Himani Parikh – 1322085

Deep Learning Homework 2 📚 💻 (50 points)

TensorFlow, Keras, NumPy

In this problem, you are asked to train and test a fully- connected neural network for entire MNIST handwritten digit dataset. Some information of the network is as follows:

– Its structure is 784-200-50-10. Here 784 means the input layer has 784 input neurons. This is because each image in MNIST dataset is 28x28 and you need to stretch them to a length-784 vector. 200 and 50 are the number of neurons in hidden layers. 10 is the number of neurons in output layer since there are 10 types of digits.
– The two hidden layers are followed by ReLU layers
– The output layer is a softmax layer

- Use deep learning framework (Pytorch or Tensorflow) to train and test this network. You are allowed to use the corresponding autograd or nn module to train the network.

- (Optional - maybe a good choice for your project) Use only Numpy (which can be used to load the data MNIST) or Keras to train and test this network. You are NOT allowed to use deep learning framework (e.g. Pytorch, Tensorflow etc.) and the corresponding autograd or nn module to train the network.

- Performance Requirement and Submission:
  – The test accuracy should achieve above 95%
  – Submission should include your source codes and screen snapshot of your train and test accuracy, plus the training time
  Suggestion for hyperparameter setting (not necessary to follow):
  – Learning rate can be set as 0.01
  – If you choose to use mini-batch SGD, the batch size can be set as 128 – Number of epochs can be set as 10.

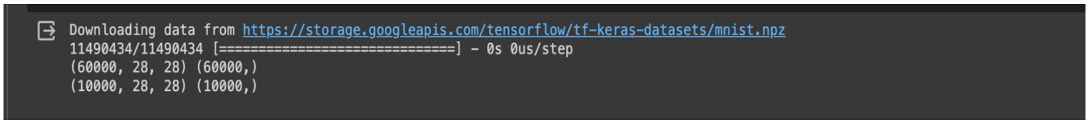
Code and Output:

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

#*Load the minst dataset*:

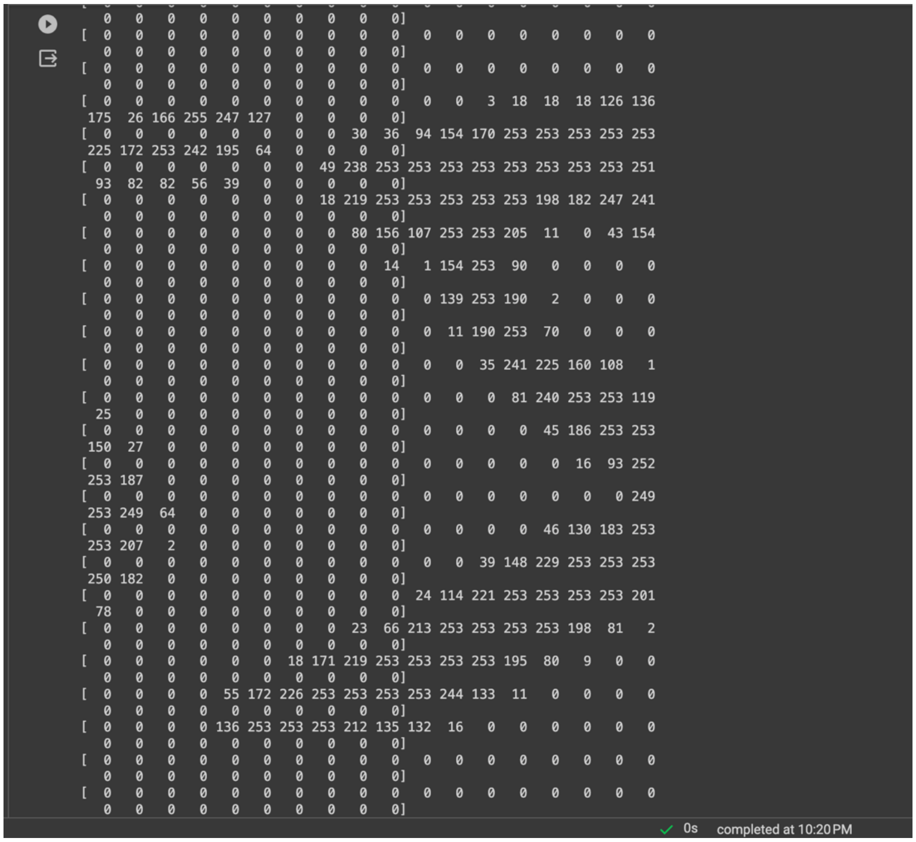
minst_dataset = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test)= minst_dataset.load_data()

print(x_train.shape , y_train.shape)
print(x_test.shape, y_test.shape)



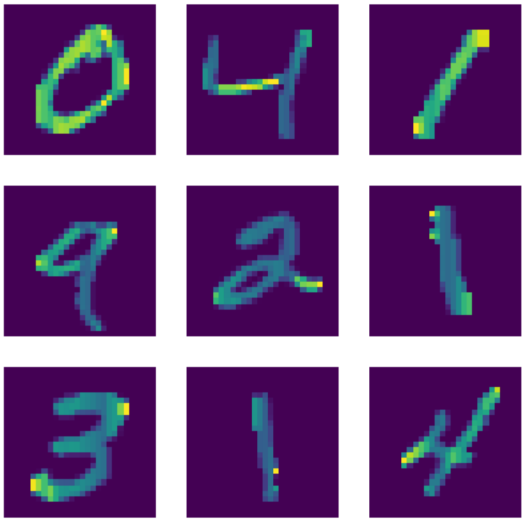
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

print(x_train[0])

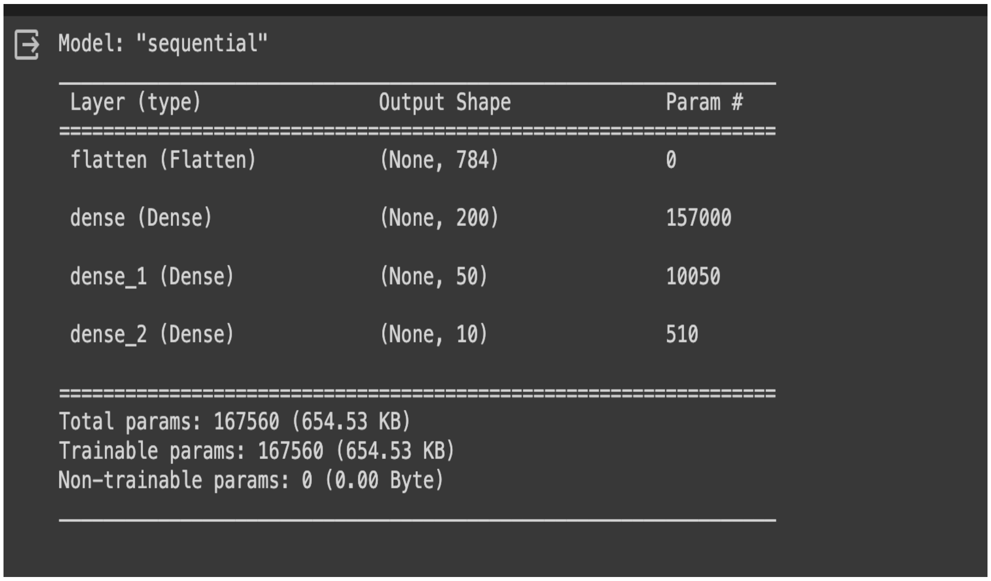


```
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
   78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
```

```
#Data Pre-Processing : Normalize(Min-Max scaling)

x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)

plt.figure(figsize=(5,5))
for i in range(1, 10):
  plt.subplot(3, 3, i)
  plt.axis('off')
  plt.imshow(x_train[i].squeeze())
```



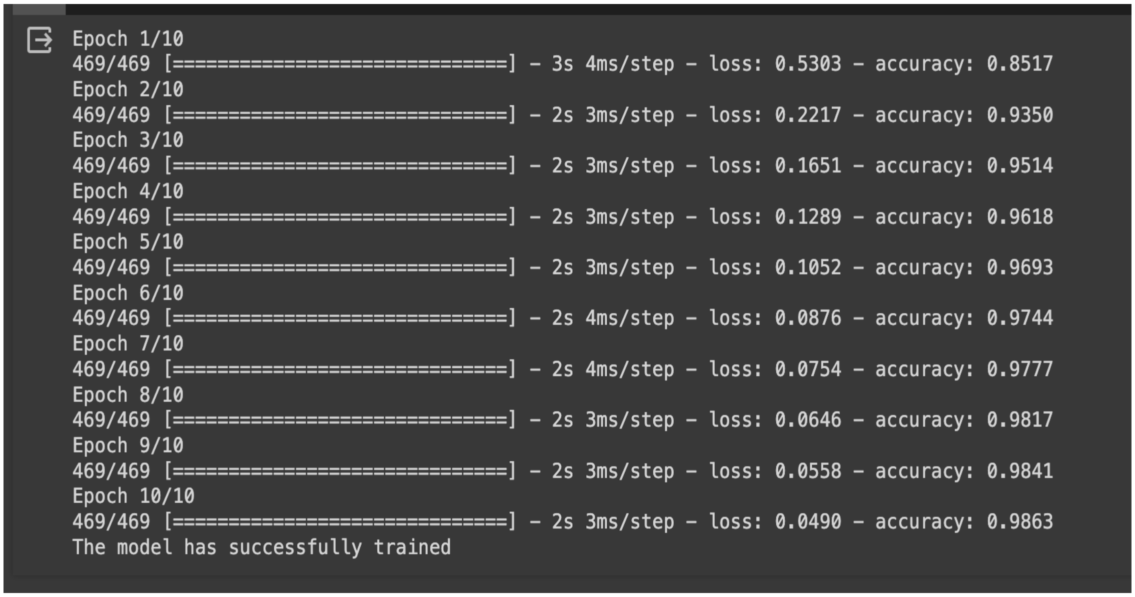
```
#Create Fully connected with 2 hidden layers network model

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
model.add(tf.keras.layers.Dense(200, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(50, activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.02, momentum=0.9),
          loss=tf.keras.losses.sparse_categorical_crossentropy,
          metrics=['accuracy'])

model.summary()
```



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 200)               157000

 dense_1 (Dense)             (None, 50)                10050

 dense_2 (Dense)             (None, 10)                510

=================================================================
Total params: 167560 (654.53 KB)
Trainable params: 167560 (654.53 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

<u>#Training Model</u>

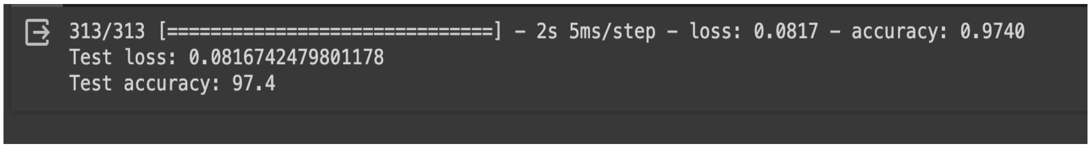
```
model.fit(x_train, y_train, batch_size = 128, epochs=10)
print("The model has successfully trained")

model.save('sample.model')
```



```
Epoch 1/10
469/469 [==============================] - 3s 4ms/step - loss: 0.5303 - accuracy: 0.8517
Epoch 2/10
469/469 [==============================] - 2s 3ms/step - loss: 0.2217 - accuracy: 0.9350
Epoch 3/10
469/469 [==============================] - 2s 3ms/step - loss: 0.1651 - accuracy: 0.9514
Epoch 4/10
469/469 [==============================] - 2s 3ms/step - loss: 0.1289 - accuracy: 0.9618
Epoch 5/10
469/469 [==============================] - 2s 3ms/step - loss: 0.1052 - accuracy: 0.9693
Epoch 6/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0876 - accuracy: 0.9744
Epoch 7/10
469/469 [==============================] - 2s 4ms/step - loss: 0.0754 - accuracy: 0.9777
Epoch 8/10
469/469 [==============================] - 2s 3ms/step - loss: 0.0646 - accuracy: 0.9817
Epoch 9/10
469/469 [==============================] - 2s 3ms/step - loss: 0.0558 - accuracy: 0.9841
Epoch 10/10
469/469 [==============================] - 2s 3ms/step - loss: 0.0490 - accuracy: 0.9863
The model has successfully trained
```

#Model Evaluation

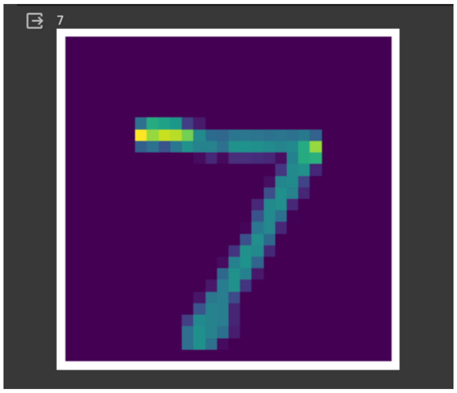
```
s_model = tf.keras.models.load_model('sample.model')
loss, accuracy = s_model.evaluate(x_test, y_test)
print('Test loss:', loss)
print('Test accuracy:', np.round((accuracy)*100, 2))
```



```
313/313 [==============================] - 2s 5ms/step - loss: 0.0817 - accuracy: 0.9740
Test loss: 0.0816742479801178
Test accuracy: 97.4
```

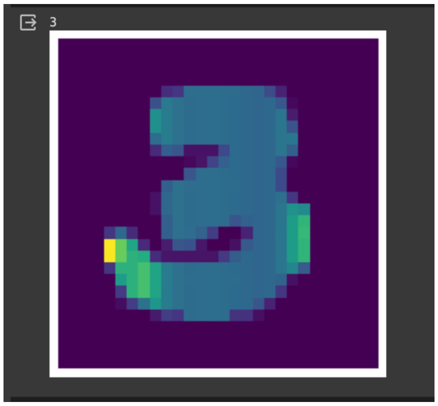
#Testing Model

```
y_predicted = model.predict(x_test)

plt.imshow(x_test[0].squeeze())
plt.axis('off')
print(np.argmax(y_predicted[0]))
```



```
7
```

```
plt.imshow(x_test[200].squeeze())
plt.axis('off')
print(np.argmax(y_predicted[200]))
```



```
3
```