Alphabet Soup’s Neural Network Model Analysis Report

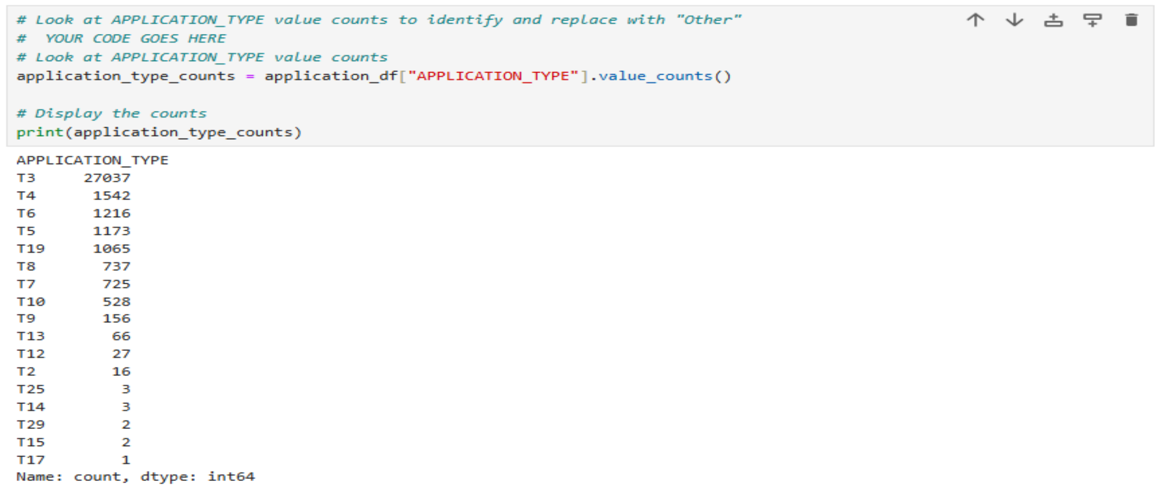
1.0 Overview

This report presents the findings of the developed neural network model for Alphabet Soup, a nonprofit foundation, which needed a classification model that could help them in selecting the applicants most likely to succeed in their ventures once funded by Alphabet Soup. Using the data provided by the foundation’s business team, the model was built sequentially and its results presented in this report’s subsequent sections.

2.0 Results

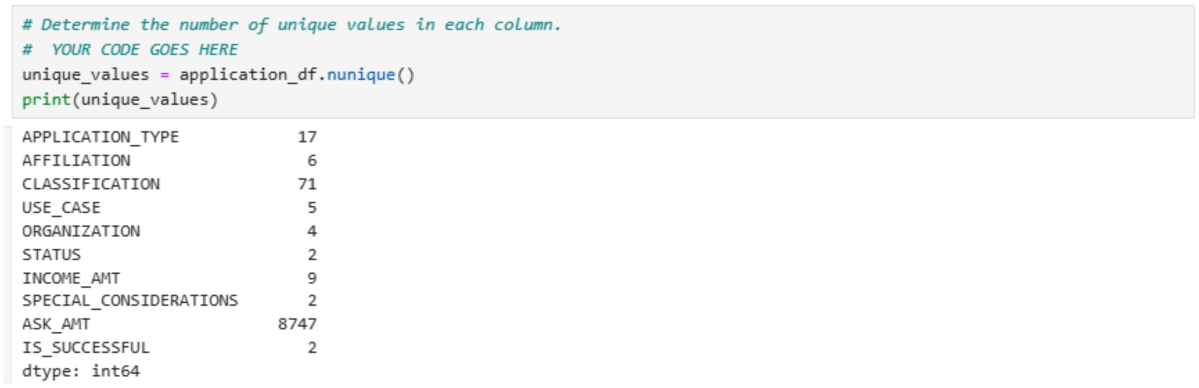
2.1 Data Preprocessing

- The analysis begun with the importation of relevant dependencies such as tensorflow, train_test_split, StandardScaler, and pandas. The dataset from Alphabet Soup was also read in as a pandas dataframe.
- The EIN and NAME columns were removed at this stage as they were irrelevant to the analysis.
- The number of unique values for each column along with their counts was then determined as shown in the below images:



```
# Look at APPLICATION_TYPE value counts to identify and replace with "Other"
#  YOUR CODE GOES HERE
# Look at APPLICATION_TYPE value counts
application_type_counts = application_df["APPLICATION_TYPE"].value_counts()

# Display the counts
print(application_type_counts)
```

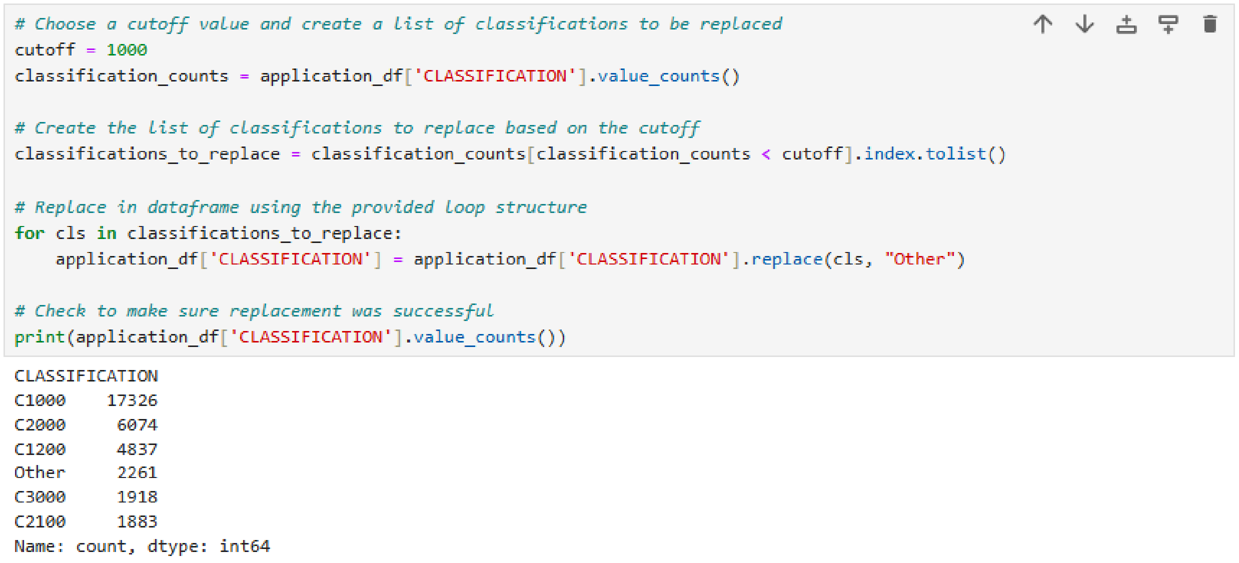
```
APPLICATION_TYPE
T3      27037
T4       1542
T6       1216
T5       1173
T19      1065
T8        737
T7        725
T10       528
T9        156
T13        66
T12        27
T2         16
T25         3
T14         3
T29         2
T15         2
T17         1
Name: count, dtype: int64
```



```
# Determine the number of unique values in each column.
#  YOUR CODE GOES HERE
unique_values = application_df.nunique()
print(unique_values)
```

```
APPLICATION_TYPE            17
AFFILIATION                  6
CLASSIFICATION              71
USE_CASE                     5
ORGANIZATION                 4
STATUS                       2
INCOME_AMT                   9
SPECIAL_CONSIDERATIONS       2
ASK_AMT                   8747
IS_SUCCESSFUL                2
dtype: int64
```

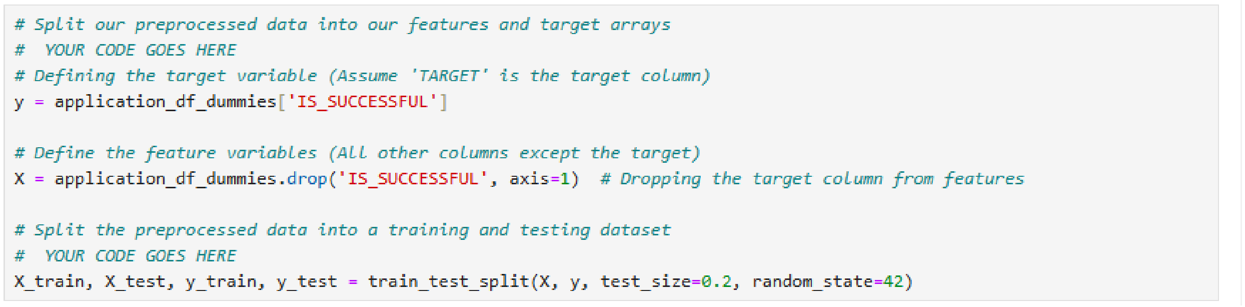
- From this, a cutoff value of 1000 – informed by the unique value counts was selected and a list of classifications to be replaced created as shown below:



```
# Choose a cutoff value and create a list of classifications to be replaced
cutoff = 1000
classification_counts = application_df['CLASSIFICATION'].value_counts()

# Create the list of classifications to replace based on the cutoff
classifications_to_replace = classification_counts[classification_counts < cutoff].index.tolist()

# Replace in dataframe using the provided loop structure
for cls in classifications_to_replace:
    application_df['CLASSIFICATION'] = application_df['CLASSIFICATION'].replace(cls, "Other")

# Check to make sure replacement was successful
print(application_df['CLASSIFICATION'].value_counts())
```

```
CLASSIFICATION
C1000    17326
C2000     6074
C1200     4837
Other     2261
C3000     1918
C2100     1883
Name: count, dtype: int64
```

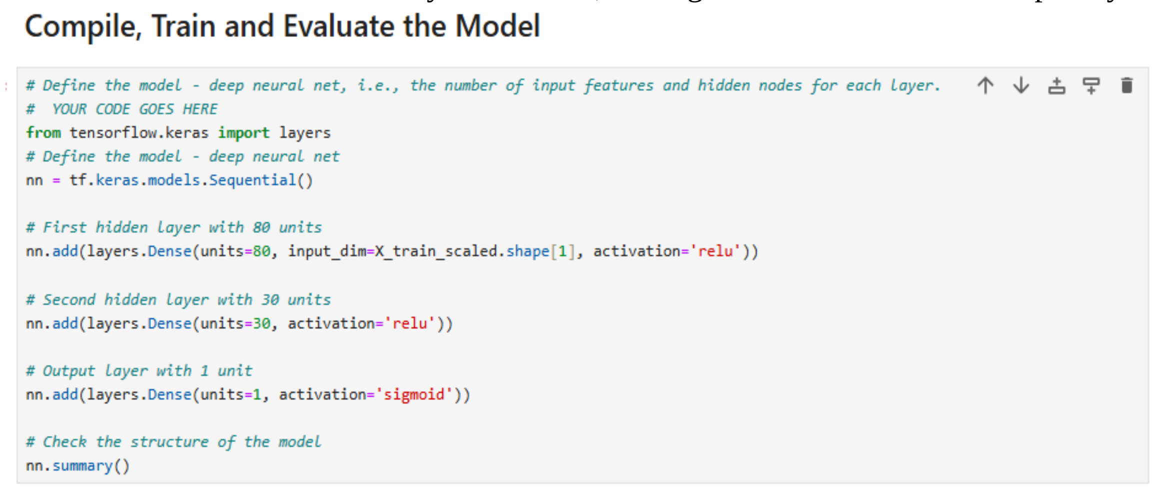
- Categorical data was then converted to numeric using pd.get_dummies as numeric data is best used for such tasks.
- Thereafter, the preprocessed data was split into feature array – denoted as X -, and target array, Y. The target variable was identified as 'IS_SUCCESSFUL', and the feature variable were the rest of the columns once the target variable was dropped.



```
# Split our preprocessed data into our features and target arrays
#  YOUR CODE GOES HERE
# Defining the target variable (Assume 'TARGET' is the target column)
y = application_df_dummies['IS_SUCCESSFUL']

# Define the feature variables (All other columns except the target)
X = application_df_dummies.drop('IS_SUCCESSFUL', axis=1)  # Dropping the target column from features

# Split the preprocessed data into a training and testing dataset
#  YOUR CODE GOES HERE
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2.2 Compile, Train, and Evaluate the Model

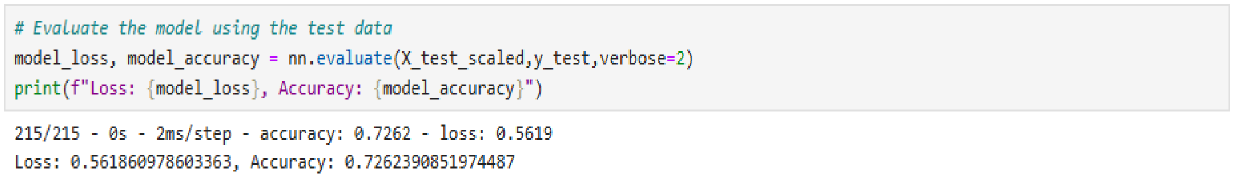
- The neural network consists of 3 layers, 111 neurons, and 3 activation functions. The first hidden layer, as shown in the image below, contains 80 neurons. The second hidden layer contains 30 neurons, and Output layer consists of 1 neuron. The activation function used for the first and second hidden layers was relu, and sigmoid was used for the output layer.



Compile, Train and Evaluate the Model

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
#  YOUR CODE GOES HERE
from tensorflow.keras import layers
# Define the model - deep neural net
nn = tf.keras.models.Sequential()

# First hidden layer with 80 units
nn.add(layers.Dense(units=80, input_dim=X_train_scaled.shape[1], activation='relu'))

# Second hidden layer with 30 units
nn.add(layers.Dense(units=30, activation='relu'))

# Output layer with 1 unit
nn.add(layers.Dense(units=1, activation='sigmoid'))

# Check the structure of the model
nn.summary()
```

- This allowed the network to learn efficiently and attempt to provide the correct output.
- The developed neural network model produced a model accuracy of 72.62%, which falls short of the target model accuracy of 75%.



```
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
215/215 - 0s - 2ms/step - accuracy: 0.7262 - loss: 0.5619
Loss: 0.561860978603363, Accuracy: 0.7262390851974487
```

- In an attempt to increase the model's accuracy, a number of possible solutions were explored as depicted in the AlphabetSoupCharity_Optimization notebook. They included:
  - Dropping more columns: 'APPLICATION_TYPE_Other', 'INCOME_AMT_1-9999', 'SPECIAL_CONSIDERATIONS_N', 'APPLICATION_TYPE_T10', 'INCOME_AMT_50M+'.
  - Adding more neurons to a hidden layer.
  - Adding more hidden layers.
  - Using different activation functions for the hidden layers.
  - Adding the number of epochs to the training regimen.
  - Reducing the number of epochs from the training regimen.
- Despite the solutions above, the model's accuracy still fell below the desired 75%. As seen in the AlphabetSoupCharity_Optimization notebook, each solution produced a different accuracy score, all which were below 75%.

3.0 Summary

- The aim of this analysis was to develop a neural network model that was to be used by Alphabet Soup in predicting the candidates most likely to succeed once they had received funding. The developed model produced an accuracy of 72.62%, and a loss of 0.571%.

Despite falling short of the desired 75%, an accuracy score of 72.62% means that the developed model correctly predicted the outcome about 72.62% of the time.

- The loss value of 0.571% speaks to the deviation of the model’s predictions from the actual results. This, while being higher and as such unideal, shows that the model is learning and poised to improve over time.
- Despite the attempts of increasing the model’s accuracy falling short, the analyst recommends experimenting with more data as a way of further establishing the model’s accuracy, along with further tuning and training of the model.