In [1]:

```
import os
import numpy as np
import time
import random
import tensorflow as tf
from tensorflow import keras

import matplotlib.pyplot as plt


np.random.seed(1234)
tf.random.set_seed(1234)
```

In [2]:

```
tf.config.list_physical_devices('GPU')
```

```
2022-03-22 14:10:48.680669: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
```

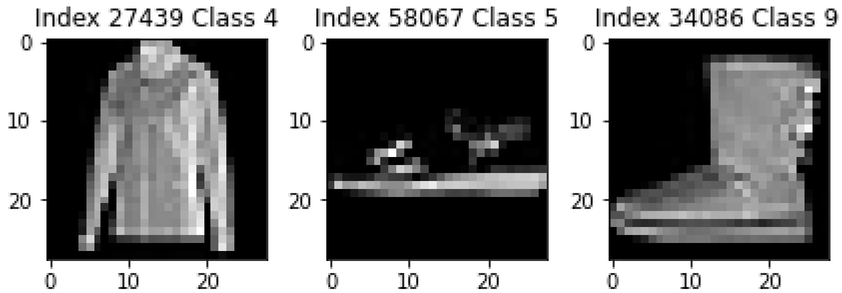
Out[2]:

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
2022-03-22 14:10:48.686086: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-22 14:10:48.686419: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
```

In [3]:

```
(X_train, y_train), (X_test, y_test) = keras.datasets.fashion_mnist.load_data() #
 Load MNIST or FMNIST
assert X_train.shape == (60000, 28, 28)
assert X_test.shape  == (10000, 28, 28)
assert y_train.shape == (60000,)
assert y_test.shape  == (10000,)


# Display randomly selected data
indices = list(np.random.randint(X_train.shape[0],size=3))
for i in range(3):
    plt.subplot(1,3,i+1)
    plt.imshow(X_train[indices[i]].reshape(28,28), cmap='gray', interpolation='non
e')
    plt.title("Index {} Class {}".format(indices[i], y_train[indices[i]]))
    plt.tight_layout()
```



In [4]:

```
# Split train dataset into train and validation
X_val   = X_train[50000:60000]
X_train = X_train[0:50000]
y_val   = y_train[50000:60000]
y_train = y_train[0:50000]

print("size of training set is", str(X_train.shape[0]), "samples")
print("every train example is", str(X_train.shape[1]), "by", str(X_train.shape[2
]))

print("size of validation set is", str(X_val.shape[0]), "samples")
print("every validation example is", str(X_val.shape[1]), "by", str(X_val.shape[2
]))

X_train = X_train.reshape(50000, 28*28)
X_val = X_val.reshape(10000, 28*28)
X_test = X_test.reshape(10000, 28*28)

print("size of training set is", str(X_train.shape[0]), "samples")
print("every train example has", str(X_train.shape[1]), "features")

print("size of validation set is", str(X_val.shape[0]), "samples")
print("every validation example has", str(X_val.shape[1]), "features")

# Split dataset into batches
#train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).batch(16)
#test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(4)
```

```
size of training set is 50000 samples
every train example is 28 by 28
size of validation set is 10000 samples
every validation example is 28 by 28
size of training set is 50000 samples
every train example has 784 features
size of validation set is 10000 samples
every validation example has 784 features
```

In [5]:

```
#Normalize Data

X_train = X_train/255
X_val   = X_val/255
X_test  = X_test/255

print(f'max: ', np.max(X_train))
print(f'min: ', np.min(X_train))
```

```
max:  1.0
min:  0.0
```

In [6]:

```
size_input = X_train.shape[1]

size_hidden1 = 128
size_hidden2 = 128
size_hidden3 = 128

size_output = 10

number_of_train_examples = X_train.shape[0]
number_of_test_examples  = X_test.shape[0]

y_train = tf.keras.utils.to_categorical(y_train, num_classes=10) # Other function
 is tf.one_hot(y_train,depth=10)
y_val   = tf.keras.utils.to_categorical(y_val, num_classes=10)
y_test  = tf.keras.utils.to_categorical(y_test, num_classes=10)

print(tf.shape(y_val))
```

```
2022-03-22 14:10:49.654034: I tensorflow/core/platform/cpu_feature_gua
rd.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the approp
riate compiler flags.
```

```
tf.Tensor([10000    10], shape=(2,), dtype=int32)
```

```
2022-03-22 14:10:49.654925: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-22 14:10:49.656209: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-22 14:10:49.657047: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-22 14:10:50.023539: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-22 14:10:50.023846: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-22 14:10:50.024099: I tensorflow/stream_executor/cuda/cuda_gpu
_executor.cc:936] successful NUMA node read from SysFS had negative va
lue (-1), but there must be at least one NUMA node, so returning NUMA
node zero
2022-03-22 14:10:50.024345: I tensorflow/core/common_runtime/gpu/gpu_d
evice.cc:1525] Created device /job:localhost/replica:0/task:0/device:G
PU:0 with 6925 MB memory:  -> device: 0, name: NVIDIA GeForce GTX 107
0, pci bus id: 0000:01:00.0, compute capability: 6.1
```

In [7]:

```
# Define class to build mlp model
class MLP(object):
    def __init__(self, size_input, size_hidden1, size_hidden2, size_hidden3, size_
output, device=None):
        """
        size_input: int, size of input layer
        size_hidden1: int, size of the 1st hidden layer
        size_hidden2: int, size of the 2nd hidden layer
        size_output: int, size of output layer
        device: str or None, either 'cpu' or 'gpu' or None. If None, the device to
be used will be decided automatically during Eager Execution
        """
        self.size_input, self.size_hidden1, self.size_hidden2, self.size_hidden3,
self.size_output, self.device =\
        size_input, size_hidden1, size_hidden2, size_hidden3, size_output, device

        # Initialize weights between input mapping and a layer g(f(x)) = layer
        self.W1 = tf.Variable(tf.random.normal([self.size_input, self.size_hidden1
],stddev=0.1)) # Xavier(Fan-in fan-out) and Orthogonal
        # Initialize biases for hidden layer
        self.b1 = tf.Variable(tf.zeros([1, self.size_hidden1])) # 0 or constant(0.
01)

        # Initialize weights between input layer and 1st hidden layer
        self.W2 = tf.Variable(tf.random.normal([self.size_hidden1, self.size_hidde
n2],stddev=0.1))
        # Initialize biases for hidden layer
        self.b2 = tf.Variable(tf.zeros([1, self.size_hidden2]))

        # Initialize weights between 1st hidden layer and 2nd hidden layer
        self.W3 = tf.Variable(tf.random.normal([self.size_hidden2, self.size_hidde
n3],stddev=0.1))
        # Initialize biases for hidden layer
        self.b3 = tf.Variable(tf.zeros([1, self.size_hidden3]))

         # Initialize weights between 2nd hidden layer and output layer
        self.W4 = tf.Variable(tf.random.normal([self.size_hidden3, self.size_outpu
t],stddev=0.1))
        # Initialize biases for output layer
        self.b4 = tf.Variable(tf.zeros([1, self.size_output]))

        # Initializing gamma and beta for all the layers for batch norm
        self.gamma = {"1": tf.Variable(tf.ones(self.W1.shape[-1])),
                      "2": tf.Variable(tf.ones(self.W2.shape[-1])),
                      "3": tf.Variable(tf.ones(self.W3.shape[-1]))
                      #"4": tf.Variable(tf.ones(self.W4.shape[-1]))
        }
        self.beta  = {"1": tf.Variable(tf.zeros(self.W1.shape[-1])),
                      "2": tf.Variable(tf.zeros(self.W2.shape[-1])),
                      "3": tf.Variable(tf.zeros(self.W3.shape[-1]))
                      #"4": tf.Variable(tf.zeros(self.W4.shape[-1]))
        }
        # Define variables to be updated during backpropagation
```

```
        self.variables = [self.W1, self.W2, self.W3, self.W4,
                          self.b1, self.b2, self.b3, self.b4,
                          self.gamma["1"], self.beta["1"],
                          self.gamma["2"], self.beta["2"],
                          self.gamma["3"], self.beta["3"]]
                          #self.gamma["4"], self.beta["4"]]

        self.mean = {"1": tf.Variable(tf.zeros(self.W1.shape[-1])),
                     "2": tf.Variable(tf.zeros(self.W2.shape[-1])),
                     "3": tf.Variable(tf.zeros(self.W3.shape[-1]))
                     #"4": tf.Variable(tf.zeros(self.W4.shape[-1])),
        }
        self.var = {"1": tf.Variable(tf.zeros(self.W1.shape[-1])),
                    "2": tf.Variable(tf.zeros(self.W2.shape[-1])),
                    "3": tf.Variable(tf.zeros(self.W3.shape[-1]))
                    #"4": tf.Variable(tf.zeros(self.W4.shape[-1])),
        }

    def forward(self, X, run):
        """
        forward pass
        X: Tensor, inputs
        """
        if self.device is not None:
            with tf.device('gpu:0' if self.device=='gpu' else 'cpu'):
                if run == "train":
                    self.y = self.compute_output_train(X)
                else:
                    self.y = self.compute_output_test(X)
        else:
            if run == "train":
                self.y = self.compute_output_train(X)
            else:
                self.y = self.compute_output_test(X)

        return self.y

    def loss(self, y_pred, y_true):
        '''
        y_pred - Tensor of shape (batch_size, size_output)
        y_true - Tensor of shape (batch_size, size_output)
        '''
        #y_true_tf = tf.cast(tf.reshape(y_true, (-1, self.size_output)), dtype=tf.
float32)
        y_true_tf = tf.cast(y_true, dtype=tf.float32)
        y_pred_tf = tf.cast(y_pred, dtype=tf.float32)
        cce = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
        loss_x = cce(y_true_tf, y_pred_tf)
        # Use keras or tf_softmax, both should work for any given model
        #loss_x = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_
pred_tf, labels=y_true_tf))

        return loss_x

    def backward(self, X_train, y_train, opti):
        """
```

```
        backward pass
        """
        optimizer = opti

        with tf.GradientTape() as tape:
            predicted = self.forward(X_train, "train")
            current_loss = self.loss(predicted, y_train)

        grads = tape.gradient(current_loss, self.variables)
        optimizer.apply_gradients(zip(grads, self.variables))

    def batch_norm(self, inputs, layer):
        mean, var = tf.nn.moments(inputs, [0])
        self.mean[layer] = 0.9*self.mean[layer] + 0.1*mean
        self.var[layer]  = 0.9*self.var[layer]  + 0.1*var

        return tf.add(tf.divide(tf.multiply(self.gamma[layer], tf.subtract(inputs,
mean)), tf.sqrt(var+1e-8)), self.beta[layer])

    def compute_output_train(self, X):
        """
        Custom method to obtain output tensor during forward pass
        """
        # Cast X to float32
        X_tf = tf.cast(X, dtype=tf.float32)
        #X_tf = X

        # Compute values in hidden layers
        z1 = tf.matmul(X_tf, self.W1) + self.b1
        h1 = tf.nn.relu(z1)
        h1 = self.batch_norm(h1, "1")

        z2 = tf.matmul(h1, self.W2) + self.b2
        h2 = tf.nn.relu(z2)
        h2 = self.batch_norm(h2, "2")

        z3 = tf.matmul(h2, self.W3) + self.b3
        h3 = tf.nn.relu(z3)
        h3 = self.batch_norm(h3, "3")

        # Compute output
        output = tf.matmul(h3, self.W4) + self.b4
        #output = self.batch_norm(output, "4")
        #Now consider two things , First look at inbuild loss functions if they wo
rk with softmax or not and then change this
        # Second add tf.Softmax(output) and then return this variable
        return (output)

    def compute_output_test(self, X):
        """
        Custom method to obtain output tensor during forward pass
        """
        # Cast X to float32
        X_tf = tf.cast(X, dtype=tf.float32)
        #X_tf = X
```

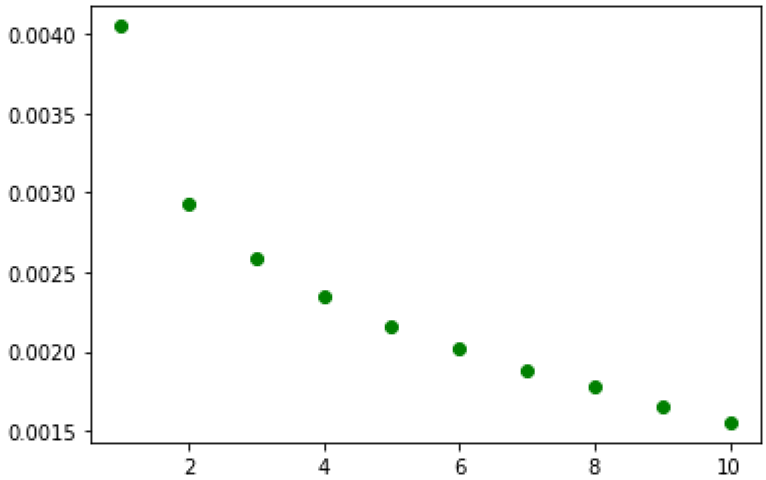
```
        # Compute values in hidden layers
        z1 = tf.matmul(X_tf, self.W1) + self.b1
        h1 = tf.nn.relu(z1)
        h1 = tf.add(tf.divide(tf.multiply(self.gamma["1"], tf.subtract(h1, self.me
an["1"])), tf.sqrt(self.var["1"]+1e-8)), self.beta["1"])

        z2 = tf.matmul(h1, self.W2) + self.b2
        h2 = tf.nn.relu(z2)
        h2 = tf.add(tf.divide(tf.multiply(self.gamma["2"], tf.subtract(h2, self.me
an["2"])), tf.sqrt(self.var["2"]+1e-8)), self.beta["2"])

        z3 = tf.matmul(h2, self.W3) + self.b3
        h3 = tf.nn.relu(z3)
        h3 = tf.add(tf.divide(tf.multiply(self.gamma["3"], tf.subtract(h3, self.me
an["3"])), tf.sqrt(self.var["3"]+1e-8)), self.beta["3"])

        # Compute output
        output = tf.matmul(h3, self.W4) + self.b4

        #Now consider two things , First look at inbuild loss functions if they wo
rk with softmax or not and then change this
        # Second add tf.Softmax(output) and then return this variable
        return (output)
```

In [8]:

```
numTrials = 3
seeds = random.sample(range(1000, 9999), numTrials)
for trail in range(numTrials):

    seed = seeds[trail]

    print(f'\n\n***************** Running MLP with post-activation Batch Normaliza
tion for seed {seed} *****************\n')
    # Set number of epochs
    NUM_EPOCHS = 10

    # Initialize model using CPU
    mlp_on_gpu = MLP(size_input, size_hidden1, size_hidden2, size_hidden3, size_ou
tput, device='gpu')

    time_start = time.time()
    opti = tf.keras.optimizers.SGD(learning_rate = 0.1)

    for epoch in range(NUM_EPOCHS):
        loss_total = tf.zeros([1,1], dtype=tf.float32)
        lt = 0

        train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).shuffle(
25, seed=epoch*(seed)).batch(128)
        kz = 0
        accuracy_z = 0.0
        cur_train_acc = 0.0
        for inputs, outputs in train_ds:
            qw, tr = tf.shape(inputs)
            kz = kz + 1
            preds = mlp_on_gpu.forward(inputs, "train")
            loss_total = loss_total + mlp_on_gpu.loss(preds, outputs)
            lt = lt + mlp_on_gpu.loss(preds, outputs)
            mlp_on_gpu.backward(inputs, outputs, opti)

        preds = mlp_on_gpu.forward(X_train, "train")
        # Get probs, remember we only have logits from our forward function, we ne
ed to apply softmax on top of it to get probs
        preds = tf.nn.softmax(preds)
        correct_prediction = tf.equal(tf.argmax(preds, 1), tf.argmax(y_train, 1))
        accuracy_z = accuracy_z + tf.reduce_mean(tf.cast(correct_prediction, "floa
t"))
        cur_train_acc += accuracy_z.numpy()
        ds = cur_train_acc
        print('\nTrain Accuracy: {:.4f}'.format(ds))
        print('Number of Epoch = {} - Average Cross Entropy:= {} '.format(epoch +
1, np.sum(loss_total) / X_train.shape[0]))
        preds_val = mlp_on_gpu.forward(X_val, "train")
        preds_val = tf.nn.softmax(preds_val)
        correct_prediction = tf.equal(tf.argmax(preds_val, 1), tf.argmax(y_val, 1
))

        # Calculate accuracy
```

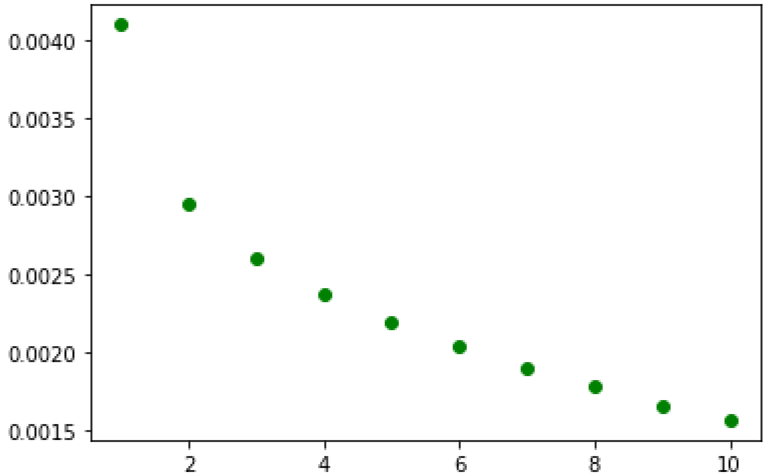
```
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
        cur_val_acc = accuracy.numpy()

        print('\nValidation Accuracy: {:.4f}'.format(cur_val_acc))

        plt.plot(epoch + 1, np.sum(loss_total) / X_train.shape[0], 'go')


    time_taken = time.time() - time_start
    plt.show()
    # Validate model



    print('\nTotal time taken (in seconds): {:.2f}'.format(time_taken))
    #For per epoch_time = Total_Time / Number_of_epochs

    test_loss_total = tf.Variable(0, dtype=tf.float32)
    correct_prediction = tf.Variable(0, dtype=tf.float32)


    test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(24)


    #test_loss_total = 0.0
    for inputs, outputs in test_ds:
        preds = mlp_on_gpu.forward(inputs, "test")
        test_loss_total = test_loss_total + mlp_on_gpu.loss(preds, outputs)
    print('Test loss: {:.4f}'.format(np.sum(test_loss_total.numpy()) / X_test.shap
e[0]))

    # Test model
    preds_test = mlp_on_gpu.forward(X_test, "test")
    preds_test = tf.nn.softmax(preds_test)
    correct_prediction = tf.equal(tf.argmax(preds_test, 1), tf.argmax(y_test, 1))

    # Calculate accuracy
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    cur_test_acc = accuracy.numpy()
    print('\nTest Accuracy: {:.2f}'.format(cur_test_acc))
```

```
***************** Running MLP with post-activation Batch Normalization
for seed 6697 *****************


Train Accuracy: 0.8638
Number of Epoch = 1 - Average Cross Entropy:= 0.004050693664550781

Validation Accuracy: 0.8488

Train Accuracy: 0.8804
Number of Epoch = 2 - Average Cross Entropy:= 0.0029234954833984375

Validation Accuracy: 0.8655

Train Accuracy: 0.8915
Number of Epoch = 3 - Average Cross Entropy:= 0.0025858731079101563

Validation Accuracy: 0.8717

Train Accuracy: 0.8992
Number of Epoch = 4 - Average Cross Entropy:= 0.002349191131591797

Validation Accuracy: 0.8737

Train Accuracy: 0.9070
Number of Epoch = 5 - Average Cross Entropy:= 0.002153983154296875

Validation Accuracy: 0.8787

Train Accuracy: 0.9127
Number of Epoch = 6 - Average Cross Entropy:= 0.0020170329284667967

Validation Accuracy: 0.8767

Train Accuracy: 0.9175
Number of Epoch = 7 - Average Cross Entropy:= 0.0018850112915039063

Validation Accuracy: 0.8783

Train Accuracy: 0.9224
Number of Epoch = 8 - Average Cross Entropy:= 0.0017811859130859376

Validation Accuracy: 0.8764

Train Accuracy: 0.9273
Number of Epoch = 9 - Average Cross Entropy:= 0.0016485203552246094

Validation Accuracy: 0.8781

Train Accuracy: 0.9266
Number of Epoch = 10 - Average Cross Entropy:= 0.0015529104614257813

Validation Accuracy: 0.8768
```


0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
2
4
6
8
10

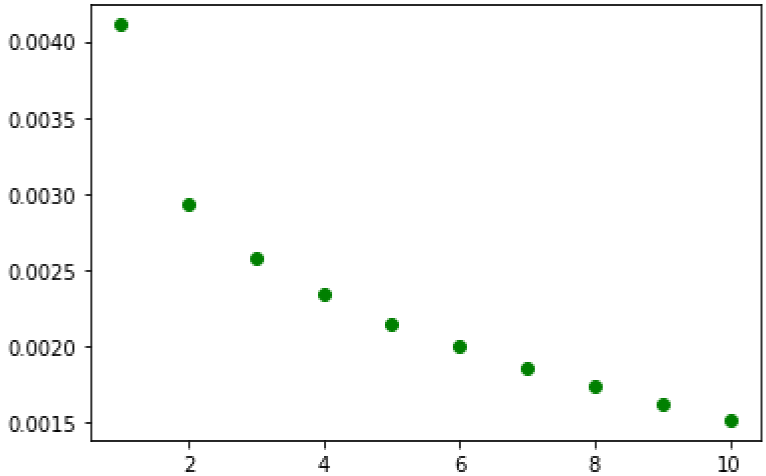
```
Total time taken (in seconds): 157.46
Test loss: 0.0171

Test Accuracy: 0.86


****************** Running MLP with post-activation Batch Normalization
for seed 1888 *****************


Train Accuracy: 0.8619
Number of Epoch = 1 - Average Cross Entropy:= 0.004101782836914062

Validation Accuracy: 0.8501

Train Accuracy: 0.8810
Number of Epoch = 2 - Average Cross Entropy:= 0.00294813720703125

Validation Accuracy: 0.8643

Train Accuracy: 0.8911
Number of Epoch = 3 - Average Cross Entropy:= 0.002599829406738281

Validation Accuracy: 0.8710

Train Accuracy: 0.9006
Number of Epoch = 4 - Average Cross Entropy:= 0.0023720100402832033

Validation Accuracy: 0.8765

Train Accuracy: 0.9068
Number of Epoch = 5 - Average Cross Entropy:= 0.002190786895751953

Validation Accuracy: 0.8764

Train Accuracy: 0.9142
Number of Epoch = 6 - Average Cross Entropy:= 0.0020375729370117187

Validation Accuracy: 0.8761

Train Accuracy: 0.9196
Number of Epoch = 7 - Average Cross Entropy:= 0.001903392333984375

Validation Accuracy: 0.8782

Train Accuracy: 0.9222
Number of Epoch = 8 - Average Cross Entropy:= 0.0017770565795898437

Validation Accuracy: 0.8769

Train Accuracy: 0.9268
Number of Epoch = 9 - Average Cross Entropy:= 0.0016511749267578125

Validation Accuracy: 0.8777
```

```
Train Accuracy: 0.9275
Number of Epoch = 10 - Average Cross Entropy:= 0.0015646278381347657

Validation Accuracy: 0.8780
```



```
Total time taken (in seconds): 149.71
Test loss: 0.0164

Test Accuracy: 0.87


***************** Running MLP with post-activation Batch Normalization
for seed 8760 *****************


Train Accuracy: 0.8640
Number of Epoch = 1 - Average Cross Entropy:= 0.004108681640625

Validation Accuracy: 0.8498

Train Accuracy: 0.8813
Number of Epoch = 2 - Average Cross Entropy:= 0.0029310879516601562

Validation Accuracy: 0.8655

Train Accuracy: 0.8957
Number of Epoch = 3 - Average Cross Entropy:= 0.0025746478271484376

Validation Accuracy: 0.8714

Train Accuracy: 0.9031
Number of Epoch = 4 - Average Cross Entropy:= 0.0023422163391113283

Validation Accuracy: 0.8753

Train Accuracy: 0.9084
Number of Epoch = 5 - Average Cross Entropy:= 0.002151478729248047

Validation Accuracy: 0.8774

Train Accuracy: 0.9136
Number of Epoch = 6 - Average Cross Entropy:= 0.0019973666381835936

Validation Accuracy: 0.8785

Train Accuracy: 0.9186
Number of Epoch = 7 - Average Cross Entropy:= 0.0018560171508789063

Validation Accuracy: 0.8801

Train Accuracy: 0.9253
Number of Epoch = 8 - Average Cross Entropy:= 0.001741551055908203

Validation Accuracy: 0.8797

Train Accuracy: 0.9259
Number of Epoch = 9 - Average Cross Entropy:= 0.0016267584228515624

Validation Accuracy: 0.8805
```

```
Train Accuracy: 0.9297
Number of Epoch = 10 - Average Cross Entropy:= 0.0015190045166015625

Validation Accuracy: 0.8816
```




```
Total time taken (in seconds): 149.31
Test loss: 0.0181

Test Accuracy: 0.87
```

In [ ]: