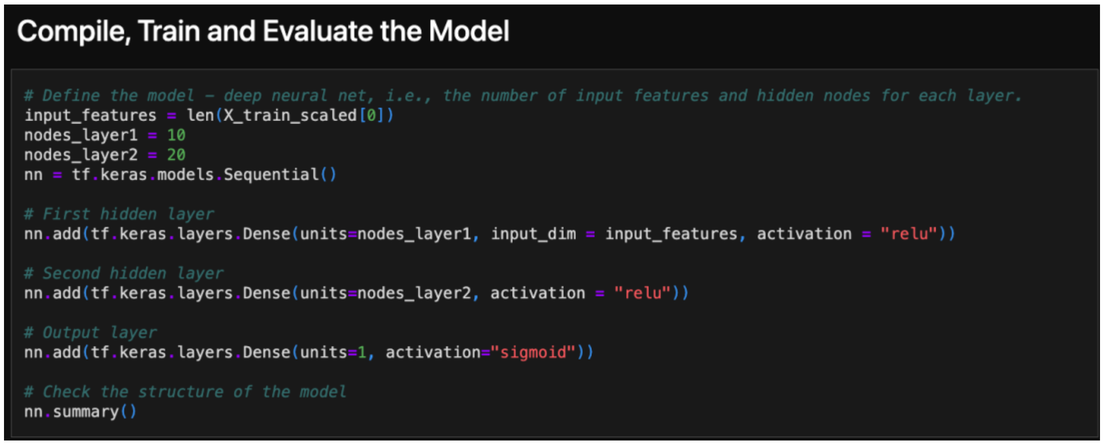
Analysis Report on Alphabet Soup Neural Network Model

1. **Overview**: Alphabet Soup has tasked us with developing a tool that can help it in selecting the funding applicants with the highest chance of success in their ventures. Using Alphabet Soup's CSV data containing more than 34,000 organizations that have received Alphabet Soup's funding over the years, we have developed a binary classifier neural network model to be used as a tool for finding funding applicants that are best likely to succeed. The goal of this analysis is to evaluate the created model for effectiveness in completing this desired task.

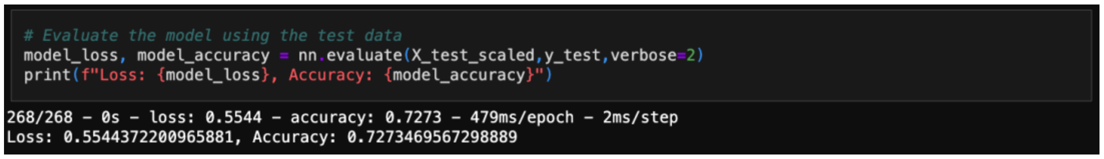
2. **Results**:

- Data Preprocessing
  - What variable(s) are the target(s) for your model?
    - Our target variable, "y", is the "IS_SUCCESSFUL" column.
  - What variable(s) are the features for your model?
    - Our features, "X", would be all other columns aside from the "IS_ SUCCESSFUL" column, which are: "APPLICATION_TYPE", "AFFILIATION", "CLASSIFICATION", "USE_CASE", "ORGANIZATION", "STATUS", "INCOME_AMT", "SPECIAL_CONSIDERATIONS", and "ASK_AMT".
  - What variable(s) should be removed from the input data because they are neither targets nor features?
    - "EIN" and "NAME" were removed from the input data as they were neither targets nor features.

- Compiling, Training, and Evaluating the Model
  - How many neurons, layers, and activation functions did you select for your neural network model, and why?
    - For the initial model, I took a basic approach and used 2 hidden layers and an outer layer. Layer 1 had 10 neurons and “relu” activation, Layer 2 had 20 neurons and “relu” activation, and the Outer Layer had 1 unit, and “sigmoid” activation.
    - 

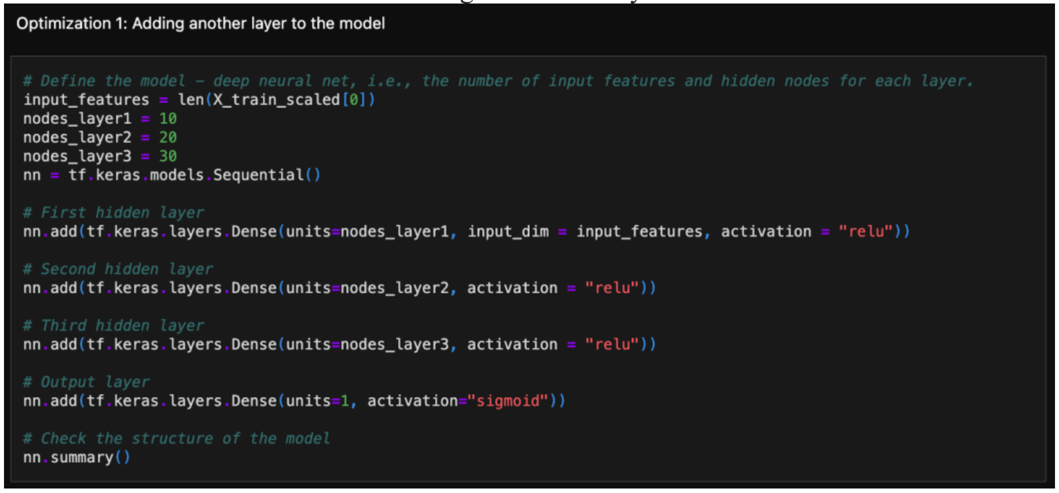
Compile, Train and Evaluate the Model

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = len(X_train_scaled[0])
nodes_layer1 = 10
nodes_layer2 = 20
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer1, input_dim = input_features, activation = "relu"))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer2, 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?
    - I only achieved an accuracy of 0.7273 and did not meet the target performance of 0.75.
    - 

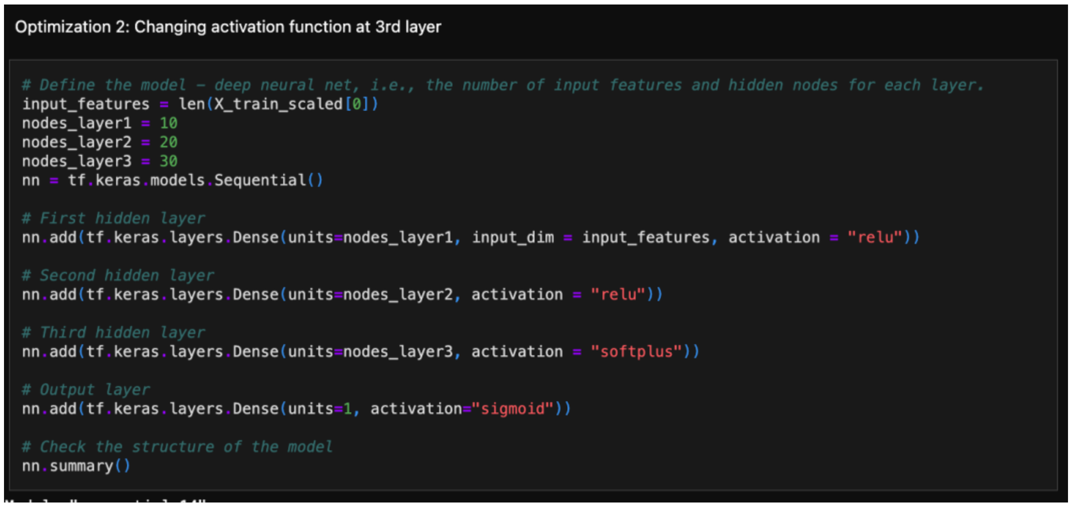
```
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 0s - loss: 0.5544 - accuracy: 0.7273 - 479ms/epoch - 2ms/step
Loss: 0.5544372200965881, Accuracy: 0.7273469567298889
```

- What steps did you take in your attempts to increase model performance?
    - For the first optimization, I added another hidden layer to the model with 30 neurons and “relu” activation function. This brought the accuracy to 0.7276.
    - 

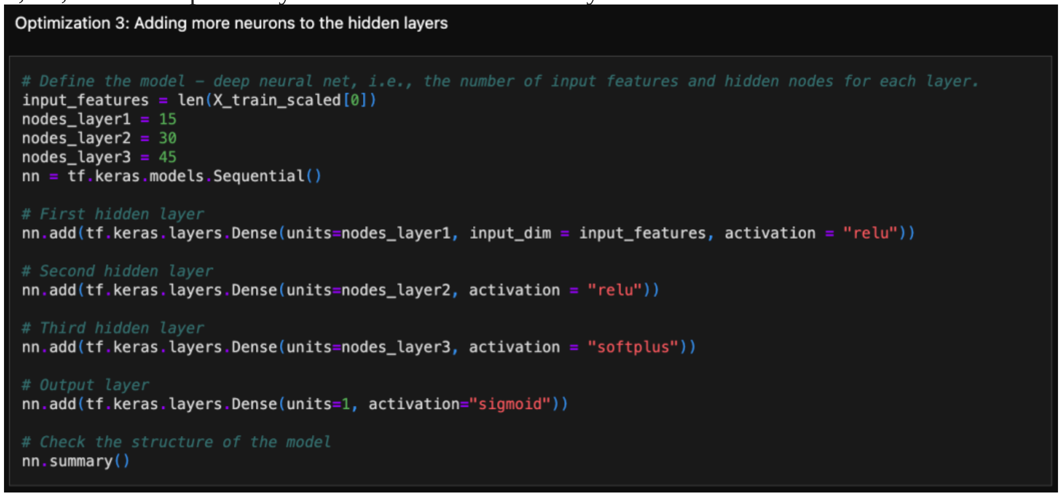
Optimization 1: Adding another layer to the model

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = len(X_train_scaled[0])
nodes_layer1 = 10
nodes_layer2 = 20
nodes_layer3 = 30
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer1, input_dim = input_features, activation = "relu"))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer2, activation = "relu"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer3, activation = "relu"))

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

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

- For the second optimization, I changed the activation function for Layer 3 from “relu” activation function to “softplus” activation function. This brought the accuracy to 0.7299.
- 

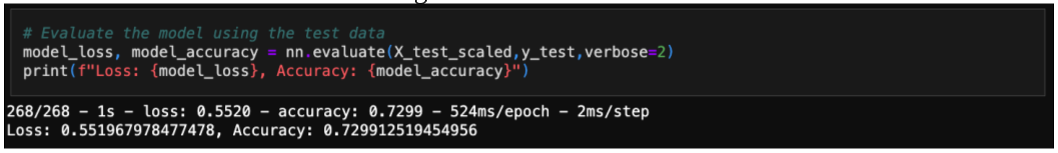
Optimization 2: Changing activation function at 3rd layer

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = len(X_train_scaled[0])
nodes_layer1 = 10
nodes_layer2 = 20
nodes_layer3 = 30
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer1, input_dim = input_features, activation = "relu"))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer2, activation = "relu"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer3, activation = "softplus"))

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

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

- For the third optimization, I added more neurons to the hidden layers – increased by 5, 10, and 15 respectively. This lowered the accuracy to 0.7292.
- 

Optimization 3: Adding more neurons to the hidden layers

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = len(X_train_scaled[0])
nodes_layer1 = 15
nodes_layer2 = 30
nodes_layer3 = 45
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer1, input_dim = input_features, activation = "relu"))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer2, activation = "relu"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=nodes_layer3, activation = "softplus"))

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

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

- The second optimization brought about the best results with an accuracy of 0.7299, which was still below the desired target of 0.75.
- 

```
# 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 - 1s - loss: 0.5520 - accuracy: 0.7299 - 524ms/epoch - 2ms/step
Loss: 0.551967978477478, Accuracy: 0.729912519454956
```

3. **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.

Overall, the model only achieved an accuracy of 0.7299, which was below the desired target of 0.75. This was after several optimization attempts where a third hidden layer was added, the activation function of the third layer was changed, and changing the number of neurons for each of the hidden layers.

I think adding even more layers would help make the model better in solving this classification problem at the cost of additional processing power. Additionally, the “NAME” column was removed from the dataset, but adding the column back in and using it for binning to account for rare occurrences may yield better results.

The model as is should not be used to classify funding applicants. Further optimizations, such as those mentioned above, should be tested before rolling out the model for implementation at Alphabet Soup.