CPSC 501 – Assignment 4 – Welcome to TensorFlow
Part 2 – Logistic Regression on a Replacement for MNIST Dataset

For Part 2 of this assignment, I've chosen to build off of the neural network developed in part 1.

Building the model

I've changed the three layer model in part 1 by making it a four layer model

- tf.keras.layers.Dropout(0.1)
- I added an extra dropout layer between the dense ReLU and dense softmax layers of part 1 with a value of 10%.
- The dropout layer appears reduce effects of overfitting based on the Keras documentation, but in general appears to have improved the performance of my model

In addition, I've changed the epochs to 30 when fitting the model

- model.fit(x_train, y_train, batch_size=128, epochs=30, verbose=2)
- The larger epoch allows the model to train for more cycles of the training data versus the lower 10 used in part 1

Results: 97.5% on training data, 94.4% on testing data

Changing the predict files

To change the predict files, three changes needed to be made:

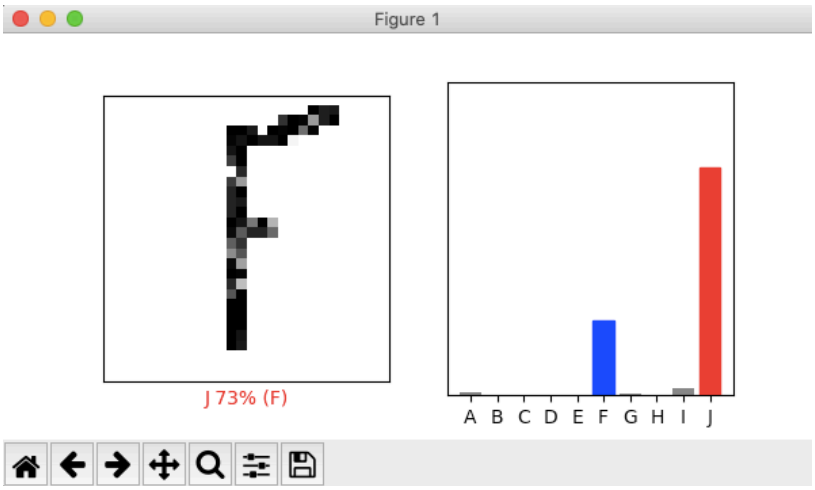
1. Loading the model required the following line of code:
   model = tf.keras.models.load_model(sys.argv[2])
   This allows Keras to deserialize an HDF file representing a model, and load it into the program

2. Making the prediction required the following line of code:
   prediction = model.predict(img)[0]
   This allows the model to predict the class of the input image, and then gets the first array of the NumPy two-dimensional array (since this will always be a 2D array, with the outer dimension being size 1)

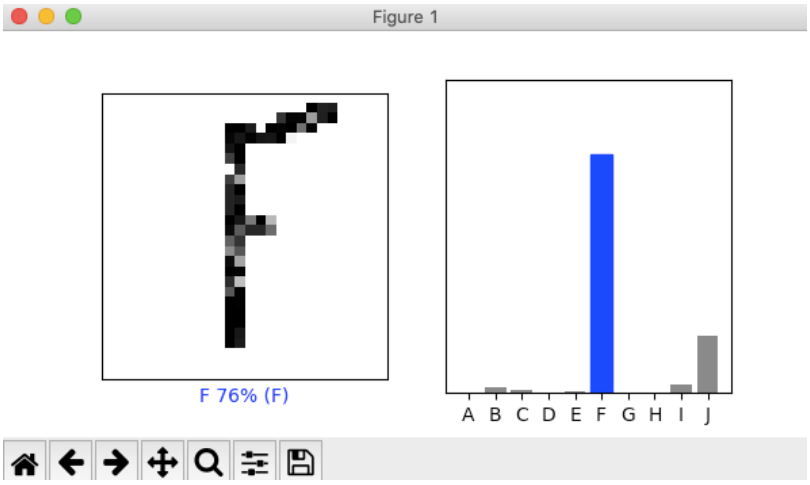
3. Getting the predicted label required the following line of code:
   predicted_label = prediction.argmax(axis=-1)
   Using the argmax function from NumPy, the maximum prediction will be taken from the last (and only) axis of the NumPy array.

Bad prediction

- The initial model was unable to correctly classify the following image of an F, where it predicted with 73% confidence that it was a J



- I changed the activation function in the dense layer to LeakyReLU with an alpha value of 0.1 rather than just using the previous ReLU function. This appears to have a positive effect on the classification of the F, as it now has a 76% confidence it was a F
- tf.keras.layers.LeakyReLU(alpha=0.1)
  Note: This was needed to be added as a new layer rather than just as the activation function to the dense layer as there is a serialization bug in TensorFlow regarding LeakyReLU
  https://github.com/tensorflow/tfjs/issues/1093



- From what was learned in class, the LeakyReLU activation function appears to have an advantage over ReLU by giving some difference between wrong predictions, which appears to be enough to have correctly classified this image.

- Results after change: 96.1% (1.4% loss from original) on training data, 93.8% (0.6% loss from original) on testing data.

The notMNISTModel.py code can simply be ran with:
python notMNISTModel.py
So long as the necessary pre-reqs are installed (TensorFlow 2 and NumPy)