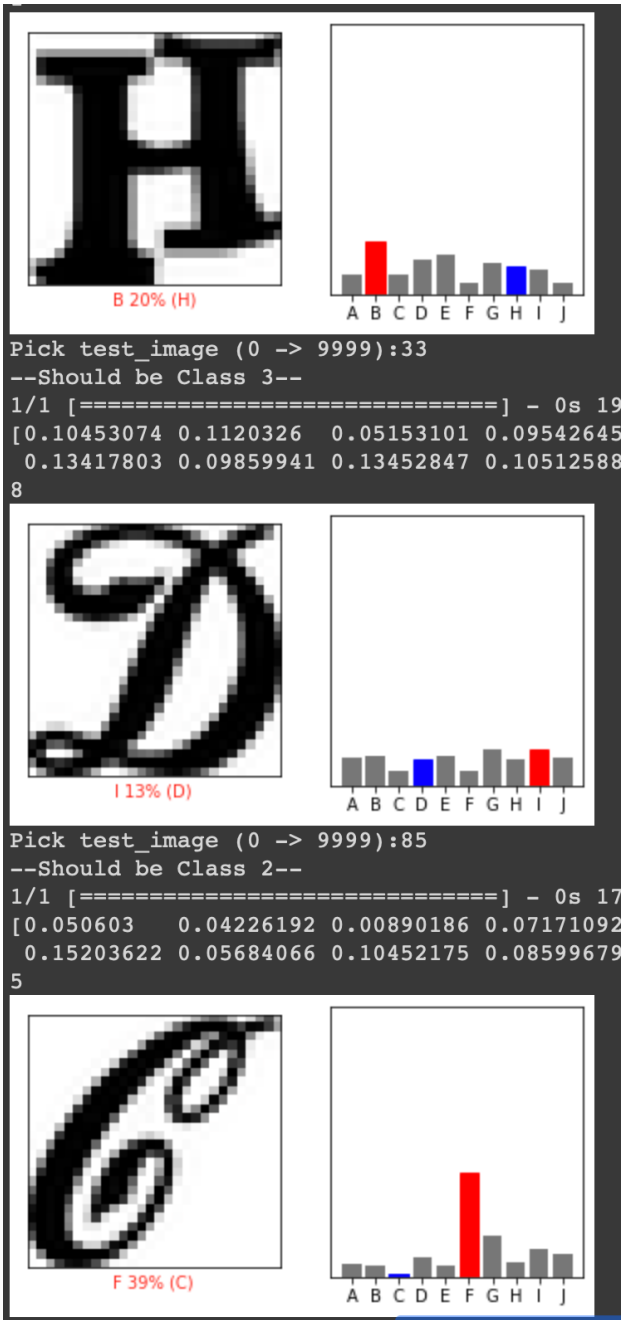
Part 2 Report Scott Peters 10102681

My partial model has a training accuracy of 84.5%, and a test accuracy of 88.8%. I used a single relu dense layer with 300 units, as well as added a learning rate schedule using Adam optimization. It begins with a learning rate of 0.01, and a decay rate of 0.9. I also used sparse categorical cross entropy, taking this from my previous MNIST model. This all allowed my model to work pretty well (almost 90%), however I can do a lot better.

With this partial model, I was able to find images 16, 33, and 85 which were not categorized correctly. 16 should be classified as an H, 33 should be a D and 85 should be a C.



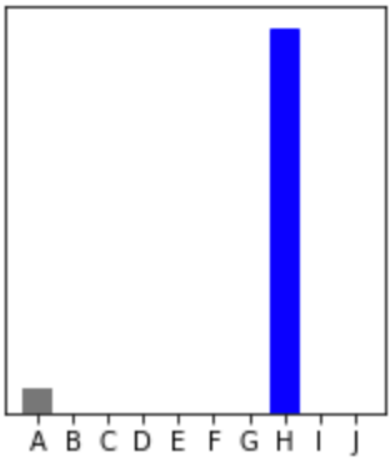
I then went back to my original model, and added a number of things to hopefully improve my performance.

```
print("--Make model--")
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

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'])

print("--Fit model--")
model.fit(x_train, y_train, epochs=25, verbose=2, batch_size = 50)
```

I edited my first dense layer to have 512 units, which enabled me to add another hidden layer, with relu activation and 256 units. This gave my model two hidden layers that help make connections between the neural network and the images of handwriting. I also added a dropout of 0.2 after each relu layer, which helps prevent overfitting. I also changed my optimizer to Adamax, which seems to have worked A LOT better. When running my changes with Adam, I get a test accuracy of 90.6 vs the 94% I get with Adamax. If I am being honest, when trying to research why this is I didn't fully grasp the concepts that I was reading about. Keras documentation tells me that Adamax is a variant of Adam based on the infinity norm, and that it can be superior to Adam specifically in cases with embedding. As far as I can tell I never used embedding, so I am still a bit confused as to why this worked so much better, the reason I ended up trying Adamax was just simple guessing and testing. I also added a batch_size when I was fitting my model. This was just to make my model require less memory and increase the speed of the epochs; however (I believe?) this also can help with our model. I did some research, and it seems that this also updates our weights after each batch is done, which can aid our model. This was proven when I tested a bunch of different batch sizes, and when they were too big or too small it negatively affected my accuracy (although this was just by around 0.5 of a percent). I then increased my epoch size from 15 to 25, this was just due to me noticing that the accuracy was continuing to go up when I reached 15 epochs, so I added 10 and this definitely helped my overall training accuracy, although I am unsure how much this affected my test accuracy. This ended up in me reaching a 94.7% train accuracy, and a 94.0% test accuracy.

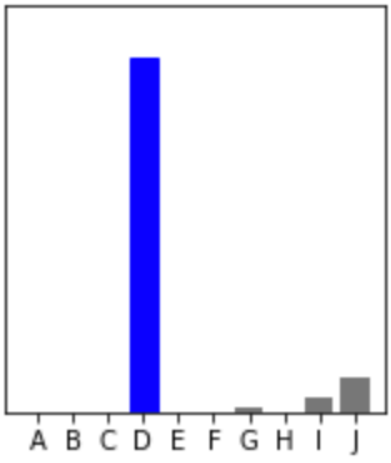
As you can see, my model now predicts H and D with very high confidence, however the confidence of C is still quite low, although it does correctly predict it.

H 94% (H)




```
Pick test_image (0 -> 9999):33
--Should be Class 3--
1/1 [==============================] - 0s 16
```

D 87% (D)




```
Pick test_image (0 -> 9999):85
--Should be Class 2--
1/1 [==============================] - 0s 17
```

C 19% (C)