CIS2350

Red Wine Neural Network

Neil Bugeja 51000L

Computing and Business

L-Università ta' Malta

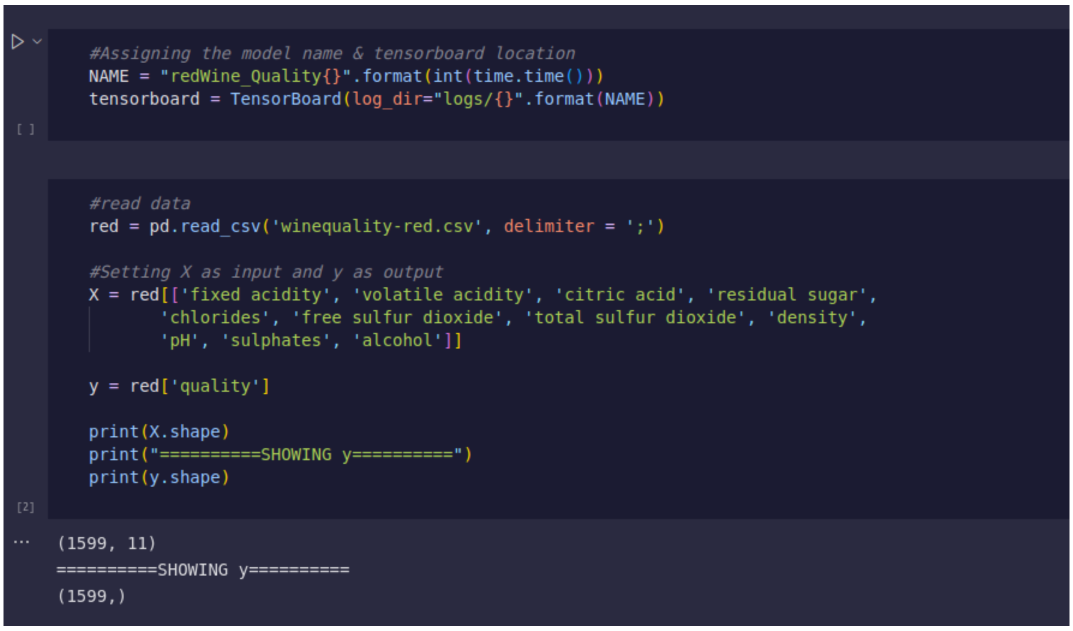
November 26, 2022

A sequential neural network was constructed using Tensorflow and Keras

Setting up and reading data

At the beginning of the program, multiple imports are present that are used throughout the program. Their functionality range from reading the data set, to constructing the NN and to even create models that will be used to demonstrate the application.

First, apart from the imports, we set a name that will be used for the model and set the TensorBoard location. After this, panda is used to read the red wine dataset and the input variables (fixed acidity, density, alcohol, etc.) are assigned as X and the output variable (quality) is assigned as y. Finally, the data is simply outputted to confirm that everything is assigned as intended.



```
#Assigning the model name & tensorboard location
NAME = "redWine_Quality{}".format(int(time.time()))
tensorboard = TensorBoard(log_dir="logs/{}".format(NAME))
```

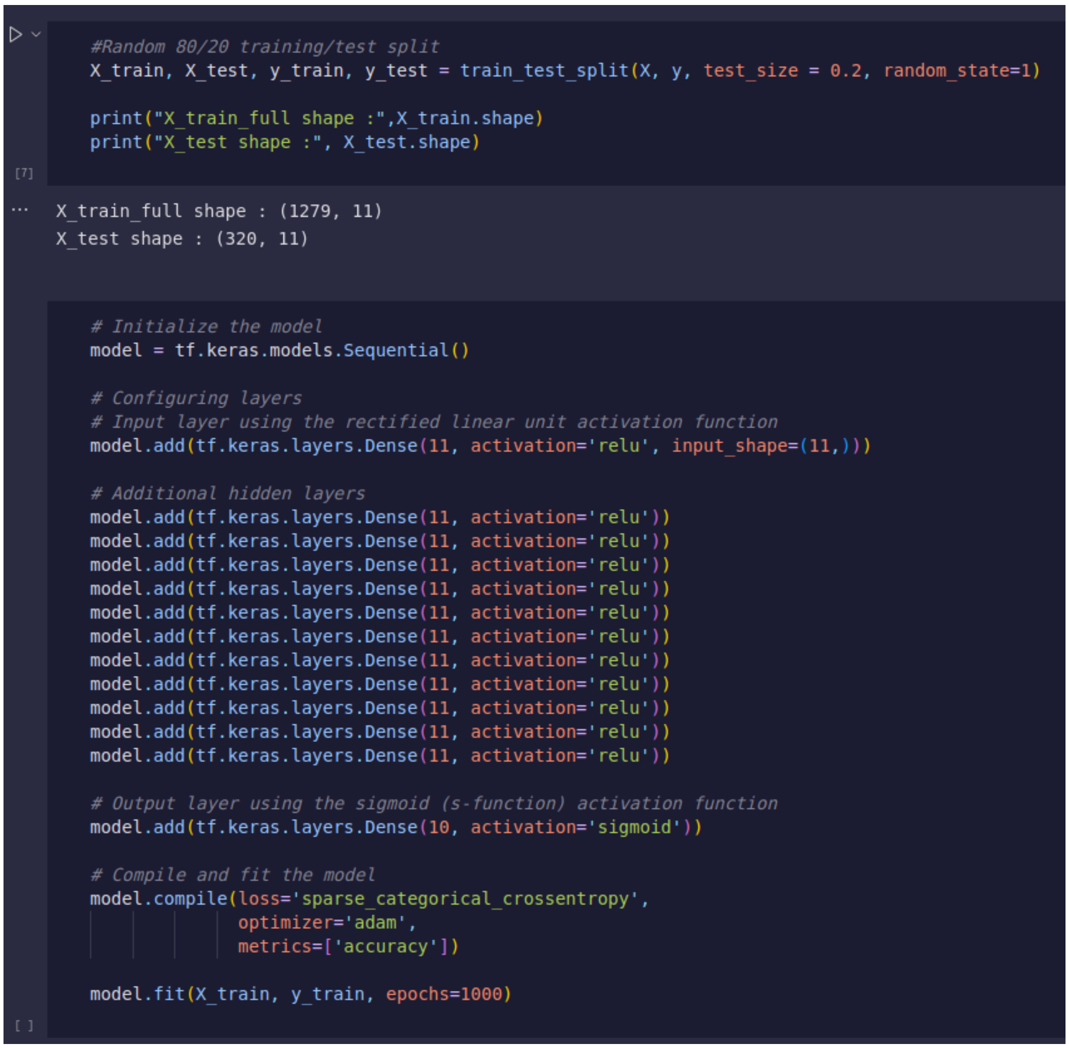
```
#read data
red = pd.read_csv('winequality-red.csv', delimiter = ';')

#Setting X as input and y as output
X = red[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol']]

y = red['quality']

print(X.shape)
print("==========SHOWING y==========")
print(y.shape)
```

```
(1599, 11)
==========SHOWING y==========
(1599,)
```

Training, testing and creating the model

As stated in the assignment requirements, a 80/20 split was done, resulting in 1279 training cases and 320 test cases. After this, a sequential model is created and multiple layers are added to it. We first have the input layer, which is where the data will be inputted from. It's shape is set to the number of elements we have in the input (fixed acidity, density, alcohol, etc). After this, we have a number of hidden layers that are used by the NN to learn what elements make up a good quality wine. This number is set to 11 as per the assignment guidelines. Finally, we have our output layer.

The model is compiled using generic loss, optimizer and metrics parementers and later the training data is fed into the model so the training can commence.



```
#Random 80/20 training/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=1)

print("X_train_full shape :",X_train.shape)
print("X_test shape :", X_test.shape)
```

[7]

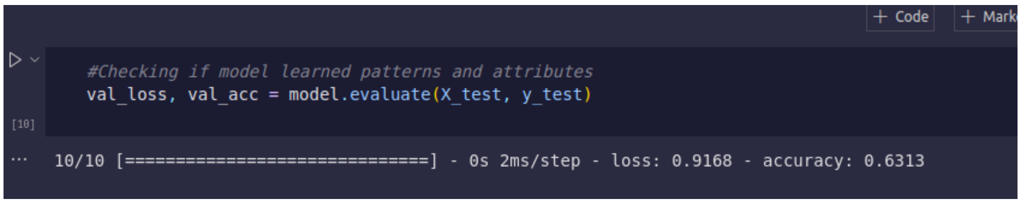
```
X_train_full shape : (1279, 11)
X_test shape : (320, 11)
```

```
# Initialize the model
model = tf.keras.models.Sequential()

# Configuring layers
# Input layer using the rectified linear unit activation function
model.add(tf.keras.layers.Dense(11, activation='relu', input_shape=(11,)))

# Additional hidden layers
model.add(tf.keras.layers.Dense(11, activation='relu'))
model.add(tf.keras.layers.Dense(11, activation='relu'))
model.add(tf.keras.layers.Dense(11, activation='relu'))
model.add(tf.keras.layers.Dense(11, activation='relu'))
model.add(tf.keras.layers.Dense(11, activation='relu'))
model.add(tf.keras.layers.Dense(11, activation='relu'))
model.add(tf.keras.layers.Dense(11, activation='relu'))
model.add(tf.keras.layers.Dense(11, activation='relu'))
model.add(tf.keras.layers.Dense(11, activation='relu'))
model.add(tf.keras.layers.Dense(11, activation='relu'))
model.add(tf.keras.layers.Dense(11, activation='relu'))

# Output layer using the sigmoid (s-function) activation function
model.add(tf.keras.layers.Dense(10, activation='sigmoid'))

# Compile and fit the model
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=1000)
```

[ ]

Loss, Accuracy and model

In the final parts of the project, we display the NN's loss and accuracy so we can determine if the model has actually learned patterns rather then simple memorising the data. After this, ann_visualizer is used to create a diagram of out NN.



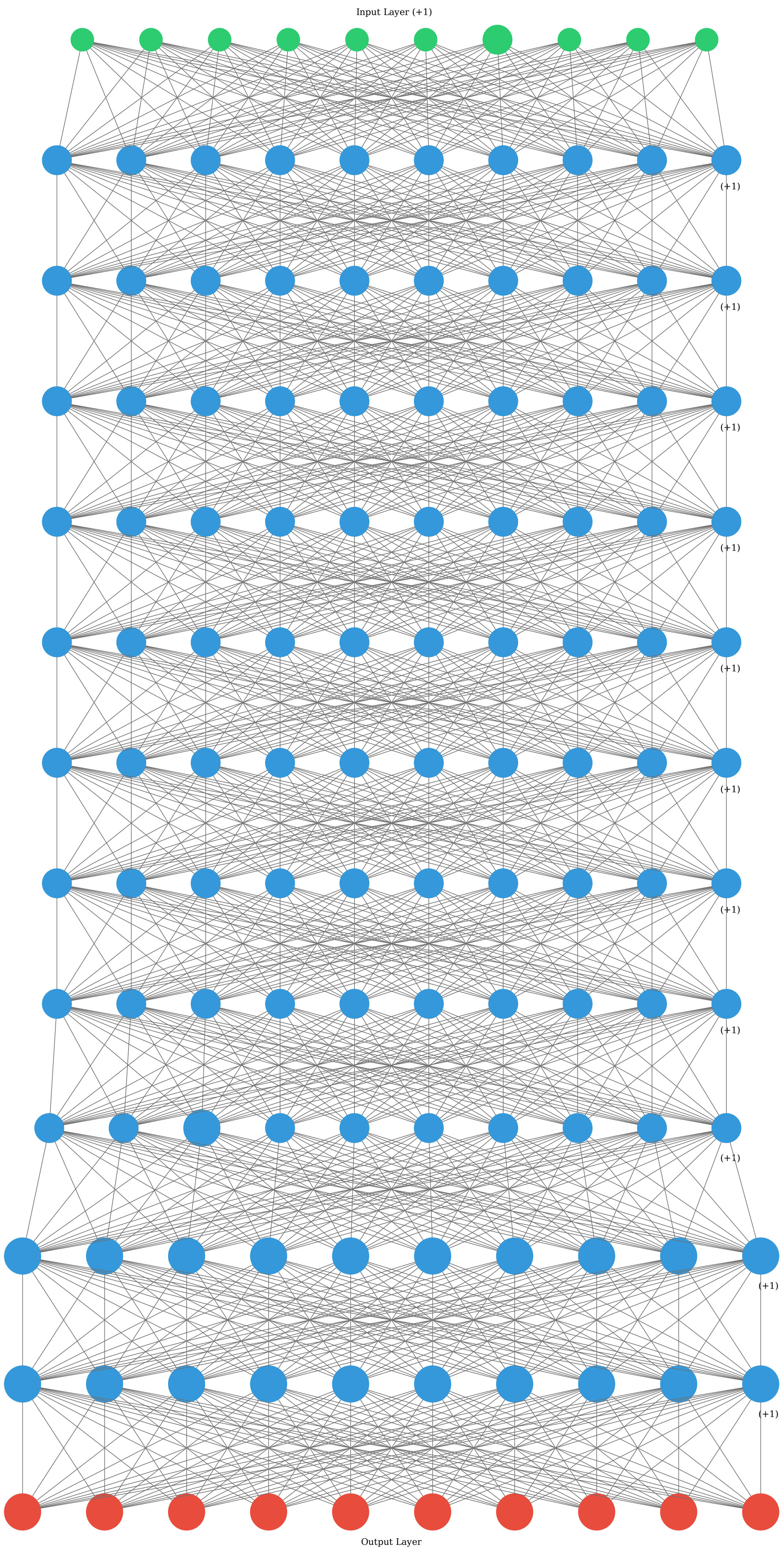
```
#Checking if model learned patterns and attributes
val_loss, val_acc = model.evaluate(X_test, y_test)
```

[10]

```
10/10 [==============================] - 0s 2ms/step - loss: 0.9168 - accuracy: 0.6313
```

Below please find a diagram of the model created.

Red Wine Quality NN

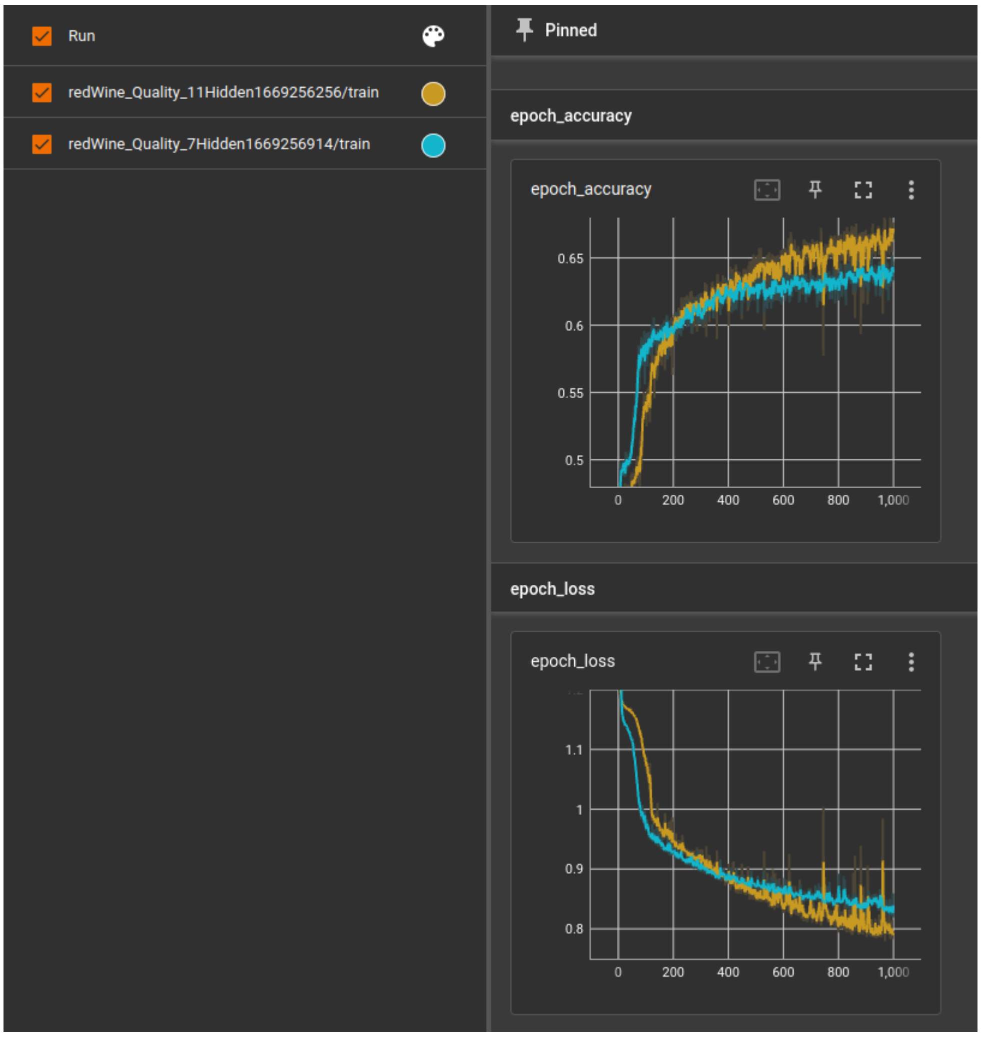


Optimisation

TensorBoard

TensorBoard is used to save the model everytime a change is made. For example, we first have 11 hidden layers and we name the model appropriatly by altering the NAME variable at the beginning. Next, we re run the program but this time we only keep 7 hidden layers. Again, we name the model appropriatly. After running both models, we can access TensorBoard and we have a various graphs that show us the difference in performance between the two models.

Using TensorBoards, we can make multiple changes to the model by altering its number of hidden layers, epochs, activation and loss function, etc.



Validation Data

During each epoc our model is being trained on the training data and will learn the features of that data. The goal is to create a model that is able to accurately predict data that the model has not seen based on what it has been trained on.

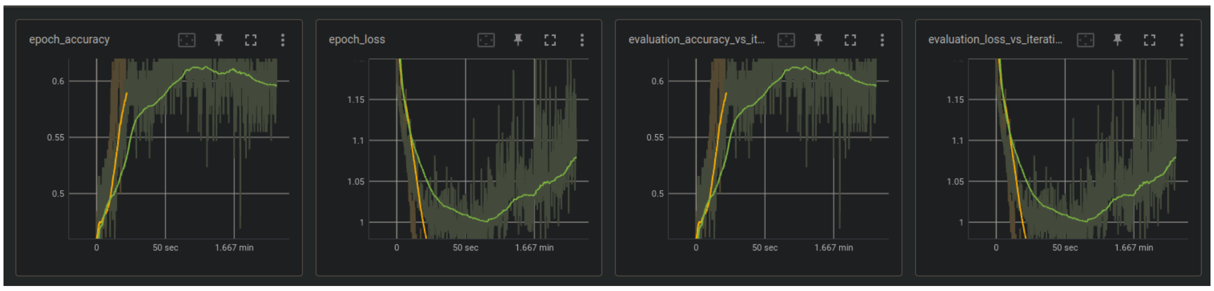
With a validation set, we are removing a percentage of the training set and we will use that percentage as validation. During training, the model is going to predict on the validation set during data and see how well it did. This means that we are going to able to see the model's loss and accuracy on the training set and the validation set.

Another benifit of including validation data is that it makes sure that the model is not over fitting. This means that the model is not only learning the specifics of the training set and memorize them, but it is learning after each epoc by the predictions that it is performing on the validation set which it has not seen before

Number of epocs

Two models were ran (one with 200 epocs and one with 1000 epocs) and their performance was saved on TensorBoard. Let the orange line represent the model with 200 epocs, and the green line represents the model with 1000 epocs. In this instance, we are comparing the validation data not the training data. The reason for this is that the validation data provides a more accurate representation of how well the model is performing as it is making predictions on data that it has not yet seen.

As is demonstrated in the graphs below, the 1000 epocs model is reaching a higher accuracy, but over time it's losses are increasing significantly. This means that the model is accuratly predicting the data better then its counterpart, but the times in which it fails it is doing so with a bigger margin.



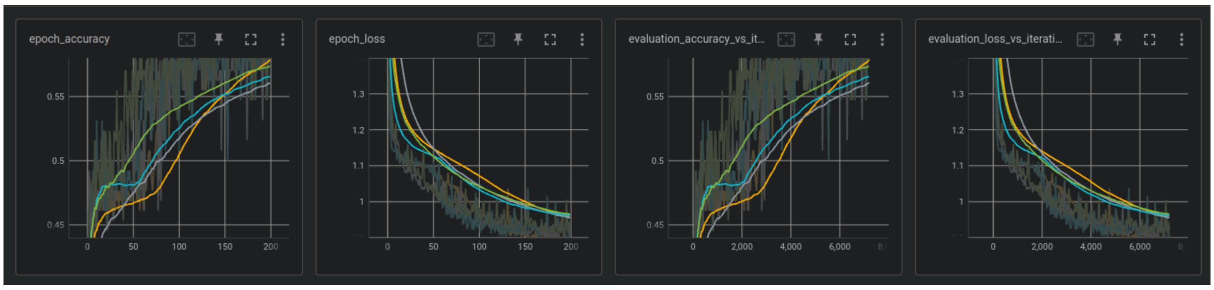
Batch Size

A batch size is simply the number of samples that will be passed through to the network at one time. A batch size will greatly improve the time it takes to perform the training, but again, this is a variable that must be fine tuned using TensorBoard.

Hidden Layers

The number of hidden layers to construct the model with was also tested. Below find various graphs demonstrating the model's performance using variable data. The graphs are as follows:

- Blue : 11 hidden layers
- Orange : 6 hidden layers
- Green : 3 hidden layers
- Grey : 31hidden layers



Using the same reasoning we used in determining the appropriate number of epocs, we can see do great difference between the various models, but the model with the 6 hidden layers appears to be performing the best.

Final Model

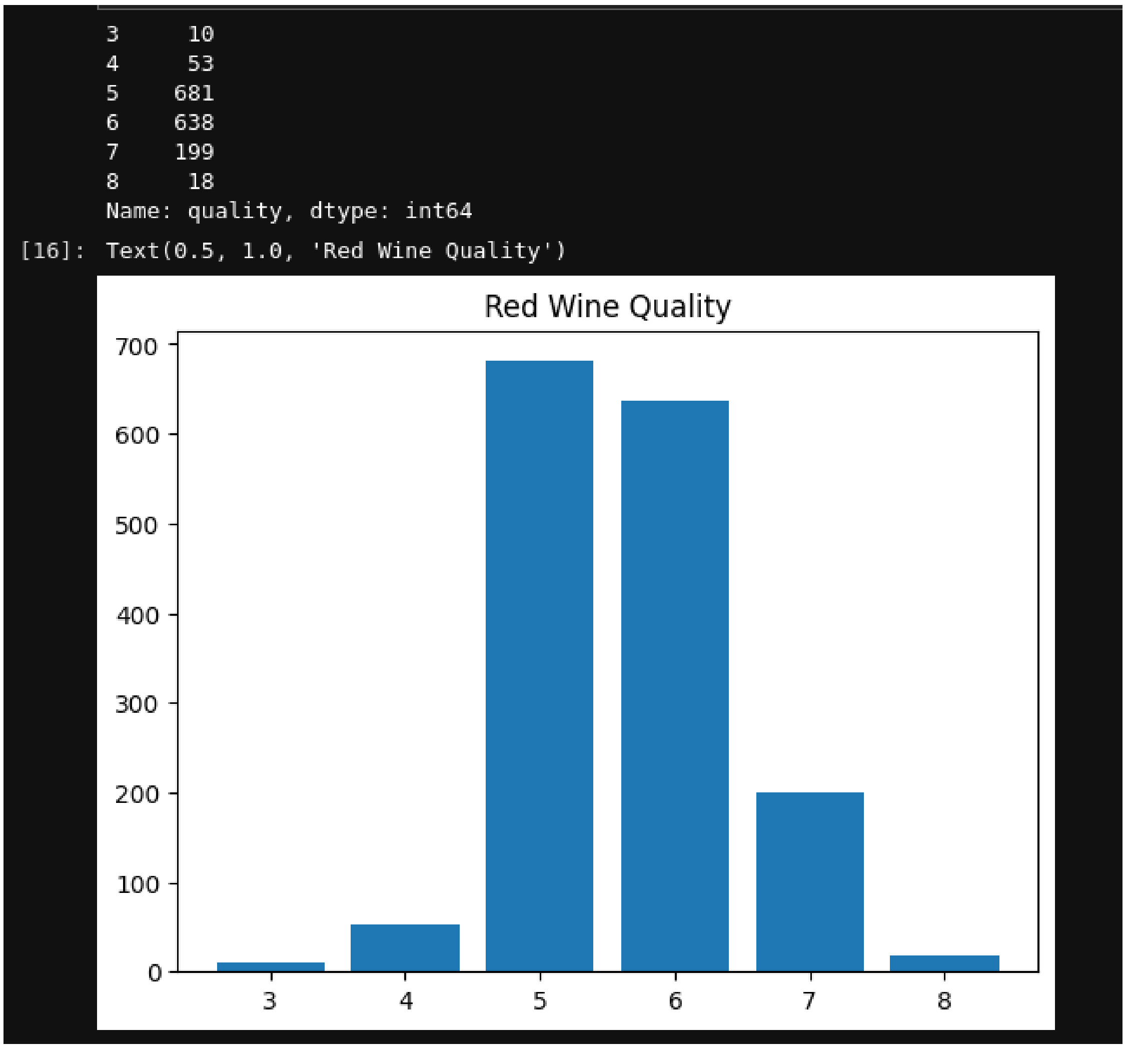
Using the optimisations mentioned above, the final model looks like this:

```python
# Initialize the model
model = tf.keras.models.Sequential()

# Configuring layers
# Input layer using the rectified linear unit activation function
model.add(tf.keras.layers.Dense(11, activation='relu', input_shape=(11,)))

# Additional hidden layers
model.add(tf.keras.layers.Dense(11, activation='relu'))

# Output layer using the sigmoid (s-function) activation function
model.add(tf.keras.layers.Dense(10, activation='sigmoid'))

# Compile and fit the model
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=200, validation_split=0.1, shuffle=True,
    callbacks=[tensorboard])
```

Models

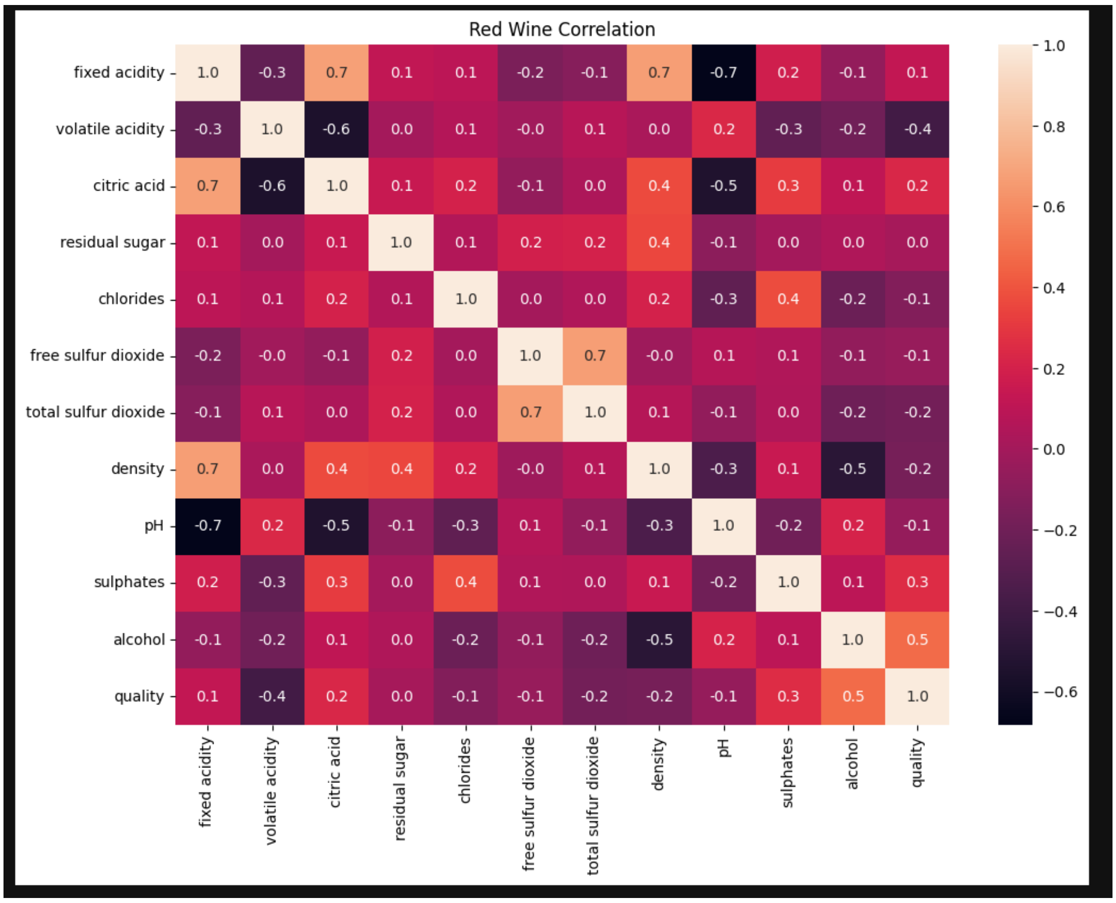
A number of models can be applied on the dataset to better understand how the quality of the wine is being determined.

Using pyplot the total quality of wines is able to be shown, along with a barchart to graphically present this data.



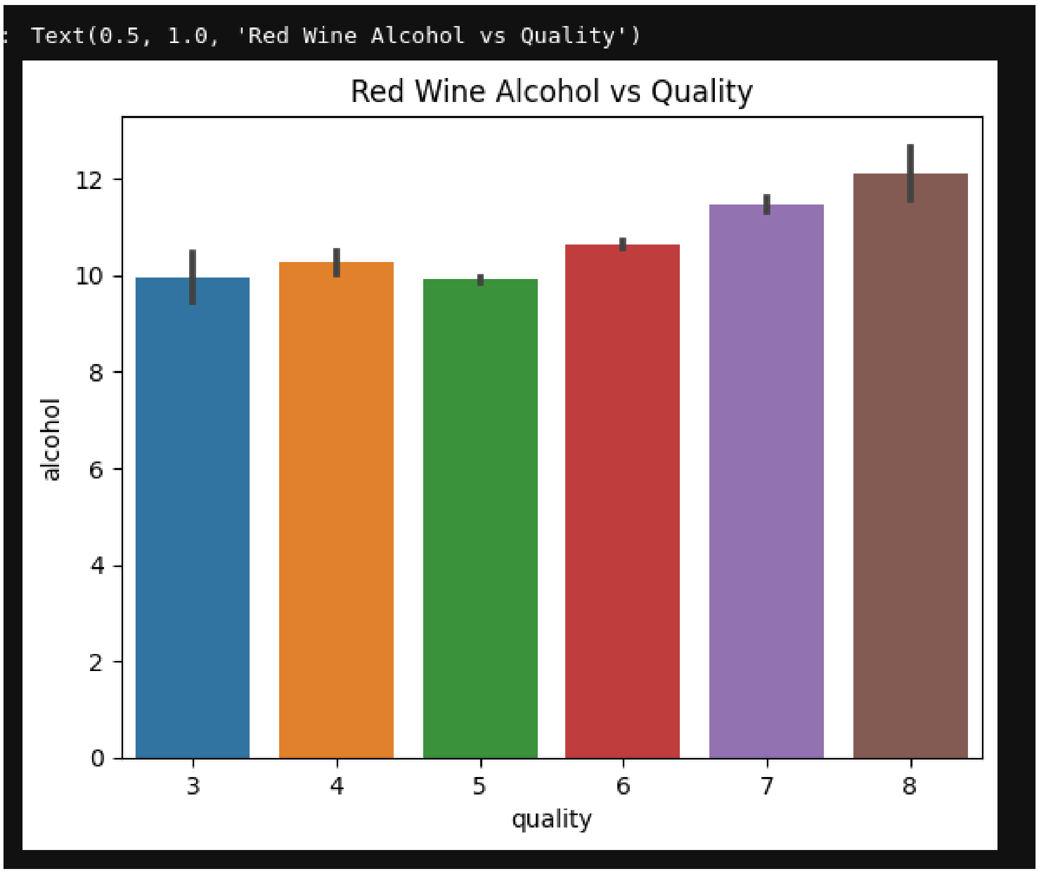
```
       3     10
       4     53
       5    681
       6    638
       7    199
       8     18
       Name: quality, dtype: int64
[16]:  Text(0.5, 1.0, 'Red Wine Quality')
```

Additionally, seaborn was used to draw a heatmap of the red wine dataset. This is unseful as it informs us of which element is effecting quality the most



From the heat map, we can see that alcohol plays the biggest effect on quality (0.5) with sulphates taking the second place (0.3).

Also using seaborn, a bar graph is constructed demonstrating the alcohol level vs quality of the wines.



```
: Text(0.5, 1.0, 'Red Wine Alcohol vs Quality')
```

codeFinal

November 26, 2022

```
[ ]: #imports
     import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt

     from sklearn.neural_network import MLPClassifier
     import tensorflow as tf

     #Tools for modelling
     from sklearn.preprocessing import StandardScaler
     from sklearn.metrics import confusion_matrix, classification_report,␣
      ↪accuracy_score
     from sklearn.model_selection import cross_val_score, train_test_split

     #Used to display NN diagram
     from ann_visualizer.visualize import ann_viz

     from tensorflow.keras.callbacks import TensorBoard
     import time
```

```
[ ]: #Assigning the model name & tensorboard location
     NAME = "redWine_Quality_1H{}".format(int(time.time()))
     tensorboard = TensorBoard(log_dir="logs/{}".format(NAME))
```

```
[ ]: #read data
     red = pd.read_csv('winequality-red.csv', delimiter = ';')

     #Setting X as input and y as output
     X = red[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
              'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
              'pH', 'sulphates', 'alcohol']]

     y = red['quality']

     print(X.shape)
     print("==========SHOWING y==========")
print(y.shape)
```

```
[ ]: #Displaying data from red db
     #Using seaborn to display the data
     dims = (11.7,8.27)
     fig,ax = plt.subplots(figsize=dims)
     CorMap = sns.heatmap(red.corr(), annot = True,fmt= '.1f', ax=ax)
     plt.title('Red Wine Correlation')
```

```
[ ]: #Red Wine Display Quality Bar Chart
     barClass = pd.value_counts(red['quality']).sort_index()
     print(barClass)
     plt.bar(barClass.index, barClass, align = 'center')
     plt.xticks(barClass.index)
     plt.title('Red Wine Quality')
```

```
[ ]: sns.barplot(x = 'quality', y='alcohol', data = red)
     plt.title('Red Wine Alcohol vs Quality')
```

```
[ ]: #Random 80/20 training/test split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
      ↪random_state=1)

     print("X_train_full shape :",X_train.shape)
     print("X_test shape :", X_test.shape)
```

```
[ ]: # Initialize the model
     model = tf.keras.models.Sequential()

     # Configuring layers
     # Input layer using the rectified linear unit activation function
     model.add(tf.keras.layers.Dense(11, activation='relu', input_shape=(11,)))

     # Additional hidden layers
     model.add(tf.keras.layers.Dense(11, activation='relu'))

     # Output layer using the sigmoid (s-function) activation function
     model.add(tf.keras.layers.Dense(10, activation='sigmoid'))

     # Compile and fit the model
     model.compile(loss='sparse_categorical_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])

     model.fit(X_train, y_train, epochs=200, validation_split=0.1, shuffle=True,␣
      ↪callbacks=[tensorboard])
```

```
[ ]: model.summary()
```

```
[ ]: #Checking if model learned patterns and attributes
     val_loss, val_acc = model.evaluate(X_test, y_test)
```

```
[ ]: #saving model
     model.save(NAME)
```

```
[ ]: #creating NN diagram
     ann_viz(model, title="Red Wine Quality NN")
```

```
[ ]:
```