Assignment 2 Scott Peters 10102681

Part 1:

Our model begins by inputting the image of a handwritten digit as x, and the value associated with the digit in y, which we will be using to train our model. We divide the x values by 255.0 to get the images to black and white and remove any discrepancies. Following that, we begin to structure our model using different layers.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(300, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

Since each image is a 28x28 array, flatten is used to turn this into a 1-D tensor of size 784. Relu returns 0 if there is a negative but returns the value if it is positive. This dense layer has 300 units, which tells us that the size of the output is 300 neurons (from what I understand?). This is then followed by a 10 unit softmax dense layer which makes a connection from every image spot to every 0-9 integer output spot. The reason I added the relu layer is because this added a new hidden layer to my model before using the output softmax layer, which helped the model train itself better. I guessed and tested the number of units for the relu layer and settled on 300 being a good number. I cannot perfectly explain why this number worked best, but seemingly having a highe number of neurons in the first layer, and using softmax on that worked best.

```
initial_learning_rate = 1e-2
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
  initial_learning_rate, decay_steps=100000, decay_rate=0.9, staircase=True)

optimizer = tf.keras.optimizers.Adamax(learning_rate=lr_schedule)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

I edited the optimizer to be on a schedule with Adamax instead of the previously used sgd. This schedule allows me to start with a learning rate of 1e-2, and continuously reduce it over time to find the optimized learning rate. Using Adamax vs SGD was more of just guessing and testing than anything. I believe the reason this works better is because (from what I can tell) adamax combines the good features of SGD but eliminates a lot of the downsides, since it calculates learning rates for different parameters. This also pairs nicely with the learning rate schedule, as that also would calculate different learning rates. We use sparce categorical cross entropy because we need to use categorical cross entropy since we are trying to categorize the image into one of the 10 different integers. We are iterating over each different integer and seeing what gives us the closest value, which it calculates using a logarithmic function. The sparse part of this loss function is due to our dataset using integer targets instead of categorical vectors.

```
print("--Fit model--")
model.fit(x_train, y_train, epochs=15, verbose=2)
```

I then changed the epochs to 15, which allowed me to get an end accuracy of 100% on train and 98.3% on test.

```
Epoch 1/15
1875/1875 - 9s - loss: 0.1883 - accuracy: 0.9435 - 9s/epoch - 5ms/step
Epoch 2/15
1875/1875 - 8s - loss: 0.0807 - accuracy: 0.9750 - 8s/epoch - 4ms/step
Epoch 3/15
1875/1875 - 7s - loss: 0.0534 - accuracy: 0.9837 - 7s/epoch - 4ms/step
Epoch 4/15
1875/1875 - 7s - loss: 0.0367 - accuracy: 0.9884 - 7s/epoch - 4ms/step
Epoch 5/15
1875/1875 - 8s - loss: 0.0257 - accuracy: 0.9914 - 8s/epoch - 4ms/step
Epoch 6/15
1875/1875 - 8s - loss: 0.0191 - accuracy: 0.9939 - 8s/epoch - 4ms/step
Epoch 7/15
1875/1875 - 8s - loss: 0.0133 - accuracy: 0.9961 - 8s/epoch - 4ms/step
Epoch 8/15
1875/1875 - 7s - loss: 0.0094 - accuracy: 0.9972 - 7s/epoch - 4ms/step
Epoch 9/15
1875/1875 - 8s - loss: 0.0064 - accuracy: 0.9983 - 8s/epoch - 4ms/step
Epoch 10/15
1875/1875 - 8s - loss: 0.0055 - accuracy: 0.9984 - 8s/epoch - 4ms/step
Epoch 11/15
1875/1875 - 8s - loss: 0.0037 - accuracy: 0.9988 - 8s/epoch - 4ms/step
Epoch 12/15
1875/1875 - 8s - loss: 0.0031 - accuracy: 0.9991 - 8s/epoch - 4ms/step
Epoch 13/15
1875/1875 - 8s - loss: 0.0020 - accuracy: 0.9996 - 8s/epoch - 4ms/step
Epoch 14/15
1875/1875 - 7s - loss: 0.0015 - accuracy: 0.9997 - 7s/epoch - 4ms/step
Epoch 15/15
1875/1875 - 8s - loss: 0.0015 - accuracy: 0.9997 - 8s/epoch - 4ms/step
--Evaluate model--
1875/1875 - 5s - loss: 6.5545e-04 - accuracy: 0.9998 - 5s/epoch - 2ms/step
313/313 - 1s - loss: 0.0889 - accuracy: 0.9829 - 882ms/epoch - 3ms/step
Train / Test Accuracy: 100.0% / 98.3%
```

The reason for the 15 epochs was nothing more than guessing and testing. Using only 5 or 10 epochs didn't give me a good enough accuracy and going upwards of 50 took way too long and the accuracy did not increase very much. This amount of epochs doesn't take all that long to complete while also generating a high test accuracy.