Indian Institute of Technology Hyderabad

Fraud Analytics (CS6890)

Assignment: 4 | Fraud Detection Using an autoencoder and variational autoencoder

Name	Roll Number
Shreesh Gupta	CS23MTECH12009
Hrishikesh Hemke	CS23MTECH14003
Manan Patel	CS23MTECH14006
Yug Patel	CS23MTECH14019
Bhargav Patel	CS23MTECH11026

Contents

1 | Problem statement

- To develop and evaluate neural autoencoder and variational autoencoder models for detecting fraudulent transactions within a credit card dataset.

2 | Description of the data set

- **Size:** The dataset consists of 284,807 credit card transactions.
- **Fraudulent Transactions:** Among the transactions, there are 492 instances labeled as fraudulent. This indicates an imbalanced dataset where the majority of transactions are non-fraudulent.
- **Attributes:** The dataset contains a total of 30 attributes:
  - 28 principal components
  - Time between each transaction and the first transaction in the dataset.
  - Amount paid for each transaction
- **Target Variable:** The target variable for the task is the "Class" column, which indicates whether a transaction is fraudulent (labeled as 1) or not fraudulent (labeled as 0).

3 | Our approach to solve the problem

3.1 | Data Exploration

- In the data exploration we have loaded the **creditcard.csv** into the pandas dataframe. Plotted a count plot to visualize the distribution of fraud vs. non-fraud transactions in a dataset.

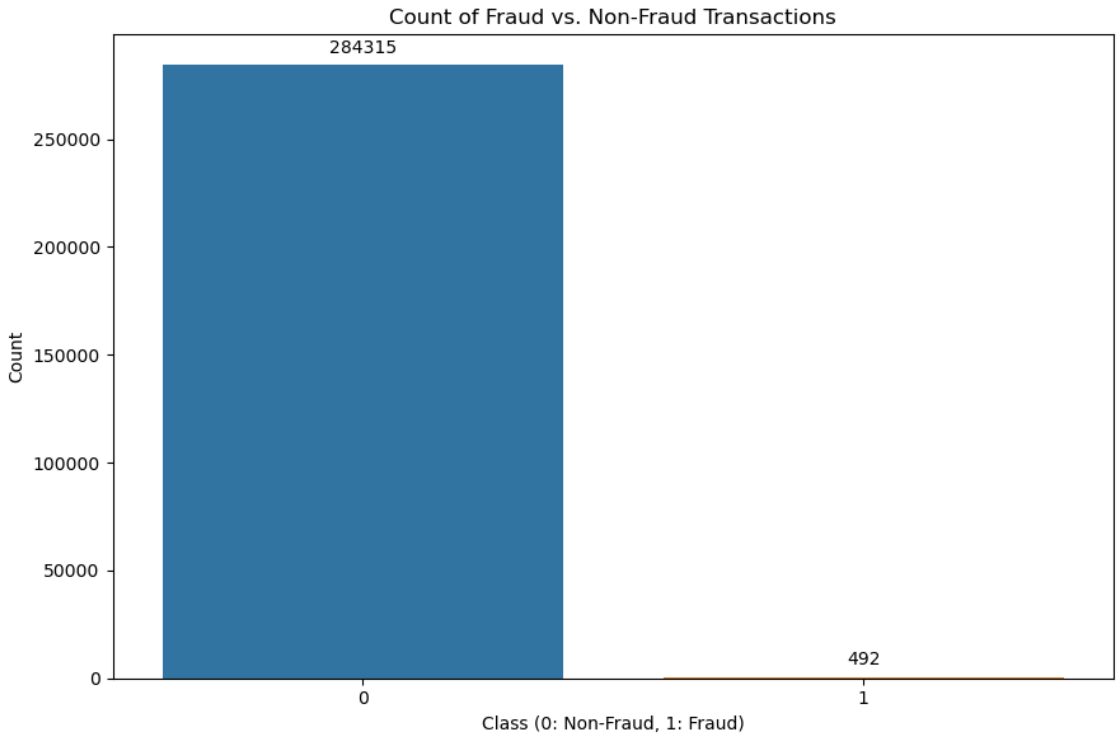

Figure 3.1: Count of Fraud vs. Non-Fraud Transactions

- Plotted histogram that visually compares the distribution of transaction amounts for normal and fraud transaction. Here we can detect differences between the two classes which are Non-Fraud and Fraud.

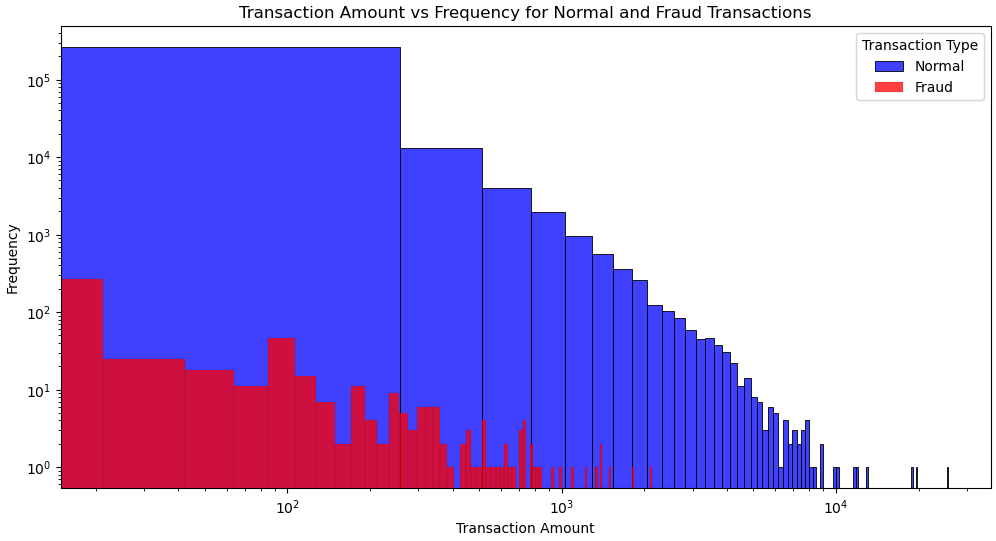

Figure 3.2: Transaction Amount vs Frequency for Normal and Fraud Transactions

- We have column name **TIME** which is in seconds. We are converting time into hours and min to do feature selection. Plotted graph between "Hours vs Amount for Normal and Fraud Transactions". We have observed that graph doesn't reveal much about the relationship between fraudulent and normal transactions against the time they occurred. Therefore, we are dropping the **'Time'** column.

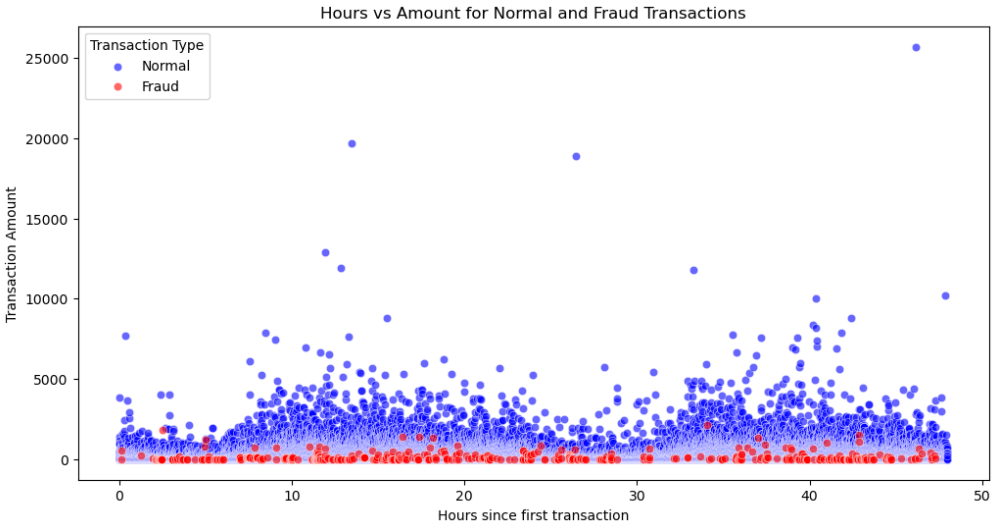

Figure 3.3: Hours vs Amount for Normal and Fraud Transactions

3.2 | Data Preprocessing

- **Handling Missing Values:** Check for any missing values in the dataset and handle them appropriately. Depending on the severity of missing data, options include imputation (e.g., replacing missing values with the mean or median) or removal of affected records.

- **Data Scaling:** Scale the features to ensure that they have similar ranges. We have converted 'Amount' to log scale to compress dynamic range and added 1 before taking the log to avoid log(0). We also standardize the created log feature column.
- **Dropping column**: We have dropped the **"TIME"** column since graph doesn't reveal much about the relationship between fraudulent and normal transactions against the time they occurred.
- **Train Test split**: We are splitting the data into train and test. Here test is 20% of whole data.

3.3 | Data After Preprocessing

3.3.1 | t-Distributed Stochastic Neighbor Embedding (t-SNE)

The t-SNE algorithm is a powerful dimensionality reduction technique that is particularly well-suited for visualizing high-dimensional data in two or three dimensions.

Why Apply t-SNE After Preprocessing:

- **Feature Compression:** t-SNE allows us to represent complex, multi-dimensional data in a 2D or 3D space, preserving the local structure of the data. After preprocessing, which often includes normalization and handling of outliers, t-SNE can provide a more accurate visual distinction between different classes.
- **Insightful Visuals:** It provides a visual means to assess the separability of different classes in the dataset, which can be an indirect measure of how well a machine learning model might perform in classifying the data.
- **Anomaly Detection:** For fraud detection, we're particularly interested in how well the fraud cases stand out from normal cases, which can guide further feature engineering and model selection.

3.3.2 | Our Visualisation

- Here we observe there are two cluster which are fraud and non-fraud. Both of the clusters are distinctly separable which implies the machine learning algorithms or deep learning algorithms will be able to easily learn them.

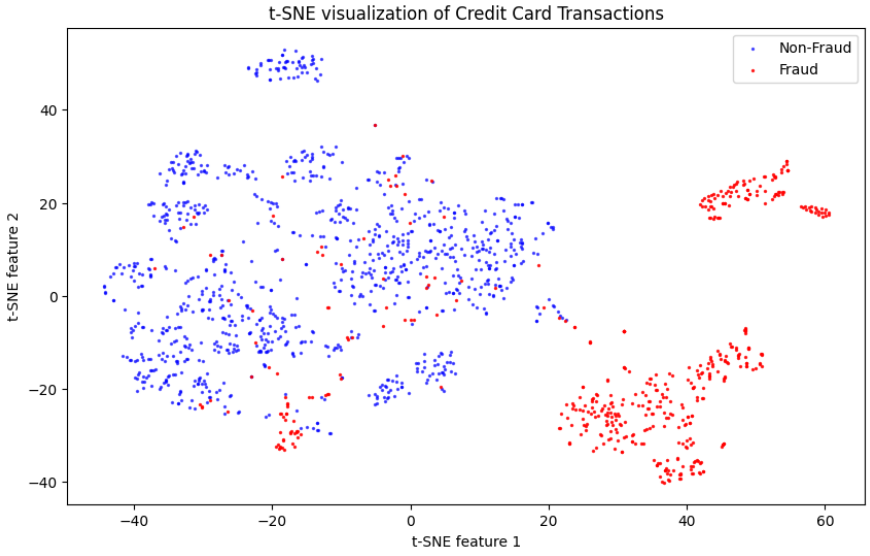

Figure 3.4: t-SNE visualisation

4 | Autoencoder

4.1 | Introduction

An autoencoder is a type of neural network that learns to copy its input to its output, ideally creating a "compressed" representation of the data in the process. We have designed with a symmetrical architecture, consisting of two main components: the encoder and the decoder.

4.2 | Architecture

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 29)	0
dense_8 (Dense)	(None, 14)	420
dense_9 (Dense)	(None, 7)	105
dense_10 (Dense)	(None, 7)	56
dense_11 (Dense)	(None, 29)	232

Total params: 2,441 (9.54 KB)
Trainable params: 813 (3.18 KB)
Non-trainable params: 0 (0.00 B)

1. **Input Layer:** The input dimensionality (input-dim) is set to match the number of features in the credit card dataset (X-train.shape[1]). This layer takes the raw input data.
2. **First Encoder Layer:** This layer uses 14 neurons (encoding-dim) and a tanh activation function, optimized with L1 regularization to encourage sparsity in the representations. The L1 regularization helps in reducing overfitting by penalizing the weights, useful in high-dimensional data.
3. **Second Encoder Layer:** Reduces the dimensionality further to half of the previous layer using a relu activation function. This layer focuses on capturing the most significant features from the data, essential for identifying underlying patterns in normal and fraudulent transactions.
4. **First Decoder Layer:** Begins the process of reconstructing the input data from the encoded state, using the same number of neurons as the second encoder layer and a tanh activation function. This symmetry aids in learning a compressed knowledge of the data.
5. **Second Decoder Layer:** Continues to reconstruct the data, aiming to restore it to its original dimensionality with a relu activation function. The objective is to output a representation of the input data that is as close as possible to the original input.
6. **Output Layer:** Outputs the reconstructed data, which should ideally resemble the original input if the model has learned well.

We are using the above mentioned architecture because of the following reasons:

1. Implement the undercomplete autoencoder to ensure that our architecture does not merely learn to replicate the input data, but instead learns a compressed representation of the significant features

4.3 | Training results

1. **Loss vs epoch graph:** Here we can observe the training and testing loss are decreasing over the epoch. This implies the autoencoder is learning.

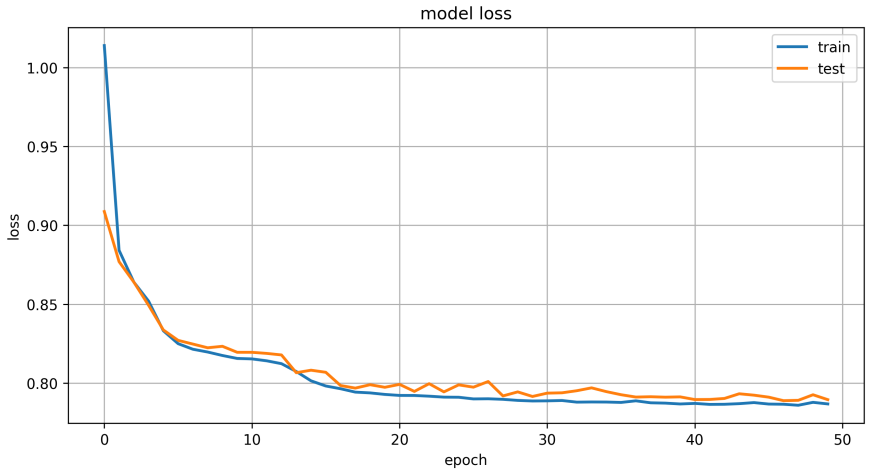

Figure 4.1: Loss vs epoch graph

2. **Reconstruction Error for Fraud Transactions:** The histograms compare the distribution of reconstruction error in normal vs. fraud transactions. Normal transactions have a tight clustering of errors close to zero, indicating good model reconstruction. Fraud transactions show a wider spread of errors, suggesting anomalies the model couldn't reconstruct well, which is typical in fraud detection

Figure 4.2: Reconstruction Error for Fraud Transactions

4.4 | Results

1. **ROC curve:** The ROC curve below represents the performance of a classification model, with an AUC (Area Under the Curve) of 0.9574, indicating a high level of accuracy. The true positive rate is high across different thresholds, while maintaining a low false positive rate. This model is effective at distinguishing between the classes (e.g., fraud and non-fraud).

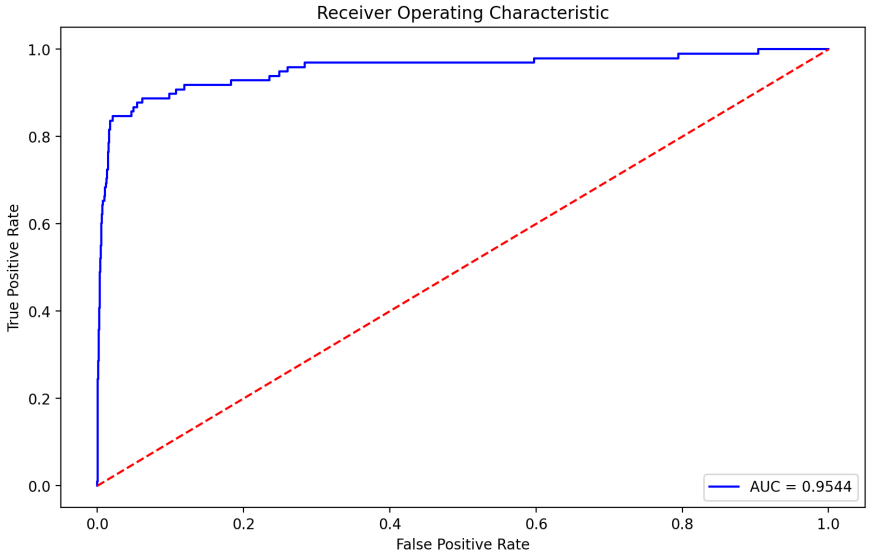

Figure 4.3: ROC curve

2. **Recall vs Precision:** The Precision-Recall curve below shows the trade-off between precision and recall for a classification model at different thresholds. High precision near the y-axis suggests that the model is accurate when it predicts positive classes, but recall drops quickly, indicating it doesn't capture all positive cases. The model is precise but not highly sensitive to all positives.

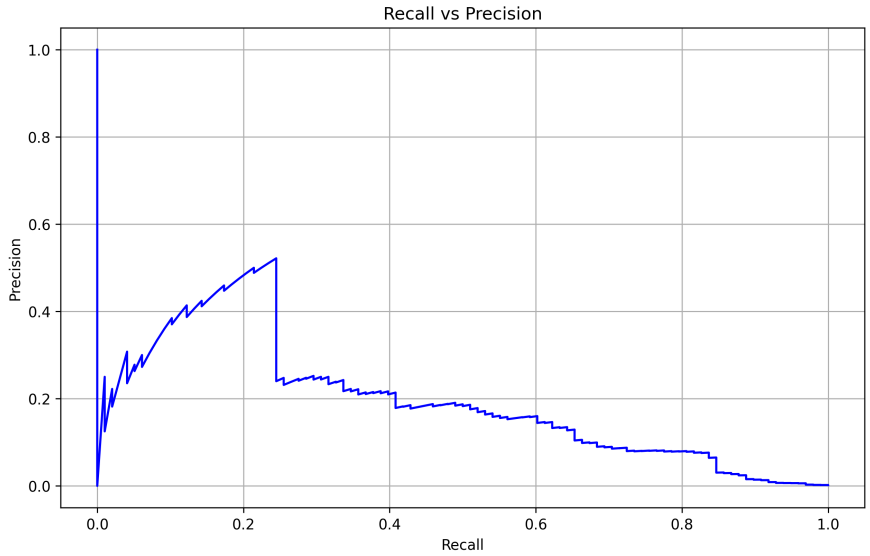

Figure 4.4: Recall vs Precision

3. **Precision for different threshold values:** The graph shows how the precision of a predictive model varies with different threshold values. As the threshold increases, precision fluctuates and tends to decrease before shooting up to perfect precision at the highest threshold. This suggests that the model becomes very selective, possibly only predicting positives when very sure, which may not be practical if high recall is also desired.

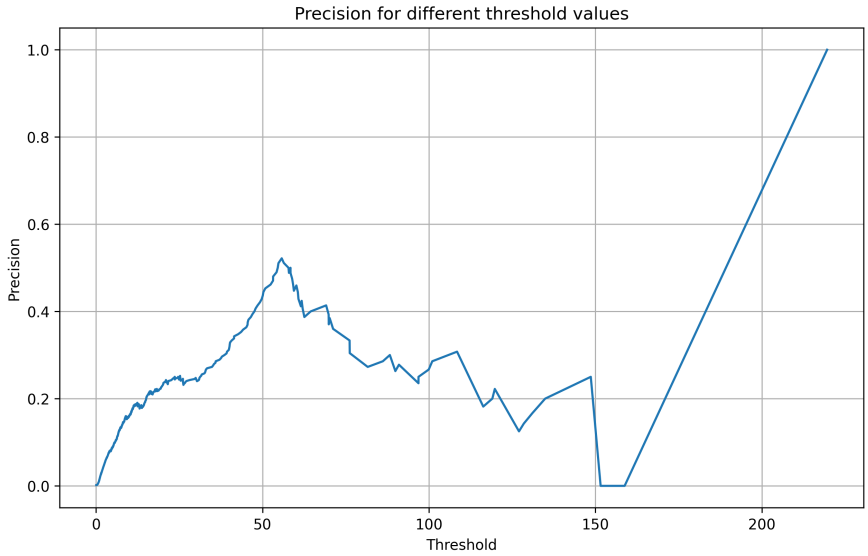

Figure 4.5: Precision for different threshold values

4. **Threshold-Recall curve:** The graph illustrates recall as a function of varying reconstruction error thresholds in a predictive model. Initially, recall is high, indicating most positive cases are captured, but it declines sharply as the threshold increases, implying fewer positives are detected. At high threshold levels, recall is low, suggesting many positive instances are missed by the model.

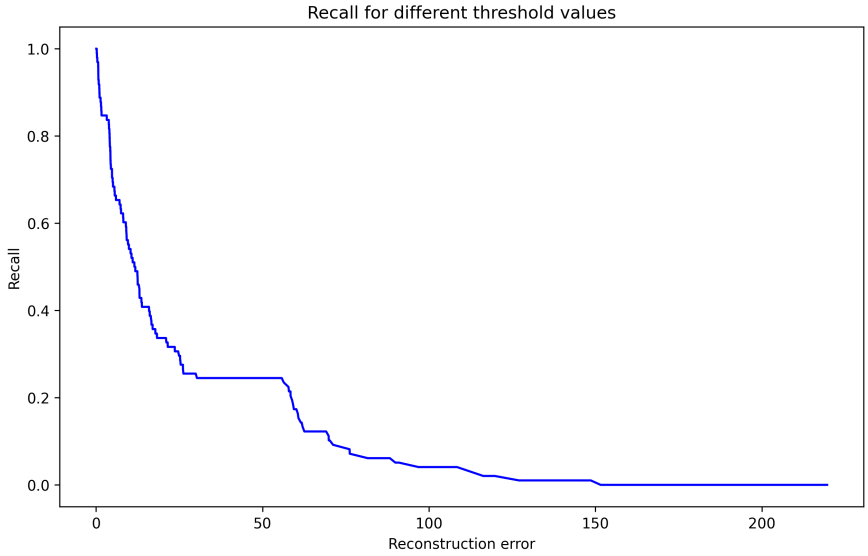

Figure 4.6: Threshold-Recall curve

5. **Confusion matrix:** This confusion matrix shows that for a binary classification problem, the classifier correctly identified 54,964 normal cases and 83 fraud cases. However, it incorrectly classified 1,900 normal cases as fraud (false positives) and failed to identify 15 fraud cases (false negatives). The matrix suggests the model is relatively good at detecting normal cases but less effective at identifying fraud.

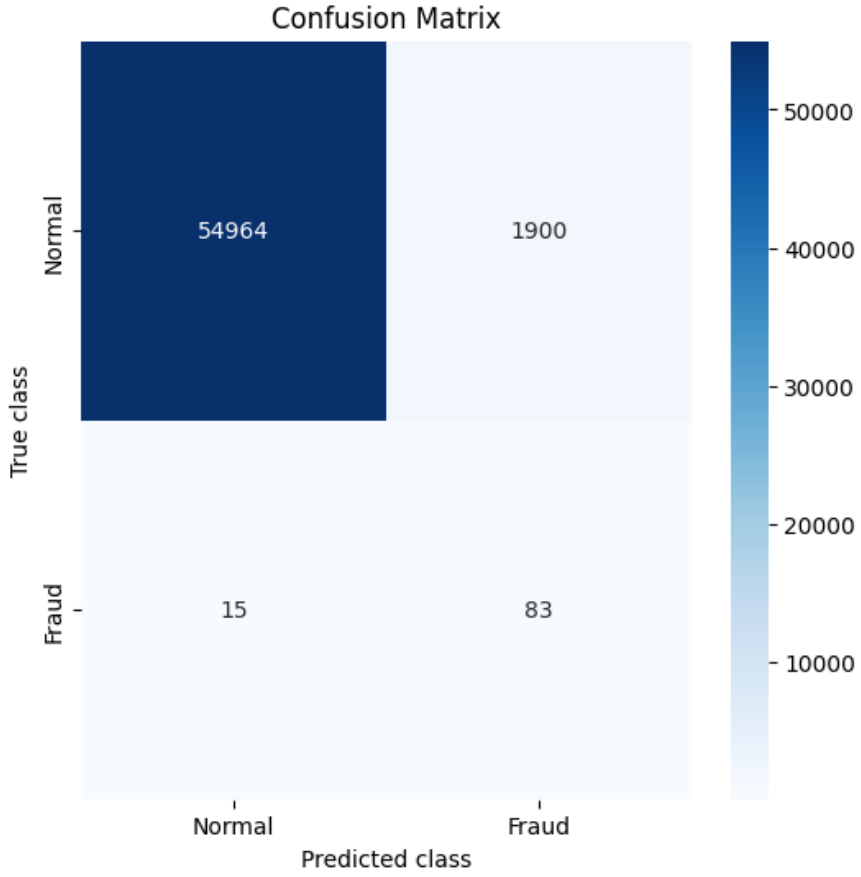

Figure 4.7: Confusion matrix

6. F1 score

The calculation of the optimal threshold that maximizes the F1 score involves these concise mathematical steps:

[a] Precision and Recall Calculation:

- Generate a set of predictions or error scores S.
- Use the function `precision_recall_curve(`y, S`)` to obtain precision (P) and recall (R) values at various thresholds (T) derived from S.

[b] F1 Score Calculation for Each Threshold:

- For each threshold t in T, calculate the F1 score:

$$F1(t) = 2 \times \frac{P(t) \times R(t)}{P(t) + R(t)}$$

This formula ensures that the F1 score is computed for each possible threshold based on the corresponding precision and recall values.

[c] Optimal Threshold Determination:

- Identify the threshold t_{opt} that maximizes $F1(t)$:

$$t_{\text{opt}} = \text{argmax}\, F1(t)$$

Here, argmax is the operation that finds the argument (threshold) that results in the maximum F1 score.

[d] Using the Optimal Threshold:

- With t_{opt} known, classify future data points by comparing their score s against t_{opt}.

Table 4.1: A Simple 2x2 Table

Optimal Threshold	**1.34081953**
Maximum F1 Score 0.910052910	**0.915789473** height**Optimal F1 Score**

5 | Variational Autoencoder

5.1 | Introduction

Variational Autoencoders (VAEs) are a class of generative models that combine deep learning with Bayesian inference to produce new data samples. They consist of two main components: an encoder that maps input data into a latent space representation, and a decoder that reconstructs the input from this latent space. VAEs are trained to minimize reconstruction loss and a regularization term based on the Kullback-Leibler divergence, which ensures a meaningful latent space. This framework allows VAEs to effectively generate high-quality, diverse samples across various applications

5.2 | Architecture

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 29)	0	-
dense_8 (Dense)	(None, 64)	1,920	input_layer_2[0][0]
dense_9 (Dense)	(None, 128)	8,320	dense_8[0][0]
z_mean (Dense)	(None, 2)	258	dense_9[0][0]
z_log_var (Dense)	(None, 2)	258	dense_9[0][0]
sampling (Sampling)	(None, 2)	0	z_mean[0][0], z_log_var[0][0]

Total params: 10,756 (42.02 KB)
Trainable params: 10,756 (42.02 KB)
Non-trainable params: 0 (0.00 B)

1. Encoder: The encoder model is responsible for mapping the input data to a latent space representation. It approximates the posterior distribution of the latent variables given the input.

   - **Input Layer**:
     - **Function**: Receives the original input data.
     - **Shape**: (input_dim,) where input_dim is the dimension of the input data.
   - **Dense Layer (64 units)**:
     - **Activation**: ReLU (Rectified Linear Unit).
     - **Function**: Applies a transformation that introduces non-linearity, allowing the model to learn more complex patterns.
   - **Dense Layer (128 units)**:
     - **Activation**: ReLU.
     - **Function**: Further transforms the data, increasing the representational capacity of the network.
   - **z_mean and z_log_var Layers (latent_dim units each)**:
     - **Function**: Outputs the mean (z_{mean}) and logarithm of variance ($z_{\text{log\_var}}$) of the latent distribution. These layers define the parameters of the Gaussian distribution assumed for the latent variables.
   - **Sampling Layer**:
     - **Function**: Uses the reparameterization trick to sample from the latent distribution defined by z_{mean} and $z_{\text{log\_var}}$. This step is crucial for backpropagation as it allows gradients to pass through the stochastic sampling process.

2. Decoder: The decoder model maps the latent space representation back to the original input space, aiming to reconstruct the input data.

   - **Input Layer (Latent Space)**:

- **Name**: z_sampling.
- **Shape**: (latent_dim,).
- **Function**: Takes a sampled latent vector as input.

- **Dense Layer (128 units)**:
  - **Activation**: ReLU.
  - **Function**: Transforms the latent vector into a higher-dimensional space, beginning the process of reconstruction.
- **Dense Layer (64 units)**:
  - **Activation**: ReLU.
  - **Function**: Continues to reconstruct the original data from the transformed latent representation.
- **Output Layer**:
  - **Activation**: Sigmoid.
  - **Function**: Produces the final reconstruction of the input data. The sigmoid activation ensures that the output values are in the range [0, 1], suitable if the input data is normalized.

3. VAE Model: The VAE model integrates the encoder and decoder into a single model for training.

- **Integration of Encoder and Decoder**:
  - **Function**: The VAE model takes input data, processes it through the encoder to obtain the latent variables, and then passes these through the decoder to reconstruct the input data.
- **KL Divergence Loss**:
  - **Function**: Adds a regularization term to the loss function, representing the Kullback-Leibler divergence between the approximated latent distribution and a standard Gaussian distribution. This term encourages the latent variables to follow a distribution that closely resembles the Gaussian distribution, helping in learning a well-formed latent space and improving the generative capability of the model.
- **Total Loss**:
  - **Function**: The total loss for the VAE is a combination of the reconstruction loss (how well the decoder reconstructs the input data) and the KL divergence loss. This setup helps in balancing the reconstruction quality with the distribution of latent variables.

We are using the above mentioned architecture because of the following reasons:

1. Implement the undercomplete autoencoder to ensure that our architecture does not merely learn to replicate the input data, but instead learns a compressed representation of the significant features

5.3 | Training results

1. **Loss vs epoch graph:** The chart shows the training and validation loss of a model over 50 epochs. The training loss decreases sharply and flattens almost immediately, indicating the model quickly learned to fit the training data. The validation loss is low and stable across epochs, suggesting the model generalizes well and is not overfitting.

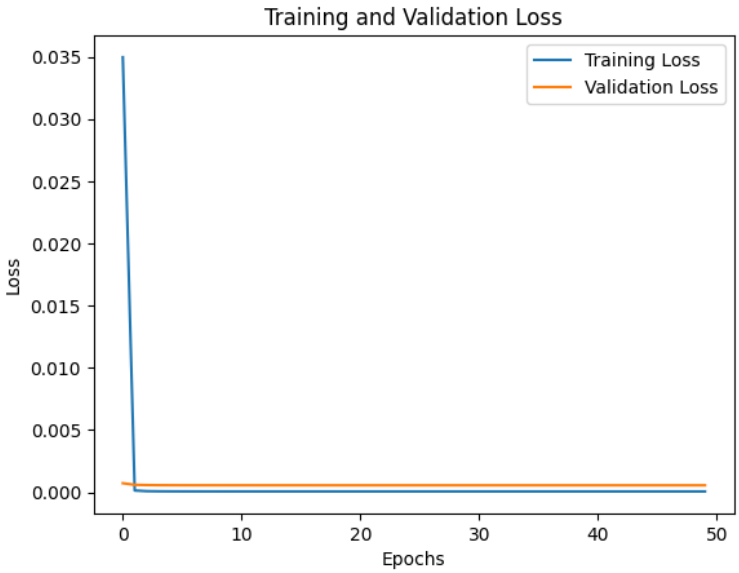

Figure 5.1: Loss vs epoch graph

2. **Reconstruction Error for Fraud Transactions:** Normal transactions are tightly clustered around a low reconstruction error, indicating accurate model predictions. Fraud transactions are less frequent and have a wider distribution of reconstruction errors, likely due to their anomalous nature which the model finds harder to reconstruct accurately.

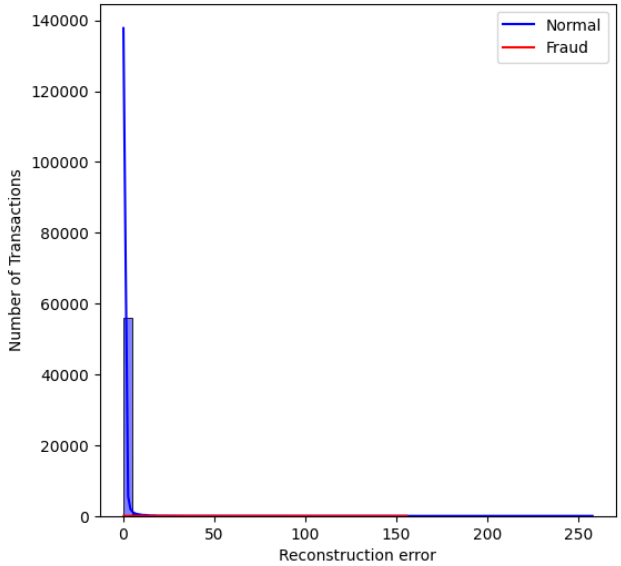

Figure 5.2: Reconstruction Error for Fraud Transactions

5.4 | Results

1. **ROC curve:** The ROC curve depicts the performance of a Variational Autoencoder (VAE) with an AUC of 0.953, which is quite high, indicating strong discriminative power. The curve stays close to the top-left corner, suggesting a high true positive rate with a low false positive rate, which is desirable in many classification tasks.

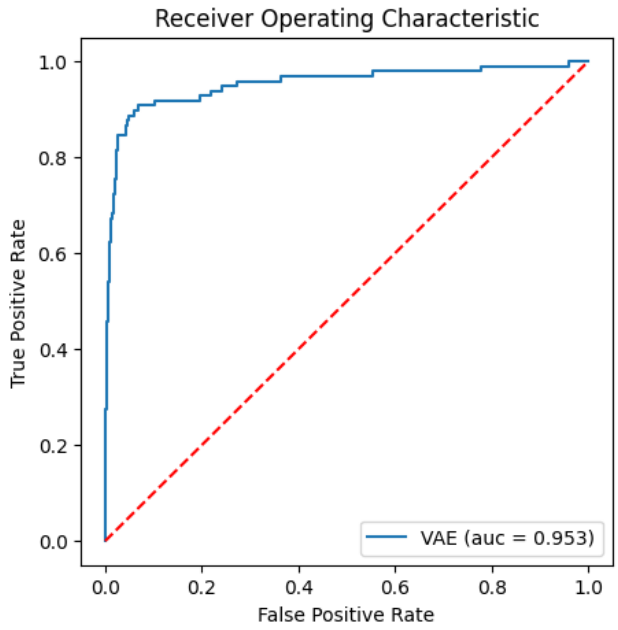

Figure 5.3: ROC curve

2. **Recall vs Precision:** The graph shows precision and recall values for various thresholds in a classification model. Precision starts high but quickly drops and fluctuates as the threshold increases, while recall decreases more steadily. The dashed line may represent an optimal threshold where a balance between precision and recall is achieved, often used to optimize a model's performance on both metrics.

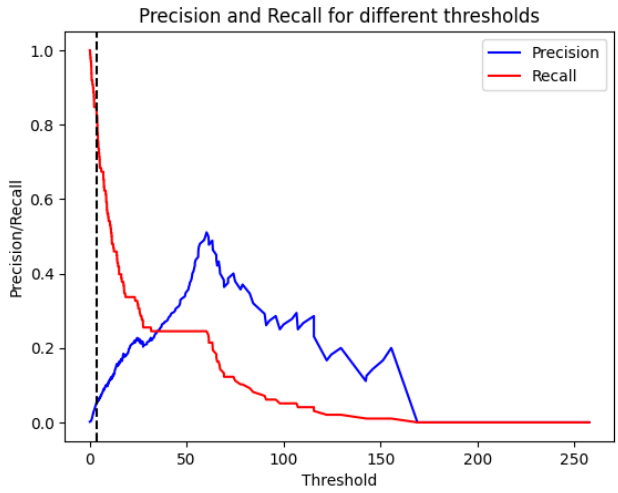

Figure 5.4: Recall vs Precision

3. **Reconstruction log probability:** The histogram compares the log probability of reconstruction for non-fraud and fraud transactions. Non-fraud transactions show a high frequency of high log probability scores, indicating the model reconstructs them with high likelihood. Fraud transactions, however, have a lower frequency and are spread out across lower log probability values, suggesting they are less likely to be reconstructed accurately by the model.

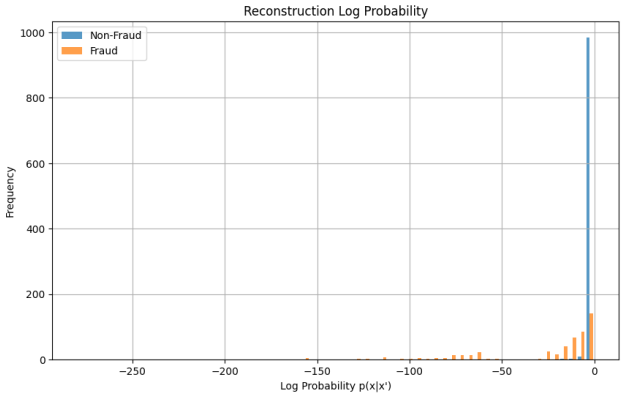

Figure 5.5: Reconstruction log probability

4. **Latent space mean:** This scatter plot represents data points in a two-dimensional latent space, with each point colored according to its class. The dense cluster of purple points near the center suggests a concentration of one class, while the sparser yellow points indicate another class. The plot could be showing how a model like a Variational Autoencoder separates features of different classes in its learned latent space.

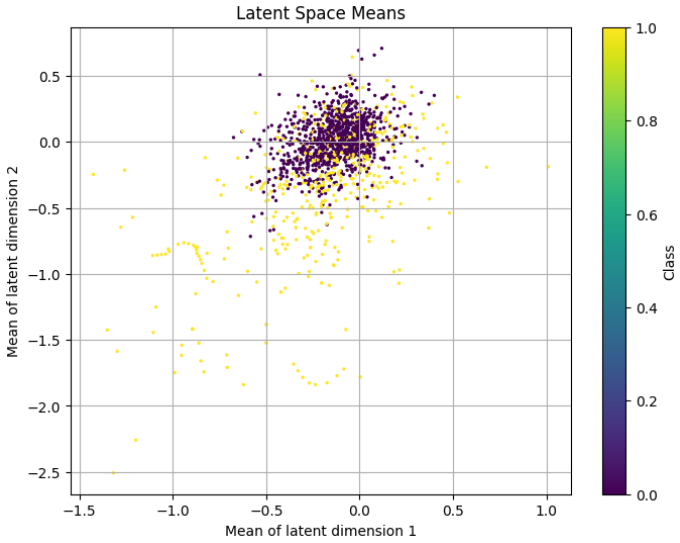

Figure 5.6: Latent space mean

5. **Latent space standard deviation:** The scatter plot illustrates data points in terms of their standard deviation across two latent dimensions, colored by class. A cluster of purple points indicates a class with lower variability, while yellow points dispersed throughout indicate higher variability in another class. This visualization can be useful for understanding the spread and overlap of the different classes in a model's latent space

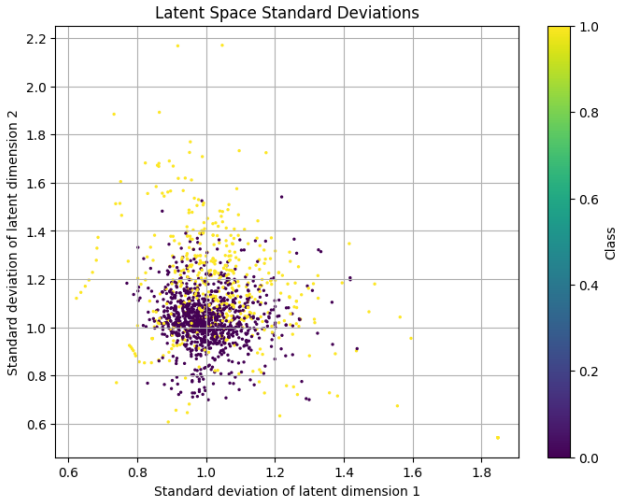

Figure 5.7: Latent space standard deviation

6. **Confusion matrix:** The confusion matrix shows a classifier's predictions, with 55,451 true negatives and 81 true positives, indicating correct classifications for normal and fraud cases respectively. However, there are 1,413 false positives and 17 false negatives, showing instances where the model incorrectly classified normal transactions as fraud and missed identifying some fraud transactions.

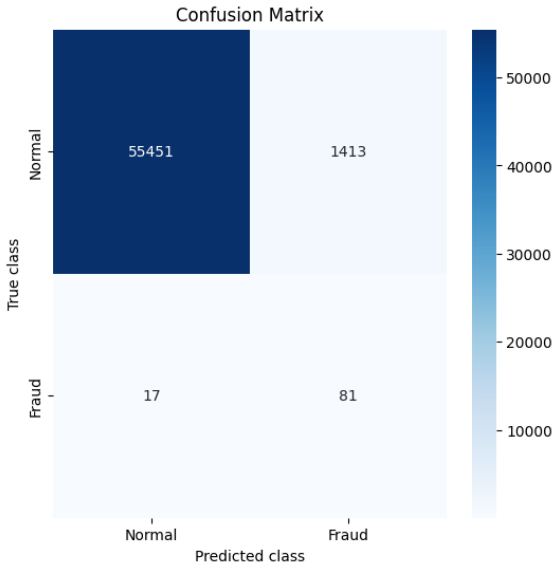

Figure 5.8: Confusion matrix

7. **F1 score:** With the optimal threshold set at approximately 0.00076, the F1 Score achieved is 0.9231. This indicates a good balance between precision and recall, suggesting that the model performs well in classifying the positive class while maintaining a low rate of false positives and false negatives. An F1 Score close to 1 is typically indicative of a highly effective model.
Optimal Threshold: 0.0007648746
Optimal F1 Score: 0.923076923076923

Table 5.1: A Simple 2x2 Table

Optimal Threshold	**0.0007648746**
Optimal F1 Score	**0.923076923076923**

6 | References

[1] https://venelinvalkov.medium.com/credit-card-fraud-detection-using-autoencoders-in-keras-tensorflow-for-hackers-part-vii-20e0c85301bd