# CI7520 - Coursework 2

## Machine Learning with Deep Neural Networks

**Group Report on**

## Timeseries Anomaly Detection

**By:**

**Muskan Asmath - K2279003**

**Abin Varghese - K2275285**

**Neha Gupta - K2263969**

**Amirali Monjar - K2205064**

# ABSTRACT

Due to an increase in fraud rates, academics have begun employing various machine learning techniques to identify and analyse online transaction fraud. In this project, we have built a deep learning based web system using Keras that will allow real time anomaly detection for timeseries dataset for credit card fraud detection. As the dataset is imbalance, we have handled the data using Oversampling techniques such as SMOTE(Synthetic Minority Oversampling Technique) and Normal sampling technique. We have used 4 different Deep Learning models - CNN(Convolutional Neural Network), LSTM(Long Short-Term Memory), Autoencoder and GAN(Generative Adversarial Network ) to classify the dataset and use it to predict fraud transactions.The Web UI created has 4 pages for Dataset analysis , User input for cheching fraud transactions, evaluation of the 4 model results.We have compared all the 4 models results using F1 precision, recall and confusion matrix.

# TABLE OF CONTENTS

# 1. INTRODUCTION

Although credit card fraud only occurs in a small percentage of transactions, the resulting financial losses could be substantial. Due to the varied nature of fraudster behaviour, automatic fraud detection systems (FDS) must be designed in order to detect fraudulent transactions with high precision. In fact, the type of fraud behaviour might vary significantly depending on the payment method (such as an e-commerce or store terminal), the nation, and the population group.

Credit card fraud detection is like looking for needles in a haystack. It requires identifying the fraudulent transactions among the millions that occur every day. It is currently practically difficult for a human specialist to identify significant patterns in transaction data due to the exponential growth in data. Because information extraction from huge datasets is necessary in the field of fraud detection, machine learning techniques are increasingly widely used.

This coursework project presents a comparative study on deep learning approaches for credit card fraud detection. In particular we present **Convolutional Neural Network (CNN)** model that considers the data individually at each time instance, **Generative Adverserial Networks (GANs)** which is a state-of-the-art model that considers them as a timeseries, **Long short-term memory network(LSTM)** model to perform as classification and **Autoencoder** model to handle the problem as anomaly detection. The 4 models are developed, trained and evaluated on an imbalance dataset of credit card transactions.

## 1.1 Problem Statement

The Credit Card Fraud Detection problem includes modeling past credit card transactions with the knowledge of the ones that turned out to be a fraud. These modesl are then used to identify whether a new transaction is fraudulent or not. while minimizing the incorrect fraud classifications.

## Imbalanced Dataset

Imbalanced data refers to a dataset in which one class (the minority class fraud) is significantly underrepresented compared to the other class (the majority class). Imbalanced learning addresses classification problems where the number of examples representing one class is much lower than the ones of the other classes. Since most learning algorithms are not built to handle a significant discrepancy in the number of examples belonging to distinct classes, learning from imbalanced datasets is a challenging undertaking.

Although the fraction of frauds in real-world datasets might be as low as 0.01% [Ban20, DP15, LJ20], credit card fraud is an example of an imbalanced problem. The dataset used for the coursework includes credit card transactions made by European cardholders in September 2013.

Many solutions have been proposed to deal with this problem, both for standard learning algorithms and ensemble techniques. There are several approaches that can be used to handle imbalanced data in machine learning:

1. **Resampling techniques**: These techniques involve either oversampling the minority class or undersampling the majority class in order to balance the dataset.

2. **Cost-sensitive learning:** This approach involves modifying the loss function of the model to give more weight to the minority class, so that the model is more sensitive to misclassifying examples from the minority class.

3. **Ensemble methods:** Ensemble methods, such as bagging or boosting, can be effective at handling imbalanced data by combining the predictions of multiple models.

4. **Thresholding:** In some cases, it may be appropriate to adjust the classification threshold, so that the model is more or less sensitive to the minority class.

5. **Generative adversarial networks (GANs)**: GANs can be used to generate synthetic examples of the minority class, which can be used to balance the dataset.

Algorithms in cost-sensitive approaches are changed to favour the detection of the minority class. This typically means that the optimization function in the learning algorithm's training phase needs to be modified. Resampling techniques, on the other hand, work at the data level by introducing a pre-processing phase to rebalance the dataset before the training algorithm is used. Removal of examples from the majority class (undersampling techniques), addition of examples from the minority class (oversampling techniques), or a combination of undersampling and oversampling are all ways to achieve resampling..

For our project we have used Oversampling methods like Random sampling and SMOTE for handling the challenges by keeping the below objectives in mind.

## 1.2 Primary Objectives

- In fraud detection, the primary objective is to maximize recall while capping the False Positive Rate (FPR). Recall measures the ability of the model to correctly identify fraudulent transactions, while FPR measures the rate at which legitimate transactions are incorrectly classified as fraudulent.

- If the validation set has a low proportion of fraudulent transactions relative to non-fraudulent transactions, the FPR can be misleading. Specifically, the FPR equation, which measures the ratio of False Positives to the sum of False Positives and True Negatives (FPR = FP/(FP + TN)), can produce a low FPR even if the model incorrectly classifies a large number of legitimate transactions as fraudulent.

- In the presence of class imbalance, a model can classify a large number of charges incorrectly and still maintain a low FPR due to the high number of True Negatives in the dataset.

## 2. Literature Review:

This literature review explores the application of deep learning neural networks to credit card fraud detection datasets.

2.1. The paper "**Deep Transfer Learning Approaches for Credit Card Fraud Detection**" by Lebichot et at (2019) addresses the problem of designing automatic Fraud Detection Systems (FDS) for credit card transactions. Given the high cost involved, the paper emphasizes the importance of transferring existing FDS pipelines to different domains and contexts instead of designing data-driven FDSs from scratch. The authors focus on the problem of transferring classification models learned on a specific category of transactions, such as e-commerce transactions, to another category, such as face-to-face transactions. The paper compares two domain adaptation techniques in a deep neural network setting with three state-of-the-art benchmarks. The first technique proposed in the paper is an original domain adaptation strategy that creates additional features to transfer properties of the source domain to the target domain. The second technique is an extension of recent work by Ganin et al. Both techniques are evaluated on a five-month dataset of more than 80 million e-commerce and face-to-face transactions provided by a major card issuer. The paper highlights the importance of transfer learning in credit card fraud detection, as the nature of fraud behaviour may strongly differ according to the payment system, the country, and the population segment. The authors conclude that the proposed deep transfer learning approaches show promising results in adapting e-commerce classification models to face-to-face transactions, and can potentially be applied to other domains as well.

**2.2.** The paper "**Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline**" by Wang et al (2016) proposes a simple yet effective baseline for time series classification using deep neural networks. The authors focus on developing end-to-end models without any heavy preprocessing or feature crafting. They introduce a Fully Convolutional Network (FCN) and explore the ResNet structure with very deep neural networks, both achieving premium performance compared to state-of-the-art approaches. The authors highlight the use of global average pooling in their convolutional model, which enables the exploitation of Class Activation Maps (CAMs) to identify the contributing region in the raw data for specific labels. The proposed models are considered a good starting point for future research and provide a simple choice for real-world applications. The paper analyses the models' generalization capability, learned features, network structures, and classification semantics. The findings suggest that the proposed models have a strong generalization capability and can effectively learn discriminative features from the raw data.

# 3. METHODOLOGY

## 3.1 Models Description:

## CNN (Convolutional Neural Network) :

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in a 3D input, assign importance (learnable weights and biases) to various aspects/objects in the input and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The structure of a ConvNet is similar to the connectivity pattern of neurons in the human brain and was modelled after how the visual cortex is organised. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

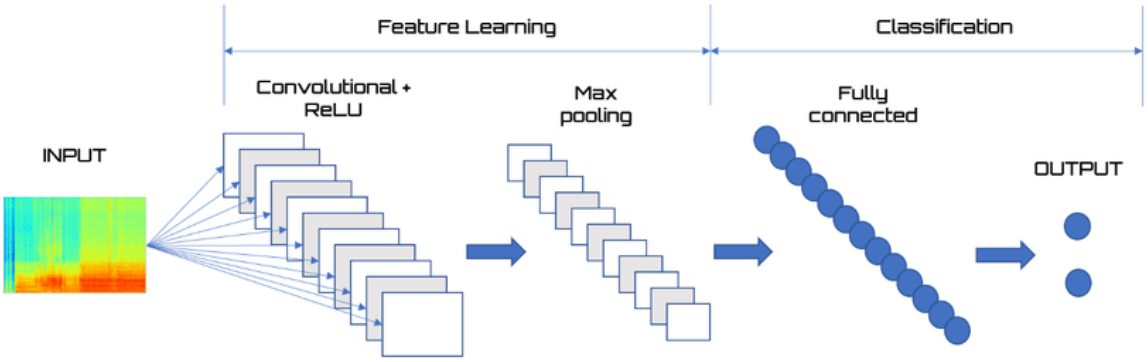



Fig: CNN Architecture

## Architecture:

For handling the imbalanced time series dataset, it is split and random sampled to have sufficient number of both fraud and normal cases in the training data to get accurate results. Thus, the training is balanced to let the model learn the characteristics of the minority and majority class equally. Then we will do the classification based on the training.

Before passing into the CNN model, the timeseries data is scaled to 3D since it is a one-dimensional image with only a temporal dimension, as opposed to the two dimensions that a normal image has: width and height. A multivariate time series may have an arbitrary number of channels, which may have different correlations and properties. The hyperparameters used in the

ConvNet model are Batch Normalization, Dropout and Flatten. Batch normalisation has a regularising effect since it adds noise to the inputs of every layer for each mini-batch This discourages overfitting since the model no longer produces deterministic values for a given training example alone. Dropout is a way of cutting too much association among features by dropping the weights (edges) at a probability. Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector. The flattened matrix is fed as input to the fully connected layer to classify the transactions. And the activation function we have used for hidden layers are "relu" while for the binary classifier output, it is "Sigmoid" function.

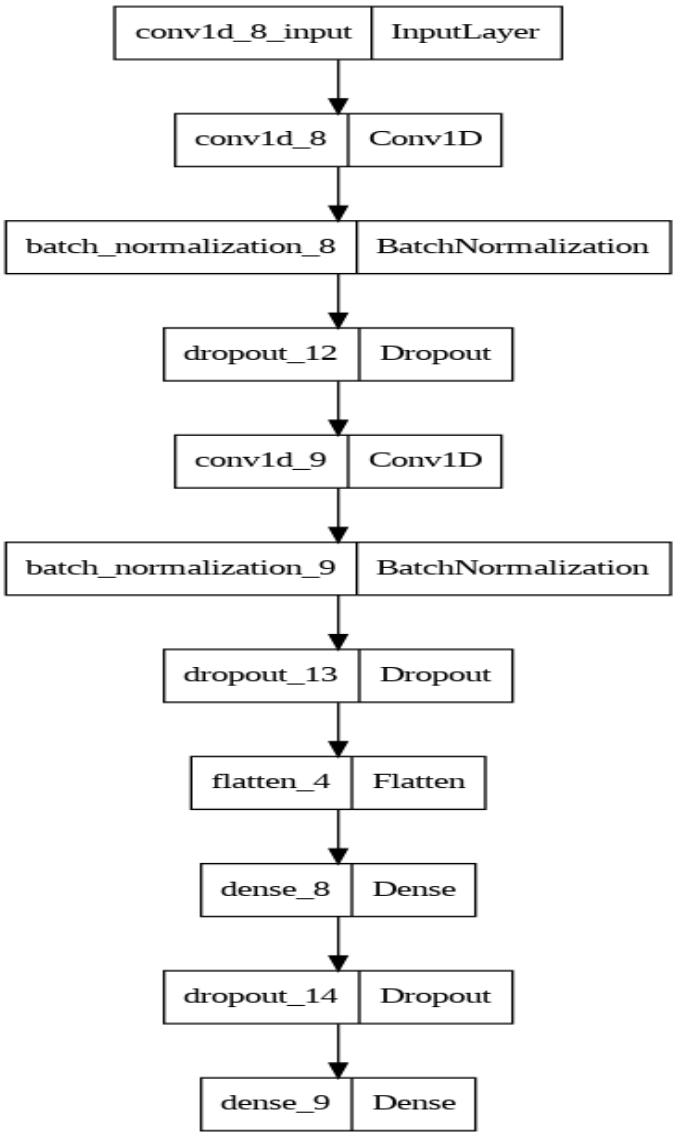



Figure: Plotted CNN model

## 2. Autoencoder:

## Model Description

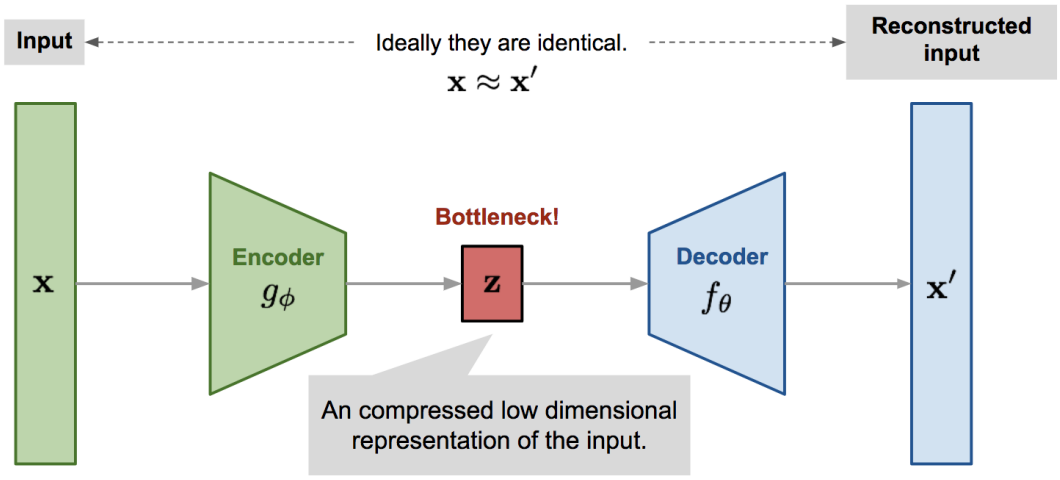



Figure: Autoencoder Architecture

Autoencoders are very useful in the field of unsupervised machine learning. It is a type of artificial neural network developed to find new, more effective methods to represent data. They can be used to compress data and reduce its size. Then reconstruct our original input to capture the essential components of the input data from a simple compressed version of it. Common applications for autoencoders include dimension reduction, data compression, feature extraction, and data denoising. They have been used in a wide range of applications, including anomaly detection, recommendation systems, image denoising, and image compression.

## Architecture :

The encoder network and the decoder network are the two fundamental components of an autoencoder's design. The input data is compressed by the encoder network into a lower-dimensional representation, which is then used by the decoder network to reconstruct the original data. Depending on the particular autoencoder type being utilised, an autoencoder's architecture might change. Yet a fundamental autoencoder generally has the following layers:

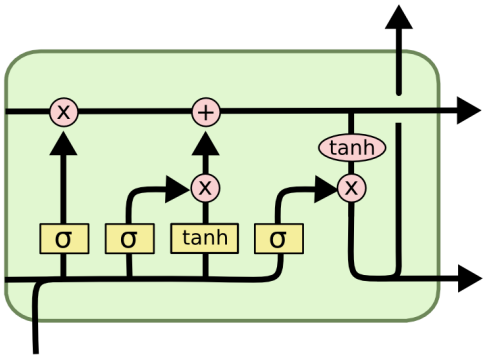
1. **Input layer:** The encoder network receives data from this layer, which receives it as input.
2. **Encoder Network :** An encoder network, also known as "latent space," is made up of one or more hidden layers that compress the input data into a lower-dimensional representation.
3. **Decoder Network :** A decoder network is made up of one or more hidden layers that use the compressed representation to reconstruct the original data.

4. **Output layer:** This layer creates the final output, which should be as similar to the original input data as possible.

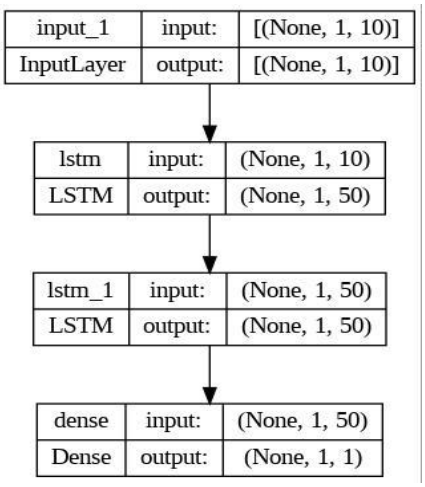
Depending on the complexity of the input and the task at hand, an autoencoder's layer count and size may also change. The amount of data that must be compressed during training for an autoencoder model depends on the bottleneck size. It is the autoencoder's most important hyperparameter. It can also act as a regularisation term. The quantity of layers is important when creating an autoencoder model. A smaller depth is quicker to process, while a larger depth makes the model more complicated. When the input to each layer gets smaller throughout the layers, the number of nodes gets smaller with each additional layer in the autoencoder. Finally, it's important to remember that the MSE loss and the L1 loss are two well-known losses for reconstruction.

## 3. RNN(LSTM)

LSTM models and RNN in general pass data as a feedback as well as feedforward.
These feedbacks allow the model to pass the output of a node to an earlier node, resulting in a short-term memory effect that will detect patterns that are hidden between data entries from different timestamps.




This model views the data as a time series. Since the credit card fraud detection database is timely-sorted and the pattern in which a credit card is used throught time can affect the judgment, Viewing the data as a timeseries can be helpful in finding out which transaction may be fraud. Our model architecture consists of two hidden layers of LSTMs and a detput layer with sigmoid as the activation function.




This model is trained on the credit card dataset after undersampling. There is a 0.3 dropout and a 0.2 recurrent dropout on LSTM nodes. The data by itself is an imbalance, hence only plotting the accuracy curve of the learning will not be informative. The following plot shows the changes in the recall, loss, precision and accuracy on the train(sampled) and validation(imbalanced) datasets:

# 4. Generative Adverserial Networks

Generative Adversarial Networks (GANs) can be utilized for credit card fraud detection by generating synthetic examples of fraudulent transactions that resemble the actual data. In the field of unsupervised machine learning, generative modeling entails the automatic discovery and learning of regularities or patterns in input data, allowing the model to output new examples that are plausible and could have been drawn from the original dataset.

In the context of credit card fraud detection, GANs can generate synthetic examples of fraudulent transactions using a generator that learns to produce the target output and a discriminator that distinguishes between real and synthetic data. The generator is trained to generate synthetic data that looks authentic while the discriminator is trained to differentiate between real and synthetic data.

The **generator** receives a random distribution, usually a Gaussian distribution, as input and generates data, such as transaction records. The random inputs can be viewed as latent representations or codings of the transaction records that need to be generated. Although generators perform a similar function to decoders in a variational autoencoder, they are trained differently.

The **discriminator** categorizes examples as either genuine or synthetic and takes either a synthetic or real transaction record as input. During training, the generator and discriminator have opposing goals. The discriminator tries to distinguish between genuine and synthetic transaction records, while the generator attempts to produce synthetic transaction records that look genuine enough to deceive the discriminator.
By generating synthetic transaction records, GANs can aid in improving the accuracy of credit card fraud detection systems by augmenting the limited dataset with a larger number of synthetic examples that can help the model better learn to detect fraudulent transactions.

## 4.1. Architecture:

## Training the GAN:

The code defines a **`GAN`** class which implements Generative Adversarial Networks (GANs). The **`GAN`** class has several methods that define the architecture of the generator and discriminator models, and a **`build_GAN`** method that creates a GAN model by combining the generator and discriminator models.

The **`generator`** method constructs a generative model using a given input tensor **`G_in`** and hyperparameters. The generative model consists of a fully connected layer, an activation layer, and another fully connected layer. It is compiled using the binary cross-entropy loss function and stochastic gradient descent optimizer with a given learning rate. The method returns a tuple of the generative model and the output tensor of the model.

The **`discriminator`** method builds and compiles a discriminative model using a Convolutional Neural Network. The method takes as input an input tensor **`D_in`** and several hyperparameters. The returned model is a **`tf.keras.Model`** object, which accepts **`D_in`** as input and produces **`D_out`** as output. The model is built using a Reshape layer, a Conv1D layer with ReLU activation function, a Dropout layer, a Flatten layer, a fully connected Dense layer with ReLU activation function, and a fully connected Dense layer with sigmoid activation function. The model is compiled using binary crossentropy loss and Adam optimizer with a given learning rate.

The **`set_trainability`** method sets the trainability of all layers of a given model to either True or False.

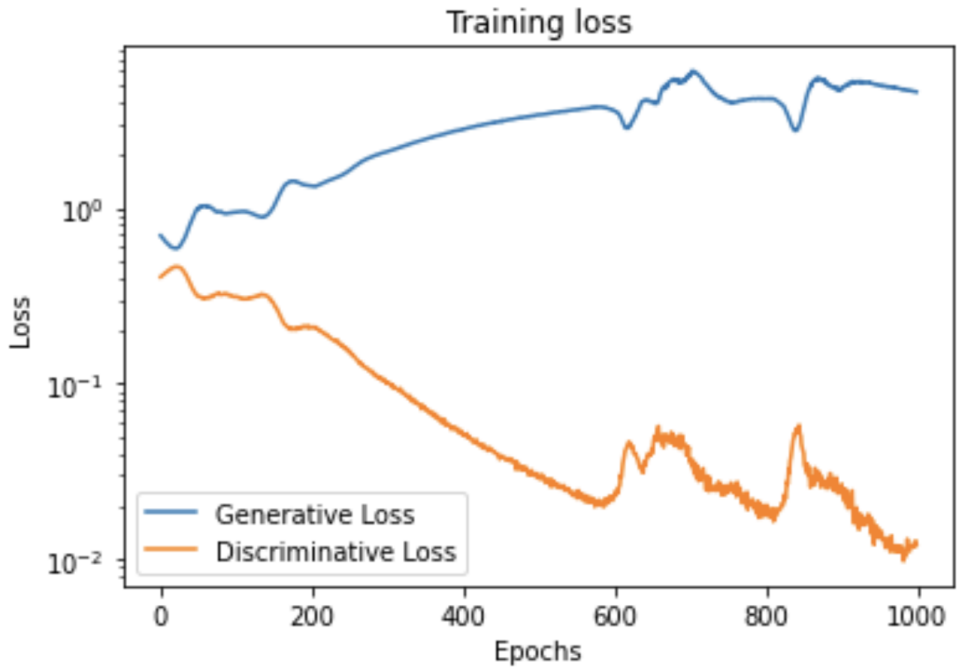
The **`build_GAN`** method creates a GAN model given an input tensor, generative model G, and discriminative model D.

The **`sample_data_and_gen`** method generates a sample of real and generated data for the GAN training process.

The **`pretrain`** method pretrains the discriminative model D with a single batch of generated and real data.

The **`generate_random_sample_noise`** method generates noise for the generative model G.

The **`train`** method trains the GAN model given a GAN model, generative model G, discriminative model D, number of epochs, noise dimension, batch size, and verbosity parameters.




## Building the Deep Neural Network:

This code defines a class called **`GANDeepLearningModel`** which is used to create and train a Generative Adversarial Network (GAN) model in Python. The GAN model is trained using a dataset called **`final_df`** which is first prepared by dropping the "Class" column and splitting it into training and testing sets using a test size of 0.1 and a random state of 42.

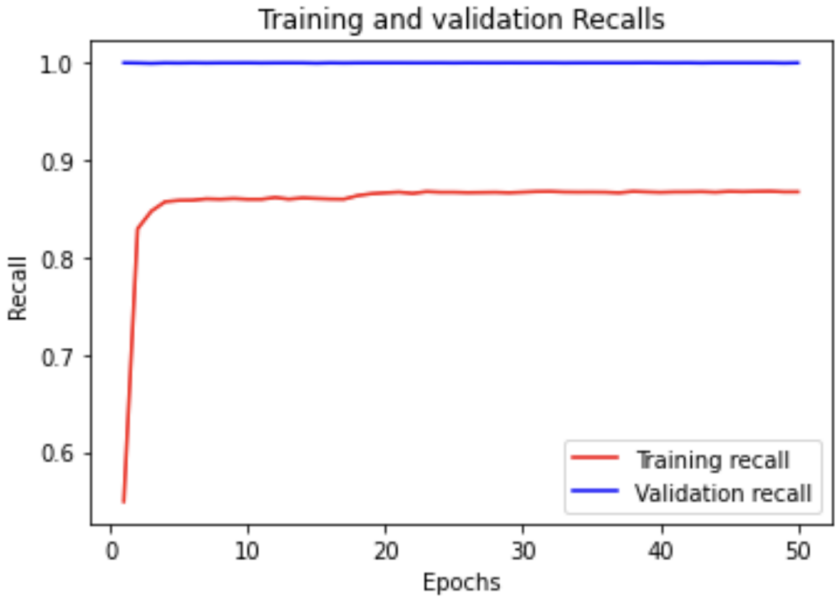
The training set is further split into a partial training set and a validation set with the first 100,000 samples used for validation. All input data is converted to numpy arrays of type float32.

The model architecture consists of a sequential stack of dense layers with ReLU activation and a dropout rate of 0.5. The final layer is a dense layer with a softmax activation function and two output nodes, representing the binary classification problem.

The model is compiled using the RMSprop optimizer with a learning rate of 0.001 and a binary cross-entropy loss function with recall as the evaluation metric. The function then performs one-hot encoding on the target variables of the training and validation sets.

Finally, the model is trained on the partial training set with 50 epochs, a batch size of 500, and the validation set is used for monitoring the model performance during training. The function returns the training history object that contains the loss and accuracy values for both the training and validation sets at each epoch.

There is also a method called **`evaluate_GAN_Recall`** which is used to evaluate the model's recall on a new dataset. It splits the dataset in the same way as the train_deep_learning_model method and performs the necessary data preparation before evaluating the model's recall on the test set.



# 4. SYSTEM REQUIREMENTS

## 4.1. Software Requirements:

- Google Colab
- Keras
- Tensorflow
- Matplotlib
- Pandas
- Numpy
- Sklearn
- Imblearn
- Streamlit UI
- Heroku

## 4.2. Dataset:

Credit card transactions dataset used that has 492 frauds in a total of 284,807 observations.

- The dataset consists of numerical values from the 28 ‘Principal Component Analysis (PCA)’ transformed features, namely V1 to V28.
- There is no metadata about the original features provided, so pre-analysis or feature study could not be done.
- The ‘Time’ and ‘Amount’ features are not transformed data.
- There is no missing value in the dataset.
- 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.
- The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning.
- Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

## 4.3 FUNCTIONAL REQUIREMENTS

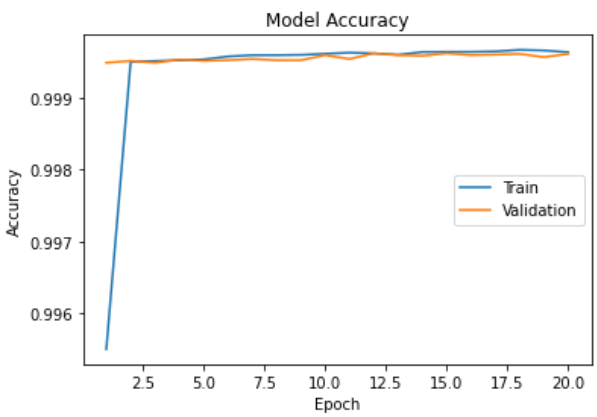
- The web application is a user-friendly interface hosted on Heruko that predicts if the credit card transaction details given by a user is a fraud or normal transaction.
- The Web UI created has 3 pages for Dataset analysis , user input for checking fraud transactions and evaluation of the 4 model results.
- We have compared all the 4 models results using F1, precision, recall and confusion matrix.

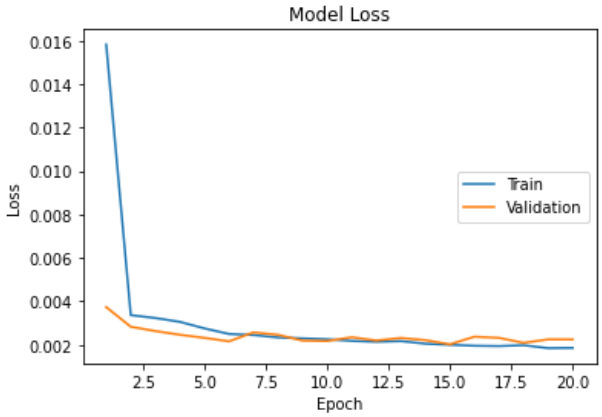
# 4. EVALUATION AND RESULTS

## 4.1. Model Outputs

## CNN Model:

By the below evaluation results we were able to train the CNN model better after balancing the dataset and upsampling the dataset. The test set thus gives better accuracy and lower loss for the model.




Evaluation results of the CNN model:

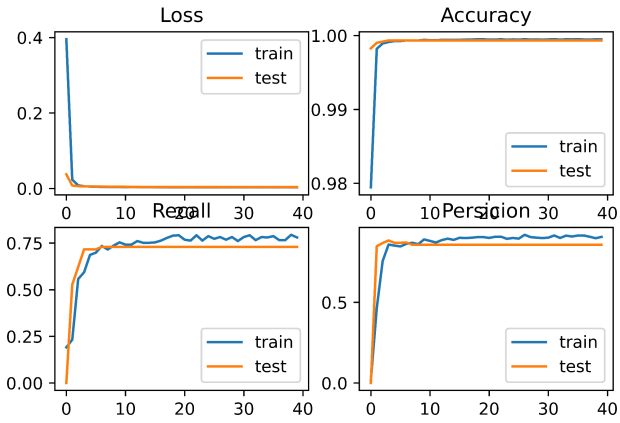
CNN evaluation metrics:

precision: 84.52%

recall: 73.95%

fscore: 78.88%

## LSTM Model:




We can see that recall and precision of the validation go still after a few epochs which is a result of the huge imbalance in the data.

Evaluation of the RNN model (Using lstm):

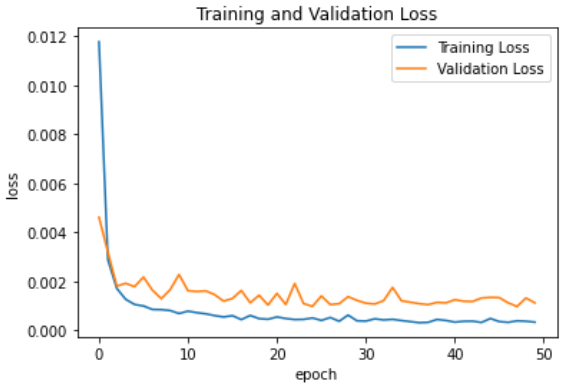
Precision: 81.70%

Recall: 90.54%

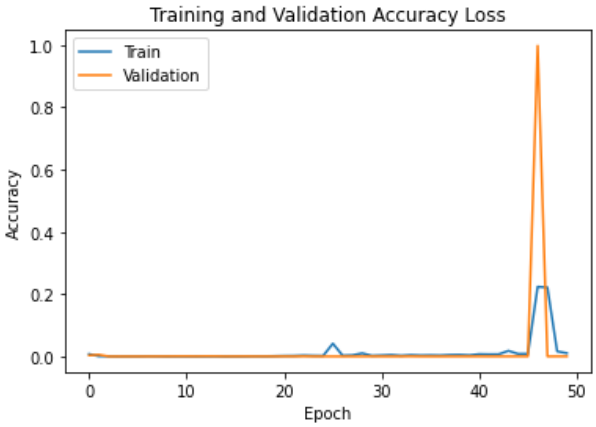
Fscore: 85.89%

## Autoencoder Model :

Graph below shows Training and Validation Loss:



Graph below shows Training and Validation Accuracy Loss:




Evaluation results of the Autoencoder model:

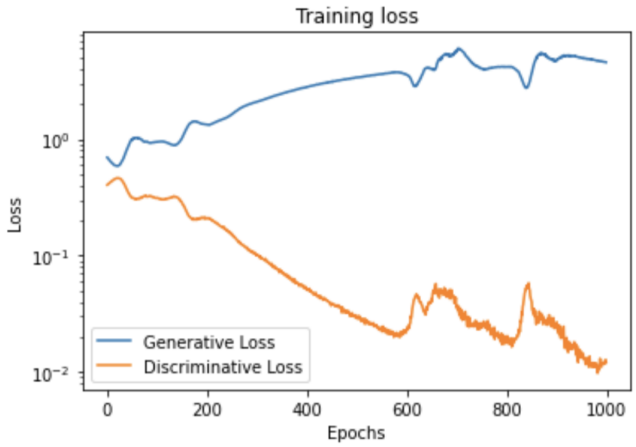
Autoencoder evaluation metrics:

precision: 59.50%

recall: 87.50%

fscore: 70.83%

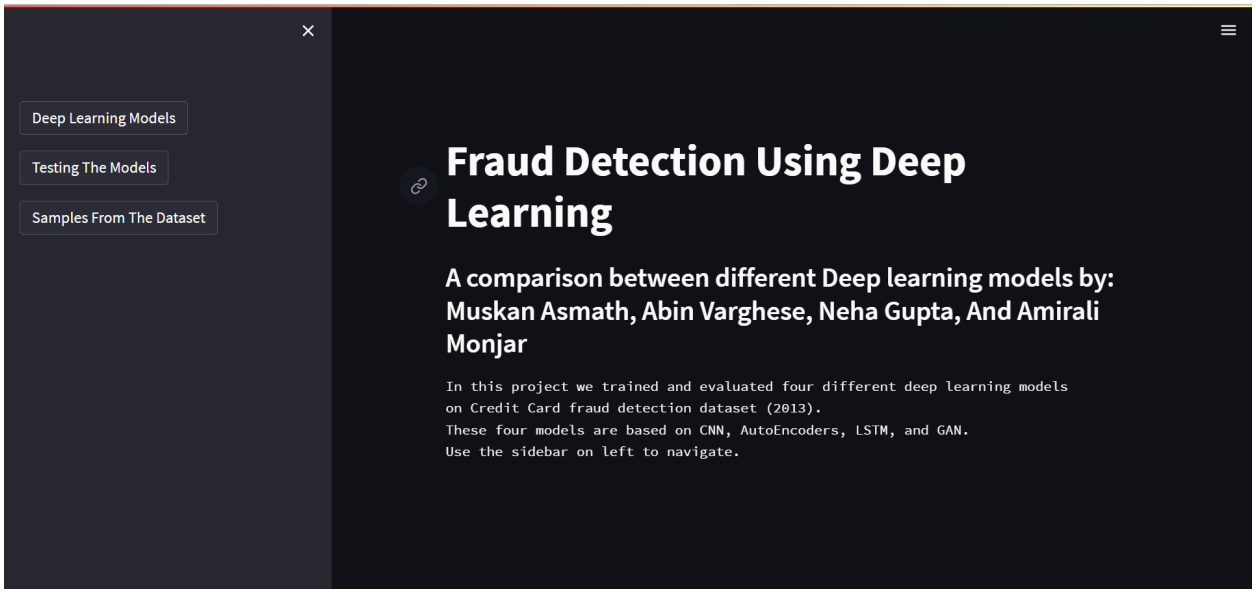
## Generative Adversarial Networks (GANs) Model :




recall: 72.41%

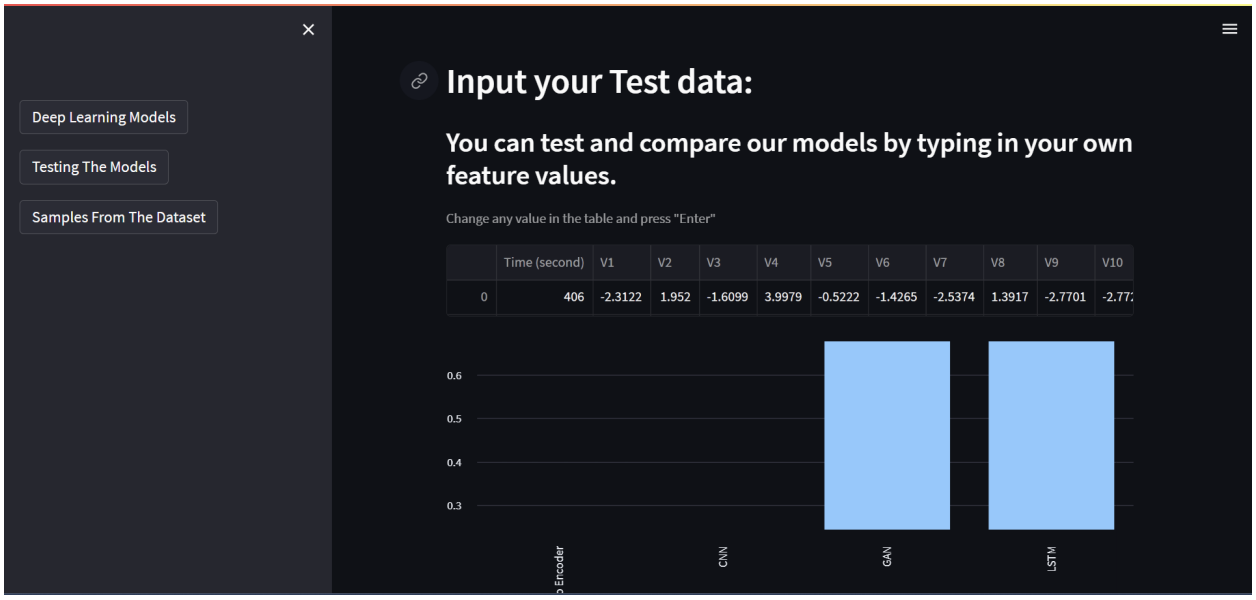
## 4.2. Screenshots of Web Application

Page 0 - Intro:

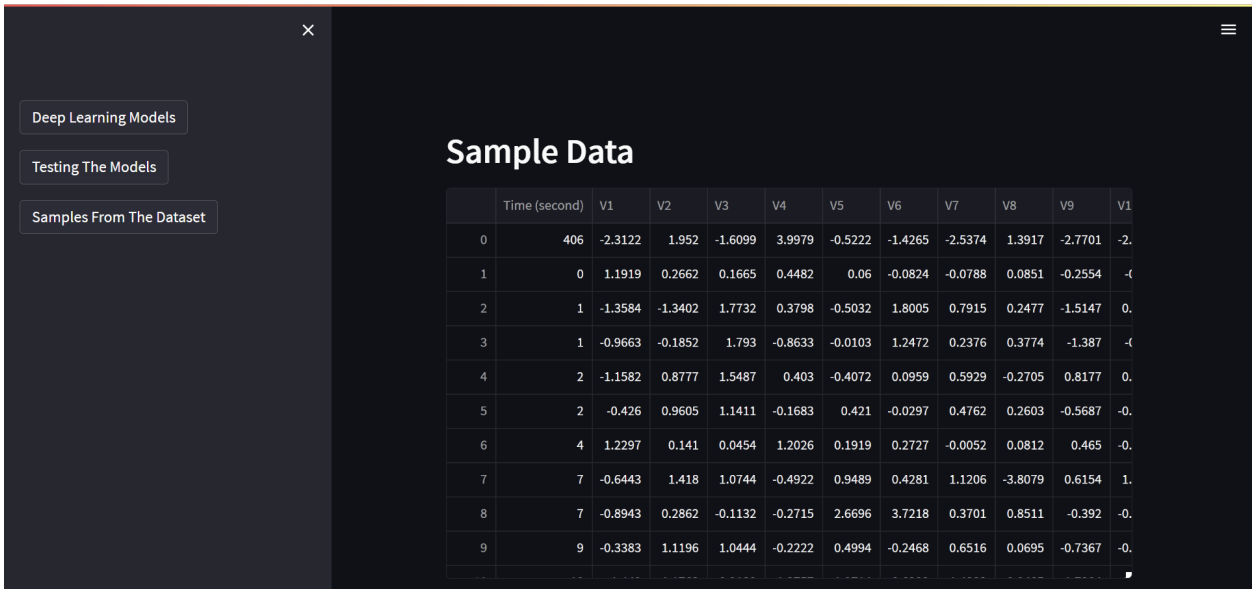



Page 1 - Overview and comparison of our models:

Page 2 - Testing models using User's input:




page 3 - Some sample entries of the Credit Card Dataset:



# 5. CONCLUSION

This project presents the overall description and design of the system of detecting credit card frauds . All the algorithms have been analyzed and compared on basis of accuracy on basis of predicting normal cases and outliers or frauds. We implemented and compared different types of algorithms. This was done to attain the best approach for the purpose.
Upon analyzing we got to know that LSTM is the best predicting model for the fraud cases with an recall of 90% followed by

## FUTURE ENHANCEMENTS

- Fine tune the hyperparameters to increase the precision and recall of the model in detecting fraud cases.

- Build a mobile application so that the application is more accessible and convenient for users.

# 6. REFERENCES


1. Brownlee, J. (2021) Random oversampling and undersampling for imbalanced classification, MachineLearningMastery.com. Available at: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/ (Accessed: March 13, 2023).

2. Lebichot, Bertrand & Le Borgne, Yann-Aël & He, Liyun & Oblé, Frédéric & Bontempi, Gianluca. (2019). Deep-Learning Domain Adaptation Techniques for Credit Cards Fraud Detection. 10.1007/978-3-030-16841-4_8.

3. Wen, T. and Keyes, R. (2019) Time series anomaly detection using convolutional neural networks and transfer learning, arXiv.org. Available at: https://arxiv.org/abs/1905.13628 (Accessed: March 12, 2023).

4. Wang, Z., Yan, W. and Oates, T. (2016) Time Series classification from scratch with Deep Neural Networks: A strong baseline, arXiv.org. Available at: https://arxiv.org/abs/1611.06455 (Accessed: March 14, 2023).

5. Credit card fraud detection using autoencoders in keras. Available at: https://curiousily.com/posts/credit-card-fraud-detection-using-autoencoders-in-keras/ (Accessed: March 12, 2023).