Jacob Cuke
UCID: 30030179

CPSC 501 - Assignment 4

All programs can be run with: 'python3 f*ile_name*'
Libraries needed:
pip install numpy
pip install matplotlib
pip install Pillow
pip install pyscreenshot
pip install tensorflow
pip install pandas

Part 1

The base program had the following performance:

Training
Loss: 1.0441
Accuracy: 78.13%

Testing
Loss: 0.57
Accuracy: 87.5%

In order to improve this, I made a number of adjustments to the source code.

Firstly, I changed the activation function of the output layer from 'sigmoid' to 'softmax' as shown in the lectures, which gave better accuracy. I also changed the optimizer from SGD to Adam for the same reason.

Additionally I also added a hidden layer of 128 nodes before the output layer, with the activation function 'relu'. This was an idea taken from the TensorFlow tutorial: https://www.tensorflow.org/datasets/keras_example.

Next, I used a schedule to gradually decrease the learning rate as the training stage went on. After experimenting with different rates, I settled on a starting rate of 0.001, decay rate of 0.96 and decay steps of 1000, which seemed to give the best results.

Finally, I increased the epochs from 1 to 15, which was enough to increase performance of the model to the following:

Training
Loss: 0.0037
Accuracy: 99.97%
Testing
Loss: 0.08
Accuracy: 98.1%

Jacob Cuke
UCID: 30030179

Part 2

In order to make the first version of my model, I applied the same changes that I did in Part 1, but this time I only added 64 nodes in the hidden layer, as I didn't want the time to increase by too much with the following improvements. Next, I added a layer of convolution, and a layer of pooling as described in the lecture slides to try and get better results. Because each epoch now took much longer, I settled for just 10 epochs this time, as I found performance didn't improve significantly past this point. This initial version of my program is saved in the file *notMNISTStarter_before.py* and had the following performance during the testing phase:

Testing
Loss: 0.22
Accuracy: 94.7%

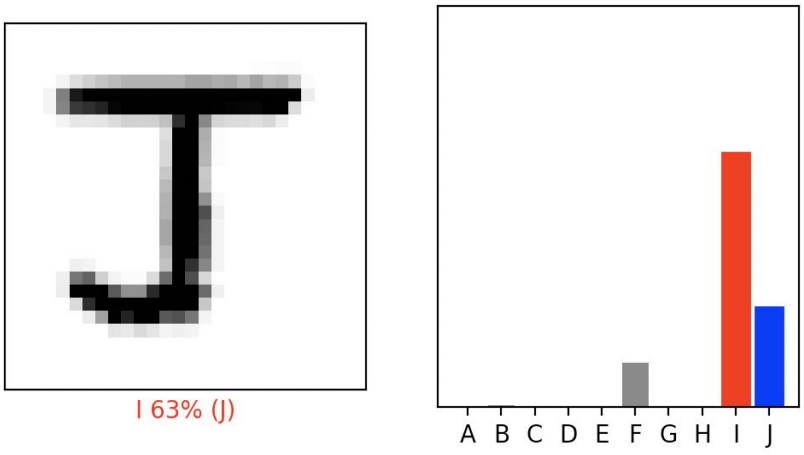
In order to test my program against a hand drawn image, I adjusted the code in *predict.py* so that it would load my model and use it to predict a result based on the supplied image. The code for loading the model is as follows:

```
model = tf.keras.models.load_model(sys.argv[2])
```

Next, I adjusted the prediction code:

```
prediction_results = model.predict(img)[0]
prediction = [0] * 10
for i in range(0, 10):
    prediction[i] = prediction_results[i]
predicted_label = prediction.index(max(prediction))
```

Finally, I fed it an image I drew in GIMP of a 'J' to see how my model held up, and this was the result:

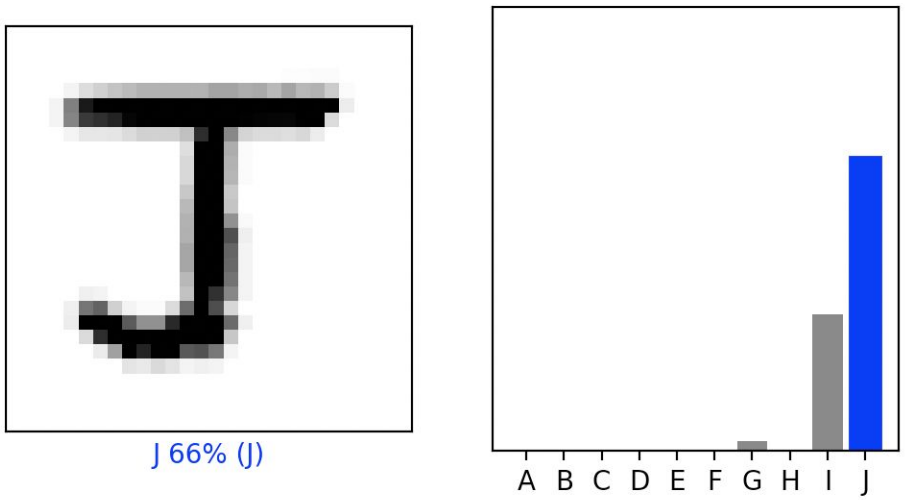


Jacob Cuke
UCID: 30030179

As you can see, my model incorrectly guessed that it was an 'I'. In order to try to improve the results, I added a second convolution and pooling layer to the model, this time with 50 filters. I also added a Dropout layer with a rate of 0.2 to avoid overtraining the model. This second version of the model is saved in the file *notMNISTStarter.py* and had the following performance:

Testing
Loss: 0.17
Accuracy: 94.6%

And when fed the same picture gave the result:



Which is the expected output.

Part 3

Part 3 gave me the most trouble when trying to get it to behave as I expected, but I will document all of the steps and methods that I used.

Firstly, loading the data was done by following the posted tutorial at https://www.tensorflow.org/tutorials/load_data/csv. In order to split heart.csv into heart_train.csv and heart_test.csv, I used the website https://www.splitcsv.com/ to divide the dataset into approximately 85% training and 15% testing. The data was then loaded in using the Pandas API.

Secondly, to handle the pre-processing of the data the tutorial provided 2 different methods. The simpler one involved converting the Present/Absent entries in the dataset into 1/0, popping off the 'chd' column to be our response variable and returning the rest as the indicators. This is the method used in the *CHDModel.py* program. The other, more robust method of pre-processing described in the TensorFlow tutorial involved baking the mapping and pre-processing into the model itself. I used this method in the *CHDModel2.py* program. Finally, I

added a normalization layer into CHDModel.py, as CHDModel2.py already had normalization as part of its preprocessing. The initial performance at 20 epochs was as follows:

CHDModel.py (No normalization)
Training Accuracy: 70%
Testing Accuracy: 75.4%

CHDModel.py (With normalization)
Training Accuracy: 73.5%
Testing Accuracy: 81.2%

CHDModel2.py
Training Accuracy: 71.8%
Testing Accuracy: 78.3%

Next, I tried to apply some of the improvements that I made in parts 1 and 2. I added a scheduler to gradually reduce the learning rate. However this had no effect on the result, and so I decided to remove it. Instead, I changed my current set up of 1 hidden layer of 128 nodes, to 2 hidden layers with 512 nodes each, with the 'elu' activation. This produced the results:

CHDModel.py
Training Accuracy: 70%
Testing Accuracy: 82.6%

CHDModel2.py
Training Accuracy: 73.8%
Testing Accuracy: 81.2%

Finally, to combat overfitting I followed the posted tutorial at https://www.tensorflow.org/tutorials/keras/overfit_and_underfit. I added L2 weight regularization and a Dropout layer with a dropout rate of 0.3. The performance results were as follows:

CHDModel.py
Training Accuracy: 69%
Testing Accuracy: 78.3%

CHDModel2.py
Training Accuracy: 72.8%
Testing Accuracy: 78.3%

Not only did the overfitting countermeasures not improve performance, but it actually got slightly worse. Before adding these measures my models were already getting above 80% accuracy on the test data, and so it would seem that my models were never suffering from overfitting in the first place, despite the small dataset. Nothing I attempted to do improved either the training or

testing accuracies to values above 83%, and adding weight regularization and dropout did nothing to improve performance. I am unsure of the cause for this, but have decided to leave both programs in this final state with the overfit countermeasures added for the assignment submission.