CPSC 501 – Assignment 4 – Welcome to TensorFlow
Part 1 – MNIST Logistic Regression

For Part 1 of this assignment, I've chosen to build a neural network similar to that outlined in the TensorFlow 2 quickstart tutorial found here https://www.tensorflow.org/tutorials/quickstart/beginner

I've changed the Keras model from a two layer model into a three layer model.

1. Flattening
    - tf.keras.layers.Flatten(input_shape=(28, 28))
    - The first layer of the model is the same as the starter code, simply flattening the input 28x28 image into a single dimensional vector.
2. Dense
    - tf.keras.layers.Dense(512, activation='relu')
    - The second layer of the model is a densely connected layer, with 512 outputs. Additionally, the ReLU activation function was chosen.
    - Based on some information learned in class and this article https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a it appears as though the ReLU function both rejects very wrong guesses which appears to be beneficial for digits in MNIST, as well as this rejection gives the layer a higher performance.
3. Dense
    - tf.keras.layers.Dense(10, activation='softmax')
    - The third layer of the model is a densely connected layer with 10 outputs. Additionally, the softmax activation function was chosen.
    - Again, based on the article in layer 2, the softmax function was chosen because it seems to be more favorable in the output layer, especially when the output layer needs to classify more than two categories (which it does).

In addition to changing the model, I also changed the optimizer used from SGD to Adam

- model.compile(optimizer='Adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
- Adam appears to be quite commonly used, and also works relatively well without tweaking the hyperparameters. According to the article https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f it also corrects problems which are seen in other optimizer techniques.

Finally, I've changed the batch_size to 128 and epochs to 10 when fitting the model

- model.fit(x_train, y_train, batch_size=128, epochs=10, verbose=2)
- The larger batch size (from the default 32) appears to at the very least speed up each epoch
- The larger epoch allows the model to train for many cycles of the training data versus just once

Results: 99.8% on training data, 98.1% on testing data

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