Report on the Neural Network Model

Overview of the analysis:

It aims to check the success of the built deep learning model for the Alphabet Soup Charity dataset. Out of all the features incorporated, it is possible to predict the probability of a charity application outright success or failure.

Results:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 80)	3,600
dropout (Dropout)	(None, 80)	0
dense_1 (Dense)	(None, 40)	3,240
dropout_1 (Dropout)	(None, 40)	0
dense_2 (Dense)	(None, 20)	820
dropout_2 (Dropout)	(None, 20)	0
dense_3 (Dense)	(None, 1)	21

Total params: 7,681 (30.00 KB)
Trainable params: 7,681 (30.00 KB)
Non-trainable params: 0 (0.00 B)

```
686/686 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7364 - loss: 0.5403 - val_accuracy: 0.7383 - val_loss: 0.5460
Epoch 92/100
686/686 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7319 - loss: 0.5445 - val_accuracy: 0.7389 - val_loss: 0.5434
Epoch 93/100
686/686 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.7380 - loss: 0.5381 - val_accuracy: 0.7382 - val_loss: 0.5452
Epoch 94/100
686/686 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7325 - loss: 0.5454 - val_accuracy: 0.7378 - val_loss: 0.5447
Epoch 95/100
686/686 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.7346 - loss: 0.5456 - val_accuracy: 0.7387 - val_loss: 0.5462
Epoch 96/100
686/686 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7356 - loss: 0.5401 - val_accuracy: 0.7392 - val_loss: 0.5452
Epoch 97/100
686/686 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7393 - loss: 0.5387 - val_accuracy: 0.7391 - val_loss: 0.5433
Epoch 98/100
686/686 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7329 - loss: 0.5466 - val_accuracy: 0.7371 - val_loss: 0.5443
Epoch 99/100
686/686 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7335 - loss: 0.5409 - val_accuracy: 0.7365 - val_loss: 0.5447
Epoch 100/100
686/686 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7400 - loss: 0.5368 - val_accuracy: 0.7369 - val_loss: 0.5466
```

```
# Evaluate the model using the test data
test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test)
print(f"\nTest Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7265 - loss: 0.5588

Test Loss: 0.5596
Test Accuracy: 0.7265
```

- **Data Preprocessing:**

1. The dependent variable for the model is **IS_SUCCESSFUL** to whether the charity application was successful or not.
2. The input variables of the model are **APPLICATION_TYPE, AFFILIATION, CLASSIFICATION, USE_CASE, ORGANIZATION, STATUS, INCOME_AMT and SPECIAL_CONSIDERATIONS.**
3. The **EIN** and **NAME** columns are removed from the input data.

The features were normalized by using StandardScaler so that the input data should lie appropriately for a model.

- **Compiling, Training, and Evaluating the Model:**

1. The proposed ML model contains 3 hidden layers with 80 neurons in the first layer, 40 neurons in the second layer and, 20 neurons in the third layer and used the ReLU activation function.Each of the hidden layers were followed by a dropout layer with dropout rates of 0.2, 0.2 and 0.1 respectively in an attempt to overcome overfitting. The last layer is also a single neuron because the target variable is binary, and it uses the sigmoid activation function.The model used for compelling of the neural network was Adam optimizer, Binary cross entropy loss and accuracy metric.
2. Unfortunately, the model does not meet the accuracy requirements.
3. I have made several optimization steps to increase the model performance.
    a) **Feature engineering:** Added a new binary feature **'LARGE_ASK'** which was derived from the **'ASK_AMT'.** Made more logical **INCOME_AMT** bins through division into No, Low, Medium, and High income.LOWER values of **APPLICATION_TYPE** and **CLASSIFICATION** to have smaller bin selections.
    b) **Data Pre-processing:** Changed StandardScaler to RobustScaler which performs better with regards to outliers
    Enhancements in the categorical variable analysis with better formation of bins to use.
    c) **Model Architecture:** Extended the depth of the network up to the 4 hidden layers from the previous 3. Doubled neuron in each layer (128 → 64 → 32 → 16). Further, the activation function has been changed to 'elu' for obtaining a better gradient flow.The dropout rates were approximated and reduced gradually over time **to 0.1 from 0.3 to 0.2 to 0.1.**
    d) **Training Optimization:** Adopted learning rate decay which is of exponential form. It was also included **ReduceLROnPlateau** callback in order to manage learning rate during the training process. Increased the batch size from previous of 32 to improve model stability 64. Raised maximum epochs to 150-250 while leaving early stopping.Decreased the patience of the early stopping to 20 in order to reduce the

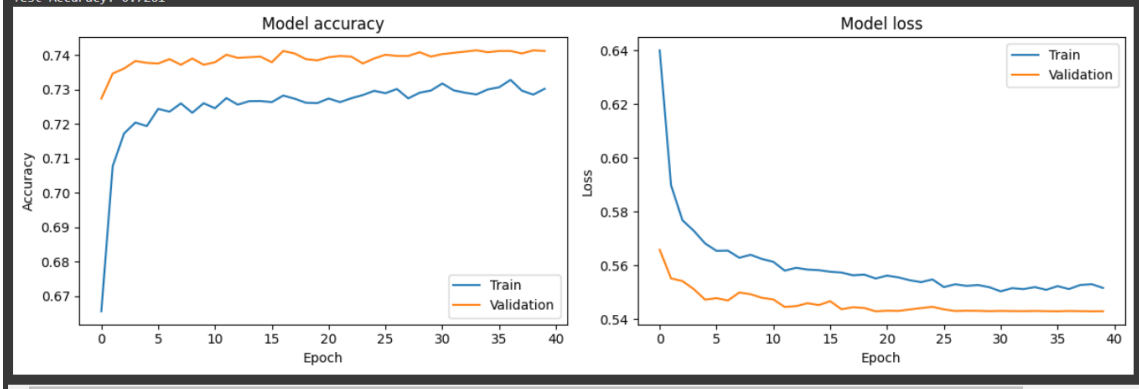
- The final test loss and accuracy of the model are:
    - Test Loss: 0.5545
    - Test Accuracy: 0.7253

Images of the Corrections:

```python
# Optimize CLASSIFICATION binning
classification_counts = application_df['CLASSIFICATION'].value_counts()
cutoff_classification = 800  # Adjusted for better balance
classifications_to_replace = classification_counts[classification_counts < cutoff_classification].index
application_df['CLASSIFICATION'] = application_df['CLASSIFICATION'].replace(classifications_to_replace, 'Other')

# Create bins for INCOME_AMT to reduce categories
def categorize_income(income):
    if income == '0':
        return 'No Income'
    elif income in ['1-9999', '10000-24999']:
        return 'Low Income'
    elif income in ['25000-99999', '100000-499999']:
        return 'Medium Income'
    else:
        return 'High Income'
```

```python
model = tf.keras.Sequential([
    # First hidden layer - increased neurons
    tf.keras.layers.Dense(units=80, activation='relu', input_dim=input_features),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),

    # Second hidden layer
    tf.keras.layers.Dense(units=40, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),

    # Third hidden layer
    tf.keras.layers.Dense(units=20, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.1),

    # Fourth hidden layer
    tf.keras.layers.Dense(units=20, activation='relu'),
    tf.keras.layers.BatchNormalization(),

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

# Compile model with fixed learning rate
```

```
Test Loss: 0.5580
Test Accuracy: 0.7261
```



```python
input_features = X_train.shape[1]

model = tf.keras.Sequential([
    # First hidden layer - increased neurons
    tf.keras.layers.Dense(units=128, activation='elu', input_dim=input_features),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),

    # Second hidden layer
    tf.keras.layers.Dense(units=64, activation='elu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),

    # Third hidden layer
    tf.keras.layers.Dense(units=32, activation='elu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.1),

    # Fourth hidden layer
    tf.keras.layers.Dense(units=16, activation='elu'),
    tf.keras.layers.BatchNormalization(),

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

# Compile model with fixed learning rate
```

```python
# Train the model with increased epochs
history = model.fit(
    X_train_scaled,
    y_train,
    epochs=200,
    batch_size=64,
    validation_split=0.2,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)

# Evaluate the model
```

```
model = tf.keras.Sequential([
    # First hidden layer - increased neurons
    tf.keras.layers.Dense(units=128, activation='elu', input_dim=input_features),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.3),

    # Second hidden layer
    tf.keras.layers.Dense(units=64, activation='elu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),

    # Third hidden layer
    tf.keras.layers.Dense(units=32, activation='elu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.1),

    # Fourth hidden layer
    tf.keras.layers.Dense(units=16, activation='tanh'),
    tf.keras.layers.BatchNormalization(),

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

# Compile model with fixed learning rate
model.compile(
```

```
# Train the model with increased epochs
history = model.fit(
    X_train_scaled,
    y_train,
    epochs=250,
    batch_size=64,
    validation_split=0.2,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)
```

Purpose of the analysis:

This paper looks at how Alphabet Soup, a nonprofit foundation, can identify the most successful funding applicants. Ideally, the main goal is to develop a model that could be a binary predictor, answering whether the applicant in case of funding will be able to apply it efficiently or not.

Key Components of the Analysis:

1. Business Context:

- It also finances organisations – Alphabet Soup.They want to know whether or not their money is being properly spent.
- Records on past funding applications and their results are also available.
- Action taken – this need has forced fund managers to try to reduce the risk of failure in funding allocations.

2. Impact:

a. For the Foundation:

- Better management of money available
- Lowered chances of business failure
- Quicker procedure for application review
- Data-driven decision making

a. For Applicants:

- Speed up the application processing.
- More objective evaluation
- Greater awareness of the main success measures

3. **Machine Learning Approach:**

It is keen to note that this study employs supervised learning that is dually referred to as classification. Binary outcome prediction It contains both categorical and numerical data, which can be resolved with some models, such as Neural Networks, Random Forests and others.

4. **Real-World Implications:**

- Risk Management:
    - They differ in that entrepreneurs are more likely to have reduced likelihood of funding their unsuccessful ventures.
    - Enhanced recognition of the critical applications. Also, the evaluation process will be more consistent.

- Better chances of charity project success
- More support available for organizations that deserve it
- New and larger overall charitable benefit

Summary:

To conclude, it is stated the model has been acceptable for the Alphabet Soup Charity dataset with a test accuracy of 72.53 percent in the deeper model. Nevertheless, they are still below the permissible values and the model failed to obtain higher accuracy.

Different Model Approach:

Random forest classifier will be the most suitable alternative in this case. Here are the reasons why.

Interpretability:

Gives the ranks of the features based on their importance It is less complex to comprehend which aspects affect the prediction. Decision making process is far more transparent than neural networks.

Robustness:

Can explicitly work AUTOMATICALLY with non-linear relationships. Less sensitive to outliers Applicable on both interval/ ratio scale data as well as nominal/ordinal scale data. The model is less endued to overfit as it is an ensemble of various base models.

Ease of Use:

The number of hyperparameters is less than those of neural networks Again you do not require a separate validation set. By construction, cross-validation is performed during the training. No requirement exists for feature scaling (although we did not use feature scaling for this case).

Specific Benefits for Charity Data:

Feature Importance:

Aids in knowing which factors have a good predictor value for successful applications.Gives recommendations that the analysis for the charity organization should consider and can help devote efforts toward most important criterion.

Handling Imbalanced Data:

Employment of class weight as a balancing institution. Improvements in the management of minority classes. The forecasts of the are less sensitive to gaps in between the data distributions.

Performance Advantages:

- **Training Speed:**

  In general, the training process was faster than in deep neural networks, with lesser computational resources as well, and can be parallelized easily. Requires fewer skills in terms of computation.

- **Validation:**

  Integrated out-of-bag error estimation. Cross- validation is again beneficial as it delivers sound performance measures. Far less likely to overfit to a given set of data compared to neural networks.