AI – ML PROJECT: CREDIT CARD FRAUD DETECTION USING NEURAL NETWORKS

Alan Babu - 200050005

Anubhab Ray – 200050010

Ameeya Ranjan Sethy - 200050006

Ebrahim Sohail Haris - 200050039

TABLE OF CONTENTS

INTRODUCTION

Since the mid-20th century Machine learning has seen huge advancements in its theoretical aspects and had a steady rise in popularity. But it was only the last decade where its popularity blew exponentially to become what it is now. This rise in popularity was caused by the discovery of TensorFlow, from making use of AI and ML in games such as tic-tac-toe and checkers, we have now come a long way to use these tools and make our lives better.

In this project, we aim to show the same and apply machine learning to detect if a transaction using a credit card is fraudulent or not. We will be implementing our code in python with the help of NumPy and sklearn and Keras. It need not be stressed how important the problem at the hand is. If we could somehow detect fraudulent transactions with high accuracy, it safeguards the customers from paying for what they did not purchase.

Like any other real-life problem, there are quite a few challenges with ours as well. One major problem is that the fraudulent transactions are extremely rare compared to the genuine transaction, this leads to a very unbalanced data. We tackle these problems along the way and try to get the best performance.

DATA AND ITS NORMALIZATION

One other challenge in credit card fraud detection is the lack of data since people or banks do not want to make their data public. But we obtained the data from Kaggle for the purpose of this project. This is given in the “creditcard.csv” file. Let us briefly analyze this dataset.

It contains a total of 31 columns of which the final column, called class is 0 or 1 depending on whether the given transaction is actually genuine or fraudulent. The other 30 columns are the features of a transaction we use having the relative time at which it happened, the amount transacted etc. The time at which the transaction happened does not give us any additional information so we can drop this.

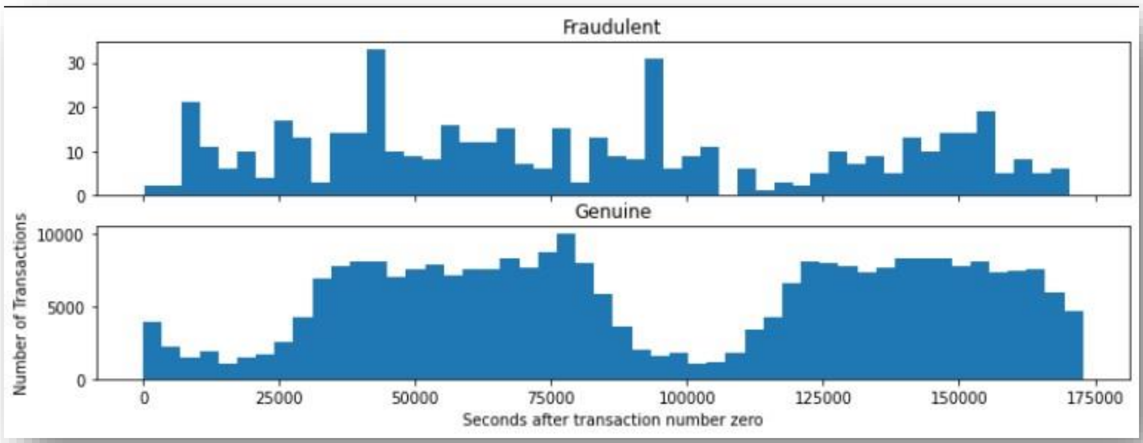


Fig: Distribution of Time Feature

Hence, we create the feature vector for each transaction having 29 columns or features.

Now we have to scale all the features to an appropriate scale so that it is easy for our neural network to converge and also that one feature does not have a control over the entire data. Notice all the other 28 features other than the amount of transaction has mean 0 and variance 1 approximately. So, we are only needed to scale the Amount feature. To do this we use the StandardScaler() in sklearn.preprocessing to scale each value by removing the mean and scaling to unit variance. We then split the data into test and training set in a 30:70 ratio.

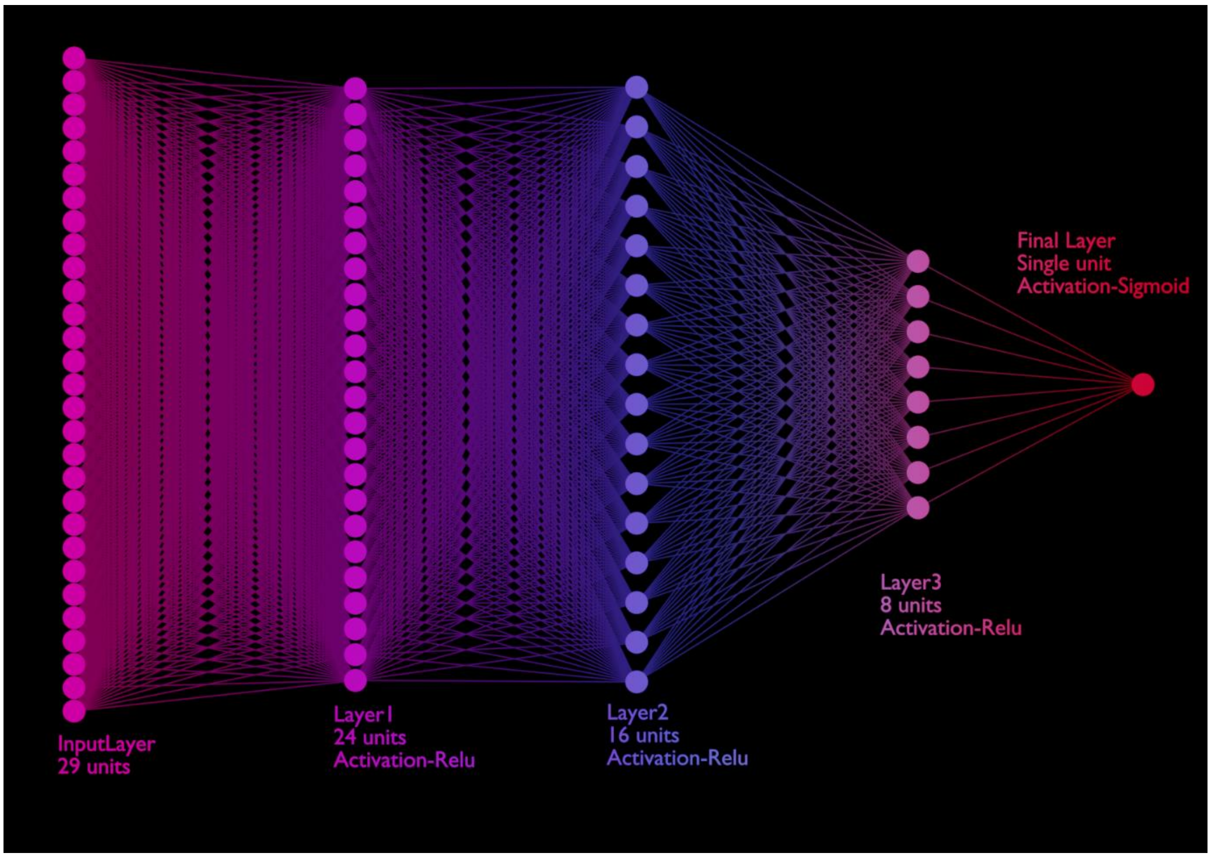
THE MODEL

We now discuss one of the most important part of our project, the model. The model we used for this purpose is of the keras.Sequential type and is defined as shown:

```
model = Sequential()
model.add(Dense(input_dim = 29, units = 24,
activation = 'relu'))
model.add(Dense(units = 16, activation = 'relu'))
model.add(Dense(units = 8, activation = 'relu'))
model.add(Dense(units = 1, activation = 'sigmoid'))
```

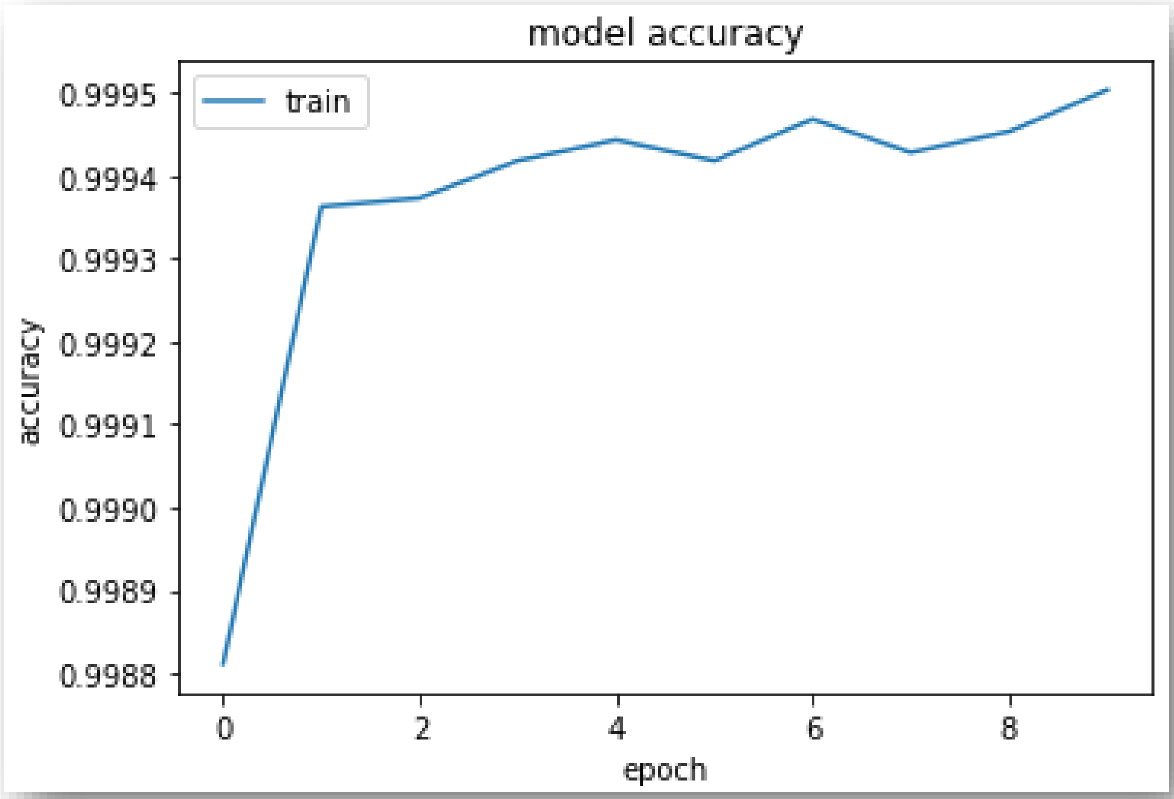
as can be observed the model is a 4 layered neural network. The first layer has 24 units whose input dimension is 29 which is the feature vector for the transaction. Its activation is relu. Which then is passed to another 2 layers having 16 and 8 units respectively and each having the activation relu. Finally, we have a single unit having sigmoid activation. This layer will output 1 or 0 depending on if the transaction is fraudulent or not respectively. Also note that all the layers are dense layers.

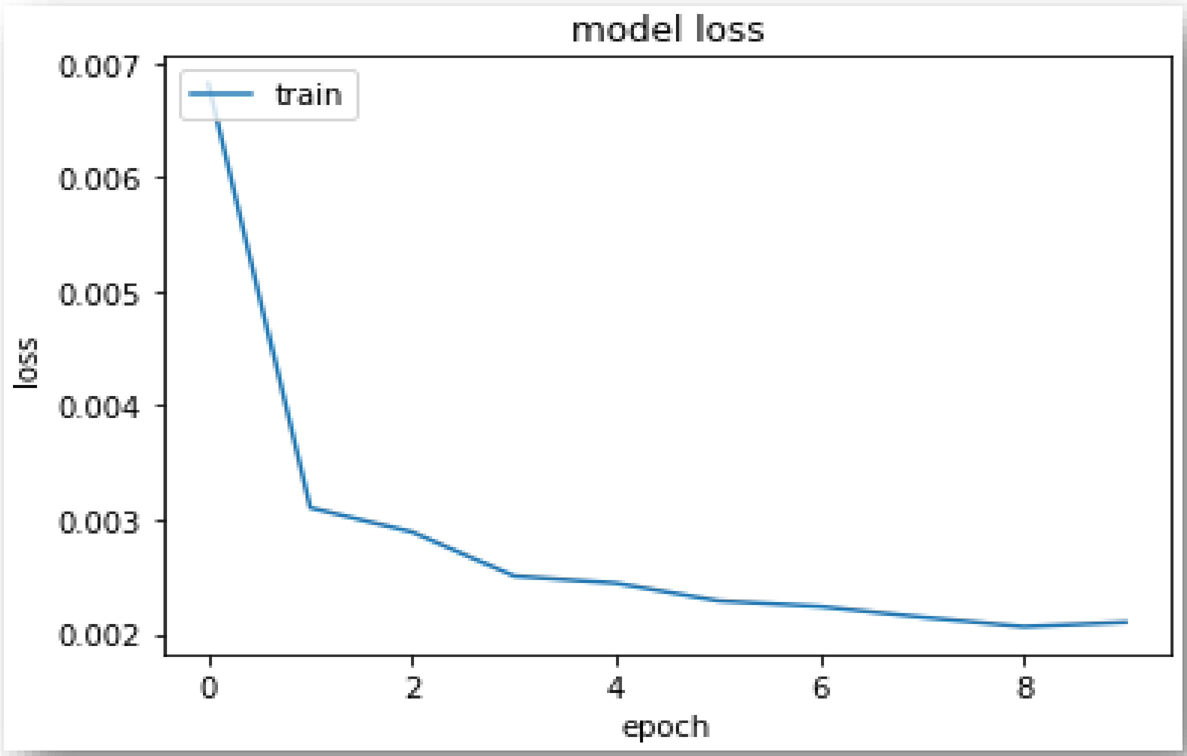
The neural network contains a total of 1265 trainable parameters and is shown below:



TRAINING THE MODEL

The model is now trained using the Adam optimizer and binary cross entropy loss for train data we have. For this we have used the inbuilt functions compile and fit in sklearn. We train it for 10 epochs and check the accuracy for the train data. We observe that the model converges fast and gives a very convincing precision of more that 99.9 percent. The loss and accuracy with epochs can be visualized from the graphs below:

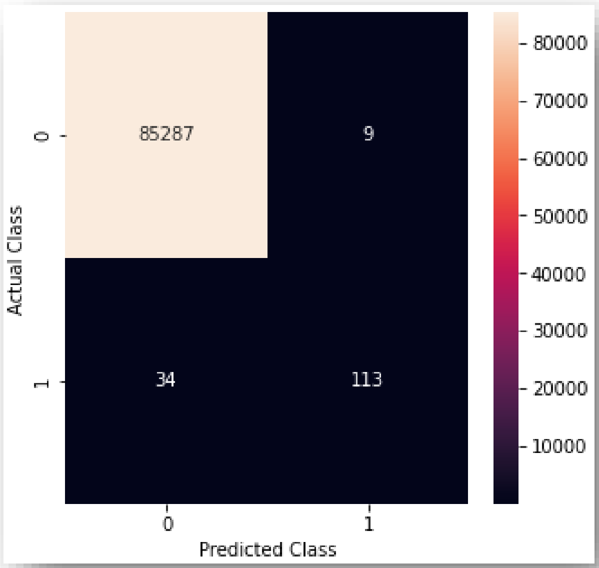




Now it is time to check the accuracy for our test data using the model we have in hand. This gives a 99.94% accuracy. At this point it is natural for us to mistake the model as a reliant fraud detection model, but there are some problems with this which we will see next.

PROBLEM WITH THE TRAINING

So, what really is the issue with our model? Clearly it is giving a very high accuracy. Let us forget about accuracy and look at another famous metric used, the f1 score. This turns out to be only about 0.8 or 80%. Indeed, a low value compared to its exceptional accuracy. To further investigate on the same, we will now inspect the confusion matrix with heat map for the given data as shown:



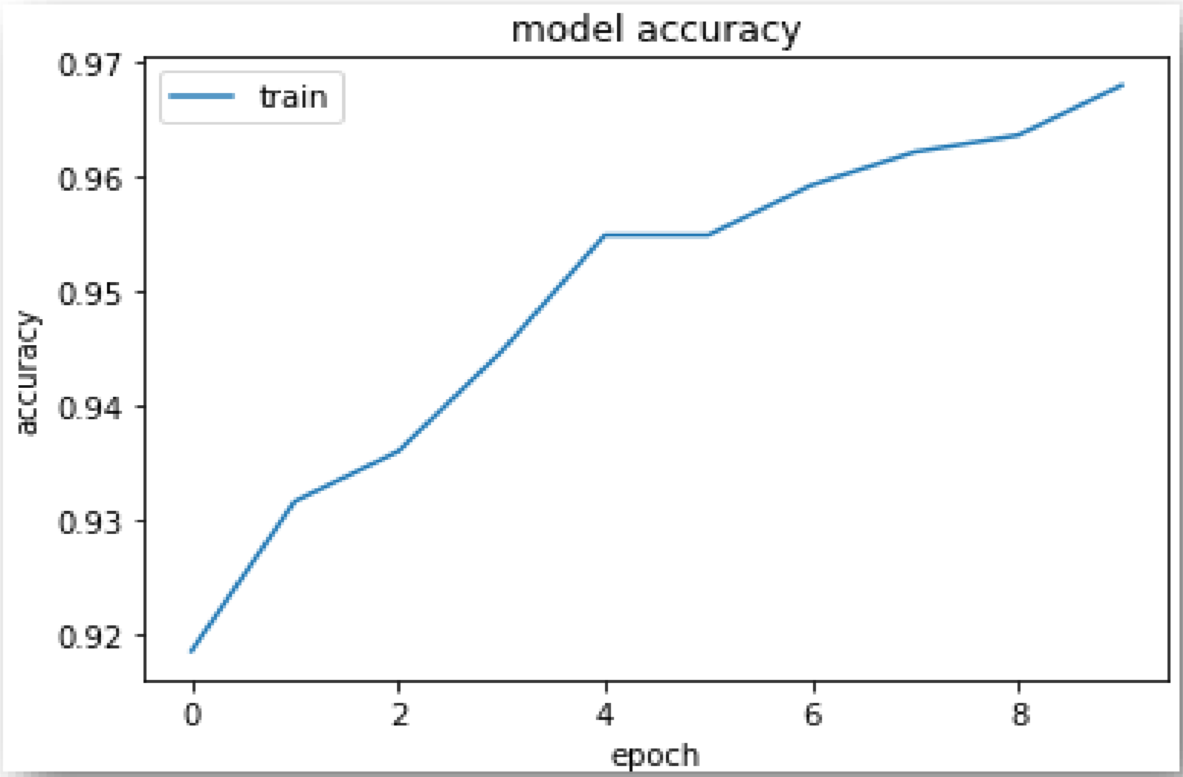
This picture made us understand where we went wrong. It was not the problem of the network we made but the data we took. As clearly visible from the confusion matrix the model has misclassified about 50 data points and correctly classified 113 points. But this is very comparable and hence the low f1 score. But why the high accuracy one might ask. This is because of the correct prediction when the transaction was not fraudulent, which comprises of the majority of the data.

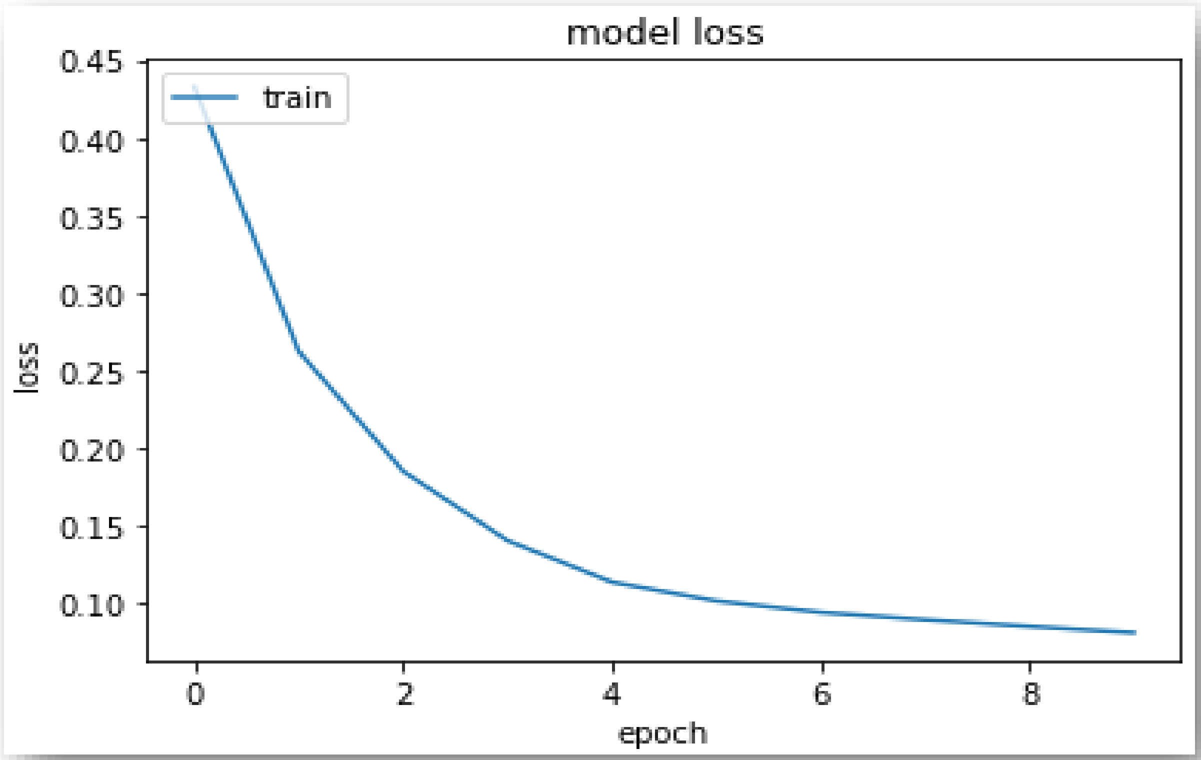
As an example, the worst model possible which predicts 0 for every input will also give a 99.9% accuracy because of the imbalance in the number of positive and negative points in the data. Now that we know the problem with our training, let us fix this problem. There are 2 known methods for solving this issue of imbalance in the data. Namely, under-sampling and over-sampling.

UNDER-SAMPLING

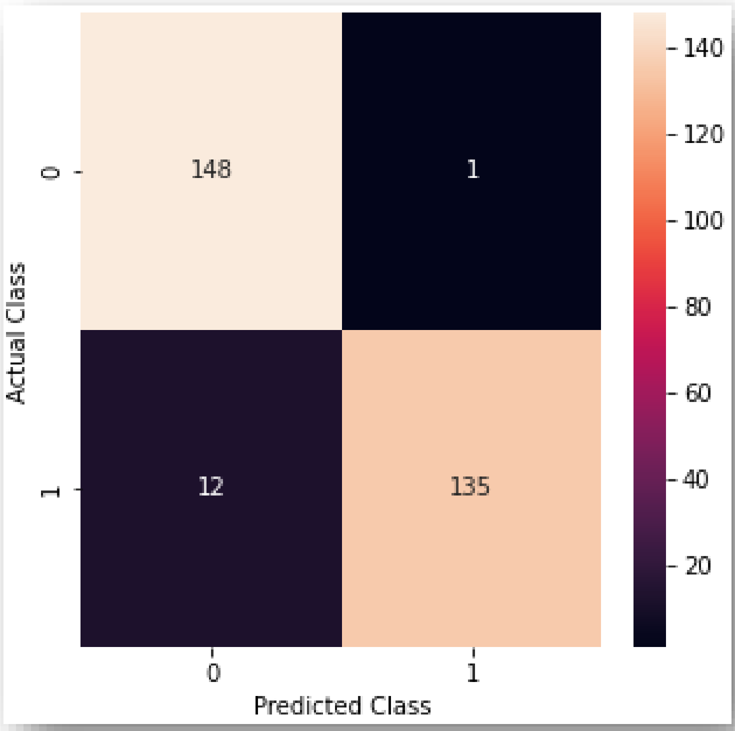
To understand under-sampling let us take an example. Let a dataset have 2000 positive points and 100 negative points. This data is imbalanced just like ours. Now we create the new dataset using the 100 negative points and any random 100 positive points from the 2000 points. This is known as under-sampling. The data split to train and test is done only after this. Now our training data contains equal number of positive and negative points. Hence the imbalance is removed.

Now let us train and check if our model has improved for detection of fraud. The loss and accuracy at each epoch are plotted again and is shown below





Once again, we have an exceptional accuracy and low loss. Even for test data we now have a high accuracy. But we will not be tricked by these metrics alone this time. Let us now check the confusion matrix for the prediction done for the new test data using the new model.



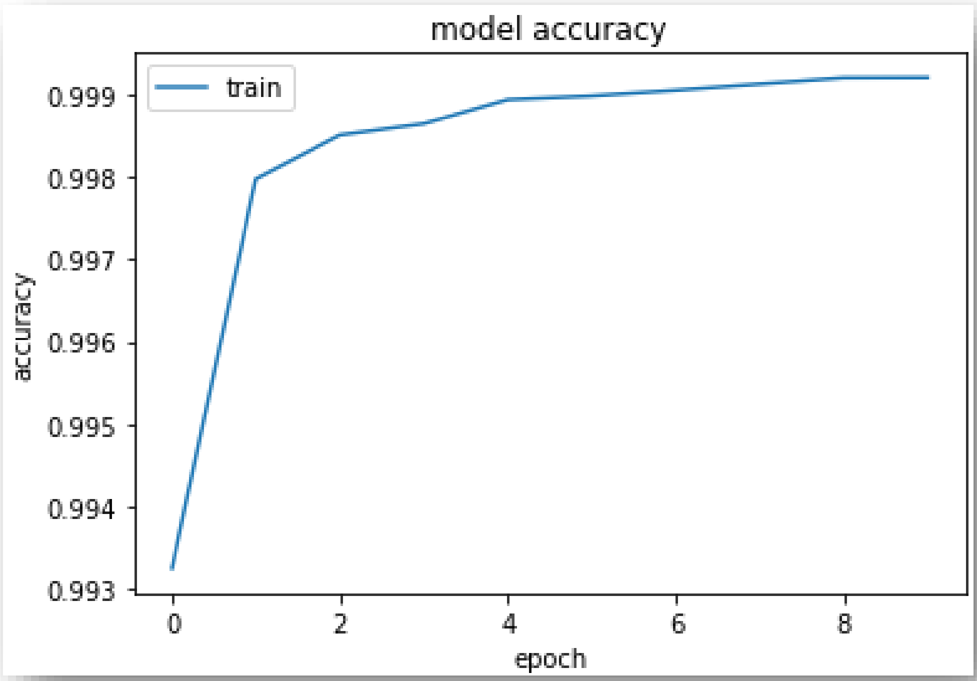
Clearly the misclassified points in comparison to the correctly classified fraud transactions have really improved this time. Let us further investigate this improvement using the f1 score metric. We find that this turns out to be 95.4%, which compared to 84% we got without under-sampling is really outstanding. We have thus created a much better model. But still we have quite a significant percentage of misclassified data, about 5%.

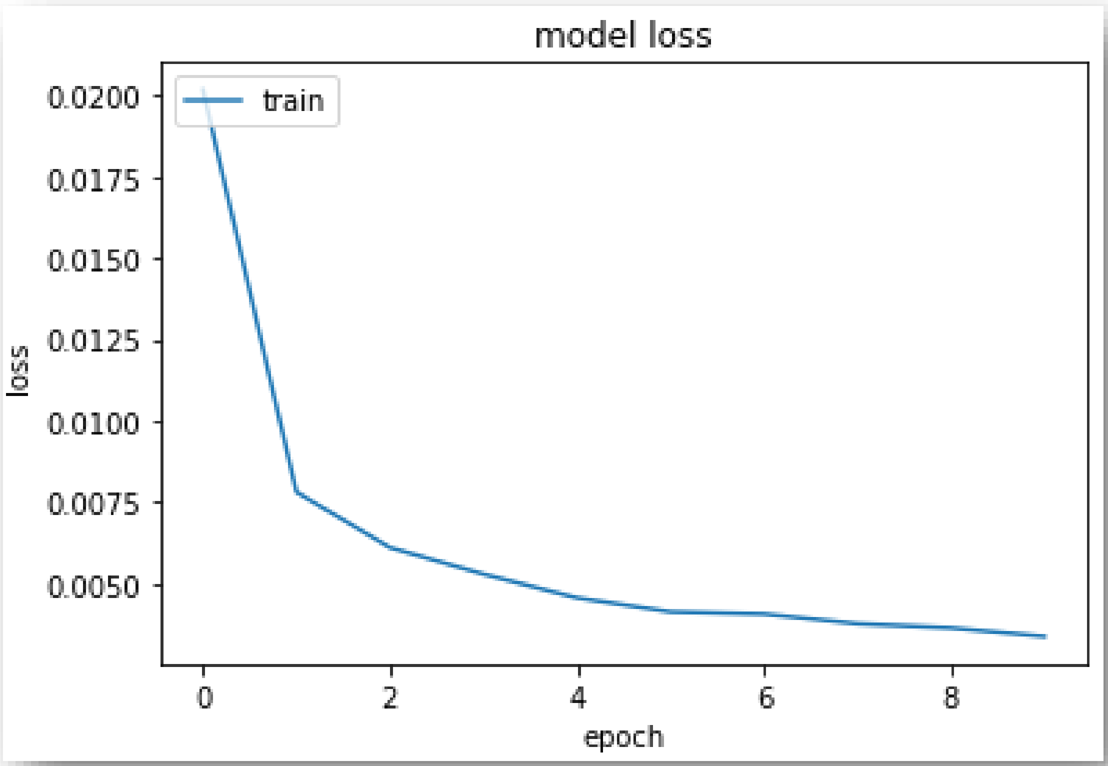
Moreover, there is a lot of available but unused data. We would like to improve on it further and hence we will use another method called over-sampling with the help of the SMOTE library.

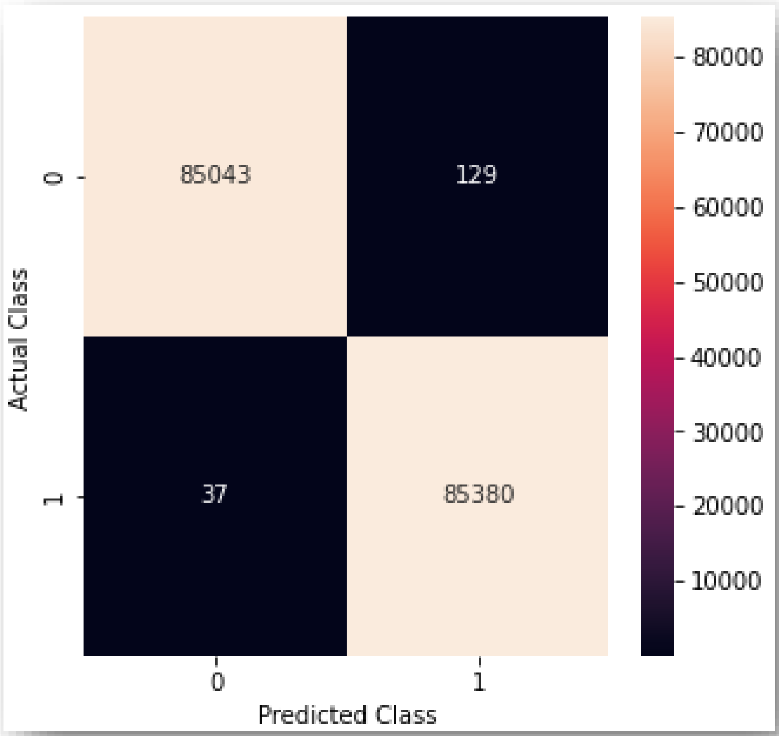
OVERSAMPLING AND SMOTE

Let us consider the example we used for under sampling again. Another intuitive thing we could have done instead of reducing the number of positives is to increase the number of negatives. This is exactly what we are going to do. But we are going to use the SMOTE library for this purpose. SMOTE is short for Synthetic Minority Oversampling Technique. In general, what it does is that instead of just copying the same data over and over for oversampling, it creates synthetic data using the existing one. This leads to a very precise model. After passing our data to SMOTE, we get the new data. We split this data into test and train and do the previous procedure again hoping for a better result.

The accuracy and loss with the number of epochs, the confusion matrix is also plotted again and is shown




model loss
train
loss
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0
2
4
6
8
epoch


85043
129
37
85380
Actual Class
0
1
Predicted Class
0
1
80000
70000
60000
50000
40000
30000
20000
10000

Using the heat map, we can clearly observe that the misclassified data is extremely rare in comparison to correctly classified data. Also, the f1 score turns out to be 99.9%.

Finally, we have a convincing f1 score and accuracy. Our model is finally ready to safeguard banks and its customers from fraud.

SUMMARY

We have shown how effective machine learning is at solving real world problems which people a few years would not even think about. Let us summarize what we have done in our project.

Assigned with the task of detection of fraudulent credit card transaction, we first got data from Kaggle and preprocessed the data for training, then we created a 4 layered neural network model to solve this. But we met with the problem of imbalance of data. We first solved this problem using under-sampling technique and got better result, but we needed a better model as the misclassification was still a significant.

Hence, we finally used SMOTE for synthetic oversampling and got an extremely precise and accurate model.

We also show some of the metrics we used to determine how good our model is as follows:

- *Accuracy*: Defined as the ratio of correct predictions and the total number of predictions.
- *Precision*: Defined as ratio of true positives and total positives predicted.
- *Recall*: Defined as the ratio of true positives to all positives in ground truth.
- *F1 score*: Defined as the harmonic mean of precision and recall.

Now we check all the four metrics for each of the three training we did.

	Accuracy	Precision	Recall	F1 Score
Raw Data	99.949	92.623	76.871	84.014
Under-sampling	95.608	99.264	91.836	95.406
Over-sampling	99.902	99.849	99.956	99.903

Hence with the help of some Machine Learning tools we have finally created a really powerful model for the detection of fraudulent credit card transaction.

REFERENCES

1. Neural Model You-Tube Playlist https://www.youtube.com/playlist?list=PLeo1K3hjS3uu7CxAacxVndI4bE_o3BDtO
2. Dataset link from Kaggle https://www.kaggle.com/datasets/arockiaselciaa/creditcardcsv
3. Scikit-learn official User Guide https://scikit-learn.org/stable/user_guide.html
4. https://www.google.com/amp/s/www.geeksforgeeks.org/ml-credit-card-fraud-detection/amp/
5. SMOTE related https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
6. Metrices https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9