Machine Learning Engineering Capstone Project

Nathan Brasher

1. Problem Definition

1.1 Project Overview

Payment fraud losses reached nearly $28B worldwide in 2018[1], according to the Nilson Report, a financial services industry newletter. While the US only accounted for ~20% global transaction volume in 2018, it represented 34% of total card fraud losses[2] – making credit card fraud a real American problem. Globally, credit card issuers take the brunt of the associated losses (~70%) while merchants and ATM operators account for the rest. At the same time, the payments market is enormously valuable, representing nearly $1.9T in revenue (not transaction volume) globally. Credit card payments in North America alone represent ~$230B in revenue to credit card issuers according to the 2019 McKinsey Global Payments report[3]. Additionally, payments revenue growth has outpaced global growth at a 6% CAGR from 2013-2018, while electronic payments have grown even faster – at a 15% CAGR over the same time period[4]. The sheer scale, profitability, and momentum of the electronic payments market means that financial institutions must invest in and facilitate the ease of electronic payments – while at the same time dealing with the material losses in fraud. The objective of this project is to deploy an example supervised machine learning model for detecting fraud in financial transactions, using Tensorflow and Sagemaker.

---

[1] https://www.prnewswire.com/news-releases/payment-card-fraud-losses-reach-27-85-billion-300963232.html
[2] Same as 1.
[3] *Global Payments Report 2019: Amid sustained growth, accelerating challenges demand bold actions*, Mckinsey Global Institute
[4] Same as 3.

1.2 Problem Statement

Given the scale of the problem, machine learning is a natural technological fit for identifying cases of payment fraud. Still, fraud detection presents challenges. However, the problem of identifying false positives, legitimate transactions flagged as likely fraud, remains a challenge to even the most sophisticated models[5]. Financial transaction data is by nature highly imbalanced, that is the vast majority of financial transactions are legitimate and fraud represents only a very small percentage, contributing to the false-positive problem.

The Kaggle Credit Card Fraud Detection dataset[6] which forms the basis for this project is only 0.17% fraudulent, and 99.83% legitimate. The challenge is to build an automated solution which will correctly identify the majority of the (very few) fraudulent transactions, without mislabeling too many legitimate transactions and creating a burden on consumers.

1.3 Metrics

The highly-imbalanced nature of our training data makes the selection of evaluation metrics important. That is, in a transaction data set that is 99.8% legitimate (non-fraudulent) a model that predicts non-fraud every time would still be 99.8% accurate, although completely useless. Recall (number of true positives divided by total actual positives) is a useful metric. As fraud is such a costly problem, it is important to capture as much fraud as possible. However, as stated above it is also important to reduce false positives, as such Precision (true positives divided by total predicted positives) is also important. Additionally, Receiver Operating

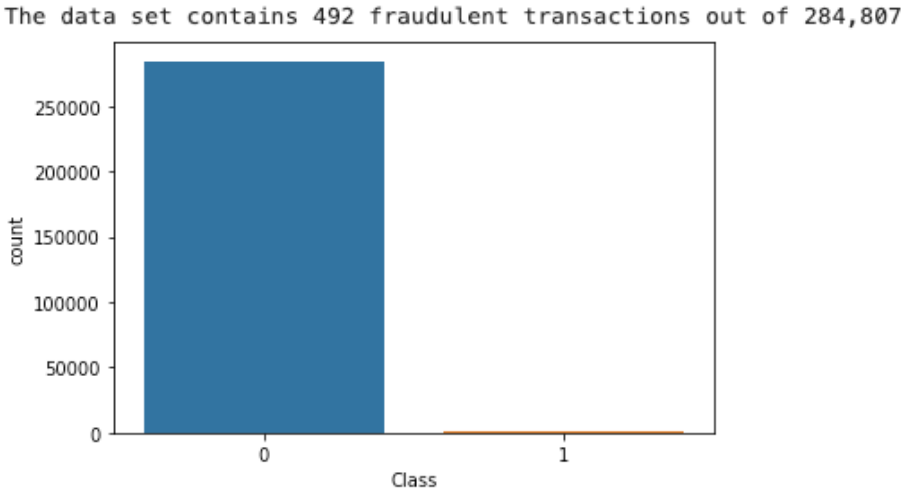
[5] http://news.mit.edu/2018/machine-learning-financial-credit-card-fraud-0920
[6] https://www.kaggle.com/mlg-ulb/creditcardfraud

Characteristic – Area Under the Curve (ROC-AUC) should help balance both Precision and Recall and help evaluate a model's true predictive accuracy.

2. Analysis

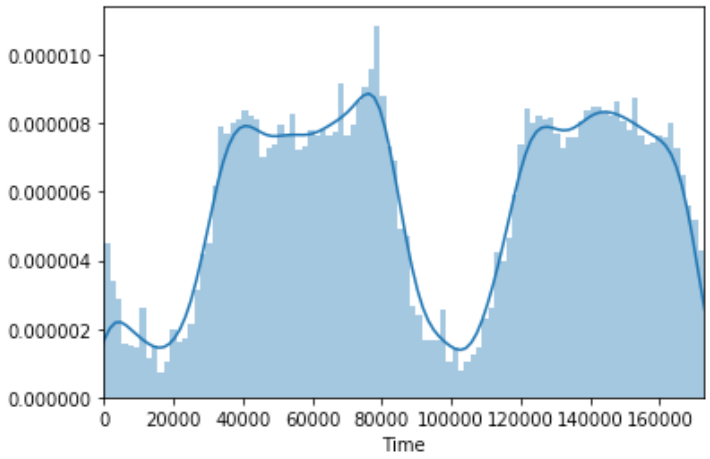
2.1 Data Exploration

The Kaggle dataset[7] contains ~280K financial transactions with 30 features that have been masked to maintain the anonymity of the underlying data. Each feature is a floating point value with the labels V1 through V28 as well as "Time" and "Amount". There are no missing values in the data. Each observation also contains a dollar amount, as well as a 0 or 1 fraud classification. The data is roughly 0.17% fraudulent, making the data imbalanced in the extreme:

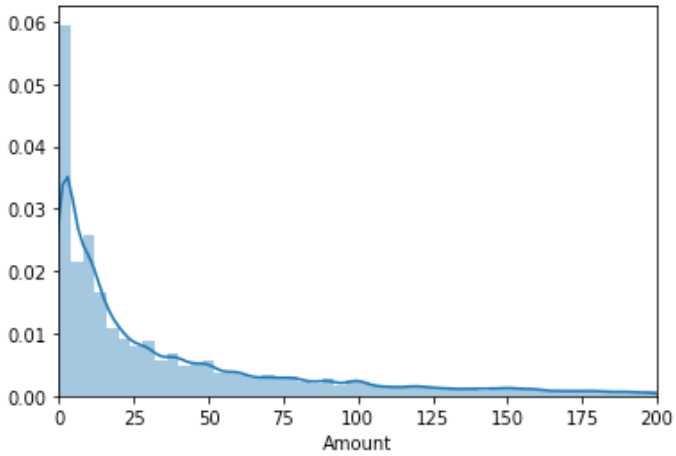


The data also contains a "time" feature that is an integer value, and shows a highly cyclical nature. The repeated variation in the time feature suggests that this is a daily transaction dataset for a limited (i.e. non-global) region, where the dips in transaction volume would logically correlate to periods of reduced human activity (i.e. 2-4am):

[7] https://www.kaggle.com/mlg-ulb/creditcardfraud

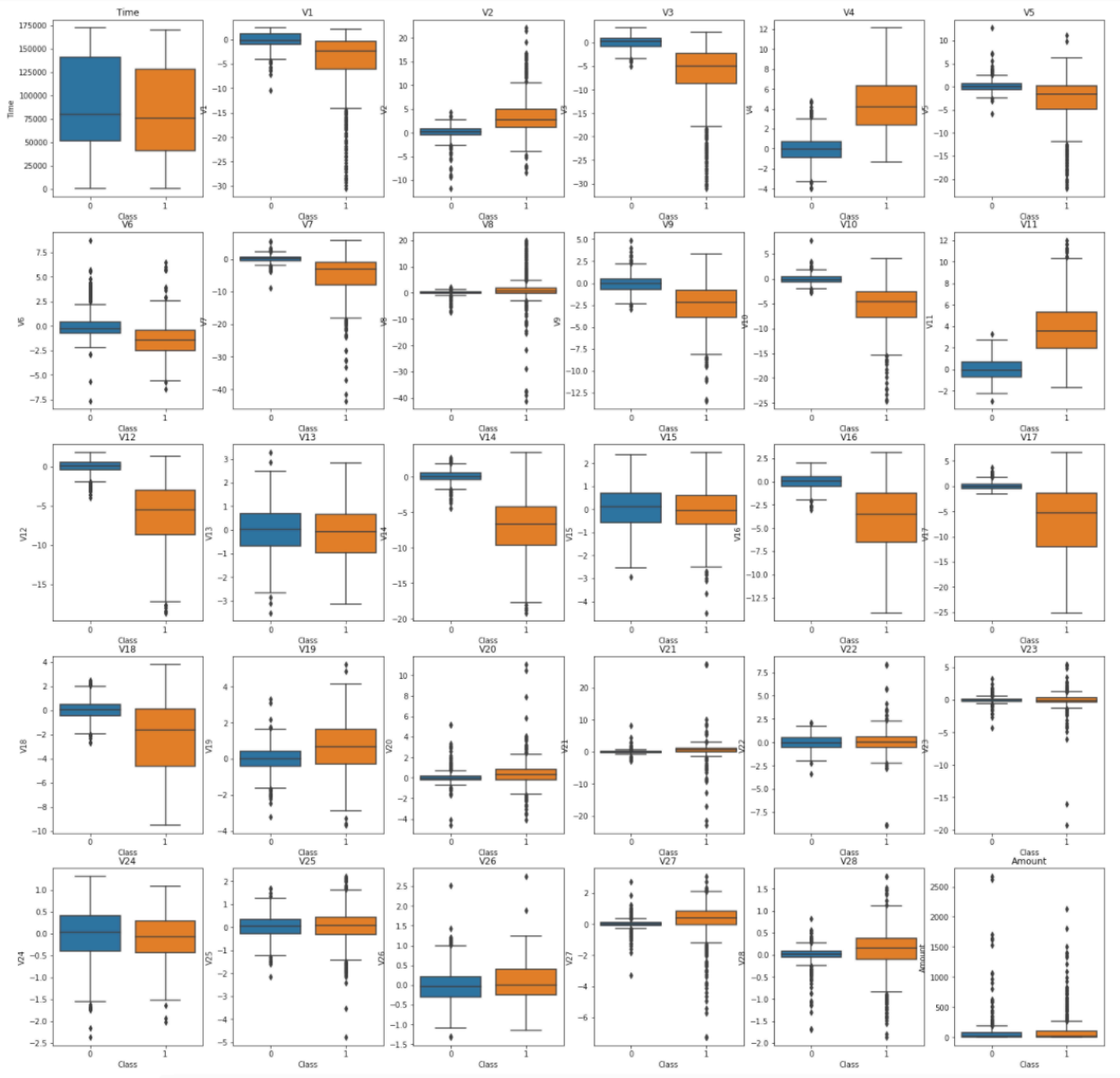


Finally, the data also shows a roughly exponential distribution in terms of transaction amount, with the vast majority of transactions of relatively small amounts, but a long tail of ~10% of transactions are above $200 in total value, with a maximum transaction value of over $25K.




```
There is a long tail with 28,837 transactions over $200
The maximum transaction is $25,691.16
```

Next, the boxplots on the following page show the distribution of individual features when split into the Fraud and non-fraud transactions. The plots were derived from an under-sampled data set, that is a subset of the data containing all 497 fraudulent transactions, and a randomly sample subset of 497 non-fraudulent transactions in order to derive relationships from balanced data:



The above boxplots the feature distribution for fraudulent transactions is on the left in blue, non-fraud is on the right in orange. The limits of each solid box show the 25-75% inter-quartile range. A number of features show substantial distinctions between each set of data, but V4, V10, V12 and V14 show the clearest separation – with little to overlap in the main part of the distribution of those features (25-75% percentile) between fraud and non-fraud transactions. These features are clearly important and will form the basis of our modelling efforts.

2.2 Algorithms, Techniques and Benchmark Models

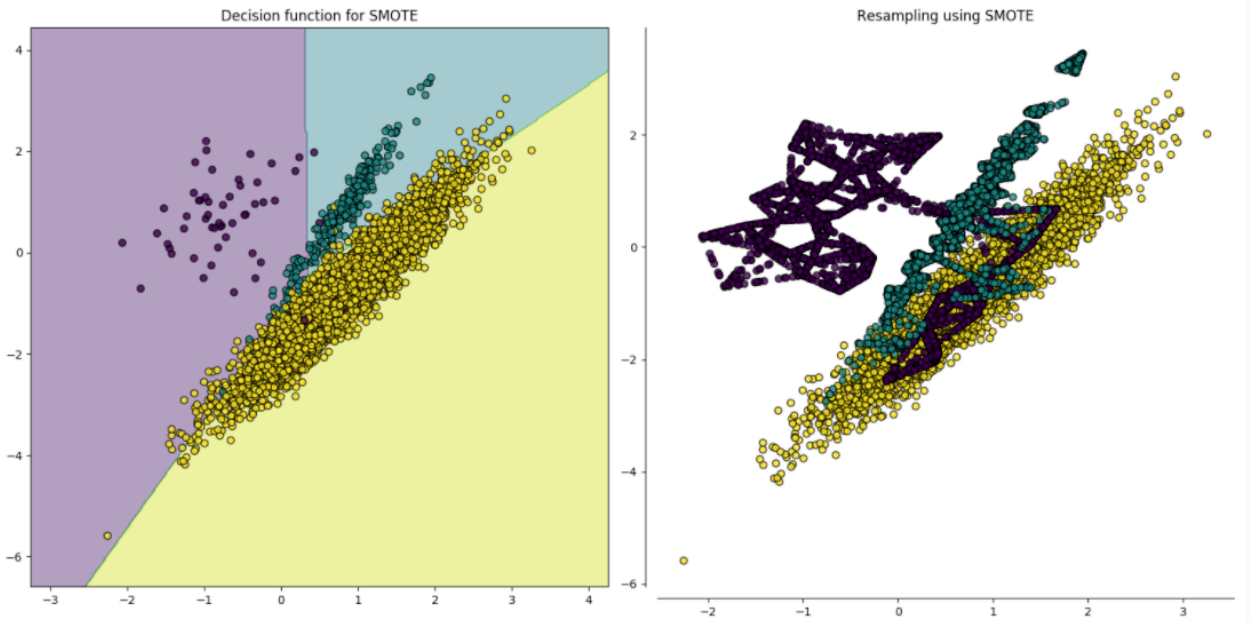
In the project proposal I highly-upvoted proposed a Kaggle notebook[8] as the benchmark model, which used a combination of feature engineering and under-sampling to fit a Logistic regression to the data, and achieve a ~93% recall on the test data, at a cost of predicting 11K false positives (out of 85K data points) for an abysmal precision of ~1%. After further research it was apparent that there were a number of other Kaggle notebooks with much more sophisticated techniques including this one[9] which generally guided my methodology. The second benchmark notebook achieved 66% recall with 85% precision on the original unbalanced data, remarkable numbers considering there were only 98 fraudulent data points out of 57K data points in their validation set.

My overall approach was to borrow heavily from the second notebooks' approach for dealing with imbalanced nature of the data was to use SMOTE[10] (Synthetic Minority Oversampling Technique) to synthesize new data points and fitting a neural network classifier to the resulting data. SMOTE is implemented in an easily accessible form the Python package imbalanced-learn[11]. In the SMOTE algorithm, each data point in the "minority" (in this case the fraudulent data) is matched up with its *k* nearest neighbors (where *k* is an integer hyperparameter, usually 5). Then a line is drawn in feature space between the two neighboring points, and new synthetic data points are interpolated along this line. This behavior is more readily evident in the figure below from the imbalanced-learn documentation[12] where the left-hand side has the

---

[8] https://www.kaggle.com/joparga3/in-depth-skewed-data-classif-93-recall-acc-now
[9] https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets
[10] *SMOTE: Synthetic Minority Over-sampling Technique,* Journal of Artificial Intelligence Research vol. 16 pg. 321–357, published June 2002
[11] https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html
[12] https://imbalanced-learn.readthedocs.io/en/stable/auto_examples/over-sampling/plot_comparison_over_sampling.html

original data points, with the minority class in purple, and the right side shows the synthesized data points, with the straight-line interpolation behavior clearly evident.

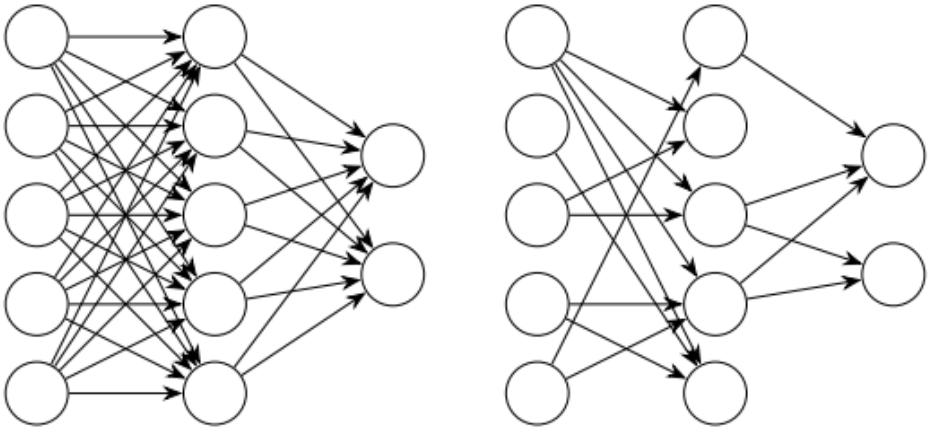


As is partially evident in the figure above, SMOTE may do a poor job of synthesizing data points if outliers are included, adding more data points between the main group and the outlier, potentially worsening classifier performance. As such, cleaning outliers before synthetization is important. My methodology for cleaning outlier data points in the fraud data is described in section 3.1.

For the classifier itself, I used a fairly straightforward deep neural net consisting of three dense layers and two dropout layers. An important part of the model itself was the use of dropout layers. Neural Nets are prone to overfitting, and dropout layers, or random deletion of connections between neurons during training, helps reduce overfitting[13]. The figure below from a paper in Neural and Evolutionary Computing[14] helps illustrate the use of dropout layers. The

[13] *Survey of Dropout Methods for Deep Neural Networks*, Neural and Evolutionary Computing, arXiv:1904.13310
[14] Same as 13.

illustration on the left below shows a typical dense network, where the illustration on the right shows the same network with half of neuron connections between each layer randomly removed.



The dropout connections are removed for one iteration of training, then replaced and a new random set removed in the next iteration. The repeated removal acts as a regularization process for the network, avoiding the creation of overly-strong connections on very specific data batches, and favoring those weights and connections that persist despite brief removal. The process helps regularize the neural nets much the same way that bagging helps regularize tree structures. The full model structure and implementation is described in more detail in section 3.2.

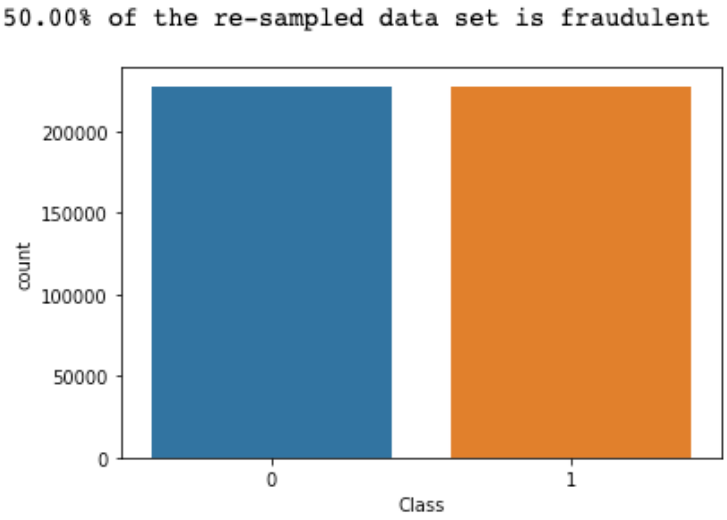
3. Methodology

3.1 Data Preprocessing

As stated previously, the biggest challenge in modelling was the imbalanced nature of the training data. Previous example on Kaggle had dealt with this by under sampling the non-fraudulent data – which creates a small data set by excluding nearly all of the 280K non-fraud data points and selecting ~500 non-fraud transactions to match the ~500 fraudulent examples. Synthetic over-sampling strategies like SMOTE allow us to create a balanced data set, while also

not throwing away the majority of the non-fraudulent data, but effectively means we are training the model on "fake" data. Additionally, SMOTE can struggle in cases of data with outliers[15] so before synthesizing a balanced training set I cleaned the important features of outlier values.

Before pre-processing the data, I set aside ~20% of the original unbalanced data set as a holdout set used for later evaluation. For the remaining 80% I cleaned outlier values for several features that held promise for cleanly separating fraud and non-fraud (V4, V10, V12 and V14). I removed fraudulent data points whose values on these features fell outside of the 25-75% percentile range in these columns. I only removed outliers in the fraud category, as these were the only data points that were being synthesized via SMOTE. Next, I synthesized a balanced data set with SMOTE.



Finally, as neural networks are highly sensitive to feature scale, I scaled the data using Sci-kit learn's RobustScaler.

[15] Same as 12.

3.2 Implementation and Refinement

In modelling the data I made use of Sagemaker's ready-made Tensorflow images. My first Neural Network architecture was a simple two-layer neural net with one input layer with 30 neurons (one for each input feature), and a single output neuron. I used rectified-linear units (ReLu) for all hidden/input layers and a sigmoid neuron for the output layer. The resultant simple neural net quickly fit the synthetic balanced data and achieved >0.99 AUC on holdout data, but didn't quite match benchmark results on the unbalanced holdout data. In an effort to improve performance and also control overfitting, I added a two hidden later of 16 and 8 ReLu units respectively with dropout layers in-between. The resulting model had ~1,500 total parameters, and as such was fairly quick to train. I used a CPU-based ml.m4.large instance for training, and the speed associated with GPU training did not justify the cost. As discussed in the next section, the resulting model performed well and beat benchmark example results handily.

4. Results

4.1 Model Evaluation and Validation

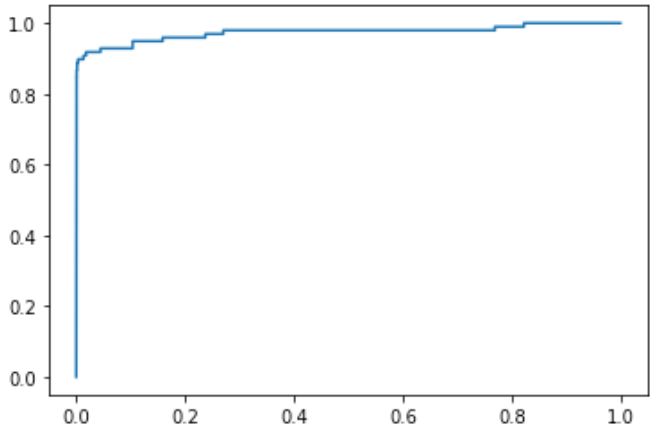
The TensorFlow model as described in the previous section quickly fit the synthetic balanced data set, even with 20% of the synthesized data used as a "validation" set during model training, the model quickly achieved near perfect ROC-AUC.

```
Epoch 1/10
318431/318431 - 5s - loss: 0.3378 - auc: 0.9338 - val_loss: 0.0994 - val_auc: 0.9940
Epoch 2/10
318431/318431 - 3s - loss: 0.1242 - auc: 0.9894 - val_loss: 0.0617 - val_auc: 0.9974
Epoch 3/10
318431/318431 - 3s - loss: 0.0864 - auc: 0.9946 - val_loss: 0.0464 - val_auc: 0.9987
Epoch 4/10
318431/318431 - 3s - loss: 0.0685 - auc: 0.9967 - val_loss: 0.0383 - val_auc: 0.9991
Epoch 5/10
318431/318431 - 3s - loss: 0.0590 - auc: 0.9976 - val_loss: 0.0317 - val_auc: 0.9994
Epoch 6/10
318431/318431 - 3s - loss: 0.0517 - auc: 0.9983 - val_loss: 0.0280 - val_auc: 0.9995
Epoch 7/10
318431/318431 - 3s - loss: 0.0460 - auc: 0.9987 - val_loss: 0.0233 - val_auc: 0.9995
```

```
Epoch 8/10
318431/318431 - 3s - loss: 0.0415 - auc: 0.9988 - val_loss: 0.0215 - val_auc: 0.9996
Epoch 9/10
318431/318431 - 3s - loss: 0.0384 - auc: 0.9990 - val_loss: 0.0187 - val_auc: 0.9997
Epoch 10/10
318431/318431 - 3s - loss: 0.0362 - auc: 0.9992 - val_loss: 0.0168 - val_auc: 0.9996
```

The real test however comes after deploying the model and evaluating on the original un-balanced holdout data. The deployed model achieved > 0.97 AUC on the holdout data, predicting ~90% of fraud (90% recall) while keeping false positives relatively low. Displayed below is the ROC curve, along with the confusion matrix on holdout data:

```
Test Imbalanced AUC: 0.9738368775337934
[[56636   228]
 [   10    88]]
```



Further evaluation shows 90% of fraudulent data and 28% precision. Given the high ROC-AUC above these could be balanced for a given business objective if need be:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.28	0.90	0.43	98
accuracy			1.00	56962
macro avg	0.64	0.95	0.71	56962
weighted avg	1.00	1.00	1.00	56962

5. Conclusions

Overall the project was an interesting investigation to prediction on highly imbalanced data as well as an opportunity to investigate the state of the art in model building with SMOTE and Tensorflow 2.0. Deploying the model on Sagemaker was a challenge, as much of the Sagemaker API is not well documented, and the Tensorflow containers in particular have changed extensively with the advent of TF 2.0. In the end, I was able to deploy a well-performing model on a challenging data set, capping off what I've learned in this course.