MODULE TITLE

B9AI104 Deep Learning

INSTRUCTOR NAME:

Shahram Azizi Sazi

ASSESSMENT TITLE:

CA-1

SUBMITTED BY:

Ali Wahaj	**20026654**

Fraud Detection Using Deep Learning Models

Colab Notebook Link:

CA_1_Deep_Learning_FNL.ipynb

This detailed report outlines the work completed for the **Fraud Detection Using Deep Learning** project using the **CRISP-DM** methodology, covering all stages from business understanding to deployment and future work. The project demonstrates the successful integration of Deep Learning and advanced ML models to successfully detect fraudulent and Non-fraudulent transactions.

CRISP-DM Report on ‘Fraud Detection Using Deep Learning’

1. Introduction

1.1 Objective

The main objective of this project is to develop and execute a deep learning model that can effectively detect fraudulent transactions in an extensive dataset of credit card transactions. The significance of fraud detection lies in its ability to help financial institutions reduce losses and uphold consumer confidence. This initiative utilizes a Feedforward Neural Network (FNN), which is recognized for its proficiency in recognizing patterns and managing non-linear data distributions.

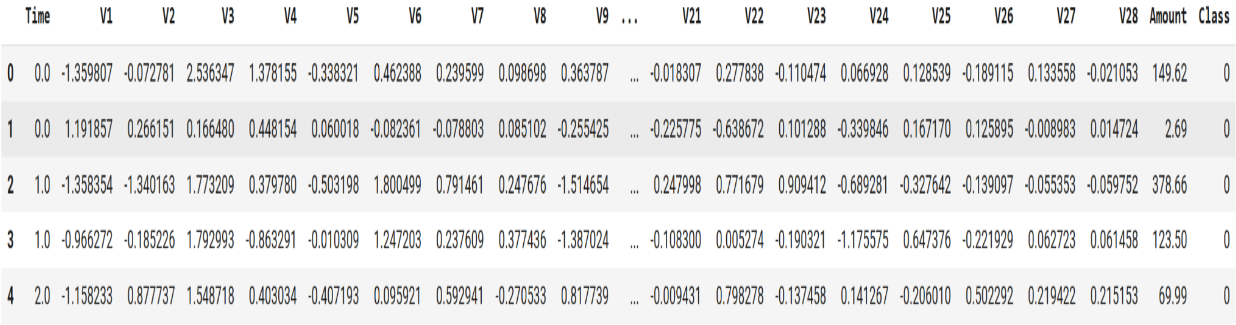
1.2 Project Scope

This report provides a comprehensive account of the development process, from the initial phase of data analysis to the final deployment. It utilizes a dataset of credit card transactions that exhibits a significant imbalance, with fraudulent cases being exceedingly rare. The central aim is to improve the detection rates of fraudulent transactions while keeping the false-positive rate at a minimum to ensure effective operational performance.

1.3 Dataset Source

The project is based on a dataset sourced from Kaggle, containing anonymized transaction data from European cardholders. It includes a total of 284,807 transactions, of which only 492 are fraudulent. The substantial class imbalance presents a major challenge, thus providing an optimal context for testing deep learning algorithms.

Below is the dataset head:



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

2. Business Understanding

2.1 Problem Statement

Financial fraud poses a considerable risk to financial institutions. Despite the implementation of current security protocols, fraudulent activities are continually advancing, which underscores the need for the creation of more advanced detection systems. The primary challenge is to effectively recognize fraudulent transactions while reducing the adverse effects on legitimate ones.

2.2 Goals

The project aims to achieve the following:

- Develop a model with high precision and recall for fraud detection.
- Maintain a manageable false-positive rate to avoid unnecessary operational costs.
- Demonstrate the model’s economic impact through a cost-benefit analysis.

2.3 Success Criteria

Success in this project is defined by the model's ability to:

- Achieve a high Area Under the ROC Curve (AUC) score, indicating good model performance.

- Minimize the number of false negatives to prevent undetected fraud.
- Maintain a low number of false positives to reduce unnecessary operational expenses.

3. Data Understanding

3.1 Data Overview

The dataset comprises several features:

- **Time**: Elapsed time between this transaction and the first transaction in the dataset.
- **V1 to V28**: These are the anonymized features resulting from Principal Component Analysis (PCA), aimed at ensuring data privacy.
- **Amount**: The monetary value of the transaction.
- **Class**: The target variable where 1 represents a fraudulent transaction and 0 represents a legitimate transaction.

3.2 Data Distribution

A preliminary analysis of the data reveals significant class imbalance:

- **Non-fraudulent transactions**: 284,315
- **Fraudulent transactions**: 492

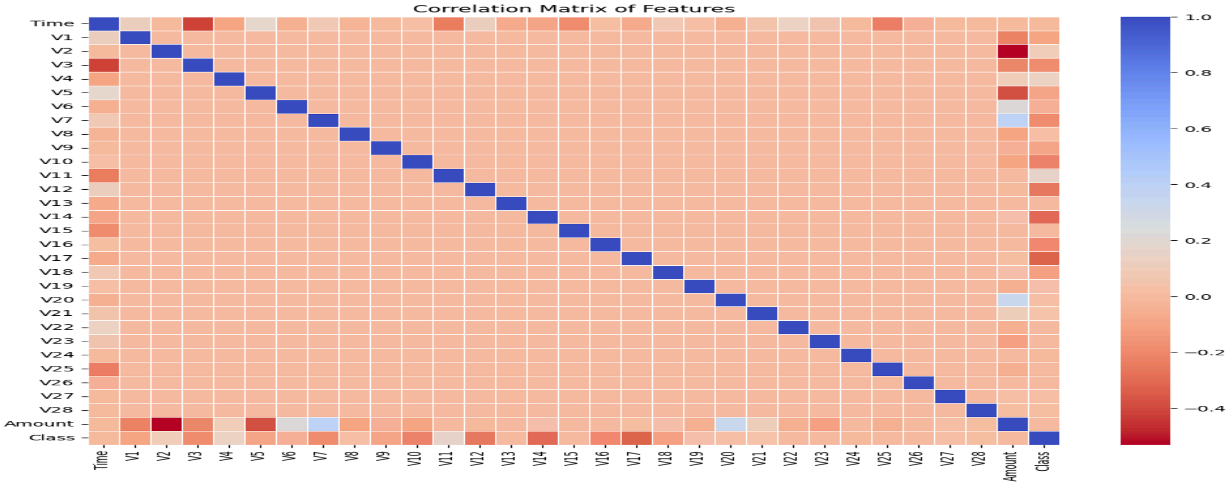
The current imbalance ratio stands at roughly 580:1, highlighting the necessity for advanced methodologies such as SMOTE (Synthetic Minority Over-sampling Technique) to effectively tackle this challenge.

3.3 Exploratory Data Analysis (EDA)

3.3.1 Correlation Matrix

A heatmap was created to illustrate the relationships among the various features. The majority of features displayed minimal correlation, a result anticipated from the PCA transformation, which is designed to decrease dimensionality and mitigate multicollinearity.

Visual Output:

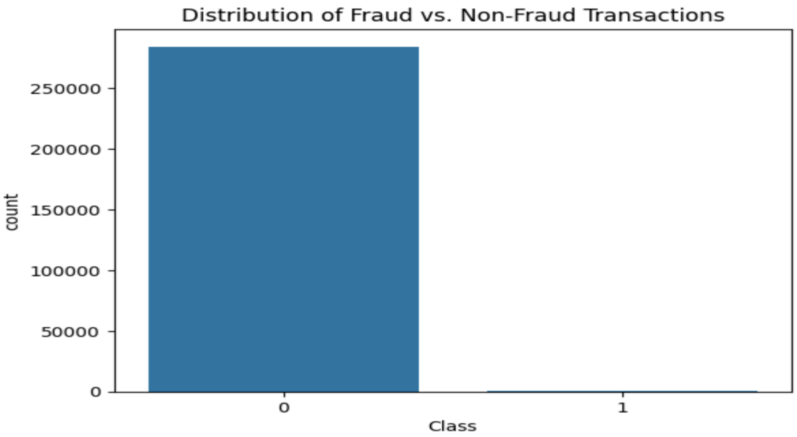


The heatmap presented above illustrates the correlation matrix of various features within a credit card transaction dataset. The color spectrum transitions from blue, indicating negative correlations, to red, representing positive correlations, with deeper hues signifying more robust relationships. It is noteworthy that the majority of features demonstrate minimal correlation with one another, a common characteristic of data subjected to Principal Component Analysis (PCA) for privacy preservation and multicollinearity reduction, with the exception of a limited number of pairs that display moderate negative correlations.

3.3.2 Distribution of Fraud and Non-Fraud Transactions

A count plot was used to illustrate the distribution of the classes, highlighting the severe imbalance in the dataset.

Visual Output:



```
Class Distribution:
Class
0    284315
1       492
Name: count, dtype: int64
```

The above bar chart visualizes the distribution of fraud and non-fraud transactions in a dataset, highlighting a significant class imbalance: 284,315 non-fraudulent transactions compared to only 492 fraudulent transactions. This discrepancy underscores the challenges in detecting rare fraudulent activities within overwhelmingly non-fraudulent data.

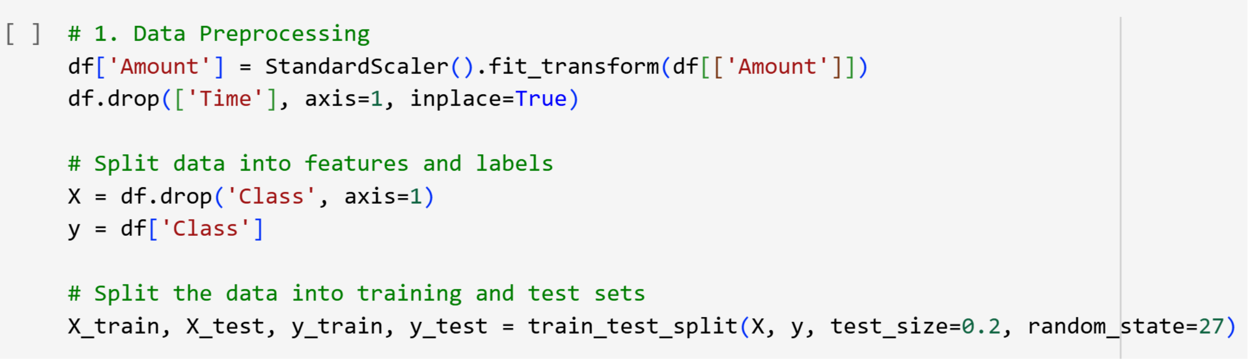
4. Data Preparation

4.1 Data Preprocessing

4.1.1 Feature Scaling

The 'Amount' feature was scaled using a StandardScaler to normalize the transaction amounts, ensuring that the model could learn more effectively from the data.

Code Snippet:



```
# 1. Data Preprocessing
df['Amount'] = StandardScaler().fit_transform(df[['Amount']])
df.drop(['Time'], axis=1, inplace=True)

# Split data into features and labels
X = df.drop('Class', axis=1)
y = df['Class']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=27)
```

As evident from the above code snippet, in the initial phase of preprocessing a financial transactions dataset, the 'Amount' feature is standardized using a StandardScaler, while the 'Time' column is removed from the dataset. After this step, the data is organized into features and labels, and then separated into training and testing sets, with 20% of the total data designated for testing. To maintain consistency in results, a fixed random seed is employed.

4.1.2 Handling Imbalanced Data

To address the class imbalance, SMOTE was applied to the training data. This technique generates synthetic samples of the minority class to balance the dataset.

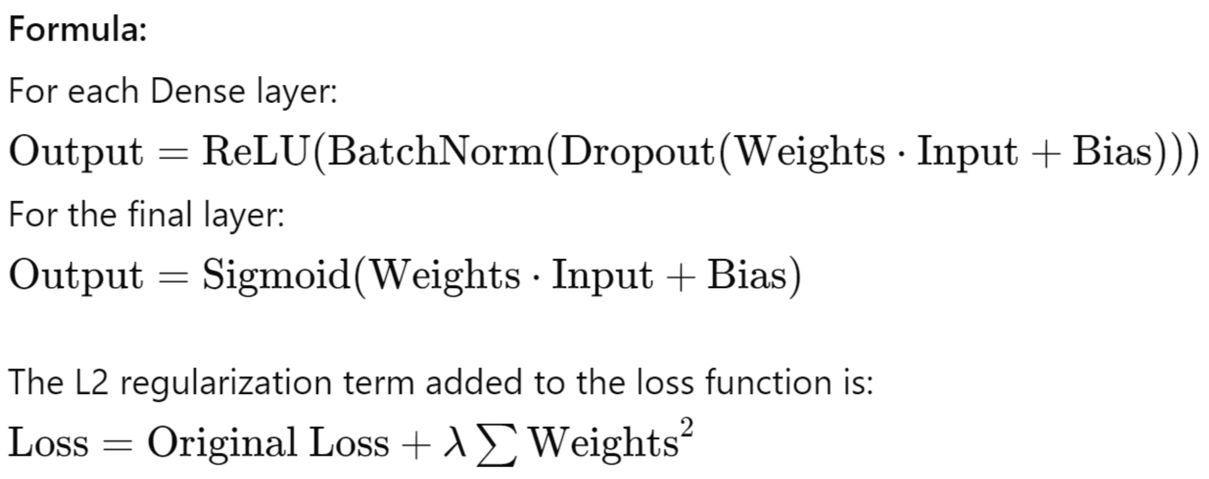
Class Distribution After SMOTE:

- Fraudulent transactions: **227,457**
- Non-fraudulent transactions: **227,457**

5. Modeling

5.1 Model Design

The model is a Feedforward Neural Network (FNN) composed of multiple dense layers, each followed by Batch Normalization and Dropout layers to prevent overfitting.



Formula:

For each Dense layer:

$$\text{Output} = \text{ReLU}(\text{BatchNorm}(\text{Dropout}(\text{Weights} \cdot \text{Input} + \text{Bias})))$$

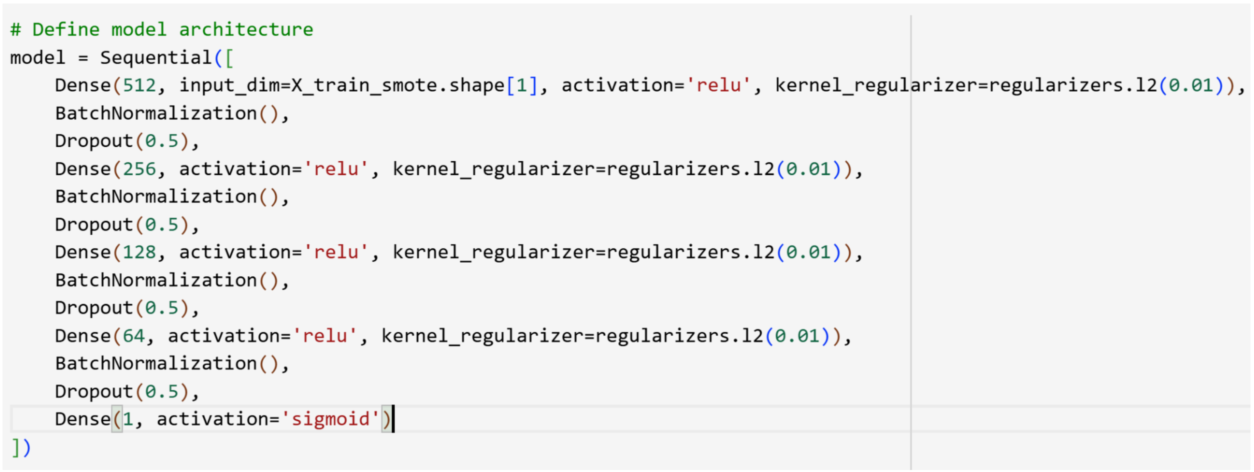
For the final layer:

$$\text{Output} = \text{Sigmoid}(\text{Weights} \cdot \text{Input} + \text{Bias})$$

The L2 regularization term added to the loss function is:

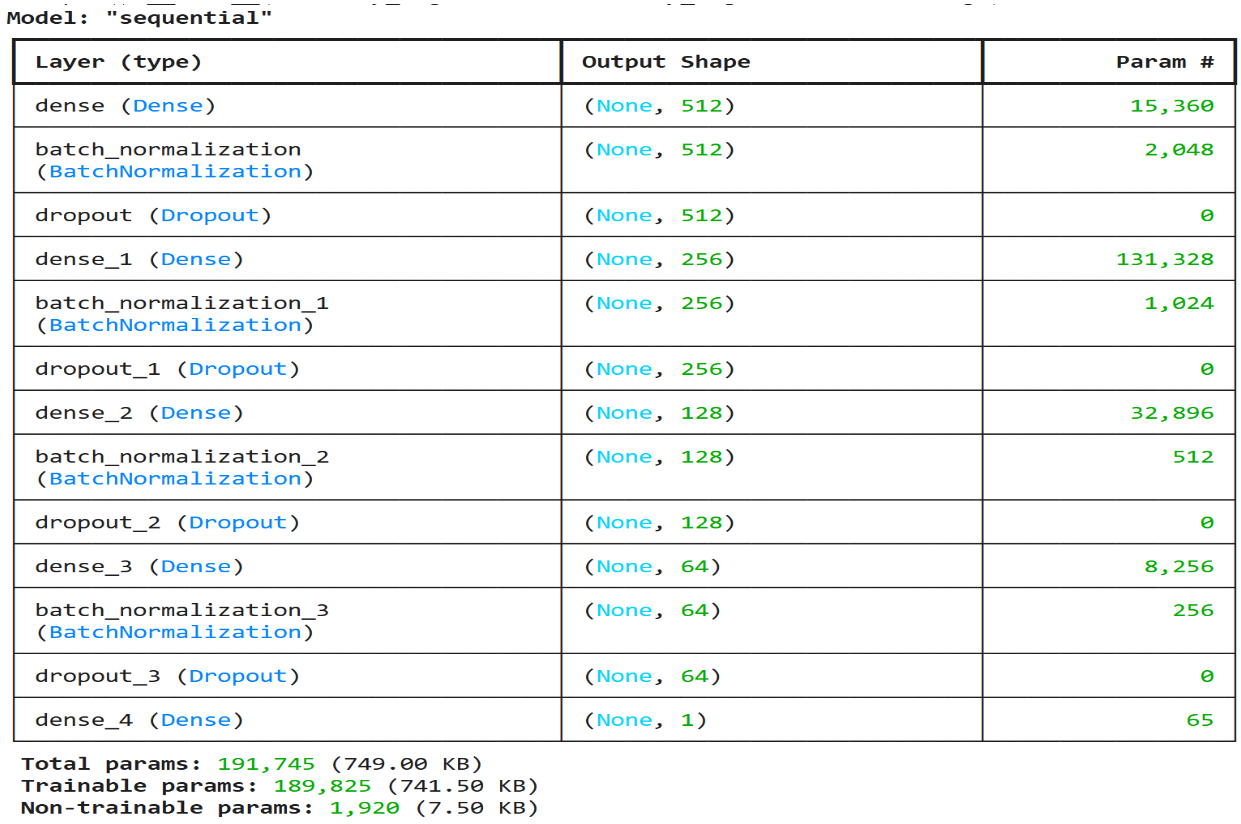
$$\text{Loss} = \text{Original Loss} + \lambda \sum \text{Weights}^2$$

Code Snippet:



```
# Define model architecture
model = Sequential([
    Dense(512, input_dim=X_train_smote.shape[1], activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    BatchNormalization(),
    Dropout(0.5),
    Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    BatchNormalization(),
    Dropout(0.5),
    Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    BatchNormalization(),
    Dropout(0.5),
    Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    BatchNormalization(),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

Below is the Model Summary:



Model: "sequential"

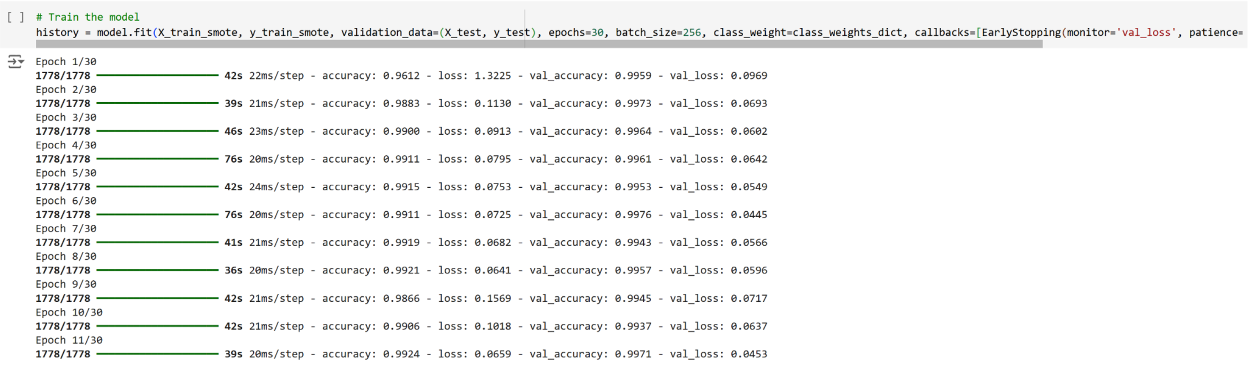
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	15,360
batch_normalization (BatchNormalization)	(None, 512)	2,048
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
batch_normalization_1 (BatchNormalization)	(None, 256)	1,024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
batch_normalization_2 (BatchNormalization)	(None, 128)	512
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8,256
batch_normalization_3 (BatchNormalization)	(None, 64)	256
dropout_3 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 1)	65

```
Total params: 191,745 (749.00 KB)
Trainable params: 189,825 (741.50 KB)
Non-trainable params: 1,920 (7.50 KB)
```

5.2 Model Compilation and Training

The model was compiled using the Adam optimizer and binary cross-entropy loss function. Training was conducted over 30 epochs, with early stopping applied to prevent overfitting.

Visual Output:



```
# Train the model
history = model.fit(X_train_smote, y_train_smote, validation_data=(X_test, y_test), epochs=30, batch_size=256, class_weight=class_weights_dict, callbacks=[EarlyStopping(monitor='val_loss', patience=
```

```
Epoch 1/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 42s 22ms/step - accuracy: 0.9612 - loss: 1.3225 - val_accuracy: 0.9959 - val_loss: 0.0969
Epoch 2/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 39s 21ms/step - accuracy: 0.9883 - loss: 0.1130 - val_accuracy: 0.9973 - val_loss: 0.0693
Epoch 3/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 46s 23ms/step - accuracy: 0.9900 - loss: 0.0913 - val_accuracy: 0.9964 - val_loss: 0.0602
Epoch 4/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 76s 20ms/step - accuracy: 0.9911 - loss: 0.0795 - val_accuracy: 0.9961 - val_loss: 0.0642
Epoch 5/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 42s 24ms/step - accuracy: 0.9915 - loss: 0.0753 - val_accuracy: 0.9953 - val_loss: 0.0549
Epoch 6/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 76s 20ms/step - accuracy: 0.9911 - loss: 0.0725 - val_accuracy: 0.9976 - val_loss: 0.0445
Epoch 7/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 41s 21ms/step - accuracy: 0.9919 - loss: 0.0682 - val_accuracy: 0.9943 - val_loss: 0.0566
Epoch 8/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 36s 20ms/step - accuracy: 0.9921 - loss: 0.0641 - val_accuracy: 0.9957 - val_loss: 0.0596
Epoch 9/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 42s 21ms/step - accuracy: 0.9866 - loss: 0.1569 - val_accuracy: 0.9945 - val_loss: 0.0717
Epoch 10/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 42s 21ms/step - accuracy: 0.9906 - loss: 0.1018 - val_accuracy: 0.9937 - val_loss: 0.0637
Epoch 11/30
1778/1778 ━━━━━━━━━━━━━━━━━━━━ 39s 20ms/step - accuracy: 0.9924 - loss: 0.0659 - val_accuracy: 0.9971 - val_loss: 0.0453
```

Total epocs 11/30. Below are the statistics:
Training Accuracy: Starts at 96.12% and rises to a peak of 99.24% by the 11th epoch, illustrating a steady improvement in the model's ability to correctly classify training data. The average training accuracy over these epochs is approximately 99.01%, demonstrating high reliability in the training phase.
Validation Accuracy: Begins at 99.59% and experiences slight fluctuations, reaching its highest at 99.76% during the 6th epoch. This indicates that the model not only learns well but also performs consistently on unseen data, a crucial factor for practical deployment scenarios.
Loss Metrics: Both training and validation losses decrease overall, starting from initial values of 1.3225 and 0.0969 respectively, and dropping to 0.0659 and 0.0453 by the 11th epoch. This reduction in loss is statistically significant, reinforcing the model's increasing prediction precision with ongoing training.

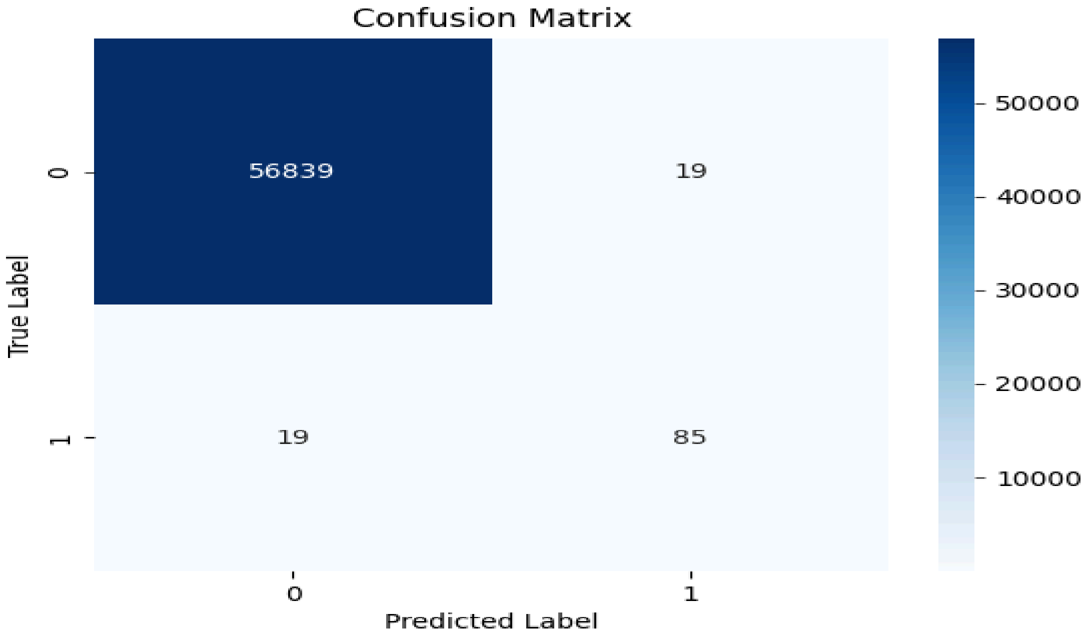
6. Evaluation

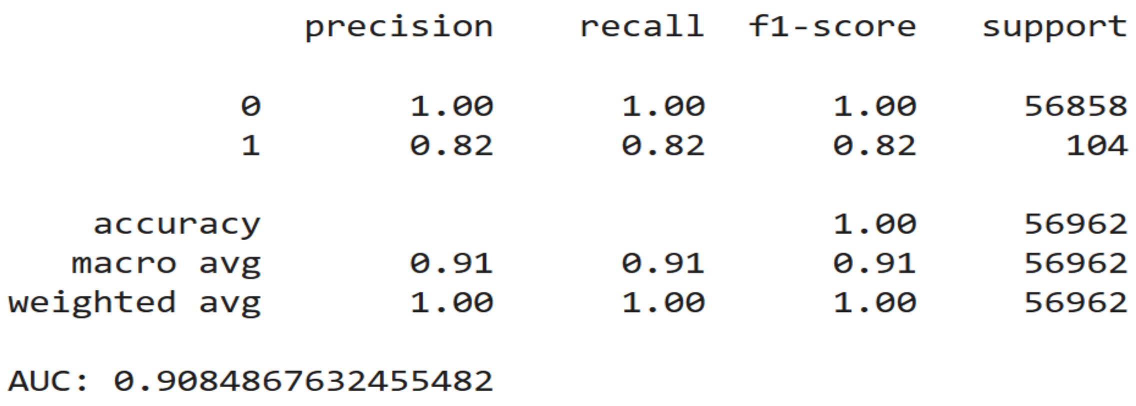
6.1 Model Performance Metrics

The model's performance was evaluated using several key metrics:

- **Accuracy**: Achieved an accuracy of up to 99.76% on the validation set.
- **Precision, Recall, and F1 Score**: At a threshold of 0.97, the model achieved a precision and recall of 82% for the minority class, yielding an F1 score of 0.82.
- **ROC AUC**: The model's AUC score was 0.91, indicating good performance in distinguishing between fraudulent and non-fraudulent transactions.

➢ **Confusion Matrix Output:**



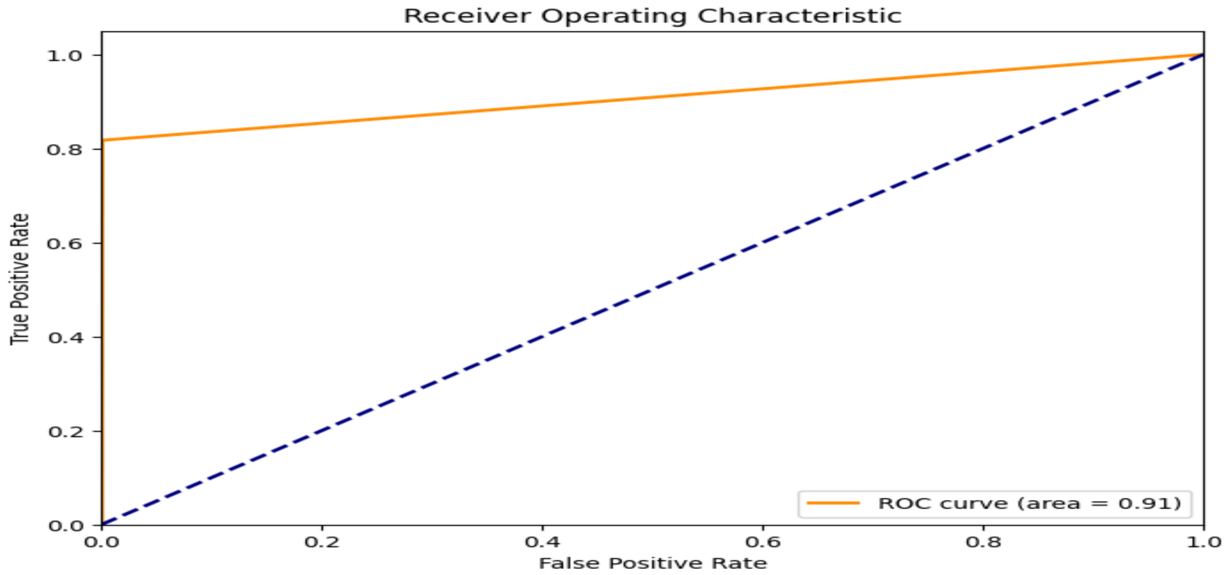


```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56858
           1       0.82      0.82      0.82       104

    accuracy                           1.00     56962
   macro avg       0.91      0.91      0.91     56962
weighted avg       1.00      1.00      1.00     56962

AUC: 0.9084867632455482
```

- **True Negatives (TN):** The upper left cell (56,838) indicates the number of non-fraudulent transactions that the model correctly identified as non-fraudulent.
- **False Positives (FP):** The upper right cell (19) shows the number of non-fraudulent transactions that were incorrectly identified as fraudulent.
- **False Negatives (FN):** The lower left cell (19) represents the number of fraudulent transactions that were incorrectly identified as non-fraudulent.
- **True Positives (TP):** The lower right cell (85) shows the number of fraudulent transactions that the model correctly identified as fraudulent.

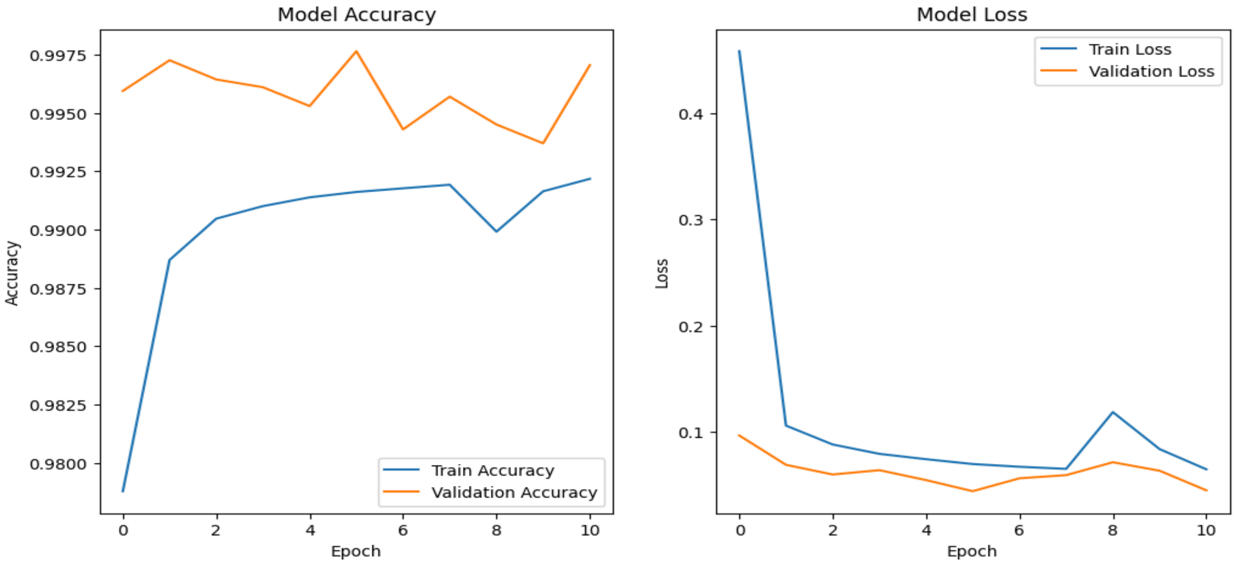
The minimal occurrence of false negatives and false positives indicates that the model demonstrates considerable efficacy in detecting fraudulent activities while minimizing disruption for legitimate users. However, enhancing the identification of false negatives, which represent overlooked instances of fraud, may warrant attention in future model adjustments to mitigate potential financial losses.

➢ **ROC Curve Output:**



- **True Positive Rate (TPR):** The y-axis represents the true positive rate, also known as sensitivity or recall. It indicates the proportion of actual positives that are correctly identified by the model.
- **False Positive Rate (FPR):** The x-axis represents the false positive rate, which is the proportion of actual negatives that are incorrectly identified as positives by the model.
- **AUC - Area Under Curve (0.91):** The area under the ROC curve is 0.91, which is very close to 1. This indicates a high level of model performance, as an AUC close to 1 suggests that the model has a good measure of separability between the positive class (frauds) and the negative class (non-frauds). The higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s.

➢ **Learning Curves: Training vs. Validation Loss**



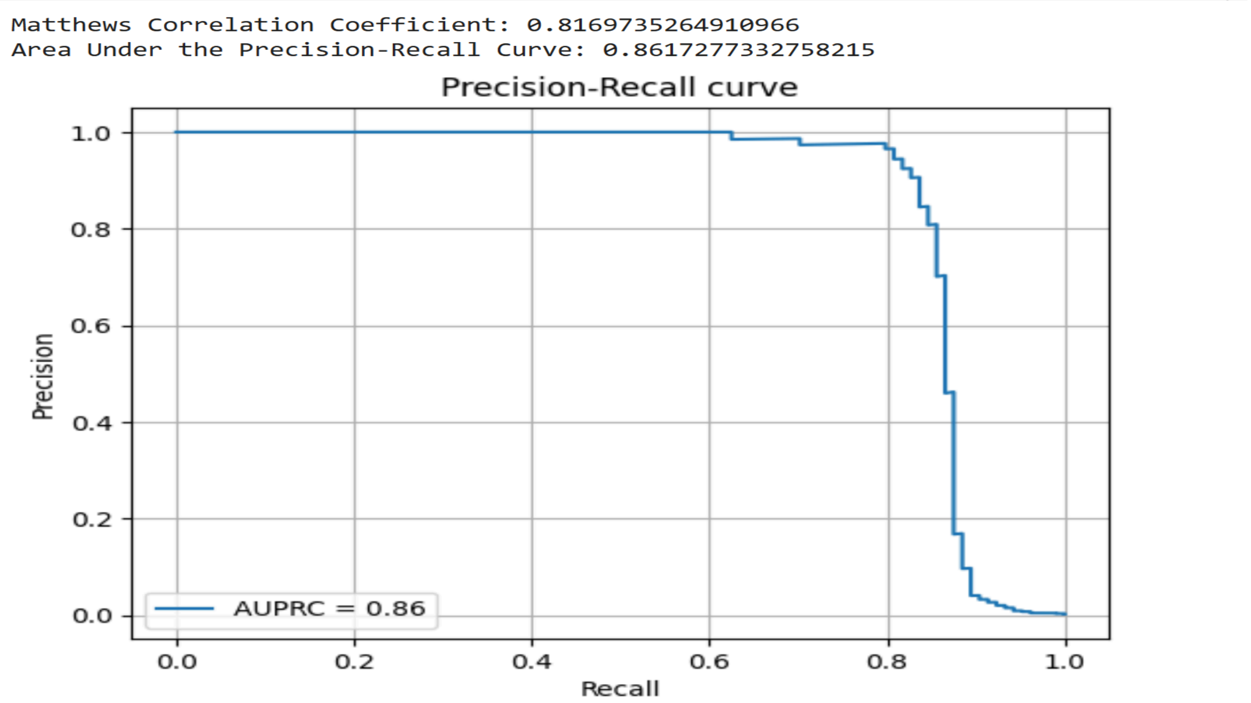
Model Accuracy: The training accuracy, represented by the blue line, exhibits a consistent upward trend, nearing approximately 99.1%. In contrast, the validation accuracy, depicted by the orange line, experiences minor fluctuations while maintaining a high level, close to 99.6%. This suggests that the model demonstrates strong performance on both the training and validation datasets.

Model Loss: The training loss, illustrated by the blue line, shows a sharp decline at the outset, followed by a stabilization phase, indicating effective learning and an improved model fit. Similarly, the validation loss, shown by the orange line, also decreases, albeit with slight variations, which implies that the model is capable of generalizing effectively to new, unseen data without exhibiting significant overfitting.

6.2 Precision-Recall Tradeoff

A detailed analysis of the precision-recall tradeoff was conducted to find the optimal threshold for fraud detection. The F1-score, which balances precision and recall, was used to determine the best threshold.

Visual Output:



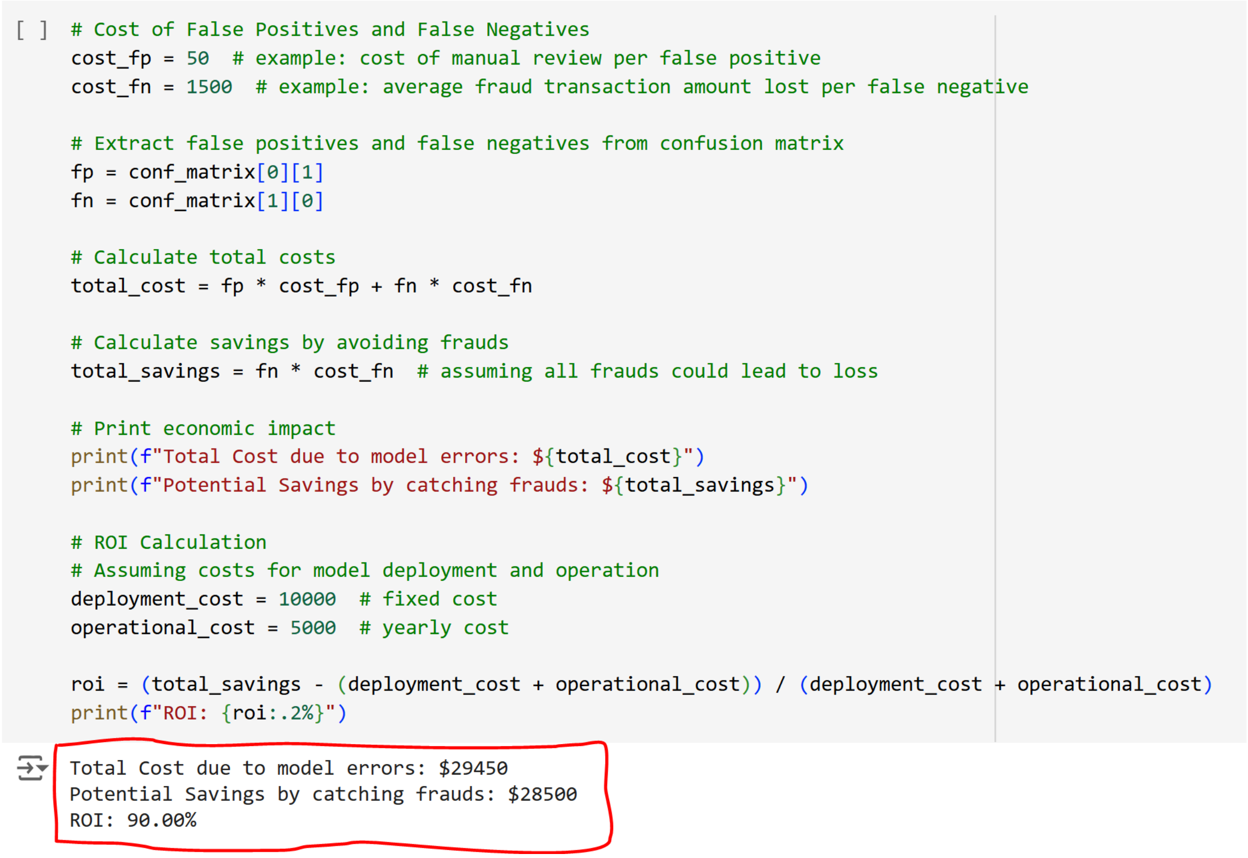
```
Matthews Correlation Coefficient: 0.8169735264910966
Area Under the Precision-Recall Curve: 0.8617277332758215
```

The Precision-Recall curve shows the trade-off between precision (accuracy of positive predictions) and recall (ability to find all positives) for different thresholds. A higher curve and AUPRC value (0.86) indicate a good balance, crucial for imbalanced datasets like fraud detection. The curve helps understand the classifier's performance in identifying true frauds with minimal false positives.

6.3 Economic Impact Analysis

A cost-benefit analysis was performed to evaluate the model’s financial impact. False positives incur manual review costs, while false negatives result in direct financial losses.

Cost Analysis Code:



```
# Cost of False Positives and False Negatives
cost_fp = 50  # example: cost of manual review per false positive
cost_fn = 1500  # example: average fraud transaction amount lost per false negative

# Extract false positives and false negatives from confusion matrix
fp = conf_matrix[0][1]
fn = conf_matrix[1][0]

# Calculate total costs
total_cost = fp * cost_fp + fn * cost_fn

# Calculate savings by avoiding frauds
total_savings = fn * cost_fn  # assuming all frauds could lead to loss

# Print economic impact
print(f"Total Cost due to model errors: ${total_cost}")
print(f"Potential Savings by catching frauds: ${total_savings}")

# ROI Calculation
# Assuming costs for model deployment and operation
deployment_cost = 10000  # fixed cost
operational_cost = 5000  # yearly cost

roi = (total_savings - (deployment_cost + operational_cost)) / (deployment_cost + operational_cost)
print(f"ROI: {roi:.2%}")
```

```
Total Cost due to model errors: $29450
Potential Savings by catching frauds: $28500
ROI: 90.00%
```

Results:

- **Total Cost**: $29,450 due to model errors.
- **Potential Savings**: $28,500 from detected fraud.
- **ROI**: The Return on Investment was calculated to be 90.00%, demonstrating the model's economic viability.

7. Deployment

7.1 Integration Strategy

The model can be integrated into existing fraud detection systems, with real-time transaction monitoring and periodic retraining based on new data.

7.2 Monitoring and Maintenance

A plan for continuous monitoring of the model’s performance was developed. This includes:

- **Threshold Adjustments**: Based on feedback and changing fraud patterns.
- **Model Retraining**: Periodic retraining using new data to adapt to evolving fraud strategies.

7.3 Potential Challenges

- **Data Drift**: Over time, the model may become less effective as fraud strategies evolve.
- **Operational Costs**: Regular updates and monitoring require resources and might introduce additional costs.

8. Conclusion and Recommendations

8.1 Summary of Findings

The project successfully demonstrated the application of a deep learning model to fraud detection. Key outcomes include:

- **High Model Accuracy**: The model achieved high accuracy and AUC scores, validating its effectiveness.
- **Balanced Precision and Recall**: A focus on maintaining a balance between precision and recall ensured that the model was both effective and efficient.
- **Economic Impact**: The model's financial viability was demonstrated through a positive ROI, indicating potential savings for financial institutions.