<u>Report on the Neural Network Model</u>

Overview:
The nonprofit foundation Alphabet Soup wants to use a binary classifier tool that can predict whether applicants will be successful if funded by Alphabet Soup. Having a large dataset of about 34,000 organizations that recieved funding from Alphabet Soup over theyears, the purpose of this analysis was to clean (preprocess) the data, compile, train, optimize and evaluate a machine learning model to predict if applicant will be successful upon funding by Alphabet Soup.

Results:
The data was first cleaned by removing irrelevant infomration. The data was then split for training and testing sets. The target variable was then labelled accordingly (IS_SUCCESSFUL with value of 1 for yes and 0 for no). For binning, the classification method was used and several data points were used as a cutoff to bin "rare" variables together with the new value of "Other" for each unique value. Categorical variables were encoded by get_dummies() after checking to see if the binning was successful.

Overall, two hidden layers and one output layer were used for each model respectively. 3 optimization models were executed in total. The 3 models produced 72.6%, 72.4% and 72.1% accuracy over 8,561, 7,999 and 13,953 parameters respectively. The following figures represent layers used for each model along with their evaluation results.

Conclusion:

Overall, the 3 neural network models scored fairly in predicting applicant success rate upon funding by Alphabet Soup. The accuracy varied from 72.1 to 72.6% in 3 different scenaries. Adapting the model to incorporate more rare variables should help in increasing the accuracy of the model.

Model 1

Compile, Train and Evaluate the Model

In [12]:
```
# 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])

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=90, activation="tanh", input_dim=input_features))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=50, activation="relu"))

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

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

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 90)                3960

 dense_1 (Dense)             (None, 50)                4550

 dense_2 (Dense)             (None, 1)                 51

=================================================================
Total params: 8,561
Trainable params: 8,561
Non-trainable params: 0
```

In [15]:
```
# 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: 0.5750 - accuracy: 0.7259 - 591ms/epoch - 2ms/step
Loss: 0.5750189423561096, Accuracy: 0.7259474992752075
```

Model 2

Compile, Train and Evaluate the Model

In [18]:

```
# 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])

nn = tf.keras.models.Sequential()

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

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

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=64, activation="relu"))

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

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

```
Model: "sequential_1"
_________________________________________________________________
click to expand output; double click to hide output  Output Shape    Param #
=================================================================
 dense_3 (Dense)             (None, 80)                3520

 dense_4 (Dense)             (None, 30)                2430

 dense_5 (Dense)             (None, 64)                1984

 dense_6 (Dense)             (None, 1)                 65

=================================================================
Total params: 7,999
Trainable params: 7,999
Non-trainable params: 0
_________________________________________________________________
```

In [21]:

```
# 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: 0.5985 - accuracy: 0.7240 - 526ms/epoch - 2ms/step
Loss: 0.5985429286956787, Accuracy: 0.7239649891853333
```

Model 3

Compile, Train and Evaluate the Model

In [22]:
```
# 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])

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=128, activation="relu", input_dim=input_features))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=64, activation="relu"))

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

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

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 128)               5632

 dense_8 (Dense)             (None, 64)                8256

 dense_9 (Dense)             (None, 1)                 65

=================================================================
Total params: 13,953
Trainable params: 13,953
Non-trainable params: 0
_________________________________________________________________
```

In [25]:
```
# 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: 0.6827 - accuracy: 0.7217 - 565ms/epoch - 2ms/step
Loss: 0.6826791167259216, Accuracy: 0.7217492461204529
```