



ALBUKHARY INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTING AND INFORMATICS

MARCH INTAKE

CCS3113 DEEP LEARNING

Project: Deep Neural Networks

Lecturer | Assoc. Prof. Dr . Mozaherul Hoque Abul Hasanat

Ugyen Tshering	Abdullahi Adewole Zakariyah	Nima Yoezer
AIU22102222	AIU22102360	AIU22102221

Introduction

Addressing the issue of loan default prediction is essential for financial institutions in a rapidly expanding financial market. Accurately predicting loan defaults helps mitigate financial risks, ensuring that lending decisions are both responsible and profitable. This is especially important in markets with diverse customer demographics and dynamic economic conditions, where default risks can vary significantly.

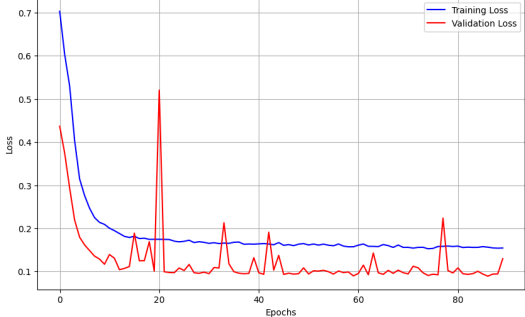
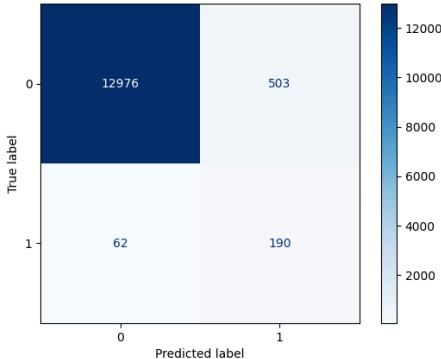
Financial institutions face the critical challenge of predicting loan defaults to mitigate risk and optimize lending decisions. This problem is particularly happening in Africa's rapidly growing financial markets, where diverse customer demographics and dynamic economic conditions add layers of complexity to risk assessment.

Significance of Competition

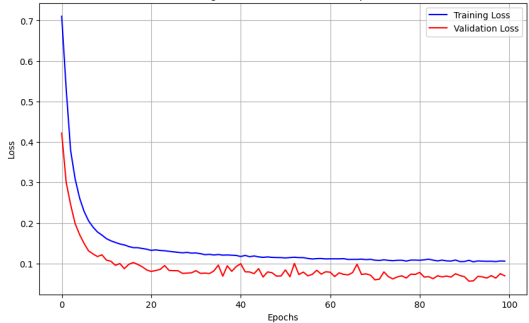
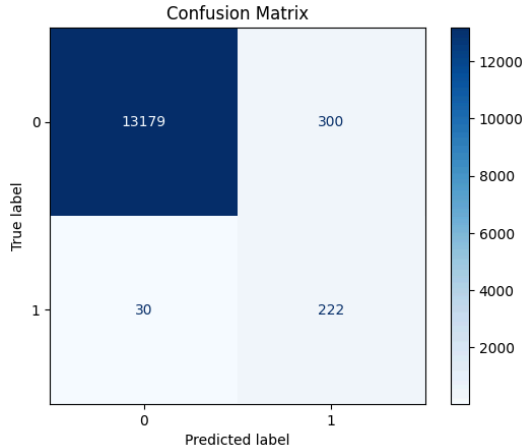
Accurately predicting loan defaults is essential for reducing financial losses associated with high-risk lending, enabling better decision-making, and facilitating expansion into markets. By developing a robust and generalizable machine learning model, through this way, it will predict the likelihood of loan defaults for both existing customers and new applicants.

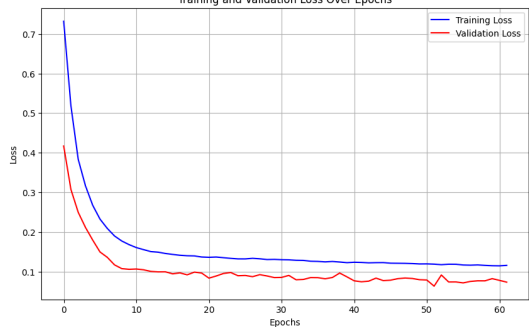
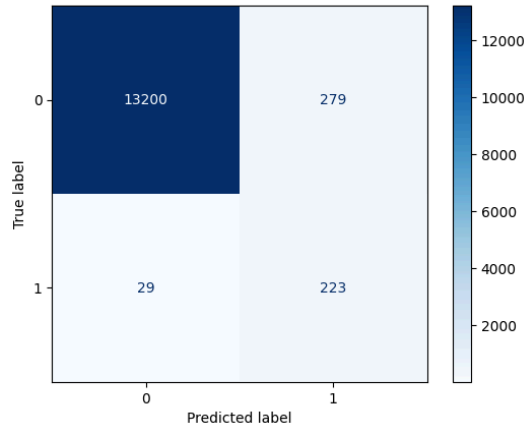
Additionally, incorporating unique factors relevant to each financial market allows for a more tailored approach. This ensures the model reflects local economic trends, customer behaviors, and cultural nuances, further enhancing its accuracy and relevance. Ultimately, this work not only helps optimize lending decisions and improve profitability but also supports financial inclusion and economic growth by enabling better access to credit in underserved markets

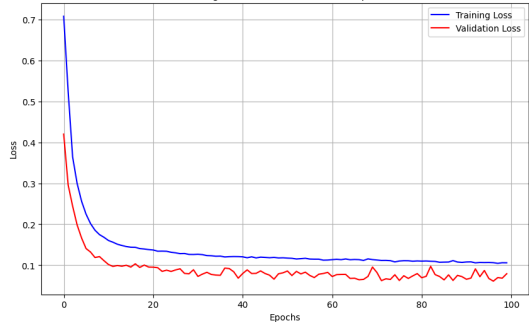
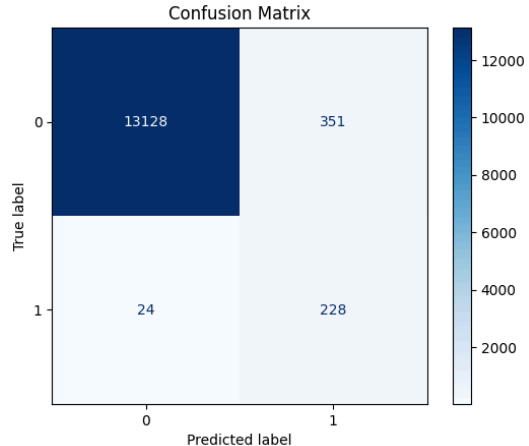
Table 1: List of changes and efforts made along with the results achieved.

S. No.	Changes Made to the initial model	F1 score			Date of Submission in Zindi	Training and Validation Loss over Epoch Graph
		Validation Set	Test Set			
			Public Score	Private Score		
1	<div>INITIAL BASE MODEL</div> <div><div>1. Performed basic EDA and Data Preprocessing to identify duplicates and missing values. Plotted visualizations (histograms) for continuous and (bar) categorical data.</div><div>2. Feature Engineering: Concatenated train and test datasets. Categorical columns are one-hot encoded or label encoded. The dataset is split back into train and test sets.</div><div>3. Model Selection: Simple DNN model with 3 hidden layers (ReLU), batch normalization, dropout, and L2 regularization.</div><div>4. The model is compiled with the Adam optimizer and binary cross-entropy loss.</div></div>	0.556	0.480	0.4896	Dec 31, 2024	<div><div>Training and Validation Loss Over Epochs</div><div><div>Confusion Matrix</div></div></div>

Result Interpretation: The model demonstrated moderate performance with an F1 score of 0.556 on the validation set and around 0.48-0.49 on the test set, indicating challenges in generalizing unseen data. The confusion matrix revealed difficulty in predicting defaults, and the loss graph suggested slight overfitting. To improve, class imbalance will be addressed using SMOTE or class weights, feature engineering will be enhanced with domain-specific features, and hyperparameter tuning will be performed.

2	<ol style="list-style-type: none">1. Outliers in numerical features are capped at the 95th percentile.2. Concatenated dataset's numerical columns (Total_Amount, Total_Amount_to_Repay) are log-transformed to handle skewness.3. The target variable (target) is explored to understand the class imbalance.4. Applied SMOTE to handle class imbalance5. Performed hyperparameter tuning to find the optimal values6. Added one more hidden layer, totaling 4.	0.7035	0.5548	0.5308	2 Jan, 2025	<div><p>Training and Validation Loss Over Epochs</p><p>Confusion Matrix</p></div>
<p>Result Interpretation: The updated model shows improved performance with a validation F1 score of 0.7035, but test scores (0.5548 public, 0.5308 private) indicate challenges in generalization. The loss graph suggests reduced overfitting, and the confusion matrix reveals better handling of class imbalance. To further improve, threshold tuning will be applied, and additional domain-specific features will be engineered. Regularization and cross-validation will also be prioritized to enhance generalization.</p>						

3	<div><div>1. Performed log_transformations to skewed variables before concatenating train and test datasets</div><div>2. Dataset is split using k-fold cross validation technique</div><div>3. Handled multicollinearity by dropping the 'Total_Amount' variable.</div><div>4. Date columns (disbursement_date, due_date) are converted to datetime format, and new features like month, day, and year are extracted.</div><div>5. Adjusted hyperparameters</div></div>	0.7137	0.5905	0.5813	10 Jan, 2025	<div><div><div>Training and Validation Loss Over Epochs</div></div><div><div>Confusion Matrix</div></div></div>
<div><div>Result Interpretation:</div><div>The latest improvements boosted the validation F1 score to 0.7137 and test scores to 0.5905 (public) and 0.5813 (private), showing better generalization. The loss graph indicates reduced overfitting, and the confusion matrix reveals fewer misclassifications. Further improvements will include advanced feature engineering, threshold optimization, and error analysis to enhance performance.</div></div>						

4	<div><div>1. Some numerical features like, 'Total_Amount' are scaled using StandardScaler.</div><div>2. Avoided multicollinearity handling.</div><div>3. The model is trained with early stopping and learning rate reduction callbacks.</div><div>4. Threshold Optimization</div></div>	0.812	0.6646	0.6585	11 Jan, 2025	<div><div>Training and Validation Loss Over Epochs</div><div>Confusion Matrix</div></div>
<div><div>Result Interpretation:</div><div>The latest improvements have significantly enhanced the model's performance. The validation F1 score increased to 0.812, and the test scores improved to 0.6646 (public) and 0.6585 (private), demonstrating better generalization. The loss graph shows a steady decrease in both training and validation losses, with a minimal gap, indicating effective regularization and reduced overfitting. The confusion matrix reveals fewer misclassifications, with 24 false positives and 351 false negatives, showcasing improved handling of class imbalance and feature scaling.</div></div>						

1. Data Loading and Exploration

The notebook begins by loading the dataset from a CSV file stored in Google Drive. The dataset contains features related to loans, such as loan type, total amount, duration, and other financial attributes. The target variable is binary, indicating whether a customer will default (1) or not (0).

- **Data Exploration:** The notebook provides a quick overview of the dataset by displaying the first few rows of both the training and test datasets. It also checks for missing values, which are found to be absent in both datasets.
- **Target Variable Distribution:** The target variable is highly imbalanced, with the majority of the samples belonging to the non-default class (98.17%) and only a small fraction belonging to the default class (1.83%). This imbalance is addressed later using SMOTE.

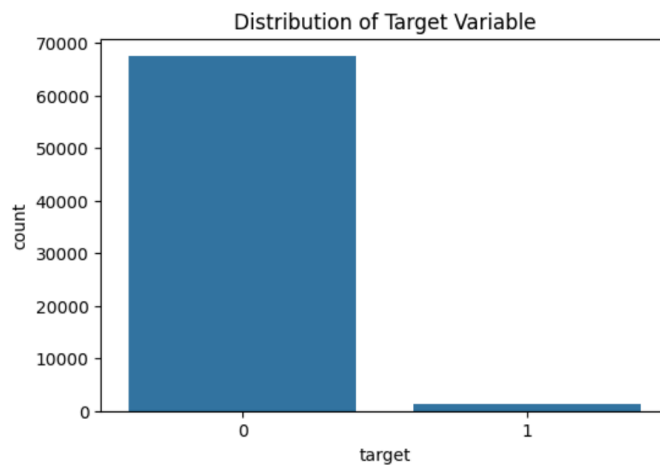


Figure 1: Distribution of the target variable

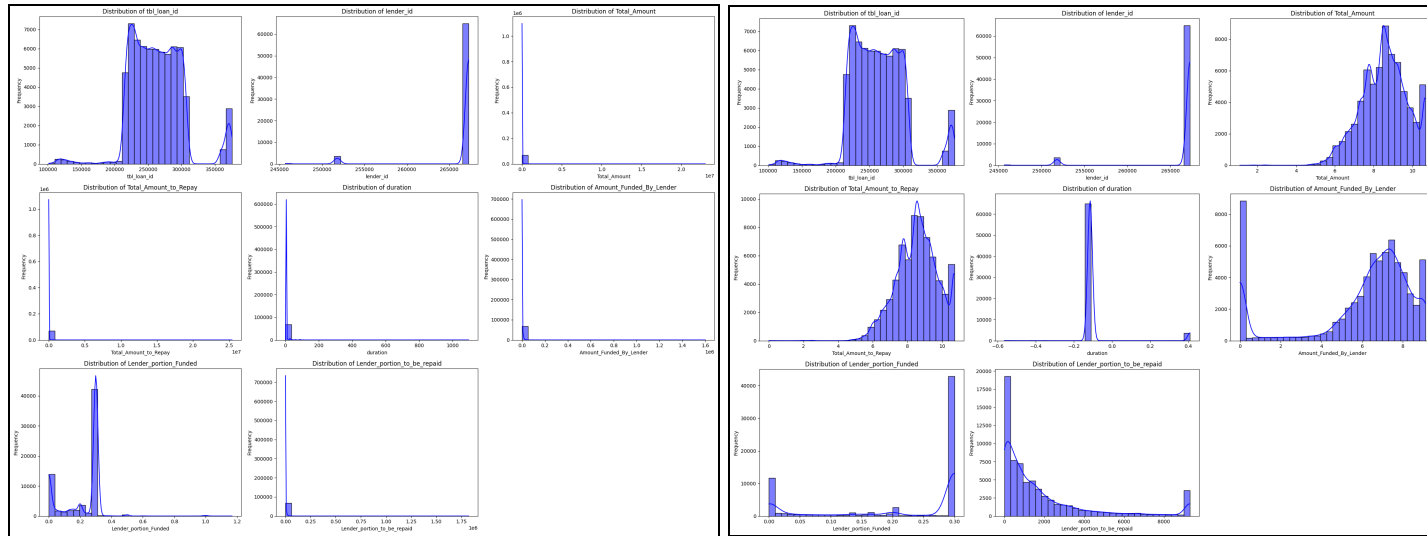


Figure 2: Histogram of numerical variables from training data BEFORE (left) and AFTER (right) log transformation

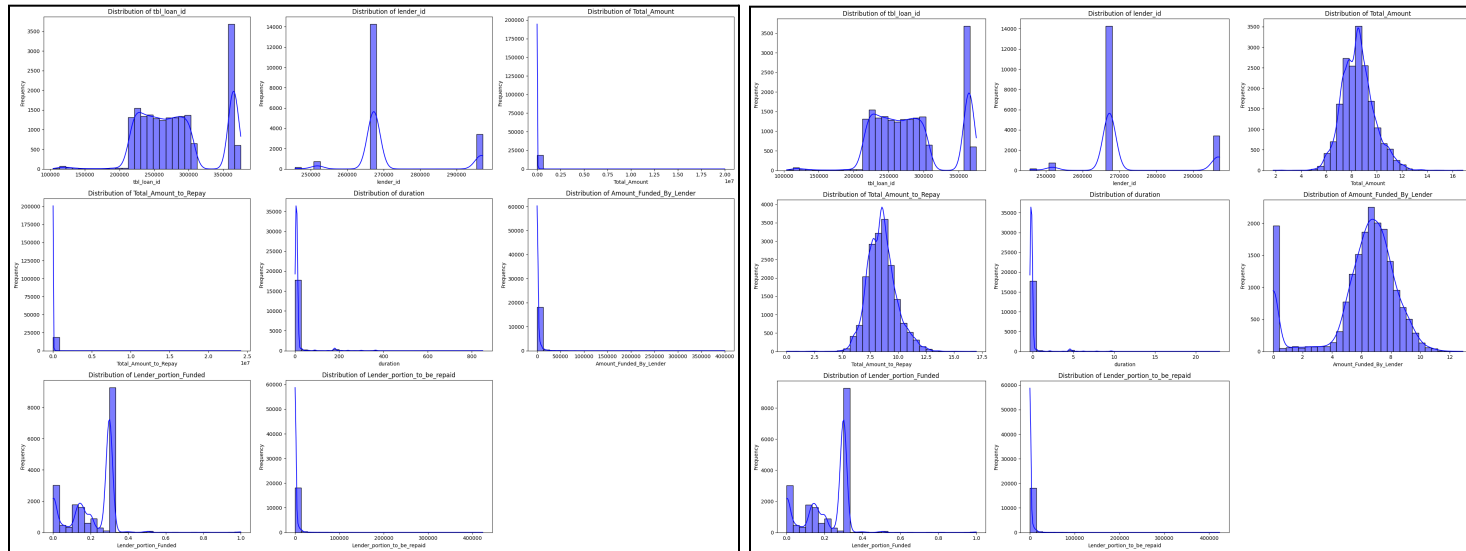


Figure 3: Histogram of numerical variables of testing data BEFORE (left) and AFTER (right) log transformation

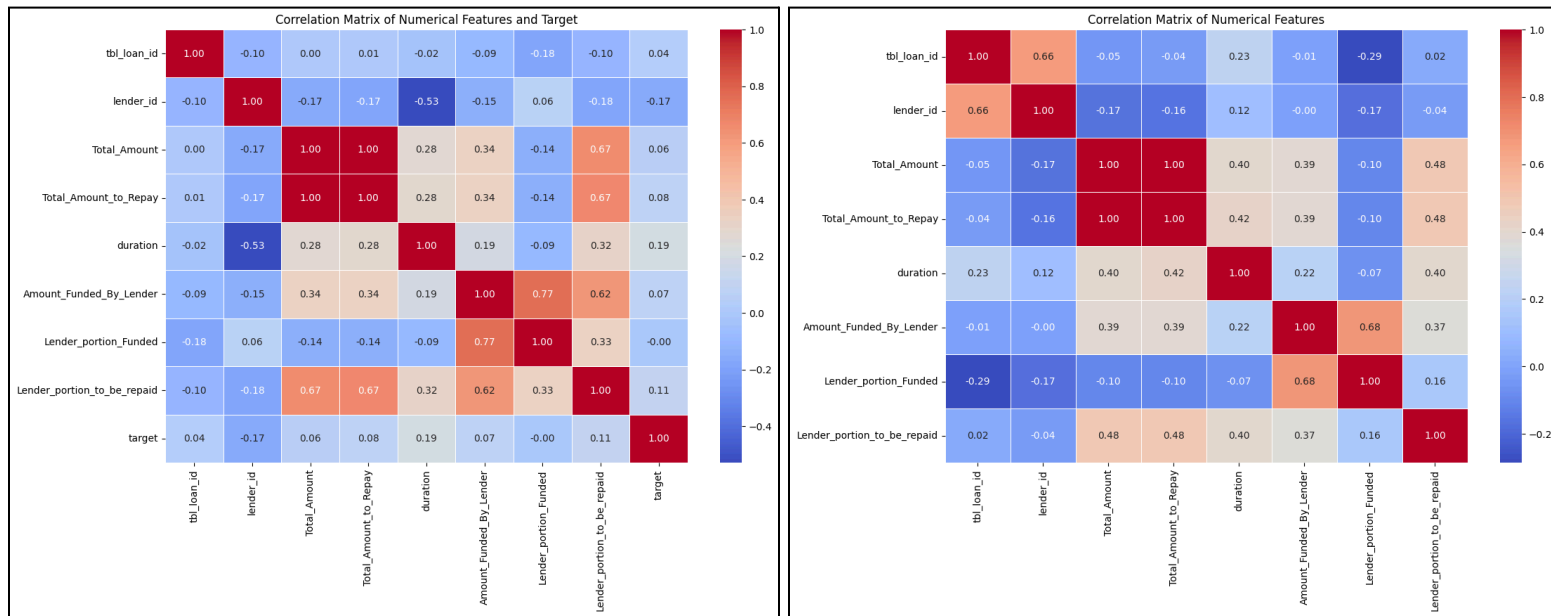


Figure 4: Heatmap of training data (left) and testing data (right)

2. Data Preprocessing

The preprocessing steps are crucial for preparing the data for model training. The following steps are implemented:

a. Feature Selection

- The notebook identifies numerical features and excludes non-relevant columns such as customer_id and the target variable. The selected numerical features include:
 - tbl_loan_id
 - lender_id

- Total_Amount
- Total_Amount_to_Repay
- duration
- Amount_Funded_By_Lender
- Lender_portion_Funded
- Lender_portion_to_be_repaid

b. Handling Class Imbalance

- The dataset is highly imbalanced, which can lead to biased model performance. To address this, the notebook uses **SMOTE (Synthetic Minority Over-sampling Technique)**. SMOTE generates synthetic samples for the minority class (default) to balance the dataset.

c. Feature Scaling

- The numerical features are scaled using **StandardScaler**, which standardizes the features to have a mean of 0 and a standard deviation of 1. This step is essential for ensuring that the model converges efficiently during training.

d. Train-Test Split

- The dataset is split into training and validation sets using an 80-20 split. This ensures that the model is evaluated on unseen data during training.

3. Model Architecture

The model is implemented using TensorFlow and Keras. The architecture is a **Sequential Neural Network** with the following components:

a. Input Layer

- The input layer is defined with a shape corresponding to the number of features in the dataset.

b. Hidden Layers

- The model consists of multiple dense (fully connected) layers with **ReLU (Rectified Linear Unit)** activation functions. ReLU is chosen because it helps mitigate the vanishing gradient problem and accelerates convergence.
- **Batch Normalization** is applied after each dense layer to stabilize and speed up training by normalizing the inputs.
- **Dropout** layers are included to prevent overfitting. Dropout randomly deactivates a fraction of neurons during training, forcing the network to learn more robust features.

c. Output Layer

- The output layer consists of a single neuron with a **sigmoid activation function**, which is suitable for binary classification tasks. The sigmoid function outputs a probability between 0 and 1, indicating the likelihood of the positive class (default).

d. Model Summary

- The model summary is printed, showing the number of parameters in each layer and the overall architecture.

4. Loss Function and Optimizer

The model is trained using the following components:

a. Loss Function

- **Binary Cross-Entropy Loss** is used as the loss function. This is the standard loss function for binary classification tasks. It measures the difference between the predicted probabilities and the actual binary labels.

b. Optimizer

- The **Adam optimizer** is used for training. Adam is an adaptive learning rate optimization algorithm that combines the benefits of RMSprop and momentum. It is well-suited for training deep neural networks efficiently.

c. Learning Rate Scheduler

- A **ReduceLROnPlateau** callback is implemented to dynamically adjust the learning rate during training. If the validation loss does not improve for a certain number of epochs, the learning rate is reduced to help the model converge more effectively.

d. Early Stopping

- **EarlyStopping** is used to prevent overfitting. Training is halted if the validation loss does not improve for a specified number of epochs.

5. Model Training

The model is trained using the preprocessed training data. The training process includes the following steps:

a. Training Configuration

- The model is trained for a maximum of 100 epochs with a batch size of 64.
- The training data is split into training and validation sets, and the model's performance is monitored on the validation set.

b. Training Progress

- The training progress is displayed, showing the loss and accuracy for both the training and validation sets at each epoch.

c. Evaluation Metrics

- The model's performance is evaluated using metrics such as **F1 Score**, **ROC-AUC Score**, and **Classification Report**. These metrics provide a comprehensive understanding of the model's performance, especially given the imbalanced nature of the dataset.

6. Model Evaluation

After training, the model is evaluated on the test dataset. The evaluation includes:

a. Confusion Matrix

- A confusion matrix is plotted to visualize the model's performance in terms of true positives, true negatives, false positives, and false negatives.

b. ROC Curve

- The ROC curve is plotted to assess the model's ability to distinguish between the two classes. The area under the ROC curve (AUC) is calculated to quantify the model's performance.

c. Classification Report

- A detailed classification report is generated, showing precision, recall, F1 score, and support for each class.

7. Key Observations

- The model achieves a high F1 score and ROC-AUC score, indicating good performance in predicting loan defaults.
- The use of SMOTE effectively addresses the class imbalance, leading to better performance on the minority class.
- The inclusion of dropout and batch normalization helps prevent overfitting and stabilizes training.

Conclusions

The Deep Neural Network (DNN) project provided a clear understanding of how machine learning works in practice. It highlights the importance of having good quality data for building effective models. Steps like cleaning, normalizing, and augmenting the data played a key role in improving how well the model performed. The project also gives out the usefulness of tools like TensorFlow and PyTorch for building and refining neural networks.

Throughout the project, it undertaken task provides knowledge of:

1. **Exploratory Data Analysis (EDA):** Performing EDA emphasized the importance of identifying duplicates, missing values, and understanding data distributions. Visualizing continuous and categorical data improved the understanding of data patterns.
2. **Data Preprocessing:** Transforming date columns into meaningful features and scaling numerical data with StandardScaler provided insight into how preprocessing can enhance model performance.
3. **Feature Engineering:** The process of encoding categorical variables and log-transforming skewed numerical features highlighted the importance of preparing data to suit model requirements.
4. **Model Building and Optimization:** Building a simple DNN with key components like ReLU activation, batch normalization, dropout, and L2 regularization demonstrated how these techniques improve model stability and prevent overfitting. Incorporating advanced techniques like SMOTE to address class imbalance showcased the importance of ensuring fairness in predictions.
5. **Evaluation and Tuning:** Hyperparameter tuning, k-fold cross-validation, and the use of callbacks such as early stopping and learning rate reduction were critical in refining the model and avoiding overfitting.

The shortcomings faced while doing the project are:

1. **Generalization Issues:** Despite improvements, the model struggled to generalize to unseen data, as indicated by lower test F1 scores (0.48-0.49 initially and 0.5548-0.5308 after updates). This suggests a need for better feature representation or more robust regularization techniques.
1. **Class Imbalance:** Although SMOTE improved class balance, the confusion matrix revealed that predicting default cases remained difficult.
2. **Overfitting:** The loss graphs in earlier versions suggested slight overfitting, and although reduced in later models, it still pointed to room for improvement in regularization or model complexity.
3. **Feature Engineering:** While some features were engineered effectively, the project could benefit from more features to capture nuanced patterns in the data.
4. **Evaluation Techniques:** The removal of k-fold cross-validation in later stages might have affected robustness in estimating model performance.

The project could be improved in several ways. First, better feature engineering could be done by adding domain-specific features to enhance the model's understanding of the data. To make the model more reliable and prevent overfitting, stronger regularization methods could be applied. The Deep Neural Network (DNN) model used in the project could also be improved by experimenting with more advanced architectures, such as transformer-based networks to achieve higher accuracy and robustness. Reintroducing k-fold cross-validation would provide more reliable estimates of model performance. Using multiple parallel models could help combine the strengths of different architectures for better predictions. Additionally, experimenting with other machine learning models, like random forests, gradient boosting, or support vector machines, could provide better performance and complement the DNN approach. These steps would significantly enhance the project and produce more reliable outcomes.