Carlos Lopez

Professor Hare

CS 461

8 December 2022

Program 3 Report

Notes:

I processed the dataset and built the neural network model in Google Colaboratory. To run, “processed\_cleveland.csv” must be manually uploaded onto the Google Colab page under the “Files” tab. It was then saved in the GitHub repository as a .ipynb file. The .ipynb has the output for each code block beneath it. At the end of the file I made the predictions for each case in the validation dataset and output it along with the actual known classification.

Data Preparation:

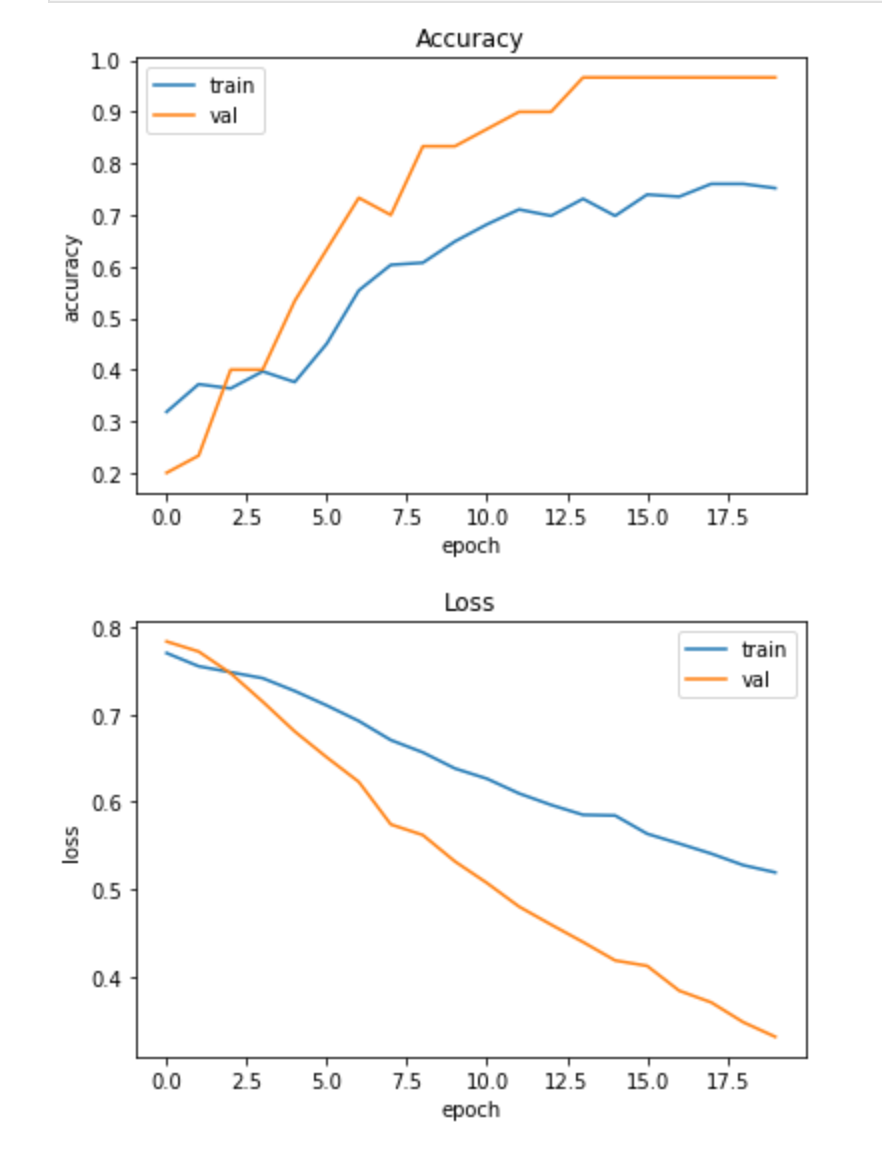
First, I read through the dataset notes file to have a good general understanding of the dataset as a whole. Which included understanding data held by each column and identifying the target label column. After visual inspection of the actual dataset table, I saw that there were 6 rows where a question mark (“?”) was in the “ca” and “thal” columns. An option would have been to just drop these rows but I chose instead to replace every instance of a question mark into NaN, then I filled all of the NaN values with the median number of that particular column. I did this to avoid making the already small dataset even smaller. But in order to fill the NaN with the median I had to convert those two columns into float types instead of object types. After that I went ahead and converted them into int types to be consistent with the rest of the columns.

From here I was ready to prepare the data for training and testing purposes. I split the dataset into the variable X which held only the features and dropped the target label “num”. The target label “num” was then placed by in the variable y. In X, the “trestbps", "chol", and "thalach" columns had the biggest deviations within themselves so I chose to only standardize these three columns. Standardizing these columns was more efficient on the model than not standardizing them because it made the accuracy output through the epochs less erratic as is shown below:

**Before Standardization:**

Chart, line chart

Description automatically generated

**After Standardization:**

After standardizing those columns in X, I changed the data in the target y where if the data num was 1 or greater, then I just changed it to be 1. Since 1,2,3, or 4 meant a positive diagnosis and 0 meant a negative diagnosis. This would give us only two outputs, the model would only need to give a probability of it being closer to 0 or 1.

X and y were then split into train, test, and validation. 80% of the dataset went into training the model, 10% went into testing the model, and the last 10% went into validation.

Network Configuration:

I used a sequential neural network model. It has an input layer with the shape set to 13 for the input features. This is followed by three hidden layers, each hidden layer has 8 neurons. The hidden layers are followed by the output layer. I chose the sigmoid activation function for the output since the output was predicting the probability of 0 and 1. Sigmoid also gave me better results over tanh. After building and compiling the model, it was time to fit it with the training data. I trained it using X\_train and y\_train. The training data runs through the model forwards and backwards over 100 epochs or until the Early Stopping Callback triggers to end the training.

Validation Strategy:

The data was divided into 80% training, 10% testing, and 10% validation. Inside the fit function, I inserted the validation\_data(X\_test, y\_test) parameter. I used this method to validate the model as it was training. As each epoch progressed, a visual output displaying the training accuracy and loss and the validation accuracy and loss were shown. I used this display to compare how the model was training on the data and how it was being validating on the test data. The early stopping callback was set to monitor the validation accuracy to stop if its progress stagnated or diminished. This helped prevent overfitting as well. Also to help identify overfitting, I used a visual plot of the epochs, if the accuracy of the training set kept increasing and the accuracy of the testing set stopped increasing or decreased then I could visually see if the model was over fitting. After this, I used the validation set that the model had not seen before and evaluated it on the trained model. Its accuracy score should be similar to the the trained set score, if it was lower then that would be an indicator that the model memorized the training set.

Results:

The validation set had an accuracy score of 77%, it was able to correctly identify ¾ of the cases. Tweaking the model by removing or adding a layer, changing the activation functions, changing the number of epochs, changing the call back function all had an impact on the model. It was a tweaking game and a challenge to make small changes until the model performed desirably.

Comments:

The results could be better, out of all the changes I made it was the best I could get with the current model. I think accuracy could improve given a bigger data set. With a bigger data set we could build a slightly bigger model that might have better accuracy. I was surprised that standardizing the “trestbps", "chol", and "thalach" columns improved performance. The deviation within those data columns didn’t seem that large but in the end it did make a difference.

References:

<https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html>

https://keras.io/guides/sequential\_model/