Christilyn Arjona
30033435

CPSC 501: Fall 2019
Assignment 4

Part 1

Preparing the Data

In order to eliminate data redundancy prior to making the model, I proceeded to re-shape and normalize the given dataset. The x-values in both the training and the test data sets were transformed to floating point values in order to retain decimal points. To normalize the images, the domain in both sets were divided by 255, which is the maximum RGB value.

Building the Model

To build the model, I added convolutional layer to the model. Having convolutional layers allows each image to contain smaller sets of pixels all while retaining original information. In this case, I applied convolution to the data set providing it 3x3 for the kernel size (a seemingly common choice in CNN) and a stride of 1. A stride of 2 is also possible to downsample an image however I decided to apply max pooling technique for this instead. To downsample an input representation, I applied max pooling by 2x2. By having a pooling layer, the problem of overfitting is less of an issue.
To combat overfitting, I introduced dropout layers to 'thin out' the network by disregarding some of the neurons. I provided it a rate of 0.2, representing the fraction of the input units to drop. Moreover, I flattened the data to be used with a 'Dense' layer in order to couple information together. In the final Dense layer, we have 10 values since there are a total of 10 digits to choose from. As for activation functions, I experimented with a couple and found ReLU and softmax to be suitable.

Compiling and Fitting the Model

I chose the optimizer function to be adam as I found it to perform the best out of all other optimizers. Moreover, the original code had the number of epochs to be 1, which is too small, so I experimented with different epoch values and found 10 to be sufficient.

Result

The test data was able to achieve **98.4%** accuracy while the training data showed **99.9%** accuracy.

```
--Evaluate model: TEST DATA --
10000/1 - 2s - loss: 0.0323 - accuracy: 0.9840
Model Loss:    0.06
```

```
Model Accuracy: 98.4%
--Evaluate model: TRAINING DATA --
60000/1 - 11s - loss: 0.0030 - accuracy: 0.9987
Model Loss:    0.00
Model Accuracy: 99.9%
```

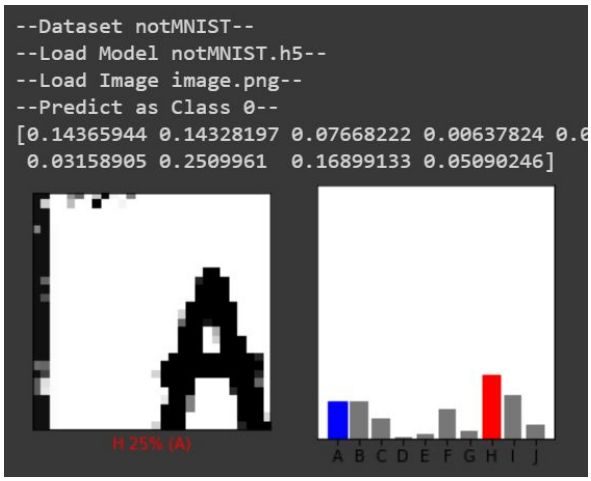
Part 2

Predict files

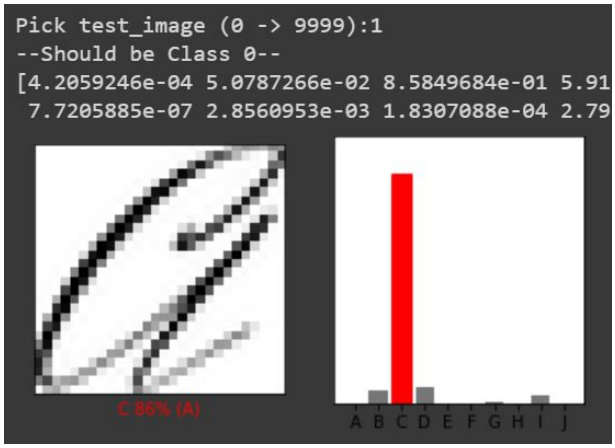
- Loading model
  - To load the mode, I added the following line:
  - `model = tf.keras.models.load_model(sys.argv[2])`
- Array of confidence
  - To get the array of each percent confidence in each class, I used the 'predict' method providing it the image as the input sample
  - `prediction = model.predict(img)[0] # line added`
  - Prior to this, I re-shaped the data so that it corresponds with the model that I created previously through the following:
  - ```
    img = img.reshape(img.shape[0], 28, 28, 1) # line added
    img = img.astype('float32') # line added
    img = img / 255.0 # line added
    ```
- Index of highest prediction:
  - `predicted_label = prediction.argmax()  # line changed`
  - Where argmax returns the index of the maximum value in the array of predictions

First Predictions

predict.py


    ```

predict_test.py



Part 3

Loading Data

To load the data in the google colab, heart.csv was downloaded and placed in the current Runtime Environment. Then, the data was processed through the following:

```
dataset_url = 'heart.csv'
data = pd.read_csv(dataset_url)
```

Splitting Data

Dividing the data into training set, test set, and validation set was done through the following:

```
train, test = train_test_split(data, test_size=0.2)
train, val = train_test_split(train, test_size=0.2)
```

This results in 295 train samples, 74 samples, and 93 test samples (as outputted by the program)

Data Pre-Processing

To process the raw csv data to fit in the model, the data was converted into TensorFlow data set in order to utilize feature columns. The columns were divided into 2 categories. Namely, numerical columns and categorical/indicator columns. In this case, the numerical columns were: sbp, ldl, adiposity, typea, obesity, alcolol, and age. In the categorical column we have the famhist (Family History of Heart Disease 0 = Absent, 1 = Present).

Building the Model

In building the model, a feature layer was added through the following:

```
feature_layer = tf.keras.layers.DenseFeatures(feature_columns)
```

Where the whole model sequence is as follows:

```
model = tf.keras.Sequential([
  feature_layer,
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(rate=0.2),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Training and Compiling the Model

In compiling the model, binary cross entropy was used for the loss function since we are dealing with a ‘Yes’ or ‘No’ decision with 0 = No, and 1 = Yes
To train the model, it was provided with the validation data that was previously extracted, and provided an epoch of 20. The number 20 was chosen in this case as it seems to give the highest accuracy consistently.

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(train_ds,
          validation_data=val_ds,
          epochs=20)
```

Initial Results

The model was able to achieve around 62 - 68% accuracy.

Initial Results

```
---- Evaluate TEST SET ----
3/3 [==============================] - 0s 66ms/step - loss: 0.6890 - accuracy:
0.6129
Model Loss:    0.69
Model Accuracy: 61.3%
---- Evaluate TRAIN SET ----
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.5213 - accuracy:
0.7525
Model Loss:    0.52
Model Accuracy: 75.3%
---- Evaluate VALIDATION SET ----
3/3 [==============================] - 0s 4ms/step - loss: 0.6121 - accuracy:
0.6216
Model Loss:    0.61
Model Accuracy: 62.2%
```

Overfitting Issue

Overfitting occurs when the model fails to generalize a pattern. In this case, since there are only a few test samples, overfitting became prominent as it was not enough to train the model to reach peak accuracy.

To combat overfitting in my initial attempt, I used validation in my data set in which I split the data to include a validation set. Without validation, the model would be sheltered into its own test set and train set, without accounting for the real world performance. However, as seen from the result, this was not sufficient to achieve desirable accuracy.

To improve from this result, I tried to shuffle data while splitting so that each set are somewhat similar to each other and have the same distribution. However, this was still not sufficient in improving accuracy of the model. Moreover, I also tried to vary the split ratio between the different sets. But then again, this failed to improve overall accuracy.

Another method is to incorporate cross validation, further splitting the data into *k* subsets. This helps to combat overfitting by randomly choosing equal sized subsets of the data to be used throughout the training run. The biggest advantage to this is that the model will be less prone to selection bias due to the fact that the training and testing are performed on different parts of the data throughout the training session.