REPORT ON NEURAL NETWORK MODEL

OVERVIEW:

The nonprofit foundation Alphabet Soup wants a tool that can help select the applicants for funding with the best chance of success in their ventures. Using machine learning and neural networks, we use the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup. The purpose of this analysis was to utilize said tool to see if it could accurately determine which applicants would be successful if funded by Alphabet Soup.

RESULTS:

- Data Preprocessing
  - What variable(s) are the target(s) for your model?

    Our "Y" variable, which is the "IS_SUCCESSFUL" is our target; the dependent variable.

  - What variable(s) are the features for your model?

    Our "x" variable, the independent variable is represented by the features of our model. Our features are all the other columns save for "IS_SUCCESSFUL" (which was dropped).

  - What variable(s) should be removed from the input data because they are neither targets nor features?

    Initially "EIN" and "NAME" were dropped. However in my final model, I only dropped the "EIN" column.

- Compiling, Training, and Evaluating the Model
  - How many neurons, layers, and activation functions did you select for your neural network model, and why?

    1st Optimization

```
# 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])
hidden_nodes_layer1 = 10
hidden_nodes_layer2 = 20

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units= hidden_nodes_layer1, input_dim= number_input_features, activation='relu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units= hidden_nodes_layer2, activation='relu'))

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

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

2nd Optimization

```
# 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])
hidden_nodes_layer1 = 10
hidden_nodes_layer2 = 20
hidden_nodes_layer3 = 20

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units= hidden_nodes_layer1, input_dim= number_input_features, activation='relu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units= hidden_nodes_layer2, activation='relu'))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units= hidden_nodes_layer3, activation='relu'))

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

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

3rd Optimization

```
# 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])
hidden_nodes_layer1 = 10
hidden_nodes_layer2 = 20
hidden_nodes_layer3 = 30

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units= hidden_nodes_layer1, input_dim= number_input_features, activation='relu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units= hidden_nodes_layer2, activation='relu'))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units= hidden_nodes_layer3, activation='relu'))

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

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

- Were you able to achieve the target model performance?

  1st Optimization

```
# 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 - 1ms/step - accuracy: 0.7242 - loss: 0.5556
Loss: 0.5556439161300659, Accuracy: 0.7241982221603394
```

  2nd Optimization

```
# 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 - 2ms/step - accuracy: 0.7258 - loss: 0.5542
Loss: 0.5542140603065491, Accuracy: 0.7258309125900269
```

  3rd Optimization

```
# 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 - 1ms/step - accuracy: 0.7521 - loss: 0.4983
Loss: 0.4983372688293457, Accuracy: 0.7520699501037598
```

- What steps did you take in your attempts to increase model performance?

  More layers with more nodes were added but that didn't increase the accuracy like I predicted. I then had to adjust the bins and the bin sizes. I didn't drop the "NAME" column but kept it. This finally brought my accuracy up to the 75% threshold.

SUMMARY: Summarize the overall results of the deep learning model. Include a recommendation for how a different model could solve this classification problem, and then explain your recommendation.

A bigger model with more layers and nodes for more power computational power could possibly solve this classification problem however that utilizes a lot of resources and is not cost effective.