CPSC 501 – Assignment 4 – Welcome to TensorFlow
Part 3 – Build a logistic regression model to predict if someone has coronary heart disease

For Part 3 of this assignment, I've mainly used the tutorials on loading CSV data https://www.tensorflow.org/tutorials/load_data/csv and on overfitting and underfitting data https://www.tensorflow.org/tutorials/keras/overfit_and_underfit

Building the model
Initially, I decided to build a program to split the provided heart.csv file into two files, heart_train.csv and heart_test.csv where heart_train.csv contains 85% of the initial data and heart_test.csv contains the rest. This file can be found in the repository (https://github.com/natejacko/CPSC501-Assignment4) as Part3/split_data.py.

Following splitting the data, I followed closely with the loading CSV data tutorial to load CSV data, split out the numerical and categorical data, normalized the numerical data, then combined the two into a single pre-processing layer to be used in the Keras model.

To build the model, I used a four layer model very similar to that found in the tutorial:

1. Preprocessing
    - tf.keras.layers.DenseFeatures(categorical_columns+numeric_columns)
    - The first layer of the model is a preprocessing layer to combine the flattened categorical and numerical data
2. Dense
    - tf.keras.layers.Dense(256, activation='relu')
    - The next layer is a densely connected layers, with 256 outputs, using the ReLU activation function
3. Dense
    - Same as 2
4. Dense
    - tf.keras.layers.Dense(1, activation='sigmoid')
    - The final layer is a densely connected layer, with 1 output, using the sigmoid activation function

To compile the model, I used the same compiler as the tutorial:
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

Finally, to fit the model, I used 20 epochs, and 128 steps per epoch:
model.fit(packed_train_data, epochs=20, steps_per_epoch=STEPS_PER_EPOCH)
This same amount of steps were also used when evaluating the model

Results: 98.4% on training data, 68.0% on testing data

Adjusting the overfit model

To adjust the overfit model, I played around with suggestions from the overfit and underfit tutorial. Suggestions taken from this tutorial were to simplify the model, add weight regularization to the layers (namely L2 regularization), and to add dropout layers.

To build the model I now used six layers instead of four:

1. Preprocessing
    - This layer stayed the same
2. Dense
    - tf.keras.layers.Dense(32, kernel_regularizer=tf.keras.regularizers.l2(0.01), activation='relu')
    - I simplified this layer by making it have less outputs (32 down from 256) as well as added L2 regularization, in which the regularization appears to put constraints on the weights in the model
3. Dropout
    - tf.keras.layers.Dropout(0.5)
    - I added a dropout layer to randomly drop 50% of output such that further layers do not as heavily depend on the output
4. Dense
    - Same as 2
5. Dropout
    - Same as 3
6. Dense
    - This layer stayed the same from the original model

To compile and fit, I did not change anything about the model

Results: 72.2% on training data, 84.4% on testing data.

Overall, the training data accuracy significantly dropped (from 98.4%), however this led to the testing data accuracy to significantly rise (from 68.0%). I'm unsure which outcome is preferred, but now there at least appears to be a smaller gap between the training and testing data accuracy compared to the old model.

The overfit model changes can be found under GitHub hash code cf46761

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