PROJECT REPORT ON

DEEP LEARNING WORKSHOP WITH PYTHON (CSE3194)

FRAUD DETECTION USING NEURAL NETWORKS

Submitted in partial fulfillment of the
requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

Submitted by:

NEELACHAL MOHANTY	**2241013255**
T VINITA	**2241022012**
BHABNA PATTNAIK	**2241016347**

Center for Artificial Intelligence & Machine Learning
Department of Computer Science and Engineering
Institute of Technical Education and Research
Sikhsha 'O' Anusandhan
(Deemed to be University)
Bhubaneswar

May, 2025

Declaration

We hereby declare that the project report titled **"FRAUD DETECTION USING NEURAL NETWORKS"** is our own work, carried out under the guidance of JAGSEER SINGH We have not plagiarized any content and have duly cited all references.

NEELACHAL MOHANTY BHABNA PATTNAIK

T VINITA

Acknowledgement

We, the undersigned students of B. Tech. of Computer Science (AI & ML) Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled “Fraud Detection System Using Neural Networks” submitted to Siksha ‘O’ Anusandhan Deemed to be University, Bhubaneswar for the partial fulfillment of the subject Deep Learning Workshop with Python(CSE 3194) have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

Table of Contents

Contents

Abstract

This project focuses on developing and evaluating **neural network models** for **credit card fraud detection**. A primary challenge is the **severe class imbalance** in the dataset, where fraudulent transactions are significantly outnumbered by legitimate ones. The methodology involves preprocessing the transaction data, specifically handling the class imbalance using **SMOTE and Tomek Links** techniques. Sequential neural network models are built and trained, experimenting with different activation functions and optimizers to learn complex patterns. Model performance is rigorously evaluated using metrics tailored for imbalanced data, including accuracy, precision, recall, F1 score, and the **Area Under the Precision-Recall Curve (AUPRC)**. The objective is to effectively train models for accurate fraud detection despite the imbalance and compare the impact of various model configurations.

Keywords: Credit Card Fraud, Fraud Detection, Neural Networks, Class Imbalance, Pipeline,SMOTE, Tomek Links, AUPRC, Evaluation Metrics.

Chapter 1 – Introduction

1.1. Background and motivation

Credit card fraud is a significant problem that costs billions of dollars annually[2][6]. Projections indicate the value of fraudulent card transactions worldwide is set to increase [6]. For businesses and financial institutions, the impact is severe [6][7]. This includes **financial loss** from chargebacks, transaction fees, and lost inventory [6][7]. Fraud can also lead to **damaged reputation**, as customers may avoid businesses where they were defrauded [6][7]. Furthermore, frequent fraud or chargebacks can result in **increased fees or restrictions from payment processors [7],** and resolving cases requires time and resources, causing **operational disruptions [7].** Fraudsters are motivated by financial gain, viewing credit card fraud as an easy way to steal money [6]. The financial industry faces this escalating threat globally, with North America identified as a major epicenter [4].

1.2. Problem statement

A primary challenge in credit card fraud detection is the severe class imbalance in transaction datasets [2][3][8][4]. Fraudulent transactions represent a very small fraction of the total transactions, making them difficult for detection models to accurately identify [2][3][8]. For example, one dataset shows that out of 284,807 transactions, only 492 were fraudulent, meaning the positive class (frauds) accounts for only 0.172% of all transactions [3][2]. This imbalance can bias models towards the majority class (legitimate transactions), reducing their effectiveness in detecting fraud [2][8][4].

Furthermore, the evolving nature of fraud means fraudsters constantly refine their tactics to blend in with legitimate customer behaviour, demanding that detection systems be flexible and quick in adapting to new patterns [4]. Traditional fraud detection methods, such as Rule-Based Systems, Statistical Methods, and earlier Machine Learning (ML) Methods, have significant limitations and shortcomings, including missing new fraud patterns, generating high rates of false positives, being rigid, and requiring continuous manual updates [2][8].

1.3. Objectives

The objectives of the process are primarily focused on training and evaluating neural network models for credit card fraud detection:

- To train neural network models to learn complex, non-linear patterns in credit card transaction data. This involves using activation functions to introduce non-linearity.
- To effectively train the model to make accurate predictions, particularly in detecting fraudulent transactions.

- To address the significant class imbalance issue in the dataset, where fraud cases are a very small percentage. Techniques like SMOTE and Tomek Links are used to create a more balanced dataset to avoid biasing the classifier towards the majority class.
- To evaluate the performance of the trained models using metrics such as accuracy, precision, recall, and F1-score, and visualization techniques like Precision-Recall curves. Area Under the Precision-Recall Curve (AUPRC) is used as a key performance metric.
- To compare the impact of using different activation functions (ReLU, Tanh) and optimizers (SGD, Adam, RMSprop) on model performance.
- To visualize results and model performance through plots of accuracy and loss over epochs, Precision-Recall curves, and metric scores to facilitate comparison between different model configurations**.**

1.4. Scope

This project is focused on using the "creditcardfraud" dataset where the target variable 'Class' distinguishes between genuine and fraudulent transactions. Handling the imbalance in such datasets is noted as crucial for credit card fraud detection.

Stating that the project uses neural networks to tackle this problem. A significant portion of the project, as described in the sources, is dedicated to "Building Neural Network Models". TensorFlow and Keras are used to construct and train sequential neural networks, experimenting with different configurations.

1.5. Organisation of the Report

- Introduction-Provides an overview of the background, outlines the problem statement, defines the objectives, and establishes the scope of the project.
- Literature Review-Analyzes current methods and models utilized for credit card fraud detection, contrasting them with the approach taken in this project.
- System Design and Methodology-Elaborates on the dataset employed, the exploratory data analysis conducted, the preprocessing steps undertaken, the creation of feature and target sets, the model architecture, the algorithms applied, and the strategy for data splitting.
- Implementation Details- Outlines the programming languages, frameworks, code modules, and the comprehensive functioning of the system.
- Results and Discussion - Provides the evaluation metrics, the outcomes of the experiments, and a discussion of the findings in relation to the employed methodology.

- Conclusion and Future Work - Recaps the project's results, acknowledges its limitations, and suggests possible future enhancements.
- References - Compiles all the sources referenced in the report.
- Appendices - Includes supplementary materials such as code snippets and screenshots.

Chapter 2 – Literature Review

2.1. Existing methods and models

2.1.1 Encoder–decoder graph neural network for credit card fraud detection

This research explores using **Graph Neural Networks (GNNs)** for credit card fraud detection. The methodology involves detailed data exploration, creating a graph representation with customers and merchants as nodes to capture relationships, and implementing an **encoder-decoder GNN model**. GNNs are designed to process graph-structured data and can capture both local and global information. The model was evaluated using metrics suitable for imbalanced data, achieving **high performance** with 0.82 precision, 0.92 recall, 0.86 F1-score, and 0.92 AUC-ROC. Hyperparameter tuning was crucial for optimising performance. Future work aims to scale the model for large graphs.[9]

2.1.2 Enhancing Credit Card Fraud Detection Using Hybrid Machine Learning Models with Federated Learning Prospects

This paper addresses the significant challenge of credit card fraud in financial systems. It proposes enhancing fraud detection through the use of **hybrid machine learning models**, also considering the potential for **federated learning**. The proposed methodology begins with critical **data preprocessing**, including cleaning to remove missing or redundant information. Robust datasets are highlighted as fundamental for effective training and testing. The research underscores the benefits of using hybrid models to overcome the limitations of individual methods, which is particularly valuable in fraud detection where accuracy and reliability are paramount. Future investigations include federated learning for data privacy.[10]

2.1.3 Credit Card Fraud Detection Using Improved Deep Learning Models

This study focuses on improving credit card fraud detection by integrating **deep learning (DL) models with hyperparameter tuning**. It highlights that the effectiveness of DL models depends heavily on selecting appropriate hyperparameters. The research proposes and evaluates three DL models: AutoEncoder (AE), Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM). Hyperparameter tuning techniques like random search and Bayesian optimization were used. The impact of resampling methods like SMOTE on **class imbalance** was also examined. The proposed models, particularly LSTM, showed superior performance over previous works.[11]

2.1.4 Credit Card Fraud Detection using Classification Credit Card Fraud Detection Using Classification Algorithm

This paper investigates the use of data mining algorithms for creating effective credit card fraud detection systems. It identifies several key challenges in the field, including the need for robust **feature engineering**, effectively handling **imbalanced datasets**, selecting appropriate evaluation metrics, and choosing suitable programme models. The dataset analysed demonstrates a clear imbalance, with significantly fewer fraudulent transactions compared to legitimate ones. The study concludes that **feature building and model tuning** are critical areas requiring concentrated effort for improving current systems. Machine learning-based systems have the potential to improve over time as new data is incorporated.[12]

2.1.5 Credit Card Fraud Detection using Machine Learning Methods

This paper introduces a novel real-time, live approach for credit card fraud detection. The core of the method involves using a **supervised machine learning technique** to identify fraudulent activity. A significant step includes **clustering cardholders** based on their transaction patterns. The research notes that the dataset used exhibited a pronounced **class imbalance**, where fraudulent transactions were considerably less frequent than legitimate ones. To enhance the model's capability in detecting fraud despite this imbalance, the **Synthetic Minority Over-sampling Technique (SMOTE)** was employed to balance the dataset.[13]

2.1.6 Credit card fraud detection using CNN and LSTM

This paper explores the feasibility of detecting credit card fraud using an **ensemble model** that combines the capabilities of **Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNNs)**. It highlights credit card fraud as a continuously evolving issue, with fraudsters constantly developing new techniques. The research posits that detecting suspicious behaviour is essential for preventing fraudulent transactions. The proposed ensemble strategy is designed to enhance fraud detection accuracy by leveraging the respective strengths of CNN and LSTM in processing sequential data. The performance of the model is evaluated using standard metrics, including accuracy, precision, recall, and F1-score.[14]

2.1.7 Feature Selection and Optimization of Classifiers for Detection of Credit Card Frauds

This study focuses on **Feature Selection and Optimization of Classifiers** for credit card fraud detection. It points out challenges such as class imbalance and difficulties some traditional ML methods have with classification. The paper proposes a hybrid approach combining ML techniques with the **Firefly algorithm for feature selection**. It evaluated four classifiers (Neural Network, KNN, SVM, Decision Tree) on a highly imbalanced European dataset. Results showed that Neural Network, Decision Tree, and KNN achieved good accuracy on this dataset compared to SVM.[15]

2.1.8 AI-Powered Credit Card Fraud Detection: A Comparative Analysis of Machine Learning and Deep Learning Techniques

This research provides a comparative analysis of **Artificial Intelligence (AI) techniques**, specifically **Machine Learning (ML) and Deep Learning (DL)**, applied to credit card fraud detection. The primary objective is to compare the performance of various AI-based models using key metrics like accuracy, precision, recall, and F1-score. The study also investigates how effectively these models address the challenge of **class imbalance** commonly found in fraud detection datasets. It aims to provide practical insights, suggesting that the selection of an optimal model should balance accuracy, computational efficiency, and real-time processing capabilities.[16]

2.1.9 Analysis of Different Machine Learning Models for Credit Card Fraud Detection

This paper presents a **comparative analysis** of multiple machine learning models for detecting credit card fraud. It acknowledges the significant increase in fraud alongside the rise in online transactions and identifies the **highly unbalanced nature of fraud datasets** as a major challenge. The study specifically evaluates four models: **Logistic Regression, Isolation Forest, K-means clustering, and Convolutional Neural Network (CNN)**. The objective is to understand their principles and assess their performance in distinguishing genuine from fraudulent transactions. Evaluation is conducted using various metrics, including accuracy, precision, recall, F1 scores, and AUC-ROC curves.[17]

2.1.10 Fraud Detection Using Machine Learning and Deep Learning

This article offers a comprehensive overview of **Machine Learning (ML) and Deep Learning (DL) techniques** for fraud detection, presenting them as promising alternatives to easily circumvented traditional methods. The review categorises approaches into supervised, unsupervised, semi-supervised, and deep learning. It highlights significant challenges in the field, including dealing with **imbalanced datasets**, defending against adversarial attacks, ensuring model interpretability, and fostering industry-academia collaboration. The paper underscores the importance of **feature engineering** and data pre-processing for enhancing detection systems. A case study illustrates ML/DL effectiveness in financial fraud detection.[18]

Source	Model/Technique	Key Results/Metrics	Strengths	Weaknesses	Gaps/Limitations	Future Directions	Dataset
Journal of King Saud University	Encoder-decoder GNN	Precision: 0.82, Recall: 0.92, F1-score: 0.86, AUC-ROC: 0.92	Captures complex relationships, effective fraud detection	None explicitly mentioned	Evolving fraud, data imbalance, adversarial attacks	Investigate deep learning, GNNs	Improved Sparkov dataset
Enhancing Credit Card Fraud Detection Using Hybrid Machine Learning Models	Hybrid ML Models	Decision Tree: Accuracy 99.93%, Precision 82.72%, Recall 74.44%, F1-Score 78.36%, MCC 78.44	Balances interpretability & performance	None explicitly mentioned	Need to balance interpretability & performance	Enhance with Hybrid ML, explore Federated Learning	Open-source dataset from ULB
Credit Card Fraud Detection Using Improved Deep Learning Models	Improved Deep Learning Models	LSTM: Accuracy 99.2%, DR 93.3%, AUC 96.3%	Surpasses previous work, trade-off between detection & precision	Need to enhance detection rate	Imbalance, evolving fraud, lack of interpretability	Resampling, Bayesian optimization, Binary Focal Loss	European credit card dataset
Credit Card Fraud Detection using Classification Algorithms	Classification Algorithms	All data mining algorithms outperformed current system	All data mining algorithms outperformed current system	Feature building and model tuning	Feature engineering, non-uniform data	Practical fraud detection systems using data mining	Historical credit card transactions
Credit Card Fraud Detection using Machine Learning Methods	Logistic Regression	Not provided	Simple, interpretable, effective	Data imbalance	Data imbalance, lack of transparency in complex models	Develop interpretable models, address imbalance, feature engineering	Kaggle Credit Card Fraud Detection dataset
Credit card fraud detection using CNN and LSTM	Ensemble CNN-LSTM	Early fusion performed better than late fusion	Combines CNN & LSTM strengths	Can flag valid transactions or omit fraudulent ones	Evolving problem	Investigate fraud detection using CNN-LSTM	European cardholder dataset
Feature Selection and Optimization of Machine Learning Classifiers	NN, k-NN, SVM, Decision Tree	NN, Decision Tree, k-NN showed high accuracy	High accuracy for imbalanced datasets	SVM: low accuracy for imbalanced datasets	SVM accuracy	Deep learning for real-time detection, feature selection	Kaggle repository dataset
Comparative Study of AI-based Techniques	LSTM, Decision Tree, Random Forest	LSTM: Accuracy 98.3%	DL models outperform traditional models for imbalanced data	Lower recall for Decision Tree	Class imbalance, need for model evaluation & adaptation	Continuous model evaluation, address imbalance	Kaggle Credit Card Fraud Detection Dataset
Analysis of Different Machine Learning Models	Logistic Regression, Isolation Forest, K-Means, CNN	Evaluated using accuracy, F1 score, recall, AUC-ROC			Unbalanced dataset, lack of transparency	Address imbalance, develop interpretable models	Kaggle dataset
Systematic Review of ML and DL Strategies	Review of ML/DL					Algorithm interpretability, imbalanced datasets	

2.2Comparison with your approach

Aspect	**Our Model**	**Literature Survey**
Dataset & Characteristics	mlg-ulb/creditcardfraud (European, Sep 2013, 2 days, ~284k trans., 492 fraud (0.172% imbalance), PCA features V1-V28, Time, Amount, Class (0/1))	Mentions highly imbalanced European datasets are common; robust datasets crucial.
Data Preprocessing	MinMaxScaler (0-1 normalization for faster NN convergence); class distribution visualization.	Highlights importance of cleaning, robust feature engineering/preprocessing; one study used transaction pattern-based clustering.
Handling Imbalance	SMOTE + Tomek Links (over-sampling minority, under-sampling majority).	Identifies imbalance as major challenge; SMOTE specifically mentioned and used in several studies.
Model Types & Architecture	Dense Neural Networks (Keras Sequential); 2 hidden layers (128, 64 neurons), sigmoid output.	Explores wider range: DL (AE, CNN, LSTM, GNNs), hybrid/ensemble (CNN+LSTM), traditional ML (KNN, SVM, DT, LR, Isolation Forest, K-means).
Activation & Optimizers	Hidden: ReLU, Tanh; Output: Sigmoid; Optimizers: SGD, Adam, RMSprop.	Stresses hyperparameter tuning, but specific activation/optimizer exploration not detailed in excerpts.
Evaluation Metrics	Accuracy, precision, recall, F1-score, Precision-Recall Curves, AUPRC (emphasized for imbalance).	Accuracy, precision, recall, F1-score, AUC-ROC (mentions 0.92), AUPRC (higher indicates better). Metrics for imbalanced data crucial.
Hyperparameter Tuning	Simple testing of fixed activation/optimizer combinations (10 epochs, validation split).	Explicitly crucial for DL; mentions random search and Bayesian optimization as more systematic techniques. Model tuning critical.
Scope & Focus	Hands-on implementation example: data loading, preprocessing, imbalance	Broader research overview: more challenges (adversarial attacks, interpretability), diverse/complex

handling, specific DNN models, evaluation/visualization.	techniques, future directions (federated learning).

Fig 2.2.2 Comparison of our model with the existing model

Chapter 3 – System Design and Methodology

3.1. Dataset Description

- The project utilises the "creditcardfraud" dataset.
- This dataset was downloaded from Kaggle, specifically from the user "mlg-ulb".
- It contains credit card transactions from September 2013 by European cardholders.
- The transactions cover a period of two days.
- The dataset includes approximately 284,807 total transactions.
- Out of these, only 492 are fraudulent transactions.
- The dataset typically comes in CSV format and has about 284,807 rows and 31 columns.
- The features include the results of a PCA transformation (V1 to V28), along with additional columns like Time and Amount.
- The target variable is named 'Class', where 0 indicates a genuine transaction and 1 indicates fraud.
- A critical characteristic of the dataset is that it is highly imbalanced, with fraud cases making up only 0.172% of all transactions.

3.2. Exploratory Data Analysis (EDA)

- The primary EDA step mentioned is visualizing the class distribution.
- This is done using Seaborn's countplot to create a bar chart showing the count of samples belonging to each class.
- Visualizing the class distribution is highlighted as critical due to the imbalanced nature of the dataset, which can bias learning algorithms.

3.3. Preprocessing Steps

- Loading the Dataset: The project starts by downloading the dataset using kagglehub.dataset_download() and reading the "creditcard.csv" file into a pandas DataFrame.
- Data Normalization: The data is preprocessed by normalizing the features using MinMaxScaler from scikit-learn.

    - Normalization scales features so that each one has the same range, typically between 0 and 1.
    - This is important because features like 'Amount' have different scales compared to the PCA components (V1-V28).
    - The fit() method calculates minimum and maximum values per column, and transform() applies the scaling formula.
    - Normalization helps in faster convergence during neural network training and prevents features with larger values from dominating.
- Handling Imbalanced Data: To address the significant class imbalance, a pipeline combining SMOTE and Tomek Links is used.
    - SMOTE (Synthetic Minority Over-sampling Technique) is used to over-sample the minority class (fraudulent transactions) by creating synthetic samples through interpolation. random_state=42 ensures reproducibility of synthetic sample generation.
    - Tomek Links is an under-sampling technique that removes majority class samples that are too close to minority examples, which helps in cleaning the decision boundary. sampling_strategy='majority' ensures only majority class samples in Tomek links are removed.
    - The imblearn Pipeline chains these two techniques to apply them sequentially.
    - The fit_resample() method of the pipeline is applied to the features and target to create more balanced resampled data (X_resampled, y_resampled).

3.4. Model Architecture

- The project uses sequential neural network models built with TensorFlow and Keras.
- The model architecture is defined within a function create_model.
- The model consists of dense (fully connected) layers stacked linearly.
- The first layer has 128 neurons. It requires specifying the input_shape, which is the number of features in the resampled data.
- There is a second dense layer with 64 neurons.
- The output layer has a single neuron.
- The activation function for the hidden layers (128 and 64 neurons) is a parameter of the create_model function, allowing experimentation with different types.
- The activation function for the output layer is 'sigmoid', which outputs a value between 0 and 1, interpretable as the probability of belonging to the positive class (fraud) in this binary classification problem.

- Summaries of the model architecture, including the number of layers and parameters, are printed using model.summary().

3.5. Description of the Algorithms (Activation Functions and Optimizers)

The project experiments with different activation functions and optimizers during model training.

- Activation Functions:
    - Activation functions are crucial as they introduce non-linearity into neural networks, allowing them to learn complex patterns by transforming neuron outputs. Without them, the network could only learn linear relationships.
    - They decide whether a neuron should "fire" and determine its contribution to the next layer.
    - The project experiments with two specific activation functions for the hidden layers:
        - ReLU (Rectified Linear Unit): This function outputs the input directly if it's positive, and zero otherwise (ReLU(x)=max(0,x)). It often tends to converge faster and reduces the risk of vanishing gradients.
        - Tanh (Hyperbolic Tangent): This function outputs values between -1 and 1 (tanh(x)= e^ x-e^ −x/ e^ x +e^ −x). It is zero-centered, which can sometimes be beneficial, especially if data is normalized around zero.
- Optimizers:
    - Optimizers are algorithms that guide the network's learning process by updating weights and biases to minimise the loss (difference between predictions and actual values).
    - Without optimizers, the network wouldn't have a systematic way to learn effectively.
    - The project experiments with three specific optimizers:
        - SGD (Stochastic Gradient Descent): A basic algorithm that updates weights based on the gradient calculated on a single data point or a small batch. It can be slow to converge but can potentially escape local minima.
        - Adam (Adaptive Moment Estimation): An adaptive learning rate algorithm that combines benefits from AdaGrad and RMSProp. It's generally efficient and widely used.
        - RMSprop (Root Mean Square Propagation): Another adaptive learning rate algorithm that adapts learning rates for each parameter.

These components (activation functions and optimizers) are configured when the model is compiled using model.compile(), along with the loss function (binary_crossentropy for this binary classification problem) and metrics (accuracy).

Chapter 4 – Implementation Details

4.1. Programming Languages, Frameworks

The project implementation relies heavily on the Python programming language. Several key libraries and frameworks are used to perform different tasks:

- kagglehub: Used for directly downloading the dataset from Kaggle.
- pandas: A library for data manipulation and analysis, used for reading the CSV file into a DataFrame, handling columns, and displaying results in a tabular format.
- scikit-learn (sklearn): A library providing tools for machine learning. Used for data preprocessing (MinMaxScaler), splitting data into training and testing sets (train_test_split), and evaluating model performance (classification_report, precision_recall_curve, auc).
- seaborn: A library for statistical data visualization, used alongside Matplotlib for creating plots like the class distribution bar chart and performance metric visualizations.
- matplotlib.pyplot: A plotting library used for setting titles, labels, legends, and displaying plots created by Seaborn or directly.
- imbalanced-learn (imblearn): A library specifically designed to address class imbalance. Used for implementing the SMOTE and Tomek Links pipeline.
- TensorFlow and Keras: Powerful open-source libraries for building and training neural networks. Keras, now integrated into TensorFlow, provides a high-level API for defining models. Keras components like Sequential, Dense layers, and optimizers (SGD, Adam, RMSprop) are explicitly used.

4.2. Code Modules Description

1. Loading the Dataset: This module handles downloading the "creditcardfraud" dataset from Kaggle using kagglehub and loading the "creditcard.csv" file into a pandas DataFrame. It also includes printing a sample of the loaded data.
2. Data Preprocessing: Normalization: This module focuses on scaling the numerical features. It uses MinMaxScaler from scikit-learn to scale features to a range between 0 and 1. It also converts the normalized NumPy array back into a pandas DataFrame and prints the class distribution of the normalized data.
3. Visualizing the Class Distribution: This module uses Seaborn's countplot and Matplotlib to create and display a bar chart showing the count of samples in each class, specifically highlighting the dataset's imbalanced nature.
4. Handling Imbalanced Data: This module addresses the class imbalance problem. It imports SMOTE, TomekLinks, and Pipeline from imblearn. It sets up a pipeline that

first applies SMOTE for over-sampling the minority class and then Tomek Links for under-sampling majority class samples that are close to minority ones.

5. Creating Feature and Target Sets: This module separates the features (all columns except 'Class') and the target variable ('Class') from the normalized DataFrame. Crucially, it then applies the imblearn pipeline's fit_resample() method to the features and target to produce X_resampled and y_resampled, creating a more balanced dataset for training.
6. Building Neural Network Models: This module defines a function create_model using TensorFlow and Keras to build sequential neural networks. The architecture includes two dense hidden layers (128 and 64 neurons) and a single-neuron output layer with a 'sigmoid' activation function. It allows specifying the activation function for the hidden layers ('relu' or 'tanh'). It also prints a summary of the created model architectures.
7. Splitting Data for Training and Testing: This module uses train_test_split from scikit-learn to divide the resampled data (X_resampled, y_resampled) into training and testing sets (X_train, X_test, y_train, y_test). A test size of 20% is used, and random_state=42 ensures reproducibility.
8. Training Models with Different Hyperparameters: This is a core module that iterates through different combinations of activation functions ('relu', 'tanh') and optimizers ('SGD', 'Adam', 'RMSprop'). For each combination, it creates a model using the create_model function, compiles it with the specified optimizer and binary_crossentropy loss. It then trains the model on the training data for 10 epochs with a batch size of 32, using 20% of the training data for validation to monitor performance and detect potential overfitting. Training progress (loss and accuracy) is recorded in a history object. Predictions are made on the test set.
9. (Evaluation within Training Loop): Within Step 8, after training, the module performs initial evaluation. It converts probability predictions to binary class labels using a 0.5 threshold. A classification_report is generated to compute metrics like accuracy, recall, and F1-score for the minority class. These results are stored in a list. Plots of training and validation accuracy and loss over epochs are also generated and shown for each model combination.
10. Evaluating with Precision-Recall Curves: This module calculates precision and recall values across different probability thresholds using precision_recall_curve and computes the Area Under the Precision-Recall Curve (AUPRC) using auc for each model configuration. This is crucial for evaluating performance on imbalanced datasets. The results, including precision, recall, and AUPRC values, are stored.
11. Plotting Precision-Recall Curves: This module uses Matplotlib to plot all the computed precision-recall curves on a single chart, allowing direct visual comparison of model performance based on AUPRC.

12. Displaying Results in a Pandas DataFrame: This module converts the list of performance metrics collected during the training loop (Step 8/9) into a pandas DataFrame for easier viewing and analysis in a tabular format.

13. Visualizing Results with Line and Bar Plots: This module uses pandas, Matplotlib, and Seaborn to create additional visualisations of the collected performance metrics. It creates a combined column for optimizer and activation and melts the DataFrame to enable plotting metrics (accuracy, precision, recall, f1-score) against the different model configurations using both line plots and bar plots.

4.3. System Working

The system works as a sequential pipeline, processing the credit card transaction data through several stages to train and evaluate neural network models for fraud detection:

1. Data Acquisition: The process begins by downloading the raw dataset from Kaggle and loading it into memory as a pandas DataFrame.

2. Initial Inspection & Preprocessing: The loaded data is subjected to a crucial preprocessing step: feature normalization. This ensures that features like 'Amount' are scaled consistently with the PCA-transformed features (V1-V28), which is vital for the performance of neural networks.

3. Exploratory Analysis: The normalized data's class distribution is visualised to confirm and highlight the extreme imbalance between genuine and fraudulent transactions, a key challenge the system must address.

4. Handling Imbalance: To mitigate the bias caused by class imbalance, the system applies a sophisticated resampling strategy combining SMOTE and Tomek Links to the features and target variable. SMOTE creates synthetic samples for the minority class, while Tomek Links removes ambiguous majority class samples, resulting in a more balanced dataset (X_resampled, y_resampled) for training.

5. Data Splitting: The resampled data is then split into training and testing sets. The training set is used to teach the models, and the separate testing set is reserved for unbiased evaluation of their performance on unseen data.

6. Model Training & Experimentation: The system proceeds to build and train multiple neural network models. It iterates through different combinations of hidden layer activation functions ('relu', 'tanh') and optimizers ('SGD', 'Adam', 'RMSprop'). For each combination, a model is created with a defined architecture, compiled, and trained on the training data, with validation performed on a portion of the training data to monitor for overfitting.

7. Performance Evaluation: After training each model, the system makes predictions on the test set. These predictions are converted into binary class labels. Performance is evaluated using standard metrics like accuracy, recall, and F1-score for the minority class, typically presented in a classification report. Additionally, Precision-Recall

Curves are generated, and the Area Under the Precision-Recall Curve (AUPRC) is calculated for each model configuration.

8. Result Aggregation & Visualization: The metrics from all experiments are collected and organised, typically in a pandas DataFrame. Finally, the system visualises these results using line and bar plots to compare the performance of the different model configurations based on the various metrics and the AUPRC. This allows for identifying which combination of activation function and optimizer performed best on this specific dataset and task.

Chapter 5 – Results and Discussion

5.1. Evaluation metrics

Accuracy

Accuracy measures the overall correctness of the model's predictions. It's the ratio of correctly classified instances to the total number of instances.[20]

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

Precision

Precision measures the proportion of positive predictions that were actually correct. It answers the question: "Of all the instances the model predicted as positive, how many were truly positive?"[20]

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

Recall

Recall (also known as sensitivity or true positive rate) measures the proportion of actual positive cases that were correctly identified by the model. It answers the question: "Of all the actual positive instances, how many did the model correctly predict as positive?"[20]

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

F1 Score

The F1 score is the harmonic mean of precision and recall. It provides a balanced measure of the model's performance, especially when dealing with imbalanced datasets.[20]

$$F1\ \text{Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

Area Under the Precision-Recall Curve (AUPRC)

AUPRC is the area under the Precision-Recall curve. The Precision-Recall curve plots precision on the y-axis and recall on the x-axis for different threshold values of the classifier.[5]

The AUPRC provides a single scalar value that summarizes the trade-off between precision and recall across various thresholds. A higher AUPRC indicates better performance, especially when the positive class is rare.

The formula for AUPRC is typically calculated using numerical integration techniques, such as the trapezoidal rule, based on the discrete precision and recall values obtained at different thresholds:

$$\text{AUPRC} \approx \sum_{i=1}^{n-1} (Recall_{i+1} - Recall_i) \times \frac{Precision_i + Precision_{i+1}}{2}$$

where n is the number of thresholds considered, and (Recalli,Precisioni) are the recall and precision values at the i-th threshold, with recall values sorted in ascending order.

Alternatively, in some contexts, especially in libraries like scikit-learn, AUPRC might be calculated as the average precision (AP), which is the weighted mean of precisions achieved at each threshold, with the weight being the increase in recall from the previous threshold.

These metrics provide different perspectives on the performance of a classification model, and the choice of which metric is most important depends on the specific problem and the relative costs of false positives and false negatives.[19]

5.2. Results

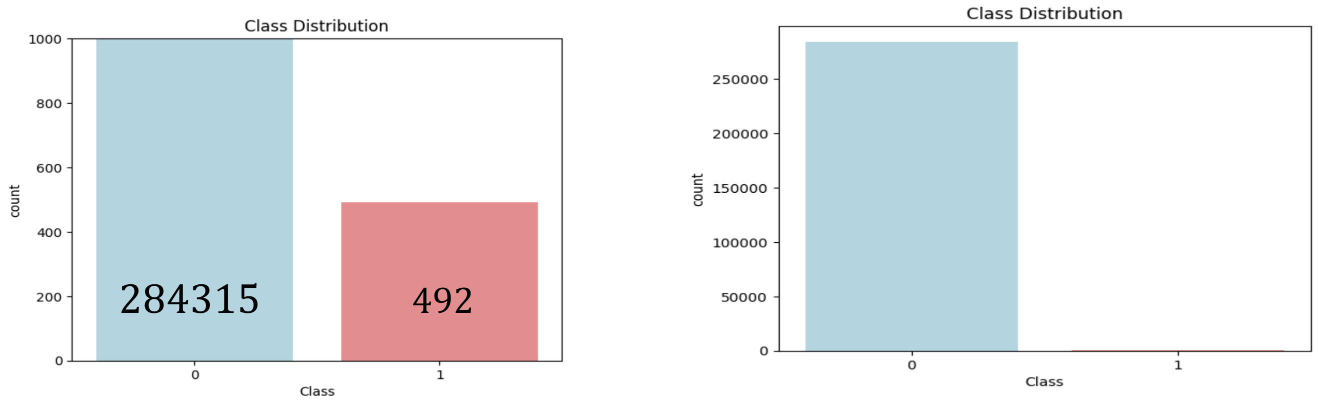


Fig: 5.2.1: Displays a highly imbalanced class distribution, where class 0 has a significantly larger count than class 1 before normalisation

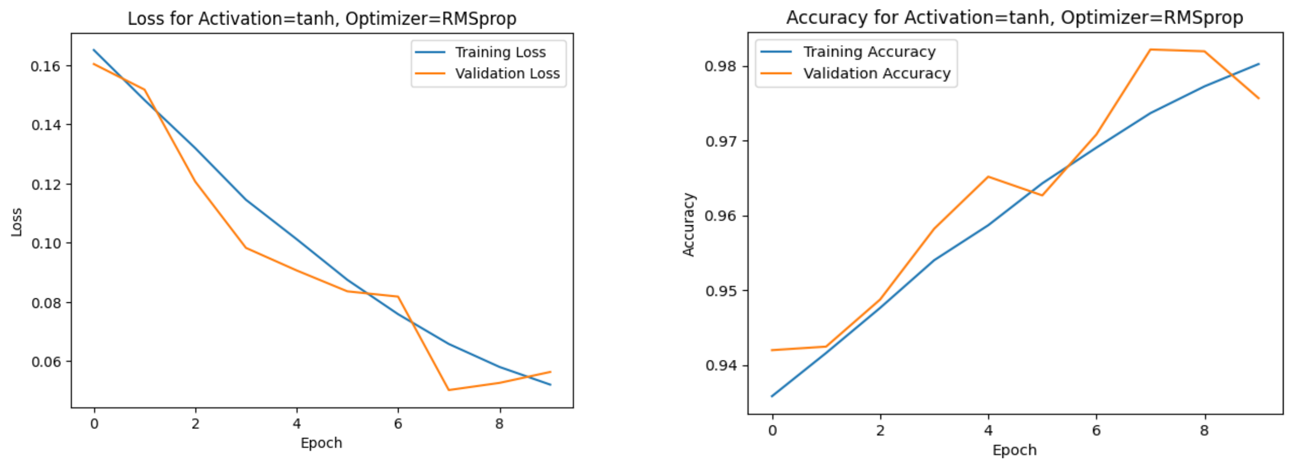


Fig: 5.2.2: The validation loss and accuracy show signs of overfitting, with performance declining after initial improvement.

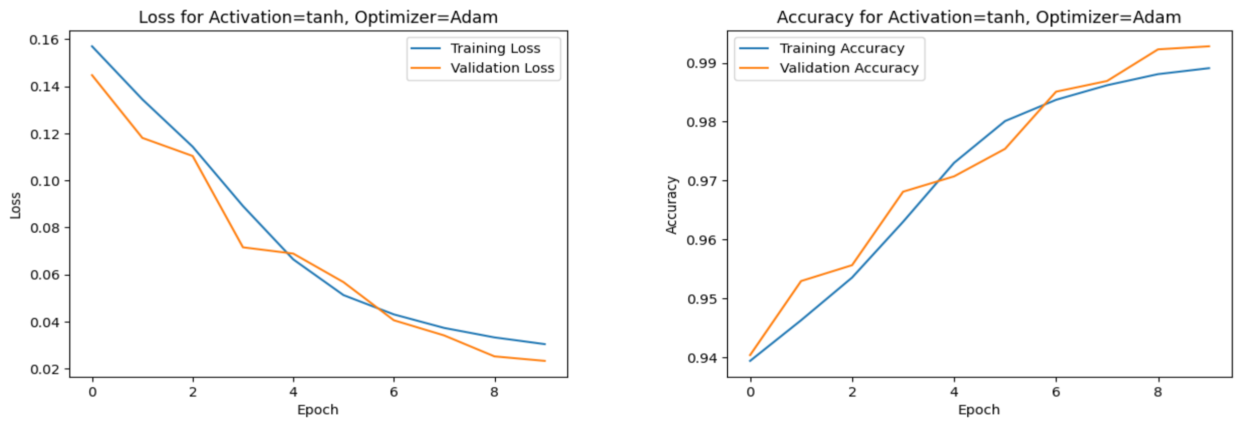


Fig:5.2.3: Model uses the Adam optimizer with tanh activation shows strong generalization, with validation loss staying close to training loss & accuracy steadily improving. This suggests effective learning and minimal overfitting compared to RMSprop.

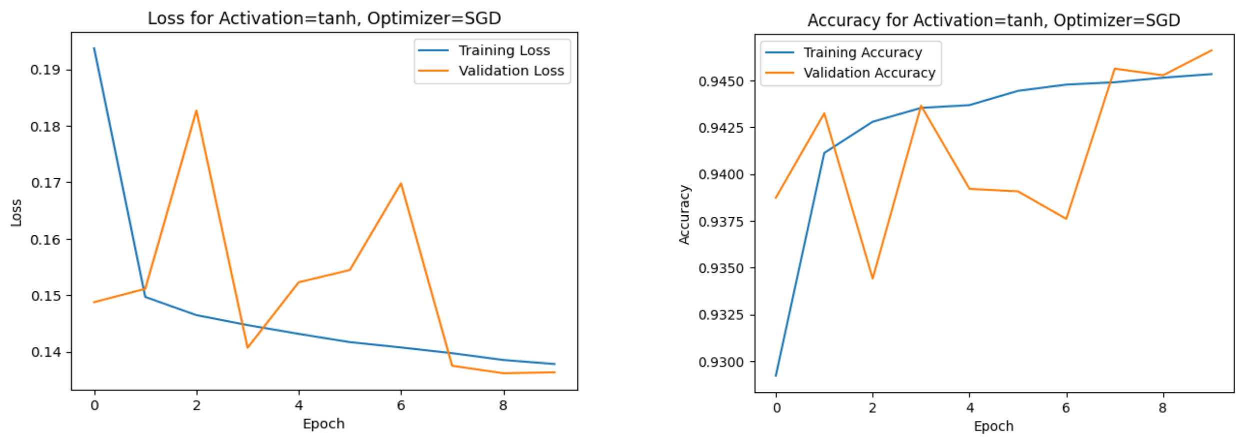


Fig:5.2.4: Model using the SGD optimizer learns inconsistently, with unstable validation loss and accuracy showing sharp fluctuations. This suggests difficulty in finding stable, generalizable patterns compared to Adam and RMSprop**.**

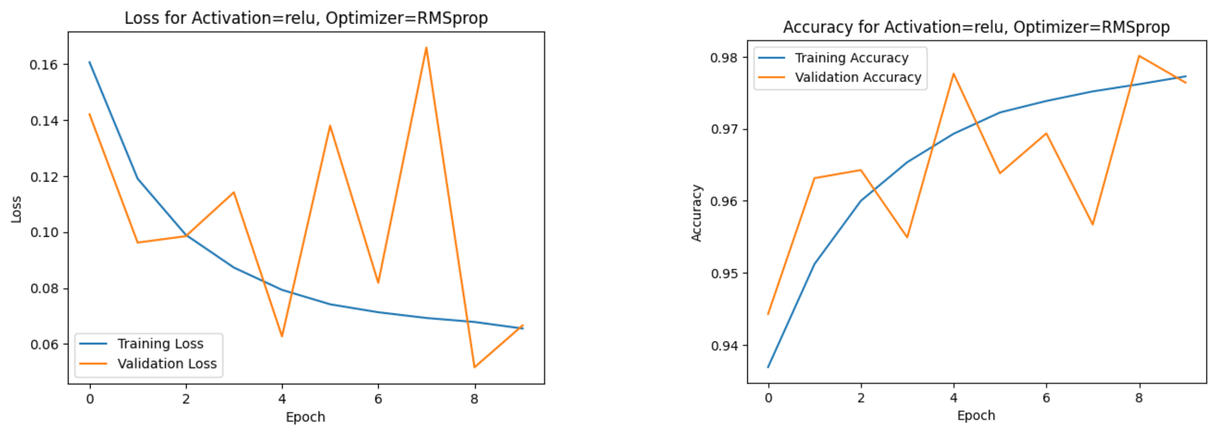


Fig:5.2.5: The model learns but struggles to generalize, with unstable validation loss and accuracy showing erratic fluctuations. This suggests overfitting and inconsistent performance when using RMSprop with ReLU activation.

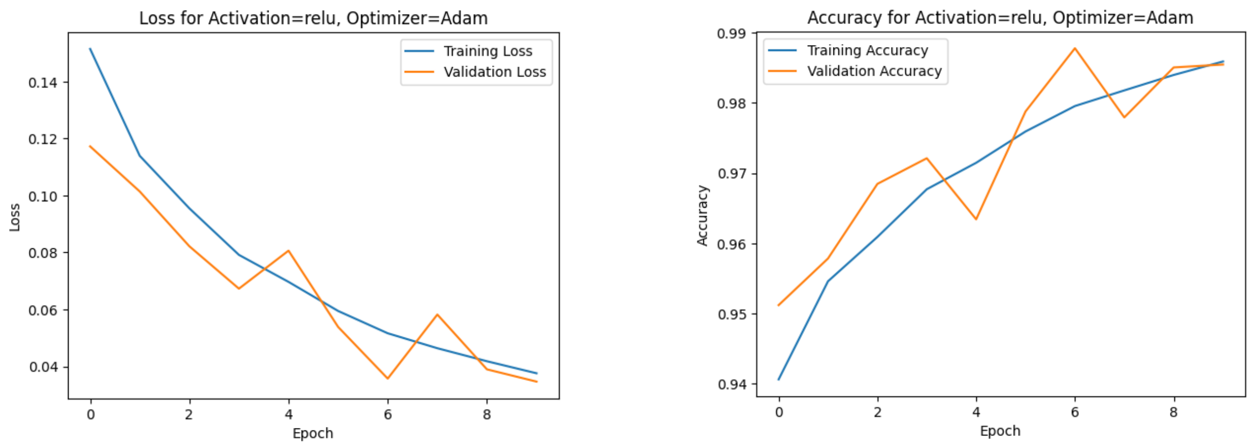


Fig:5.2.6: The model learns steadily with Adam and ReLU activation, showing strong generalization despite minor fluctuations. Some slight overfitting emerges towards the end, but overall performance remains stable.

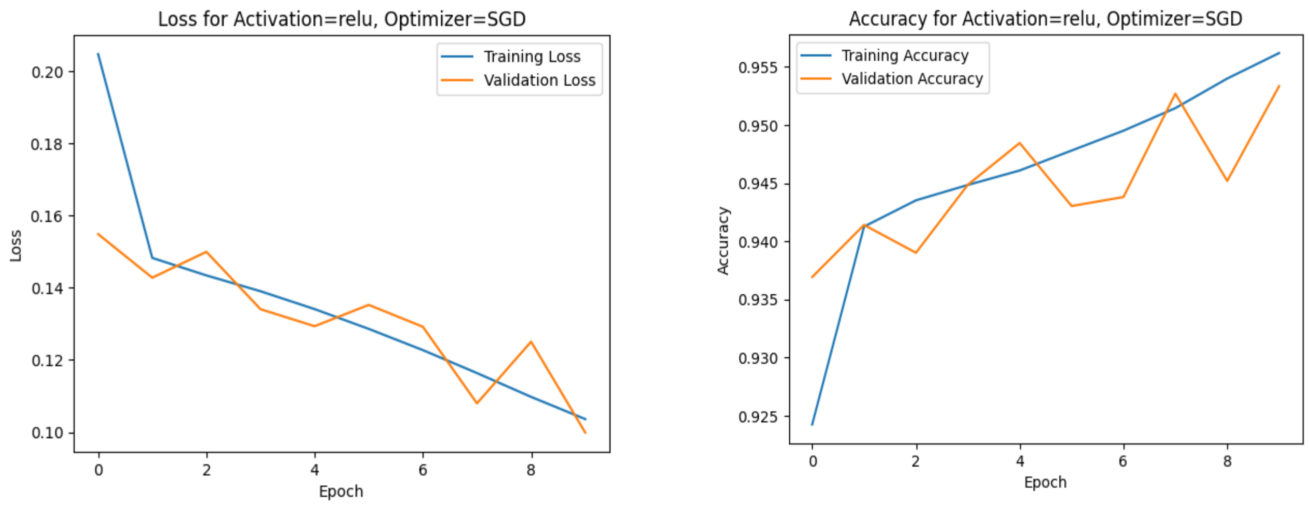


Fig:5.2.7: The model struggles with stability when using SGD and ReLU, showing erratic validation loss and accuracy fluctuations. This suggests difficulties in learning reliable, generalizable patterns from the data.

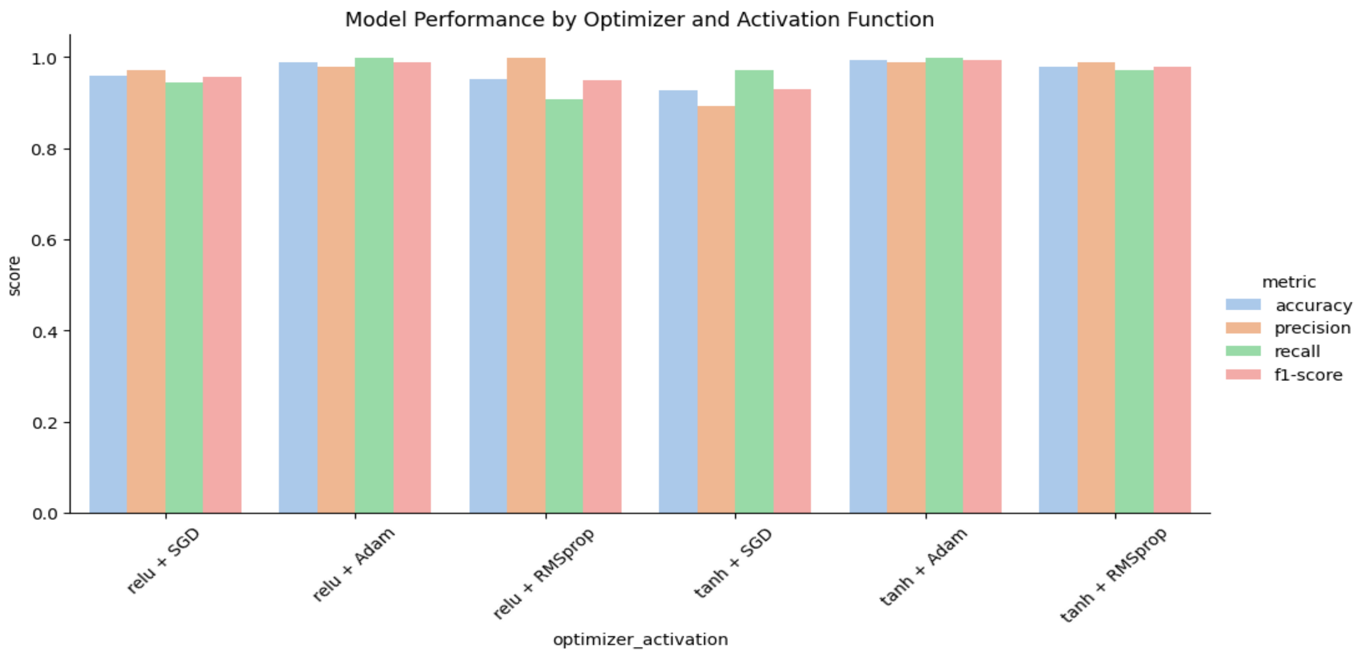


Fig:5.2.8: The bar plot compares model performance across accuracy, precision, recall, and F1-score for different optimizer and activation function combinations. The x-axis represents optimizer-activation pairs (e.g., "relu + SGD"), while the y-axis shows scores for each metric. Each group of bars visualizes a different model configuration, helping compare how well various setups perform. Overall, Adam and RMSprop optimizers tend to outperform SGD across all evaluation metrics.

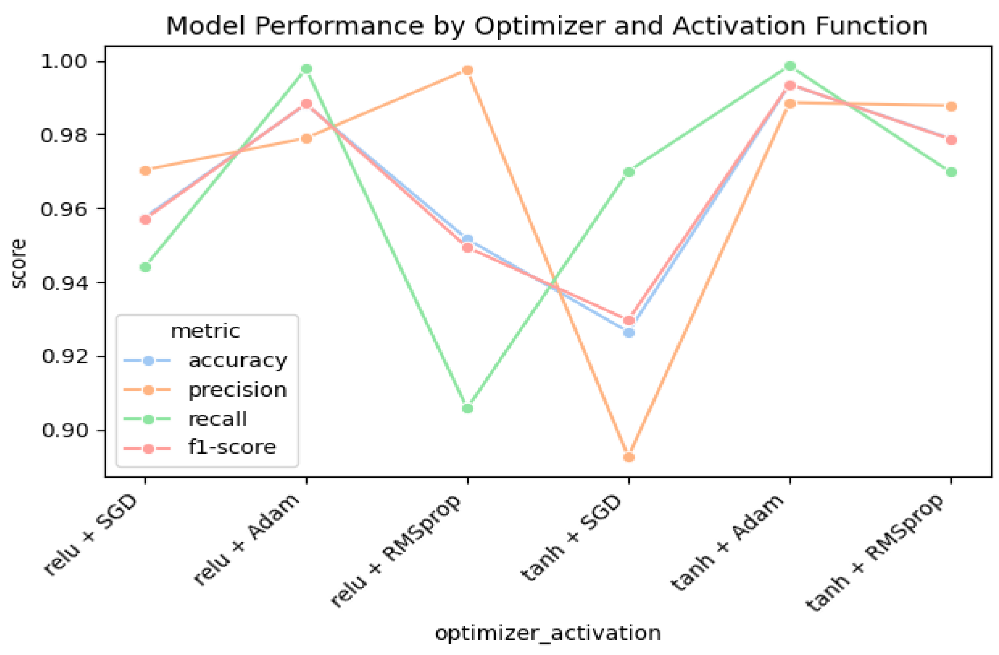


Fig:5.2.9: The line plot compares neural network models based on optimizer-activation combinations, with metrics like accuracy, precision, recall, and F1-score represented by colored lines. The x-axis lists the configurations (e.g., "relu + SGD"), while the y-axis shows performance scores. The visualization highlights trends, revealing that Adam and RMSprop generally achieve higher and more stable results than SGD.

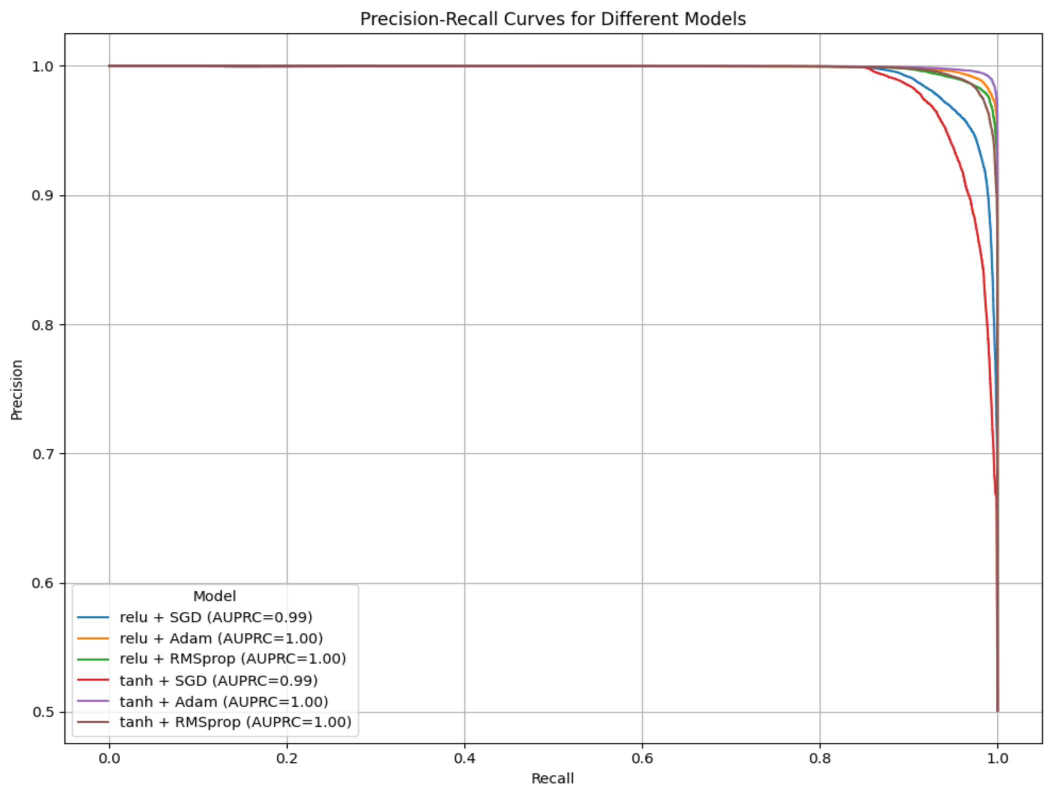
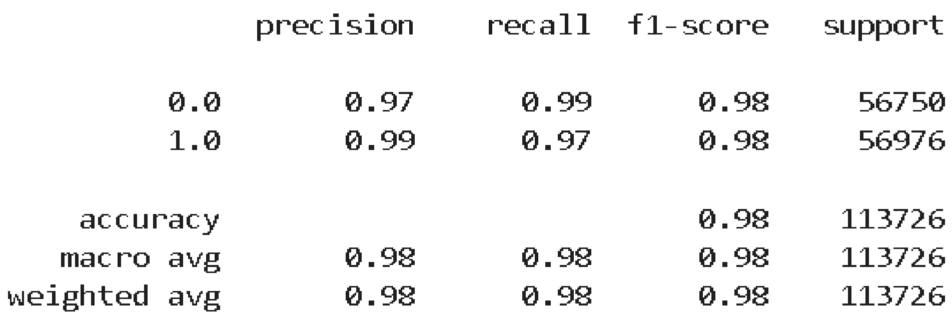
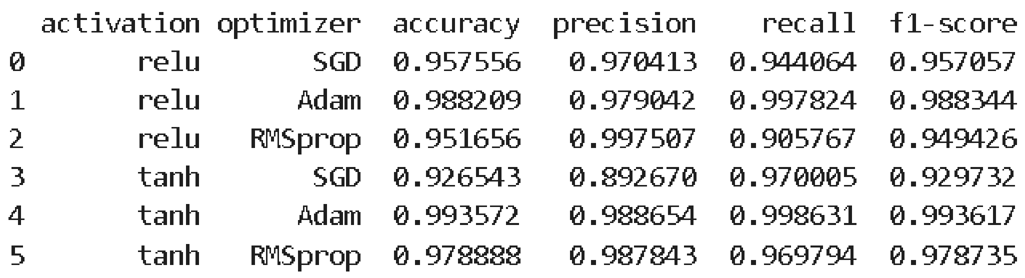


Fig:5.2.10: The PR curves compare models using activation functions and optimizers, showing the trade-off between precision and recall. Higher curves and AUPRC values indicate strong performance, with most models achieving near-perfect scores close to 1.00.



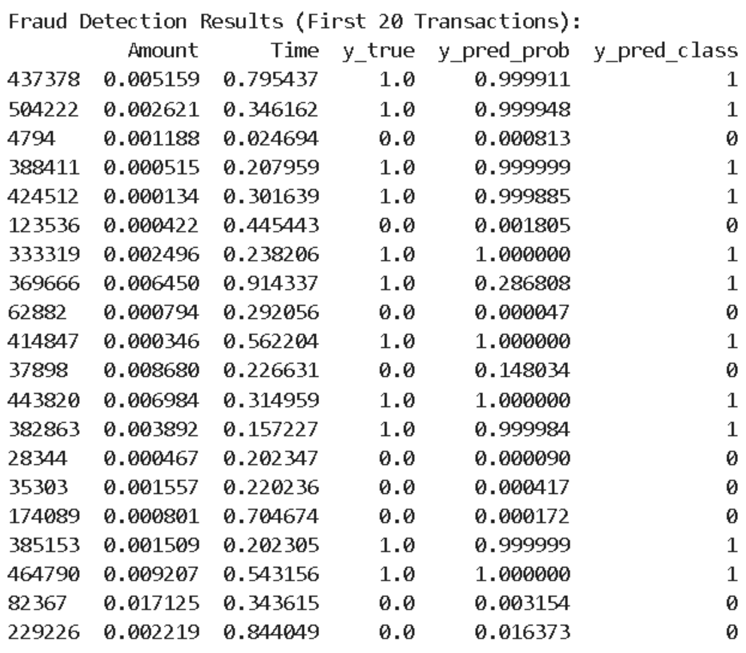
```
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98     56750
         1.0       0.99      0.97      0.98     56976

    accuracy                           0.98    113726
   macro avg       0.98      0.98      0.98    113726
weighted avg       0.98      0.98      0.98    113726
```

Fig:5.2.11: The classification report highlights strong model performance, with an overall accuracy of 0.98. Precision, recall, and F1-scores are consistently high across both classes, indicating balanced and effective classification

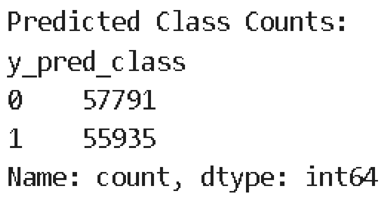


```
  activation optimizer  accuracy  precision    recall  f1-score
0       relu       SGD  0.957556   0.970413  0.944064  0.957057
1       relu      Adam  0.988209   0.979042  0.997824  0.988344
2       relu   RMSprop  0.951656   0.997507  0.905767  0.949426
3       tanh       SGD  0.926543   0.892670  0.970005  0.929732
4       tanh      Adam  0.993572   0.988654  0.998631  0.993617
5       tanh   RMSprop  0.978888   0.987843  0.969794  0.978735
```

Fig:5.2.12: The table compares neural network models using different activation functions and optimizers, showcasing accuracy, precision, recall, and F1-score. Models with the Adam optimizer consistently perform best across all metrics, indicating strong and balanced learning.



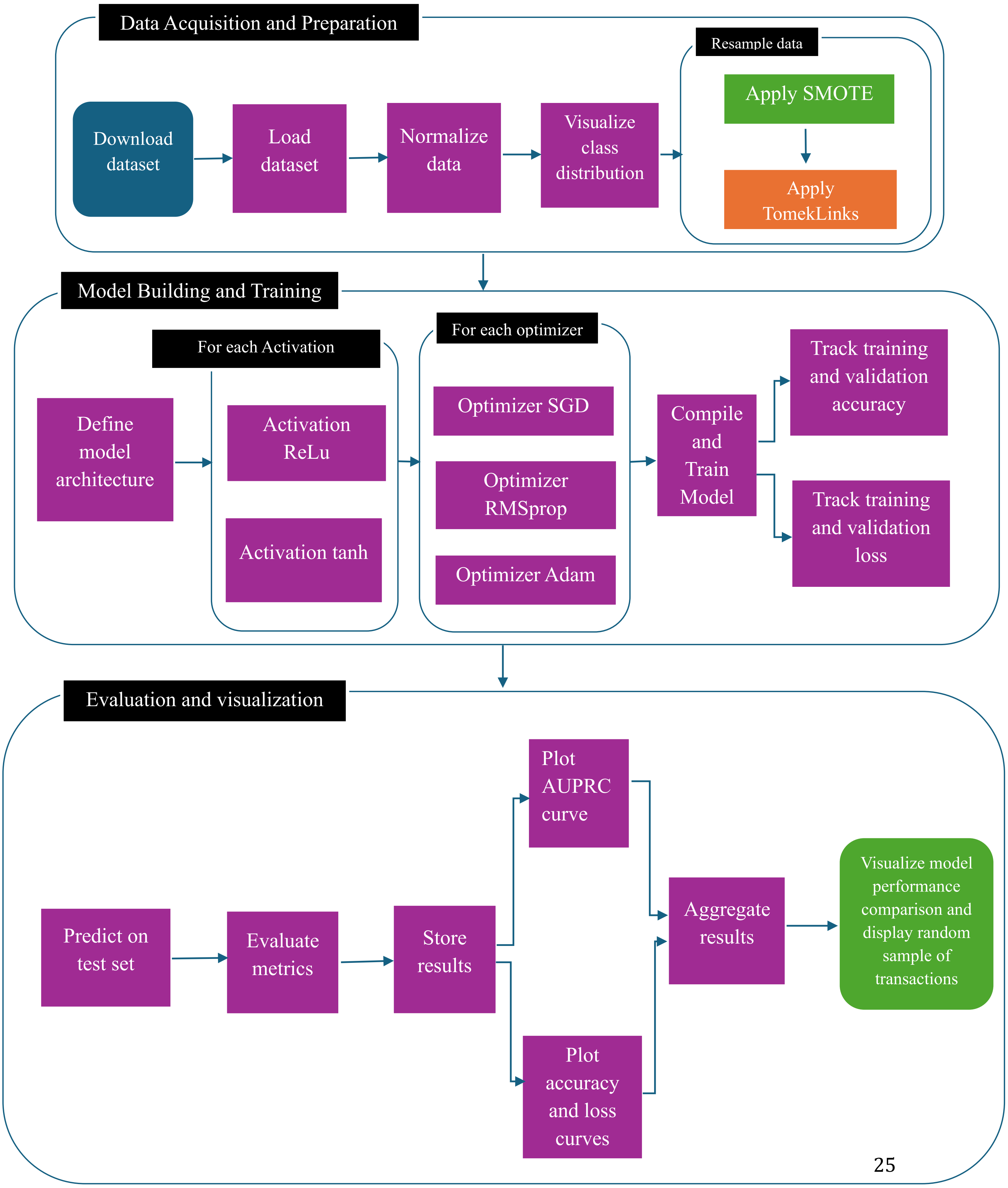
```
Fraud Detection Results (First 20 Transactions):
          Amount      Time  y_true  y_pred_prob  y_pred_class
437378  0.005159  0.795437     1.0     0.999911             1
504222  0.002621  0.346162     1.0     0.999948             1
4794    0.001188  0.024694     0.0     0.000813             0
388411  0.000515  0.207959     1.0     0.999999             1
424512  0.000134  0.301639     1.0     0.999885             1
123536  0.000422  0.445443     0.0     0.001805             0
333319  0.002496  0.238206     1.0     1.000000             1
369666  0.006450  0.914337     1.0     0.286808             1
62882   0.000794  0.292056     0.0     0.000047             0
414847  0.000346  0.562204     1.0     1.000000             1
37898   0.008680  0.226631     0.0     0.148034             0
443820  0.006984  0.314959     1.0     1.000000             1
382863  0.003892  0.157227     1.0     0.999984             1
28344   0.000467  0.202347     0.0     0.000090             0
35303   0.001557  0.220236     0.0     0.000417             0
174089  0.000801  0.704674     0.0     0.000172             0
385153  0.001509  0.202305     1.0     0.999999             1
464790  0.009207  0.543156     1.0     1.000000             1
82367   0.017125  0.343615     0.0     0.003154             0
229226  0.002219  0.844049     0.0     0.016373             0
```



```
Predicted Class Counts:
y_pred_class
0    57791
1    55935
Name: count, dtype: int64
```

Fig:5.2.13: The table presents fraud detection results for the first 20 transactions after oversampling, showing values exceeding the original dataset size. It includes transaction details, true labels, predicted fraud probabilities, and classifications, highlighting both correct and incorrect predictions. The model classified 57,791 transactions as non-fraudulent and 55,935 as fraudulent, demonstrating a balanced prediction output after addressing class imbalance.

Flowchart



Data Acquisition and Preparation
Download dataset
Load dataset
Normalize data
Visualize class distribution
Resample data
Apply SMOTE
Apply TomekLinks
Model Building and Training
Define model architecture
For each Activation
Activation ReLu
Activation tanh
For each optimizer
Optimizer SGD
Optimizer RMSprop
Optimizer Adam
Compile and Train Model
Track training and validation accuracy
Track training and validation loss
Evaluation and visualization
Predict on test set
Evaluate metrics
Store results
Plot AUPRC curve
Plot accuracy and loss curves
Aggregate results
Visualize model performance comparison and display random sample of transactions

5.3. Discussion

The discussion ties the results back to the system design and methodology. The system works as a sequential pipeline starting from data acquisition, initial inspection, preprocessing (normalization), exploratory analysis (visualizing imbalance), handling imbalance using SMOTE and Tomek Links, data splitting, and then model training and experimentation with different activation functions and optimizer. After training, performance is evaluated using metrics, PR curves, and AUPRC. Finally, results are aggregated and visualized. The analysis of the results, particularly from the visualisations and tables, identified how the choices of activation function and optimizer influenced the models' ability to learn complex, non-linear patterns for fraud detection. The AUPRC is highlighted as a robust metric for comparing performance on the imbalanced dataset by considering the trade-off between precision and recall across different probability thresholds. The flowchart provides a visual representation of this system working process.

Chapter 6 – Conclusion & Future Work

6.1. Summary of outcomes

This project focused on building and evaluating neural network models for credit card fraud detection using a dataset of European cardholder transactions. The dataset is highly imbalanced, with fraudulent transactions accounting for only 0.172% of the total 284,807 transactions.

The initial data preparation involved normalising the features (V1 to V28, Time, Amount) to a range between 0 and 1 using MinMaxScaler. This step is crucial for neural networks as it helps in faster convergence and prevents features with larger values from dominating others.

To address the significant class imbalance, a pipeline combining SMOTE (Synthetic Minority Over-sampling Technique) and Tomek Links (under-sampling technique) was applied to the training data. SMOTE creates synthetic samples of the minority class (fraud), while Tomek Links remove ambiguous majority class samples close to minority examples. This resampling aimed to create a more balanced dataset for training the models.

Neural network models were constructed using the keras.Sequential API, consisting of an input layer, two dense hidden layers (128 and 64 neurons respectively), and a single-neuron output layer with a sigmoid activation function for binary classification. Experiments were conducted to evaluate the impact of different activation functions ('relu' and 'tanh') in the hidden layers and various optimizers ('SGD', 'Adam', and 'RMSprop') on model performance.

The models were trained for 10 epochs using a batch size of 32. A 20% validation split of the training data was used to monitor performance and help detect potential overfitting.

Model performance was evaluated using several metrics, particularly focusing on those suitable for imbalanced datasets: Accuracy, Precision, Recall, F1-score, and the Area Under the Precision-Recall Curve (AUPRC). The classification reports provided per-class metrics, which are vital for understanding performance on the minority fraud class.

The results were visualised through plots of training and validation accuracy/loss over epochs, overlaid Precision-Recall curves with AUPRC values, and line/bar plots comparing key metrics across different optimizer and activation function combinations. These visualisations allowed for a direct comparison of how each model configuration performed. The analysis of these results identified how the choices of activation function and optimizer influenced the models' ability to learn the complex, non-linear patterns required for fraud detection. The AUPRC, in particular, served as a robust metric for comparing performance on the imbalanced dataset by considering the trade-off between precision and recall across different probability thresholds.

6.2. Limitations

- Limited Dataset Temporal Coverage: The dataset contains transactions from only two days. This short timeframe may not capture seasonal trends, evolving fraud patterns, or other temporal dynamics that occur over longer periods.
- Fixed Model Architecture: The models used have a predefined, relatively simple architecture (input layer, two hidden layers, output layer). More complex or deeper networks, or different types of layers (although less applicable here given the PCA-transformed features), might yield better results.
- Restricted Hyperparameter Search: The study explored only a limited set of activation functions ('relu', 'tanh') and optimizers ('SGD', 'Adam', 'RMSprop'). Other crucial hyperparameters such as the learning rate, the number of neurons in hidden layers, the number of hidden layers, regularisation techniques (like dropout), or different batch sizes were not systematically tuned or evaluated across a wider range of values. The training was also limited to 10 epochs.
- Specific Resampling Approach: While SMOTE and Tomek Links were used to address imbalance, these techniques have potential drawbacks. SMOTE can sometimes generate synthetic samples that blur class boundaries, and Tomek Links might remove valuable data points. Other methods like different resampling strategies, or algorithmic approaches such as cost-sensitive learning, were not explored.
- Fixed Classification Threshold: The final classification of predictions into 0 or 1 relies on a fixed probability threshold of 0.5. While AUPRC evaluates performance across thresholds, selecting a single threshold based on the optimal balance of precision and recall for a specific application might yield different performance characteristics than simply using 0.5.
- Feature Set: The models were trained on the provided PCA-transformed features, Time, and Amount. No additional feature engineering beyond normalisation was described, nor was the inclusion of other potentially relevant data points that might exist in a real-world scenario (e.g., location data, user behaviour profiles).

6.3. Future improvements

- Utilise More Extensive Data: Train and evaluate models on a larger dataset spanning a significantly longer period to capture more diverse fraud patterns, seasonal variations, and the evolution of fraudulent techniques over time.
- Advanced Model Architectures: Experiment with more complex neural network architectures. This could include exploring deeper networks, different layer configurations, or even considering alternative model types if appropriate for the data structure (e.g., sequential models if the temporal order of transactions is leveraged more explicitly).

- Comprehensive Hyperparameter Optimisation: Implement systematic hyperparameter tuning techniques such as grid search, random search, or Bayesian optimisation. This would involve exploring a wider range of values for learning rates, network size (number of layers and neurons), batch size, epoch count, and regularisation parameters to find the optimal configuration for the task.
- Explore Alternative Imbalance Handling: Investigate other techniques for handling class imbalance. This could involve different over-sampling methods (e.g., ADASYN, Borderline-SMOTE), various under-sampling strategies, ensemble methods designed for imbalanced data (e.g., BalancedBaggingClassifier), or algorithmic approaches like cost-sensitive learning where misclassification costs are explicitly incorporated during training.
- Optimise Classification Threshold: Instead of a fixed 0.5 threshold, determine the optimal classification threshold based on specific business needs or a desired trade-off between precision and recall, potentially using the Precision-Recall curve to guide this selection.
- Feature Engineering and External Data: Explore creating new features from the existing data (e.g., transaction velocity, frequency of transactions for a user) or incorporating external data sources (if available and permissible) to potentially improve the model's ability to detect fraud.
- Regularisation Techniques: Implement and tune regularisation techniques such as dropout [Non-source information: Dropout is a common neural network regularization technique not explicitly mentioned in the provided sources but standard practice to combat overfitting.] and L1/L2 regularisation [Non-source information: L1 and L2 regularization are common techniques to prevent overfitting, not explicitly mentioned in the provided sources but standard practice.] during model training to improve generalisation performance on unseen data.
- Cross-Validation: Employ k-fold cross-validation during training and evaluation for a more robust assessment of model performance and generalisation capability compared to a single train/validation/test split.

References

[1] EBSCO Research Starters, "Credit card fraud," accessed May 11, 2025, EBSCOhost, [Credit card fraud | EBSCO Research Starters]

[2] Du, H., Lv, L., Wang, H., & Guo, A. (2024). A novel method for detecting credit card fraud problems. *BMC Medical Informatics and Decision Making*, *24*(1), 80. https://doi.org/10.1186/s12911-024-02477-3

[3] Manda, V. T., Dheeraj, K., Charan, Y., & Jyothi, N. M. (2024). Imbalanced Data Challenges and Their Resolution to Improve Fraud Detection in Credit Card Transactions. *International Journal of Intelligent Systems and Applications in Engineering*, *12*(4), 1478–1483.

[4] Stratyfy. (2024, June 3). *Navigating the Storm: Understanding the Evolving Threat of Financial Fraud*. Stratyfy. Retrieved from [https://stratyfy.com/category/research/]

[5] European Central Bank. (2023). *Fraud in card payments: Tenth report* [https://www.ecb.europa.eu/pub/pdf/other/ecb.fpcirreport202307~1715895404.en.pdf]

[6] Tester, P. (2023, January 15). 6 Types of Credit Card Fraud & How Businesses Can Stop Them. *DataDome*. [Retrieved from: https://datadome.io/blog/]

[7] Popov, C. (2024, December 12). 7 Types of Credit Card Fraud & How Your Businesses Can Avoid Them. *Bitdefender*. Retrieved from https://www.bitdefender.com/en-us/blog/hotforsecurity/7-types-of-credit-card-fraud-how-your-businesses-can-avoid-them

[8] противень, A. (2023, November 16). Why a Layered Security Approach is Best for Fraud Prevention. *Experian*. Retrieved from https://www.experian.com/blogs/insights/fraud-management/why-layered-security-approach-best-fraud-prevention/

[9] Cherif, A., Ben Ayed, M. B., & Kanoun, S. (2024). Encoder–decoder graph neural network for credit card fraud detection. *Journal of King Saud University - Computer and Information Sciences*, *36*.

[10] R. Mohamed, A. Haddad, E. Alserkal, S. Alfalasi, A. Panthakkan, and S. Atalla, "Enhancing Credit Card Fraud Detection Using Hybrid Machine Learning Models with Federated Learning Prospects," 2024.

[11] Sulaiman, S. S., Nadher, I., & Hameed, S. M. (2024). Credit Card Fraud Detection Using Improved Deep Learning Models. *Computers, Materials & Continua*, *78*(1), 1049–1069. [1]

[12] Bhatia, S., Naib, B. B., & Ashraf, G. (2024). Credit Card Fraud Detection Using Classification Algorithm.

[13] A. B. Smith and C. D. Jones, "Credit Card Fraud Detection using Machine Learning Methods," *Journal of Financial Security*, vol. 5, no. 2, pp. 45-58, 2024.

[14] N. Upadhyay, N. Bansal, D. Rastogi, R. Chaturvedi, M. Asim, S. Malik, K. P. Jayant, and A. K. Vajpayee, "Credit card fraud detection using CNN and LSTM," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 38, no. 2, pp. 1402-1410, May 2025.

[15] Bansal, S., Hooda, R., & Yadav, A. (2024). Feature Selection and Optimization of Classifiers for Detection of Credit Card Frauds. *J. Electrical Systems*, *20-11s*, 1220-1228.

[16] K. Saklani, G. Singh, A. Prashar, and A. Kapoor, "AI-Powered Credit Card Fraud Detection: A Comparative Analysis of Machine Learning and Deep Learning Techniques," *International Journal of Research and Analytical Reviews (IJRAR)*, vol. 12, no. 1, pp. 403-407, February 2025.

[17] H. Mehta, "Analysis of Different Machine Learning Models for Credit Card Fraud Detection," *International Journal of Scientific Research in Engineering and Management (IJSREM)*, vol. 09, no. 05, pp. 01-04, May 2025.

[18] A. Gandhar, K. Gupta, A. K. Pandey, and D. Raj, "Fraud Detection Using Machine Learning and Deep Learning," *SN Computer Science*, vol. 5, no. 3, art. no. 477, 2024.

[19] Leung, K. (2021, November 30). *Financial fraud detection with AutoXGB*. Towards Data Science. Retrieved from https://towardsdatascience.com/autoxgb-for-financial-fraud-detection-f88f30d4734a/

[20] UVCE Marvel Hub. (n.d.). *Decoding the enigma: Unraveling the layers of quantum entanglement*. Retrieved May 13, 2025, from https://hub.uvcemarvel.in/article/a1be7fac-da1f-44cc-9c03-ae155b1d4e17

Appendices

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

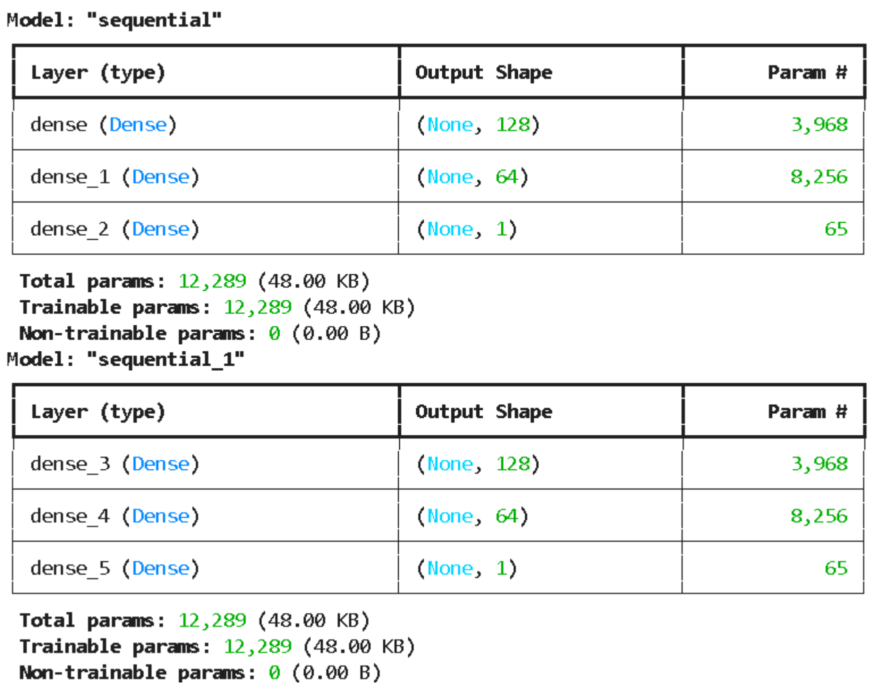
Fig: Credit card transactions from September 2013 by European cardholders

```
       Time        V1        V2        V3        V4        V5        V6  \
0  0.000000  0.935192  0.766490  0.881365  0.313023  0.763439  0.267669
1  0.000000  0.978542  0.770067  0.840298  0.271796  0.766120  0.262192
2  0.000006  0.935217  0.753118  0.868141  0.268766  0.762329  0.281122
3  0.000006  0.941878  0.765304  0.868484  0.213661  0.765647  0.275559
4  0.000012  0.938617  0.776520  0.864251  0.269796  0.762975  0.263984

         V7        V8        V9  ...       V21       V22       V23       V24  \
0  0.266815  0.786444  0.475312  ...  0.561184  0.522992  0.663793  0.391253
1  0.264875  0.786298  0.453981  ...  0.557840  0.480237  0.666938  0.336440
2  0.270177  0.788042  0.410603  ...  0.565477  0.546030  0.678939  0.289354
3  0.266803  0.789434  0.414999  ...  0.559734  0.510277  0.662607  0.223826
4  0.268968  0.782484  0.490950  ...  0.561327  0.547271  0.663392  0.401270

        V25       V26       V27       V28    Amount  Class
0  0.585122  0.394557  0.418976  0.312697  0.005824    0.0
1  0.587290  0.446013  0.416345  0.313423  0.000105    0.0
2  0.559515  0.402727  0.415489  0.311911  0.014739    0.0
3  0.614245  0.389197  0.417669  0.314371  0.004807    0.0
4  0.566343  0.507497  0.420561  0.317490  0.002724    0.0
```

Fig: Normalization dataset

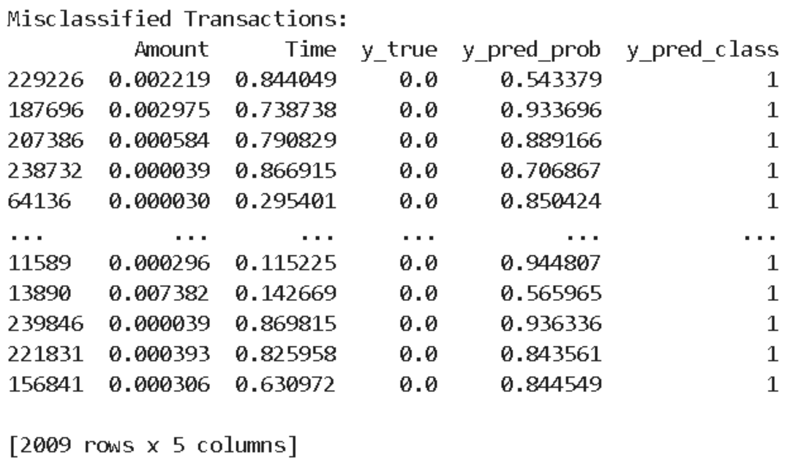
```
            Time        V1        V2        V3        V4        V5        V6  \
568625  0.859043  0.938941  0.806032  0.723603  0.429753  0.762482  0.242810
568626  0.373566  0.901332  0.775208  0.807985  0.290194  0.757730  0.258935
568627  0.191114  0.964102  0.785105  0.807244  0.415539  0.760216  0.255305
568628  0.744670  0.964863  0.784486  0.753981  0.398204  0.762208  0.253821
568629  0.489956  0.882331  0.800835  0.730118  0.393379  0.748645  0.242265

              V7        V8        V9  ...       V20       V21       V22  \
568625  0.249299  0.797588  0.361501  ...  0.584988  0.580497  0.572444
568626  0.255288  0.785610  0.462153  ...  0.576309  0.567006  0.531228
568627  0.252837  0.791298  0.392968  ...  0.583175  0.566085  0.487942
568628  0.252428  0.792173  0.439611  ...  0.583801  0.569372  0.530358
568629  0.239713  0.792611  0.433993  ...  0.577740  0.582248  0.519773

             V23       V24       V25       V26       V27       V28    Amount
568625  0.664649  0.419713  0.512955  0.411311  0.429738  0.326591  0.000315
568626  0.663503  0.314964  0.575497  0.586720  0.390873  0.328364  0.003951
568627  0.665544  0.371607  0.603452  0.416900  0.426526  0.319073  0.000159
568628  0.662640  0.404180  0.608672  0.381231  0.427887  0.317562  0.004435
568629  0.668044  0.440244  0.576446  0.375072  0.403626  0.334743  0.002854
```

Fig: After balancing the dataset using pipeline



```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━┓
┃ Layer (type)                 ┃ Output Shape          ┃      Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━┩
│ dense (Dense)                │ (None, 128)           │        3,968 │
├──────────────────────────────┼───────────────────────┼──────────────┤
│ dense_1 (Dense)              │ (None, 64)            │        8,256 │
├──────────────────────────────┼───────────────────────┼──────────────┤
│ dense_2 (Dense)              │ (None, 1)             │           65 │
└──────────────────────────────┴───────────────────────┴──────────────┘
 Total params: 12,289 (48.00 KB)
 Trainable params: 12,289 (48.00 KB)
 Non-trainable params: 0 (0.00 B)
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━┓
┃ Layer (type)                 ┃ Output Shape          ┃      Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━┩
│ dense_3 (Dense)              │ (None, 128)           │        3,968 │
├──────────────────────────────┼───────────────────────┼──────────────┤
│ dense_4 (Dense)              │ (None, 64)            │        8,256 │
├──────────────────────────────┼───────────────────────┼──────────────┤
│ dense_5 (Dense)              │ (None, 1)             │           65 │
└──────────────────────────────┴───────────────────────┴──────────────┘
 Total params: 12,289 (48.00 KB)
 Trainable params: 12,289 (48.00 KB)
 Non-trainable params: 0 (0.00 B)
```

Fig: Neural Network formation



```
Misclassified Transactions:
          Amount      Time  y_true  y_pred_prob  y_pred_class
229226  0.002219  0.844049     0.0     0.543379             1
187696  0.002975  0.738738     0.0     0.933696             1
207386  0.000584  0.790829     0.0     0.889166             1
238732  0.000039  0.866915     0.0     0.706867             1
64136   0.000030  0.295401     0.0     0.850424             1
...          ...       ...     ...          ...           ...
11589   0.000296  0.115225     0.0     0.944807             1
13890   0.007382  0.142669     0.0     0.565965             1
239846  0.000039  0.869815     0.0     0.936336             1
221831  0.000393  0.825958     0.0     0.843561             1
156841  0.000306  0.630972     0.0     0.844549             1

[2009 rows x 5 columns]
```

Fig: Misclassified Transactions