Deep Learning Challenge: Charity

Overview:

Create an algorithm to predict the success of funding for applicants. Using machine learning and neural networks, find a binary classifier that can predict the success of applicants if funded by Alphabet Soup.

Results:

I ended up dropping EIN and NAME because they were the two columns that I did not consider features for the model. However, I did add NAME back in the optimized version. I then split the data for training and testing sets.

▾ 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.
number_input_features = len( X_train_scaled[0])
hidden_nodes_layer1=7
hidden_nodes_layer2=14
hidden_nodes_layer3=21

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()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 350

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 477
Trainable params: 477
Non-trainable params: 0
```

As shown in the figure above, I was able to discover three layers for each model after Neural Networks and the number of hidden nodes correspond to the number of features. In the bottom of the figure, the data tells us that 477 parameters were created.

```
[ ]  # 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: 0.5536 - accuracy: 0.7299 - 409ms/epoch - 2ms/step
Loss: 0.5535799264907837, Accuracy: 0.729912519454956
```

However, the accuracy score is 73% which is not very pleasing to most people. The requirement is at least 75% so I tried adding back names in the optimization to see if we can get a closer score.

```
nn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 3171

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 3,298
Trainable params: 3,298
Non-trainable params: 0
_________________________________________________________________
```

In the optimized version, I was able to get many more parameters when adding back the column NAME. In the end, the accuracy score went up by 6 percent, which surpasses the 75 percent mark.