Neural Network Model Report

Overview of the analysis:

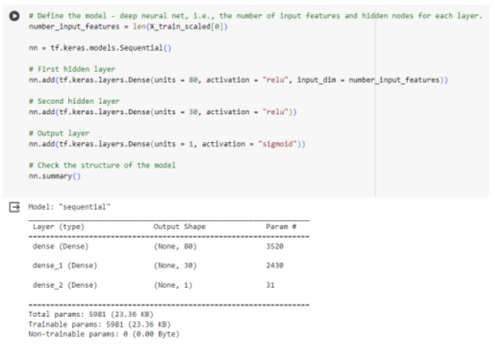
The nonprofit foundation Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their ventures. The CSV contains more than 34,000 organizations that have received funding from Alphabet Soup over the years. Within this dataset are several columns that capture metadata about each organization.

Data Preprocess:

The dataset removed any irrelevant information; therefore, EIN and NAME were dropped from the model. The remaining columns were considered features for the model. Although NAME was added back in another test. CLASSIFICATION and APPLICATION_TYPE was replaced with "Other" due to high fluctuation. The data was split into training and testing sets of data. The target variable for the model is "IS_SUCCESSFUL" and is verified by the value, 1 was considered yes and 0 was no. APPLICATION data was analyzed, and CLASSIFICATION's value was used for binning. Each unique value used several data points as a cutoff point to bin "rare" categorical variables together in a new value, "Other". Afterwards checked to see if binning was successful. Categorical variables were encoded by "pd.get_dummies()".

Compile, Train, and Evaluate the Model

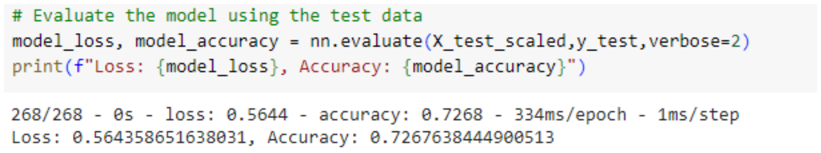
Neural Network was applied on each model multiple layers.



```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train_scaled[0])

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units = 80, activation = "relu", input_dim = number_input_features))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units = 30, activation = "relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units = 1, activation = "sigmoid"))

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 80)                3520

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 5981 (23.36 KB)
Trainable params: 5981 (23.36 KB)
Non-trainable params: 0 (0.00 Byte)
```

This first attempt came close to 72.68%, which was under the desired 75%.



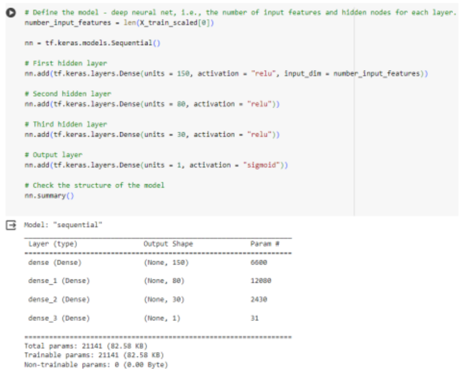
```
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 0s - loss: 0.5644 - accuracy: 0.7268 - 334ms/epoch - 1ms/step
Loss: 0.564358651638031, Accuracy: 0.7267638444900513
```

Optimization:

Here are 3 ways used to optimize the model.

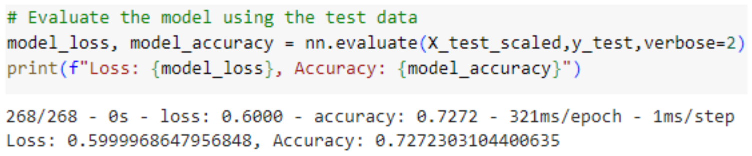
1) Added one more hidden layer using activation function 'relu' to the model.



```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train_scaled[0])

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units = 150, activation = "relu", input_dim = number_input_features))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units = 80, activation = "relu"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units = 30, activation = "relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units = 1, activation = "sigmoid"))

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 150)               6600

 dense_1 (Dense)             (None, 80)                12080

 dense_2 (Dense)             (None, 30)                2430

 dense_3 (Dense)             (None, 1)                 31

=================================================================
Total params: 21141 (82.58 KB)
Trainable params: 21141 (82.58 KB)
Non-trainable params: 0 (0.00 Byte)
```

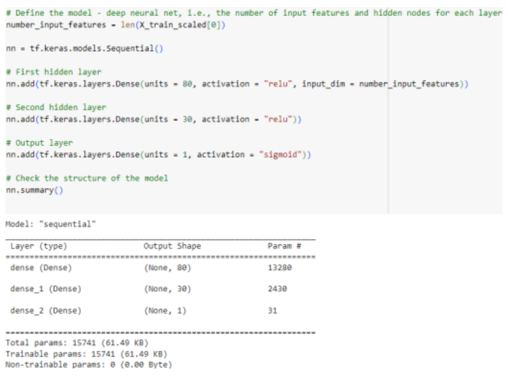
For this optimization, accuracy received was 72.72% which is less than the desired accuracy.



```
# 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}")
```

```
268/268 - 0s - loss: 0.6000 - accuracy: 0.7272 - 321ms/epoch - 1ms/step
Loss: 0.5999968647956848, Accuracy: 0.7272303104400635
```

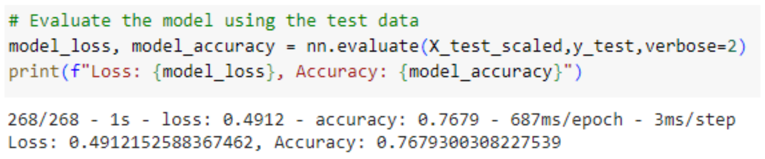
2) The second attempt with the "NAME" column in the dataset



```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer
number_input_features = len(X_train_scaled[0])

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units = 80, activation = "relu", input_dim = number_input_features))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units = 30, activation = "relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units = 1, activation = "sigmoid"))

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

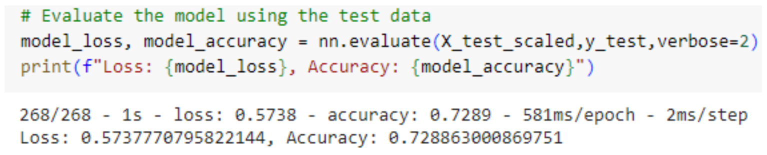
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 80)                13280

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 15741 (61.49 KB)
Trainable params: 15741 (61.49 KB)
Non-trainable params: 0 (0.00 Byte)
```

For this attempt, accuracy received was 76.79%, which is over the target.



```
# 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}")

268/268 - 1s - loss: 0.4912 - accuracy: 0.7679 - 687ms/epoch - 3ms/step
Loss: 0.4912152588367462, Accuracy: 0.7679300308227539
```

3) In the third attempt, epochs number is increased to 150 instead of 100.
   This causes the accuracy of 72.86% for this model.



```
# 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}")

268/268 - 1s - loss: 0.5738 - accuracy: 0.7289 - 581ms/epoch - 2ms/step
Loss: 0.5737770795822144, Accuracy: 0.728863000869751
```

In summary, the analysis suggests that by strategically modifying and enriching the dataset itself, Alphabet Soup can significantly improve the accuracy of their selection tool. This means they can make smarter decisions about who to fund.