Overview:
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. With your knowledge of machine learning and neural networks, you'll 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.

Data Preprocessing:

For data preprocessing, the metrics of EIN and Name were appropriately removed from the dataset provided by Alphabet Soup. All the remaining metrics included in the dataset were considered when processing the model with Application Type and Classification being utilized as features.

Compiling, Training, and Evaluating the Model:

When utilizing Neural Network for each model, the original attempt was set with two initial layers. The optimization attempt was then set to three layers in an attempt to achieve the accuracy of over 75%. Unfortunately, I was unable to achieve the target model performance and could only achieve an accuracy of less than 73%.

```python
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_features = X_train_scaled.shape[1]
hidden_layer1 = 90
hidden_layer2 = 45

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer1, input_dim=number_features, activation='relu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer2, activation='relu'))

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

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

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_45 (Dense)            (None, 90)                4140

 dense_46 (Dense)            (None, 45)                4095

 dense_47 (Dense)            (None, 1)                 46

=================================================================
Total params: 8,281
Trainable params: 8,281
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 89/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5312 - accuracy: 0.7422 - val_loss: 0.5679 - val_accuracy: 0.7282
Epoch 90/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5289 - accuracy: 0.7430 - val_loss: 0.5687 - val_accuracy: 0.7298
Epoch 91/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5289 - accuracy: 0.7427 - val_loss: 0.5647 - val_accuracy: 0.7293
Epoch 92/100
668/668 [==============================] - 2s 4ms/step - loss: 0.5289 - accuracy: 0.7425 - val_loss: 0.5662 - val_accuracy: 0.7305
Epoch 93/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5288 - accuracy: 0.7431 - val_loss: 0.5750 - val_accuracy: 0.7273
Epoch 94/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5293 - accuracy: 0.7428 - val_loss: 0.5691 - val_accuracy: 0.7305
Epoch 95/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5288 - accuracy: 0.7420 - val_loss: 0.5666 - val_accuracy: 0.7293
Epoch 96/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5292 - accuracy: 0.7424 - val_loss: 0.5677 - val_accuracy: 0.7305
Epoch 97/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5296 - accuracy: 0.7420 - val_loss: 0.5672 - val_accuracy: 0.7291
Epoch 98/100
668/668 [==============================] - 3s 4ms/step - loss: 0.5286 - accuracy: 0.7423 - val_loss: 0.5674 - val_accuracy: 0.7282
Epoch 99/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5287 - accuracy: 0.7429 - val_loss: 0.5684 - val_accuracy: 0.7286
Epoch 100/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5289 - accuracy: 0.7418 - val_loss: 0.5686 - val_accuracy: 0.7295
```

244]
```python
# 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: 0.5786 - accuracy: 0.7259 - 431ms/epoch - 2ms/step
Loss: 0.5786498188972473, Accuracy: 0.7259474992752075
```

```python
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_features = X_train_scaled.shape[1]
hidden_layer1 = 10
hidden_layer2 = 20
hidden_layer3 = 30

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer1, input_dim=number_features, activation='relu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer2, activation='relu'))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer3, 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, 10)                460

 dense_1 (Dense)             (None, 20)                220

 dense_2 (Dense)             (None, 30)                630

 dense_3 (Dense)             (None, 1)                 31

=================================================================
Total params: 1,341
Trainable params: 1,341
Non-trainable params: 0
```

```
Epoch 89/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5312 - accuracy: 0.7422 - val_loss: 0.5679 - val_accuracy: 0.7282
Epoch 90/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5289 - accuracy: 0.7430 - val_loss: 0.5687 - val_accuracy: 0.7298
Epoch 91/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5289 - accuracy: 0.7427 - val_loss: 0.5647 - val_accuracy: 0.7293
Epoch 92/100
668/668 [==============================] - 2s 4ms/step - loss: 0.5289 - accuracy: 0.7425 - val_loss: 0.5662 - val_accuracy: 0.7305
Epoch 93/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5288 - accuracy: 0.7431 - val_loss: 0.5750 - val_accuracy: 0.7273
Epoch 94/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5293 - accuracy: 0.7428 - val_loss: 0.5691 - val_accuracy: 0.7305
Epoch 95/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5288 - accuracy: 0.7420 - val_loss: 0.5666 - val_accuracy: 0.7293
Epoch 96/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5292 - accuracy: 0.7424 - val_loss: 0.5677 - val_accuracy: 0.7305
Epoch 97/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5296 - accuracy: 0.7420 - val_loss: 0.5672 - val_accuracy: 0.7291
Epoch 98/100
668/668 [==============================] - 3s 4ms/step - loss: 0.5286 - accuracy: 0.7423 - val_loss: 0.5674 - val_accuracy: 0.7282
Epoch 99/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5287 - accuracy: 0.7429 - val_loss: 0.5684 - val_accuracy: 0.7286
Epoch 100/100
668/668 [==============================] - 2s 3ms/step - loss: 0.5289 - accuracy: 0.7418 - val_loss: 0.5686 - val_accuracy: 0.7295
```

```python
[244] # 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: 0.5786 - accuracy: 0.7259 - 431ms/epoch - 2ms/step
Loss: 0.5786498188972473, Accuracy: 0.7259474992752075
```

Summary

I believe more layers could be utilized in future attempts to potentially predict the information based on the models utilized.