Report on the Neural Network Model

- **Overview** of the analysis: Explain the purpose of this analysis.

  These neural networks are an advanced form of machine learning that recognizes patterns and features of input data and provides a clear quantitative output. The neural network model is a structure composed of several connected perceptions that learn from input data to produce an output. The target of the analysis is the column that evaluates if money was used successfully. This analysis attempts to answer that question: Was money used effectively?

- Data Preprocessing
  - What variable(s) are the target(s) for your model?
    - The target for this model is the “IS_SUCCESSFUL” column.
  - What variable(s) are the features for your model?
    - The variables are the remaining columns, minus the columns mentioned in the point below.
  - What variable(s) should be removed from the input data because they are neither targets nor features?
    - Early in the code, we drop the “EIN” and “NAME: columns. These columns are non-beneficial for the analysis.

```
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df.drop(columns = ['EIN', 'NAME'], inplace = True)
```

- Compiling, Training, and Evaluating the Model
  - How many neurons, layers, and activation functions did you select for your neural network model, and why?
    - I iterated through multiple combinations of neurons laters, and activation functions.
  - Were you able to achieve the target model performance?
    - I was not able to achieve the target model performance. Running four different simulations brought it close, but not quite there.
  - What steps did you take in your attempts to increase model performance?
    - I shifted the number of neurons on the different levels, as well as the cutoff values and number of epochs.

```python
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.

nn_model = tf.keras.models.Sequential()

# First hidden layer
nn_model.add(tf.keras.layers.Dense(units=120, activation="relu", input_dim=X_train.shape[1]))

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

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

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

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 120)               6120

 dense_13 (Dense)            (None, 30)                3630

 dense_14 (Dense)            (None, 1)                 31

=================================================================
Total params: 9781 (38.21 KB)
Trainable params: 9781 (38.21 KB)
Non-trainable params: 0 (0.00 Byte)
```

- **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.

I was not able to achieve the target of 75%.

Test 1

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

```
268/268 - 0s - loss: 0.5882 - accuracy: 0.7290 - 334ms/epoch - 1ms/step
Loss: 0.5882247686386108, Accuracy: 0.7289795875549316
```

Test 4

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

```
268/268 - 1s - loss: 0.5824 - accuracy: 0.7285 - 549ms/epoch - 2ms/step
Loss: 0.5823935866355896, Accuracy: 0.7285131216049194
```

I would recommend continuing to explore the adjustment available in altering the hidden layers to look to optimize for the best possible outcome.