1. Introduction

Credit card frauds are at an ever-increasing rate and have become a major problem in the financial sector. They can be defined as a case where someone uses someone else´s credit card for personal reasons while the owner and the card issuing authorities are unaware of the fact that the card is being used.

In an era of digitalization, the need to identify credit card frauds is necessary. Fraud detection involves monitoring and analysing the behaviour of various users to perceive, detect or avoid undesirable actions. To perform credit card fraud detection effectively, there is a need to understand what is behind it. Some major challenges in credit card frauds involve the availability of public data, high class imbalance in the data, changing nature of frauds and the high number of false alarms.

Machine learning models are employed to analyse all the authorized transactions and report any suspicious activities. These reports are then investigated by professional who contact cardholders to confirm if the transaction performed was genuine or not. So, in hindsight, there is a need to train and update these models regularly to improve fraud-detection performance overtime.

Recently, developments in deep learning methodologies have been applied to solve complex problems in various areas.

Experimental results show great performance of the proposed deep learning methods against traditional machine learning models imply that the proposed approaches can be implemented effectively for real-world credit card fraud detection systems.

This report presents a study of a possible deep learning approach for the credit card fraud detection problem by presenting three distinct approaches to our neural network. We obtained the dataset from Kaggle[1], a data analysis website that provides public datasets. The dataset contains transactions made by credit cards in September 2013 by European cardholders.

The results presented in this report are present in a python file attached that should be considered to better support the results demonstrated in this report.

[1] Dataset available at: https://www.kaggle.com/mlg-ulb/creditcardfraud

2. Explanatory data analysis

It is a common practice to carry out an exploratory data analysis to gain some insights from the data. One of our main concerns when developing a classification model is whether the different classes are balanced. This means that the dataset contains an approximately equal portion of each class. For example, if we had two classes and a 95% of observations belonging to one the classes, a classifier will always output the majority class % failling all the predictions of the minority class.

Considering the python file attached to this report some data exploration is necessary as we may find many interesting patterns that may influence the modelling and performance metrics. Considering the fact that we are dealing with a heavily imbalanced data set by simply plotting the distribution functions the amount variable seems to be heavily right skewed. We opted to scale the amount feature since its heavily right-skewed and to avoid any kind of distortion. We then excluded the time and old amount feature for better and more accurate results.

There are several ways of dealing with imbalanced datasets. One first approach is to under sample the majority class and oversample the minority one, to obtain a more balanced dataset. Other approaches include using error metrics beyond accuracy such as the precision, the recall, or the F1-score. We will talk more about these metrics later.

Regarding the training of our data, we are going to define the independent (X) and the dependent variables (Y). Using the defined variables, we will split the data into a training set and testing set which is further used for modelling and evaluating. We can split the data easily using the ‘train_test_split’ algorithm in python. We considered 70% for our training set and 30% for our test set.

3. Evaluation and Performance Metrics

To predict fraudulent transactions, we opted to build a neural network and then improve it using different sampling approaches.

To build our neural network we considered the following parameters:

- For the 1st hidden layer, “input_dim” is the number of input variables, in this case 29. The “units” is the number of nodes or neurons in each layer.
- We use Rectified Linear Unit (ReLU) as an activation function for the hidden layers as it usually performs very well in classification problems.
- We added a dropout layer to prevent overfitting.
- We use the Sigmoid function in the output layer since we are dealing with a binary classification problem.

While performing the compilation and fit of the model we also considered the following:

- We use “binary_crossentropy” as the loss function and “Adam” to update network weights. “Adam” is a popular algorithm to achieve good results fast in the deep learning field.
- The model weights are updated every 15 samples. A batch defines the number of samples to iterate through before updating the internal model parameters. At the end of the batch, the predictions are compared to the expected output, and the error is calculated.
- An epoch represents one time that the entire training set is fed through the network. In our case, we considered 5 epochs as adding more does not seem to make a difference in the results.

To evaluate the results of our neural network we used the evaluation metrics available at the scikit/learn package. We considered the following metrics:

Confusion Matrix: A confusion matrix is a table layout which allows the visualization of the performance of an algorithm. A confusion matrix gives us four important measures:

Table I – Confusion matrix

		Actual Values	
		Negative (Genuine)	Positive (Fraud)
Predicted Values	Negative (Genuine)	True Negative (TN)	False Positive (FP)
	Positive (Fraud)	False Negative (FN)	True Positive (TP)

Explaining each of them we have:

- TP – Fraudulent cases that are correctly classified as fraudulent transactions.
- TN – Genuine cases that are correctly classified as genuine transactions.
- FP - Transactions that are genuine, but the model predicts as fraud.
- FN – Transactions that are fraud, but the model predicts as genuine.

Accuracy Score: Accuracy score is one of the most basic evaluation metrics which is widely used to evaluate classification models. The accuracy score is calculated simply by dividing the number of correct predictions made by the model by the total number of predictions made by the model. The formula is presented below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision: Precision measures how accurately the model can capture fraud i.e., out of the total predicted fraud cases, how many turned out to be fraud.

$$Precision = \frac{TP}{TP + FP}$$

Recall: Recall measures out of all the actual fraud cases; how many the model could predict correctly as fraud. This is an important metric to be considered as we want to find out how the model behaves in predicting fraudulent transactions.

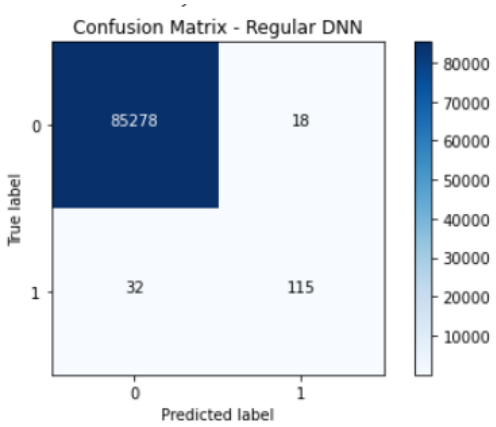
$$Recall = \frac{TP}{TP + FN}$$

F1-score: this is a balance between precision and recall.

$$F1 - Score = 2 * \frac{(precision * recall)}{precision + recall}$$

Before looking at the values that were predicted using each classification metric, we first are going to elaborate on why we are not going to consider only using accuracy as performance metric. Accuracy simply refers to the proportion of correctly classified instances. It is usually the first metric you look at when evaluating a model. Since we are dealing with imbalanced data accuracy can be misleading, a model can predict the value of the dominant class for all predictions and achieve a high accuracy score. This means that additional performance metrics are going to be required. We are mainly going to focus on the recall score because it focuses on correctly predicting fraud transactions.

We start by presenting the confusion matrix for this first approach:



Looking at the first row the model classified 85.278 as genuine transactions and only 18 (FP) were mistakenly classified as fraud. Considering the second row the model correctly predicted 115 (TP) as fraudulent ones while classifying incorrectly 32 (FN) fraudulent transactions as genuine. Our objective must be to detect as many fraudulent transactions as possible since these can have a huge negative impact. So, we need to find a way to further reduce the FN.

Looking at the performance metrics:

```
Accuracy score of the DNN model is 0.9994148145547324
----------------------------------------------------------------
F1 Score score of the DNN model is 0.8214285714285714
----------------------------------------------------------------
Recall score of the DNN model is 0.782312925170068
----------------------------------------------------------------
Precision score of the DNN is 0.8646616541353384.
----------------------------------------------------------------
```

At first glance, these performance metrics do not appear to be bad. We were able to achieve a recall score of 78%. But is there way to improve these metrics?

4. Dealing with imbalanced data: Resampling Techniques

Since we exposed the problem of imbalanced data and how it provides misleading classification we would like to know if there is a way of improving these results.

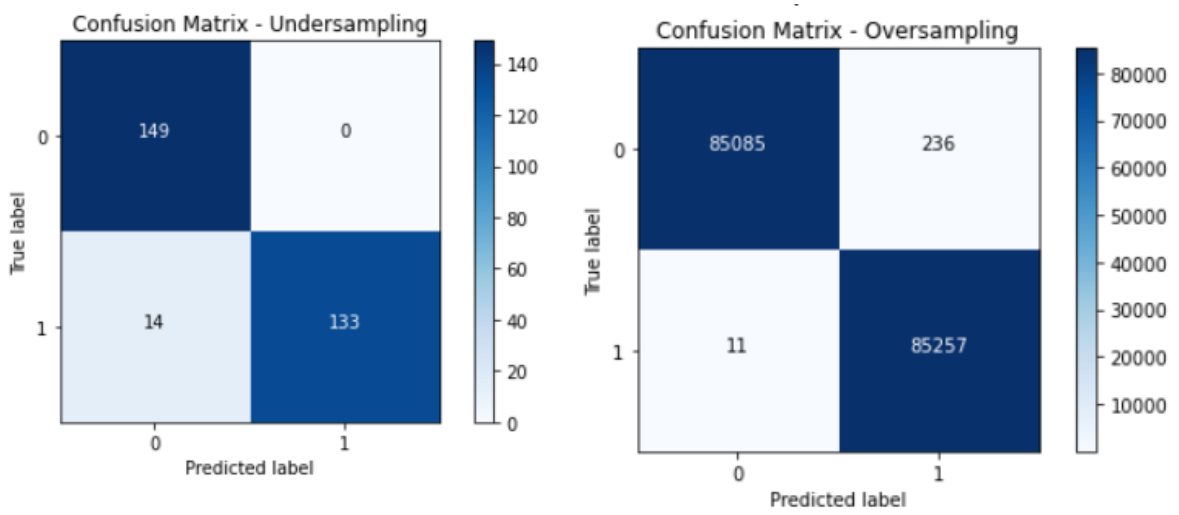
Sampling is a good approach to deal with imbalanced data. We change the dataset in such a way that the model gets balanced data as input. There are two types of sampling — undersampling and oversampling.

The idea of undersampling is to delete some instances of the oversampled class to obtain class equivalence. If you have a very large dataset, it is a good approach to try undersampling. We must make sure that we do not lose any relevant information by undersampling.

Oversampling uses copies of the underrepresented class to get a balanced data. In our case, we add copies of fraudulent class so that we get equal number of genuine and fraudulent cases. This new dataset is given as input to the classification algorithm.

Synthetic Minority Oversampling Technique (SMOTE) is a very popular oversampling technique. SMOTE technique generates a new synthetic minority class data which is used to train the model. It selects minority class data points that are close in the feature space and draws a line between the selected data points. A new sample is generated at a point along this line. This procedure is used to create the required number of synthetic samples of the minority class.

Considering both metrics the model seems to be performing much better:



The best results are achieved by oversampling the under-represented class using SMOTE. With this approach the model can detect almost 100% of all fraudulent transactions in the unseen test data set. This satisfies our primary objective, to detect the vast majority of fraudulent transactions. In addition to this the number of false positives remain at an acceptable level. This means a lot less verification work is required from a fraud department. This is clear by the metrics presented below the recall score is clearly improving:

Table II – Final Results

	Accuracy	Recall	Precision	F1-Score
Regular DNN	0.9994	0.7823	0.8647	0.8214
Undersample DNN	0.9527	0.9048	1.0000	0.9500
Oversample DNN	0.9986	0.9999	0.9972	0.9986

5. Conclusion

Fraud detection is a key area where machine learning can lead to billions of savings for businesses while providing customers with a safer environment. Through sophisticated feature engineering and modeling techniques machine learning can help detect fraudulent events as anomalies in a customer purchasing journey. Implementing the right machine learning solution means going beyond simple solutions.

Credit card frauds are an increasing threat to the financial institutions. Fraudsters tend to come up with new fraud methods. A robust classifier is one that can cope with the changing nature of the frauds. Accurately predicting the fraud cases and reducing the number of false positives is the foremost priority of a fraud detection systems. The performance of machine learning methods varies for each business case.

Using neural networks, we were almost able to achieve 100% fraud detection. In this study, sampling methods have been used to deal with the class imbalance problem increasing the performance of the network.

While we could not reach out 100% fraud detection, we did end up creating a system that can, with enough time and data, get very close to that goal. As with any project, there is some room for improvement. This model can further be improved with the addition of more algorithms into it. However, the output of these algorithms needs to be in the same format as the others. Once that condition is satisfied, the modules are easy to add as done in the code. This provides a great degree of versatility to the project.

Since the entire data set consist of transaction records of only two days, the machine learning algorithm will only increase its efficiency over time as more data is put into it.