Deep Learning Model for Alphabet Soup Performance Report

1. **Overview of the Analysis**

The purpose of this analysis is to create a deep learning model to help Alphabet Soup predict whether an applicant will be successful in their venture based on several features provided in the dataset. By building a binary classification model, we aim to guide Alphabet Soup in making informed funding decisions, increasing the likelihood of success among funded applicants.

2. **Results**

Data Preprocessing

Target Variable: the target variable for this analysis is IS_SUCCESSFUL, which indicates whether an applicant was successful after receiving funding from Alphabet Soup.

Feature Variables:

The features include APPLICATION_TYPE, AFFILIATION, CLASSIFICATION, USE_CASE, ORGANIZATION, STATUS, INCOME_AMT, SPECIAL_CONSIDERATIONS, ASK_AMT

Removed Variables:

EIN and NAME columns were removed from the dataset as they are identification columns and do not contribute to the predictive model.

Compiling, Training, and Evaluating the Model

Neurons, Layers, and Activation Functions:

The first neural network architecture was as follows:

First hidden layer: 80 neurons with the relu activation function.

Second hidden layer: 30 neurons with the relu activation function.

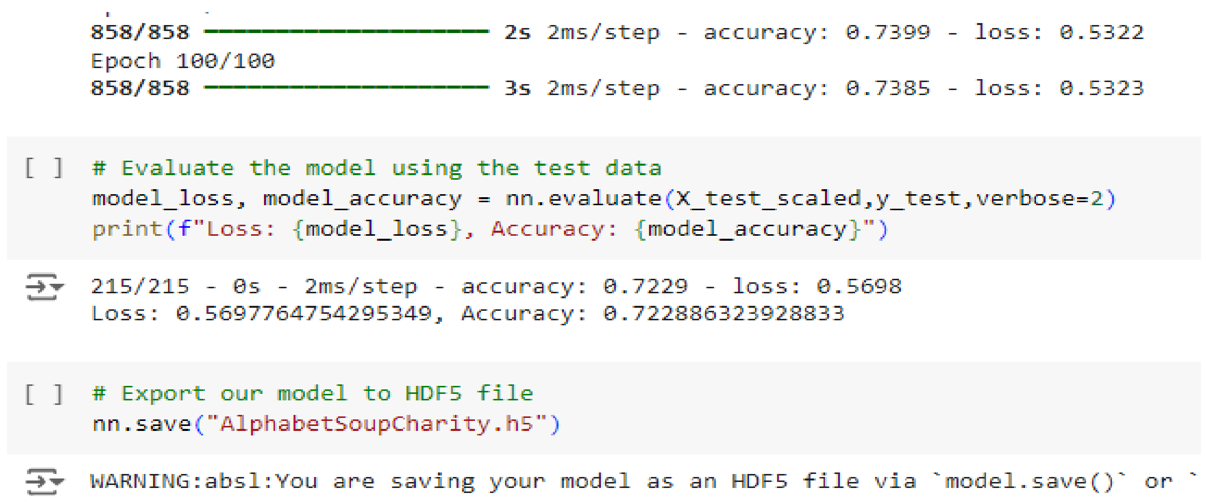
Output layer: 1 neuron with the sigmoid activation function for binary classification.

Epochs: 100

The relu activation function was chosen for the hidden layers because it helps in capturing non-linear relationships, while the sigmoid function was used in the output layer since this is a binary classification problem.

Model Performance:

The model achieved an accuracy of 72.29%, which fell short of the target accuracy of 75%.



```
858/858 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7399 - loss: 0.5322
Epoch 100/100
858/858 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7385 - loss: 0.5323
```

```
[ ] # 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}")
```

```
215/215 - 0s - 2ms/step - accuracy: 0.7229 - loss: 0.5698
Loss: 0.5697764754295349, Accuracy: 0.722886323928833
```

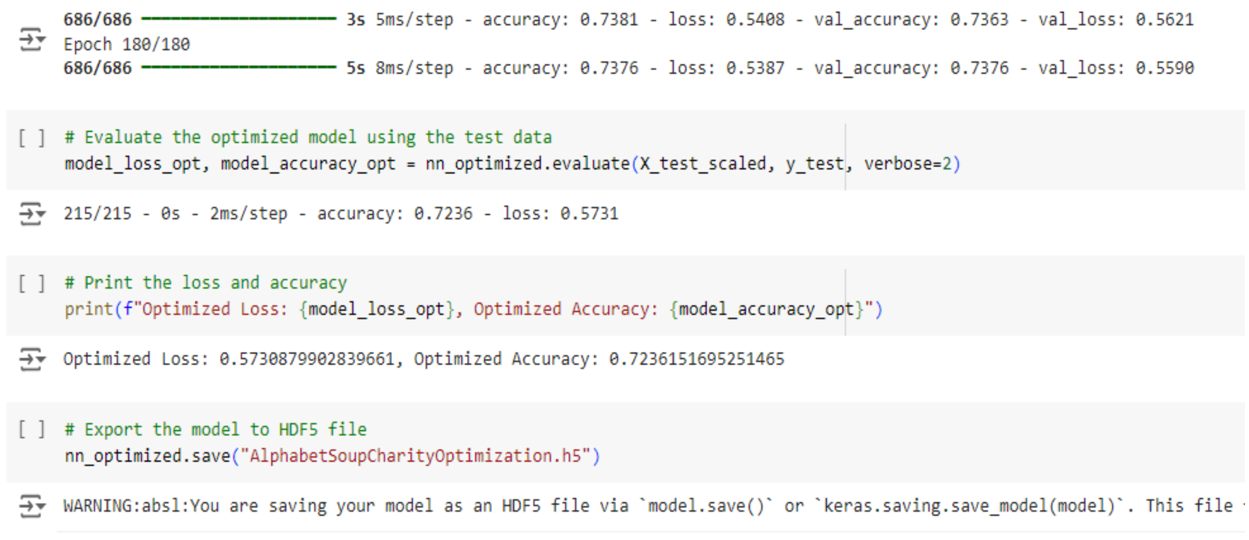
```
[ ] # Export our model to HDF5 file
    nn.save("AlphabetSoupCharity.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `
```

Steps Taken to Improve Performance:

Several optimization techniques were applied to improve the model's performance:

1) Adding more neurons to each hidden layer to increase the model's learning capacity and increased the number of epochs. The result is slightly increased to 72.36%.



```
686/686 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7381 - loss: 0.5408 - val_accuracy: 0.7363 - val_loss: 0.5621
Epoch 180/180
686/686 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.7376 - loss: 0.5387 - val_accuracy: 0.7376 - val_loss: 0.5590
```

```
[ ] # Evaluate the optimized model using the test data
    model_loss_opt, model_accuracy_opt = nn_optimized.evaluate(X_test_scaled, y_test, verbose=2)
```

```
215/215 - 0s - 2ms/step - accuracy: 0.7236 - loss: 0.5731
```

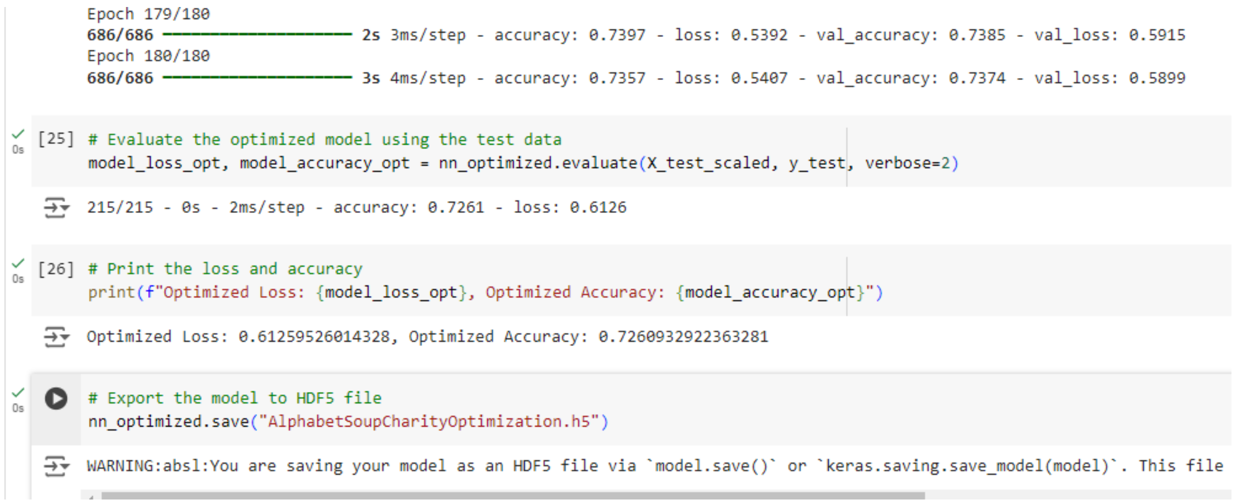
```
[ ] # Print the loss and accuracy
    print(f"Optimized Loss: {model_loss_opt}, Optimized Accuracy: {model_accuracy_opt}")
```

```
Optimized Loss: 0.5730879902839661, Optimized Accuracy: 0.7236151695251465
```

```
[ ] # Export the model to HDF5 file
    nn_optimized.save("AlphabetSoupCharityOptimization.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file
```

2) Created more bins for rare occurrences in columns, the result is slightly increased to 72.61%.



```
Epoch 179/180
686/686 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7397 - loss: 0.5392 - val_accuracy: 0.7385 - val_loss: 0.5915
Epoch 180/180
686/686 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7357 - loss: 0.5407 - val_accuracy: 0.7374 - val_loss: 0.5899
```

```
[25] # Evaluate the optimized model using the test data
     model_loss_opt, model_accuracy_opt = nn_optimized.evaluate(X_test_scaled, y_test, verbose=2)
```

```
215/215 - 0s - 2ms/step - accuracy: 0.7261 - loss: 0.6126
```

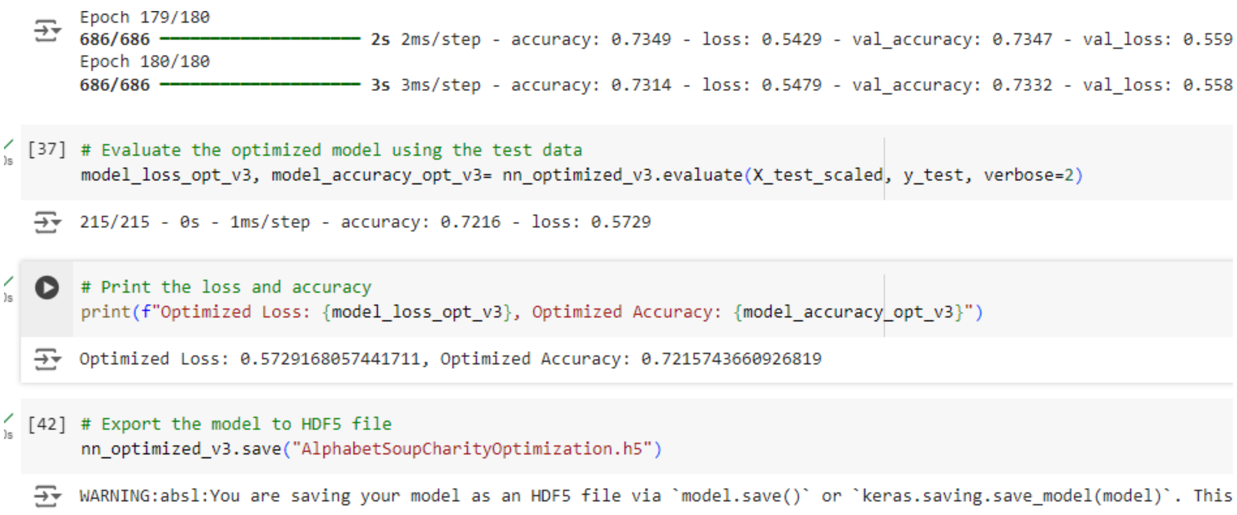
```
[26] # Print the loss and accuracy
     print(f"Optimized Loss: {model_loss_opt}, Optimized Accuracy: {model_accuracy_opt}")
```

```
Optimized Loss: 0.61259526014328, Optimized Accuracy: 0.7260932922363281
```

```
# Export the model to HDF5 file
nn_optimized.save("AlphabetSoupCharityOptimization.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file
```

3) Adjusting the activation functions by experimenting with tanh, selu, and relu, the result slightly decreased to 72.16%



```
Epoch 179/180
686/686 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7349 - loss: 0.5429 - val_accuracy: 0.7347 - val_loss: 0.559
Epoch 180/180
686/686 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.7314 - loss: 0.5479 - val_accuracy: 0.7332 - val_loss: 0.558
```

```
[37] # Evaluate the optimized model using the test data
     model_loss_opt_v3, model_accuracy_opt_v3= nn_optimized_v3.evaluate(X_test_scaled, y_test, verbose=2)
```

```
215/215 - 0s - 1ms/step - accuracy: 0.7216 - loss: 0.5729
```

```
# Print the loss and accuracy
print(f"Optimized Loss: {model_loss_opt_v3}, Optimized Accuracy: {model_accuracy_opt_v3}")
```

```
Optimized Loss: 0.5729168057441711, Optimized Accuracy: 0.7215743660926819
```

```
[42] # Export the model to HDF5 file
     nn_optimized_v3.save("AlphabetSoupCharityOptimization.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This
```

Despite trying several optimization techniques, the model did not reach the desired performance.

Summary

The deep learning model achieved a reasonable accuracy of 72.29%, but did not meet the 75% target performance. Several optimization techniques were applied, such as increasing neurons, adding layers, adjusting activation functions, but the accuracy gains remained limited. To further improve the model, recommend exploring other machine learning models for this classification problem such as Random Forest: A random forest classifier may perform better with this dataset, as it handles categorical features well and can capture complex feature interactions. Also, we could try Gradient Boosting method to help improve model performance, as they are known to work well for structured tabular data.