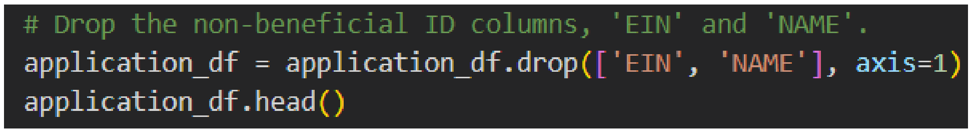
Deep Learning Project

Charity Funding Predictor

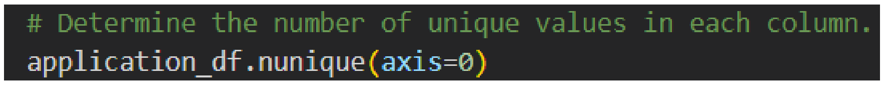
By: Robert Harvey

Preprocessing

The first step in the deep learning process was to import the sklearn, pandas, and tensorflow dependencies, and to read the csv file into a DataFrame. I uploaded the "charity_data.csv" file to Google Collab for this step. After creating a DataFrame, I dropped the non-beneficial ID columns (EID and NAME) and determined the unique number of values.

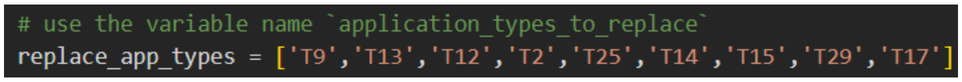


```
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df = application_df.drop(['EIN', 'NAME'], axis=1)
application_df.head()
```

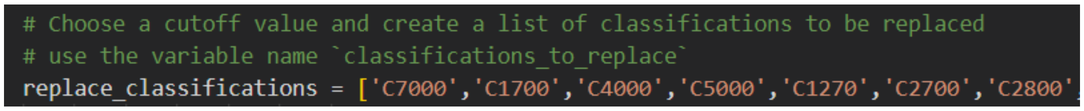


```
# Determine the number of unique values in each column.
application_df.nunique(axis=0)
```

The next step was to determine binning for both APPLICATION AND CLASSIFICATION and drop whatever values and data wouldn't be relevant.



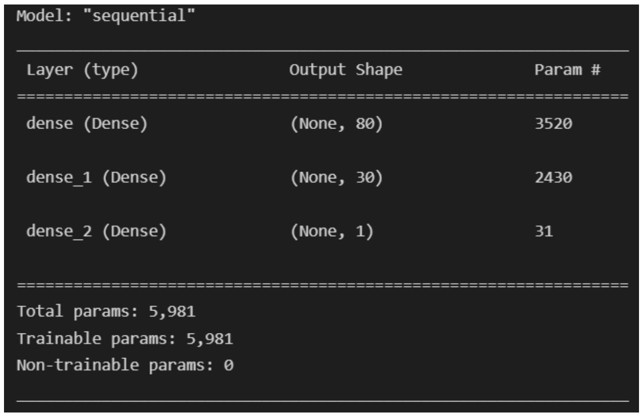
```
# use the variable name `application_types_to_replace`
replace_app_types = ['T9','T13','T12','T2','T25','T14','T15','T29','T17']
```



```
# Choose a cutoff value and create a list of classifications to be replaced
# use the variable name `classifications_to_replace`
replace_classifications = ['C7000','C1700','C4000','C5000','C1270','C2700','C2800'
```

Compile, Train and Evaluate the Model

Now it was time to deploy neural networks, including adding the first hidden layer, second hidden layer, and output layer. The results of the parameters is below:



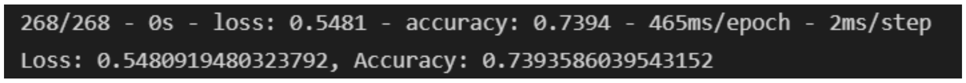
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 80)                3520

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 5,981
Trainable params: 5,981
Non-trainable params: 0
_________________________________________________________________
```

After compiling and training the model, the accuracy results were less than idea (Accuracy=73.93586%l:



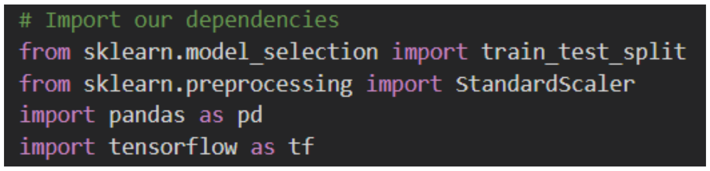
```
268/268 - 0s - loss: 0.5481 - accuracy: 0.7394 - 465ms/epoch - 2ms/step
Loss: 0.5480919480323792, Accuracy: 0.7393586039543152
```

Step 3: Optimize the Model

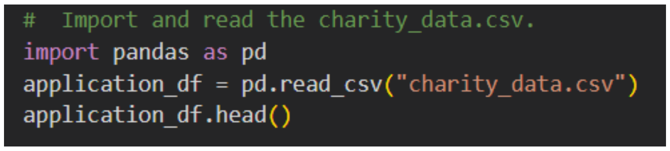
Retrain the Model for >75% Accuracy

Preprocessing

The first step in the deep learning process was to import the sklearn, pandas, and tensorflow dependencies, and to read the csv file into a DataFrame. I uploaded the “charity_data.csv” file to Google Collab for this step. After creating a DataFrame, I dropped the non-beneficial ID columns (EID only) and determined the unique number of values.

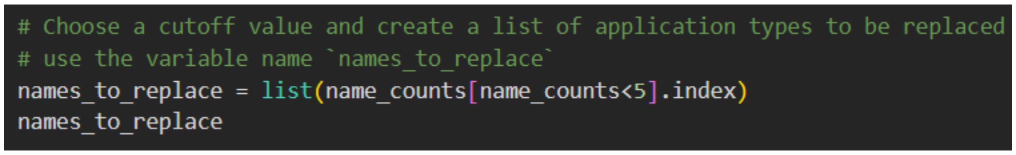


```
# Import our dependencies
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import tensorflow as tf
```

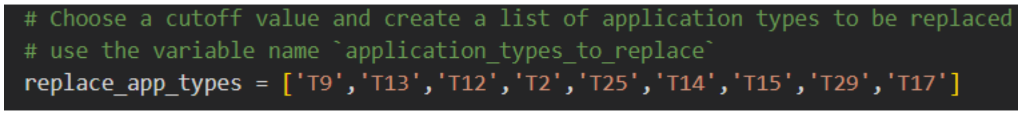


```
#  Import and read the charity_data.csv.
import pandas as pd
application_df = pd.read_csv("charity_data.csv")
application_df.head()
```

The next step was to determine binning for both APPLICATION AND CLASSIFICATION and drop whatever values and data wouldn’t be relevant. For this version of the model, I added NAME back into the model.



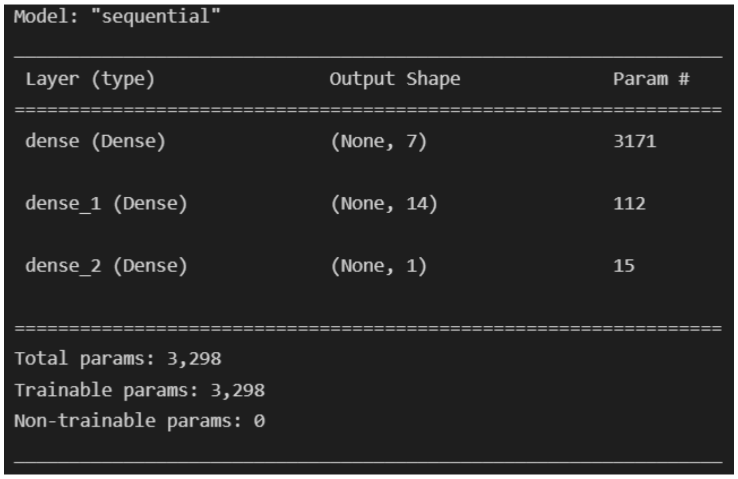
```
# Choose a cutoff value and create a list of application types to be replaced
# use the variable name `names_to_replace`
names_to_replace = list(name_counts[name_counts<5].index)
names_to_replace
```



```
# Choose a cutoff value and create a list of application types to be replaced
# use the variable name `application_types_to_replace`
replace_app_types = ['T9','T13','T12','T2','T25','T14','T15','T29','T17']
```

Compile, Train and Evaluate the Model

Now it was time to deploy neural networks, including adding the first hidden layer, second hidden layer, and output layer. The results of the parameters is below:



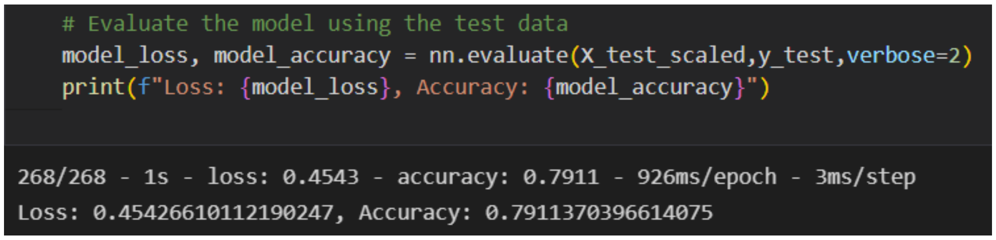
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 3171

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

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

After compiling and training the model, the accuracy results were much better than the pevious model, and above the 75% accuracy threshold (Accuracy=79.11370%l:



```
    # 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: 0.4543 - accuracy: 0.7911 - 926ms/epoch - 3ms/step
Loss: 0.45426610112190247, Accuracy: 0.7911370396614075
```

Conclusion

Even though the Optimized model had fewer Total and Trainable parameters when executing the model - 5,981 for the Original model vs. 3,298 for the Optimized model – the Optimized model (79.11%) was more accurate than the Original model (73.94%). My initial conclusion is that the structure of the neural networks when running the model favored the Optimized model because the number of second layer nodes outnumbered the number of first layer nodes. The original model had (80) first layer nodes and only (30) second layer nodes, while the Optimized model had (7) first layer nodes and (14) second layer nodes. The original model may have over-complicated the neural network.