A Mini-Project Report

for **Deep Learning and Applications**

Title: **Credit Card Fraud Detection**

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Department of Electronics and Communication Engineering

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY, PATIALA, PUNJAB

November 2024

Index

TITLE	PAGE NO
INTRODUCTION	3
DATA COLLECTION AND PREPARATION	6
MODELING	7
PERFORMANCE EVALUATION	8
FLOW CHARTS	9
SIMULATION RESULTS	10
REFERENCES	12

Introduction

Credit card fraud is a significant issue in the financial sector, resulting in billions of dollars lost each year. Traditional fraud detection systems often rely on rule-based algorithms that may not adapt well to evolving fraud patterns. These systems can generate a high number of false positives, leading to customer dissatisfaction and increased operational costs. The goal of this project is to develop a deep learning model that can effectively identify fraudulent transactions in real-time, improving both detection rates and reducing false positives compared to existing methods.

Problem: Financial fraud, especially credit card fraud, has become a significant issue, causing substantial monetary losses globally. Detecting fraudulent transactions accurately and efficiently is crucial in minimizing these losses. The objective of this project is to create a deep learning model that detects fraud in credit card transactions.

Existing Solutions

Classical models like logistic regression and decision trees have been used for fraud detection but often struggle with the subtlety and imbalance present in transaction data. Deep learning models, such as neural networks, can capture complex patterns, helping to improve fraud detection rates.
Current fraud detection systems primarily use supervised learning techniques, such as logistic regression, decision trees, and ensemble methods. While these methods can detect some fraudulent activities, they often struggle with the following limitations:

- **High False Positive Rates**: Many legitimate transactions may be flagged as fraudulent.
- **Inability to Adapt**: Rule-based systems cannot adapt to new fraud patterns without manual intervention.
- **Imbalanced Datasets**: Fraudulent transactions are rare compared to legitimate ones, leading to class imbalance that traditional algorithms may not handle well.

Deep learning models, particularly those based on Neural Networks, have shown promise in handling complex patterns and can adapt to new data more effectively.

Solution: This model uses a neural network to classify transactions as fraudulent or legitimate. We aim to improve fraud detection accuracy by recognizing patterns within transaction data that are often missed by traditional rule-based approaches.

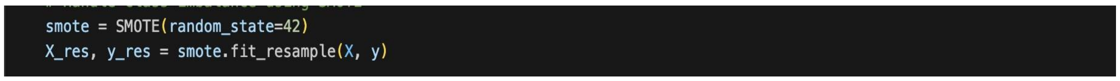
Novelty Statement: This study introduces a more nuanced approach to handling imbalanced datasets using SMOTE and evaluates the effectiveness of deep learning models in fraud detection, providing deeper insights into model performance metrics.

Data Collection and Preparation

For this project, we will use the **Kaggle Credit Card Fraud Detection Dataset**. This dataset contains 284,807 transactions made by credit cards in September 2013 by European cardholders, with 492 fraudulent transactions.

Steps for Data Preparation:

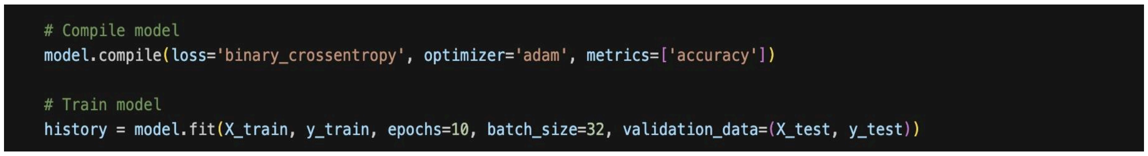
1. **Data Cleaning**: Check for missing values and remove any irrelevant features.
2. **Feature Engineering**: Scale features using StandardScaler or MinMaxScaler to normalize the data.
3. **Handling Imbalance**: Use techniques such as SMOTE (Synthetic Minority Over-sampling Technique) to balance the dataset.



```
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)
```

Fig 1.1 handling imbalance using SMOTE

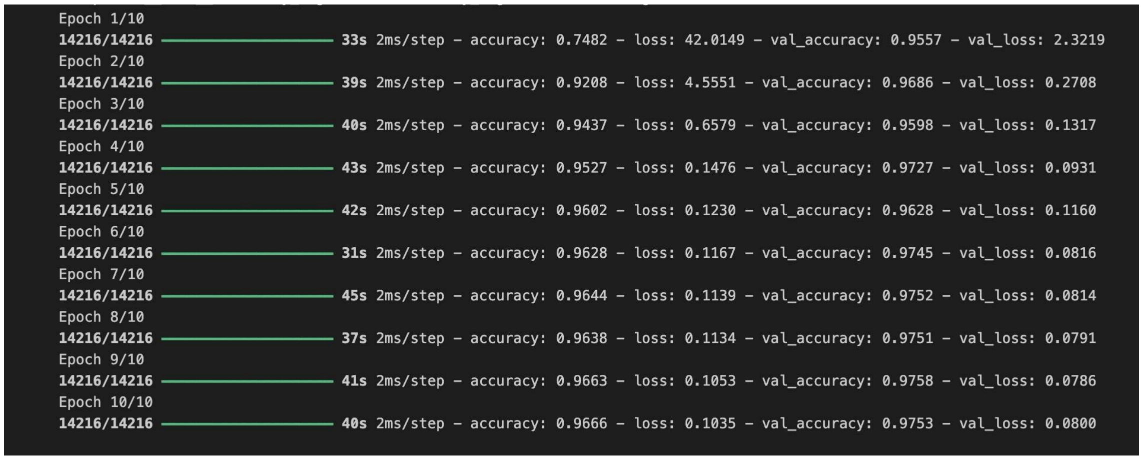
4. **Train/Test Split**: Split the dataset into training (70%), validation (15%), and test (15%) sets.



```
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train model
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

Fig 1.2 compiling and training model on the designed model



```
Epoch 1/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 33s 2ms/step - accuracy: 0.7482 - loss: 42.0149 - val_accuracy: 0.9557 - val_loss: 2.3219
Epoch 2/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 39s 2ms/step - accuracy: 0.9208 - loss: 4.5551 - val_accuracy: 0.9686 - val_loss: 0.2708
Epoch 3/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 40s 2ms/step - accuracy: 0.9437 - loss: 0.6579 - val_accuracy: 0.9598 - val_loss: 0.1317
Epoch 4/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 43s 2ms/step - accuracy: 0.9527 - loss: 0.1476 - val_accuracy: 0.9727 - val_loss: 0.0931
Epoch 5/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 42s 2ms/step - accuracy: 0.9602 - loss: 0.1230 - val_accuracy: 0.9628 - val_loss: 0.1160
Epoch 6/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 31s 2ms/step - accuracy: 0.9628 - loss: 0.1167 - val_accuracy: 0.9745 - val_loss: 0.0816
Epoch 7/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 45s 2ms/step - accuracy: 0.9644 - loss: 0.1139 - val_accuracy: 0.9752 - val_loss: 0.0814
Epoch 8/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 37s 2ms/step - accuracy: 0.9638 - loss: 0.1134 - val_accuracy: 0.9751 - val_loss: 0.0791
Epoch 9/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 41s 2ms/step - accuracy: 0.9663 - loss: 0.1053 - val_accuracy: 0.9758 - val_loss: 0.0786
Epoch 10/10
14216/14216 ━━━━━━━━━━━━━━━━━━━━ 40s 2ms/step - accuracy: 0.9666 - loss: 0.1035 - val_accuracy: 0.9753 - val_loss: 0.0800
```

Fig 1.3 Training model for epoch=10

Dataset Source: The dataset for this project is sourced from Kaggle's Credit Card Fraud Detection dataset. It contains 284,807 credit card transactions over two days, with 492 cases marked as fraud.

Features: The dataset is anonymized with features labeled V1 to V28, and two columns: **Amount** (transaction amount) and **Class** (0 for non-fraud, 1 for fraud).

Data Preparation:

- Handle data imbalance using techniques like oversampling for the minority class.
- Scale the **Amount** column to ensure similar feature ranges.
- Split data into training and testing sets.

Modeling

We will implement a feedforward Neural Network (FNN) for the classification of transactions as fraudulent or legitimate. The proposed model architecture will consist of the following layers:

1. Input layer (corresponding to the number of features)
2. Hidden layers with activation functions (ReLU)
3. Dropout layers (to prevent overfitting)
4. Output layer with a sigmoid activation function for binary classification

Frameworks and Libraries:

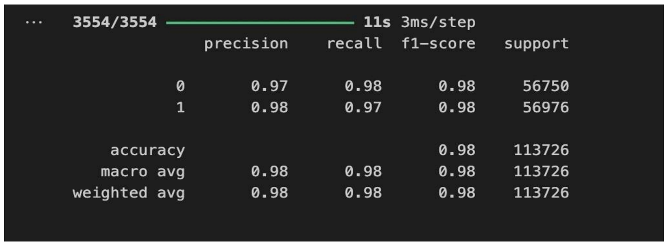
- TensorFlow/Keras for model development
- Scikit-learn for data preprocessing and evaluation metrics
- Pandas for data manipulation

Model Architecture:

- Input Layer: Features from the dataset.
- Two hidden layers:
  - **Hidden Layer 1**: 64 neurons with ReLU activation.
  - **Hidden Layer 2**: 32 neurons with ReLU activation.
- Output Layer: 1 neuron with **Sigmoid** activation (to classify fraud as 1 and non-fraud as 0).

Model Compilation:

- Loss function: **Binary Crossentropy**
- Optimizer: **Adam**
- Metrics: **Accuracy**



```
...  3554/3554 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step
              precision    recall  f1-score   support

           0       0.97      0.98      0.98     56750
           1       0.98      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 1.4 Prediction on test data & evaluation metrics

Performance Evaluation

The model's performance will be evaluated using the following metrics:

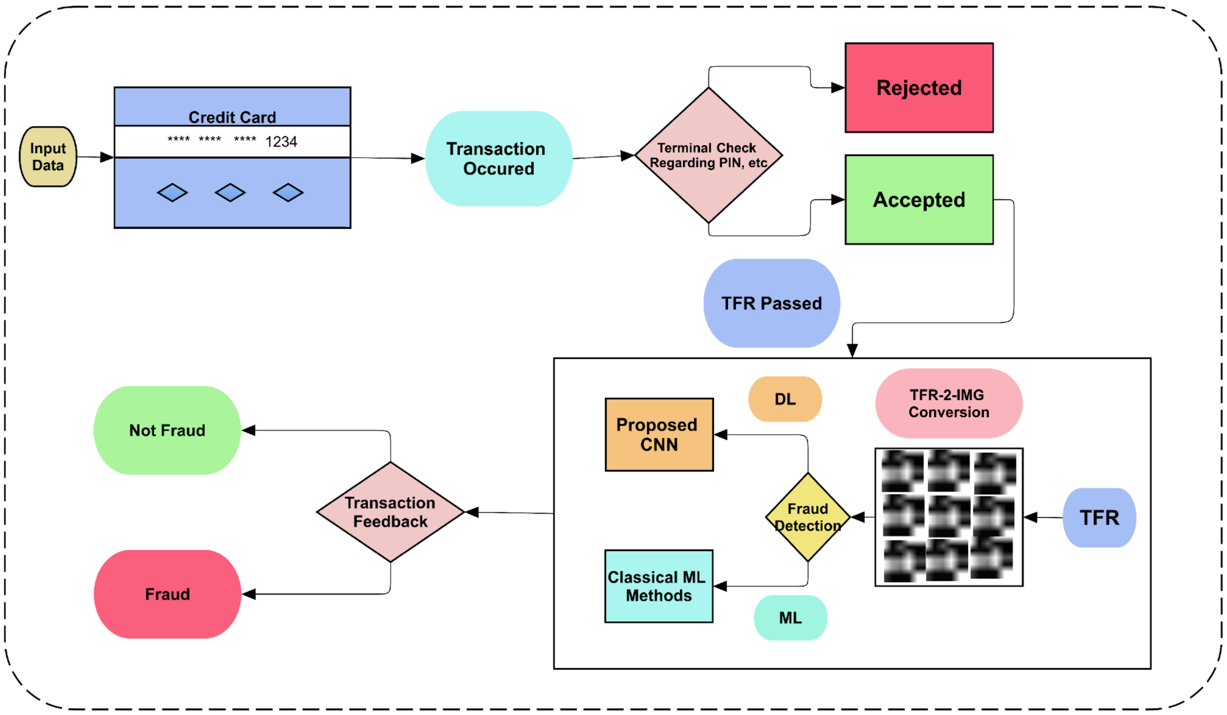
- **Accuracy**: Overall correctness of the model.
- **Precision**: The ratio of true positive predictions to the total predicted positives (important to minimize false positives).
- **Recall**: The ratio of true positive predictions to the actual positives (important to minimize false negatives).
- **F1 Score**: The harmonic mean of precision and recall, providing a balance between the two.
- **AUC-ROC Curve**: To evaluate the model’s ability to distinguish between classes.

We will compare our model's performance against traditional machine learning algorithms (e.g., logistic regression, random forests) using the same dataset.

Challenges

- **Imbalanced Dataset**: The dataset has a significant class imbalance, which can affect model performance.
- **Overfitting**: The model may perform well on training data but poorly on unseen data.
- **Computational Resources**: Training deep learning models can be resource-intensive, requiring access to GPUs.

Flow Charts



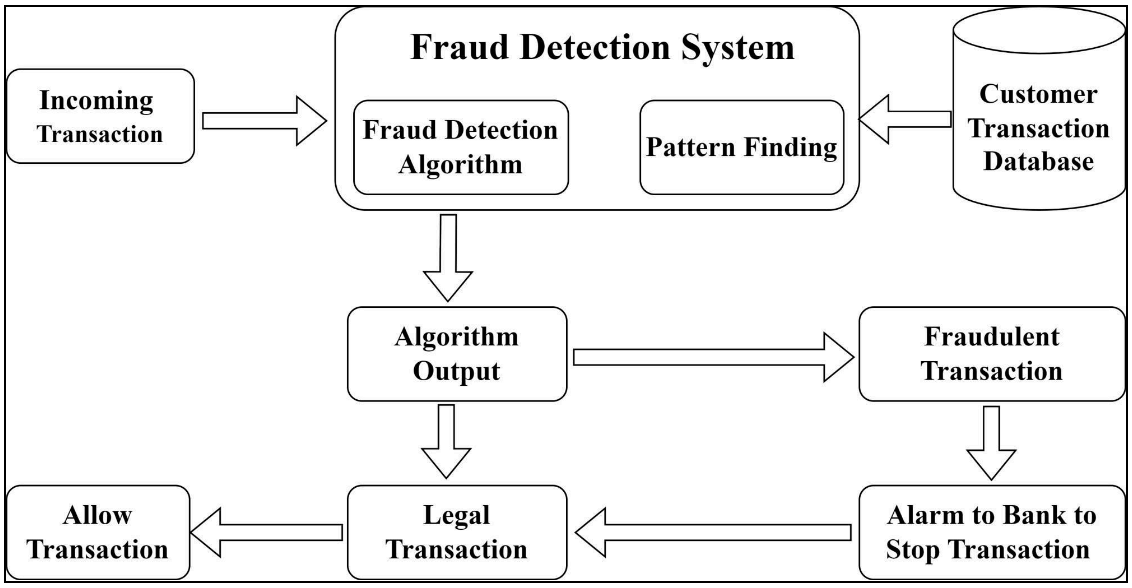


Fig 1.5 Flow chart for describing the fraud detection

Simulation Results

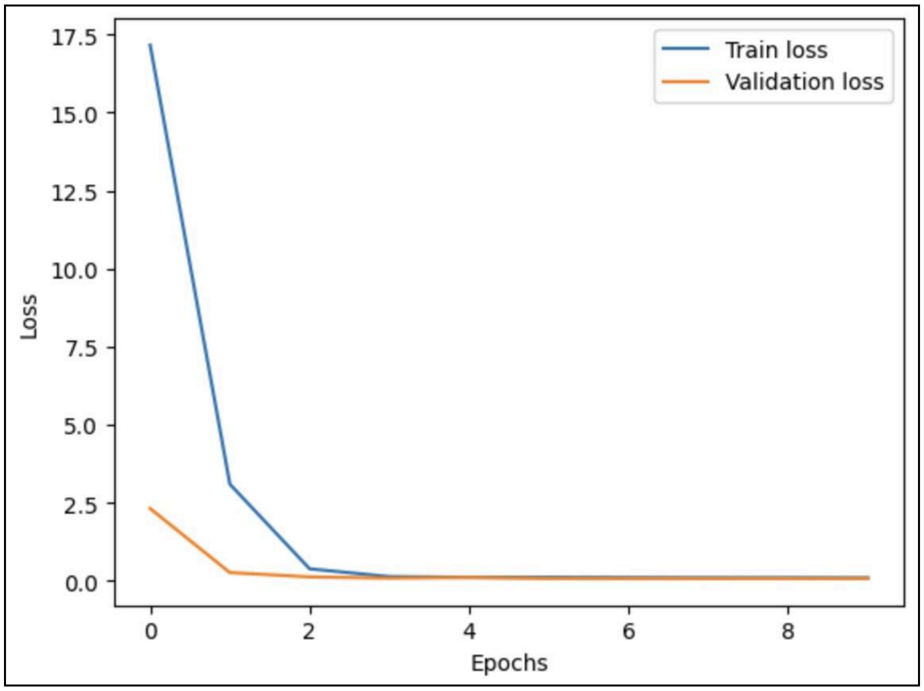


Fig 1.6 Epoch vs loss graph for training loss and validation loss

Overview

The plot presents the training and validation loss curves over 9 epochs. The training loss decreases rapidly in the initial epochs, then plateaus around epoch 2. The validation loss also decreases initially, but then starts to increase slightly from epoch 2 onwards.

Training Loss:

- The sharp decrease in the initial epochs indicates that the model is learning quickly and fitting the training data well.
- The plateauing around epoch 2 suggests that the model has reached a point of diminishing returns, where further training on the same data does not significantly improve its performance.

Validation Loss:

- The initial decrease in the validation loss indicates that the model is generalizing well to unseen data.
- The slight increase after epoch 2 suggests that the model might be overfitting to the training data. This means it is learning the noise and specific patterns in the training data, which might not be present in new, unseen data.

Potential Causes and Remedies:

- **Overfitting:** This could be due to the model being too complex for the dataset.
- **Regularization:** Techniques like L1/L2 regularization or dropout can help prevent overfitting.
    - **Early Stopping:** Stop training when the validation loss starts to increase.
    - **Data Augmentation:** Generate more training data by applying transformations to the existing data.
- **Underfitting:** This could be due to a model that is too simple or insufficient training.
    - **Increase Model Complexity:** Add more layers or neurons.
    - **Train Longer:** Increase the number of epochs.
    - **Feature Engineering:** Create more informative features from the existing data.

Additional Considerations:

- **Model Architecture:** The choice of the model architecture (e.g., neural network type, number of layers, number of neurons) can significantly impact the training and validation loss curves.
- **Hyperparameters:** The tuning of hyperparameters like learning rate, batch size, and optimizer can affect the training process.
- **Data Quality:** The quality of the training data, including data cleaning and preprocessing, is crucial for model performance.

References

1. Andrea Dal Pozzolo, et al. "Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy." *IEEE Transactions on Neural Networks and Learning Systems*, 2014.
2. Pumsirirat, A., & Yan, L. "Credit Card Fraud Detection using Deep Learning based on Auto-Encoder and Restricted Boltzmann Machine." *International Journal of Computer Applications*, 2018.
3. Aleskerov, E., Freisleben, B., & Rao, B. "CARDWATCH: A Neural Network Based Database Mining System for Credit Card Fraud Detection." *Proceedings of the IEEE/IAFE Conference on Computational Intelligence for Financial Engineering*, 1997.
4. Kaggle Credit Card Fraud Dataset: https://www.kaggle.com/mlg-ulb/creditcardfraud