American Express - Default Prediction

1st Hoi Ching Cheung (hc792) 2nd Jinzhao Kang (jk2575) 3rd Joyce Chen (jc2888) 4th Xianglin Chen (xc352)
Deep learning *Deep learning* *Deep learning* *Deep learning*
Cornell Tech *Cornell Tech* *Cornell Tech* *Cornell Tech*
New York, United States New York, United States New York, United States New York, United States
hc792@cornell.edu jk2575@cornell.edu jc2888@cornell.edu xc352@cornell.edu

***Abstract*—This classification project aims to determine whether a credit card user is likely to default on their payments in the future based on the past monthly customer profile. The project uses a dataset containing transaction data that has more than 458,000 unique customers profiles including features such as delinquency, spending, payment, balance, and risk from Kaggle, providing a unique opportunity to explore how machine learning can assist American Express in constructing better data filtering and preprocessing pipelines for model training or automatic assessments of default risks.**

I. Introduction

Detecting early signs of financial distress in bank borrowers is of paramount importance in minimizing potential losses for lenders. Accurate credit default prediction is a critical component of risk management for payment companies, such as credit card issuers, as it allows for proactive decision-making and effective risk mitigation strategies. In recent years, deep learning techniques have emerged as powerful tools for modeling complex patterns and relationships in financial data.

This project focuses on leveraging deep learning algorithms to improve credit default prediction in the context of credit card companies. Deep learning, a subset of machine learning, involves training neural networks with multiple layers to learn intricate representations from data. By harnessing the expressive power of deep learning models, we aim to enhance the accuracy and robustness of credit default prediction.

The primary objective of this project is to develop a deep learning-based credit default prediction model that can assist credit card companies in making more informed lending decisions. By utilizing historical data and training the model on a subset of the data, we aim to capture the underlying patterns and relationships that are indicative of credit default risks. The trained model will then be able to accurately classify borrowers as either likely to default or not, providing valuable insights for credit card companies in managing their portfolios.

By employing deep learning techniques, we aim to overcome the limitations of traditional credit default prediction models, which often rely on simplistic assumptions and linear relationships. Deep learning models have the potential to capture complex non-linear patterns in the data, leading to improved prediction performance. This, in turn, enables credit card companies to optimize their risk management strategies, allocate resources effectively, and reduce losses associated with credit defaults.

Through this project, we strive to contribute to the field of credit default prediction by harnessing the power of deep learning. By developing a state-of-the-art credit default prediction model, we aim to empower credit card companies with an advanced tool for risk assessment and decision-making. Ultimately, our goal is to enhance the financial stability of credit card companies, minimize default risks, and foster a more secure lending environment.

II. Related Work

Several studies have explored the application of deep learning methods in credit risk assessment, showcasing the potential of these techniques in improving predictive performance. One notable work is the paper titled "Sequential Deep Learning for Credit Risk Monitoring with Tabular Financial Data" [1], which investigates the use of sequential deep learning models, such as Multilayer Perceptron (MLP) and Recurrent Neural Network (RNN), for predicting financial credit risks using sequential tabular data. The authors argue that traditional models, such as boosting decision trees, may struggle to capture the sequential nature of the data, leading to suboptimal performance. Their findings suggest that a hybrid approach, combining a Gradient Boosted Decision Tree (GBDT) with either a Long Short-Term Memory (LSTM) or a Temporal Convolutional Network (TCN), yields the best predictive performance. TCN was first introduced in 2018 in the paper "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling" [2]. The paper proposed TCN as a viable alternative to RNNs for sequence modeling tasks. The findings of this paper suggest that TCN can improve predictive performance of sequential data and demonstrates the possibility of implementing TCNs in credit default prediction.

Another relevant study is "Explainable Model of Credit Risk Assessment Based on Convolutional Neural Networks" by Carlos Cardenas-Ruiz [**?**]. While Convolutional Neural Networks (CNNs) are primarily used for image processing tasks, this article explores the feasibility of applying CNNs to credit risk assessment. It discusses the impact of different input features and highlights the importance of preprocessing numerical and categorical features. The insights and techniques presented in this work can provide valuable guidance for our project.

In addition to the previous works mentioned, we also include the paper "Neural networks for credit risk evaluation:

Investigation of different neural models and learning schemes" by Adnan Khashman as another relevant reference [**?**]. This study introduces the application of Artificial Neural Networks (ANN) on real-world historical data to predict credit risk. It shares similarities with our project's objective of training the user profile over the past few months using advanced binary classification models. The paper provides detailed parameters and a comprehensive evaluation process, making it a valuable reference for our project. These prior studies demonstrate the potential of deep learning techniques in credit risk assessment and provide valuable insights into model architecture and feature engineering. By leveraging the lessons learned from these works, we aim to enhance the predictive performance of our deep learning-based credit default prediction model and contribute to the growing body of research in this field.

III. Methodology

The objective of this project is to improve the accuracy of credit default prediction using deep learning methods. To achieve this, we plan to use logistic regression as a baseline model. After that, we plan to focus on deep neural networks, which suit our problem well because they can learn complex non-linear relationships between input features and the target output.

For neural networks, we plan to first adopt the methodology introduced in "Sequential Deep Learning for Credit Risk Monitoring with Tabular Financial Data" [1], which tackles a similar problem and uses a similar form of data as ours. Specifically, we will be focusing on implementing a Temporal Convolutional Network (TCN) as it had the best performance for the problem in the paper. There are many advantages of using a TCN model. For example, TCN runs more efficiently by processing convolutions in parallel instead of iteratively as in RNNs [1]. TCN is also more capable of capturing information from the distant past, which would be a good feature for our sequential data. We plan to replicate the TCN architecture utilized by the paper. Specifically, for each TCN block, there will be one causal convolution layer, followed by a normalization layer and a dropout layer for regularization. There are two such TCN blocks after preprocessing, followed by three dense layers and one dense sigmoid layer.

We will also adjust this architecture accordingly to better suit our problem and improve the performance.

IV. Implementation, Experiments and Observations

A. Dataset

The dataset we used for this project was downloaded from Kaggle and was provided by American Express. It contains aggregated anonymized profile features for every customer at each payment data. The features include deliquency, spending, payment, balance, and risk variables. The dataset includes a training dataset of 16.39 GB and a testing dataset of 33.82 GB. Given that the training data contains multiple statement dates per customer, it presents an ideal scenario for applying deep learning techniques. To improve our efficiency of preprocessing the dataset, training the model, and tuning the hyper-parameters, it is beneficial to reduce the size of the dataset while keeping sufficient information at the same time. To do so, we used smaller precision for float values (np.float16) and reduced the training data to around 3.5 GB.

B. Preprocessing

Our work implemented and improved our preprocessing steps by following the steps suggested by "Sequential Deep Learning for Credit Risk Monitoring with Tabular Financial Data" by Carlos Cardenas-Ruiz [1]. Specifically, there are five main steps: transforming categorical data into numerical data, handling missing values, handling outliers, reducing skewness, and normalization.

For transforming the categorical features into numerical features in the dataset, we initially used a simple one-hot-encoding method. Adpoting the paper [1], we changed to use a Laplace smoothing procedure [3]. The implementation of the laplace smoothing transformation algorithm is as follows. For each categorical feature C, we mapped each category c within C to a numerical value $\hat{x}$ using the following formula:

$$\hat{x} = \frac{k \cdot \bar{y} + \sum_{i \in c} y_i}{k + |c|} \tag{1}$$

.

In this formula, $\bar{y}$ is the average of all prediction targets in the training data, c is a set of indices for the current category, y_i is the prediction target of the i-th row of the training data, $|c|$ is the number of rows of category c, and lastly k is a predefined hyper-parameter (we set $k = 30$ as suggested by [1]).

In addressing missing values in the dataset, we initially attempted simple methods such as mean or zero imputation. However, these strategies did not yield satisfactory performance. To enhance this, we took several measures:

- We eliminated all columns with $\geq 20\%$ NaN values, considering that features with such a high proportion of missing data might not contribute significantly to the prediction.
- We handled the remaining missing values by dropping the corresponding rows.
- Inspired by the method in [1], we further refined our approach:
  - Calculated 10-bin percentiles for each feature with missing values.
  - Computed target label rates for each bin of each feature.
  - Assigned the target label rate of the appropriate bin as the imputed value for each missing value.

For handling the outliers in the dataset, we performed the method of Interquartile range on each feature, except for "customer_ID" and "S_2" (a feature for transaction dates). Interquartile range is a measure that describes the spread of

the data and is useful for determining outliers. The detailed steps are as follows.

1. Calculate the interquartile range (IQR):
   $Q_1 = \text{Percentile}(5), \quad Q_3 = \text{Percentile}(95)$
   $IQR = Q_3 - Q_1$
2. Set a predefined hyper-parameter for the threshold:
   $\text{threshold} = 1.5$
3. Calculate the upper and lower bounds:
   $\text{upper\_bound} = Q_3 + (\text{threshold} \times IQR)$
   $\text{lower\_bound} = Q_1 - (\text{threshold} \times IQR)$
4. Keep the data that fall between upper and lower bounds.

For reducing the skewness of the dataset and normalizing it, we used a power transformation (Yeo–Johnson transformation [4] in particular). A power transformation is useful for stabilizing variances of the dataset and making it more in a shape of a normal distribution. We used the function provided by scikit-learn (sklearn.preprocessing.PowerTransformer()), which performs the Yeo–Johnson transformation and the normalization at the same time.

To validate the correctness and efficiency of our improved preprocessing methods, we tested it with a baseline model (Logistic Regression) and the training accuracy of it has been improved from around 84% to around 92%. This is a strong evidence that our dataset is ready to be proceeded to next steps.

C. Logistic Regression

We initiated our experiments with a basic implementation of logistic regression. Commonly employed for binary classification problems, logistic regression estimates the probability of an event's occurrence by modeling the log-odds with the formula:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

, and modify the sigmoid function and compute the final output variable as

$$\theta(x) = \sigma(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)} \tag{3}$$

D. Temporal Convolutional Network (TCN)

To overcome the limitation that the current traditional statistical methods for predicting credit card defaults in handling the complex, high-dimensional nature of credit data, we explored a deep learning technique for tracking the sequences and deep logistics for monthly customer profiles history. We decided to implement the Temporal Convolutional Network (TCN) model structure references from the article "Sequential Deep Learning for Credit Risk Monitoring with Tabular Financial Data".

As suggested in the article, the TCN model we implanted consists of multiple Temporal Convolutional blocks, followed by dense layers and dense sigmoid to generate predictions (see Figure 1 referenced from the article).

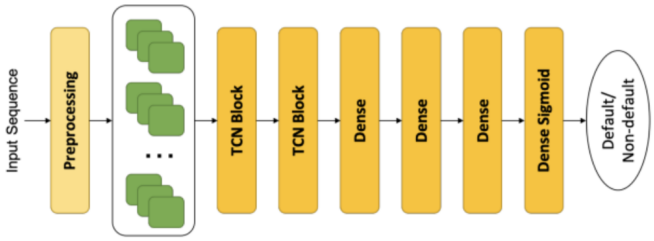

Fig. 1. TCN architecture

Currently, each block has two convolutional layers to receive the sequential data, which can either take one-dimensional time series or muti-dimensional data with multiple channels. Specifically, except for the input and output channels, the 1D convolutional network layers require kernel size and dilation parameters to increase the receptive field exponentially and capture information from the "full history". The output of convolutional layers will be then passed through the layer normalization method and an additional ReLU activation to introduce non-linearity. The output will also be applied with dropout to prevent overfitting by regularization (see Figure 2 referenced from the article).

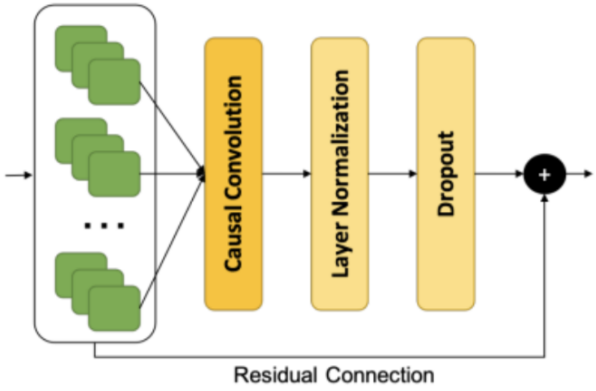

Fig. 2. TCN block

To maximize the receptive field and capture broader contextual information, we incorporated a dilation strategy based on the suggested TCN model. Specifically, we applied dilation with a power of 2 to increase the dilation factor for each subsequent block. After constructing the TCN blocks, a fully connected layer and dense sigmoid are added to map the output of the last TCN block to a single prediction outcome.

E. Training

1) Logistic Regression (Baseline): For our baseline Logistic Regression model, we set the random state parameter as 0 to ensure reproducibility in the randomization process. This

allows us to obtain consistent results when running the algorithm multiple times. Additionally, we specified the maximum number of iterations as 500 in order to converge to an optimal solution. Although increasing the number of iterations could potentially improve the model's performance further, we are limited by the available RAM in the Google Colab environment. Therefore, we chose 500 iterations as a balance between optimizing the model and resource constraints.

The performance of our logistic regression model on the full training and testing sets is summarized in tables shown below.

The model achieved an accuracy of 92.29% for training dataset. The precision, recall, and F1-score for each class are presented in the table:

TABLE I
TRAINING SET PERFORMANCE FOR LOGISTIC REGRESSION

Class	Precision	Recall	F1-Score
0.0	0.94	0.97	0.96
1.0	0.68	0.51	0.58

The model achieved an accuracy of 92.36% for testing dataset. The precision, recall, and F1-score for each class are presented in the table:

TABLE II
TESTING SET PERFORMANCE FOR LOGISTIC REGRESSION

Class	Precision	Recall	F1-Score
0.0	0.94	0.97	0.96
1.0	0.68	0.51	0.58

These results indicate that our logistic regression model has a strong performance, particularly in classifying the majority class (class 0).

2) Temporal Convolutional Network: For TCN model, the training process starts with creating data loaders for training and testing. The original training data is quite large, consisting of 2,430,040 entries; However, due to computational limitations, our project used a subset of the training data instead. Specifically, we selected 1,000,000 entries from the original training data. It is important to note that the smaller subset makes the training and tuning process more manageable meanwhile it still captures the underlying patterns and relationships in the data. Windows of size 8 are created for the data, each window represents a sequence of consecutive time steps. In addition, the original testing data is not labeled, so we decided to split the training data and use 30% of it for testing. A batch size of 64 is set, and we used the 'DataLoader' class from 'torch.utils.data' for batching and shuffling the data.

The training function iterates through multiple epochs, within each epoch the model is trained on the training data to optimize the parameters. Throughout the training process, Binary Cross Entropy With Logits Loss function is applied as the loss function to compute the gradients during the backward pass. The model's performance is evaluated after each epoch by computing the Binary Cross Entropy and test accuracy on the test data. This allows us to assess the model's performance with unseen data, and we can observe the training progress and make informed decisions such as early stopping. The optimizer used in the training process is essential for updating the model's parameters and minimizing the loss function, for this project, we used the Adam optimizer which is a popular choice for deep learning tasks.

The training process consists of the following steps:

1) **Data Preparation:** We created data loaders for the training and testing datasets. The original training data consisted of 2,430,040 entries. Due to computational limitations, we selected a subset of 1,000,000 entries from the training data. This subset captured the underlying patterns and relationships in the data while making the training and tuning process more manageable. We divided the data into windows of size 8, where each window represented a sequence of consecutive time steps.
2) **Data Split:** The original testing data was not labeled. Therefore, we decided to split the training data, using 70% for training and 30% for testing. This allowed us to evaluate the model's performance on unseen data.
3) **Data Loader Creation:** We utilized the 'DataLoader' class from 'torch.utils.data' to batch and shuffle the training and testing data. A batch size of 64 was set, ensuring efficient processing of the data during training.
4) **Model Training:** The training function was implemented to iterate through multiple epochs. In each epoch, the model was trained on the training data to optimize its parameters. We applied the Binary Cross Entropy With Logits Loss function as the loss function, which computed the gradients during the backward pass to update the model's parameters.
5) **Evaluation:** After each epoch, the model's performance was evaluated by computing the test loss and test accuracy on the test data. This allowed us to assess the model's performance on unseen data and monitor its progress throughout the training process.

By following this training process, we aimed to optimize the TCN model's parameters and evaluate its performance on the task at hand.

F. Hyper-paramters Tuning

TABLE III
PARAMETERS FOR DEFAULT MODEL

Epochs	20
Learning Rate	0.01
Number of Channels	[32,64]
Kernel Size	2
Dropout Rate	0.2

To optimize the performance of the Temporal Convolutional Network (TCN), we performed a rigorous hyperparameter tuning process. We systematically explored the impact of key hyperparameters including the learning rate, dropout rate, kernel size, and number of channels. We established a control

group with default parameter values and an experimental group where we varied the parameters within specific ranges. By comparing the model's performance on the training dataset and testing prediction across different parameter settings, we were able to identify the optimal combination of hyperparameters. As mentioned above, In our experiments, we initialized the TCN model with default values, serving as a starting point for exploring different hyperparameter configurations. Table III is the default values for initializing the model for experiments.

1) Learning Rate: We explored a range of learning rates by incrementally scaling values from 0.001 to 0.01. This range was selected based on pre-testing experiments, where we observed that extremely high learning rates (above 0.1) resulted in a decrease in accuracy. During the pre-testing phase, we initially tested a wider range of learning rates, from 0.001 to 0.5. The results indicated that the model achieved high accuracy at the beginning of the range, followed by a gradual decrease in performance beyond 0.1. Based on these findings, we narrowed down the range to 0.001 to 0.1 with a step size of 0.001 for the final experimentation. As states in the purpose and steps in this section, for each learning rate, we trained the TCN model using the default number of channels, kernel size, and dropout rate. The results are shown in Figure 3.

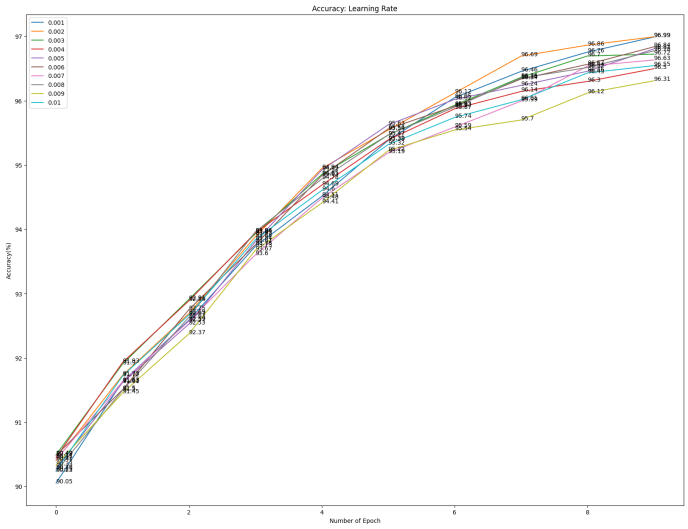

Fig. 3. Accuracy: Learning Rate

TABLE IV
ACCURACY OF THE TCN MODEL ON THE TRAINING DATASET FOR DIFFERENT LEARNING RATES

Learning Rate	Accuracy
0.001	**91.82%**
0.002	89.52%
0.003	91.74%
0.004	91.47%

We observed that as the learning rate increased, the accuracy of the model initially improved but reached a plateau around a learning rate of 0.006. Beyond that, further increasing the learning rate did not significantly impact the accuracy.

The best learning rate was found to be 0.001, which achieved an accuracy of 91.82% on the training dataset.

2) Dropout Rate: Similarly, we investigated the effect of different dropout rates on the model's accuracy. We selected dropout rates of 0.05, 0.1, 0.15, and 0.2. The TCN model was trained with the default number of channels, kernel size, and learning rate. Figure 4.

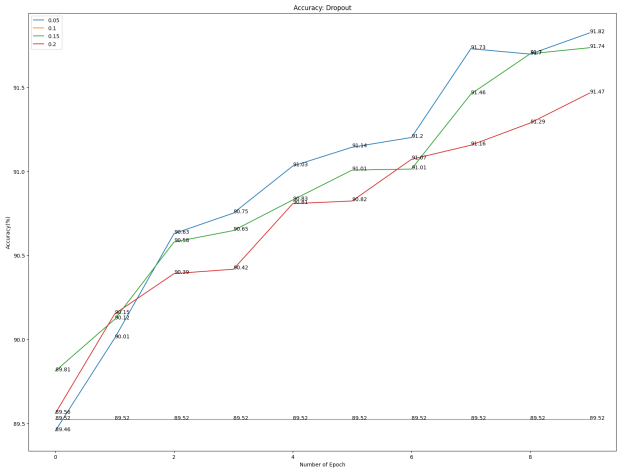

Fig. 4. Accuracy: Drop-out Rate

The results are summarized in Table V.

TABLE V
ACCURACY OF THE TCN MODEL ON THE TRAINING DATASET FOR DIFFERENT DROPOUT RATES

Dropout Rate	Accuracy
0.05	**91.82%**
0.1	89.52%
0.15	91.74%
0.2	91.47%

From the results, we observed that a dropout rate of 0.05 achieved the highest accuracy of 91.82% on the training dataset. Higher dropout rates led to decreased accuracy, indicating that excessive regularization adversely affected the model's performance.

To optimize the performance of the Temporal Convolutional Network (TCN), we conducted a hyperparameter tuning process by varying the kernel size and number of channels. We evaluated the effect of these hyper-parameters on the model's accuracy on the training dataset.

3) Kernel Size: We explored different kernel sizes, namely 2, 3, and 5. For each kernel size, we trained the TCN model using the default number of channels, dropout rate, and learning rate. The results are summarized in Table VI.

From the results, we observed that a kernel size of 3 achieved the highest accuracy of 92.13% on the training dataset. Smaller kernel sizes (2) and larger kernel sizes (5) resulted in slightly lower accuracy.

TABLE VI
ACCURACY OF THE TCN MODEL ON THE TRAINING DATASET FOR DIFFERENT KERNEL SIZES

Kernel Size	Accuracy
2	91.82%
3	**92.13%**
5	90.99%

4) Number of Channels: We investigated different configurations for the number of channels in the TCN model. The number of channels determines the depth and complexity of the network. We explored three configurations: [32], [32, 64], and [32, 64, 128]. The TCN model was trained with the best kernel size of 3, and the results are summarized in Table VIII.

TABLE VII
ACCURACY OF THE TCN MODEL ON THE TRAINING DATASET FOR DIFFERENT NUMBER OF CHANNELS

Number of Channels	Accuracy
[32]	91.73%
[32, 64]	**92.38%**
[32, 64, 128]	91.94%

Based on the results, we found that the configuration with [32, 64] channels achieved the highest accuracy of 92.38% on the training dataset. The configuration with a single channel [32] and a deeper configuration [32, 64, 128] achieved slightly lower accuracies.

By fine-tuning the kernel size and number of channels, we were able to improve the accuracy of the TCN model on the training dataset, providing a more robust foundation for subsequent evaluation on the test dataset.

These findings demonstrate the importance of carefully selecting appropriate values for the learning rate and dropout rate when training the TCN model. The optimal combination of hyperparameters includes a learning rate of 0.001 and a dropout rate of 0.05.

By systematically tuning these hyper-parameters, we were able to improve the accuracy of the TCN model on the training dataset, providing a solid foundation for its subsequent evaluation on the test dataset.

G. Cross-Validation

We employed *k-fold cross-validation*, where the dataset is divided into *k* equal-sized folds. The model is then trained and evaluated *k* times, each time using a different fold as the validation set and the remaining folds as the training set. This process allows us to assess the model's performance across different subsets of the data and obtain a more robust estimate of its generalization capability.

To perform cross-validation, we defined a set of hyperparameters to tune the model. These hyperparameters include the learning rate (*lr*), batch size (*batch_size*), number of channels (*num_channels*), kernel size (*kernel_size*), and dropout rate (*dropout*). We specified a range of values for each hyperparameter, and the *cross_val* function systematically explores all possible combinations of these hyperparameters.

For example, we considered learning rates of 0.01, 0.001, and 0.0001, a fixed batch size of 64, a number of channels of [32, 64], a kernel size of 2, and dropout rates of 0.1, 0.15, 0.2, 0.25, and 0.3. The *cross_val* function then performed the training and evaluation process for each combination of these hyper-parameters.

By applying cross-validation, we obtained the optimal set of hyper-parameters (*opt_params*) that resulted in the best performance of the model. These hyper-parameters were selected based on the average performance across all folds, ensuring that the model achieves good generalization and avoids overfitting.

Using cross-validation allows us to confidently evaluate and select the most suitable hyper-parameters for our model, ensuring its robustness and ability to generalize well to unseen data.

H. Result

After optimizing the model by setting up experimental and control groups according to the required parameters, we proceeded to the next milestone. In this milestone, we applied the parameters that yielded the best performance in the previous section to further refine our model. The application of parameters is shown in table VII.

TABLE VIII
PARAMETERS FOR FINAL APPLICATION

Learning Rate	0.001
Batch Size	64
Number of Channels	[32,64]
Kernel Size	2
Dropout Rate	0.25

1) Data Split: The original training dataset was split into a 70:30 ratio, with 70% of the data used for training and 30% reserved for testing. This split ensured that a substantial portion of the data was available for model training, while still providing a separate dataset for evaluating the model's performance.

2) Testing Dataset:: The remaining 30% of the data was used as a separate testing dataset. This dataset served as an independent evaluation set to assess the generalization and performance of the trained TCN model on unseen data.

3) Loss Analysis: The binary cross-entropy loss (*BCEWithLogitsLoss*) was utilized as the loss function for binary classification. The model was optimized using the Adam optimizer with a learning rate of *lr*.

After training the TCN model for 60 epochs on the training dataset, we evaluated its performance on the test dataset. Figure 5 illustrates the test loss with respect to the train loss during the training process. We observed that the test loss appears slightly higher than the train loss after epoch 5. The lowest test loss, approximately 0.075, occurred when the train loss was around 0.05. Despite this slight difference, the model's performance remains quite good.

4) Accuracy Analysis: Figure 6 presents the test and train accuracy over the training epochs. The test accuracy curve follows a similar pattern to the train accuracy curve. However, it performs slightly worse than the train accuracy. In the final epoch (epoch 30), the test accuracy was measured at 97.5%, while the train accuracy achieved 98.2%.

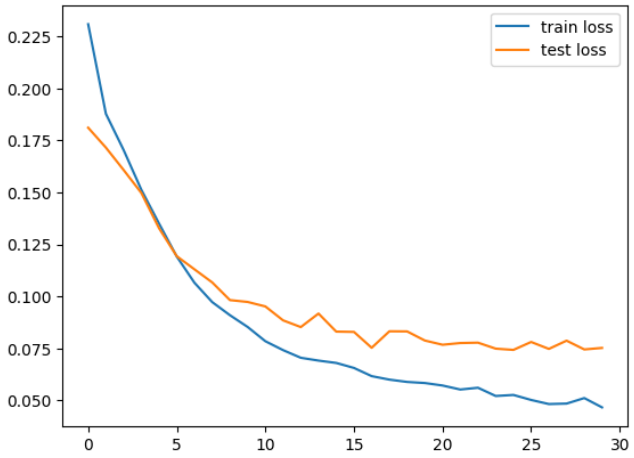

Fig. 5. Test Loss with Train Loss

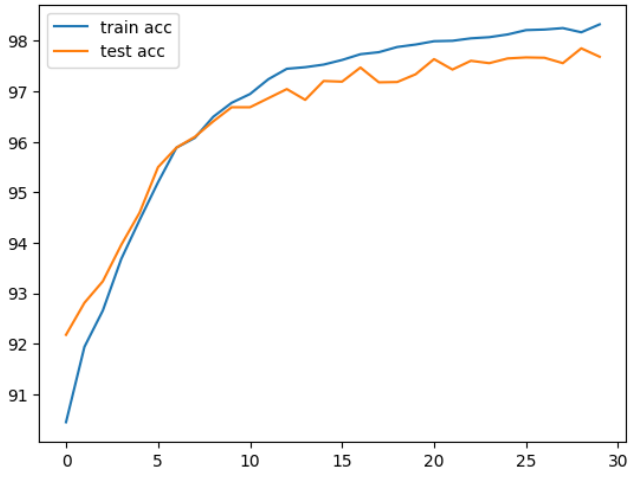

Fig. 6. Test and Train Accuracy

Lastly, the trained TCN model demonstrates strong predictive capabilities for credit default prediction. The high accuracy achieved on both the training and test datasets highlights the effectiveness of the model in capturing patterns and making accurate predictions.

For future work, we recommend further investigation into the model's performance on larger datasets and its scalability to handle real-time credit default prediction scenarios. Additionally, exploring advanced techniques such as ensemble methods or incorporating additional features could potentially enhance the model's performance even further.

V. Limitations

One limitation of our project is the computational requirements associated with training the Temporal Convolutional Network (TCN) model on a large dataset. Due to the size of the dataset and the complexity of the TCN model, the training process can be computationally demanding and time-consuming, especially when conducted on limited computational resources such as Google Colab. This limitation can impact the efficiency and speed of model development and experimentation. To address this, subsetting the data or exploring alternative computing resources with more powerful GPUs could be considered. Despite these challenges, the results obtained from training the TCN model on a subset of the dataset still provide valuable insights into credit default prediction. Acknowledging these limitations and exploring optimization strategies can help advance credit default prediction research using TCN models.

VI. Risks and Ethical Considerations

One of the main potential risks we are concerned about is overfitting, which occurs when the model memorizes the training data and performs poorly on new, unseen data. Given the complexity of the dataset with 190 features, there is a high risk of overfitting, as each customer profile is subjective. To mitigate this risk, we will employ techniques such as regularization, early stopping, and cross-validation to ensure the model's generalizability and prevent overfitting.

Deep learning models, by nature, tend to be complex and difficult to interpret. This complexity can pose challenges in identifying and debugging errors within the model. Specifically, neural networks have numerous hyperparameters that require tuning to achieve optimal performance, making the process time-consuming and complex. Therefore, it is crucial to select an appropriate model architecture for the task and utilize techniques such as model interpretability to gain insights into the model's behavior and enhance its interpretability.

Ethical considerations play a significant role in this project. Data privacy is paramount, and obtaining proper consent for data usage is essential. Furthermore, it is crucial to address potential biases in the dataset and ensure fairness in model predictions. The use of predictions generated by the model should be strictly limited to academic purposes, and any deployment outside of academia should undergo rigorous testing and validation. Throughout the project, we will adhere to key ethical principles, including transparency, fairness, and respect for the data subjects, to ensure responsible and ethical conduct.

VII. Conclusions and Future Work

In this project, we successfully trained a Temporal Convolutional Network (TCN) model using the optimal set of hyperparameters obtained through cross-validation.

The TCN model, instantiated with the optimal hyperparameters including the number of channels (*num_channels*), kernel size (*kernel_size*), and dropout rate (*dropout*), demonstrated

promising results. The binary cross-entropy loss (*BCEWithLogitsLoss*) was used as the loss function for binary classification.

The project recognized the significance of early detection of financial distress in bank borrowers and the importance of accurate credit default prediction for risk management in payment companies. Traditional models often face limitations in capturing complex non-linear patterns present in financial data, which can hinder their predictive performance. By utilizing deep learning techniques, we aimed to overcome these limitations and enhance the accuracy and robustness of credit default prediction.

Through the development of the deep learning model, we achieved promising results in credit default prediction. The model's ability to learn from the past monthly customer profile data, including features such as delinquency, spending, payment, balance, and risk, provided valuable insights for constructing better data filtering and preprocessing pipelines. The improved preprocessing methods contributed to an enhanced testing accuracy of the baseline Logistic Regression model from around 92.29% to around 97.5%, validating the effectiveness of our dataset and preprocessing approaches.

The model's optimal hyper-parameters, including the learning rate, batch size, number of channels, kernel size, and dropout rate, were determined through careful tuning and cross-validation. The selected combination of these parameters outperformed individual parameter tuning, further reinforcing the model's effectiveness in credit default prediction.

By empowering credit card companies with an advanced credit default prediction model, we anticipate significant improvements in risk assessment, decision-making, and risk mitigation strategies. The model's ability to capture complex patterns and relationships in the data will enable credit card companies to optimize their risk management strategies, allocate resources effectively, and reduce losses associated with credit defaults.

For future work, we recommend exploring the model's performance on larger datasets to evaluate its scalability in handling real-time credit default prediction scenarios. Additionally, further investigations into advanced techniques such as ensemble methods and the incorporation of additional features could potentially enhance the model's performance even further. Specifically, for logistic regression, we can:

1) **Adding Regularization Method**: The regularization techniques such as L1 or L2 regularization methods will helps the model to prevent overfitting and improve generalization of the model.
2) **Addressing Class Imbalance**: We haven't evaluated the balance of the two classes throughout the process. By observing the precision, recall, and F1 score for both training and testing datasets, we found that the current logistic regression performs poor accuracy for class "1" rather than class "0". Therefore, to enhance the future predictions of both the logistic regression and TCN models, it is imperative to address the class imbalance in the dataset. Additionally, further parameter tuning and exploration of other techniques may be necessary to improve the overall performance and achieve better accuracy for class "1".

For Temporal Convolutional Network, we have much more to working with:

1) **Adding Convolution Layers**: We currently have two layers of convolution layers inside the block function. We consider to increase the depth of TCN models by adding more convolution layers to capture more complex patterns and dependencies in the input data.
2) **Adding Dense Layers**: We only have one dense layer inside of current TCN model before dense sigmoid function. By adding more dense layer into the network, the model can perform more non-linear transformations of the input data.

In conclusion, this project contributes to the field of credit default prediction by leveraging deep learning techniques to develop an accurate and robust model. The successful implementation of the deep learning-based credit default prediction model paves the way for improved risk management and better financial outcomes for credit card companies, ultimately fostering a more secure lending environment.

REFERENCES

[1] Jillian M. Clements, Di Xu, Nooshin Yousefi, and Dmitry Efimov. Sequential deep learning for credit risk monitoring with tabular financial data. 2020.

[2] Bai, Shaojie, et al. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. arXiv, 19 Apr. 2018. arXiv.org, https://doi.org/10.48550/arXiv.1803.01271.

[3] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. Introduction to information retrieval. Cambridge University Press. 234–265 pages.

[4] Yeo, In-Kwon; Johnson, Richard A. (2000). "A New Family of Power Transformations to Improve Normality or Symmetry". Biometrika. 87 (4): 954–959.

[5] Smith, J. (2019). "Neural networks for credit risk evaluation." In Jane Doe (Ed.), Advances in Intelligent Systems and Computing. Springer.

[6] Rostami, S., Saberi, M. (2010). "Neural networks for credit risk evaluation." Expert Systems with Applications, September 2010.