Predicting Credit Card Fraud Using TensorFlow

This report shows the process behind creating a TensorFlow neural network to detect credit card fraud.

GitHub Repository

Alec McGhie

Report Date 5/1/2023

Introduction and Background

Background

Throughout the United States there were 88,354 reports of credit card fraud with $181M total loss from these cases (FTC 2021). This loss has harmful effects on the credit card holders and is a loss in revenue for the financial institution.

Objective

My goal for this project was to create a model that would be able to successfully predict a binary classification with above a 90% accuracy for both positive and negative transactions. Having the ability to predict these cases is useful for financial institutions and credit card holders because flagging fraudulent transactions will help reduce the financial loss for the institutions and improve the customer experience for the card holder.

Approach and Methodology

Dataset

The dataset for this project was collected from dataworld with the original sourcing from the Machine Learning Group at The Free University of Brussels(Dal Pozzolo, Caelen, Johnson, Bontempi, 2015). The original dataset contains 30 features, a binary classification for fraud status, and 284,807 transactions. In these transactions 492 were classified as fraudulent making up only 0.1727% of all transactions, this is an important characteristic of the dataset because it displays the unbalanced nature of the dataset and presents an issue when training a neural network.

Pre-Processing

To prepare the data for the neural network, there were several preprocessing measures that I had to use. The following methods are what I used to clean the dataset and prepare it for the neural network.

Separating the transactions into positive and negative dataframes

To solve for the drastic imbalance between positive and negative fraudulent cases, I decided to use an even split for the final dataset that the neural network would be trained on. This split is necessary because without rebalancing the dataset, the model will overfit and assume that the 492 positive cases aren't important. To do this I used *Pandas* built-in filtering framework to make two dataframes: *fraud_positive_df* and *fraud_negitive_df* . These two data frames included the binary classification from the dataset.

Reshaping and combining the datasets

The next step of the preprocessing was to resize *fraud_negitive_df* to be the same size as *fraud_positive_df* . Once the data frames were the same size, I appended *fraud_positive_df* to *fraud_negitive_df* and created a final dataset *fraud_testing_df* that contained an even split of fraudulent and non-fraudulent transactions.

Splitting the dataset into features and labels

To split the dataset into features and labels, I declared two new data frames: *fraud_testing_labels*, which contained the binary classification for fraud, and *fraud_testing_features*, which contained the feature columns in the dataset.

Train-Test Split

To create the training and testing groupings, I used *train_test_split* from *sklearn.model_selection.* This process creates groupings of the data that the neural network uses to test its predictions and train the model.

Scaling the features

To scale the features, I used *StandardScaler* from *sklearn.preprocessing*, a python library that standardizes the data to help with model performance and reduce bias for outliers.

Creating a 3 dimension dataset

Training a neural network using TensorFlow requires a 3 dimension input. To convert our two dimensional dataset to a three dimensional one, I used *to_numpy* to convert the *y_train* and *y_test* into an NumPy array. For *x_train* and *x_test*, I used the *reshape* function to add an additional dimension.

Building Neural Network

To create a model that is able to make binary classification predictions, I used *tensorflow.keras*. This library lets us define the architecture, compile, and fit the neural network model.

Report Date 5/1/2023

Defining Model Architecture

The first step of creating a convolutional neural network is to design how the model is structured. For my model architecture, I chose to make a sequential model with 10 layers and 10 epochs. The layers I used to accomplish this are: *dense*, which connects all of the neurons to the previous layer*, dropout*, which randomly dropped *0.2* of all of the neurons to prevent overfitting*, flatten*, which converted the previous layer into a one dimensional vector to be passed to the next layer*, BatchNormalization*, which normalized the previous layers activation and stabilized the training process, and *Conv1D*, which enabled the model extract the most influential features.

Compiling the model

To compile my model, I used the following arguments in my optimizer:

- *'Adam'* optimizer for improvement to the convergence and performance of neural networks.
- *binary_crossentropy* loss function for assisting with binary classification models.
- *accuracy* metric for measuring the total amount of correct predictions.

Training the model

When training the model, I used the declared *epoch* and train and test variables that I declared earlier in the code.

Results from the model

After running the first model the issues in its design are clear. When looking at the evaluation graphs (Figures 1 & 2), they indicate that there is a disparity between the training and testing cycles of the model. Evaluation methods are another way to see the effectiveness of a model, and this model didn't perform well with this architecture.

Metric	Class 0	Class 1	Overall	Explanation
Accuracy	N/A	N/A	60.41%	Proportion of correct predictions out of the total predictions made.
Precision	1.00	0.55681818	N/A	Proportion of true positive predictions out of the total positive predictions made per class.
Recall	0.21212121	1.00	N/A	Proportion of true positive predictions out of the total actual positive instances per class.

Report Date 5/1/2023

Metric	Class 0	Class 1	Overall	Explanation
F-1 Score	N/A	N/A	0.7153284671	Means of precision and recall, used to balance the trade-off between them.
Confusion Matrix	TN=21 FN=0	TP=98 FP=78	N/A	The confusion matrix shows metrics for the model's prediction of 197 test transactions.

Changes to approach

After running the first model and getting these disappointing metrics, there are clearly changes that have to be made. The solutions that I decided on were:

- The addition of a *MaxPool1D* layer to the neural network to help reduce the number of parameters in the model and focus on the most important features.
- Increasing the number of *epoch*'s from 10 to 50 to help the model loop through the dataset more times, thereby increasing the performance of the model.

Results from the second model

After fitting the second model, it was evident that the aforementioned changes made the model better designed for the dataset. The evaluation graphs demonstrated clear improvement through the *epoch*'s increase, the accuracy increasing, and the loss decreasing (Figures 4 & 5). This overall improvement wasn't just seen in the evaluation graphs, but in the metrics as well.

Metric	Class 0 (Non-Fraud)	Class 1 (Fraud)	Overall	Explanation
Accuracy	N/A	N/A	97.97%	Proportion of correct predictions out of the total predictions made.
Precision	0.97979798	0.97959184	N/A	Proportion of true positive predictions out of the total positive predictions made per class.
Recall	0.97979798	0.97959184	N/A	Proportion of true positive predictions out of the total actual positive instances per class.
F-1 Score	N/A	N/A	0.979591836735	Means of precision and recall, used to balance the trade-off between them.

Report Date 5/1/2023

Metric	Class 0 (Non-Fraud)	Class 1 (Fraud)	Overall	Explanation
Confusion Matrix	TN=97 FN=2	TP=96 FP=2	N/A	The confusion matrix shows metrics for the model's prediction of 197 test transactions.

Results

Findings

When comparing the two models, the small changes that are made to the architecture have huge implications on not only the accuracy, but also on other aspects like the feature importance. One of the most important things that I found was the ways that the models interpret the data and what they find to be the most effective way to solve the binary classification problem. The analysis portion of the models provided insight into the implementation of MaxPool1D and the increased epochs. The feature importance is shown In *figures 3 & 6*. Although 28 of the features are protected, the figures show that the second model found that some of the features didn't have a significant impact on the binary classification, and the model accounted for this. This change is because of the simplification that is brought to the model through the *MaxPool1D* layer and the additional training over 50 epochs. Being able to focus more on the features that impact the outcome helps the model make the determination because it cancels out the additional noise that can affect the accuracy.

Considerations

When looking at the evaluation metrics of this model, it is easy to conclude that this is an accurate model. An important area of consideration is the sourcing for the dataset and the other disparities, like the 0.1727% of cases in the original dataset being fraudulent. This model was able to make correct predictions on 97.97% of the selected cases from our data cleaning. Although this is a high percentage, implementation of this model into a bank would not yield the same results. The reasoning behind this is because of the lack of data that the neural network was able to train and test on. For a model like this to be implemented by a bank, the model would need to be trained with a lot more data and be retrained often to pick up on the changes relating to fraud. That being said, the basic ideas expressed through this project have the potential to make a really effective model that is able to make binary classifications for thousands of credit

card transactions, and with more development could lead to a model that helps to fight the issue of credit card fraud.

Conclusion

Summary

Initially when making the first experiments for this code, I found surprising results, like an accuracy of 99.94% when using the entire dataset. After looking into other metrics, I found that the model simply predicted everything as a negative because the positive cases were so rare and they could be viewed as outliers. It is important to be critical of your findings because when you take your conclusions at face value, they can often be misleading.

Conclusion

These models show the importance of model architecture when it comes to developing a neural network. Having a high accuracy is not the only important factor when it comes to making an effective model, but finding different ways to design your model's architecture is essential for solving the problem that you set out to solve through the use of machine learning. By structuring your model differently, you are able to find solutions to your problems, and in the case of credit card fraud, the slight changes can alter the effect of the model by millions of dollars when applying it to a financial institution.

Figures

Report Date 5/1/2023

Figure 1 - Model Accuracy Graph: First Model

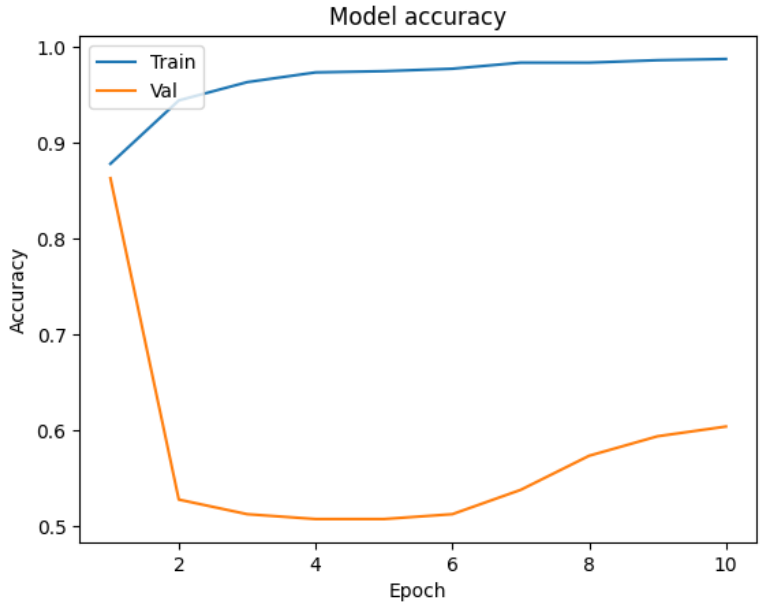

Figure 2 - Model Loss Graph: Second Model

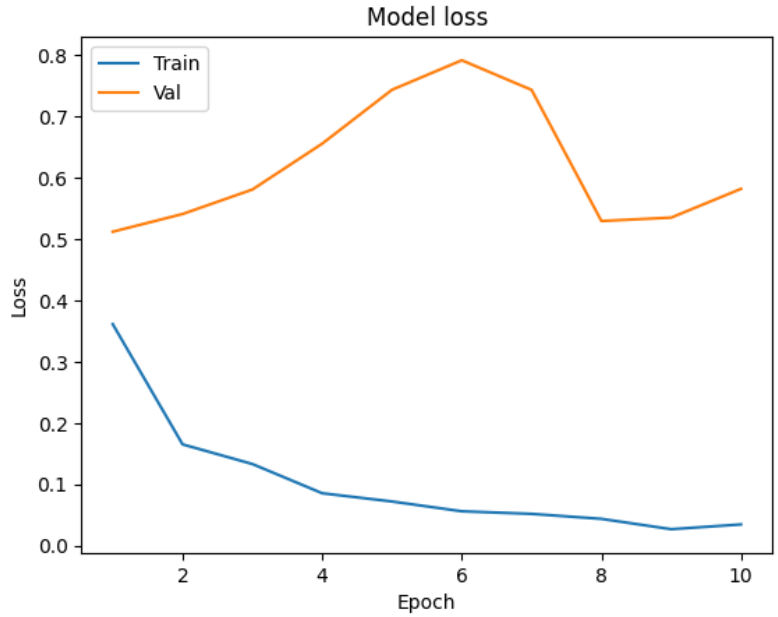

Figure 3 - Model Feature Importance: First Model

```
Feature 1: 0.04772 +/- 0.00885
Feature 2: 0.01624 +/- 0.00673
Feature 3: -0.02234 +/- 0.00793
Feature 4: 0.00000 +/- 0.00681
Feature 5: 0.03401 +/- 0.00937
Feature 6: -0.00711 +/- 0.00406
Feature 7: 0.00863 +/- 0.00720
Feature 8: -0.00558 +/- 0.00620
Feature 9: -0.02386 +/- 0.00644
Feature 10: -0.00863 +/- 0.00603
Feature 11: 0.00812 +/- 0.01045
Feature 12: -0.01726 +/- 0.00689
Feature 13: 0.02538 +/- 0.00642
Feature 14: 0.00558 +/- 0.00833
Feature 15: 0.01168 +/- 0.00558
Feature 16: -0.00355 +/- 0.00683
Feature 17: -0.01980 +/- 0.00892
Feature 18: -0.00355 +/- 0.00457
Feature 19: -0.01827 +/- 0.01045
Feature 20: 0.00152 +/- 0.00644
Feature 21: -0.00203 +/- 0.00465
Feature 22: 0.00203 +/- 0.00337
Feature 23: -0.00761 +/- 0.00943
Feature 24: -0.00203 +/- 0.00650
Feature 25: -0.00051 +/- 0.00479
Feature 26: -0.00406 +/- 0.00443
Feature 27: 0.00355 +/- 0.00644
Feature 28: 0.00000 +/- 0.00321
Feature 29: -0.01929 +/- 0.00843
Feature 30: 0.00254 +/- 0.00341
```

Figure 4 - Model Accuracy Graph: Second Model

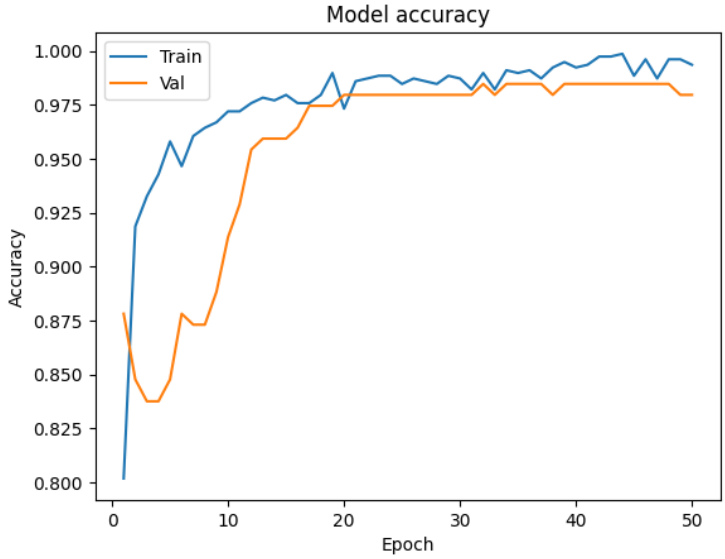

Figure 5 - Model Loss Graph: Second Model

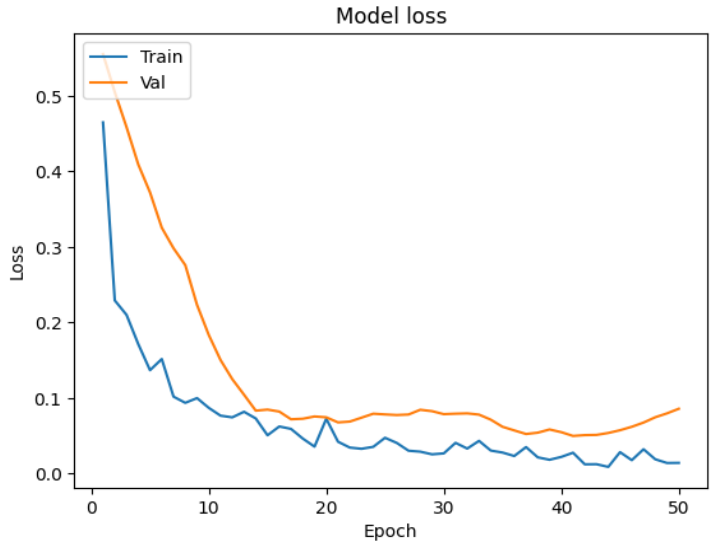

Figure 6 - Feature Importance: Second Model

```
Feature 1: 0.22437 +/- 0.01239
Feature 2: 0.01015 +/- 0.00393
Feature 3: 0.10000 +/- 0.01507
Feature 4: 0.00863 +/- 0.00510
Feature 5: 0.01269 +/- 0.00943
Feature 6: 0.00000 +/- 0.00000
Feature 7: 0.00609 +/- 0.00592
Feature 8: -0.00102 +/- 0.00305
Feature 9: 0.00355 +/- 0.00510
Feature 10: 0.00000 +/- 0.00227
Feature 11: 0.00203 +/- 0.00518
Feature 12: 0.00000 +/- 0.00000
Feature 13: 0.00406 +/- 0.00547
Feature 14: 0.00000 +/- 0.00000
Feature 15: -0.00102 +/- 0.00305
Feature 16: -0.00051 +/- 0.00152
Feature 17: 0.00102 +/- 0.00380
Feature 18: 0.00000 +/- 0.00000
Feature 19: 0.00152 +/- 0.00233
Feature 20: 0.00051 +/- 0.00273
Feature 21: 0.00203 +/- 0.00249
Feature 22: 0.00457 +/- 0.00355
Feature 23: -0.00305 +/- 0.00249
Feature 24: -0.00254 +/- 0.00341
Feature 25: 0.00051 +/- 0.00152
Feature 26: 0.00000 +/- 0.00000
Feature 27: 0.00000 +/- 0.00000
Feature 28: 0.00000 +/- 0.00000
Feature 29: 0.00000 +/- 0.00000
Feature 30: 0.00000 +/- 0.00000
```

References

Dal Pozzolo, Caelen, Johnson, Bontempi (2015). Calibrating Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE.
https://data.world/raghu543/credit-card-fraud-data

Federal Trade Commission. (2021). Consumer Sentinel Network Annual Data Book 2021.
https://www.ftc.gov/system/files/ftc_gov/pdf/CSN%20Annual%20Data%20Book%202021%20Final%20PDF.pdf