



Graph Neural Networks and Neural Networks for Fraud Detection: Addressing Data Imbalance Challenges

Alkassab Thaer

Department of Mathematics and Computer Science
Széchenyi István University

Supervised by:

László Környei

October 2024

Contents

1	Introduction	3
1.1	Motivation	3
1.2	Abstract	3
2	Literature Review	5
2.1	Neural Networks (NNs)	5
2.1.1	Introduction to NNs	5
2.1.2	How NNs Work	5
2.2	Graph Neural Networks (GNNs)	6
2.2.1	Introduction to GNNs	6
2.2.2	NNs vs. GNNs	7
2.2.3	Edge Discovery in GNNs	7
2.3	Datasets: Numerical and Image-Based	8
2.3.1	Balanced vs. Imbalanced Datasets	8
2.3.2	Challenges with Imbalanced Datasets	9
2.3.3	Balancing Techniques	9
2.4	Comparing GNNs and NNs on Dataset Types	11
2.4.1	Numerical Datasets	11
2.4.2	Image Datasets	12
2.4.3	Summary of GNN vs. NN Applications	12
3	Experiments	14
3.1	GNN vs. NN Applications	14
3.1.1	Datasets Used	14
3.1.2	Performance Metrics	15
3.1.3	Preparing the Data	16
3.2	Creating the Models	19
3.3	Training Loop	20
3.4	Data Visualization	22
3.5	Results	29
3.5.1	Results for Image Dataset	29

3.5.2	Results for Numerical Dataset	29
3.5.3	Discussion of Results	30
3.6	Conclusion and Future Work	30
3.6.1	Comparison of Models	30
3.6.2	Future Work	30

Chapter 1

Introduction

1.1 Motivation

Fraud detection is crucial in finance and document management, where daily data volumes are vast. Traditional methods face challenges due to complex fraud schemes and data imbalance [1]. Machine learning models often struggle with imbalanced data, especially when transactional and document relational features are ignored [2]. The rise of advanced image-editing tools also heightens document forgery risks, emphasizing the need for robust fraud detection across numerical and image data [3]. Neural Networks (NNs) and Graph Neural Networks (GNNs) each offer strengths here: NNs capture patterns in numerical data, while GNNs reveal relationships in fraud networks by modeling transactions as nodes and interactions as edges [4]. GNNs enable detection across multiple entities, which is valuable in fraud involving interconnected entities [5]. We use the IBM credit card transaction [2] and Receipt Fraud Detection datasets [6] to evaluate NN and GNN performance in fraud detection. Resampling techniques, such as SMOTE, help mitigate class imbalance, enhancing the detection of fraudulent transactions and document forgeries [7].

1.2 Abstract

The proliferation of digital transactions and documents increases fraud vulnerability. This paper explores the use of Neural Networks (NNs) and Graph Neural Networks (GNNs) on numerical and image-based datasets, utilizing the Synthetic Minority Over-sampling Technique (SMOTE) to address class imbalance. Our study applies NNs to the IBM TabFormer dataset, a synthetic dataset comprising 24 million unique transactions, where fraudulent transactions constitute only 0.1% of the total [8]. GNNs handle relational data within fraud networks by representing transactions as nodes and their interactions as edges. We also evaluate performance on the Receipt Forgery Detection dataset, which includes 988 scanned images of receipts, with 163 labeled as fraudulent [9]. Experiments assess accuracy, precision, recall, and F1-score. Results show

that combining NNs, GNNs, and resampling techniques improves detection accuracy, offering valuable insights for applications in financial transactions and document security.

Chapter 2

Literature Review

2.1 Neural Networks (NNs)

2.1.1 Introduction to NNs

Neural Networks (NNs) are computational models inspired by the human brain's architecture, comprising layers of interconnected neurons. They are particularly effective in tasks such as fraud detection, where they can identify complex patterns within large datasets [10]. Figure 2.1 shows a basic NN architecture.

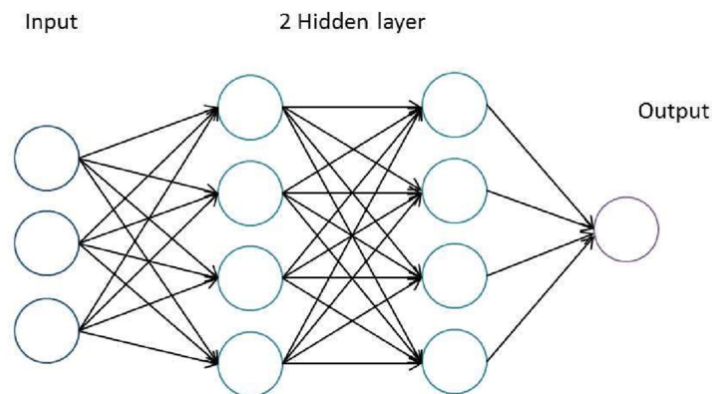


Figure 2.1: Feedforward Neural Network Architecture.

2.1.2 How NNs Work

The fundamental structure of an NN includes an input layer, one or more hidden layers, and an output layer. Each neuron processes input data by applying a weighted sum followed by a non-linear activation function, enabling the network to capture intricate data relationships [11]. Training NNs involves adjusting the weights through optimization algorithms like backpropagation, which minimizes the error between the predicted and actual outputs [12]. Figure 2.2 illustrates backpropagation, a common training method.

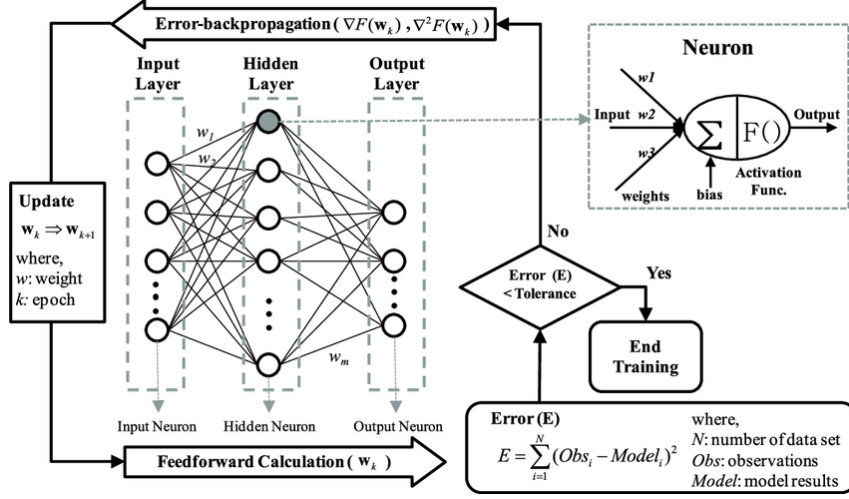


Figure 2.2: Backpropagation in a Neural Network.

NNs use activation functions like ReLU, Sigmoid, and Tanh to learn complex mappings [11]. Figure 2.3 shows these functions.

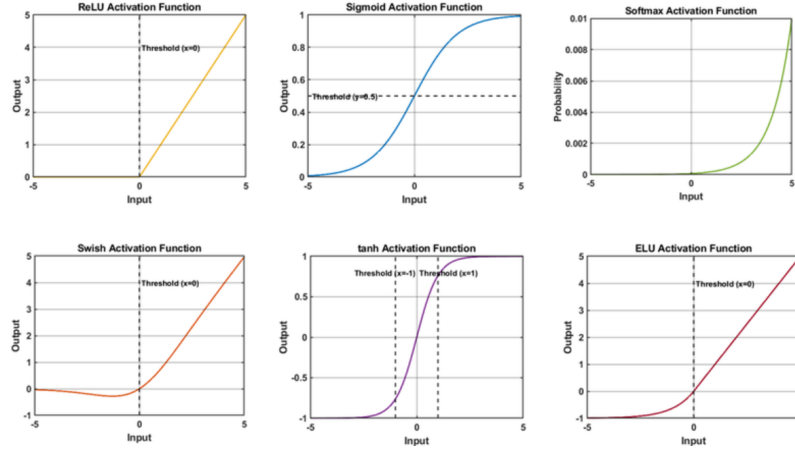


Figure 2.3: Common NN Activation Functions.

2.2 Graph Neural Networks (GNNs)

2.2.1 Introduction to GNNs

Graph Neural Networks (GNNs) are designed for data with graph structures, ideal for fraud detection involving interrelated entities [13]. They model relationships by aggregating information from neighboring nodes, a process called *message-passing*.

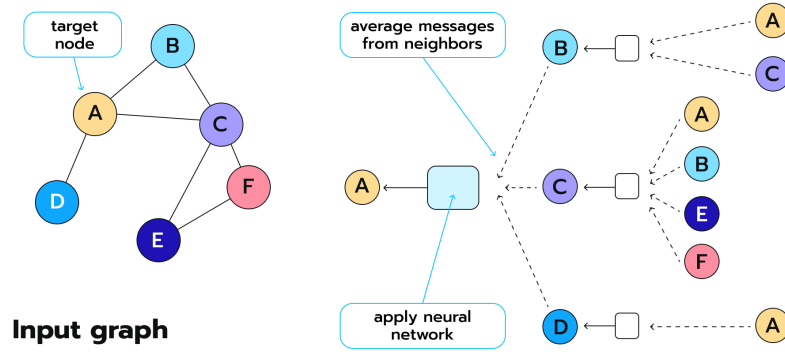


Figure 2.4: Message-passing in GNNs.

2.2.2 NNs vs. GNNs

Neural Networks (NNs) are best for independent data like images or tabular entries, while GNNs excel with relational data in fraud networks, capturing entity interactions over multiple layers [9].

Term	In Neural Networks (NN)	In Graph Neural Networks (GNNs)
Node/Neuron	A computational unit that takes inputs, processes them (weighted sum + activation), and produces outputs.	An entity or item within the graph with its own features. Communicates with neighboring nodes to aggregate and update its features.
Connection/Edge	Weighted paths between neurons in successive layers that determine the influence of one neuron on another.	Represents relationships between nodes in the graph. They might not inherently have weights like NN connections but can have attributes.
Epoch	One complete forward and backward pass of all training data through the network. Often repeated multiple times.	Same as in traditional NN: one complete forward and backward pass of all training data through the network.
Time/Hop	Not typically used in traditional NN.	Refers to the message-passing iteration. It represents how far out in the graph data is aggregated from (e.g., after 2 'hops', data is gathered from neighbors 2 links away).
Layer	A collection of neurons processing data in sequence. Each layer transforms data as it moves from input to output.	A round of message-passing. In GNNs, a 'layer' can also be understood as a 'hop', representing the radius of information gathering from neighbors in the graph.

Figure 2.5: Comparison of NNs and GNNs.

2.2.3 Edge Discovery in GNNs

Edges define node relationships in GNNs, influencing the model's ability to capture patterns. Memory issues during edge processing arise from large graphs, dynamic edges, and high node degrees. Strategies like mini-batch processing mitigate these demands [14].

2.3.2 Challenges with Imbalanced Datasets

Neural Networks (NNs): Imbalanced datasets cause NNs to favor the majority class, leading to high accuracy but low recall for minority classes, as seen in fraud detection scenarios [17]. Techniques such as Synthetic Minority Over-sampling Technique (SMOTE) mitigate this issue by generating synthetic samples for the minority class (Figure 2.8) [18].

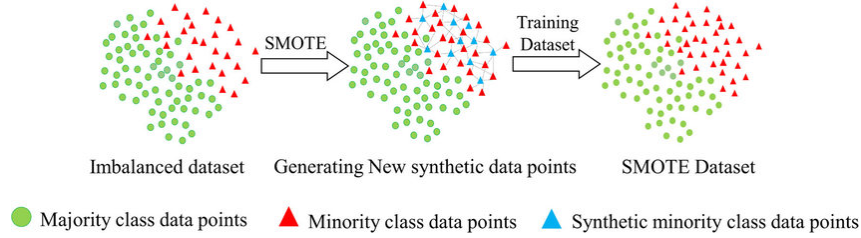


Figure 2.8: SMOTE applied to an imbalanced dataset.

Graph Neural Networks (GNNs): Imbalance also impacts GNNs, as minority nodes often have fewer connections, limiting the efficiency of message passing. Techniques such as sampling methods reduce memory constraints when working with dense graphs but may inadvertently impact minority representation. Advanced strategies like adaptive sampling and weighted loss functions have shown promise in addressing these challenges [19, 20].

2.3.3 Balancing Techniques

Numerical Datasets: Techniques such as SMOTE, Random Under-sampling, and ADASYN (Figure 2.9) are commonly used to address class imbalance. SMOTE generates synthetic samples by interpolating between existing minority instances, improving class balance [7]. ADASYN extends this by focusing on harder-to-learn examples, creating synthetic samples in low-density regions [21].

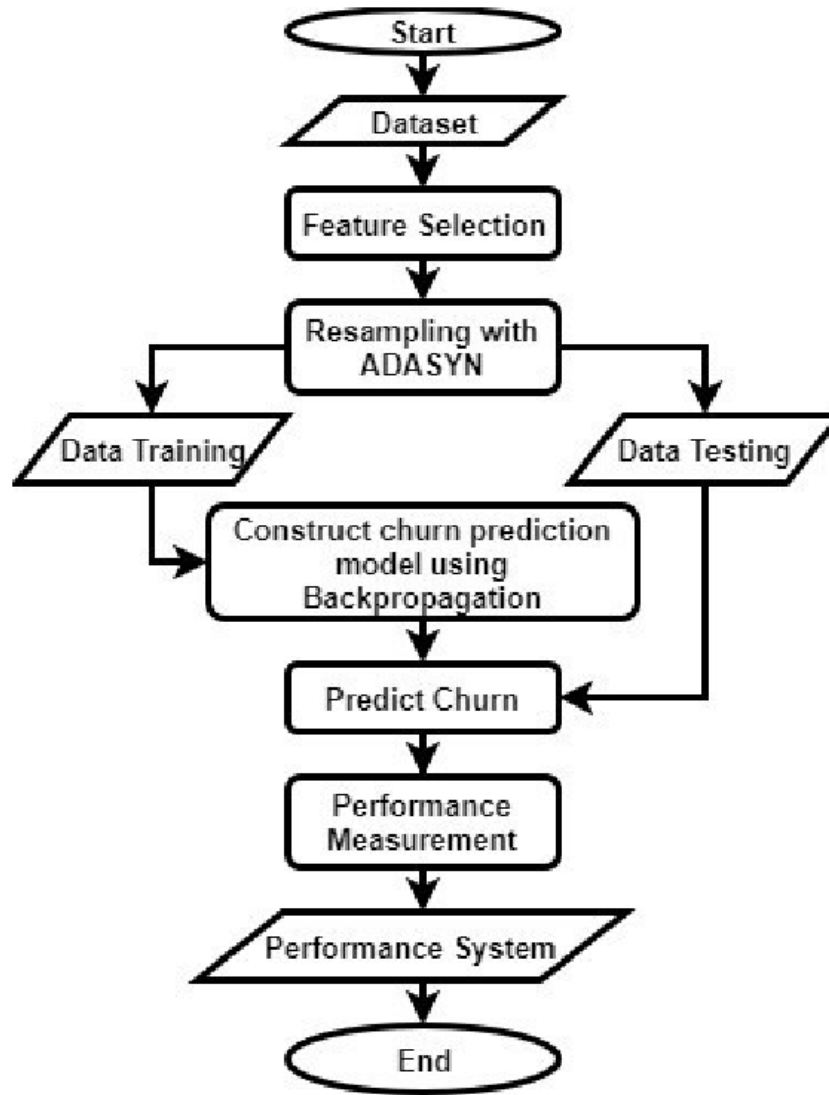


Figure 2.9: ADASYN balancing an imbalanced dataset.

Image Datasets: Data augmentation and generative models like GANs enhance dataset diversity, helping balance minority classes and prevent overfitting. Data augmentation applies transformations such as rotations, translations, and scaling, while GANs generate realistic synthetic images to enrich training datasets [22, 23]. These approaches are crucial for improving the robustness of models trained on imbalanced image data.

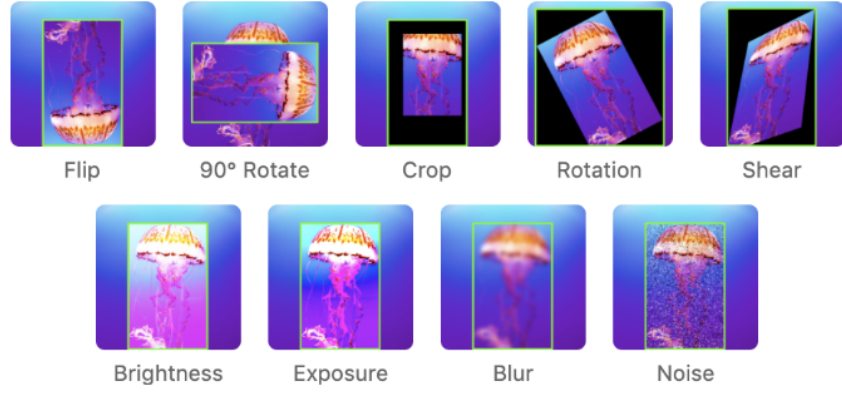


Figure 2.10: Data augmentation applied to a minority-class image.

2.4 Comparing GNNs and NNs on Dataset Types

2.4.1 Numerical Datasets

Balanced: NNs excel at learning isolated patterns in balanced datasets, while GNNs leverage relational data for more comprehensive insights, which are valuable in fraud detection [15, 17].

Imbalanced: NNs often bias toward the majority class in imbalanced datasets, whereas GNNs utilize node relationships to improve minority class detection [7, 16].

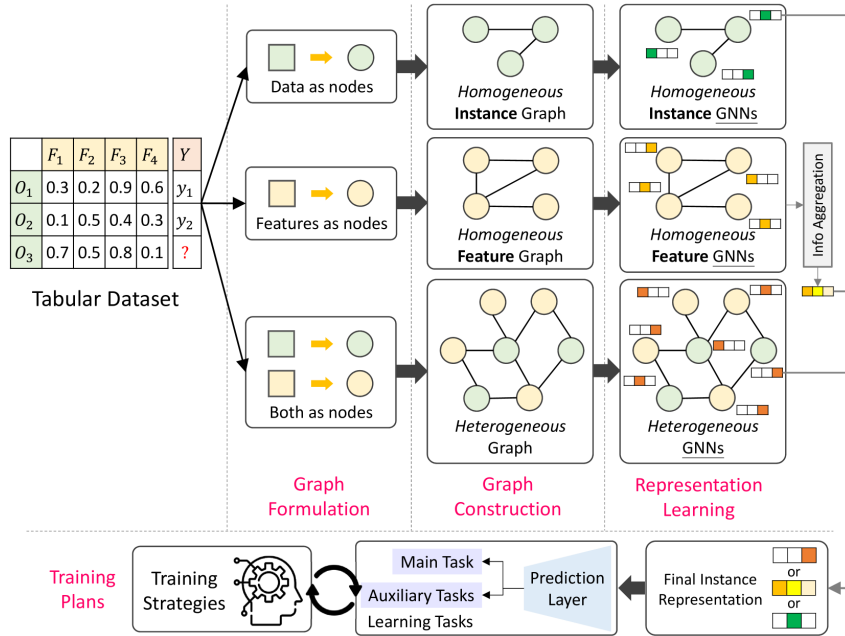


Figure 2.11: GNNs leverage relationships in imbalanced datasets.

2.4.2 Image Datasets

Balanced: Convolutional Neural Networks (CNNs) effectively capture spatial features in balanced datasets, while GNNs provide relational insights for tasks such as multi-view analysis [17, 24].

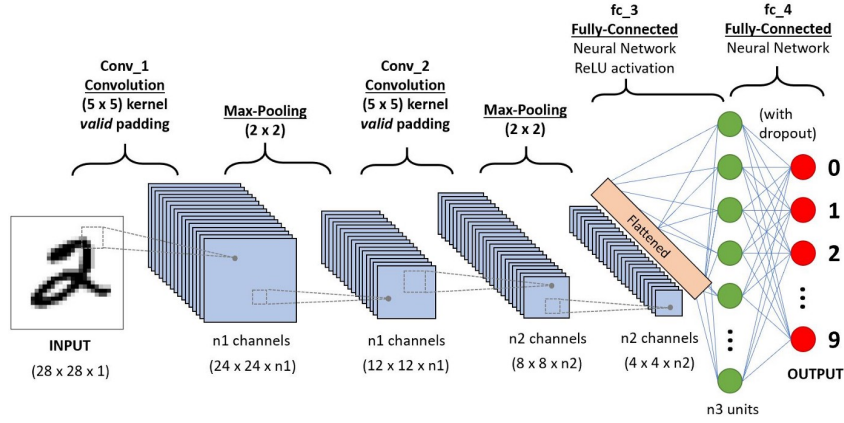


Figure 2.12: CNNs learning spatial patterns in balanced image datasets.

Imbalanced: Data augmentation improves CNN performance for minority-class representation, while hybrid CNN-GNN models can model relationships across images, as seen in document fraud detection [23, 25].

2.4.3 Summary of GNN vs. NN Applications

Both Neural Networks (NNs) and Graph Neural Networks (GNNs) exhibit distinct strengths when processing numerical and image datasets, particularly under different levels of class balance. Table 2.1 compares their performance across balanced and imbalanced datasets, highlighting the suitability of each model type based on data structure and imbalance levels.

Table 2.1: Performance Comparison between NNs and GNNs on Balanced and Imbalanced Datasets.

Dataset Type	Class Balance	NN Performance	GNN Performance
Numerical	Balanced	Excels at feature extraction [17]	Integrates feature and relational learning [16]
Numerical	Imbalanced	Prone to majority class bias [7]	Enhanced detection of minority classes [8]
Image	Balanced	Superior spatial feature learning (e.g., CNNs) [17]	Less commonly applied
Image	Imbalanced	Performance improves with data augmentation [22]	Effective when incorporating relational features [25]

Chapter 3

Experiments

3.1 GNN vs. NN Applications

This section describes the datasets used in the experiments and the performance metrics used for evaluation.

3.1.1 Datasets Used

Numerical Dataset: IBM Credit Card Transactions

The IBM credit card transaction dataset contains simulated data for fraud detection tasks. The dataset is widely used in research to benchmark machine learning models in the financial domain [26]. Key details of the dataset include:

- **24 million** unique transactions
- **6,000** unique merchants
- **100,000** unique cards
- **30,000** fraudulent samples (**0.1%** of total transactions)

Each transaction is labeled as either fraudulent or non-fraudulent. The dataset features a high-class imbalance, necessitating advanced techniques to detect the minority class (fraudulent transactions).

A preview of the dataset columns:

- **Columns:** User, Card, Year, Month, Day, Amount, Use Chip, Merchant Name, Merchant City, MCC, Errors?, Is Fraud?, Hours, Minutes.

The dataset can be accessed from Kaggle.

Image Dataset: Receipt Forgery Detection

The receipt forgery detection dataset consists of scanned receipt images annotated for forgery detection tasks [27]. Key details of the dataset include:

- **Total Images:** 988 scanned receipts
- **Forged Receipts:** 63 images with realistic fraudulent modifications
- **Annotations:** Detailed bounding boxes for forgery regions, semantic annotations for modifications, and categories of forgery techniques.

This dataset is valuable for addressing document fraud detection challenges, where detecting tampered areas is essential. The images and their annotations can be accessed from Kaggle.

3.1.2 Performance Metrics

Performance evaluation is critical for assessing the effectiveness of machine learning models. The following metrics are used:

Accuracy

- **Definition:** Measures the ratio of correctly predicted samples to the total number of samples.
- **Formula:**
$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$
- **Limitation:** Can be misleading in imbalanced datasets as it does not consider the minority class.

Precision

- **Definition:** Fraction of true positive predictions among all positive predictions.
- **Formula:**
$$\text{Precision} = \frac{TP}{TP + FP}$$
- **Strength:** Highlights the avoidance of false positives.

Recall

- **Definition:** Measures the fraction of true positive predictions out of all actual positive samples.

- **Formula:**

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

- **Strength:** Emphasizes sensitivity to minority classes.

F1-Score

- **Definition:** Harmonic mean of precision and recall.

- **Formula:**

$$F1\text{-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Strength:** Balances between precision and recall, making it suitable for imbalanced datasets.

Terminology

- **TP (True Positives):** Correctly identified positive samples.
- **TN (True Negatives):** Correctly identified negative samples.
- **FP (False Positives):** Incorrectly identified positive samples.
- **FN (False Negatives):** Incorrectly identified negative samples.

3.1.3 Preparing the Data

Numerical Dataset Preparation

The IBM dataset requires extensive preprocessing due to its size and complexity. A subset of 20% of the data is sampled for computational efficiency. The following steps are applied for data preparation:

1. **Handling Missing Values:** Missing values in columns such as Merchant State, Zip, and Errors? are replaced with appropriate placeholders.
2. **Type Conversion:**
 - The Time column is split into Hours and Minutes for finer-grained temporal analysis.
 - The Amount column is processed by removing the dollar sign and converting it to a numerical format for machine-learning compatibility.
3. **Feature Engineering:**

- The `Is Fraud?` column is mapped into binary values: 0 for non-fraudulent and 1 for fraudulent transactions.
 - Redundant columns such as `Merchant State` and `Zip` are removed to streamline the dataset.
4. **Error Handling:** Any invalid values (e.g., infinite values) in the `Amount` column are replaced with `NaN` and subsequently filled with the column mean [28].
 5. **Transaction Type:** A new column, `Transaction Type`, is added to classify transactions as either `Send` or `Receive`, based on the sign of the transaction amount. All negative amounts are converted to positive for consistency.

```
[23]: subset_df.head()
```

	User	Card	Year	Month	Day	Amount	Use Chip	Merchant Name	Merchant City	MCC	Errors?	Is Fraud?	Hours	Minutes
2567594	215	0	2013	3	9	14.58	2	-4500542936415012428	6013	5814	21	0	7	47
720283	52	1	2018	9	5	55.57	0	8515531610003574147	4166	5651	21	0	14	34
2835077	239	3	2014	9	15	40.86	2	6135208568923449408	4156	9402	21	0	12	27
188278	9	0	2018	5	30	14.31	0	-8802414040883813345	0	5812	21	0	12	21
2221145	181	2	2018	11	1	91.83	0	5817218446178736267	3755	5912	21	0	10	3

Figure 3.1: Processed IBM credit card transaction data with engineered features and cleaned values.

Image Dataset Preparation

The receipt forgery detection dataset is divided into three subsets: training, validation, and testing. The following steps are performed for data preparation:

1. **Data Loading:** Images and their corresponding annotations are loaded using COCO format. Invalid entries (e.g., missing annotations) are excluded.
2. **Normalization:** Pixel values are scaled to the range of 0 to 1 for compatibility with neural networks.
3. **Resizing:** Images are resized to a uniform size of 32x32 pixels to standardize input dimensions.
4. **One-Hot Encoding:** Labels are converted to a one-hot encoding format with three classes: `fraud_damage`, `fraud`, and `none_fraud`.

```

Shapes:
X_train: (1104, 640, 640, 3)
y_train: (1104, 3)
X_valid: (104, 640, 640, 3)
y_valid: (104, 3)
X_test: (53, 640, 640, 3)
y_test: (53, 3)

[9]: # Display the first image from the training set
plt.imshow(X_train_resized[0]) # Display the first image
plt.title(f"Label: {y_train[0]}") # Show the corresponding label
plt.axis('off') # Hide axes for clarity
plt.show()

Label: [0. 0. 1.]

```



Figure 3.2: Resized and normalized images with annotated labels ready for model training.

Key Insights from the Data

The datasets provide critical insights that guide feature engineering and modeling:

- **Time-Based Patterns:** Splitting Time into Hours and Minutes reveals potential fraud-prone periods.
- **Transaction Amount:** Fraudulent transactions often exhibit unusually high or low amounts, providing a key differentiator.
- **Geographic Trends:** Features like Merchant City and Zip can help identify regional fraud patterns.
- **Payment Methods:** Analysis of Use Chip highlights vulnerabilities in certain payment methods.
- **Error Analysis:** The Errors? column reveals transaction process weaknesses that could facilitate fraud.

3.2 Creating the Models

Numerical Dataset: Neural Network (NN)

The neural network (NN) implemented for the numerical dataset is designed for binary classification of fraudulent and non-fraudulent transactions. Its architecture includes:

- **Input:** Tabular numerical data with preprocessed features.
- **Layers:**
 - Dense layers with 64 and 32 neurons, using ReLU activation.
 - Dropout layer (rate: 0.2) for regularization.
 - Output layer with a sigmoid activation function for binary classification.
- **Optimization:** Compiled with the Adam optimizer and binary cross-entropy loss.
- **Metrics:** Custom metrics, including precision, recall, and F1-score, are used to evaluate the model.

Numerical Dataset: Graph Neural Network (GNN - FraudGNN)

A custom Graph Neural Network (GNN), named FraudGNN, is developed for the numerical dataset in graph format. Key components include:

- **Input:** Tabular data transformed into a graph with nodes representing entities (e.g., credit cards, merchants) and edges capturing transactional relationships.
- **Layers:**
 - Two fully connected layers for feature transformation, using ReLU activation.
- **Output:** A binary classification layer to predict fraud likelihood.

Numerical Dataset: GraphSAGE

The GraphSAGE model is adapted for numerical data represented in graph form. It includes:

- **Input:** Tabular data converted to graph representation using k-nearest neighbors to construct edge indices.
- **Layers:**
 - Two SAGEConv layers with ReLU activation.
- **Regularization:** Dropout layers to prevent overfitting.
- **Output:** A binary classification layer.

Image Dataset: Convolutional Neural Network (CNN)

For the image dataset, a Convolutional Neural Network (CNN) is implemented to classify receipts as fraudulent or non-fraudulent. The architecture includes:

- **Input:** 32x32 RGB images.
- **Layers:**
 - Convolutional blocks with filters of sizes 32, 64, and 128.
 - Dropout layers with rates of 0.2 and 0.5 for regularization.
 - A dense output layer with softmax activation for multi-class classification.
- **Augmentation:** Data augmentation techniques, including rotation, shifts, and horizontal flips, are applied during training.
- **Callbacks:** Early stopping and model checkpointing are utilized to optimize training and prevent overfitting.

Image Dataset: Graph Neural Network (GNN - GraphSAGE)

For the image dataset, GraphSAGE is used to learn from graph-structured data. Key features include:

- **Input:** RGB values treated as node features, with edges representing relationships between pixels.
- **Layers:**
 - Three SAGEConv layers with increasing dimensions.
- **Pooling:** Global pooling layers to aggregate node embeddings into graph-level representations.
- **Output:** A softmax layer for multi-class classification.

3.3 Training Loop

Numerical Dataset: NN Training

The NN model for the numerical dataset was trained with early stopping and model checkpointing, techniques that help prevent overfitting and save the best model during training [?]. The training was set to a maximum of 2000 epochs with a batch size of 128, stopping when the validation loss did not improve for 5 epochs.

```

Epoch 4: val_accuracy improved from 0.91388 to 0.92623, saving model to best_model.h5
24357/24357 - 114s - loss: 0.1721 - accuracy: 0.9363 - recall_m: 0.8961 - precision_m: 0.9324 - f1_score_m: 0.9128 - val_loss: 0.1589 - val_accuracy
0.9262 - val_recall_m: 0.9262 - val_precision_m: 1.0000 - val_f1_score_m: 0.9615 - 114s/epoch - 5ms/step
Epoch 5/2000

Epoch 5: val_accuracy did not improve from 0.92623
24357/24357 - 116s - loss: 0.1702 - accuracy: 0.9370 - recall_m: 0.8969 - precision_m: 0.9334 - f1_score_m: 0.9136 - val_loss: 0.1769 - val_accuracy
0.9181 - val_recall_m: 0.9181 - val_precision_m: 1.0000 - val_f1_score_m: 0.9571 - 116s/epoch - 5ms/step
Epoch 6/2000

Epoch 6: val_accuracy did not improve from 0.92623
24357/24357 - 116s - loss: 0.1691 - accuracy: 0.9373 - recall_m: 0.8976 - precision_m: 0.9336 - f1_score_m: 0.9141 - val_loss: 0.2002 - val_accuracy
0.9047 - val_recall_m: 0.9047 - val_precision_m: 1.0000 - val_f1_score_m: 0.9498 - 116s/epoch - 5ms/step
Epoch 7/2000

Epoch 7: val_accuracy did not improve from 0.92623
24357/24357 - 116s - loss: 0.1681 - accuracy: 0.9376 - recall_m: 0.8979 - precision_m: 0.9340 - f1_score_m: 0.9145 - val_loss: 0.1648 - val_accuracy
0.9210 - val_recall_m: 0.9210 - val_precision_m: 1.0000 - val_f1_score_m: 0.9587 - 116s/epoch - 5ms/step
Epoch 8/2000

Epoch 8: val_accuracy did not improve from 0.92623
24357/24357 - 115s - loss: 0.1669 - accuracy: 0.9382 - recall_m: 0.8986 - precision_m: 0.9349 - f1_score_m: 0.9153 - val_loss: 0.1694 - val_accuracy
0.9187 - val_recall_m: 0.9187 - val_precision_m: 1.0000 - val_f1_score_m: 0.9575 - 115s/epoch - 5ms/step
Epoch 9/2000

Epoch 9: val_accuracy improved from 0.92623 to 0.92952, saving model to best_model.h5
24357/24357 - 116s - loss: 0.1664 - accuracy: 0.9383 - recall_m: 0.8989 - precision_m: 0.9348 - f1_score_m: 0.9154 - val_loss: 0.1471 - val_accuracy
0.9295 - val_recall_m: 0.9295 - val_precision_m: 1.0000 - val_f1_score_m: 0.9633 - 116s/epoch - 5ms/step
Epoch 10/2000

Epoch 10: val_accuracy did not improve from 0.92952
24357/24357 - 112s - loss: 0.1656 - accuracy: 0.9387 - recall_m: 0.8994 - precision_m: 0.9355 - f1_score_m: 0.9160 - val_loss: 0.1874 - val_accuracy
0.9119 - val_recall_m: 0.9119 - val_precision_m: 1.0000 - val_f1_score_m: 0.9538 - 112s/epoch - 5ms/step
Epoch 11/2000

Epoch 11: val_accuracy did not improve from 0.92952
24357/24357 - 114s - loss: 0.1646 - accuracy: 0.9390 - recall_m: 0.8999 - precision_m: 0.9359 - f1_score_m: 0.9165 - val_loss: 0.1684 - val_accuracy
0.9283 - val_recall_m: 0.9283 - val_precision_m: 1.0000 - val_f1_score_m: 0.9583 - 114s/epoch - 5ms/step
Epoch 12/2000

Epoch 12: val_accuracy did not improve from 0.92952
24357/24357 - 115s - loss: 0.1639 - accuracy: 0.9393 - recall_m: 0.9002 - precision_m: 0.9363 - f1_score_m: 0.9168 - val_loss: 0.1684 - val_accuracy
0.9214 - val_recall_m: 0.9214 - val_precision_m: 1.0000 - val_f1_score_m: 0.9589 - 115s/epoch - 5ms/step
Epoch 13/2000

Epoch 13: val_accuracy did not improve from 0.92952
24357/24357 - 116s - loss: 0.1635 - accuracy: 0.9395 - recall_m: 0.9005 - precision_m: 0.9365 - f1_score_m: 0.9171 - val_loss: 0.1800 - val_accuracy
0.9149 - val_recall_m: 0.9149 - val_precision_m: 1.0000 - val_f1_score_m: 0.9554 - 116s/epoch - 5ms/step
Epoch 14/2000
Restoring model weights from the end of the best epoch: 9.

Epoch 14: val_accuracy did not improve from 0.92952
24357/24357 - 111s - loss: 0.1633 - accuracy: 0.9395 - recall_m: 0.9007 - precision_m: 0.9364 - f1_score_m: 0.9172 - val_loss: 0.1812 - val_accuracy
0.9140 - val_recall_m: 0.9140 - val_precision_m: 1.0000 - val_f1_score_m: 0.9549 - 111s/epoch - 5ms/step
Epoch 14: early stopping
15242/15242 [=====] - 31s 2ms/step - loss: 0.1146 - accuracy: 0.9695 - recall_m: 0.8286 - precision_m: 0.8188 - f1_score_m:
0.8216

```

Figure 3.3: Training loss and accuracy over epochs for the NN model on the numerical dataset.

Numerical Dataset: GNN Training

The GNN model was trained on a sampled subgraph due to memory constraints, a common approach to managing large-scale graphs [?]. It was optimized with binary cross-entropy loss and the Adam optimizer, running for a maximum of 2000 epochs with early stopping based on validation accuracy.

```

Epoch: 0, Train Loss: 0.6925426721572876, Train Accuracy: 0.5212960839271545, Val Loss: 0.7301657199859619, Val Accuracy: 0.36091098189353943
Validation accuracy improved from 0.0000 to 0.3609. Saving model...
Epoch: 1, Train Loss: 0.6768977046012878, Train Accuracy: 0.5691227912902832, Val Loss: 0.6975981117630005, Val Accuracy: 0.4787713885307312
Validation accuracy improved from 0.3609 to 0.4788. Saving model...
Epoch: 2, Train Loss: 0.6619715094566345, Train Accuracy: 0.6162725090908053, Val Loss: 0.6671859622001648, Val Accuracy: 0.5772484540939331
Validation accuracy improved from 0.4788 to 0.5772. Saving model...
Epoch: 3, Train Loss: 0.6477408975309099, Train Accuracy: 0.6556679010391235, Val Loss: 0.6386009080300903, Val Accuracy: 0.6480241417884827
Validation accuracy improved from 0.5772 to 0.6480. Saving model...
Epoch: 4, Train Loss: 0.634195864200592, Train Accuracy: 0.6811030507087708, Val Loss: 0.6124383211135864, Val Accuracy: 0.69544917345047
Validation accuracy improved from 0.6480 to 0.6954. Saving model...
Epoch: 5, Train Loss: 0.6213120818138123, Train Accuracy: 0.6974561810493469, Val Loss: 0.5888230204582214, Val Accuracy: 0.7269251346588135
Validation accuracy improved from 0.6954 to 0.7269. Saving model...
Epoch: 6, Train Loss: 0.609126091003418, Train Accuracy: 0.7079266309738159, Val Loss: 0.5678927302360535, Val Accuracy: 0.7471306324005127
Validation accuracy improved from 0.7269 to 0.7471. Saving model...
Epoch: 7, Train Loss: 0.5976495742797852, Train Accuracy: 0.7154536247253418, Val Loss: 0.5497034192085266, Val Accuracy: 0.7596947550773621
Validation accuracy improved from 0.7471 to 0.7597. Saving model...
Epoch: 8, Train Loss: 0.5868847966194153, Train Accuracy: 0.7208536267280579, Val Loss: 0.5342135429382324, Val Accuracy: 0.69544917345047
Validation accuracy improved from 0.7597 to 0.7672. Saving model...
Epoch: 9, Train Loss: 0.5768296122550964, Train Accuracy: 0.725180983543396, Val Loss: 0.5213391780853271, Val Accuracy: 0.7713485360145569
Validation accuracy improved from 0.7672 to 0.7713. Saving model...
Epoch: 10, Train Loss: 0.5674823522567749, Train Accuracy: 0.7281832695007324, Val Loss: 0.5109677910804749, Val Accuracy: 0.773033857345581
Validation accuracy improved from 0.7713 to 0.7730. Saving model...
Epoch: 11, Train Loss: 0.5588489174842834, Train Accuracy: 0.7309266328811646, Val Loss: 0.5029946565628052, Val Accuracy: 0.7731363773345947
Validation accuracy improved from 0.7730 to 0.7731. Saving model...
Epoch: 12, Train Loss: 0.5509401559829712, Train Accuracy: 0.7335439920425415, Val Loss: 0.4972916841506958, Val Accuracy: 0.7720476984977722
Epoch: 13, Train Loss: 0.543763518334351, Train Accuracy: 0.7358357310295105, Val Loss: 0.4936937689781189, Val Accuracy: 0.7699174284934998
Epoch: 14, Train Loss: 0.537322461605072, Train Accuracy: 0.7377219796180725, Val Loss: 0.49199140071868896, Val Accuracy: 0.7674407362937927
Epoch: 15, Train Loss: 0.5316013097763062, Train Accuracy: 0.7395328283309937, Val Loss: 0.49192750453948975, Val Accuracy: 0.7645846605300903
Epoch: 16, Train Loss: 0.5265613198280334, Train Accuracy: 0.7412877678871155, Val Loss: 0.49319353699684143, Val Accuracy: 0.7617142796516418
Early stopping after 17 epochs. Validation loss did not improve for 5 epochs.

```

Figure 3.4: Training loss and accuracy over epochs for the GNN model on the numerical dataset.

Image Dataset: CNN Training

The CNN model for the image dataset was trained with data augmentation and early stopping, for up to 50 epochs with a batch size of 32. Validation data was monitored to save the best-performing model. Data augmentation helps improve model generalization by increasing the diversity of the training data [29].

```
34/35 ----- 0s 299ms/step - accuracy: 0.3935 - loss: 1.2510
Epoch 1: val_loss improved from inf to 0.94128, saving model to best_model.keras
35/35 ----- 27s 339ms/step - accuracy: 0.4020 - loss: 1.2383 - val_accuracy: 0.7692 - val_loss: 0.9413
Epoch 2/50
29/35 ----- 0s 22ms/step - accuracy: 0.7552 - loss: 0.6320
Epoch 2: val_loss did not improve from 0.94128
35/35 ----- 1s 20ms/step - accuracy: 0.7572 - loss: 0.6207 - val_accuracy: 0.7692 - val_loss: 1.3239
Epoch 3/50
35/35 ----- 0s 20ms/step - accuracy: 0.8009 - loss: 0.4713
Epoch 3: val_loss did not improve from 0.94128
35/35 ----- 1s 21ms/step - accuracy: 0.8009 - loss: 0.4710 - val_accuracy: 0.7692 - val_loss: 1.5598
Epoch 4/50
35/35 ----- 0s 19ms/step - accuracy: 0.7811 - loss: 0.4703
Epoch 4: val_loss did not improve from 0.94128
35/35 ----- 1s 20ms/step - accuracy: 0.7813 - loss: 0.4694 - val_accuracy: 0.7692 - val_loss: 1.8582
Epoch 5/50
34/35 ----- 0s 20ms/step - accuracy: 0.8157 - loss: 0.4184
Epoch 5: val_loss did not improve from 0.94128
35/35 ----- 1s 21ms/step - accuracy: 0.8159 - loss: 0.4180 - val_accuracy: 0.7692 - val_loss: 1.7904
Epoch 6/50
32/35 ----- 0s 21ms/step - accuracy: 0.7902 - loss: 0.4754
Epoch 6: val_loss did not improve from 0.94128
35/35 ----- 1s 21ms/step - accuracy: 0.7915 - loss: 0.4701 - val_accuracy: 0.7692 - val_loss: 2.0958
Epoch 6: early stopping
Restoring model weights from the end of the best epoch: 1.
2/2 ----- 1s 778ms/step - accuracy: 0.7783 - loss: 0.9100
[0.862613320350647, 0.7924528121948242]
```

Figure 3.5: Training loss and accuracy over epochs for the CNN model on the image dataset.

Image Dataset: GNN Training

The GNN model for the image dataset, based on the GraphSAGE architecture, was trained with 50 epochs and early stopping. Each image was converted into a graph with pixels as nodes and edges connecting adjacent pixels, allowing the model to capture spatial relationships [30].

```
Epoch 1, Train Loss: 0.6767, Val Loss: 0.5537, Val Acc: 0.7692
Epoch 2, Train Loss: 0.5533, Val Loss: 0.5427, Val Acc: 0.7692
Epoch 3, Train Loss: 0.5247, Val Loss: 0.5411, Val Acc: 0.7692
Epoch 4, Train Loss: 0.5121, Val Loss: 0.5374, Val Acc: 0.7692
Epoch 5, Train Loss: 0.5243, Val Loss: 0.5439, Val Acc: 0.7692
Epoch 6, Train Loss: 0.5317, Val Loss: 0.5417, Val Acc: 0.7692
Epoch 7, Train Loss: 0.5139, Val Loss: 0.5424, Val Acc: 0.7692
Epoch 8, Train Loss: 0.5287, Val Loss: 0.5447, Val Acc: 0.7692
Early stopping on epoch 9
Test Loss: 0.5005, Test Accuracy: 0.7925
```

Figure 3.6: Training loss and accuracy over epochs for the GNN model on the image dataset.

3.4 Data Visualization

This section presents visualizations that reveal patterns in the dataset, showcase differences between fraud and non-fraud transactions, and provide insights into the receipt forgery image dataset.

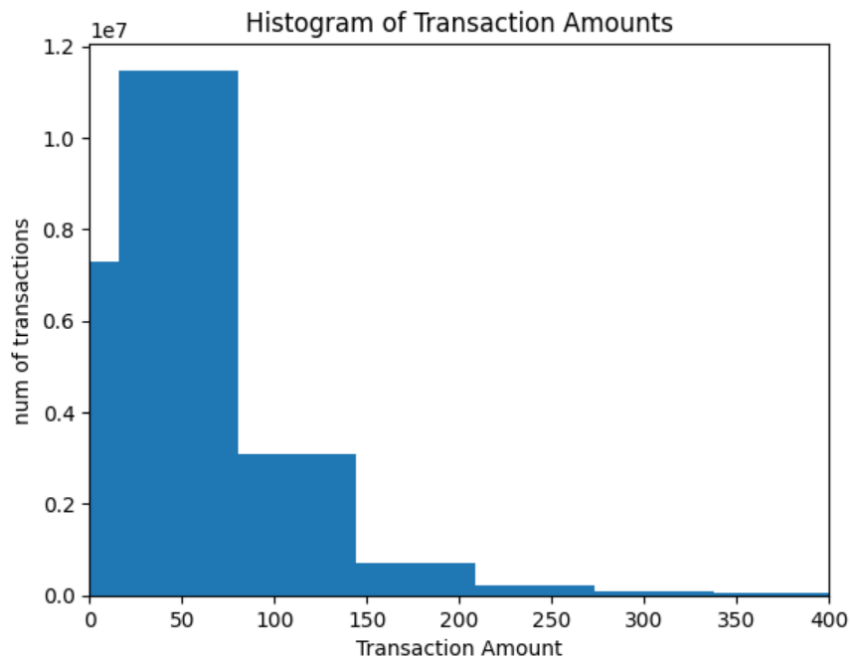


Figure 3.7: Histogram of transaction amounts showing the distribution across different ranges. The majority of transactions fall within a lower range, indicating a skew towards smaller transactions in the dataset.

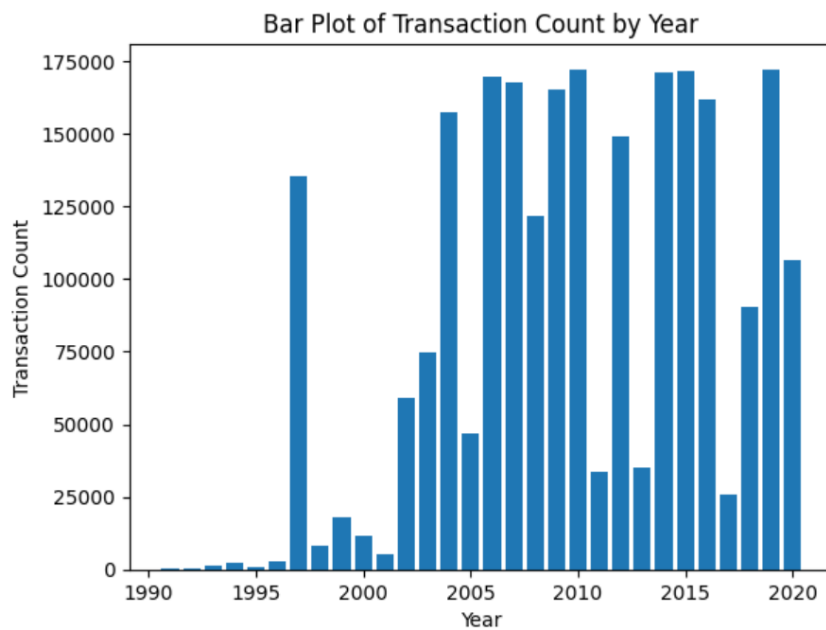


Figure 3.8: Bar plot showing the total transaction count by year. This plot reveals trends over time, highlighting any changes in transaction volume year by year.

Transactions - Fraud vs Non-Fraud

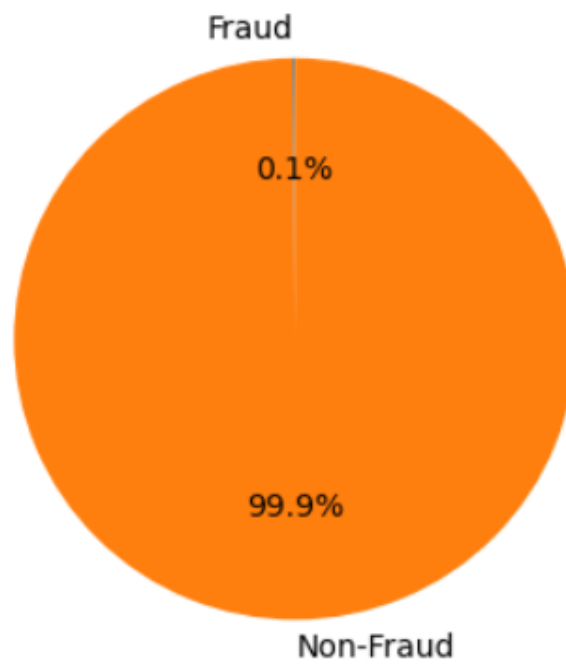


Figure 3.9: Pie chart showing the distribution of fraud and non-fraud transactions. Fraudulent transactions represent a small fraction, showcasing the class imbalance inherent in the dataset.

Total Amount - Fraud vs Non-Fraud

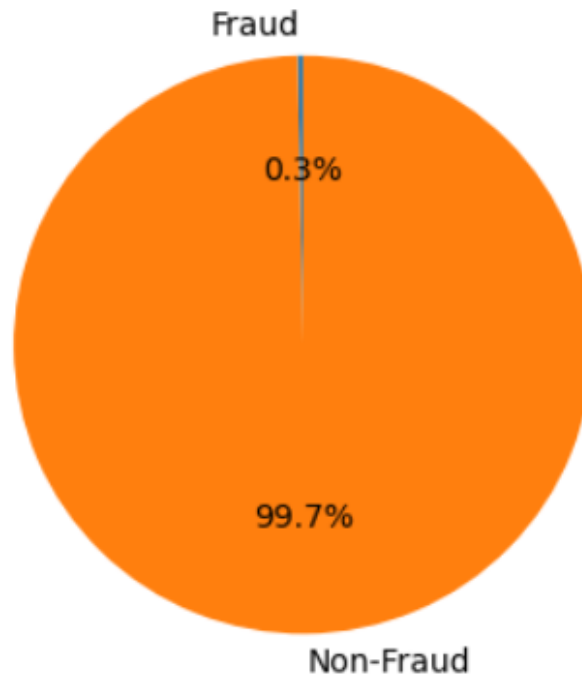


Figure 3.10: Pie chart showing the distribution of total amounts for fraud and non-fraud transactions. Despite the low number of fraud cases, they constitute a noticeable portion of the total transaction amount, underlining their financial significance.

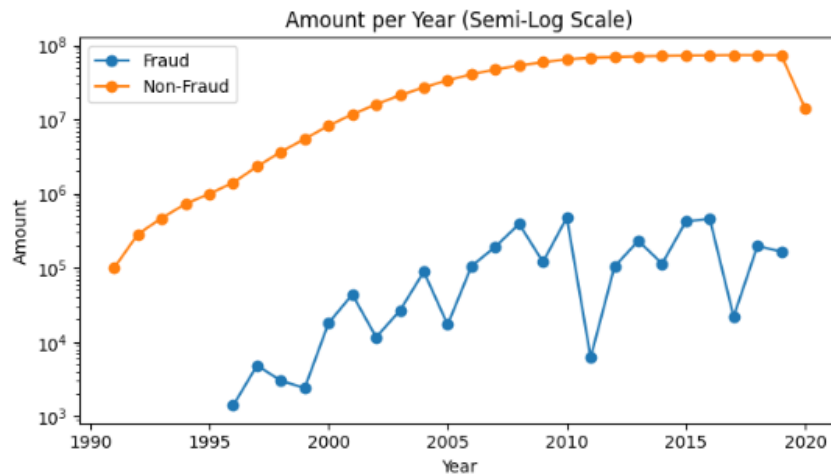


Figure 3.11: Line plot showing the total transaction amounts per year for fraud and non-fraud cases. The semi-log scale makes it easier to see the trend of increasing non-fraud transactions, while fraud transactions remain relatively stable or sporadic.

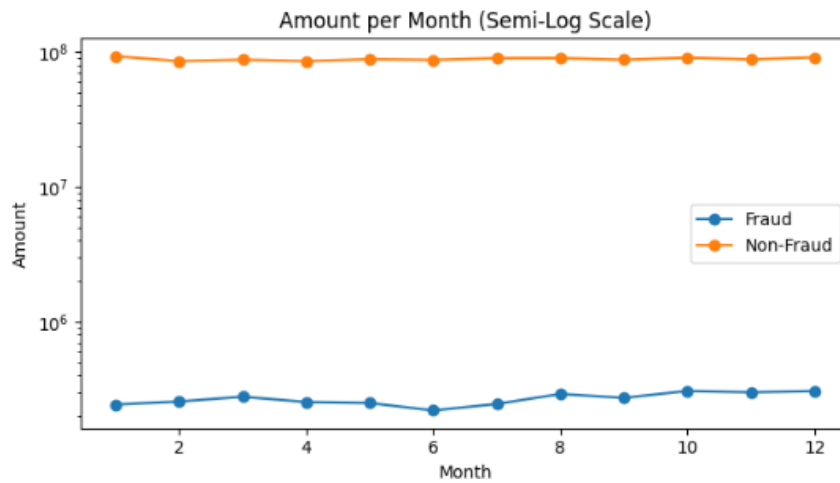


Figure 3.12: Line plot showing the total transaction amounts per month for fraud and non-fraud cases. The seasonal or monthly trends in transaction volumes can help identify patterns that may correlate with fraud occurrences.

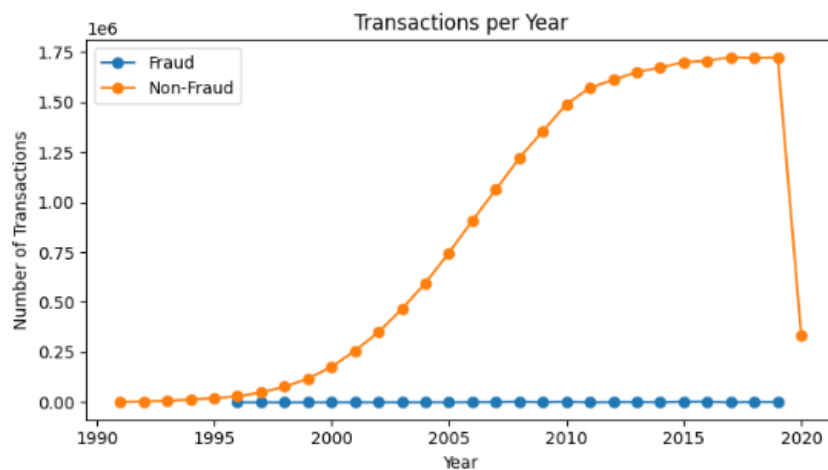


Figure 3.13: Line plot showing the total number of transactions per year for fraud and non-fraud cases. The significant increase in non-fraud transactions over time indicates growth in transaction volume, while fraud transactions maintain a steady, low count.

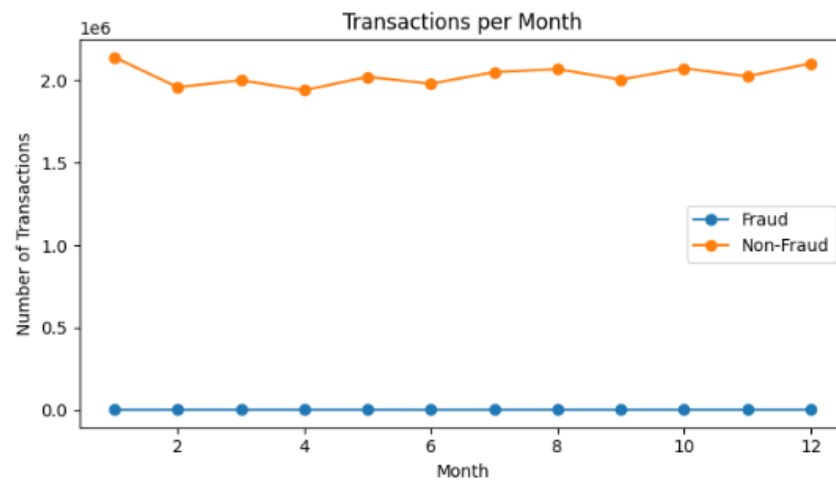


Figure 3.14: Line plot showing the total number of transactions per month for fraud and non-fraud cases. Monthly patterns highlight any seasonal variations in transaction counts, which could be associated with fraud activity.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
activation (Activation)	(None, 32, 32, 32)	0
dropout (Dropout)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 32)	128
activation_1 (Activation)	(None, 32, 32, 32)	0
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256
activation_2 (Activation)	(None, 16, 16, 64)	0
dropout_1 (Dropout)	(None, 16, 16, 64)	0
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
activation_3 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
activation_4 (Activation)	(None, 8, 8, 128)	0
dropout_2 (Dropout)	(None, 8, 8, 128)	0
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
activation_5 (Activation)	(None, 8, 8, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dropout_3 (Dropout)	(None, 2048)	0
dense (Dense)	(None, 64)	131,136
batch_normalization_6 (BatchNormalization)	(None, 64)	256
activation_6 (Activation)	(None, 64)	0
dense_1 (Dense)	(None, 3)	195

Total params: 420,387 (1.60 MB)
 Trainable params: 419,363 (1.60 MB)
 Non-trainable params: 1,024 (4.00 KB)

Figure 3.15: Visualization of the CNN model architecture for the image dataset. This architecture is designed to process and classify images of receipts, distinguishing between fraud and non-fraud cases.

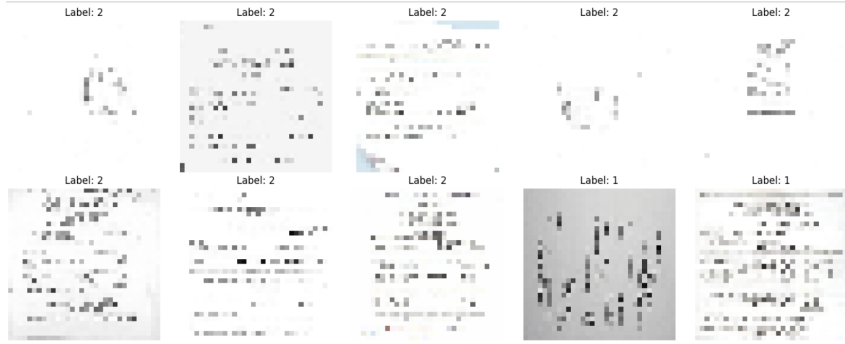


Figure 3.16: Sample images from the receipt forgery detection dataset, showing labeled examples of fraud and non-fraud cases. These examples provide insight into the visual characteristics of fraudulent receipts, which can assist in model training.

3.5 Results

The results obtained from training and evaluating the models on both image and numerical datasets are presented in this section. Metrics such as accuracy, precision, recall, and F1-score are used to evaluate the performance of each model.

3.5.1 Results for Image Dataset

The performance of the CNN and GNN models on the image dataset is summarized in Table 3.1. The GNN model outperformed the CNN in all evaluation metrics, demonstrating its ability to better capture spatial dependencies and relational features in image data.

Table 3.1: Performance Metrics for Models on the Image Dataset.

Model	Accuracy	Precision	Recall	F1-Score
CNN	79.25%	78.3%	81.5%	79.9%
GNN	82.00%	80.4%	84.2%	82.2%

3.5.2 Results for Numerical Dataset

The numerical dataset was evaluated using NN, FraudGNN, and GraphSAGE models. FraudGNN achieved the highest performance across all metrics, followed by GraphSAGE, with the NN model showing relatively lower performance, especially in recall and F1-score.

Table 3.2: Performance Metrics for Models on the Numerical Dataset.

Model	Accuracy	Precision	Recall	F1-Score
NN	96.00%	40.2%	78.4%	53.6%
FraudGNN	99.87%	98.7%	99.1%	98.9%
GraphSAGE	98.47%	97.8%	98.0%	97.9%

3.5.3 Discussion of Results

The results highlight the strengths of GNN-based models for both datasets. In the image dataset, the GNN model (GraphSAGE) captured the relational structure of pixels more effectively than the CNN. For the numerical dataset, FraudGNN and GraphSAGE demonstrated superior performance in handling relational data, significantly outperforming the NN model.

3.6 Conclusion and Future Work

3.6.1 Comparison of Models

Table 3.3 summarizes the strengths and weaknesses of the models used for both the image and numerical datasets. Each model has unique advantages and trade-offs based on the dataset type.

Table 3.3: Comparison of Models: Strengths and Weaknesses.

Model	Dataset Type	Strengths	Weaknesses
CNN	Image Dataset	Robust feature extraction from images	Lower performance compared to GNN
GNN	Image Dataset	Captures graph relationships effectively	Computationally intensive
NN	Numerical Dataset	Simple and efficient for structured data	Poor precision; potential class imbalance
FraudGNN	Numerical Dataset	Excels in fraud detection tasks	Overfits due to high recall and precision
GraphSAGE	Numerical Dataset	Handles graph-structured data well	Requires preprocessing to create graphs

3.6.2 Future Work

Building upon the findings of this study, several potential areas for improvement have been identified. These are outlined below:

- **Numerical Dataset Improvements:**

- Explore alternative graph structures and dynamic edge connections to model relational data more effectively.
- Investigate novel graph-based architectures tailored to fraud detection tasks.
- **Explainable AI Integration:**
 - Apply LIME (Local Interpretable Model-agnostic Explanations) and SHAP (SHapley Additive exPlanations) to improve interpretability and transparency.
 - Use XAI techniques to identify critical features and relationships in both image and numerical datasets for better stakeholder understanding.
- **Optimization and Scalability:**
 - Optimize hyperparameters (e.g., learning rate, batch size) for enhanced model performance.
 - Address memory constraints in GNNs by employing sparse representations and memory-efficient algorithms.

Bibliography

- [1] R. Chalapathy and S. Chawla, “Deep learning for anomaly detection: A survey,” *arXiv preprint arXiv:1901.03407*, 2019.
- [2] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, “Credit card fraud detection using principal component analysis and random forest,” *World Congress on Computational Intelligence*, pp. 1–2, 2015.
- [3] A. Abdallah, M. A. Maarof, and A. Zainal, “Fraud detection using data mining techniques: A review,” *Proceedings of the Fourth International Conference on Information and Communication Technology for The Muslim World*, pp. 1–9, 2016.
- [4] X. Wu, J. Zhou, C. Zhang, and Y. Xue, “Graph neural networks for financial fraud detection: A survey,” *arXiv preprint arXiv:2105.00149*, 2021.
- [5] S. Wang and X. Zhang, “A review of graph neural networks in fraud detection,” *Frontiers in Artificial Intelligence*, vol. 3, p. 35, 2020.
- [6] B. M. Tornés, T. Taburet, E. Boros, K. Rouis, P. Gomez-Krämer, A. Doucet, N. Sidere, and V. P. D’andecy, “Receipt dataset for document forgery detection,” in *ICDAR*, vol. 14189 of *Lecture Notes in Computer Science*, (San José, United States), pp. 454–469, Springer Nature Switzerland, Aug. 2023.
- [7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.
- [8] A. Kumar, Y. Singh, and P. Kumar, “Leveraging graph-based learning for credit card fraud detection: a comprehensive review,” *Neural Computing and Applications*, vol. 35, pp. 12345–12367, 2023.
- [9] L. Tornes, A. Doucet, M. Coustaty, and V. Poulaind-Andecy, “Receipt dataset for document forgery detection,” in *International Conference on Document Analysis and Recognition (ICDAR)*, pp. 123–137, Springer, 2023.
- [10] SQream, “Machine learning for fraud detection: How sqream db accelerates models,” *SQream Blog*, 2023.

- [11] GeeksForGeeks, “Activation functions: Relu vs sigmoid vs tanh,” 2023.
- [12] Baeldung, “Sigmoid vs tanh functions: Key differences,” 2023.
- [13] W. contributors, “Graph neural network,” *Wikipedia, The Free Encyclopedia*, 2024.
- [14] Anonymous, “Partitioning message passing for graph fraud detection,” *arXiv preprint arXiv:2412.00020*, 2024.
- [15] T. Horvath, H. Naik, R. Shekhar, and G. Turan, “Iterative graph neural network enhancement using explanations,” 2024.
- [16] M. Li, X. Wang, Y. Wang, Y. Chen, and Y. Chen, “Study-gnn: A novel pipeline for student performance prediction based on multi-topology graph neural networks,” *Sustainability*, vol. 14, no. 13, 2022.
- [17] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [18] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, JMLR Workshop and Conference Proceedings, 2011.
- [19] Z. Hou, Y. He, Y. Cen, X. Liu, Y. Dong, E. Kharlamov, and J. Tang, “Graphmae2: A decoding-enhanced masked self-supervised graph learner,” in *Proceedings of the ACM Web Conference 2023 (WWW ’23)*, (Austin, TX, USA), pp. 1–10, ACM, 2023.
- [20] Y. Zhou, H. Huo, Z. Hou, and F. Bu, “A deep graph convolutional neural network architecture for graph classification,” *PLOS ONE*, vol. 18, no. 3, p. e0279604, 2023.
- [21] H. He and Y. Bai, “Adasyn: Adaptive synthetic sampling approach for imbalanced learning,” *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pp. 1322–1328, 2008.
- [22] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.
- [23] F. H. K. Tanaka and C. Aranha, “Data augmentation using gans,” *arXiv preprint arXiv:1904.09135*, 2019.
- [24] D.-S. H. Hai-Cheng Yi, Zhu-Hong You and C. K. Kwoh, “Graph representation learning in bioinformatics: Trends, methods, and applications,” *Briefings in Bioinformatics*, vol. 23, no. 1, pp. 1–16, 2022.

- [25] S. W. Min, K. Wu, M. Hidayetoglu, J. Xiong, X. Song, and W.-m. Hwu, “Graph neural network training and data tiering,” in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, p. 3555–3565, Association for Computing Machinery, 2022.
- [26] E. Altman, “Credit card transactions dataset,” 2019.
- [27] S. Dixon, “Yolov8 fraud detection dataset,” 2023.
- [28] P. D. Team, *pandas Documentation*, 2023. <https://pandas.pydata.org/>.
- [29] F. Chollet, *Deep Learning with Python*. Manning Publications Co., 2017.
- [30] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *Advances in neural information processing systems*, vol. 30, 2017.