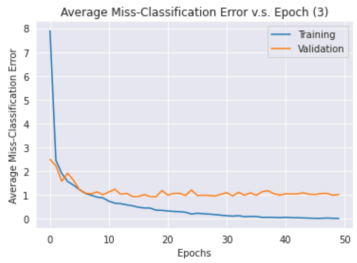


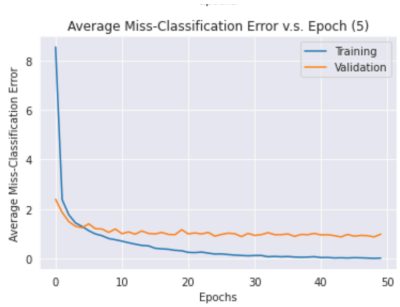
In this 2 layer CNN model, with epoch = 50, lr = 0.1, mmt = 0. our training error infinitely close to 0. Testing Error is approximately 0.0247. From the graphs we can see that the error of training set is continuous

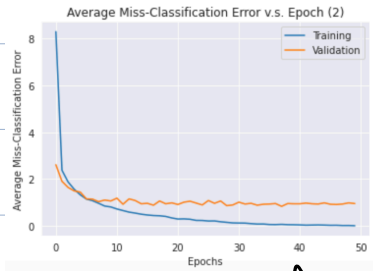
decreasing, but for test error, it always decreas to a point and then increasing. For example, in above graphs, test error decreases until about 20th epoch. After about 20epoch, the test error begin to increase.

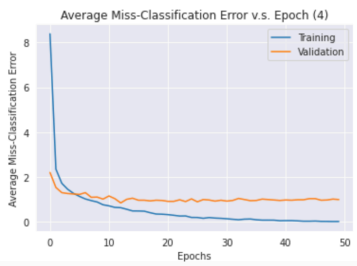
Q5(b)









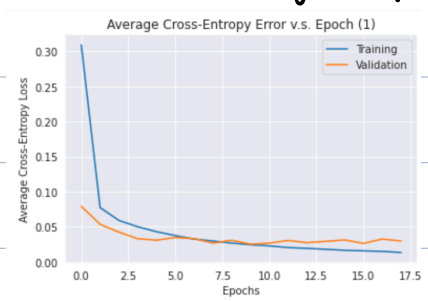


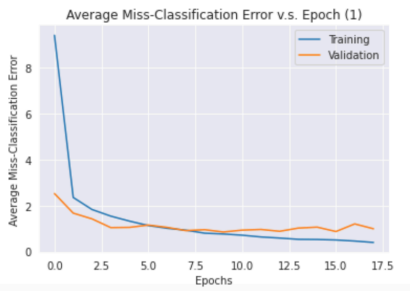
our training error infinitely close to 0%. Testing Error is approximately 0.9%. It is the smallest error rate we get. For mean miss-classification error, the test error trend to converge to some value. Unlike, the cross-entropy: the test error decreas to a point and then increasing.

Q5(c)  Because of my setting of the model, the "cnn.parameters" I get is in shape (16, 1, 5, 5). Hence, I only have 16 picture here. we can see the learned features exhibit a clear edge structure, and not seems like too correlated with each other.

Q5(d) choose lr=0.01 mtm=0.9 In this section, I use epoch=20 to do the experiment

! From 1(a) part, test error change little after epoch=20. Here, we choose lr=0.01 Since when we have relative small learning rate, we get higher accuracy but more time consume. mmt we choose 0.9. The hyperparameter momentum can accelerate training, then the model convergence faster when momentum bigger.





```
Epoch 1
---------------------------------
Train Error:
 Accuracy: 90.6%, Avg loss: 0.308977

Test Error:
 Accuracy: 97.5%, Avg loss: 0.079039

Epoch 2
---------------------------------
Train Error:
 Accuracy: 97.7%, Avg loss: 0.077330

Test Error:
 Accuracy: 98.3%, Avg loss: 0.053519

Epoch 3
---------------------------------
Train Error:
 Accuracy: 98.2%, Avg loss: 0.059044

Test Error:
 Accuracy: 98.6%, Avg loss: 0.042440
```

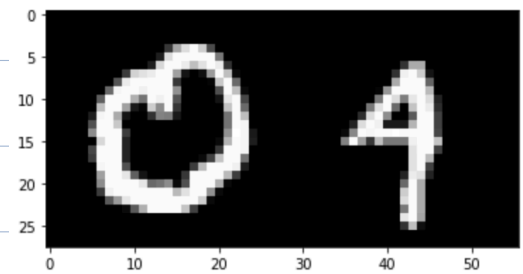
```
Epoch 4
---------------------------------
Train Error:
 Accuracy: 98.5%, Avg loss: 0.050043

Test Error:
 Accuracy: 99.0%, Avg loss: 0.032938

Epoch 5
---------------------------------
Train Error:
 Accuracy: 98.7%, Avg loss: 0.042932

Test Error:
 Accuracy: 99.0%, Avg loss: 0.030932

Epoch 6
---------------------------------
Train Error:
 Accuracy: 98.9%, Avg loss: 0.037458

Test Error:
 Accuracy: 98.9%, Avg loss: 0.034805

Epoch 7
---------------------------------
Train Error:
 Accuracy: 99.0%, Avg loss: 0.032416

Test Error:
 Accuracy: 99.0%, Avg loss: 0.033056
```

```
Epoch 8
---------------------------------
Train Error:
 Accuracy: 99.1%, Avg loss: 0.029677

Test Error:
 Accuracy: 99.1%, Avg loss: 0.026759

Epoch 9
---------------------------------
Train Error:
 Accuracy: 99.2%, Avg loss: 0.026821

Test Error:
 Accuracy: 99.1%, Avg loss: 0.030800

Epoch 10
---------------------------------
Train Error:
 Accuracy: 99.2%, Avg loss: 0.024442

Test Error:
 Accuracy: 99.2%, Avg loss: 0.025138

Epoch 11
---------------------------------
Train Error:
 Accuracy: 99.3%, Avg loss: 0.022803

Test Error:
 Accuracy: 99.1%, Avg loss: 0.026746
```

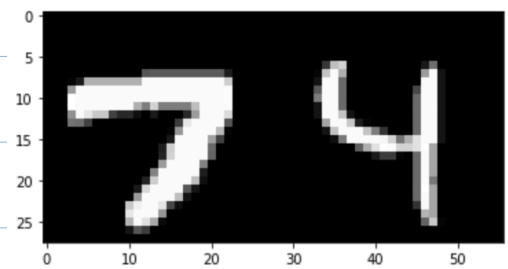
The highest accuracy we get is 99.2% => error rate is 0.8% lower than 1.4%.

For CNN model, we can see when we add more layers, the model can capture more features to increasing the accuracy of testing.

Q6.



last coordinate of line2: 4.0



last coordinate of line13: 11.0

The pixels were scanned out in row major

we plot sample from line 2 & line 13

we can see in line 2, two digits are 0 & 4, 0+4=4= last coordinate in line 2

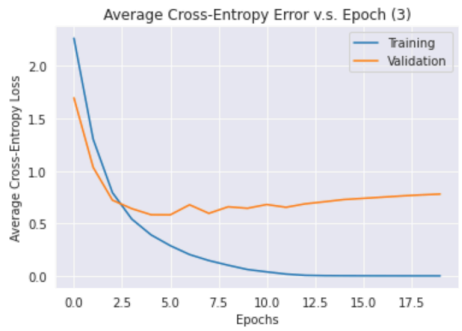
we can see in line 13, two digits are 7 & 4, 4+7=11 = last coordinate in line 13

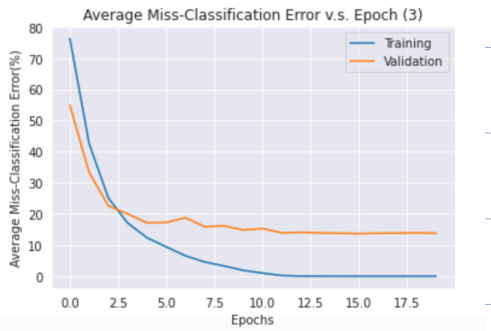
Q7 => next page

Q7 (model 1)

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.1,momentum=0.0)
#create model
model = Sequential()
#add model layers
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,56,1)))
model.add(tf.keras.layers.MaxPooling2D(2,2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(19))
```

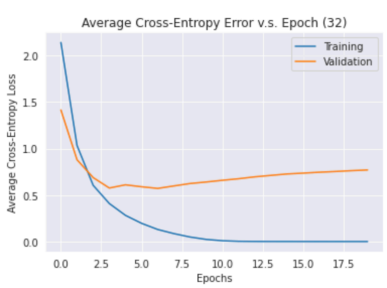
```
Epoch 1/20
625/625 [==============================] - 22s 33ms/step - loss: 2.2641 - accuracy: 0.2365 - val_loss: 1.6962 - val_accuracy: 0.4510
Epoch 2/20
625/625 [==============================] - 19s 31ms/step - loss: 1.3016 - accuracy: 0.5739 - val_loss: 1.0342 - val_accuracy: 0.6664
Epoch 3/20
625/625 [==============================] - 19s 31ms/step - loss: 0.7935 - accuracy: 0.7492 - val_loss: 0.7220 - val_accuracy: 0.7742
Epoch 4/20
625/625 [==============================] - 19s 31ms/step - loss: 0.5424 - accuracy: 0.8292 - val_loss: 0.6413 - val_accuracy: 0.8002
Epoch 5/20
625/625 [==============================] - 19s 31ms/step - loss: 0.3918 - accuracy: 0.8764 - val_loss: 0.5834 - val_accuracy: 0.8284
Epoch 6/20
625/625 [==============================] - 19s 31ms/step - loss: 0.2887 - accuracy: 0.9061 - val_loss: 0.5824 - val_accuracy: 0.8278
Epoch 7/20
625/625 [==============================] - 19s 31ms/step - loss: 0.2043 - accuracy: 0.9345 - val_loss: 0.6780 - val_accuracy: 0.8122
Epoch 8/20
625/625 [==============================] - 19s 31ms/step - loss: 0.1471 - accuracy: 0.9545 - val_loss: 0.5964 - val_accuracy: 0.8412
Epoch 9/20
625/625 [==============================] - 19s 31ms/step - loss: 0.1026 - accuracy: 0.9673 - val_loss: 0.6587 - val_accuracy: 0.8384
Epoch 10/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0619 - accuracy: 0.9814 - val_loss: 0.6447 - val_accuracy: 0.8516
Epoch 11/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0393 - accuracy: 0.9902 - val_loss: 0.6806 - val_accuracy: 0.8472
Epoch 12/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0184 - accuracy: 0.9979 - val_loss: 0.6542 - val_accuracy: 0.8610
Epoch 13/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0066 - accuracy: 0.9998 - val_loss: 0.6871 - val_accuracy: 0.8592
Epoch 14/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0035 - accuracy: 0.9999 - val_loss: 0.7069 - val_accuracy: 0.8612
Epoch 15/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0025 - accuracy: 1.0000 - val_loss: 0.7280 - val_accuracy: 0.8620
Epoch 16/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.7389 - val_accuracy: 0.8634
Epoch 17/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.7507 - val_accuracy: 0.8620
Epoch 18/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.7621 - val_accuracy: 0.8620
Epoch 19/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.7720 - val_accuracy: 0.8606
Epoch 20/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.7798 - val_accuracy: 0.8620
```

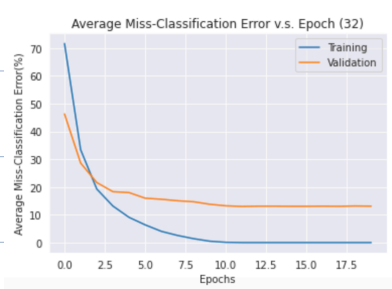




For model 1, the highest accuracy of validation data is 86.34% ⇔ error rate is 13.66%. From the graphs we can see that the error of training set is continuous decreasing, but for test error, it always decreas to a point and then increasing. For example, in above graphs, test error decreases until about 10th epoch. After about 10 epoch, the test error begin to increase. For mean miss-classification error, the test error trend to converge to some value. Unlike, the cross-entropy: the test error decreas to a point and then increasing. we can see the learned features exhibit a clear edge structure, and not seems like too correlated with each other.

best model we found is lr=0.1 mmt=0.3 With these parameters the highest accuracy we get for validation data is 86.98% ⇒ the error rate is 13.02%





```
Epoch 1/20
625/625 [==============================] - 21s 32ms/step - loss: 2.1385 - accuracy: 0.2832 - val_loss: 1.4136 - val_accuracy: 0.5368
Epoch 2/20
625/625 [==============================] - 23s 37ms/step - loss: 1.0319 - accuracy: 0.6657 - val_loss: 0.8791 - val_accuracy: 0.7136
Epoch 3/20
625/625 [==============================] - 22s 35ms/step - loss: 0.6046 - accuracy: 0.8072 - val_loss: 0.6877 - val_accuracy: 0.7838
Epoch 4/20
625/625 [==============================] - 27s 43ms/step - loss: 0.4092 - accuracy: 0.8684 - val_loss: 0.5757 - val_accuracy: 0.8168
Epoch 5/20
625/625 [==============================] - 26s 41ms/step - loss: 0.2822 - accuracy: 0.9092 - val_loss: 0.6102 - val_accuracy: 0.8200
Epoch 6/20
625/625 [==============================] - 26s 42ms/step - loss: 0.1964 - accuracy: 0.9362 - val_loss: 0.5893 - val_accuracy: 0.8402
Epoch 7/20
625/625 [==============================] - 34s 54ms/step - loss: 0.1303 - accuracy: 0.9596 - val_loss: 0.5716 - val_accuracy: 0.8440
Epoch 8/20
625/625 [==============================] - 25s 41ms/step - loss: 0.0860 - accuracy: 0.9743 - val_loss: 0.5982 - val_accuracy: 0.8494
Epoch 9/20
625/625 [==============================] - 25s 40ms/step - loss: 0.0494 - accuracy: 0.9859 - val_loss: 0.6240 - val_accuracy: 0.8530
Epoch 10/20
625/625 [==============================] - 25s 39ms/step - loss: 0.0237 - accuracy: 0.9951 - val_loss: 0.6404 - val_accuracy: 0.8620
Epoch 11/20
625/625 [==============================] - 23s 36ms/step - loss: 0.0096 - accuracy: 0.9991 - val_loss: 0.6587 - val_accuracy: 0.8676
Epoch 12/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 0.6752 - val_accuracy: 0.8698
Epoch 13/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.6957 - val_accuracy: 0.8688
Epoch 14/20
625/625 [==============================] - 24s 38ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.7112 - val_accuracy: 0.8688
Epoch 15/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.7266 - val_accuracy: 0.8692
Epoch 16/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.7356 - val_accuracy: 0.8692
Epoch 17/20
625/625 [==============================] - 19s 31ms/step - loss: 9.3699e-04 - accuracy: 1.0000 - val_loss: 0.7452 - val_accuracy: 0.8688
Epoch 18/20
625/625 [==============================] - 20s 31ms/step - loss: 8.4199e-04 - accuracy: 1.0000 - val_loss: 0.7531 - val_accuracy: 0.8692
Epoch 19/20
625/625 [==============================] - 19s 31ms/step - loss: 7.6935e-04 - accuracy: 1.0000 - val_loss: 0.7621 - val_accuracy: 0.8682
Epoch 20/20
625/625 [==============================] - 20s 32ms/step - loss: 7.0477e-04 - accuracy: 1.0000 - val_loss: 0.7695 - val_accuracy: 0.8688
```

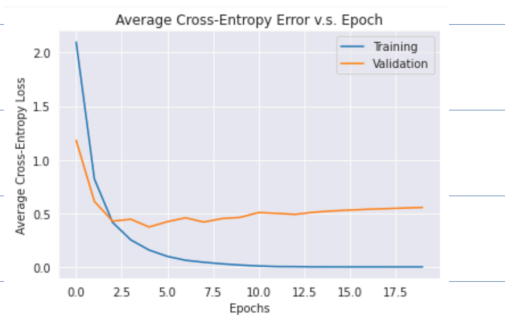
lr=0.1 mmt=0.3

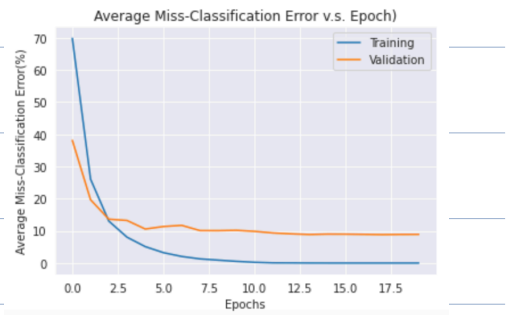
model 1

Q7.(model 2)

```
sgd = tf.keras.optimizers.SGD(learning_rate=0.1,momentum=0.0)
#create model
model2 = Sequential()
#add model layers
model2.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,56,1)))
model2.add(tf.keras.layers.MaxPooling2D(2,2))
model2.add(Conv2D(32, (3,3), activation='relu'))
model2.add(Flatten())
model2.add(Dense(64, activation='relu'))
model2.add(Dense(19))
```

```
Epoch 1/20
625/625 [==============================] - 33s 52ms/step - loss: 2.0977 - accuracy: 0.3017 - val_loss: 1.1810 - val_accuracy: 0.6188
Epoch 2/20
625/625 [==============================] - 32s 51ms/step - loss: 0.8199 - accuracy: 0.7395 - val_loss: 0.6112 - val_accuracy: 0.8034
Epoch 3/20
625/625 [==============================] - 32s 51ms/step - loss: 0.4149 - accuracy: 0.8697 - val_loss: 0.4267 - val_accuracy: 0.8638
Epoch 4/20
625/625 [==============================] - 32s 51ms/step - loss: 0.2519 - accuracy: 0.9197 - val_loss: 0.4448 - val_accuracy: 0.8678
Epoch 5/20
625/625 [==============================] - 32s 51ms/step - loss: 0.1572 - accuracy: 0.9491 - val_loss: 0.3723 - val_accuracy: 0.8944
Epoch 6/20
625/625 [==============================] - 33s 52ms/step - loss: 0.0982 - accuracy: 0.9679 - val_loss: 0.4218 - val_accuracy: 0.8866
Epoch 7/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0619 - accuracy: 0.9796 - val_loss: 0.4580 - val_accuracy: 0.8830
Epoch 8/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0433 - accuracy: 0.9872 - val_loss: 0.4184 - val_accuracy: 0.8992
Epoch 9/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0284 - accuracy: 0.9911 - val_loss: 0.4510 - val_accuracy: 0.8994
Epoch 10/20
625/625 [==============================] - 34s 54ms/step - loss: 0.0171 - accuracy: 0.9948 - val_loss: 0.4622 - val_accuracy: 0.8984
Epoch 11/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0089 - accuracy: 0.9977 - val_loss: 0.5070 - val_accuracy: 0.9016
Epoch 12/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0033 - accuracy: 0.9994 - val_loss: 0.4999 - val_accuracy: 0.9066
Epoch 13/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0025 - accuracy: 0.9996 - val_loss: 0.4897 - val_accuracy: 0.9094
Epoch 14/20
625/625 [==============================] - 32s 51ms/step - loss: 4.7297e-04 - accuracy: 0.9999 - val_loss: 0.5096 - val_accuracy: 0.9114
Epoch 15/20
625/625 [==============================] - 39s 62ms/step - loss: 2.3346e-04 - accuracy: 1.0000 - val_loss: 0.5218 - val_accuracy: 0.9102
Epoch 16/20
625/625 [==============================] - 34s 54ms/step - loss: 1.8274e-04 - accuracy: 1.0000 - val_loss: 0.5303 - val_accuracy: 0.9104
Epoch 17/20
625/625 [==============================] - 32s 52ms/step - loss: 1.5414e-04 - accuracy: 1.0000 - val_loss: 0.5378 - val_accuracy: 0.9110
Epoch 18/20
625/625 [==============================] - 32s 51ms/step - loss: 1.3427e-04 - accuracy: 1.0000 - val_loss: 0.5436 - val_accuracy: 0.9118
Epoch 19/20
625/625 [==============================] - 32s 51ms/step - loss: 1.1933e-04 - accuracy: 1.0000 - val_loss: 0.5494 - val_accuracy: 0.9112
Epoch 20/20
625/625 [==============================] - 33s 52ms/step - loss: 1.0769e-04 - accuracy: 1.0000 - val_loss: 0.5548 - val_accuracy: 0.9110
```

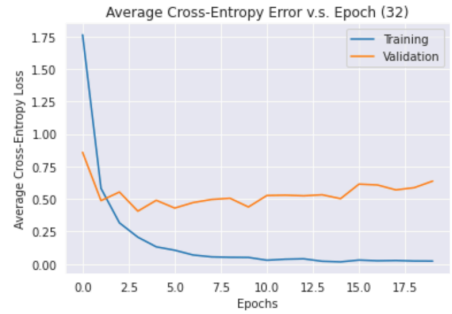




For model 2, the highest accuracy of validation data is 91.18 % ⇐> error rate is 8.82 %. From the graphs we can see that the error of training set is continuous decreasing, but for test error, it always decreas to a point and then increasing. For example, in above graphs, test error decreases until about 10 th epoch. After about 10 epoch, the test error begin to increase. For mean miss-classification error, the test error trend to converge to some value. Unlike, the cross-entropy: the test error decreas to a point and then increasing.

we can see the learned features exhibit a clear edge structure, and not seems like too correlated with each other.

```
Epoch 1/20
625/625 [==============================] - 33s 52ms/step - loss: 1.7634 - accuracy: 0.4103 - val_loss: 0.8587 - val_accuracy: 0.7338
Epoch 2/20
625/625 [==============================] - 33s 52ms/step - loss: 0.5811 - accuracy: 0.8145 - val_loss: 0.4867 - val_accuracy: 0.8478
Epoch 3/20
625/625 [==============================] - 33s 52ms/step - loss: 0.3157 - accuracy: 0.9031 - val_loss: 0.5530 - val_accuracy: 0.8424
Epoch 4/20
625/625 [==============================] - 33s 52ms/step - loss: 0.2055 - accuracy: 0.9347 - val_loss: 0.4058 - val_accuracy: 0.8910
Epoch 5/20
625/625 [==============================] - 32s 52ms/step - loss: 0.1311 - accuracy: 0.9572 - val_loss: 0.4893 - val_accuracy: 0.8746
Epoch 6/20
625/625 [==============================] - 33s 52ms/step - loss: 0.1052 - accuracy: 0.9639 - val_loss: 0.4295 - val_accuracy: 0.8946
Epoch 7/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0683 - accuracy: 0.9785 - val_loss: 0.4718 - val_accuracy: 0.8936
Epoch 8/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0539 - accuracy: 0.9825 - val_loss: 0.4957 - val_accuracy: 0.8952
Epoch 9/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0508 - accuracy: 0.9833 - val_loss: 0.5050 - val_accuracy: 0.8934
Epoch 10/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0497 - accuracy: 0.9833 - val_loss: 0.4378 - val_accuracy: 0.9110
Epoch 11/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0282 - accuracy: 0.9901 - val_loss: 0.5263 - val_accuracy: 0.9104
Epoch 12/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0361 - accuracy: 0.9876 - val_loss: 0.5290 - val_accuracy: 0.8976
Epoch 13/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0391 - accuracy: 0.9870 - val_loss: 0.5239 - val_accuracy: 0.9070
Epoch 14/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0201 - accuracy: 0.9934 - val_loss: 0.5320 - val_accuracy: 0.9068
Epoch 15/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0155 - accuracy: 0.9956 - val_loss: 0.5019 - val_accuracy: 0.9174
Epoch 16/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0294 - accuracy: 0.9909 - val_loss: 0.6138 - val_accuracy: 0.9062
Epoch 17/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0240 - accuracy: 0.9927 - val_loss: 0.6073 - val_accuracy: 0.9042
Epoch 18/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0260 - accuracy: 0.9923 - val_loss: 0.5694 - val_accuracy: 0.9068
Epoch 19/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0221 - accuracy: 0.9931 - val_loss: 0.5865 - val_accuracy: 0.9036
Epoch 20/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0214 - accuracy: 0.9934 - val_loss: 0.6373 - val_accuracy: 0.9022
```



best model we found is lr=0.1 mmt=0.5 With these parameters the highest accuracy we get for validation data is 91.74% => the error rate is 8.26%

Apperantly, the better model we get is model 2 with lr=0.1 mmt=0.5

Hence, we apply this model on test data to get the test error

```
#pick best model and on test set
#best model is model2
test = np.argmax(model2.predict(X_test_reshape),axis=1)
test_error = np.mean(test != Y_test)
test_error
```

```
0.0842
```

The test error we get is 8.42% from model 2 with seed(0).

Q7. Compared with MNIST Data, the test error of two digits data is much bigger than MNIST's. I think the test error cannot lower than 1% because we have two digits here and we have a sum relationship between two digits. It is a bit complex for NN to learn. I think if add more layers to the model maybe we can achive a higher accuracy rate.

Appendix : Codes

GR5241_Project2_Q3

May 3, 2022

[14]:
```
import torchvision
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import sklearn.preprocessing
import pandas as pd

sns.set_style('darkgrid')
%matplotlib inline

# Data download and preprocessing
DOWNLOAD_MNIST = True # If already download, set as False
train_data = torchvision.datasets.MNIST(
    root = './mnist/',
    train = True, # this is training data
    #transform=torchvision.transforms.ToTensor(),
    download = DOWNLOAD_MNIST ,
)
test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)

# change the features to numpy
X_train = train_data.train_data.numpy()
X_test = test_data.test_data.numpy()

# change the labels to numpy
Y_train = train_data.train_labels.numpy()
Y_test = test_data.test_labels.numpy()
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:75:
UserWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:80:
UserWarning: test_data has been renamed data
  warnings.warn("test_data has been renamed data")
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:65:
UserWarning: train_labels has been renamed targets
  warnings.warn("train_labels has been renamed targets")
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:70:
```

```
UserWarning: test_labels has been renamed targets
  warnings.warn("test_labels has been renamed targets")
```

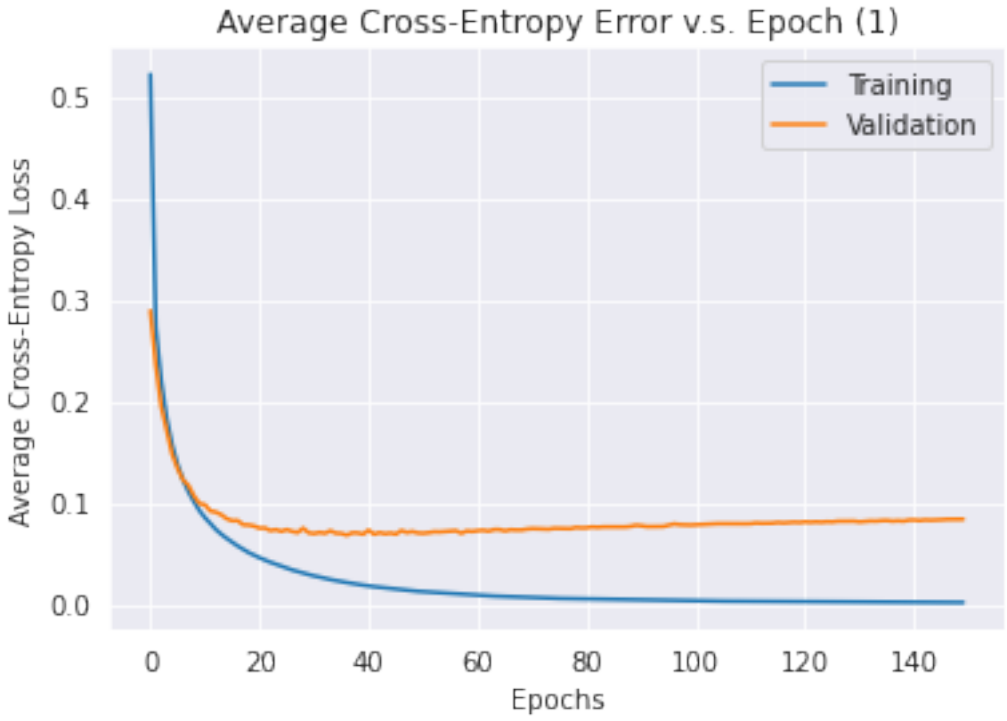
[15]:
```python
# Q3.(a)
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import torch.utils.data as Data
from torchvision.transforms import ToTensor, Lambda
import random

# batchsize of dataset
batch_size = 100

train_data = datasets.MNIST(
    root = 'data',
    train = True,
    transform = ToTensor(),
    download = True,
)

test_data = datasets.MNIST(
    root = 'data',
    train = False,
    transform = ToTensor()
)

loaders = {
    'train' : torch.utils.data.DataLoader(train_data,
                                          batch_size=batch_size,
                                          shuffle=True),

    'test'  : torch.utils.data.DataLoader(test_data,
                                          batch_size=batch_size,
                                          shuffle=True)
}
```

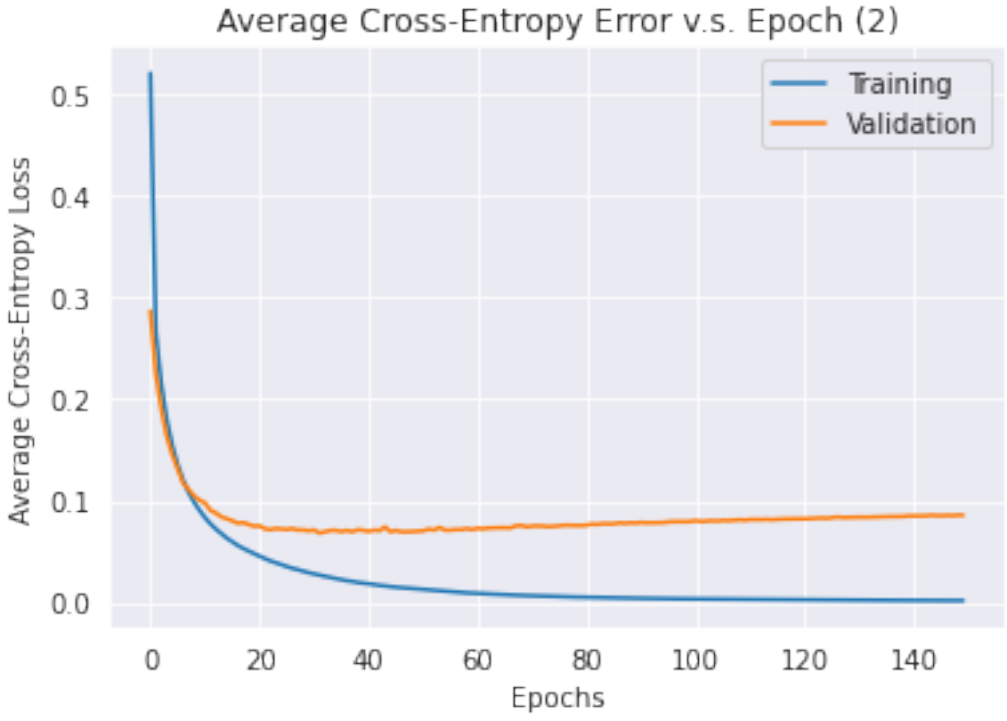
[2]:
```python
# Define a class
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(784, 100),
            nn.ReLU(),
```

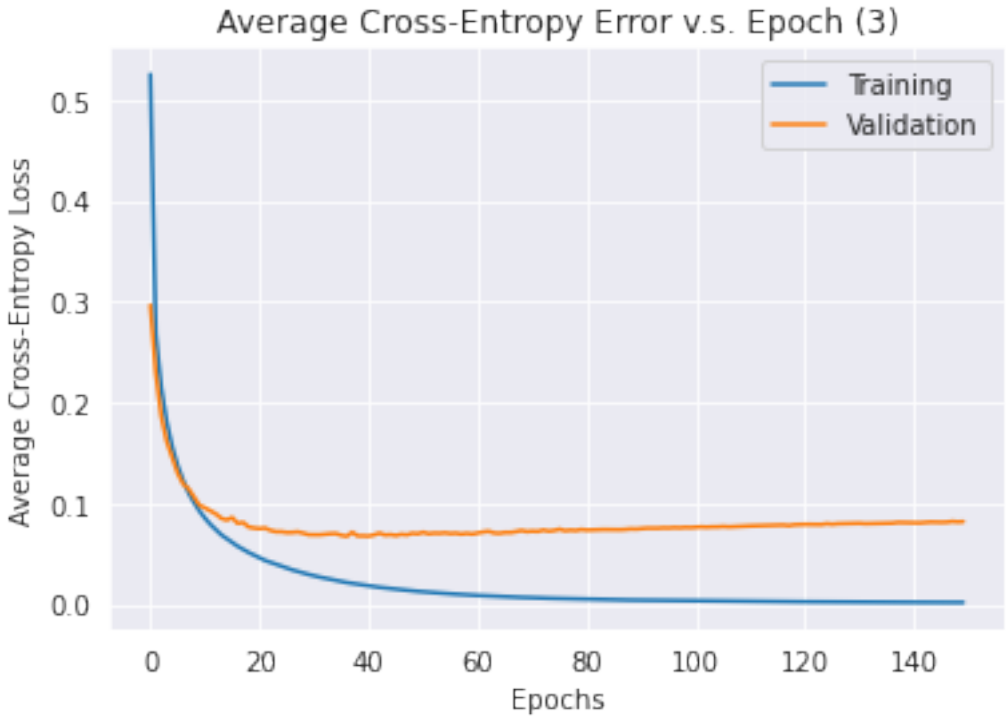
```
            nn.Linear(100, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

#train
num_epoch = 150
learning_rate = 0.1
mtm = 0.0

cee_train = []
cee_test = []
mce_train = []
mce_test = []

# use_cuda = torch.cuda.is_available()

for i in range(5):
    random.seed(i+1)
    avg_cee_training = []
    avg_cee_validation = []
    avg_mce_training = []
    avg_mce_validation = []

    net = SimpleNN()
    optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum = mtm)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(num_epoch):
        #print(f"Epoch {epoch+1}\n-------------------------------")

        train_size = len(loaders["train"].dataset)
        num_batches = len(loaders["train"])
        train_loss, correct = 0, 0
        for batch, (batch_x, batch_y) in enumerate(loaders["train"]):
            out = net(batch_x)
            loss = criterion(out, batch_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if batch % 100 == 0:
                loss_p, current = loss.item(), batch * len(batch_x)
                #print(f"loss: {loss_p:>7f}  [{current:>5d}/{train_size:>5d}]")
```

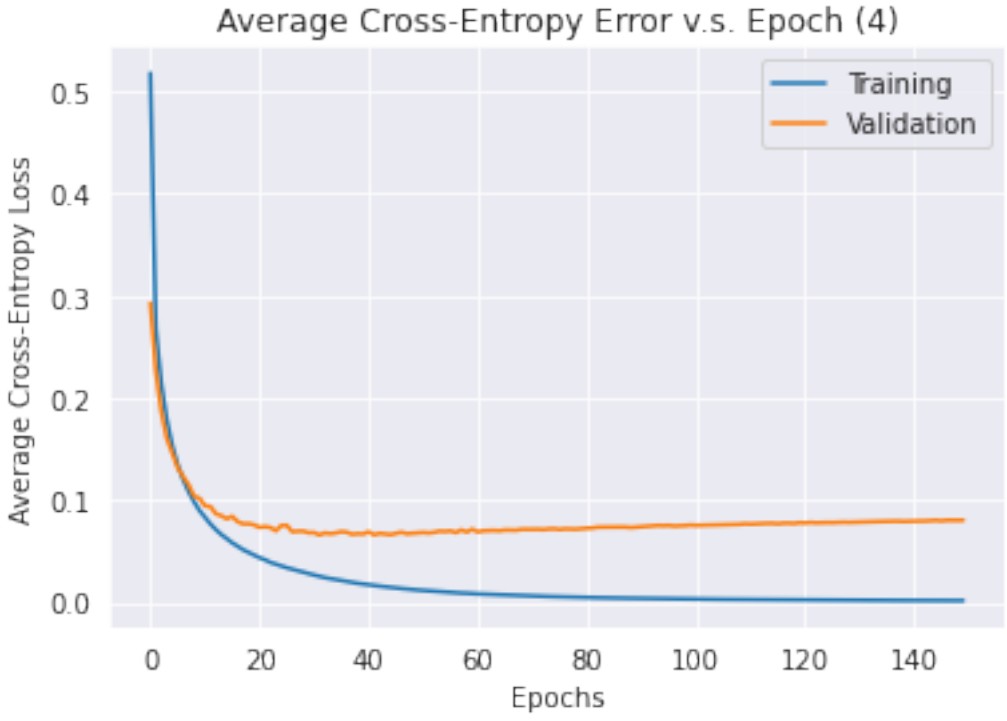
```
            train_loss += loss.item()
            correct += (out.argmax(1) == batch_y).type(torch.float).sum().item()

        train_loss /= num_batches
        correct /= train_size
        #print(f"Train Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{train_loss:>8f} \n")
        avg_cee_training.append(train_loss)
        avg_mce_training.append(100*correct)

        # validation
        test_size = len(loaders["test"].dataset)
        num_batches = len(loaders["test"])
        test_loss, correct = 0, 0

        with torch.no_grad():
            for X, y in loaders["test"]:
                pred = net(X)
                loss = criterion(pred, y)
                test_loss += loss.item()
                correct += (pred.argmax(1) == y).type(torch.float).sum().item()

        test_loss /= num_batches
        correct /= test_size
        #print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{test_loss:>8f} \n")
        avg_cee_validation.append(test_loss)
        avg_mce_validation.append(100*correct)

    cee_train.append(avg_cee_training)
    cee_test.append(avg_cee_validation)
    mce_train.append(avg_mce_training)
    mce_test.append(avg_mce_validation)
```

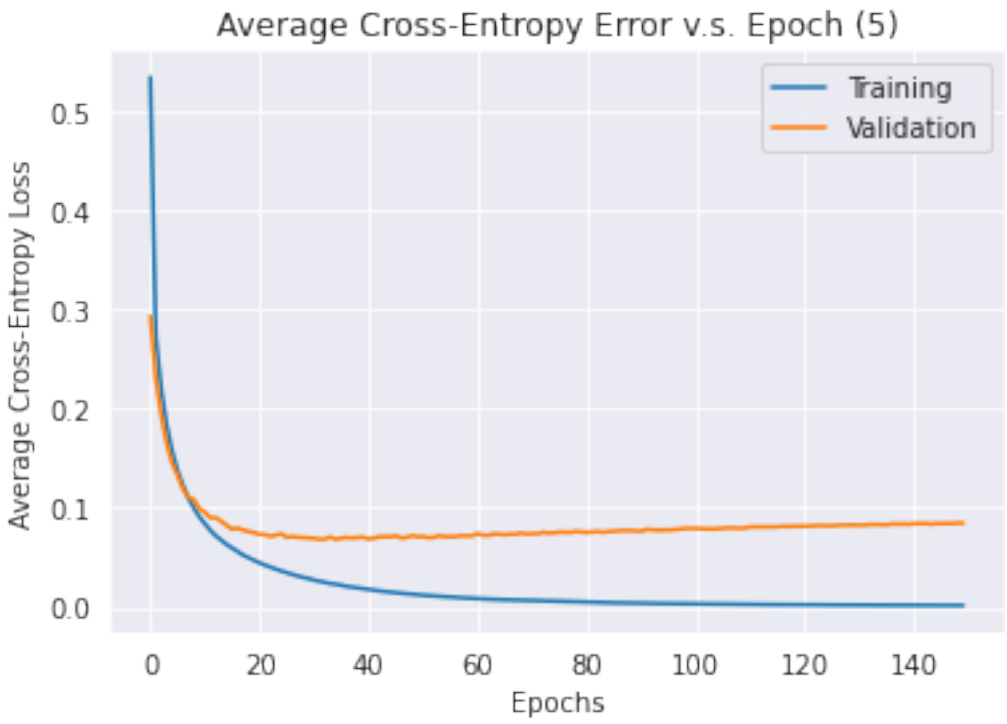
[7]:
```
#plot
for i in range(5):
    plt.title(f"Average Cross-Entropy Error v.s. Epoch ({i+1})")
    plt.plot(np.array(cee_train[i]), label="Training")
    plt.plot(np.array(cee_test[i]), label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel("Average Cross-Entropy Loss")
    plt.legend()
    plt.show()
```


Average Cross-Entropy Error v.s. Epoch (1)
Training
Validation
Average Cross-Entropy Loss
0.0
0.1
0.2
0.3
0.4
0.5
0
20
40
60
80
100
120
140
Epochs

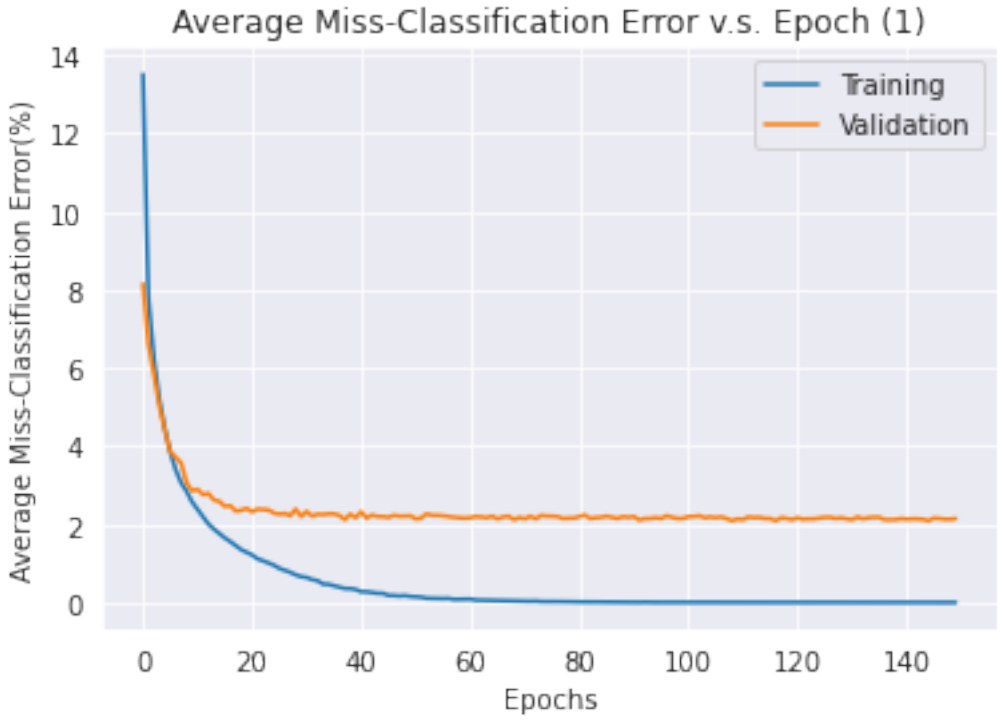

Average Cross-Entropy Error v.s. Epoch (2)
Training
Validation
Average Cross-Entropy Loss
0.0
0.1
0.2
0.3
0.4
0.5
0
20
40
60
80
100
120
140
Epochs

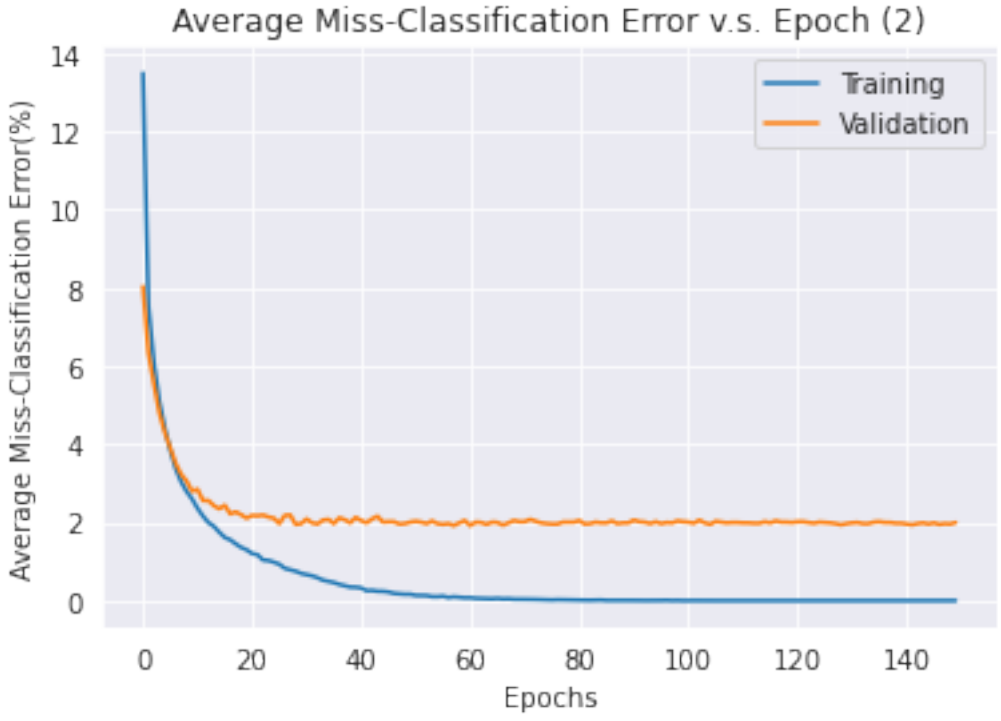

Average Cross-Entropy Error v.s. Epoch (3)
Training
Validation
Average Cross-Entropy Loss
0.0
0.1
0.2
0.3
0.4
0.5
0
20
40
60
80
100
120
140
Epochs

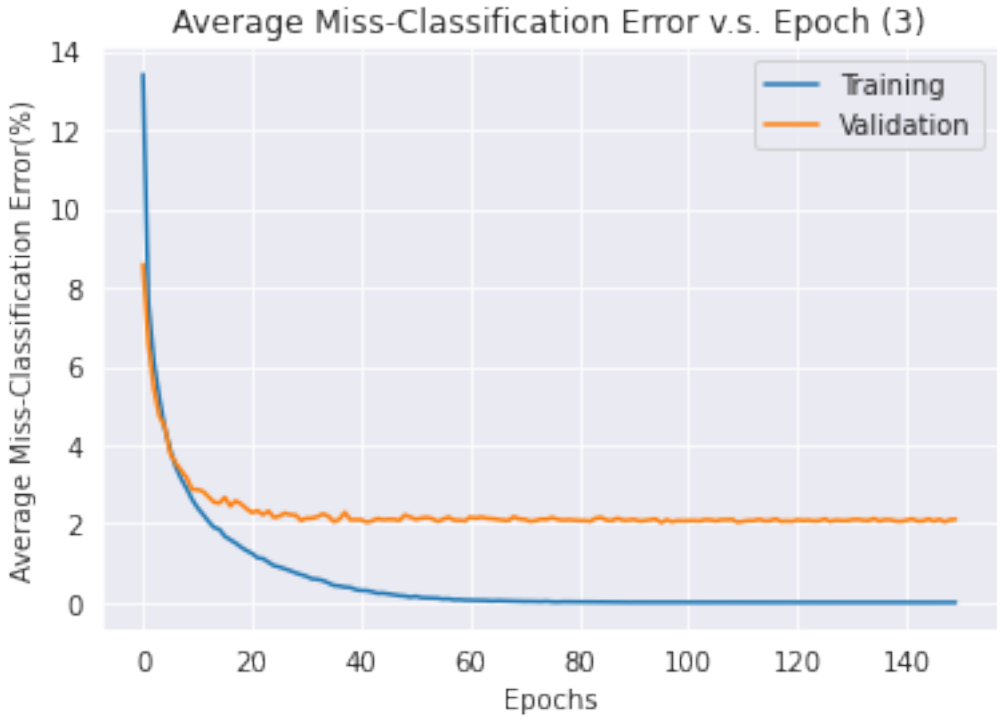

Average Cross-Entropy Error v.s. Epoch (4)
Training
Validation
Average Cross-Entropy Loss
0.0
0.1
0.2
0.3
0.4
0.5
0
20
40
60
80
100
120
140
Epochs

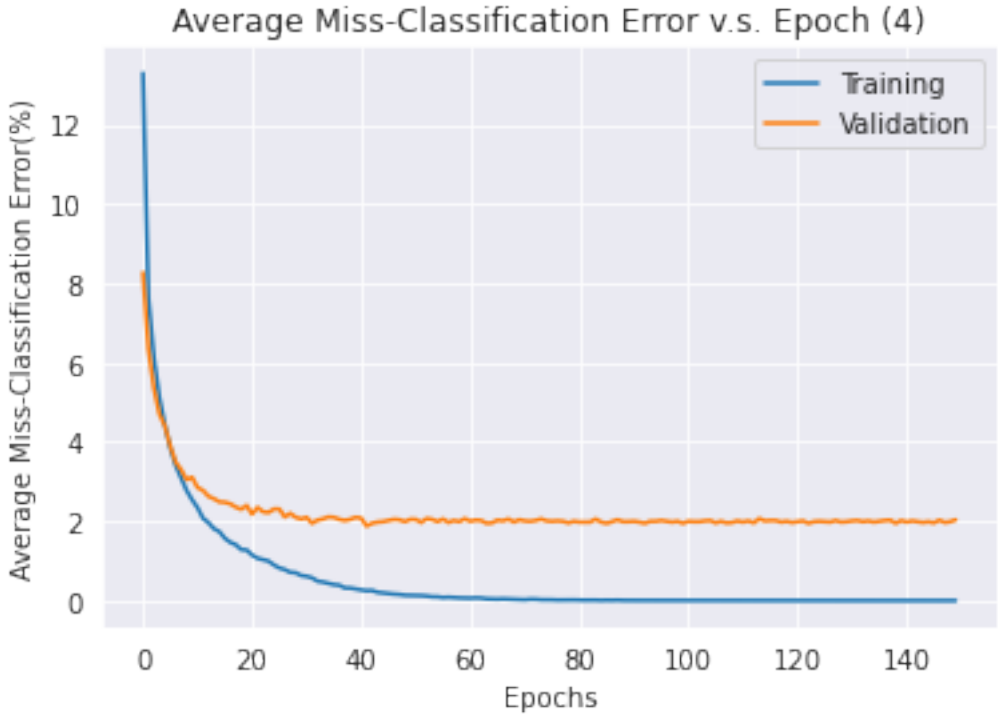


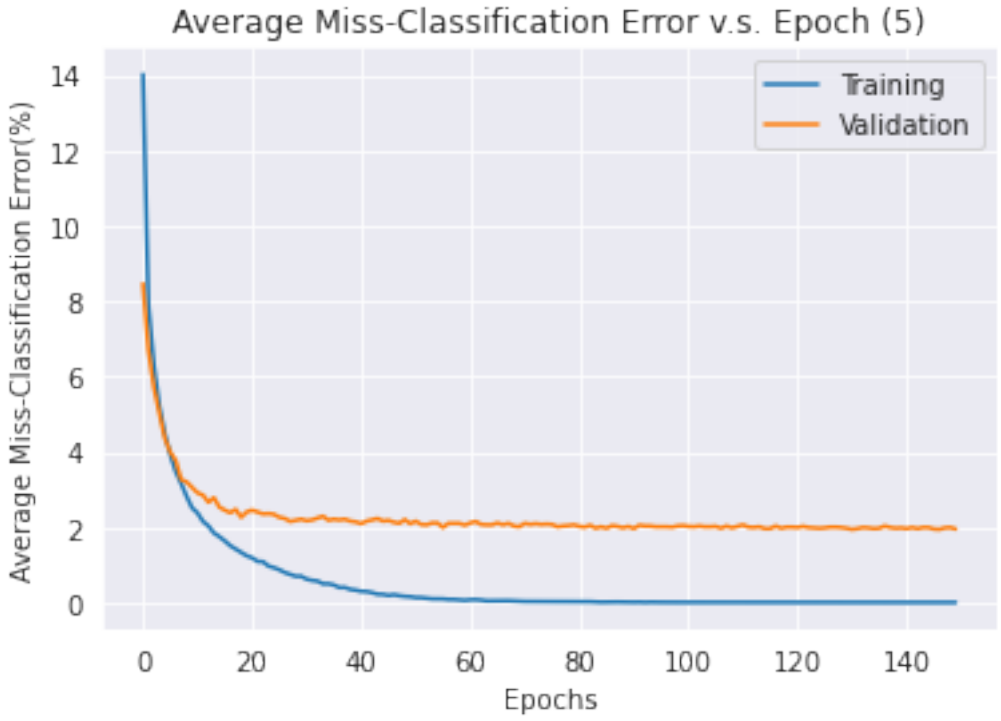
[10]:
```
#Q3.(b)
#plot
for i in range(5):
    plt.title(f"Average Miss-Classification Error v.s. Epoch ({i+1})")
    plt.plot(100-np.array(mce_train[i]), label="Training")
    plt.plot(100-np.array(mce_test[i]), label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel("Average Miss-Classification Error(%)")
    plt.legend()
    plt.show()
```


Average Miss-Classification Error v.s. Epoch (1)
Training
Validation
Average Miss-Classification Error(%)
0
2
4
6
8
10
12
14
0
20
40
60
80
100
120
140
Epochs

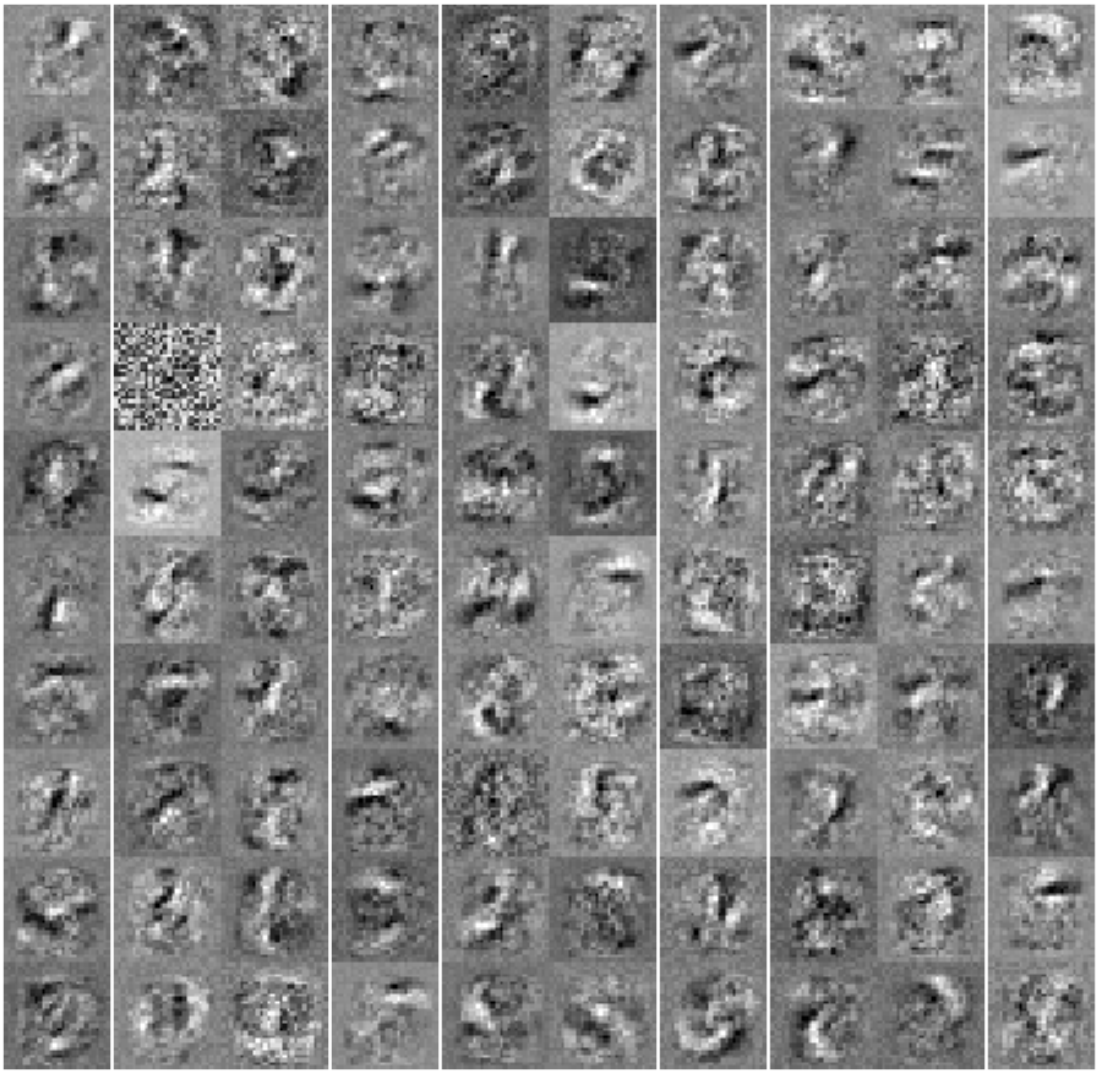

Average Miss-Classification Error v.s. Epoch (2)
Training
Validation
Average Miss-Classification Error(%)
0
2
4
6
8
10
12
14
0
20
40
60
80
100
120
140
Epochs


Average Miss-Classification Error v.s. Epoch (3)
Training
Validation
Average Miss-Classification Error(%)
Epochs
0
2
4
6
8
10
12
14
0
20
40
60
80
100
120
140


Average Miss-Classification Error v.s. Epoch (4)
Training
Validation
Average Miss-Classification Error(%)
Epochs
0
2
4
6
8
10
12
0
20
40
60
80
100
120
140



[6]:
```
#Q3.(c)

## find the best model first
params = list(net.parameters())[0]
plt.figure(figsize=(8, 8))
for i in range(params.shape[0]):
    plt.subplot(10, 10, i + 1) # Since we know it is a 10 x 10 grid
    x = params[i,:].detach().numpy()
    plt.imshow(x.reshape((28, 28)), cmap = "gray", interpolation = "nearest")
    plt.axis("off")
plt.subplots_adjust(wspace=0, hspace=0)
plt.savefig("3_c.png")
```



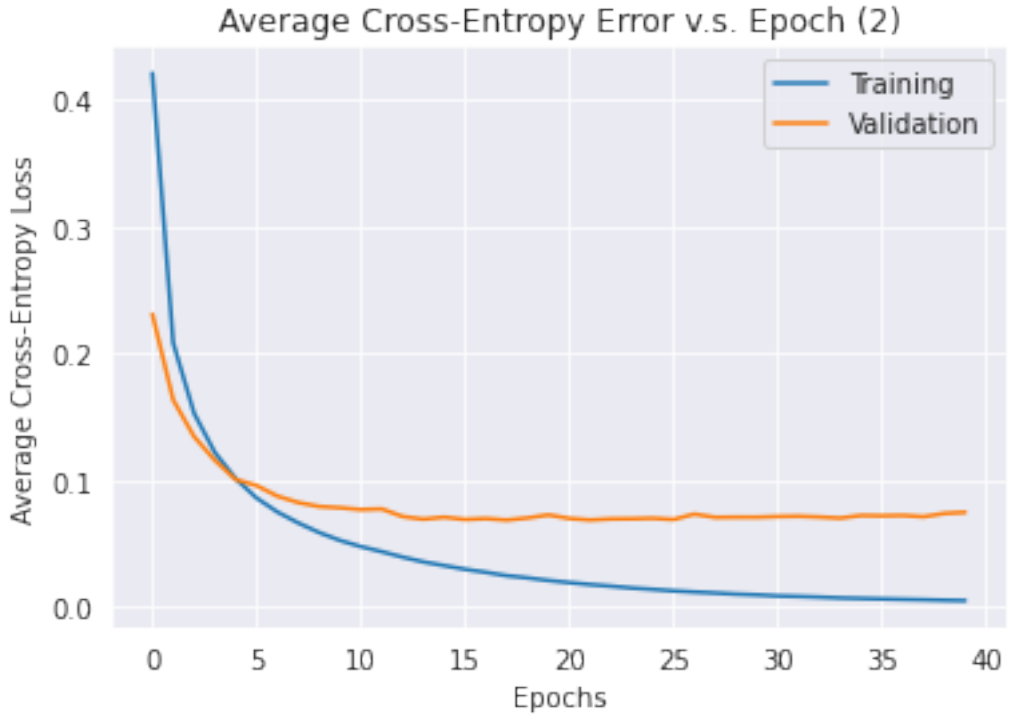
[23]:
```
#Q3.(d)
# change lr 0.1/0.01/0.2/0.5
# change momentum 0.0/0.5/0.9

#train
num_epoch = 40 #base on previous tests, normally convergence after 40 epoch
learning_rate = 0.1
#learning_rate = 0.01
mtm = 0.5
#mtm = 0.9

cee_train = []
cee_test = []
```

```
mce_train = []
mce_test = []

# use_cuda = torch.cuda.is_available()

for i in range(5):
    random.seed(i+1)
    avg_cee_training = []
    avg_cee_validation = []
    avg_mce_training = []
    avg_mce_validation = []

    net = SimpleNN()
    optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum = mtm)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(num_epoch):
        #print(f"Epoch {epoch+1}\n-------------------------------")

        train_size = len(loaders["train"].dataset)
        num_batches = len(loaders["train"])
        train_loss, correct = 0, 0
        for batch, (batch_x, batch_y) in enumerate(loaders["train"]):
            out = net(batch_x)
            loss = criterion(out, batch_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if batch % 100 == 0:
                loss_p, current = loss.item(), batch * len(batch_x)
                #print(f"loss: {loss_p:>7f}  [{current:>5d}/{train_size:>5d}]")

            train_loss += loss.item()
            correct += (out.argmax(1) == batch_y).type(torch.float).sum().item()

        train_loss /= num_batches
        correct /= train_size
        #print(f"Train Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{train_loss:>8f} \n")
        avg_cee_training.append(train_loss)
        avg_mce_training.append(100*correct)

        # validation
        test_size = len(loaders["test"].dataset)
        num_batches = len(loaders["test"])
        test_loss, correct = 0, 0
```

```
        with torch.no_grad():
            for X, y in loaders["test"]:
                pred = net(X)
                loss = criterion(pred, y)
                test_loss += loss.item()
                correct += (pred.argmax(1) == y).type(torch.float).sum().item()

        test_loss /= num_batches
        correct /= test_size
        #print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
 ↪{test_loss:>8f} \n")
        avg_cee_validation.append(test_loss)
        avg_mce_validation.append(100*correct)

    cee_train.append(avg_cee_training)
    cee_test.append(avg_cee_validation)
    mce_train.append(avg_mce_training)
    mce_test.append(avg_mce_validation)
```

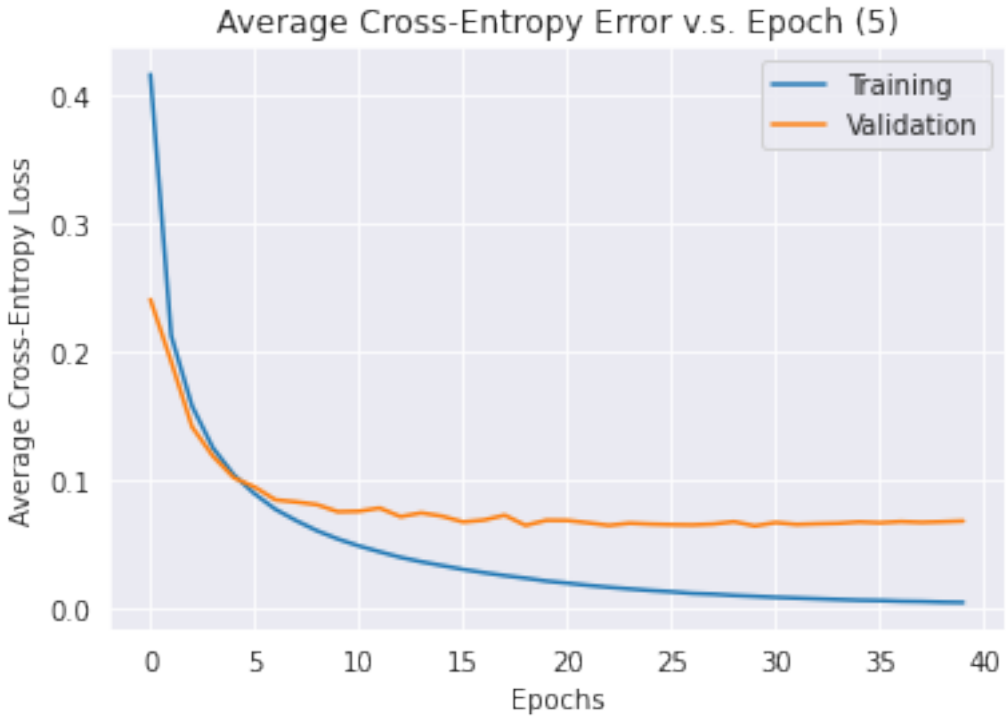
[24]:
```
#plot
for i in range(5):
    plt.title(f"Average Cross-Entropy Error v.s. Epoch ({i+1})")
    plt.plot(np.array(cee_train[i]), label="Training")
    plt.plot(np.array(cee_test[i]), label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel("Average Cross-Entropy Loss")
    plt.legend()
    plt.show()

#plot
for i in range(5):
    plt.title(f"Average Miss-Classification Error v.s. Epoch ({i+1})")
    plt.plot(100-np.array(mce_train[i]), label="Training")
    plt.plot(100-np.array(mce_test[i]), label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel("Average Miss-Classification Error(%)")
    plt.legend()
    plt.show()
```

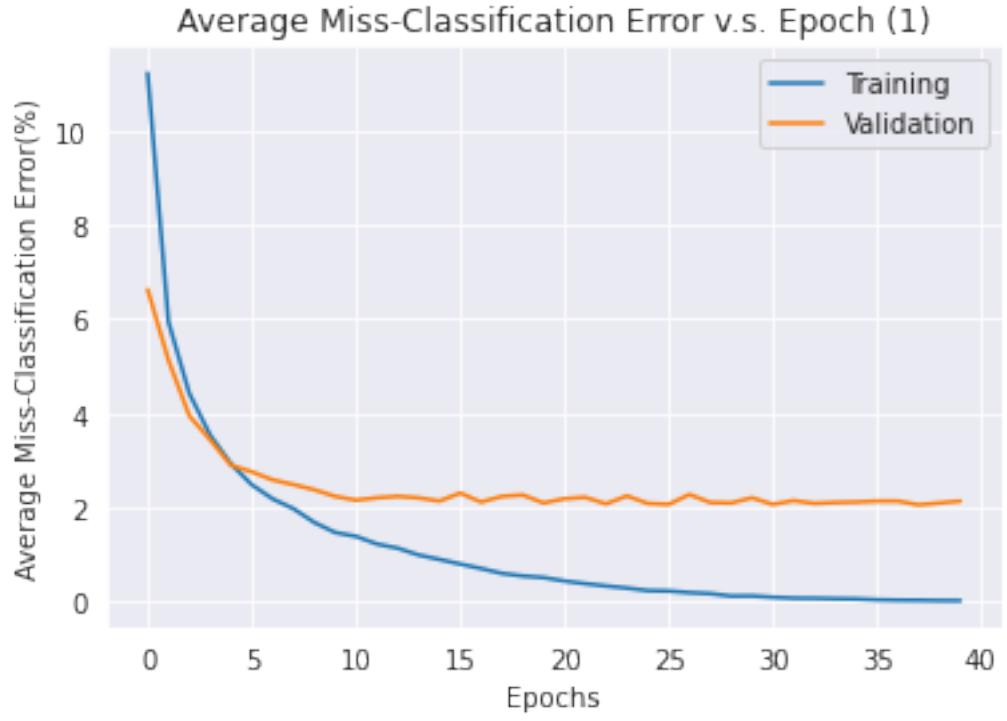

Average Cross-Entropy Error v.s. Epoch (1)
Training
Validation
Average Cross-Entropy Loss
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
25
30
35
40
Epochs

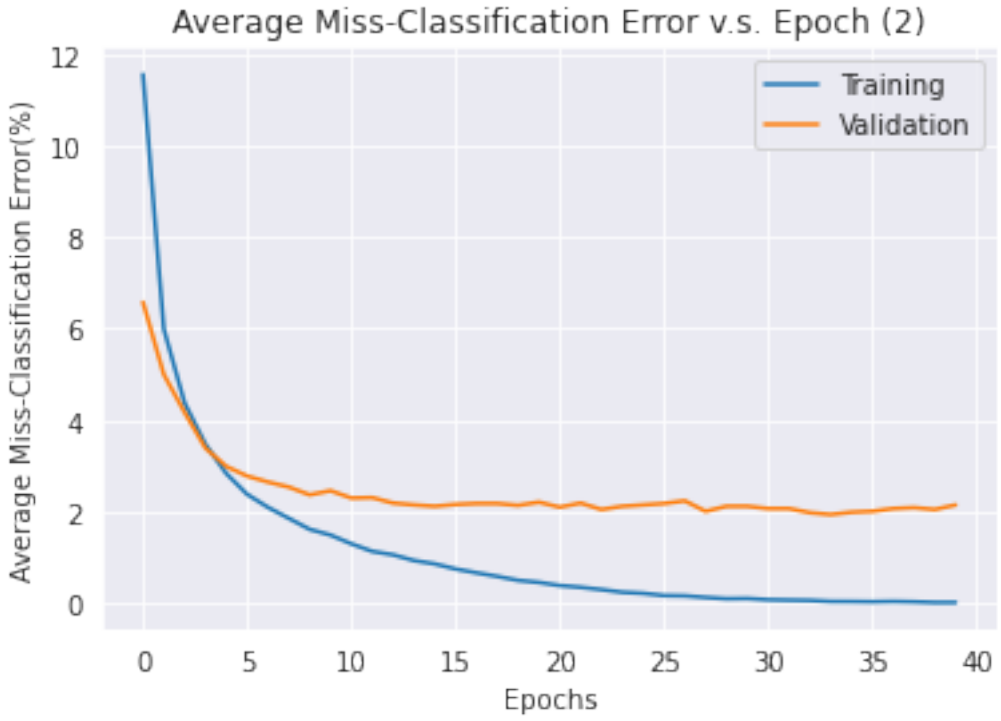

Average Cross-Entropy Error v.s. Epoch (2)
Training
Validation
Average Cross-Entropy Loss
0.4
0.3
0.2
0.1
0.0
0
5
10
15
20
25
30
35
40
Epochs

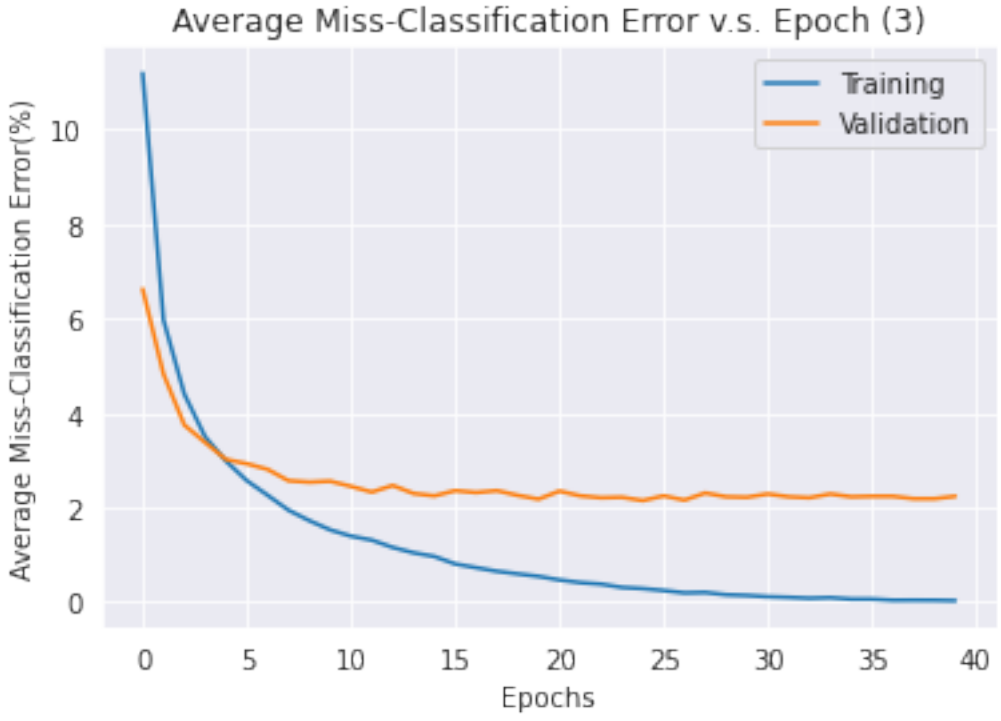

Average Cross-Entropy Error v.s. Epoch (3)
Training
Validation
Average Cross-Entropy Loss
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
25
30
35
40
Epochs

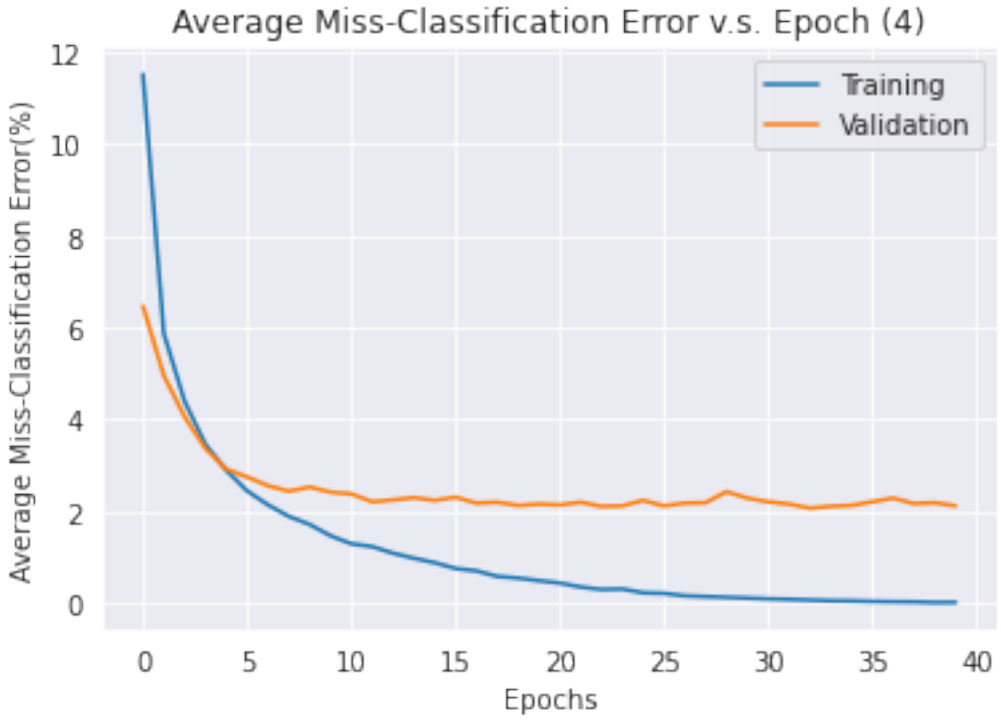

Average Cross-Entropy Error v.s. Epoch (4)
Training
Validation
Average Cross-Entropy Loss
0.4
0.3
0.2
0.1
0.0
0
5
10
15
20
25
30
35
40
Epochs

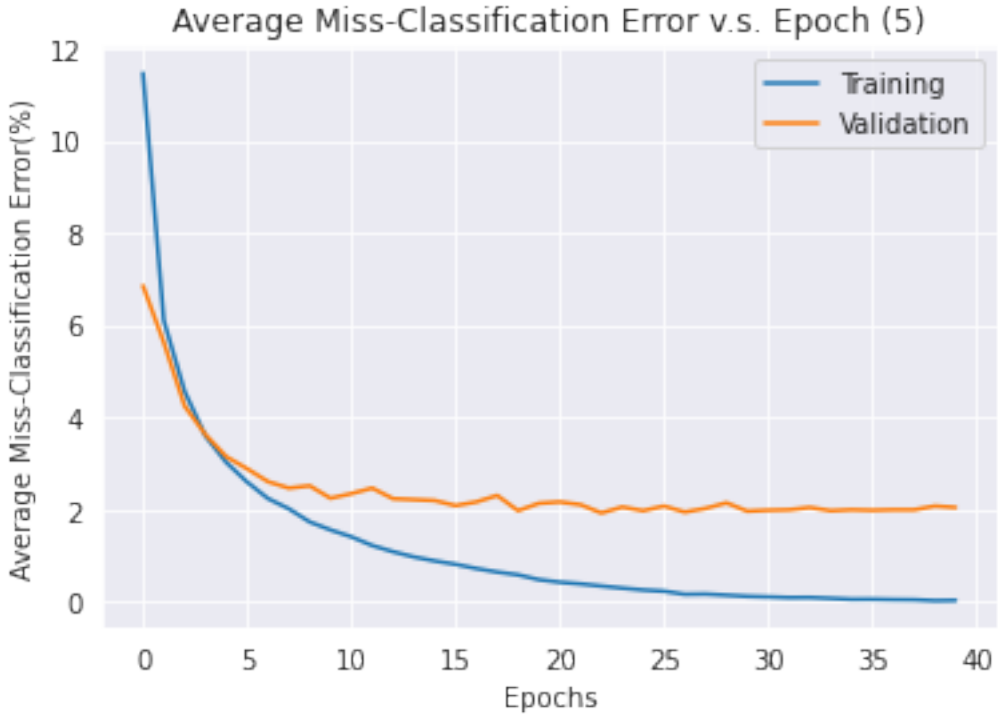

Average Cross-Entropy Error v.s. Epoch (5)
Training
Validation
Average Cross-Entropy Loss
0.0
0.1
0.2
0.3
0.4
0
5
10
15
20
25
30
35
40
Epochs


Average Miss-Classification Error v.s. Epoch (1)
Training
Validation
Average Miss-Classification Error(%)
0
2
4
6
8
10
0
5
10
15
20
25
30
35
40
Epochs


Average Miss-Classification Error v.s. Epoch (2)
Training
Validation
Average Miss-Classification Error(%)
Epochs
0
2
4
6
8
10
12
0
5
10
15
20
25
30
35
40


Average Miss-Classification Error v.s. Epoch (3)
Training
Validation
Average Miss-Classification Error(%)
Epochs
0
2
4
6
8
10
0
5
10
15
20
25
30
35
40


Average Miss-Classification Error v.s. Epoch (4)
Training
Validation
Average Miss-Classification Error(%)
Epochs
0
2
4
6
8
10
12
0
5
10
15
20
25
30
35
40


Average Miss-Classification Error v.s. Epoch (5)
Training
Validation
Average Miss-Classification Error(%)
Epochs
0
2
4
6
8
10
12
0
5
10
15
20
25
30
35
40

GR5241_Project2_Q4

May 4, 2022

[1]:
```
import torchvision
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import sklearn.preprocessing
import pandas as pd
sns.set_style('darkgrid')
%matplotlib inline
```

[2]:
```
# Q4.(a)
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import torch.utils.data as Data
from torchvision.transforms import ToTensor, Lambda
import random

# batchsize of dataset
batch_size = 100

train_data = datasets.MNIST(
    root = 'data',
    train = True,
    transform = ToTensor(),
    download = True,
)

test_data = datasets.MNIST(
    root = 'data',
    train = False,
    transform = ToTensor()
)

loaders = {
    'train' : torch.utils.data.DataLoader(train_data,
                                          batch_size=batch_size,
```

```
                                            shuffle=True),

    'test'  : torch.utils.data.DataLoader(test_data,
                                            batch_size=batch_size,
                                            shuffle=True)
}
```

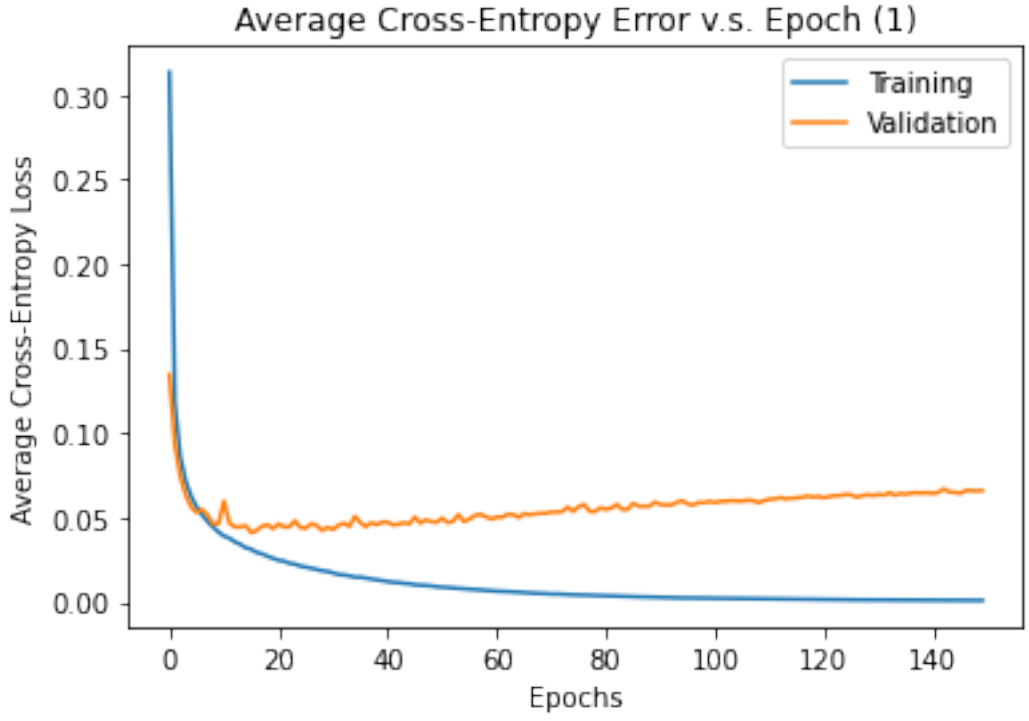
[3]:
```
#Q4.(a)
# Define a CNN class
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=5,
                stride=1,
                padding=2
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        # fully connected layer, output 10 classes
        self.out = nn.Linear(3136, 10)
    def forward(self, x):
        x = self.conv1(x)
        # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output, x    # return x for visualization

num_epoch = 150
learning_rate = 0.1
mtm = 0.0

cee_train_cnn = []
cee_test_cnn = []
mce_train_cnn = []
mce_test_cnn = []
```

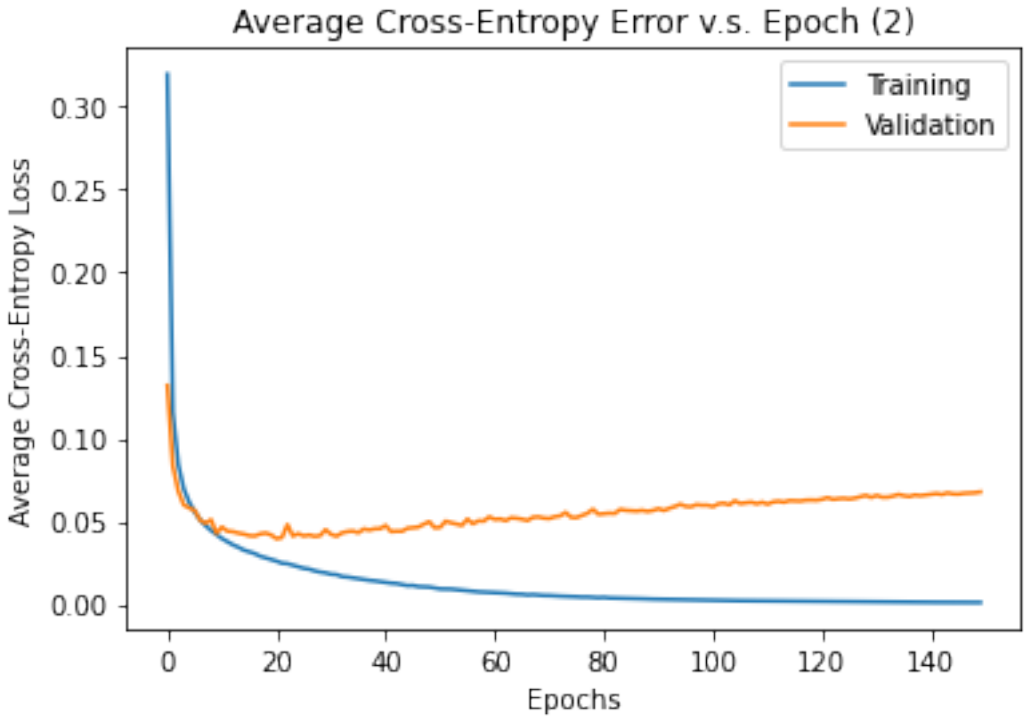
[4]:
```
for i in range(5):
    random.seed(i+1)
    avg_cee_training = []
    avg_cee_validation = []
    avg_mce_training = []
    avg_mce_validation = []
```

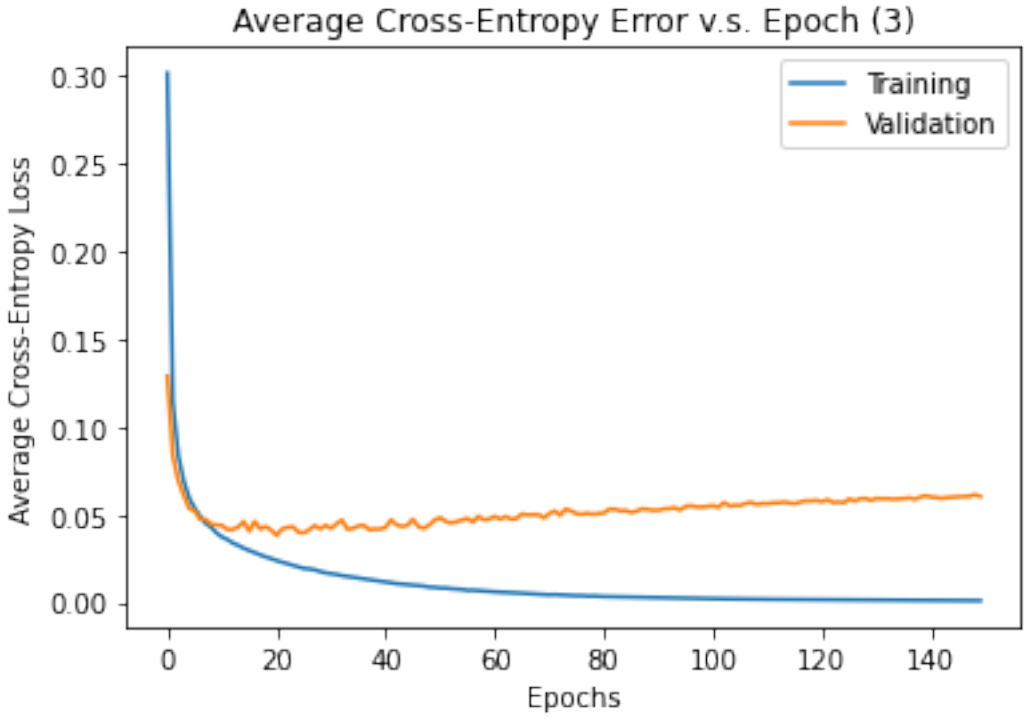
```
    cnn = CNN()
    optimizer = optim.SGD(cnn.parameters(), lr=learning_rate, momentum = mtm)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(num_epoch):
        #print(f"Epoch {epoch+1}\n-------------------------------")

        train_size = len(loaders["train"].dataset)
        num_batches = len(loaders["train"])
        train_loss, correct = 0, 0
        for batch, (batch_x, batch_y) in enumerate(loaders["train"]):
            # forward passing
            out = cnn(batch_x)[0]
            # print(type(out))
            loss = criterion(out, batch_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if batch % 100 == 0:
                loss_p, current = loss.item(), batch * len(batch_x)
                #print(f"loss: {loss_p:>7f}  [{current:>5d}/{train_size:>5d}]")

            train_loss += loss.item()
            correct += (out.argmax(1) == batch_y).type(torch.float).sum().item()

        train_loss /= num_batches
        correct /= train_size
        #print(f"Train Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{train_loss:>8f} \n")
        avg_cee_training.append(train_loss)
        avg_mce_training.append(100*correct)

        # validation
        test_size = len(loaders["test"].dataset)
        num_batches = len(loaders["test"])
        test_loss, correct = 0, 0

        with torch.no_grad():
            for X, y in loaders["test"]:
                pred = cnn(X)[0]
                loss = criterion(pred, y)
                test_loss += loss.item()
                correct += (pred.argmax(1) == y).type(torch.float).sum().item()

        test_loss /= num_batches
```

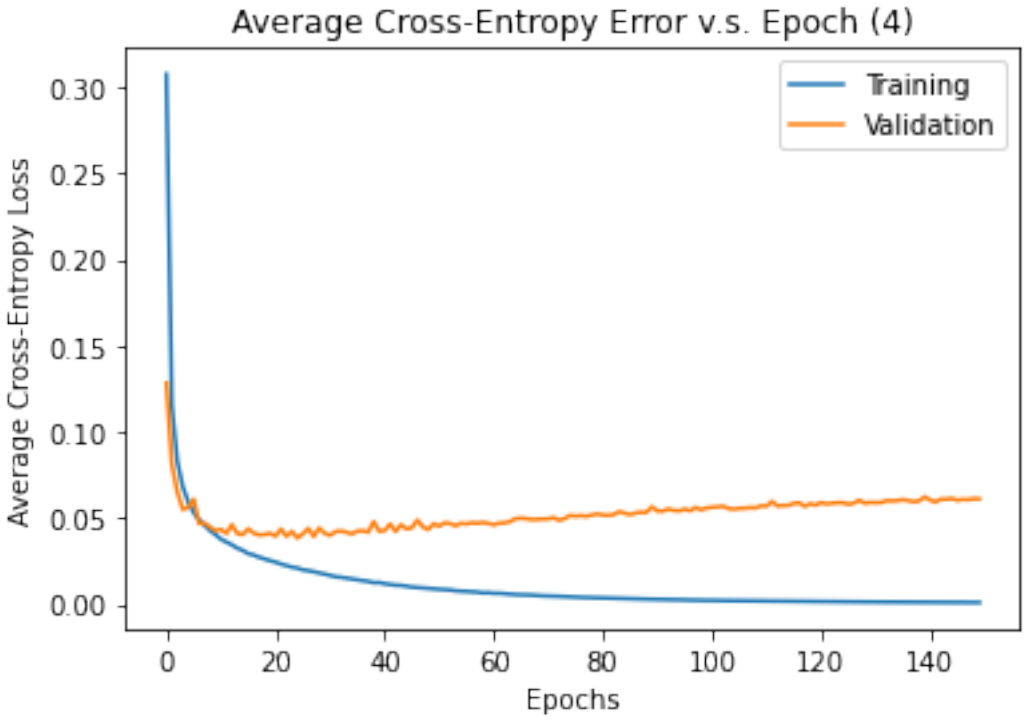
```
        correct /= test_size
        #print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{test_loss:>8f} \n")
        avg_cee_validation.append(test_loss)
        avg_mce_validation.append(100*correct)

    cee_train_cnn.append(avg_cee_training)
    cee_test_cnn.append(avg_cee_validation)
    mce_train_cnn.append(avg_mce_training)
    mce_test_cnn.append(avg_mce_validation)
```

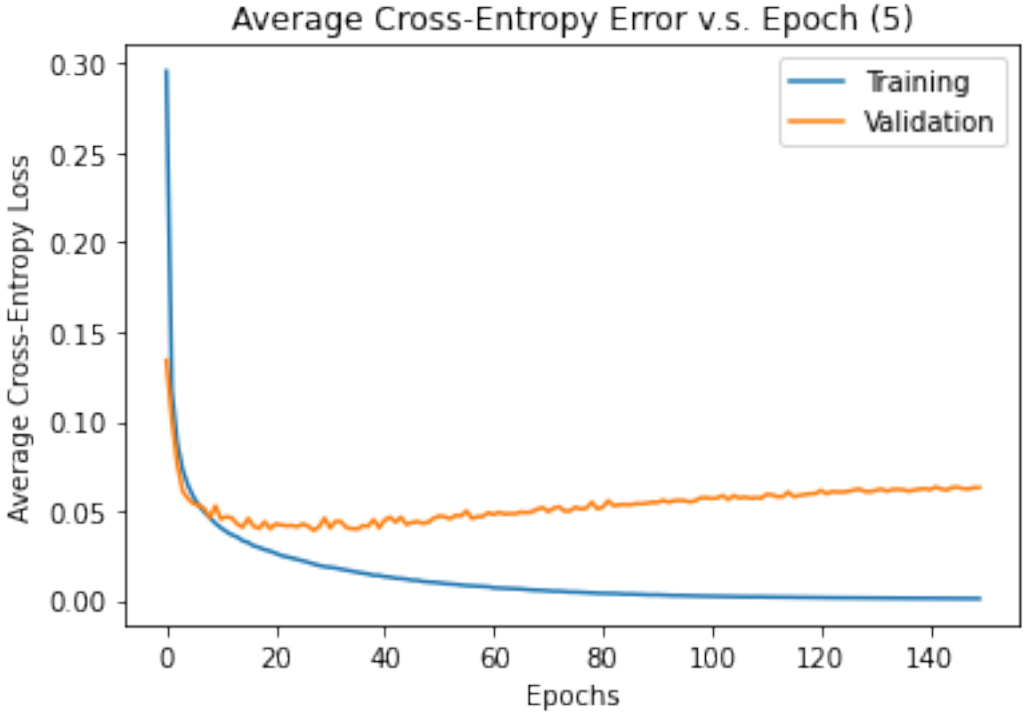
[5]:
```
#Q4.(a)
sns.set_style('darkgrid')
%matplotlib inline

for i in range(5):
    plt.title(f"Average Cross-Entropy Error v.s. Epoch ({i+1})")
    plt.plot(np.array(cee_train_cnn[i]), label="Training")
    plt.plot(np.array(cee_test_cnn[i]), label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel("Average Cross-Entropy Loss")
    plt.legend()
    plt.show()
```



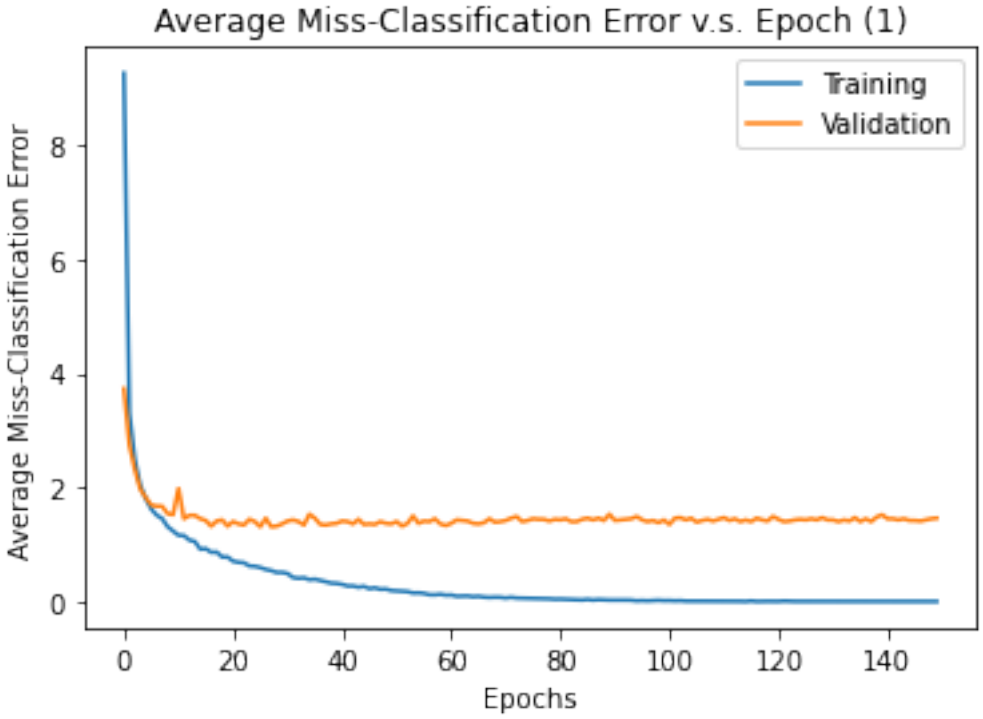

Average Cross-Entropy Error v.s. Epoch (2)
Training
Validation
Average Cross-Entropy Loss
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
120
140
Epochs

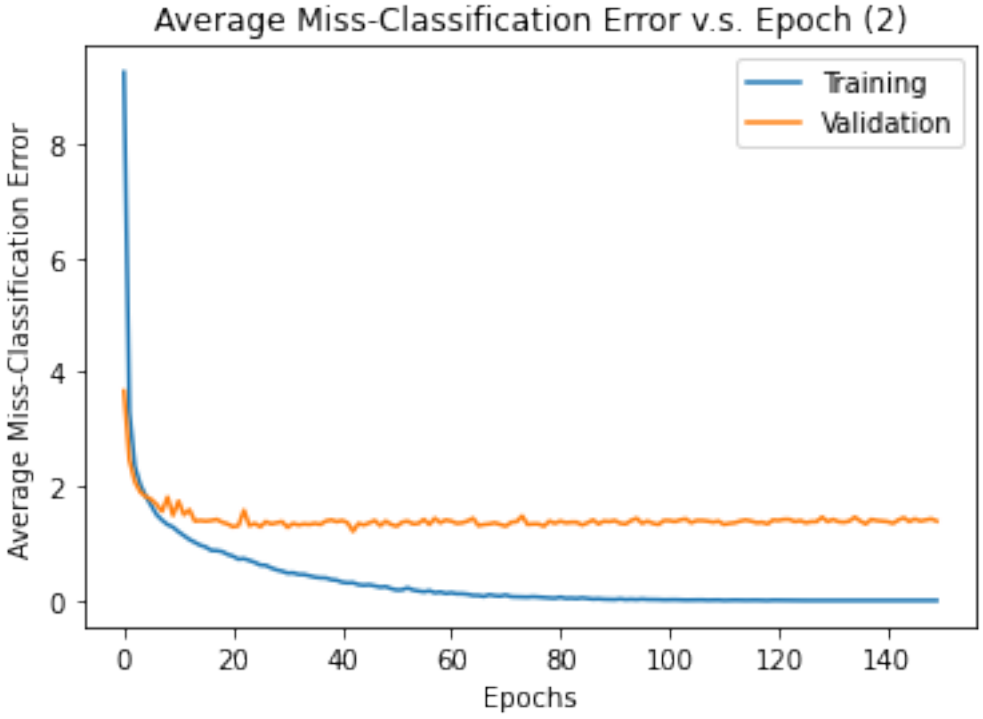

Average Cross-Entropy Error v.s. Epoch (3)
Training
Validation
Average Cross-Entropy Loss
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
120
140
Epochs

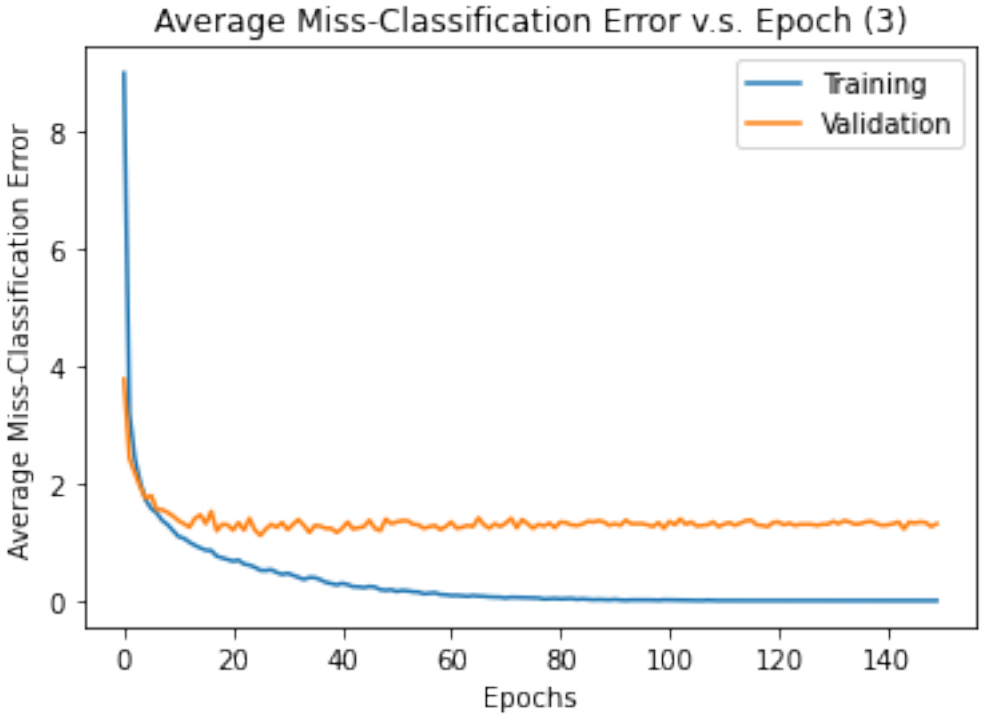

Average Cross-Entropy Error v.s. Epoch (4)
Training
Validation
Average Cross-Entropy Loss
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
120
140
Epochs

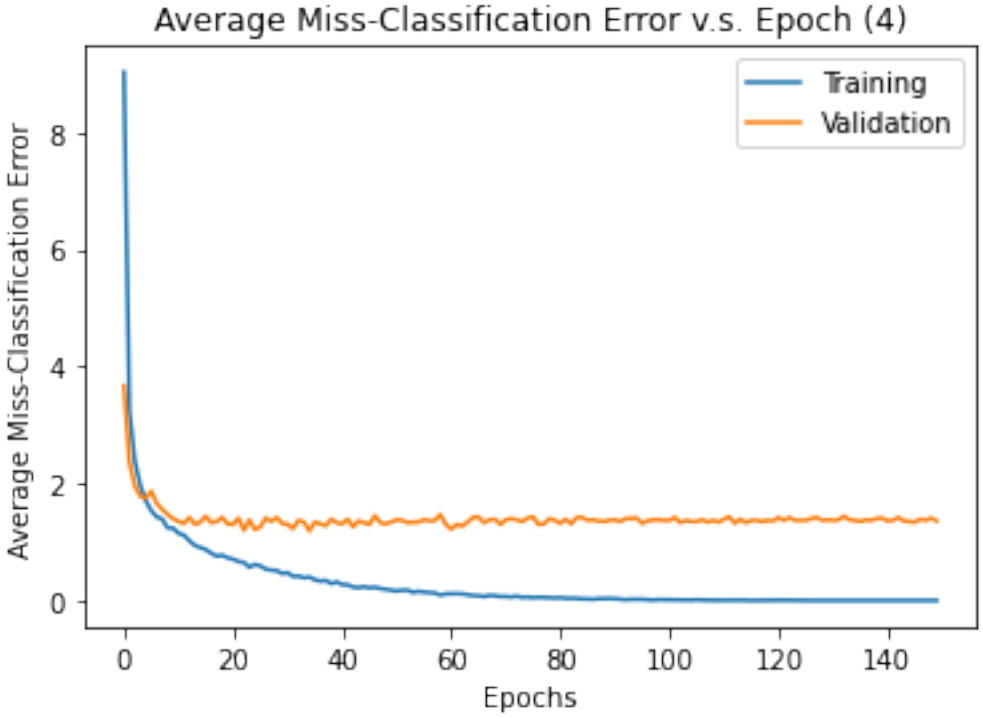

Average Cross-Entropy Error v.s. Epoch (5)
Training
Validation
Average Cross-Entropy Loss
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
100
120
140
Epochs

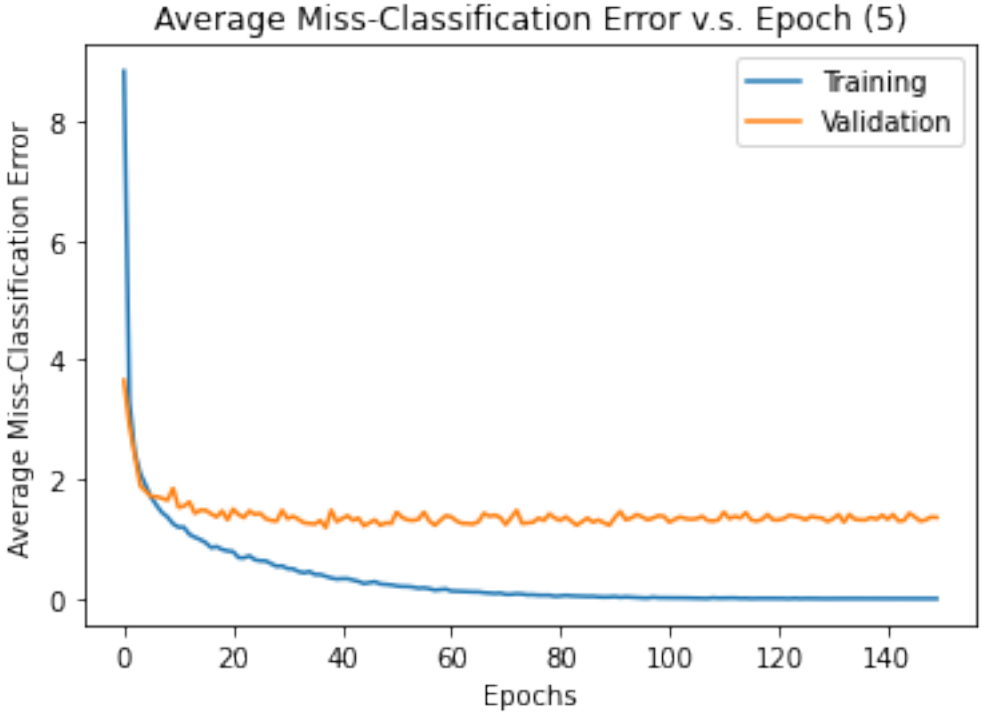
[6]:
```
#Q4.(b)
#plot
for i in range(5):
    plt.title(f"Average Miss-Classification Error v.s. Epoch ({i+1})")
    plt.plot(100-np.array(mce_train_cnn[i]), label="Training")
    plt.plot(100-np.array(mce_test_cnn[i]), label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel("Average Miss-Classification Error")
    plt.legend()
    plt.show()
```




Average Miss-Classification Error v.s. Epoch (2)
Training
Validation
Average Miss-Classification Error
0
2
4
6
8
0
20
40
60
80
100
120
140
Epochs


Average Miss-Classification Error v.s. Epoch (3)
Training
Validation
Average Miss-Classification Error
0
2
4
6
8
0
20
40
60
80
100
120
140
Epochs


Average Miss-Classification Error v.s. Epoch (4)
Training
Validation
Average Miss-Classification Error
0
2
4
6
8
0
20
40
60
80
100
120
140
Epochs


Average Miss-Classification Error v.s. Epoch (5)
Training
Validation
Average Miss-Classification Error
0
2
4
6
8
0
20
40
60
80
100
120
140
Epochs

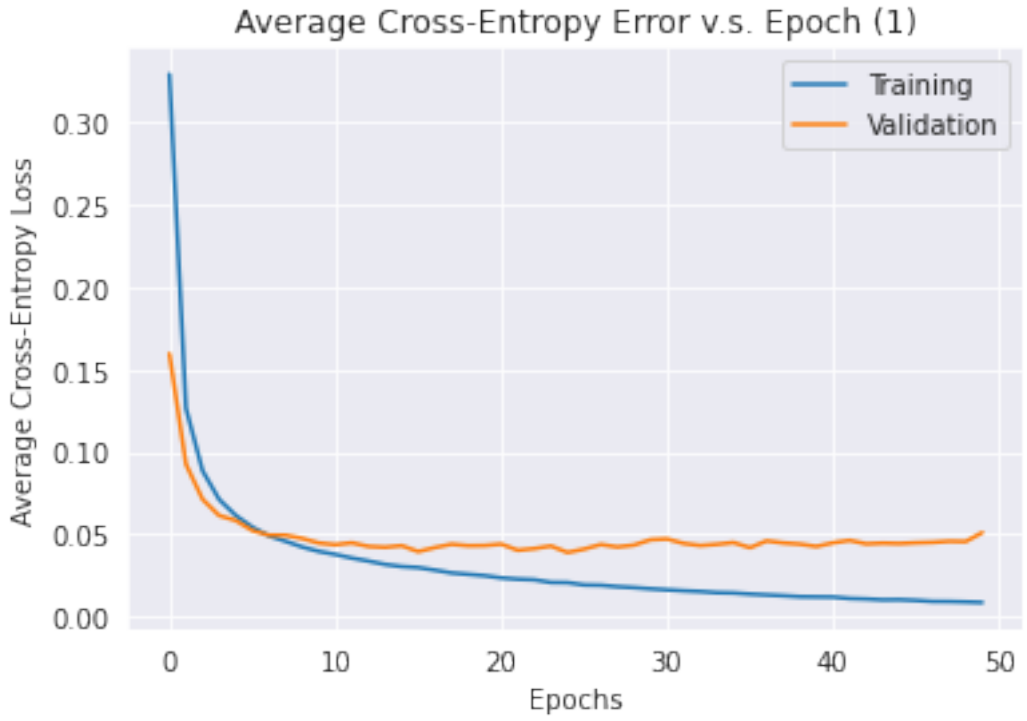
[27]:
```
#Q4.(c)
# find the best model first
params = list(cnn.parameters())[0]
plt.figure(figsize=(8, 8))
for i in range(params.shape[0]):
    plt.subplot(10, 10, i + 1) # Since we know it is a 10 x 10 grid
    x = params[i,:].detach().numpy()
    plt.imshow(x.reshape((5, 5)), cmap = "gray", interpolation = "nearest")
    plt.axis("off")
plt.subplots_adjust(wspace=0, hspace=0)
plt.savefig("4_c.png")
```

[18]:
```
#Q4.(d)
# change lr 0.1/0.01/0.2/0.5
# change momentum 0.0/0.5/0.9
num_epoch = 50
learning_rate = 0.01
mtm = 0.9

cee_train_cnn = []
cee_test_cnn = []
mce_train_cnn = []
mce_test_cnn = []

for i in range(1):
    random.seed(i+1)
    avg_cee_training = []
    avg_cee_validation = []
    avg_mce_training = []
    avg_mce_validation = []

    cnn = CNN()
    optimizer = optim.SGD(cnn.parameters(), lr=learning_rate, momentum = mtm)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(num_epoch):
        #print(f"Epoch {epoch+1}\n-------------------------------")
```

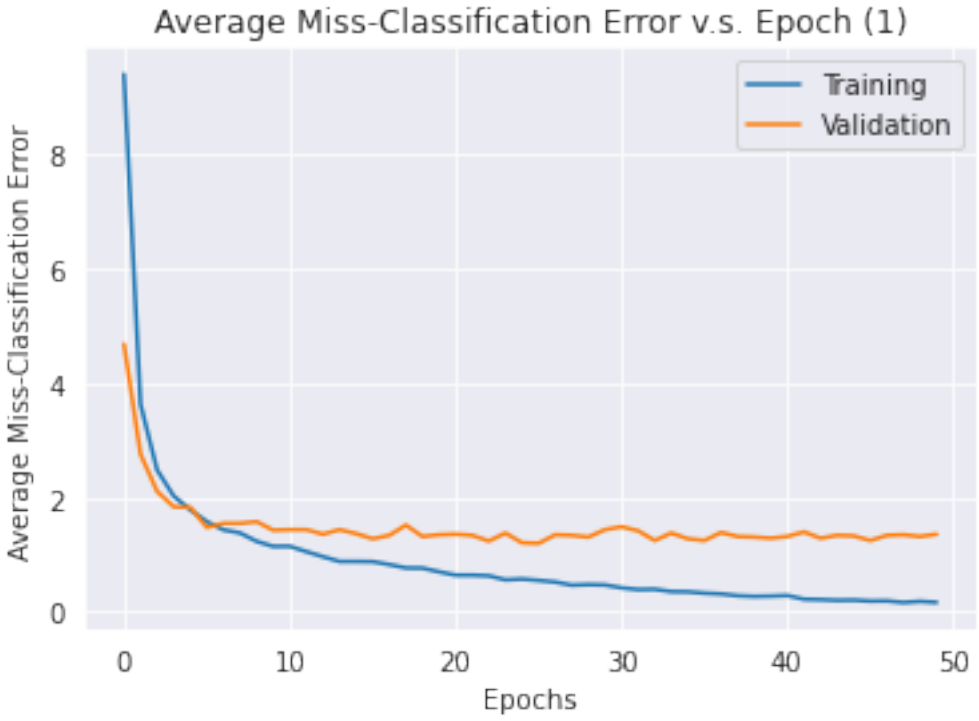
```
        train_size = len(loaders["train"].dataset)
        num_batches = len(loaders["train"])
        train_loss, correct = 0, 0
        for batch, (batch_x, batch_y) in enumerate(loaders["train"]):
            # forward passing
            out = cnn(batch_x)[0]
            # print(type(out))
            loss = criterion(out, batch_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if batch % 100 == 0:
                loss_p, current = loss.item(), batch * len(batch_x)
                #print(f"loss: {loss_p:>7f}  [{current:>5d}/{train_size:>5d}]")

            train_loss += loss.item()
            correct += (out.argmax(1) == batch_y).type(torch.float).sum().item()

        train_loss /= num_batches
        correct /= train_size
        #print(f"Train Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{train_loss:>8f} \n")
        avg_cee_training.append(train_loss)
        avg_mce_training.append(100*correct)

        # validation
        test_size = len(loaders["test"].dataset)
        num_batches = len(loaders["test"])
        test_loss, correct = 0, 0

        with torch.no_grad():
            for X, y in loaders["test"]:
                pred = cnn(X)[0]
                loss = criterion(pred, y)
                test_loss += loss.item()
                correct += (pred.argmax(1) == y).type(torch.float).sum().item()

        test_loss /= num_batches
        correct /= test_size
        #print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{test_loss:>8f} \n")
        avg_cee_validation.append(test_loss)
        avg_mce_validation.append(100*correct)

    cee_train_cnn.append(avg_cee_training)
```

```
    cee_test_cnn.append(avg_cee_validation)
    mce_train_cnn.append(avg_mce_training)
    mce_test_cnn.append(avg_mce_validation)
```

[19]:
```
#plot
for i in range(1):
    plt.title(f"Average Cross-Entropy Error v.s. Epoch ({i+1})")
    plt.plot(np.array(cee_train_cnn[i]), label="Training")
    plt.plot(np.array(cee_test_cnn[i]), label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel("Average Cross-Entropy Loss")
    plt.legend()
    plt.show()
```



[20]:
```
#plot
for i in range(1):
    plt.title(f"Average Miss-Classification Error v.s. Epoch ({i+1})")
    plt.plot(100-np.array(mce_train_cnn[i]), label="Training")
    plt.plot(100-np.array(mce_test_cnn[i]), label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel("Average Miss-Classification Error")
    plt.legend()
    plt.show()
```


Average Miss-Classification Error v.s. Epoch (1)
Training
Validation
Average Miss-Classification Error
0
2
4
6
8
0
10
20
30
40
50
Epochs

GR5241_Project2_Q5

May 4, 2022

[1]:
```
import torchvision
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import sklearn.preprocessing
import pandas as pd
sns.set_style('darkgrid')
%matplotlib inline
```

[2]:
```
# Q5.(a)
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import torch.utils.data as Data
from torchvision.transforms import ToTensor, Lambda
import random

# batchsize of dataset
batch_size = 100

train_data = datasets.MNIST(
    root = 'data',
    train = True,
    transform = ToTensor(),
    download = True,
)

test_data = datasets.MNIST(
    root = 'data',
    train = False,
    transform = ToTensor()
)

loaders = {
    'train' : torch.utils.data.DataLoader(train_data,
                                          batch_size=batch_size,
```

```
                                          shuffle=True),

    'test'  : torch.utils.data.DataLoader(test_data,
                                          batch_size=batch_size,
                                          shuffle=True)
}
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
data/MNIST/raw/train-images-idx3-ubyte.gz
```

```
  0%|          | 0/9912422 [00:00<?, ?it/s]
```

```
Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
data/MNIST/raw/train-labels-idx1-ubyte.gz
```

```
  0%|          | 0/28881 [00:00<?, ?it/s]
```

```
Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
data/MNIST/raw/t10k-images-idx3-ubyte.gz
```

```
  0%|          | 0/1648877 [00:00<?, ?it/s]
```

```
Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
data/MNIST/raw/t10k-labels-idx1-ubyte.gz
```

```
  0%|          | 0/4542 [00:00<?, ?it/s]
```

```
Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw
```

[3]:
```
#Q5.(a)
# Define a CNN class
class FavNN1(nn.Module):
    def __init__(self):
        super(FavNN1, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=5,
                stride=1,
```

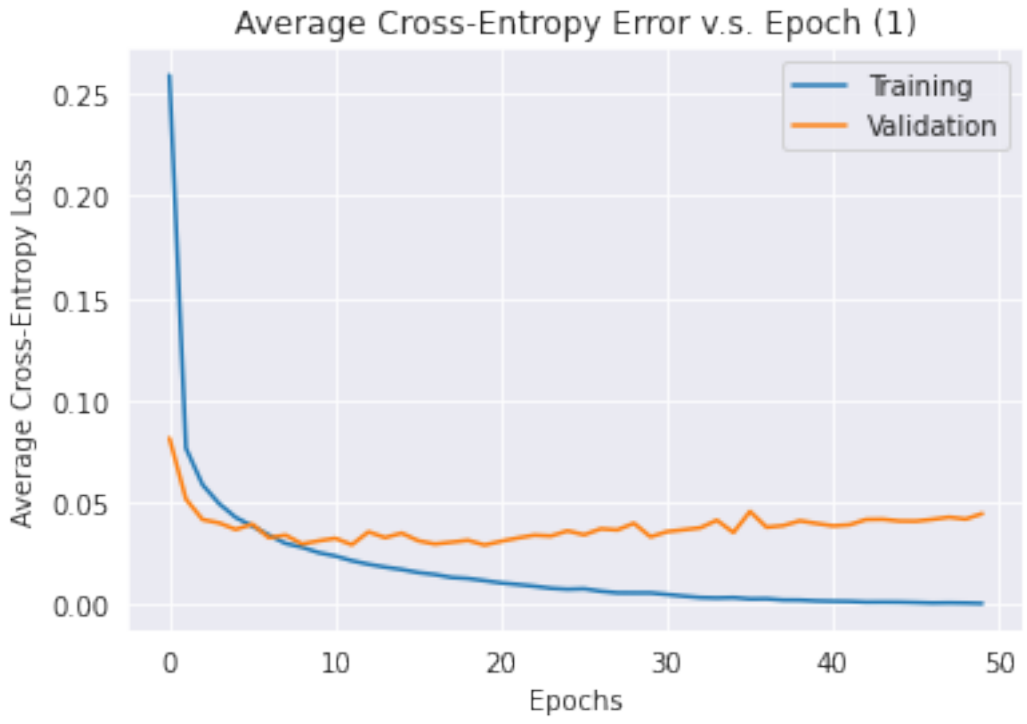
```
                padding=2
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )

        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        # fully connected layer, output 10 classes
        self.out = nn.Linear(32 * 7 * 7, 10)
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output, x    # return x for visualization

num_epoch = 50
learning_rate = 0.1
mtm = 0.0

cee_train_fav = []
cee_test_fav = []
mce_train_fav = []
mce_test_fav = []
```

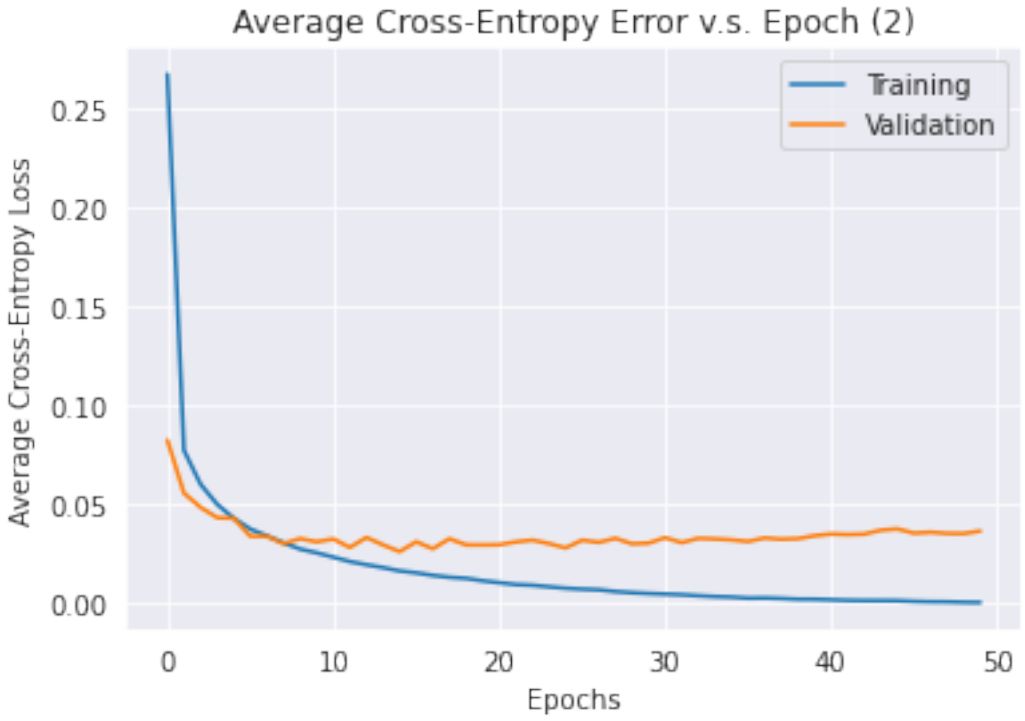
[4]:
```
for i in range(5):
    random.seed(i+1)
    avg_cee_training = []
    avg_cee_validation = []
    avg_mce_training = []
    avg_mce_validation = []

    fav = FavNN1()
    optimizer = optim.SGD(fav.parameters(), lr=learning_rate, momentum = mtm)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(num_epoch):
        #print(f"Epoch {epoch+1}\n-------------------------------")

        train_size = len(loaders["train"].dataset)
        num_batches = len(loaders["train"])
        train_loss, correct = 0, 0
```

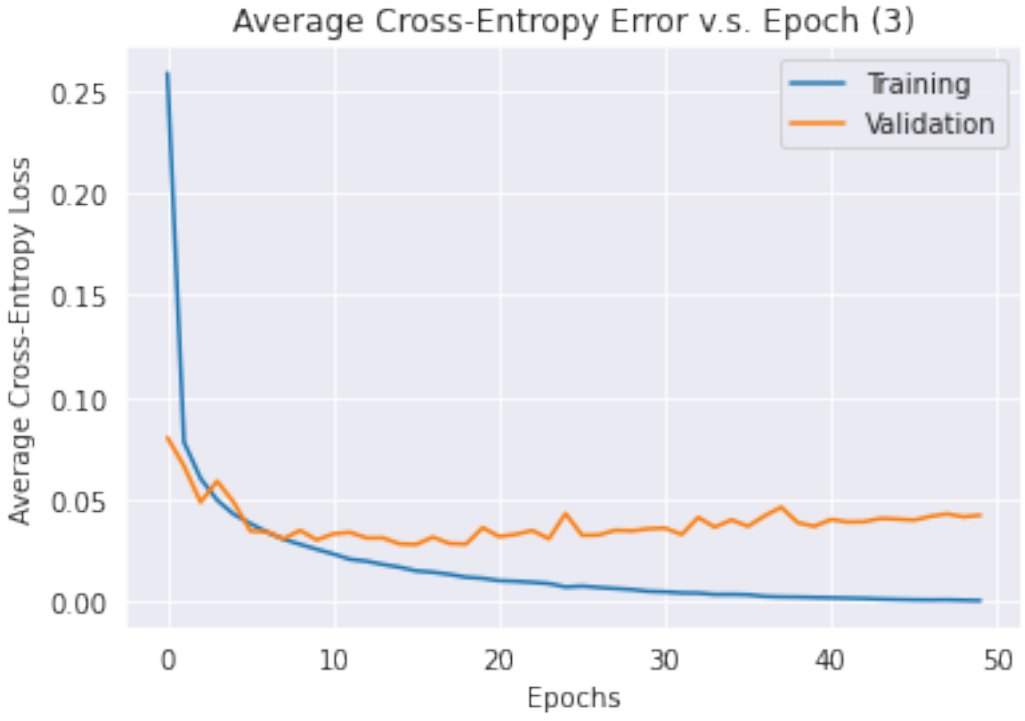
```
        for batch, (batch_x, batch_y) in enumerate(loaders["train"]):
            out = fav(batch_x)[0]
            loss = criterion(out, batch_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if batch % 100 == 0:
                loss_p, current = loss.item(), batch * len(batch_x)
                #print(f"loss: {loss_p:>7f}  [{current:>5d}/{train_size:>5d}]")

            train_loss += loss.item()
            correct += (out.argmax(1) == batch_y).type(torch.float).sum().item()

        train_loss /= num_batches
        correct /= train_size
        #print(f"Train Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{train_loss:>8f} \n")
        avg_cee_training.append(train_loss)
        avg_mce_training.append(100*correct)

        # validation
        test_size = len(loaders["test"].dataset)
        num_batches = len(loaders["test"])
        test_loss, correct = 0, 0

        with torch.no_grad():
            for X, y in loaders["test"]:
                pred = fav(X)[0]
                loss = criterion(pred, y)
                test_loss += loss.item()
                correct += (pred.argmax(1) == y).type(torch.float).sum().item()

        test_loss /= num_batches
        correct /= test_size
        #print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{test_loss:>8f} \n")
        avg_cee_validation.append(test_loss)
        avg_mce_validation.append(100*correct)

    cee_train_fav.append(avg_cee_training)
    cee_test_fav.append(avg_cee_validation)
    mce_train_fav.append(avg_mce_training)
    mce_test_fav.append(avg_mce_validation)
```

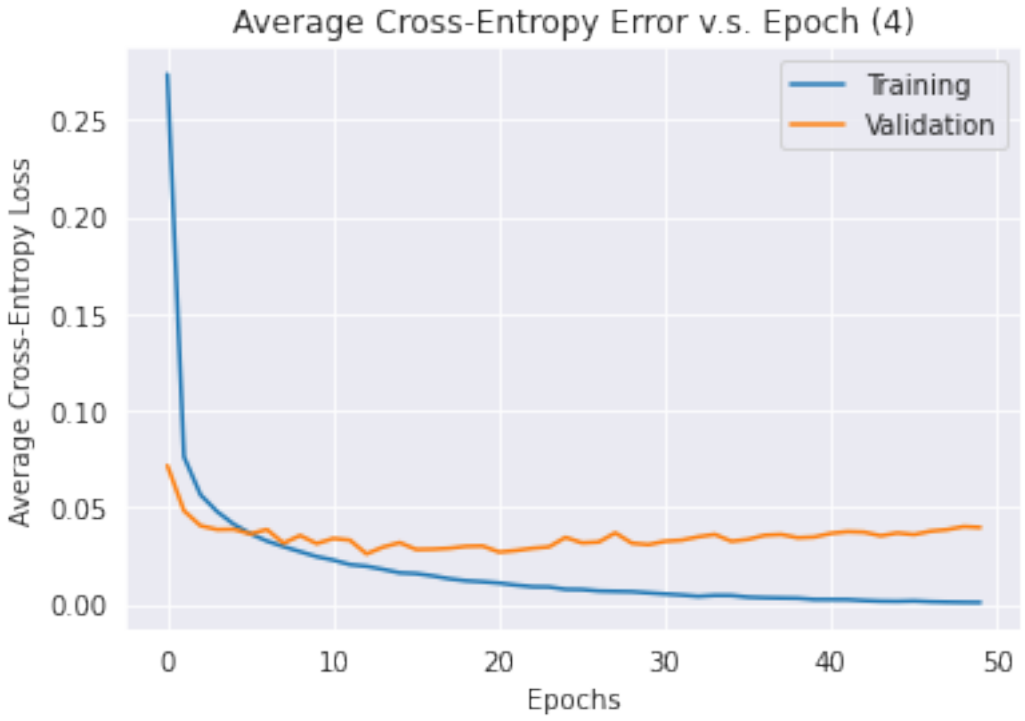
[5]:
```
#Q5.(a)
#plot
```

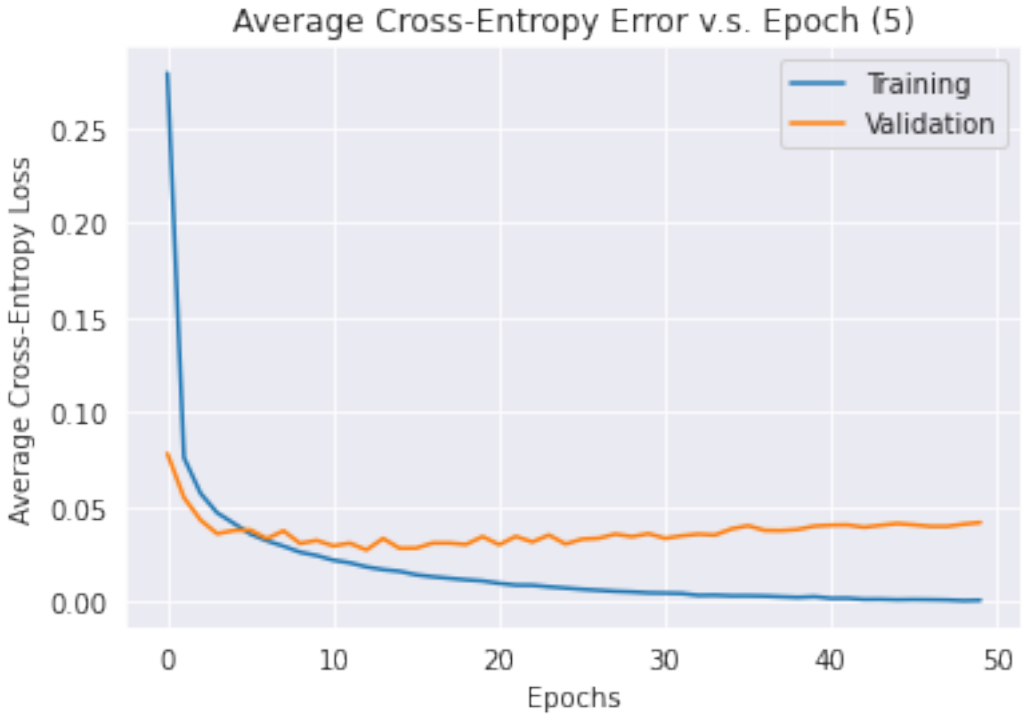
```
for i in range(5):
    plt.title(f"Average Cross-Entropy Error v.s. Epoch ({i+1})")
    plt.plot(np.array(cee_train_fav[i]), label="Training")
    plt.plot(np.array(cee_test_fav[i]), label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel("Average Cross-Entropy Loss")
    plt.legend()
    plt.show()
```



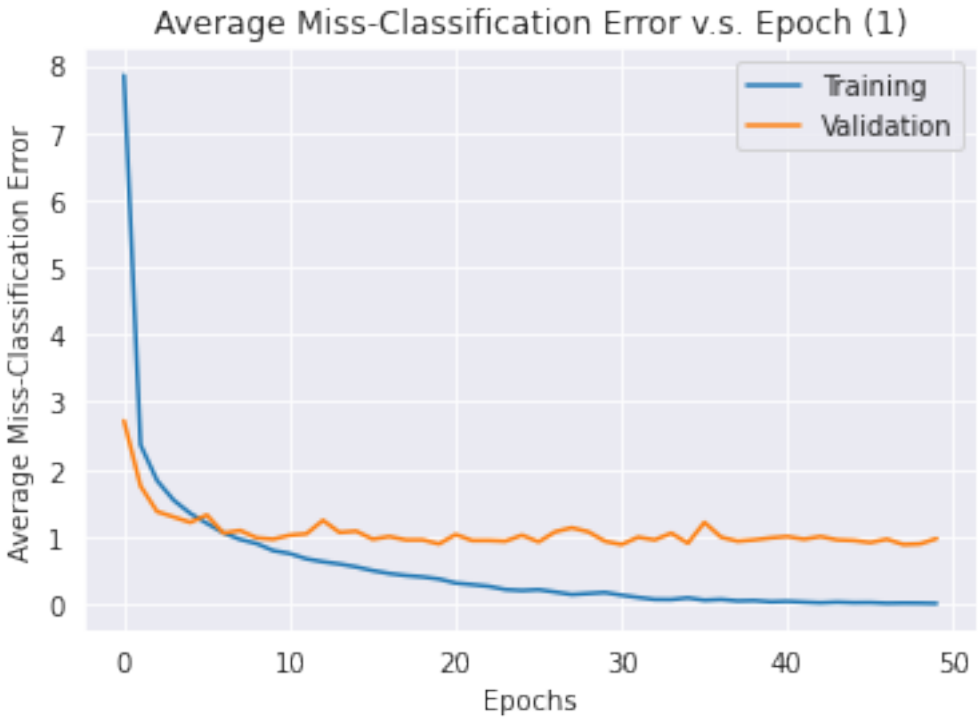

Average Cross-Entropy Error v.s. Epoch (2)
Training
Validation
Average Cross-Entropy Loss
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50
Epochs

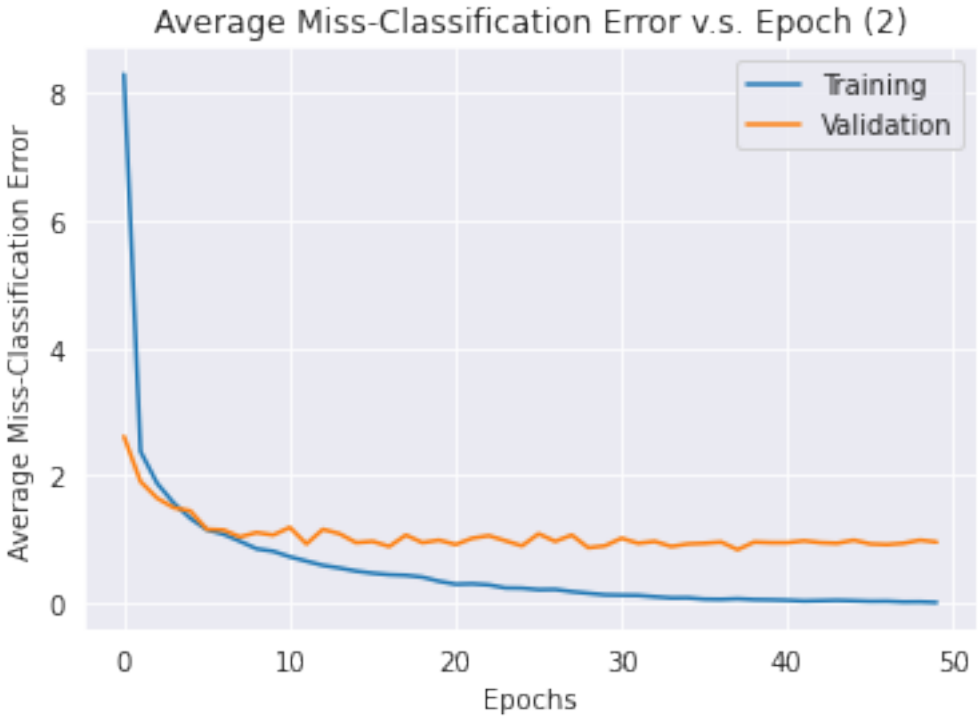

Average Cross-Entropy Error v.s. Epoch (3)
Training
Validation
Average Cross-Entropy Loss
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50
Epochs

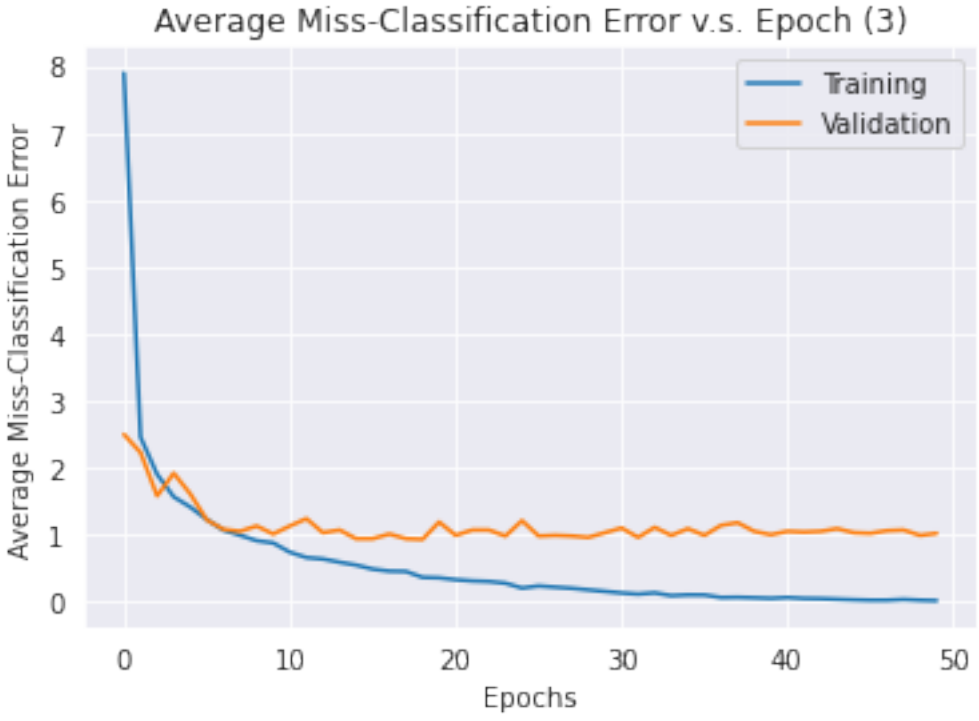

Average Cross-Entropy Error v.s. Epoch (4)
Training
Validation
Average Cross-Entropy Loss
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50
Epochs

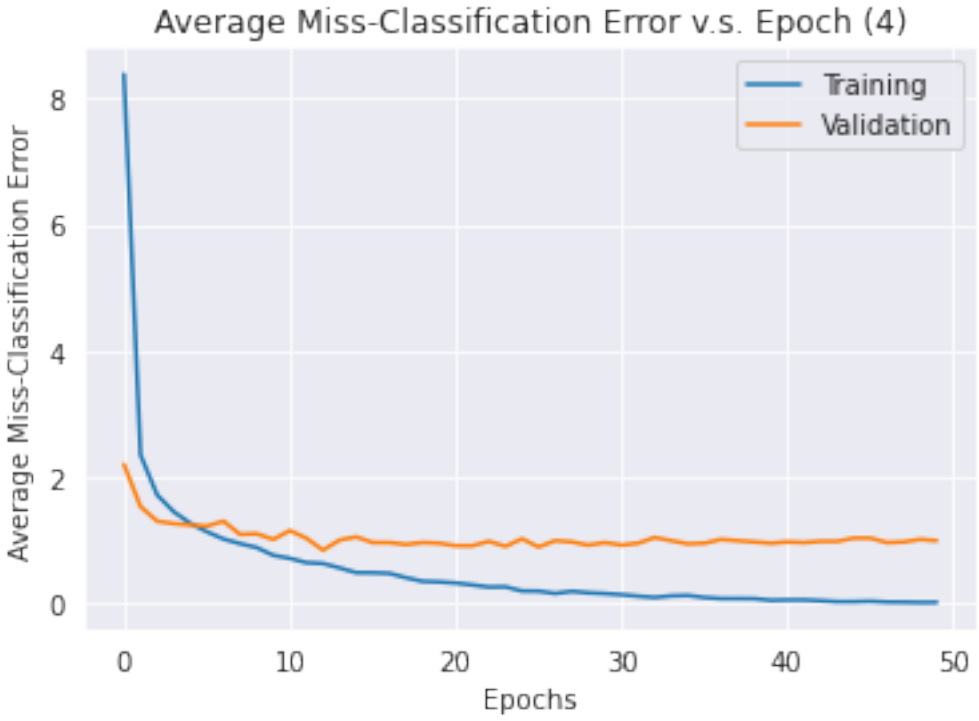

Average Cross-Entropy Error v.s. Epoch (5)
Training
Validation
Average Cross-Entropy Loss
0.25
0.20
0.15
0.10
0.05
0.00
0
10
20
30
40
50
Epochs

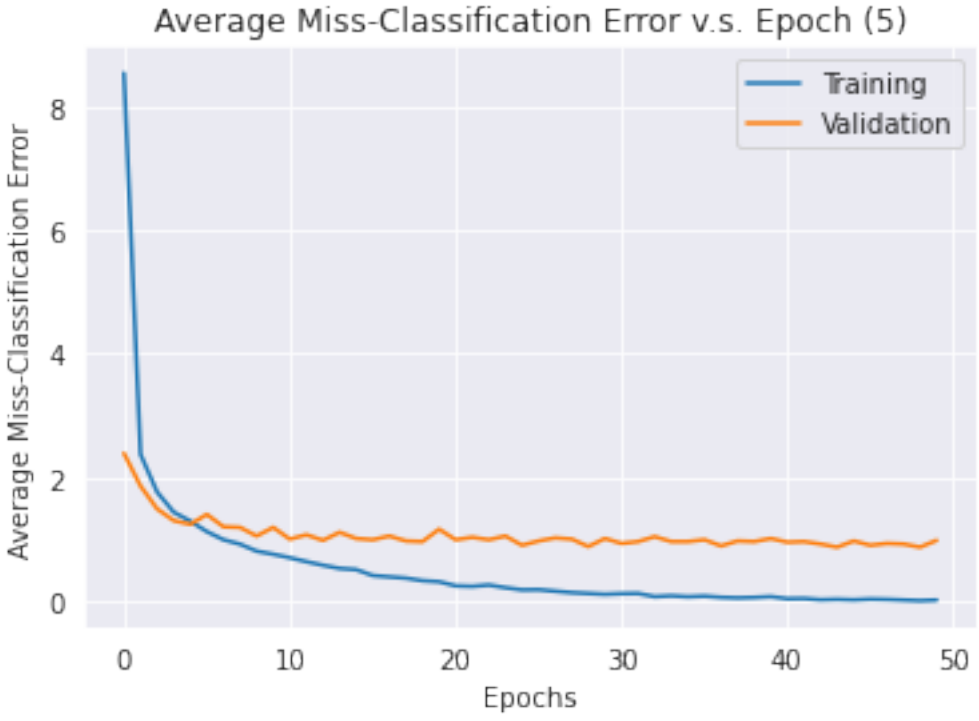
[6]:
```
#Q5.(b)
#plot
for i in range(5):
    plt.title(f"Average Miss-Classification Error v.s. Epoch ({i+1})")
    plt.plot(100-np.array(mce_train_fav[i]), label="Training")
    plt.plot(100-np.array(mce_test_fav[i]), label="Validation")
    plt.xlabel("Epochs")
    plt.ylabel("Average Miss-Classification Error")
    plt.legend()
    plt.show()
```



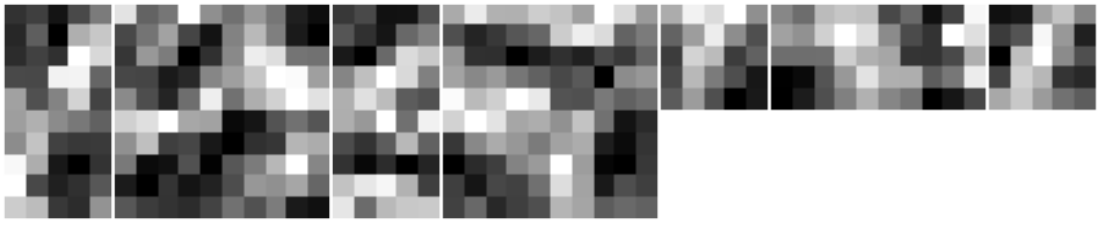

Average Miss-Classification Error v.s. Epoch (2)
Training
Validation
Average Miss-Classification Error
0
2
4
6
8
0
10
20
30
40
50
Epochs


Average Miss-Classification Error v.s. Epoch (3)
Training
Validation
Average Miss-Classification Error
0
1
2
3
4
5
6
7
8
0
10
20
30
40
50
Epochs


Average Miss-Classification Error v.s. Epoch (4)
Training
Validation
Average Miss-Classification Error
0
2
4
6
8
0
10
20
30
40
50
Epochs


Average Miss-Classification Error v.s. Epoch (5)
Training
Validation
Average Miss-Classification Error
0
2
4
6
8
0
10
20
30
40
50
Epochs

[7]:
```
#Q5.(c)
# find the best model first
params = list(fav.parameters())[0]
plt.figure(figsize=(8, 8))
for i in range(params.shape[0]):
    plt.subplot(10, 10, i + 1) # Since we know it is a 10 x 10 grid
    x = params[i,:].detach().numpy()
    plt.imshow(x.reshape((5, 5)), cmap = "gray", interpolation = "nearest")
    plt.axis("off")
plt.subplots_adjust(wspace=0, hspace=0)
plt.savefig("5_c.png")
```



[8]:
```
#Q5.(d)
# change lr 0.1/0.01/0.2/0.5
# change momentum 0.0/0.5/0.9
cee_train_fav = []
cee_test_fav = []
mce_train_fav = []
mce_test_fav = []

num_epoch = 18 #afrer 18 epoch the model stop improving
learning_rate = 0.01
mtm = 0.9
for i in range(1):
    random.seed(i+1)
    avg_cee_training = []
    avg_cee_validation = []
    avg_mce_training = []
    avg_mce_validation = []

    fav = FavNN1()
    optimizer = optim.SGD(fav.parameters(), lr=learning_rate, momentum = mtm)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(num_epoch):
        print(f"Epoch {epoch+1}\n---------------------------------")
```

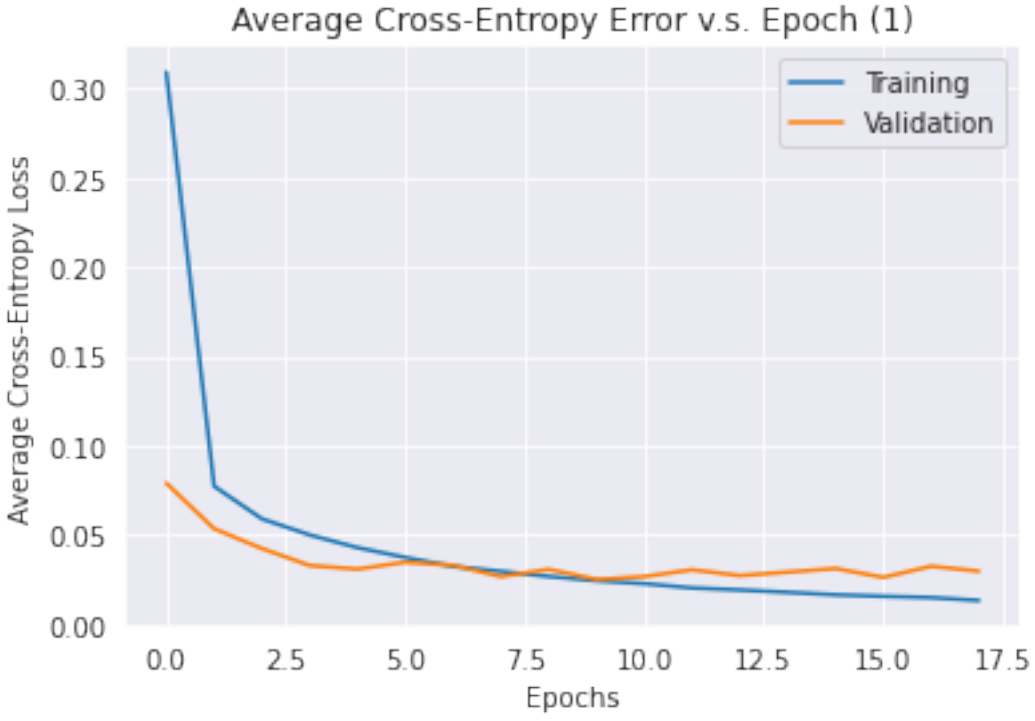
```
        train_size = len(loaders["train"].dataset)
        num_batches = len(loaders["train"])
        train_loss, correct = 0, 0
        for batch, (batch_x, batch_y) in enumerate(loaders["train"]):
            out = fav(batch_x)[0]
            loss = criterion(out, batch_y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if batch % 100 == 0:
                loss_p, current = loss.item(), batch * len(batch_x)
                #print(f"loss: {loss_p:>7f}  [{current:>5d}/{train_size:>5d}]")

            train_loss += loss.item()
            correct += (out.argmax(1) == batch_y).type(torch.float).sum().item()

        train_loss /= num_batches
        correct /= train_size
        print(f"Train Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{train_loss:>8f} \n")
        avg_cee_training.append(train_loss)
        avg_mce_training.append(100*correct)

        # validation
        test_size = len(loaders["test"].dataset)
        num_batches = len(loaders["test"])
        test_loss, correct = 0, 0

        with torch.no_grad():
            for X, y in loaders["test"]:
                pred = fav(X)[0]
                loss = criterion(pred, y)
                test_loss += loss.item()
                correct += (pred.argmax(1) == y).type(torch.float).sum().item()

        test_loss /= num_batches
        correct /= test_size
        print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss:␣
↪{test_loss:>8f} \n")
        avg_cee_validation.append(test_loss)
        avg_mce_validation.append(100*correct)

    cee_train_fav.append(avg_cee_training)
    cee_test_fav.append(avg_cee_validation)
    mce_train_fav.append(avg_mce_training)
    mce_test_fav.append(avg_mce_validation)
```

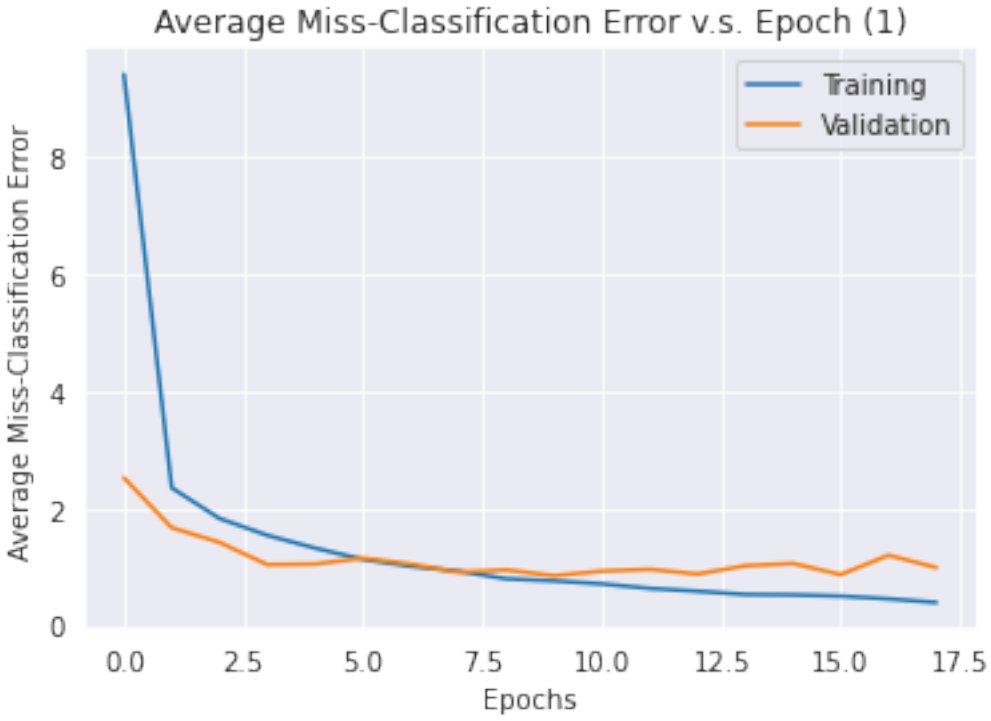
```
Epoch 1
-------------------------------
Train Error:
 Accuracy: 90.6%, Avg loss: 0.308977

Test Error:
 Accuracy: 97.5%, Avg loss: 0.079039

Epoch 2
-------------------------------
Train Error:
 Accuracy: 97.7%, Avg loss: 0.077330

Test Error:
 Accuracy: 98.3%, Avg loss: 0.053519

Epoch 3
-------------------------------
Train Error:
 Accuracy: 98.2%, Avg loss: 0.059044

Test Error:
 Accuracy: 98.6%, Avg loss: 0.042440

Epoch 4
-------------------------------
Train Error:
 Accuracy: 98.5%, Avg loss: 0.050043

Test Error:
 Accuracy: 99.0%, Avg loss: 0.032938

Epoch 5
-------------------------------
Train Error:
 Accuracy: 98.7%, Avg loss: 0.042932

Test Error:
 Accuracy: 99.0%, Avg loss: 0.030932

Epoch 6
-------------------------------
Train Error:
 Accuracy: 98.9%, Avg loss: 0.037458

Test Error:
 Accuracy: 98.9%, Avg loss: 0.034805
```

```
Epoch 7
-------------------------------
Train Error:
 Accuracy: 99.0%, Avg loss: 0.032416

Test Error:
 Accuracy: 99.0%, Avg loss: 0.033056

Epoch 8
-------------------------------
Train Error:
 Accuracy: 99.1%, Avg loss: 0.029677

Test Error:
 Accuracy: 99.1%, Avg loss: 0.026759

Epoch 9
-------------------------------
Train Error:
 Accuracy: 99.2%, Avg loss: 0.026821

Test Error:
 Accuracy: 99.1%, Avg loss: 0.030800

Epoch 10
-------------------------------
Train Error:
 Accuracy: 99.2%, Avg loss: 0.024442

Test Error:
 Accuracy: 99.2%, Avg loss: 0.025138

Epoch 11
-------------------------------
Train Error:
 Accuracy: 99.3%, Avg loss: 0.022803

Test Error:
 Accuracy: 99.1%, Avg loss: 0.026746

Epoch 12
-------------------------------
Train Error:
 Accuracy: 99.4%, Avg loss: 0.020386

Test Error:
 Accuracy: 99.0%, Avg loss: 0.030553
```

```
Epoch 13
-------------------------------
Train Error:
 Accuracy: 99.4%, Avg loss: 0.019258

Test Error:
 Accuracy: 99.1%, Avg loss: 0.027324

Epoch 14
-------------------------------
Train Error:
 Accuracy: 99.5%, Avg loss: 0.017943

Test Error:
 Accuracy: 99.0%, Avg loss: 0.029278

Epoch 15
-------------------------------
Train Error:
 Accuracy: 99.5%, Avg loss: 0.016555

Test Error:
 Accuracy: 98.9%, Avg loss: 0.031258

Epoch 16
-------------------------------
Train Error:
 Accuracy: 99.5%, Avg loss: 0.015704

Test Error:
 Accuracy: 99.1%, Avg loss: 0.026322

Epoch 17
-------------------------------
Train Error:
 Accuracy: 99.5%, Avg loss: 0.014991

Test Error:
 Accuracy: 98.8%, Avg loss: 0.032526

Epoch 18
-------------------------------
Train Error:
 Accuracy: 99.6%, Avg loss: 0.013382

Test Error:
 Accuracy: 99.0%, Avg loss: 0.029774
```

[9]:
```
#plot
plt.title(f"Average Cross-Entropy Error v.s. Epoch ({i+1})")
plt.plot(np.array(cee_train_fav[0]), label="Training")
plt.plot(np.array(cee_test_fav[0]), label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Average Cross-Entropy Loss")
plt.legend()
plt.show()

#plot
plt.title(f"Average Miss-Classification Error v.s. Epoch ({i+1})")
plt.plot(100-np.array(mce_train_fav[0]), label="Training")
plt.plot(100-np.array(mce_test_fav[0]), label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Average Miss-Classification Error")
plt.legend()
plt.show()
```




Average Miss-Classification Error v.s. Epoch (1)
Training
Validation
Average Miss-Classification Error
0
2
4
6
8
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

GR5241_Project2_Q6&Q7

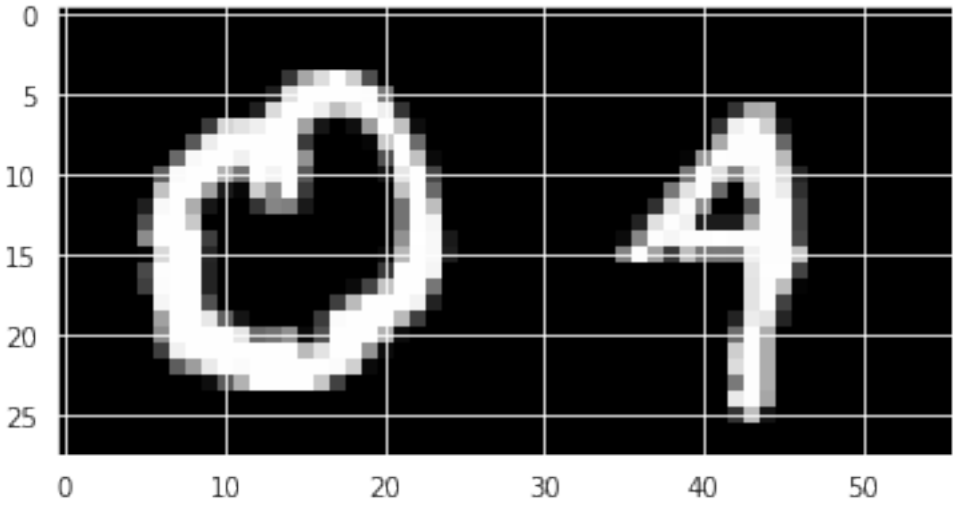
May 5, 2022

[2]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import torch.utils.data as Data
from torchvision.transforms import ToTensor, Lambda
import random
import torchvision
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import sklearn.preprocessing
import pandas as pd

sns.set_style('darkgrid')
%matplotlib inline
```

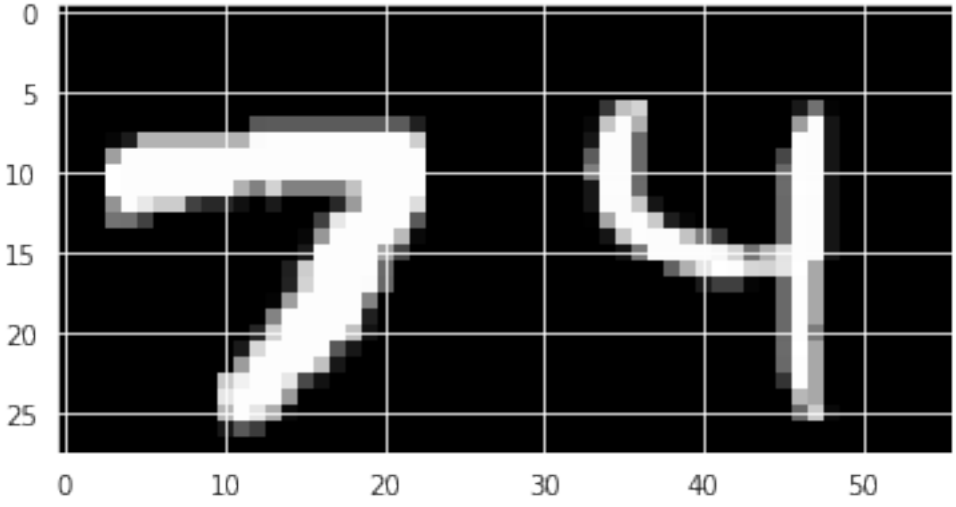
[3]:
```
#Q6
from numpy import genfromtxt
train_q6 = genfromtxt('train.txt', delimiter=',')
test_q6 = genfromtxt('test.txt', delimiter=',')
val_q6 = genfromtxt('val.txt', delimiter=',')
```

[4]:
```
# change i/j to a fix number or fix a seed to get fix result everytime
#i = random.randint(0, train_q6.shape[0])
i = 2
#j = random.randint(0, train_q6.shape[0])
j = 13

#reshape data
#sample 1
d1 = np.reshape(train_q6[i,:-1], (28,56))
extra = train_q6[i,-1]

plt.imshow(d1, cmap = plt.get_cmap('gray'))
plt.show()
print(f'last coordinate of line{i}: {extra}')
```

```
#sample 2
d1 = np.reshape(train_q6[j,:-1], (28,56))
extra = train_q6[j,-1]

plt.imshow(d1, cmap = plt.get_cmap('gray'))
plt.show()
print(f'last coordinate of line{j}: {extra}')
```




```
last coordinate of line2: 4.0
```




```
last coordinate of line13: 11.0
```

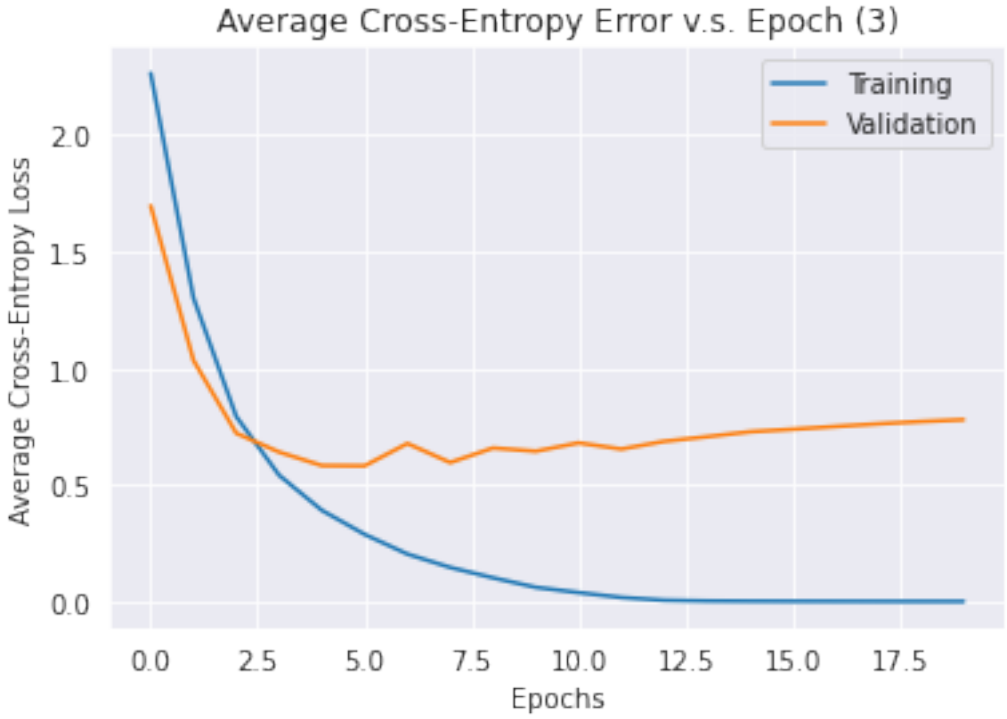
[5]:
```
#transfer data
X_train_reshape = train_q6[:,:-1].reshape((20000,28,56,1))
Y_train = train_q6[:,-1]

X_test_reshape = test_q6[:,:-1].reshape((5000,28,56,1))
Y_test = test_q6[:,-1]

X_vali_reshape = val_q6[:,:-1].reshape((5000,28,56,1))
Y_vali = val_q6[:,-1]
```

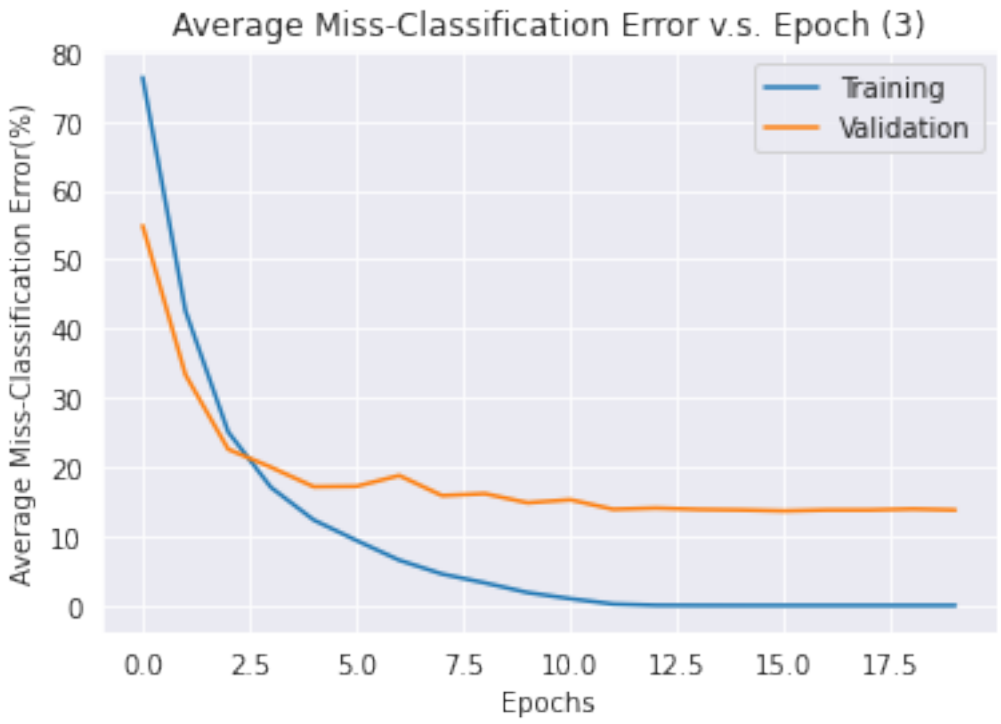
[6]:
```
#Q7(a)(b) model1
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
import tensorflow as tf

random.seed(0)
sgd = tf.keras.optimizers.SGD(learning_rate=0.1,momentum=0.0)
#create model
model = Sequential()
#add model layers
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,56,1)))
model.add(tf.keras.layers.MaxPooling2D(2,2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(19))
#compile model using accuracy to measure model performance
model.compile(optimizer=sgd, loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
#train the model
history = model.fit(X_train_reshape, Y_train, validation_data=(X_vali_reshape,␣
 ↪Y_vali), epochs=20)
res = pd.DataFrame(history.history)
res[['avg_cee_training','avg_cee_validation']] = res[['loss','val_loss']]
res[['avg_mce_training','avg_mce_validation']] =␣
 ↪(1-res[['accuracy','val_accuracy']])*100

plt.title(f"Average Cross-Entropy Error v.s. Epoch (0)")
plt.plot(res[['avg_cee_training']], label="Training")
plt.plot(res[['avg_cee_validation']], label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Average Cross-Entropy Loss")
plt.legend()
plt.show()

plt.title(f"Average Miss-Classification Error v.s. Epoch (0)")
plt.plot(res[['avg_mce_training']], label="Training")
plt.plot(res[['avg_mce_validation']], label="Validation")
```

```
plt.xlabel("Epochs")
plt.ylabel("Average Miss-Classification Error(%)")
plt.legend()
plt.show()
```

```
Epoch 1/20
625/625 [==============================] - 22s 33ms/step - loss: 2.2641 -
accuracy: 0.2365 - val_loss: 1.6962 - val_accuracy: 0.4510
Epoch 2/20
625/625 [==============================] - 19s 31ms/step - loss: 1.3016 -
accuracy: 0.5739 - val_loss: 1.0342 - val_accuracy: 0.6664
Epoch 3/20
625/625 [==============================] - 19s 31ms/step - loss: 0.7935 -
accuracy: 0.7492 - val_loss: 0.7220 - val_accuracy: 0.7742
Epoch 4/20
625/625 [==============================] - 19s 31ms/step - loss: 0.5424 -
accuracy: 0.8292 - val_loss: 0.6413 - val_accuracy: 0.8002
Epoch 5/20
625/625 [==============================] - 19s 31ms/step - loss: 0.3918 -
accuracy: 0.8764 - val_loss: 0.5834 - val_accuracy: 0.8284
Epoch 6/20
625/625 [==============================] - 19s 31ms/step - loss: 0.2887 -
accuracy: 0.9061 - val_loss: 0.5824 - val_accuracy: 0.8278
Epoch 7/20
625/625 [==============================] - 19s 31ms/step - loss: 0.2043 -
accuracy: 0.9345 - val_loss: 0.6780 - val_accuracy: 0.8122
Epoch 8/20
625/625 [==============================] - 19s 31ms/step - loss: 0.1471 -
accuracy: 0.9545 - val_loss: 0.5964 - val_accuracy: 0.8412
Epoch 9/20
625/625 [==============================] - 19s 31ms/step - loss: 0.1026 -
accuracy: 0.9673 - val_loss: 0.6587 - val_accuracy: 0.8384
Epoch 10/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0619 -
accuracy: 0.9814 - val_loss: 0.6447 - val_accuracy: 0.8516
Epoch 11/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0393 -
accuracy: 0.9902 - val_loss: 0.6806 - val_accuracy: 0.8472
Epoch 12/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0184 -
accuracy: 0.9979 - val_loss: 0.6542 - val_accuracy: 0.8610
Epoch 13/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0066 -
accuracy: 0.9998 - val_loss: 0.6871 - val_accuracy: 0.8592
Epoch 14/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0035 -
accuracy: 0.9999 - val_loss: 0.7069 - val_accuracy: 0.8612
Epoch 15/20
```

```
625/625 [==============================] - 20s 31ms/step - loss: 0.0025 -
accuracy: 1.0000 - val_loss: 0.7280 - val_accuracy: 0.8620
Epoch 16/20
625/625 [==============================] - 19s 31ms/step - loss: 0.0021 -
accuracy: 1.0000 - val_loss: 0.7389 - val_accuracy: 0.8634
Epoch 17/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0018 -
accuracy: 1.0000 - val_loss: 0.7507 - val_accuracy: 0.8620
Epoch 18/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0016 -
accuracy: 1.0000 - val_loss: 0.7621 - val_accuracy: 0.8620
Epoch 19/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0014 -
accuracy: 1.0000 - val_loss: 0.7720 - val_accuracy: 0.8606
Epoch 20/20
625/625 [==============================] - 20s 31ms/step - loss: 0.0013 -
accuracy: 1.0000 - val_loss: 0.7798 - val_accuracy: 0.8620
```





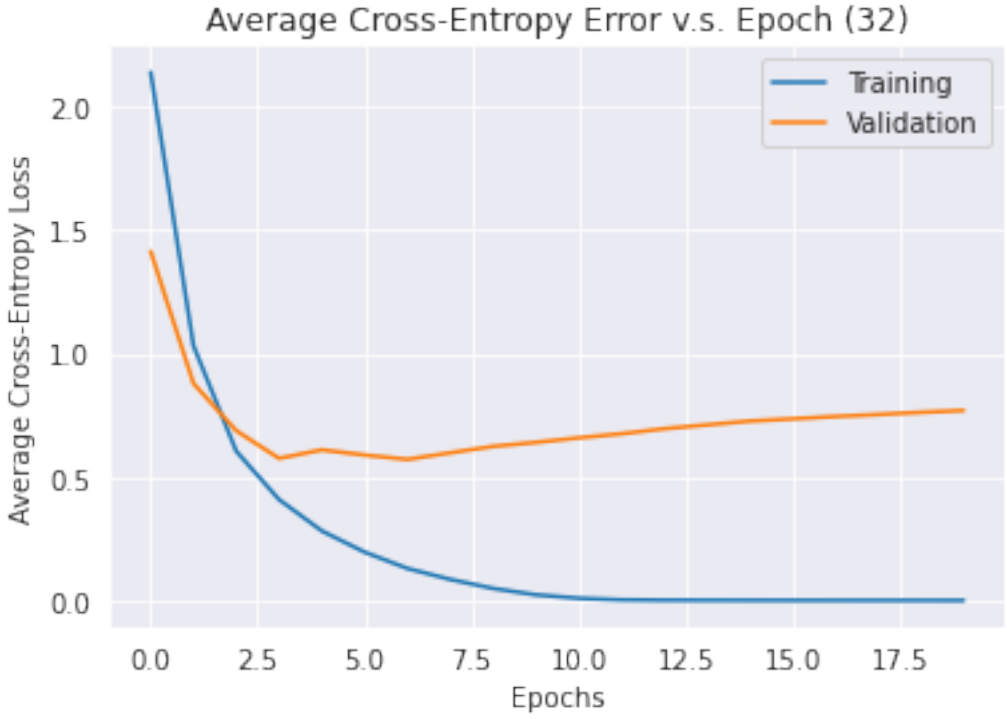
[11]:
```
#Q7(c)
params = model.layers[0].get_weights()[0].reshape(3,3,-1)
for i in range(params.shape[2]):
    plt.subplot(6, 6, i + 1)
    plt.imshow(params[:,:,i], cmap = "gray", interpolation = "nearest")
    plt.axis("off")
plt.subplots_adjust(wspace=0, hspace=0)
plt.show()
```

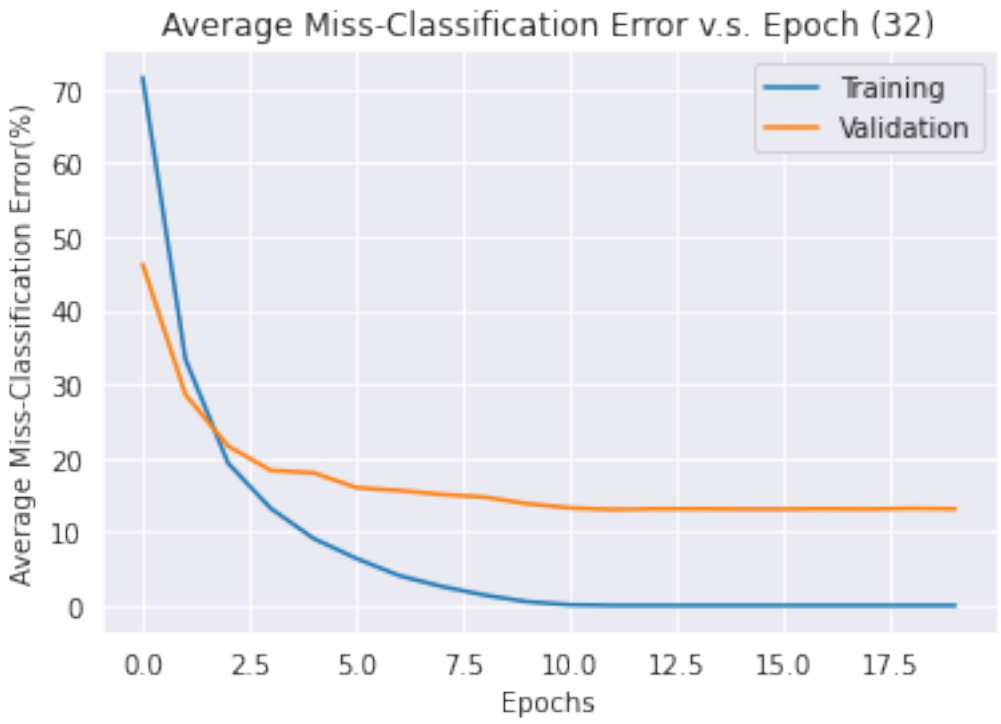
[12]:
```
#Q7(d)
random.seed(0)
sgd = tf.keras.optimizers.SGD(learning_rate=0.1,momentum=0.3)
#create model
model = Sequential()
#add model layers
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,56,1)))
model.add(tf.keras.layers.MaxPooling2D(2,2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(19))
#compile model using accuracy to measure model performance
model.compile(optimizer=sgd, loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
#train the model
history = model.fit(X_train_reshape, Y_train, validation_data=(X_vali_reshape,␣
 ↪Y_vali), epochs=20)
res = pd.DataFrame(history.history)
res[['avg_cee_training','avg_cee_validation']] = res[['loss','val_loss']]
res[['avg_mce_training','avg_mce_validation']] =␣
 ↪(1-res[['accuracy','val_accuracy']])*100

plt.title(f"Average Cross-Entropy Error v.s. Epoch (0)")
plt.plot(res[['avg_cee_training']], label="Training")
plt.plot(res[['avg_cee_validation']], label="Validation")
plt.xlabel("Epochs")
```

```
plt.ylabel("Average Cross-Entropy Loss")
plt.legend()
plt.show()

plt.title(f"Average Miss-Classification Error v.s. Epoch (0)")
plt.plot(res[['avg_mce_training']], label="Training")
plt.plot(res[['avg_mce_validation']], label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Average Miss-Classification Error(%)")
plt.legend()
plt.show()
```

```
Epoch 1/20
625/625 [==============================] - 21s 32ms/step - loss: 2.1385 -
accuracy: 0.2832 - val_loss: 1.4136 - val_accuracy: 0.5368
Epoch 2/20
625/625 [==============================] - 23s 37ms/step - loss: 1.0319 -
accuracy: 0.6657 - val_loss: 0.8791 - val_accuracy: 0.7136
Epoch 3/20
625/625 [==============================] - 22s 35ms/step - loss: 0.6046 -
accuracy: 0.8072 - val_loss: 0.6877 - val_accuracy: 0.7838
Epoch 4/20
625/625 [==============================] - 27s 43ms/step - loss: 0.4092 -
accuracy: 0.8684 - val_loss: 0.5757 - val_accuracy: 0.8168
Epoch 5/20
625/625 [==============================] - 26s 41ms/step - loss: 0.2822 -
accuracy: 0.9092 - val_loss: 0.6102 - val_accuracy: 0.8200
Epoch 6/20
625/625 [==============================] - 26s 42ms/step - loss: 0.1964 -
accuracy: 0.9362 - val_loss: 0.5893 - val_accuracy: 0.8402
Epoch 7/20
625/625 [==============================] - 34s 54ms/step - loss: 0.1303 -
accuracy: 0.9596 - val_loss: 0.5716 - val_accuracy: 0.8440
Epoch 8/20
625/625 [==============================] - 25s 41ms/step - loss: 0.0860 -
accuracy: 0.9743 - val_loss: 0.5982 - val_accuracy: 0.8494
Epoch 9/20
625/625 [==============================] - 25s 40ms/step - loss: 0.0494 -
accuracy: 0.9859 - val_loss: 0.6240 - val_accuracy: 0.8530
Epoch 10/20
625/625 [==============================] - 25s 39ms/step - loss: 0.0237 -
accuracy: 0.9951 - val_loss: 0.6404 - val_accuracy: 0.8620
Epoch 11/20
625/625 [==============================] - 23s 36ms/step - loss: 0.0096 -
accuracy: 0.9991 - val_loss: 0.6587 - val_accuracy: 0.8676
Epoch 12/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0033 -
accuracy: 1.0000 - val_loss: 0.6752 - val_accuracy: 0.8698
```

```
Epoch 13/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0019 -
accuracy: 1.0000 - val_loss: 0.6957 - val_accuracy: 0.8688
Epoch 14/20
625/625 [==============================] - 24s 38ms/step - loss: 0.0015 -
accuracy: 1.0000 - val_loss: 0.7112 - val_accuracy: 0.8688
Epoch 15/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0012 -
accuracy: 1.0000 - val_loss: 0.7266 - val_accuracy: 0.8692
Epoch 16/20
625/625 [==============================] - 20s 32ms/step - loss: 0.0011 -
accuracy: 1.0000 - val_loss: 0.7356 - val_accuracy: 0.8692
Epoch 17/20
625/625 [==============================] - 19s 31ms/step - loss: 9.3699e-04 -
accuracy: 1.0000 - val_loss: 0.7452 - val_accuracy: 0.8688
Epoch 18/20
625/625 [==============================] - 20s 31ms/step - loss: 8.4199e-04 -
accuracy: 1.0000 - val_loss: 0.7531 - val_accuracy: 0.8692
Epoch 19/20
625/625 [==============================] - 19s 31ms/step - loss: 7.6935e-04 -
accuracy: 1.0000 - val_loss: 0.7621 - val_accuracy: 0.8682
Epoch 20/20
625/625 [==============================] - 20s 32ms/step - loss: 7.0477e-04 -
accuracy: 1.0000 - val_loss: 0.7695 - val_accuracy: 0.8688
```



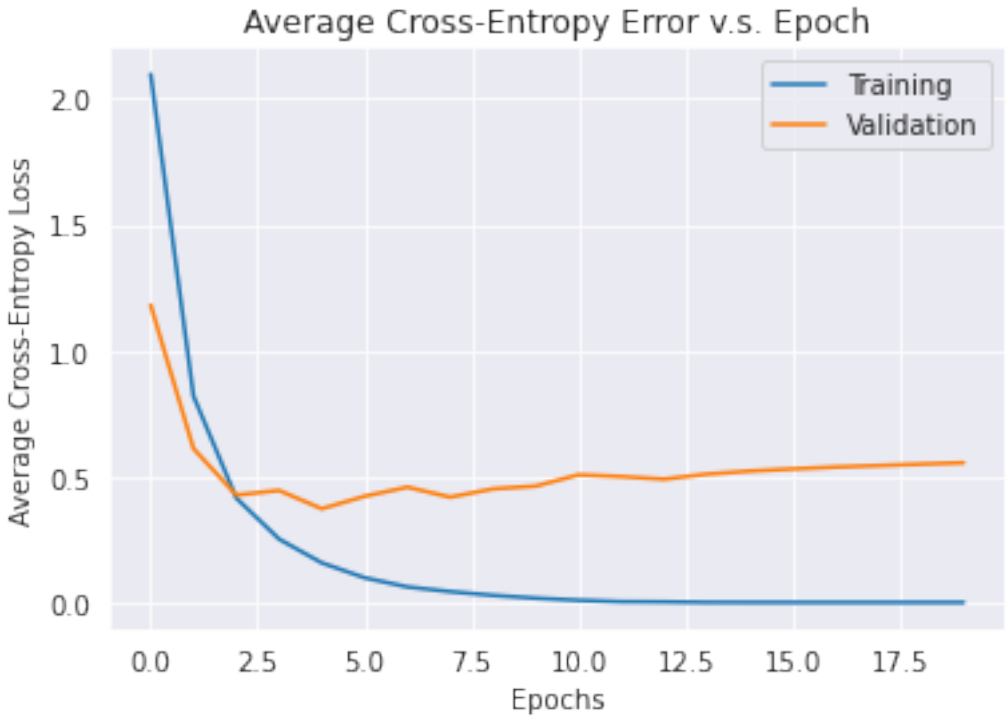


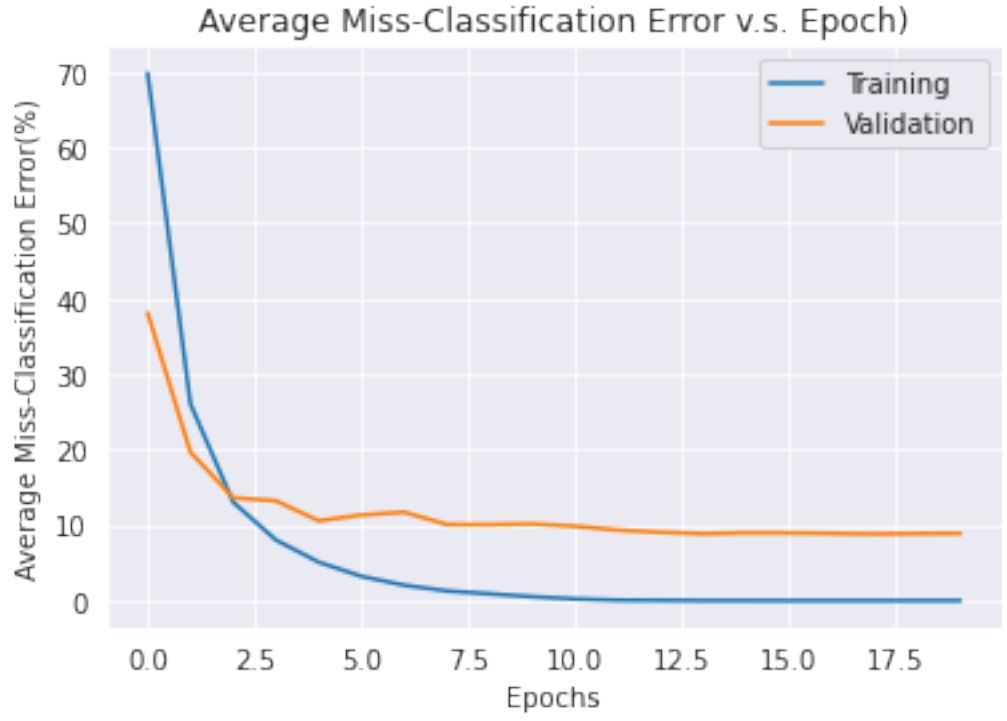
[13]:
```
#Q7(a)(b)model2
random.seed(0)
sgd = tf.keras.optimizers.SGD(learning_rate=0.1,momentum=0.0)
#create model
model2 = Sequential()
#add model layers
model2.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,56,1)))
model2.add(tf.keras.layers.MaxPooling2D(2,2))
model2.add(Conv2D(32, (3,3), activation='relu'))
model2.add(Flatten())
model2.add(Dense(64, activation='relu'))
model2.add(Dense(19))
#compile model using accuracy to measure model performance
model2.compile(optimizer=sgd, loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
#train the model
history = model2.fit(X_train_reshape, Y_train, validation_data=(X_vali_reshape,␣
 ↪Y_vali), epochs=20)
res = pd.DataFrame(history.history)
res[['avg_cee_training','avg_cee_validation']] = res[['loss','val_loss']]
res[['avg_mce_training','avg_mce_validation']] =␣
 ↪(1-res[['accuracy','val_accuracy']])*100
```

```
plt.title(f"Average Cross-Entropy Error v.s. Epoch")
plt.plot(res[['avg_cee_training']], label="Training")
plt.plot(res[['avg_cee_validation']], label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Average Cross-Entropy Loss")
plt.legend()
plt.show()

plt.title(f"Average Miss-Classification Error v.s. Epoch)")
plt.plot(res[['avg_mce_training']], label="Training")
plt.plot(res[['avg_mce_validation']], label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Average Miss-Classification Error(%)")
plt.legend()
plt.show()
```

```
Epoch 1/20
625/625 [==============================] - 33s 52ms/step - loss: 2.0977 -
accuracy: 0.3017 - val_loss: 1.1810 - val_accuracy: 0.6188
Epoch 2/20
625/625 [==============================] - 32s 51ms/step - loss: 0.8199 -
accuracy: 0.7395 - val_loss: 0.6112 - val_accuracy: 0.8034
Epoch 3/20
625/625 [==============================] - 32s 51ms/step - loss: 0.4149 -
accuracy: 0.8697 - val_loss: 0.4267 - val_accuracy: 0.8638
Epoch 4/20
625/625 [==============================] - 32s 51ms/step - loss: 0.2519 -
accuracy: 0.9197 - val_loss: 0.4448 - val_accuracy: 0.8678
Epoch 5/20
625/625 [==============================] - 32s 51ms/step - loss: 0.1572 -
accuracy: 0.9491 - val_loss: 0.3723 - val_accuracy: 0.8944
Epoch 6/20
625/625 [==============================] - 33s 52ms/step - loss: 0.0982 -
accuracy: 0.9679 - val_loss: 0.4218 - val_accuracy: 0.8866
Epoch 7/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0619 -
accuracy: 0.9796 - val_loss: 0.4580 - val_accuracy: 0.8830
Epoch 8/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0433 -
accuracy: 0.9872 - val_loss: 0.4184 - val_accuracy: 0.8992
Epoch 9/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0284 -
accuracy: 0.9911 - val_loss: 0.4510 - val_accuracy: 0.8994
Epoch 10/20
625/625 [==============================] - 34s 54ms/step - loss: 0.0171 -
accuracy: 0.9948 - val_loss: 0.4622 - val_accuracy: 0.8984
Epoch 11/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0089 -
```

```
accuracy: 0.9977 - val_loss: 0.5070 - val_accuracy: 0.9016
Epoch 12/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0033 -
accuracy: 0.9994 - val_loss: 0.4999 - val_accuracy: 0.9066
Epoch 13/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0025 -
accuracy: 0.9996 - val_loss: 0.4897 - val_accuracy: 0.9094
Epoch 14/20
625/625 [==============================] - 32s 51ms/step - loss: 4.7297e-04 -
accuracy: 0.9999 - val_loss: 0.5096 - val_accuracy: 0.9114
Epoch 15/20
625/625 [==============================] - 39s 62ms/step - loss: 2.3346e-04 -
accuracy: 1.0000 - val_loss: 0.5218 - val_accuracy: 0.9102
Epoch 16/20
625/625 [==============================] - 34s 54ms/step - loss: 1.8274e-04 -
accuracy: 1.0000 - val_loss: 0.5303 - val_accuracy: 0.9104
Epoch 17/20
625/625 [==============================] - 32s 52ms/step - loss: 1.5414e-04 -
accuracy: 1.0000 - val_loss: 0.5378 - val_accuracy: 0.9110
Epoch 18/20
625/625 [==============================] - 32s 51ms/step - loss: 1.3427e-04 -
accuracy: 1.0000 - val_loss: 0.5436 - val_accuracy: 0.9118
Epoch 19/20
625/625 [==============================] - 32s 51ms/step - loss: 1.1933e-04 -
accuracy: 1.0000 - val_loss: 0.5494 - val_accuracy: 0.9112
Epoch 20/20
625/625 [==============================] - 33s 52ms/step - loss: 1.0769e-04 -
accuracy: 1.0000 - val_loss: 0.5548 - val_accuracy: 0.9110
```


Average Cross-Entropy Error v.s. Epoch
Training
Validation
Average Cross-Entropy Loss
2.0
1.5
1.0
0.5
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs


Average Miss-Classification Error v.s. Epoch)
Training
Validation
Average Miss-Classification Error(%)
70
60
50
40
30
20
10
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

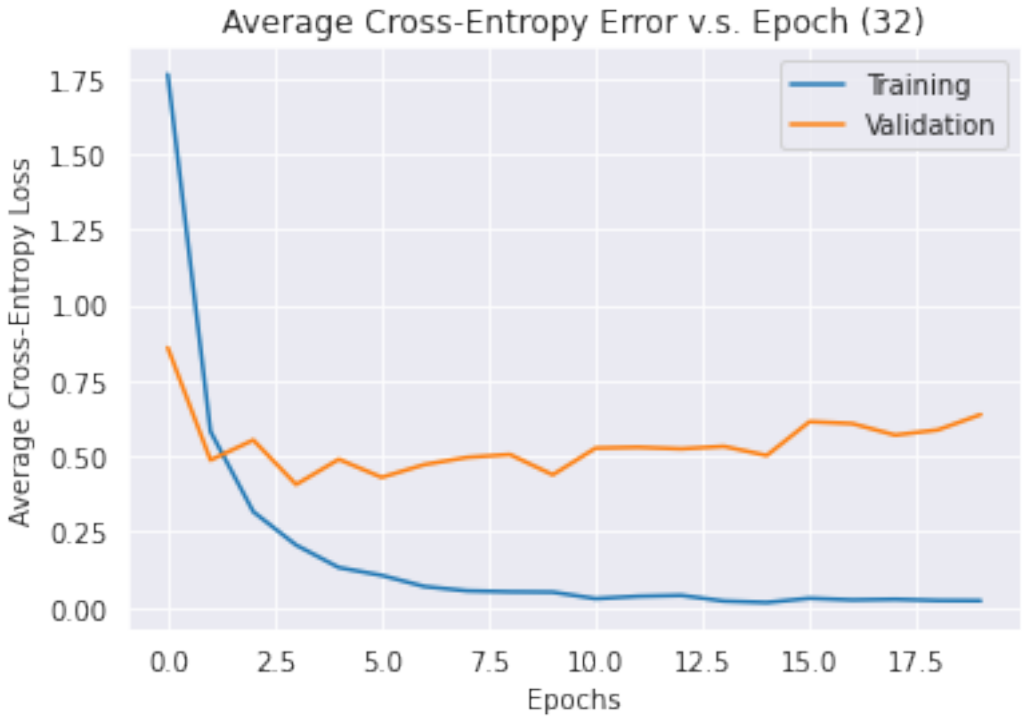
[14]:
```
#Q7(c)
params = model2.layers[0].get_weights()[0].reshape(3,3,-1)
for i in range(params.shape[2]):
    plt.subplot(6, 6, i + 1)
    plt.imshow(params[:,:,i], cmap = "gray", interpolation = "nearest")
    plt.axis("off")
plt.subplots_adjust(wspace=0, hspace=0)
plt.show()
```

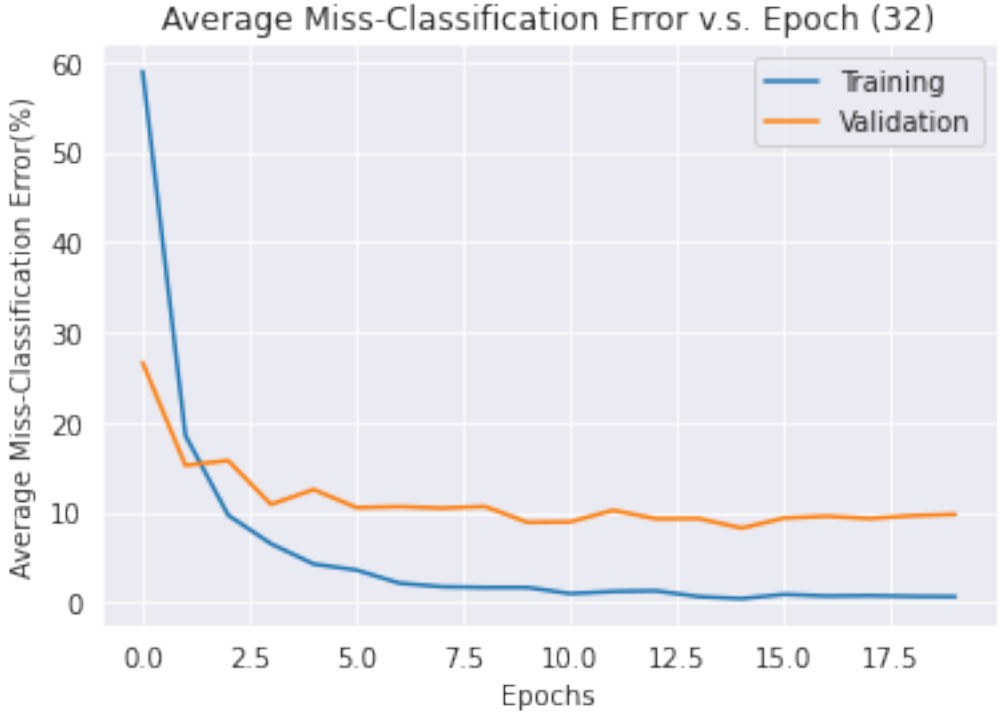
[16]:
```
#Q7(d)
random.seed(0)
sgd = tf.keras.optimizers.SGD(learning_rate=0.1,momentum=0.5)
#create model
model2 = Sequential()
#add model layers
model2.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,56,1)))
model2.add(tf.keras.layers.MaxPooling2D(2,2))
model2.add(Conv2D(32, (3,3), activation='relu'))
model2.add(Flatten())
model2.add(Dense(64, activation='relu'))
model2.add(Dense(19))
#compile model using accuracy to measure model performance
model2.compile(optimizer=sgd, loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
#train the model
history = model2.fit(X_train_reshape, Y_train, validation_data=(X_vali_reshape,␣
 ↪Y_vali), epochs=20)
```

```
res = pd.DataFrame(history.history)
res[['avg_cee_training','avg_cee_validation']] = res[['loss','val_loss']]
res[['avg_mce_training','avg_mce_validation']] =␣
 ↪(1-res[['accuracy','val_accuracy']])*100

plt.title(f"Average Cross-Entropy Error v.s. Epoch")
plt.plot(res[['avg_cee_training']], label="Training")
plt.plot(res[['avg_cee_validation']], label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Average Cross-Entropy Loss")
plt.legend()
plt.show()

plt.title(f"Average Miss-Classification Error v.s. Epoch")
plt.plot(res[['avg_mce_training']], label="Training")
plt.plot(res[['avg_mce_validation']], label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Average Miss-Classification Error(%)")
plt.legend()
plt.show()
```

```
Epoch 1/20
625/625 [==============================] - 33s 52ms/step - loss: 1.7634 -
accuracy: 0.4103 - val_loss: 0.8587 - val_accuracy: 0.7338
Epoch 2/20
625/625 [==============================] - 33s 52ms/step - loss: 0.5811 -
accuracy: 0.8145 - val_loss: 0.4867 - val_accuracy: 0.8478
Epoch 3/20
625/625 [==============================] - 33s 52ms/step - loss: 0.3157 -
accuracy: 0.9031 - val_loss: 0.5530 - val_accuracy: 0.8424
Epoch 4/20
625/625 [==============================] - 33s 52ms/step - loss: 0.2055 -
accuracy: 0.9347 - val_loss: 0.4058 - val_accuracy: 0.8910
Epoch 5/20
625/625 [==============================] - 32s 52ms/step - loss: 0.1311 -
accuracy: 0.9572 - val_loss: 0.4893 - val_accuracy: 0.8746
Epoch 6/20
625/625 [==============================] - 33s 52ms/step - loss: 0.1052 -
accuracy: 0.9639 - val_loss: 0.4295 - val_accuracy: 0.8946
Epoch 7/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0683 -
accuracy: 0.9785 - val_loss: 0.4718 - val_accuracy: 0.8936
Epoch 8/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0539 -
accuracy: 0.9825 - val_loss: 0.4957 - val_accuracy: 0.8952
Epoch 9/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0508 -
accuracy: 0.9833 - val_loss: 0.5050 - val_accuracy: 0.8934
```

```
Epoch 10/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0497 -
accuracy: 0.9833 - val_loss: 0.4378 - val_accuracy: 0.9110
Epoch 11/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0282 -
accuracy: 0.9901 - val_loss: 0.5263 - val_accuracy: 0.9104
Epoch 12/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0361 -
accuracy: 0.9876 - val_loss: 0.5290 - val_accuracy: 0.8976
Epoch 13/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0391 -
accuracy: 0.9870 - val_loss: 0.5239 - val_accuracy: 0.9070
Epoch 14/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0201 -
accuracy: 0.9934 - val_loss: 0.5320 - val_accuracy: 0.9068
Epoch 15/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0155 -
accuracy: 0.9956 - val_loss: 0.5019 - val_accuracy: 0.9174
Epoch 16/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0294 -
accuracy: 0.9909 - val_loss: 0.6138 - val_accuracy: 0.9062
Epoch 17/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0240 -
accuracy: 0.9927 - val_loss: 0.6073 - val_accuracy: 0.9042
Epoch 18/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0260 -
accuracy: 0.9923 - val_loss: 0.5694 - val_accuracy: 0.9068
Epoch 19/20
625/625 [==============================] - 32s 52ms/step - loss: 0.0221 -
accuracy: 0.9931 - val_loss: 0.5865 - val_accuracy: 0.9036
Epoch 20/20
625/625 [==============================] - 32s 51ms/step - loss: 0.0214 -
accuracy: 0.9934 - val_loss: 0.6373 - val_accuracy: 0.9022
```


Average Cross-Entropy Error v.s. Epoch (32)
Training
Validation
Average Cross-Entropy Loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs


Average Miss-Classification Error v.s. Epoch (32)
Training
Validation
Average Miss-Classification Error(%)
60
50
40
30
20
10
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

[17]:
```
#pick best model and on test set
#best model is model2
test = np.argmax(model2.predict(X_test_reshape),axis=1)
test_error = np.mean(test != Y_test)
test_error
```

[17]: 0.0842