Deep learning for combating energy theft: Modeling and analysis of Hybrid LSTM

Hongxin Gao

Technical Report

RHUL–ISG–2022–6

11 April 2022

Information Security Group
Royal Holloway University of London
Egham, Surrey, TW20 0EX
United Kingdom

Candidate Number: 2110945

Deep Learning for Combating Energy Theft: Modeling and Analysis of Hybrid LSTM

Submitted as part of the requirements for the award of the
MSc in Information Security
at Royal Holloway University of London.

Information Security Group
Royal Holloway, University of London
August 2021

Table of Contents

List of figures & tables

List of Abbreviations & Acronyms

ACF	Autocorrelation function
AMI	Advanced Metering Infrastructure
AMR	Automatic Meter Reading
AUC	Area Under the Curve
CNN-LSTM	Convolutional Neural Network & Long Short-Term Memory
ConvLSTM	Convolutional Long Short-Term Memory
ETD	Electricity Theft Detection
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristic
RNN	Recurrent Neural Networks
GAN	Generative Adversarial Network
kWh	Kilowatt Hour
KNN	k-Nearest Neighbors
LASSO	Least Absolute Shrinkage and Selection Operator
MLP	Multilayer Perceptron
NTL	Non-technical Losses
PCC	Pearson’s Correlation Coefficient
PR	Precision-Recall
SGCC	State Grid Corporation of China
SMOTE	Synthetic Minority Oversampling Technique
STL Decomposition	Seasonal and Trend Decomposition Using Loess

Executive Summary

In the energy industry, power theft has been a constant concern for many countries. With the development of technology, the means of electricity theft also reflects the characteristics of advanced and concealed. However, the emergence of artificial intelligence technology has also elevated the combat against electricity theft to a new level. The classical machine learning has been proven to be effective in electricity theft detection (ETD). Instead of further corroborating the feasibility of them, this project further explores the usability of advanced models in the emerging field of deep learning. Recent academic journals and research results show that advanced models and algorithms based on convolutional neural network (CNN), long short-term memory (LSTM), and generative adversarial network (GAN), among others, are also effective and even more flexible and efficient in ETD.

This project uses a data set of electricity consumption of real users, which contains data and labels of normal users as well as electricity theft users, i.e., a supervised learning approach is used to deal with the binary classification issue. The performance of the target model on the unseen data set is evaluated by training the model and algorithm, i.e., the performance of the model in identifying normal electricity users and electricity theft users. During the project, some advanced and effective deep learning techniques are also well applied and show good results. In terms of data pre-processing, techniques such as k-nearest neighbors (KNN) imputation processing for missing values, interquartile range (IQR) processing for outliers and borderline-synthetic minority oversampling technique (Borderline-SMOTE) processing for imbalance classification were successfully applied to the project. Also, the clear visualization analysis provides a good basis for modeling. Three models were successfully built, with multilayer perceptron (MLP) as the baseline model, CNN & LSTM (CNN-LSTM) as a comparable model and convolutional LSTM (ConvLSTM) as a novel model. The model convergence is accelerated by optimizing hyperparameters such as dropout and learning rate. And more comprehensive metrics are used on the unseen data to evaluate the feasibility, accuracy, and robustness of the model in identifying electricity theft users.

In this project, ConvLSTM outperformed the other models with accuracy, loss, precision, recall, F1-score, Cohen's kappa, receiver operating characteristic-area under the curve (ROC-AUC) and precision-recall-AUC (PR-AUC) of 0.984, 0.089, 0.984, 0.985, 0.984, 0.969, 0.993, 0.991 respectively. In addition, ConvLSTM supports multi-dimensional electricity data input for better extraction of time series features, and batch normalization technology supports direct transformation of raw electricity data in model compilation without tedious and time-consuming data pre-transformation. This also demonstrates that ConvLSTM also shows a lot of room for improvement in terms of flexibility of model architecture adjustment and efficiency of data processing. This can also better improve the timeliness of power companies in combating electricity theft, and enable them to adjust their detection deployment in a timely manner in an environment where theft methods are constantly changing.

A journal version of this project report is pending submission to International Transactions on Electrical Energy Systems for review.

1. Introduction

1.1 Background of Smart Grid Development

The smart grid has made great progress as a mainstream trend in the development of electricity networks. It can effectively integrate the electricity consumption behavior of its service users with intelligent communication and monitoring [1]. With the development of the smart grid, an evolution from automatic meter reading (AMR) to advanced metering infrastructure (AMI) has been achieved. AMI is an important component of the smart grid, which makes possible two-way communication between the electric utility and the customer. It also enables remote meter reading and access to data in an efficient and accurate manner [2]. As one of the essential devices in AMI, the smart meter plays a key role in the construction of smart grid. It can obtain information from the end-user side and measure the energy consumption of users in real time, which also greatly improves the efficiency of power supply enterprise's electricity consumption information collection [3].

A typical AMI is a mix of powerline communications, radio frequency networks, and point-to-point devices that contain direct cellular modem devices for communications. All devices are connected via an IP-based backhaul communication network. As shown in figure 1:

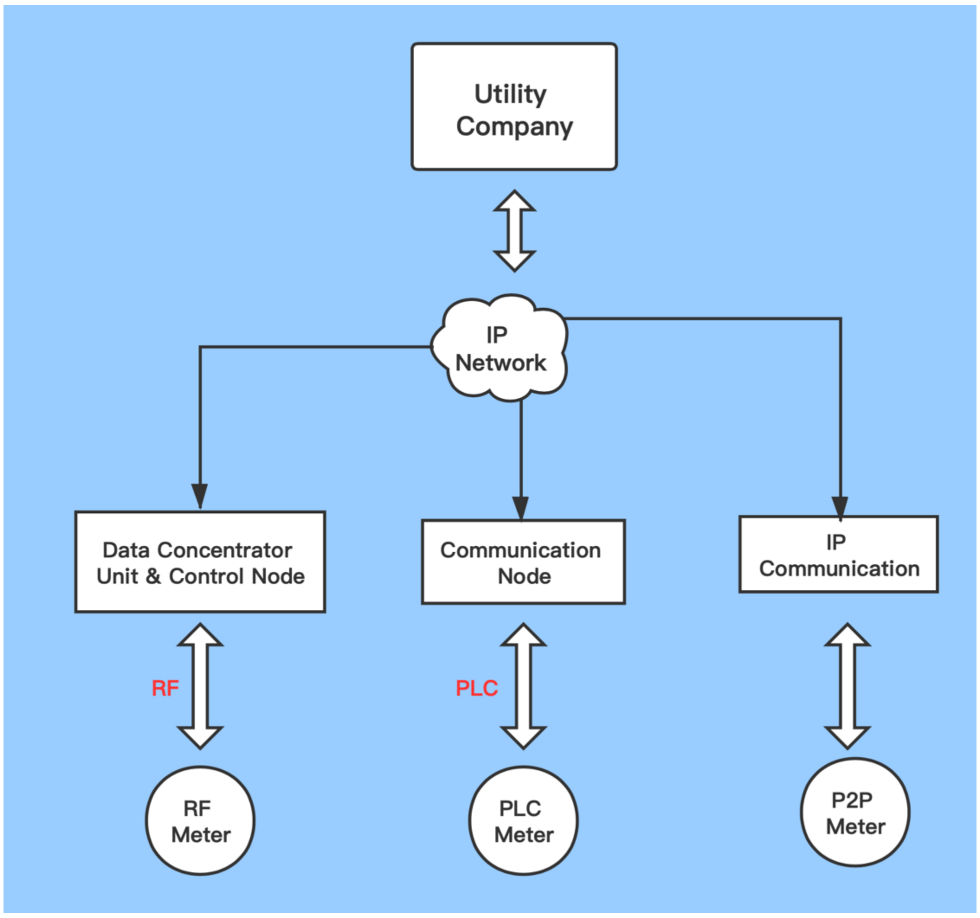


Figure 1: AMI instance layout diagram -adapted from [26]

1.2 The Current State of Electricity Theft

However, energy theft has always been an intricate problem. Currently, electricity theft has become one of the major causes of non-technical losses (NTL) in the grid [4]. One of the reasons for the rapid growth of AMI over the past few years is to reduce the NTL caused by electricity theft [26]. It has also diversified with the development of smart grids, evolving from traditional malicious tampering with physical metering devices to sophisticated remote computer penetration. The following figure 2 shows the different modes of electricity theft:

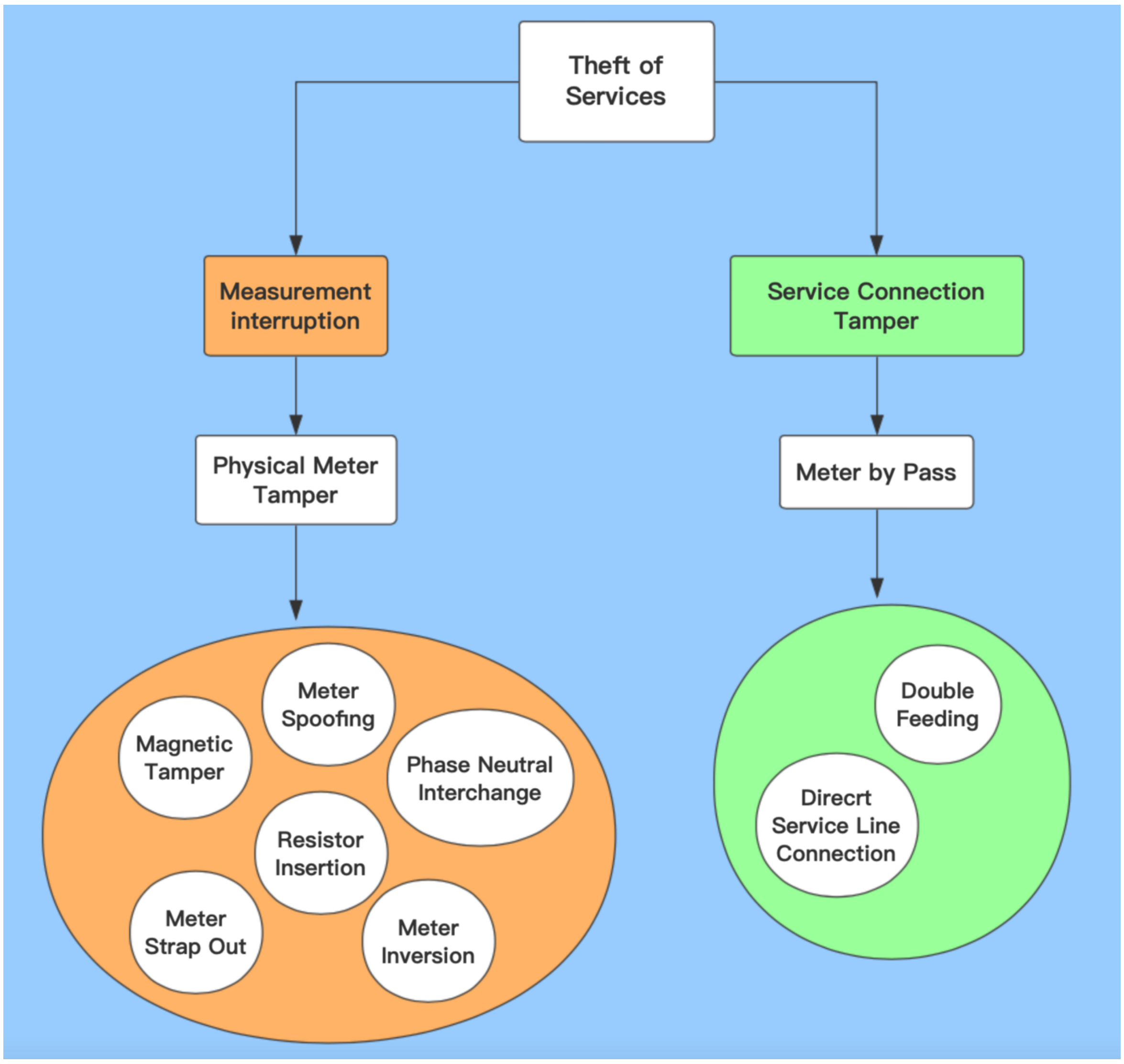


Figure 2: Different attack modes of electricity theft - adapted from [26]

According to Northeast Group, LLC. (2017) [27] report, the power supply sector worldwide loses about $96 billion a year due to non-technical losses, which include electricity theft, fraud, etc. For example, in India, the annual loss due to electricity theft is about $4.5 billion [29]. The 2020s will be a critical period for the development of the global smart grid. Because North America, Western Europe and other countries have been on the road of building smart grid. For emerging markets, however, this is one area of massive investment. The report also points out that the 50 emerging markets are well placed to invest more than $88 billion over the next few years [28]. This is undoubtedly a signal to promote the development of global AMI layout, but also implies the determination of the national level in the governance of NTL in the power grid. As shown in figure 3:

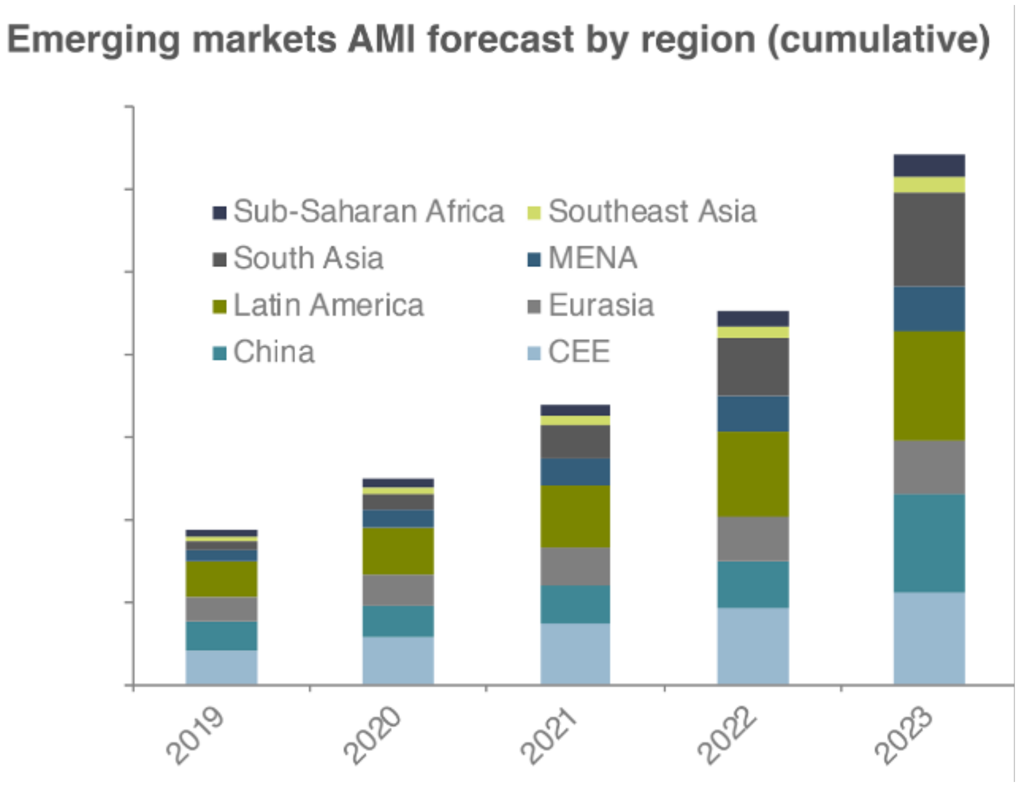


Figure 3: Emerging markets AMI forecast by region (cumulative) [28]

From the perspective of information security, electricity theft seriously compromises the integrity and availability of the power transmission process [5]. It not only causes enormous property damage, but it is also a criminal offence. Electricity consumption data is given value in the society and belongs to the assets of the residents and the power supply sector.

1.3 Motivation

The development of AMI in the smart grid has also contributed to the diversity of detecting electricity theft. It provides a large amount of data support while improving energy monitoring. In the big data environment, machine learning has also been introduced to the problem of electricity theft. For example, Jamacia Public Service Company Ltd. (2019) has applied machine learning to combat electricity theft [30]. Electricity theft is a form of damage to electric assets, which is reflected in the data as an abnormal state of electricity use. The process of detecting abnormal electricity usage is essentially a process of risk assessment as well. The process of risk assessment can be summarized into three stages: risk identification, risk analysis and risk evaluation [5]. However, classical machine learning and deep learning can significantly reduce the time of risk assessment and accurately combat electricity theft by, for example, feature engineering and modeling of electricity consumption data sets containing electricity theft features, which also means an optimization of manual detection methods [6].

At the same time, as previously mentioned, the purpose and methods of electricity theft today are tied to technological developments, such as the extraordinary popularity of bitcoin mining. A report by Criddle (2021) reveals the frightening reality that Bitcoin uses more electricity each year than the entire country of Argentina [44]. News about bitcoin power theft is commonplace on the Internet, and fighting this emerging field of theft is not enough with manual methods alone.

After the successful application of classical machine learning in ETD, some ETD modeling research has gradually emerged in the field of deep learning. However, this is one of the motivations for this project, hoping that some progress can be made in ETD modeling by studying more advanced algorithms in deep learning. Since everyone can participate in bitcoin mining and it can pose a threat of electricity theft. Then this project also hopes to make more people aware of advanced ways to combat electricity theft by demonstrating deep learning ETD modeling and analysis.

2. Summary of Literature Review in Machine Learning ETD

2.1 Introduction to Neural Network

Multilayer Perceptron

MLPs are the basis of artificial neural networks. Perceptrons are single neuron models that are likewise the building blocks of complex neural networks. They have weighted input signals and use activation functions to generate output signals [46]. A simple neuron is shown in the following figure 4:

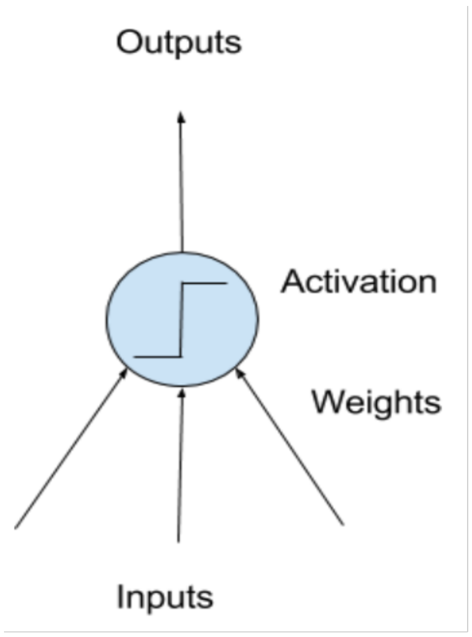


Figure 4: The architecture of neuron [46]

Neurons are arranged to form a network of neurons with a topological structure. A row of neurons is called a layer, and a network can have more than one layer. These include:
1. Input layer: the underlying layer from which the data set gets its input.
2. Hidden layer: a simple network structure in which one neuron in the hidden layer directly outputs a value.
3. Output layer: it is responsible for outputting a value or vector corresponding to the desired format of the modeling problem. For example, a regression problem may have only one output neuron and may have no activation function. While a binary classification problem may have only one output neuron and use a Sigmoid activation function to output a value between 0 and 1. This is transformed into a classification value by the setting of a threshold value. The simple structure is as following figure 5:

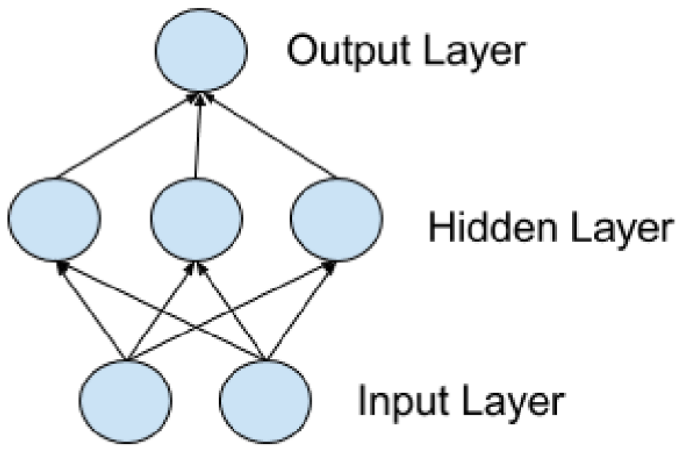


Figure 5: The architecture of MLP [46]

In addition, to prevent overfitting, dropout [48] can be added to the neural network model, which is also a regularization technique. It is a technique that ignores randomly selected neurons during the training process. They are dropped out randomly. This means that their contribution to the activation of downstream neurons is temporarily removed on the forward channel, and any weight updates are not applied to neurons on the backward channel [24].

Convolutional Neural Network (CNN)

CNN is another powerful artificial neural network. It retains the spatial structure of the problem and has been developed for target recognition problems, such as computer vision.

There are three types of layers in a CNN [46]:
a. The convolutional layer. It mainly has filters and feature maps. The filter is essentially the neuron of the layer with weighted input. The size of the output is a square receptive field. In the network structure, the convolution layer takes the input from the feature map of the previous layer. And the feature map is the output of the filter of the previous layer is applied.
b. Pooling layer. The pooling layer downsamples the feature map of the previous layer. It is relatively simple in structure and takes the average or maximum of the input values to create its own feature map.
c. Fully connected layer. This layer is used at the end of the network structure after feature extraction and integration by the convolution and pooling layers. These layers can include nonlinear activation functions and Softmax activation and perform prediction of the model. Its structure is as following figure 6:

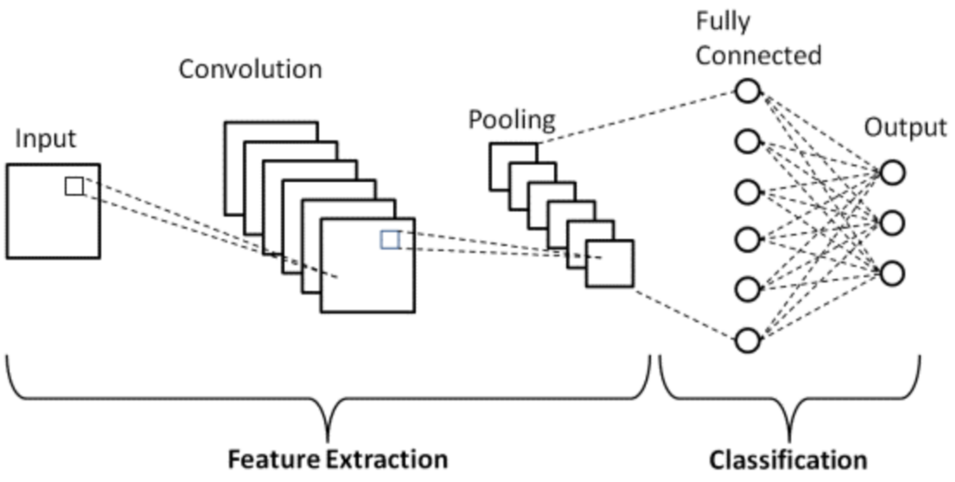


Figure 6: The architecture of CNN [53]

Zheng, Yang, Niu, Dai, & Zhou (2017) [12] used a Wide and Deep convolutional neural network in the detection model of electricity theft. The framework mainly consists of a wide component and a deep CNN component. A fully connected layer is used in the wide component. The cases of normal electricity usage in the data set show periodicity, while the data of electricity theft is less periodic. The cases learn features by memorizing 1-D time series. the CNN is robust to the location and orientation of the target in the scene, and the principle also uses 1-D sequences. That means invariance to the specific position of the feature. The case uses rectified linear unit (ReLU) as the activation function, which activates only positive values. In the back propagation process, each cell calculates its weight based on the loss values sent from the upper layers. A deep CNN component that processes electricity consumption data in two dimensions depending on the number of days. The component consists of multiple convolutional layers, a pooling layer and a fully connected layer. The convolutional layer contains unique filters, and the activation function is selected as ‘tanh’. The pooling layer is selected as maximum. The fully connected layer uses a logistic loss function, which should be 'binary_crossentropy'. The activation function is chosen as Sigmoid.

Recurrent Neural Network (RNN) and Long Short-term Memory (LSTM) Network

RNN is designed for sequential problems and their connections have loops that add feedback and memory to the network over time. Long short-Term memory (LSTM) network is a recurrent neural network trained by time back propagation (LSTM models include stacked LSTM, CNN-LSTM, bidirectional LSTM, etc). It has a unique formulation that avoids the problem that other RNNs cannot be trained and scaled. Moreover, it overcomes the problems of gradient disappearance and gradient explosion, truncated backpropagation through time (TBPTT) is a key concept in the training LSTM model. Unlike neurons, the memory blocks of LSTM networks contain states and outputs and are connected in layers. Each of these blocks has three gates: a forget gate, an input gate, and an output gate [21]. Also, a sigmoid activation function is used to control whether they are triggered or not. Sliding window is a method for transforming time series into supervised learning. Its structure is as following figure 7:

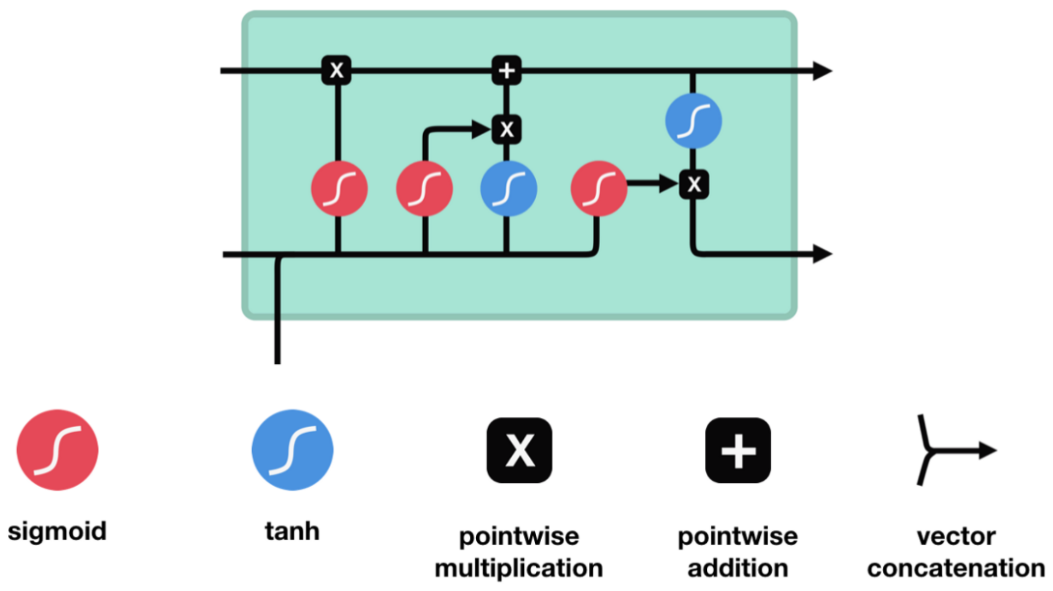


Figure 7: The architecture of LSTM [52]

CNN-LSTM

The CNN-LSTM framework consists of feature extraction of input data using CNN layers combined with LSTM to support sequence prediction. It was essentially developed for visual time series prediction problems and for applications that generate textual descriptions from image sequences [23]. The simple structure is as following figure 8:

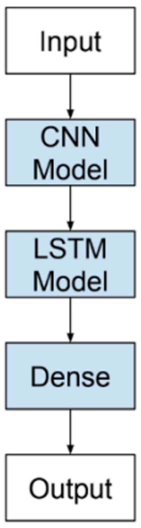


Figure 8: The architecture of CNN-LSTM [46]

Hasan, Toma, Nahid, Islam & Kim (2019) [6] and Madhure, Raman & Singh (2020) [22] use a CNN-LSTM model in the anomaly detection model. Case [6] also solves a binary classification problem with normal electricity consumption and theft data as labels. It includes seven hidden layers (the first four hidden layers perform active operations), each of which consists of twenty feature sets. The rest of the hidden layers are LSTM. the ReLU activation function is used in the product convolution layer. The maximum value is selected for the pooling layer. The SoftMax function is selected after the operation. In the LSTM model, Case [22] adds a Dropout layer after each LSTM layer. This can simply and effectively prevent the model from overfitting. The optimizer chooses Adam to update the network weights. Two fully connected layers were placed at the end of the network to make predictions through the network. One of the layers has 24 neurons and a linear activation function is chosen, and finally an output layer contains 1 neuron.

2.2 The Current Situation of Machine Learning in ETD

Predictive modeling is a significant concept in machine learning, where different data are understood by models. The benefit of predictive modeling is that it allows the development of models that make the most accurate predictions, rather than focusing on explaining why a model makes a prediction.

In classical machine learning, some algorithmic modeling can also yield good metrics on electricity theft problems. For example, linear algorithms: Logistic Regression (LG) and Linear Discriminant Analysis (LDA), nonlinear algorithms: KNN, Naive Bayes (NB), Classification and Regression Trees (CART) and Support Vector Machines (SVM). They can model and handle classification problems. Also, in dealing with regression problems, linear algorithms: Linear Regression, Bridge Regression, Least Absolute Shrinkage and Selection Operator (LASSO) Linear Regression and Elastic Net Regression, nonlinear algorithms: KNN, CART and SVM. They can also be used for related modeling problems. As summarized by Hasan et al. (2019) [6], algorithms of classical machine learning show good performance on different electricity consumption data set. In addition, ensemble algorithms can also improve the accuracy of the dataset. For example, Bagging (Bagged Decision, Random Forest and Extra Trees), Boosting (AdaBoost and Stochastic Gradient Boosting) and Voting.

Along with the development of machine learning and smart grid, deep learning as a branch of machine learning is also widely used in industry. Neural networks as a concept of deep learning are also widely used in computer vision, speech recognition and anomaly detection and other aspects [6]. MLP, CNN, RNN, and hybrid models for modeling can also cover power-use data set and deal with regression and classification problems [7]. The development of deep learning is not a complete replacement for classical machine learning, but there is a difference in the details that both focus on in terms of ideas for dealing with different data sets. Some limitations of traditional machine learning are highlighted in the processing of different data. Dorffner, G. (1996) shows that neural networks are robust to noise in the input data and mapping functions and can even support learning and prediction in the presence of missing values. At the same time, neural networks do not make strong assumptions about the mapping function and can easily learn linear and nonlinear relationships [8].

From another perspective, it can also be understood the electricity consumption data set as a time series forecasting problem [20]. Unlike simple classification and regression problems, time series problems add sequential complexity or time dependence between observations. Deep learning neural networks are able to automatically learn arbitrarily complex mappings from inputs to outputs and support multiple inputs and outputs. CNN support effective feature learning. It provides efficiency and better performance in identifying, extracting, and refining useful features from raw data [9]. CNN achieves this by directly manipulating raw data (e.g., raw pixel values) rather than deriving domain-specific or handcrafted features from raw data. RNN, such as LSTM, are used for complex natural language processing problems. This capability can be used for time series prediction. LSTM networks support efficient learning of temporal dependencies [10]. These models can also be used to advantage by mixing them, such as hybrid models like CNN-LSTM and ConvLSTM.

3. Project Outline

3.1 Project Process Description

The project is divided into five main phases:

1. Data pre-processing. Data cleaning, including missing value removal and filling (KNN Imputation), and outlier screening (IQR).

2. Data visualization. Visualization of the raw data set through histograms, curves, Pearson's correlation coefficient (PCC), autocorrelation function (AFC) and seasonal and trend decomposition using loess (STL decomposition), and preliminary examination of the relationships between the data.

3. Imbalance classification. Generate more realistic data of electricity thieves by Borderline-SMOTE.

4. Data transformation. Normalization is performed before the data is fed into the model, which also includes the use of batch-normalization directly in the model.

5. Model building. Three models are built for comparison, which include MLP (baseline), CNN-LSTM and ConvLSTM. For each model split training, validation and testing data set. The optimal model is maintained by optimizing feature extraction, hyperparameters and functions, etc.

6. Comparative analysis of performance metrics. The optimal model is determined by a comprehensive evaluation of the models on the test data set, which includes accuracy, loss, confusion matrix, F1-score, recall, precision, Cohen's kappa, receiver operating characteristic-area under the curve (ROC-AUC), precision-recall-AUC (PR-AUC), and related visualization results.

To express the project process more clearly and intuitively, please see the following figure 9:

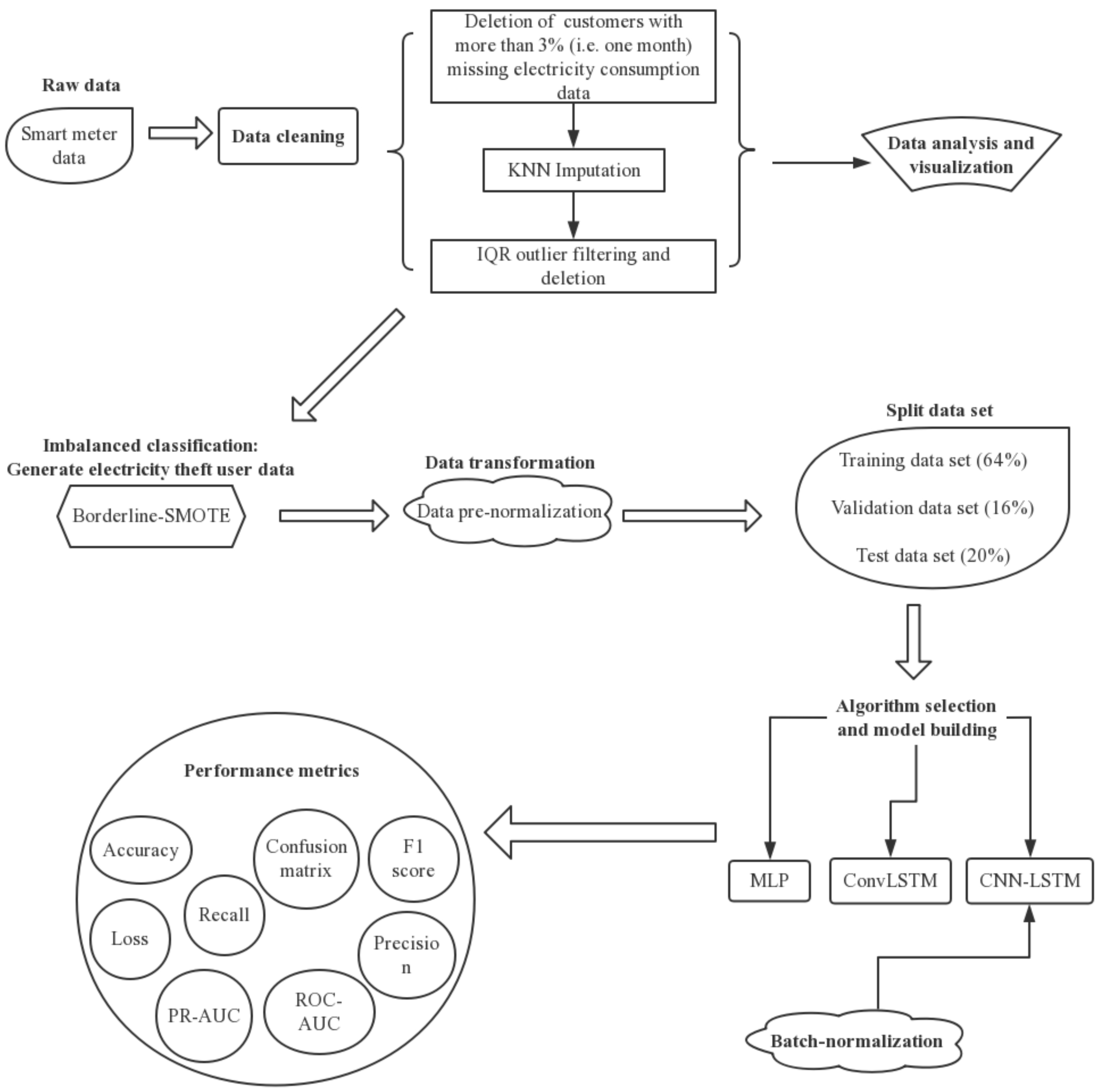


Figure 9: Project process

3.2 Related Configuration of Project Platform

Hardware Configuration
Computer model: MacBook Pro (16-inch, 2019)
Operating system: macOS Big Sur (Version 11.2.1)
Processor: 2.6 GHz 6-Core Intel Core i7
Memory: 16 GB 2667 MHz DDR4
Graphics: AMD Radeon Pro 5300M 4 GB
Cloud GPU (Paperspace): Free-GPU, 30GB RAM, 8 CPUs
Software Configuration
Python (Version 3.7.6)
Anaconda Navigator (Version 1.10.0)
Jupyter Notebook (Version 6.1.4)
Microsoft Excel (Version 16.46)
Major Python Library
Scipy (Version 1.5.2)
Numpy: (Version 1.19.4)
Matplotlib: (Version 3.3.2)
Pandas: (Version 1.1.1)
Sklearn: (Version 0.23.2)
TensorFlow (Version 2.4.0)
Keras (Version 2.4.3)

Table 1: Project platform configuration

4. Problem Analysis and Data Preparation

The input target for machine learning is data. Before applying a data set to machine learning modelling, data preparation is the first and significant aspect. Different experimenters understand and process data set differently, which can make a difference to the state of the data set. The state of the data set has a direct impact on the performance measures, such as accuracy, of the machine learning model predictions. In real life, the data set obtained by experimenters is often incomplete and accurate. For example, the electricity consumption data required for this project is often erroneous and noisy [12]. The process of data preparation can also be referred to as pre-processing of the data, which is a data mining technique [31]. Before predictive modelling, it still needs to perform important aspects such as data cleaning, dealing with imbalanced classification, data transformation and feature extraction. This will make the data fit the model and algorithm more effectively and reduce the negative issues of model failure or low accuracy during modelling due to data leakage or over-fitting and similar issues. At the same time, this project is still guided by the data for the modelling prediction problem. Therefore, improving data accuracy and the accuracy of model predictions remains a vital issue to always think about.

Data preparation includes but is not limited to:
1. Preliminary and visual analysis of data. Statistical as well as graphical and charting methods are used to analyse data for differences, anomalies, potential patterns, etc.
2. Data cleaning. Imputation of missing values, outlier and anomalous data processing, etc.
3. Imbalanced classification. When dealing with classification problems, the data set often has an unbalanced number of variables in each category, which can affect the experimenter's understanding of the evaluation metric of the machine learning model. Therefore, this requires a balancing of the predicted categories.
5. Data transformation. Machine learning requires that the input variables be numerical and that different algorithms have different levels of accuracy with respect to the data. Operations such as standardization or normalisation of the data need to be performed.

In summary, when acquiring a data set, the experimenter should use as much experience, expertise and machine learning techniques as possible to analyse and transform the raw data into a better fit for the model. This is not only a process of transforming the raw data into machine learning understandable data, but also a process of transforming the raw data into something that is as understandable as possible for the experimenter.

4.1 Preview of Raw Data Set

The data set selected for this project was obtained from real electricity consumption data published by the State Grid Corporation of China (SGCC) [12]. The data set contains the daily electricity consumption in kilowatt hour (kWh) of 42,372 customers between 1 January 2014 and 31 October 2016 (1034 days). 38,757 of these customers are normal electricity users (labeled 0) and 3,615 are customers who have been identified as electricity thieves (labeled 1).

See Appendix A for a detailed description of the data set.

Anomalies in the data can be found by looking at the statistical information in the raw data:

1. The Count of daily electricity consumption for all users varies greatly, and some of the 25th Percentile is almost zero, indicating that the data set contains a large number of missing or zero values.

2. The maximum values of daily electricity consumption are unusual, with some of the maximum daily consumption exceeding 10,000 kWh, however, the data and averages of daily electricity consumption are very much in line with the habits of residential electricity consumption. It can be analysed that this data set should contain a small amount of information on industrial or commercial electricity consumption, or that the data collection is anomalous due to a malfunctioning energy metering device. The billing and data collection methods for industrial electricity consumption also differ from those for general electricity consumption, as do the patterns of behaviour and patterns of electricity consumption between the two [32]. However, these users will be retained in order to maintain the authenticity of the data set and to validate the compatibility of the model. This issue can be dealt with at a later date when the data is transformed.

3. The standard deviation fluctuates over time. This indicates that there are seasonal fluctuations in this electricity consumption, or that it is affected by outlier.

The main anomalous states of the data set can be summarised as following table 2:

SGCC data set

Description	Quantity	Class Tag	Time of Duration	Number of Days
Normal users	38757	0	1 January 2014 to 31 October 2016	1034
Electricity theft user	3615	1		
Total user	42372	/		

Total Number of Data	Amount of Missing Values	Amount of Zero Values
43812648	11233528	5788603

Table 2: Raw data status

A summary of the anomalies in the raw data reveals that missing values account for 25.6% of the total data, and zero values account for 13.2% of the total, for a total of 38.8% of the anomalous data. As electricity consumption contains time continuity, it is time series data. Therefore, in order to maintain the integrity and inherent regularity of the data structure, zero values should also be classified as missing values and filled in.

4.2 Electricity Data Cleaning

Data cleansing is an important branching aspect of machine learning, where messy data content and structure can directly lead to biased analysis and failed predictions. As Kazil & Jarmul (2016) [34] point out, data cleaning is not the most appealing aspect of machine learning, but it does form an essential part of data collation wrangling. The analysis of the data can reveal anomalies such as missing values and outliers, or even redundant data with poor data correlation. The screening, analysis and processing of these anomalies are what needs to be done in this session.

4.2.1 Missing Data Filtering

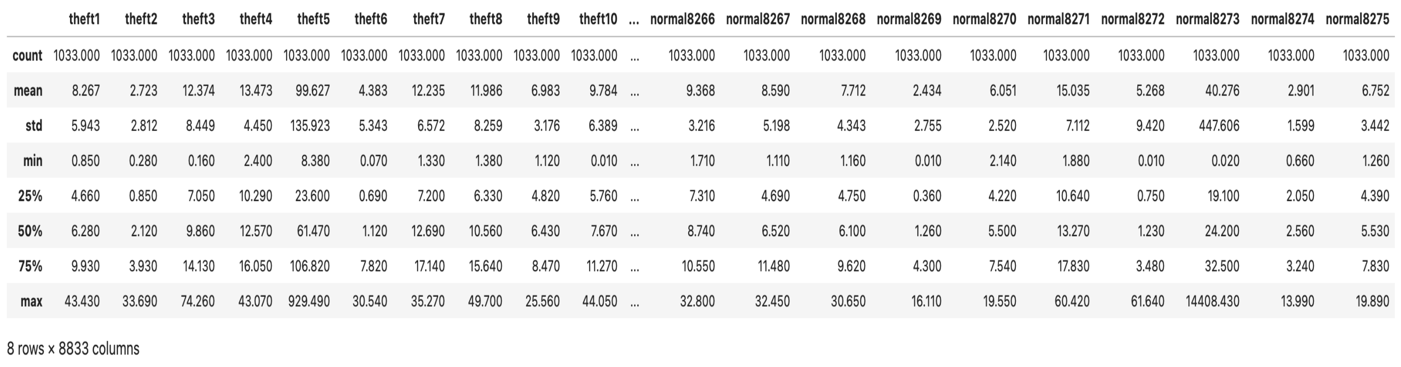
Missing data and anomalies in the SGCC data set can arise for a number of reasons, such as a failure of the metering facility during collection, transmission or storage. Firstly, there are a large number of missing and zero values in the data set (hereafter referred to as missing data). It can be counted the proportion of missing data per user on a user-by-user basis and set a rejection baseline of 3%, i.e. users with more than 1 month of missing data will be removed. This is to retain the maximum authenticity and objectivity of the raw data and model results, taking into account the experimental time required for the project and the performance of the experimental equipment. The minimum threshold of missing data is also used for monthly data in order to retain certain characteristics of the raw data.

After filtering the users according to the proportion of missing data, 8,883 users were obtained. Of these, 8,275 were normal users (labeled 0) and 558 were electricity theft users (labeled 1). The final data dimension was: daily electricity consumption data (kWh) for 8,883 electricity users between 1 January 2014 and 31 October 2016 (1,034 days).

4.2.2 Missing Data Imputation

In machine learning, missing data imputation is a method for dealing with missing data [35]. It can replace missing data in a data set after identifying the missing values. The study cases [12] [13] [14] [15] [16] use the method of linear interpolation to deal with missing values in the data pre-processing stage, and combine it with the "three-sigma rule of thumb" to deal with outliers.

Firstly, in terms of missing values, it should be considered the authenticity of the replacement data as well as preserving the integrity of the data results. The data set used for this project involves continuous missing data, so KNN (k-nearest neighbour) imputation is a more efficient method. Its algorithm is based on similarity and relies on a distance metric, the default of which is the Euclidean distance metric [11]. As stated by Beretta & Santaniello (2016) [36], KNN imputation is effective for handling missing values in continuous and ordered data, and its imputation accuracy and reduction of statistical errors are typically better than 1NN (e.g., two neighbouring data). The main point is that the imputed values are the actual values that occur, rather than the constructed values, which also allows for better preservation of the authentic data structure. This can be achieved using the KNNImputer class in the Scikit-learn library, with a default of n_neighbors of 5. After processing the missing values, it can be transposed the data and view a statistical description of the data set on a user-by-user basis, as shown in figure 10:



	theft1	theft2	theft3	theft4	theft5	theft6	theft7	theft8	theft9	theft10	...	normal8266	normal8267	normal8268	normal8269	normal8270	normal8271	normal8272	normal8273	normal8274	normal8275
count	1033.000	1033.000	1033.000	1033.000	1033.000	1033.000	1033.000	1033.000	1033.000	1033.000	...	1033.000	1033.000	1033.000	1033.000	1033.000	1033.000	1033.000	1033.000	1033.000	1033.000
mean	8.267	2.723	12.374	13.473	99.627	4.383	12.235	11.986	6.983	9.784	...	9.368	8.590	7.712	2.434	6.051	15.035	5.268	40.276	2.901	6.752
std	5.943	2.812	8.449	4.450	135.923	5.343	6.572	8.259	3.176	6.389	...	3.216	5.198	4.343	2.755	2.520	7.112	9.420	447.606	1.599	3.442
min	0.850	0.280	0.160	2.400	8.380	0.070	1.330	1.380	1.120	0.010	...	1.710	1.110	1.160	0.010	2.140	1.880	0.010	0.020	0.660	1.260
25%	4.660	0.850	7.050	10.290	23.600	0.690	7.200	6.330	4.820	5.760	...	7.310	4.690	4.750	0.360	4.220	10.640	0.750	19.100	2.050	4.390
50%	6.280	2.120	9.860	12.570	61.470	1.120	12.690	10.560	6.430	7.670	...	8.740	6.520	6.100	1.260	5.500	13.270	1.230	24.200	2.560	5.530
75%	9.930	3.930	14.130	16.050	106.820	7.820	17.140	15.640	8.470	11.270	...	10.550	11.480	9.620	4.300	7.540	17.830	3.480	32.500	3.240	7.830
max	43.430	33.690	74.260	43.070	929.490	30.540	35.270	49.700	25.560	44.050	...	32.800	32.450	30.650	16.110	19.550	60.420	61.640	14408.430	13.990	19.890

8 rows × 8833 columns

Figure 10: Descriptive statistics of missing values are processed (overview)

Now that there are no more missing data in the data set, outliers need to be identified next. Outliers are observations that are different from the mainstream data in that they are unique, rare or do not match in some way [37]. However, outliers may exist as 'false' outliers. For example, in a particular experiment, age needs to be used as the observed variable. Suppose that the age variable in some samples appear to have a value of 1000, which is clearly a true outlier that can be judged by common sense. Outliers in different fields require a combination of experience, common sense and statistical methods to identify the true outliers. In the case of electricity consumption data, however, trends in electricity consumption are characterised by sharp, peak, flat and valley, as well as seasonal and holiday influences that can result in 'false' outliers. Because fluctuations in electricity consumption data may be rare or unique, but are real and normal values, different households also have different characteristics. As this data set is used for binary classification, it requires trends, underlying patterns or seasonal characteristics to distinguish between electricity-theft user and normal electricity user. For each customer over a period of 1034 days, empirical and statistical analysis can be combined to treat the true extreme outliers and retain the 'false' outliers.

In this step, boxplot allows for better screening of outliers. Boxplot can also be called Box-whisker Plot. It actually uses the quantile of the data to identify the outliers in it. The boxplot shows the distribution of data based on a summary of five numbers (minimum, first quartile (Q1), median, third quartile (Q3), and maximum) and the maximum value is Q3 + 1.5*IQR and the minimum value Q1 - 1.5*IQR [38]. This is shown in figure 11:

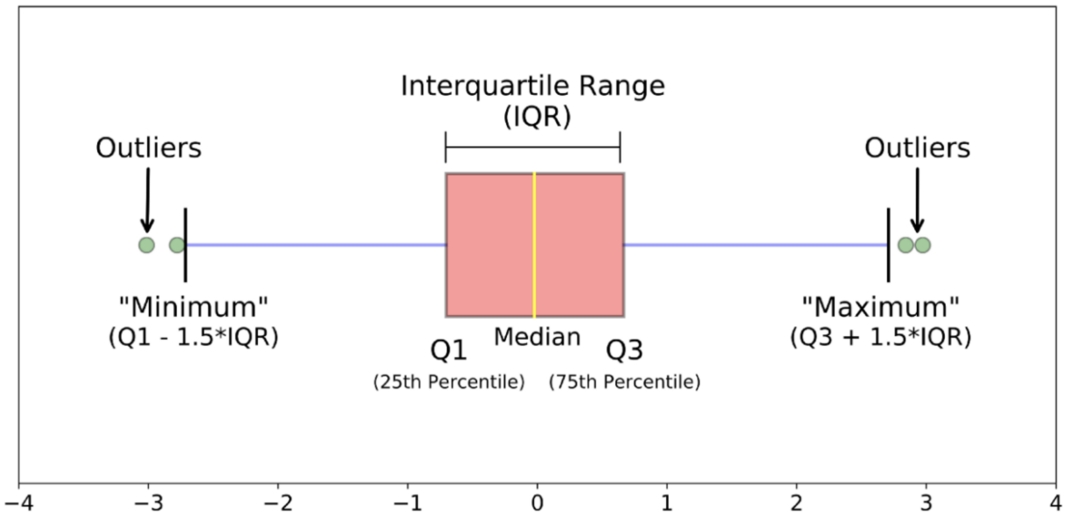
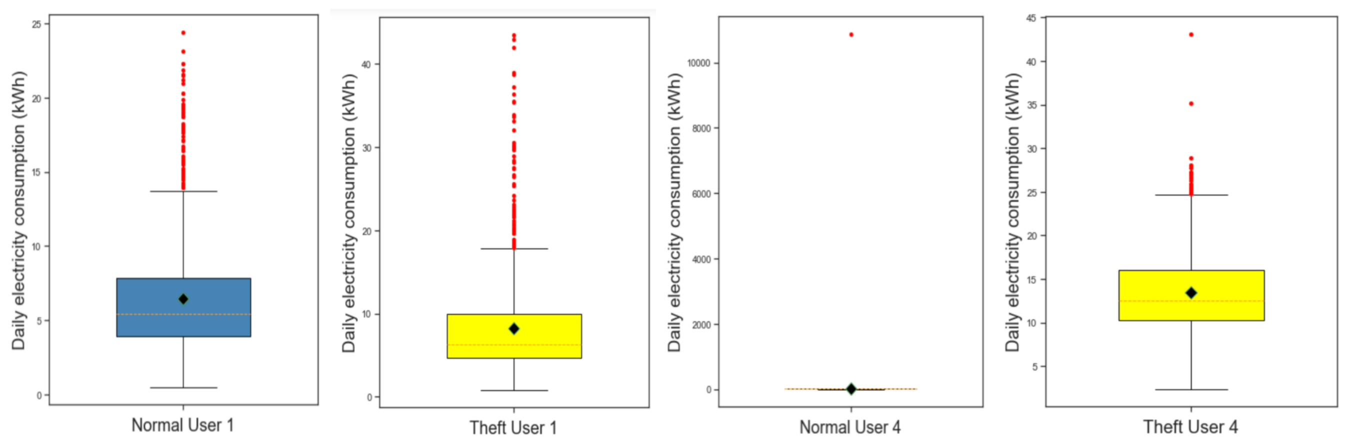


Figure 11: Description of boxplot [38]

However, due to the characteristic nature of electricity consumption data, the statistics can be conducted in such a way as to create 'false' outliers. According to Dawson (2011) [39], Q3 + 1.5*IQR/Q1 -1.5*IQR can be defined as a minor or moderate outlier and Q3 + 3*IQR/Q1 -3*IQR as an extreme outlier. By screening the minor and extreme outliers for all users over a period of 1034 days, a partial sample is shown in figure 12 and 13:



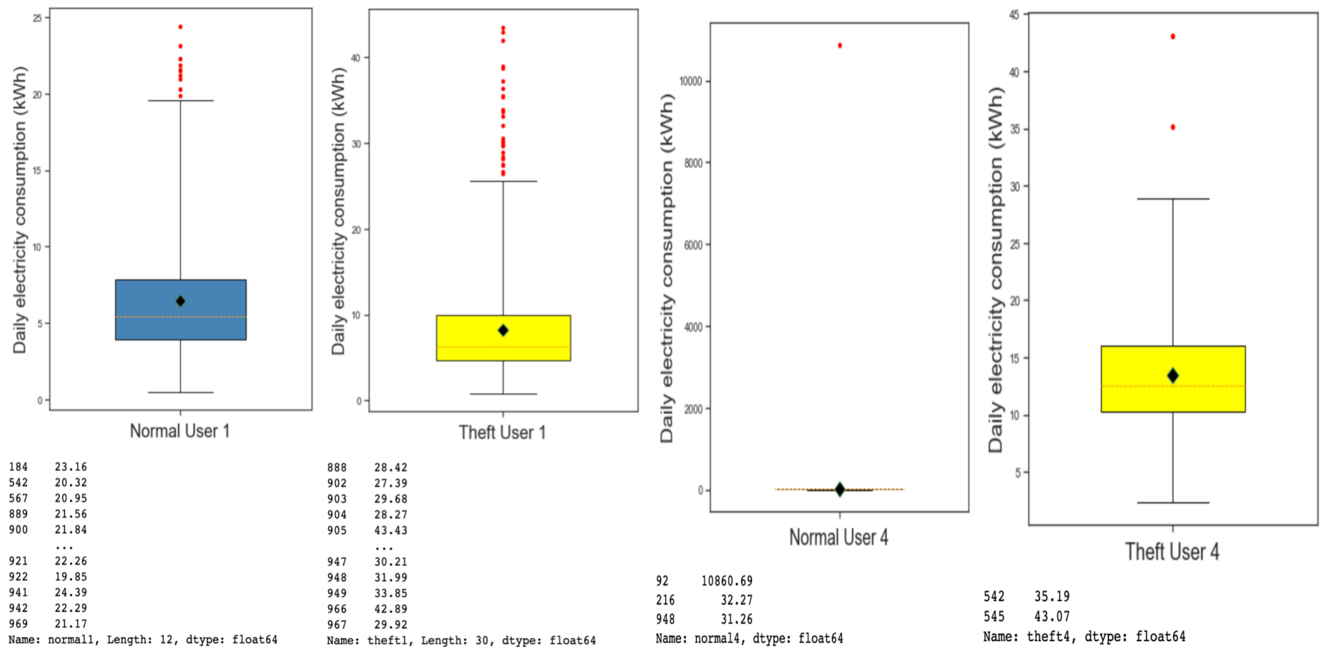
```
183     19.53
184     23.16
191     19.59
200     14.61
206     17.62
        ...
998     15.77
1000    14.95
1004    17.41
1006    14.78
1028    14.07
Name: normal1, Length: 76, dtype: float64
```

```
834     18.59
836     21.56
881     18.34
883     18.02
886     18.87
        ...
967     29.92
968     20.30
979     18.14
980     18.41
1006    18.54
Name: theft1, Length: 67, dtype: float64
```

```
92     10860.69
185       24.42
186       24.89
190       27.26
192       23.73
        ...
945       27.76
948       31.26
949       23.94
964       26.74
966       26.92
Name: normal4, Length: 38, dtype: float64
```

```
45      26.30
220     25.59
239     25.43
542     35.19
545     43.07
        ...
765     27.22
766     25.19
818     26.68
845     28.04
913     28.89
Name: theft4, Length: 18, dtype: float64
```

Figure 12: Minor outliers over 1034 days (1.5*IQR)



```
184    23.16
542    20.32
567    20.95
889    21.56
900    21.84
       ...
921    22.26
922    19.85
941    24.39
942    22.29
969    21.17
Name: normal1, Length: 12, dtype: float64
```

```
888    28.42
902    27.39
903    29.68
904    28.27
905    43.43
       ...
947    30.21
948    31.99
949    33.85
966    42.89
967    29.92
Name: theft1, Length: 30, dtype: float64
```

```
92     10860.69
216       32.27
948       31.26
Name: normal4, dtype: float64
```

```
542    35.19
545    43.07
Name: theft4, dtype: float64
```

Figure 13: Extreme outliers over 1034 days (3*IQR)

By comparing and observing the four sample users, it can be seen that the number of extreme outliers is reduced compared to the number of minor outliers, but still has a certain number of extreme outliers. A preliminary analysis can first be made by electricity consumption. For example, in 'Normal user 4', the boxplot shows anomalies, with one data greatly deviating from the data group with a value of 10,860.69 kWh per day. Combined with the overall customer electricity consumption trend and practical experience analysis, such a large discrepancy is unusual or objectively non-existent in real life and would be due to a fault in the collection or statistical equipment. It can treat outliers such as these as real outliers to be dealt with. The observation of other sample customers shows that the theoretical maximum value of extreme outliers is no more than 43 kWh per day. This can be explained in terms of behavioural characteristics of electricity consumption, where there is self-subjectivity in consumption and where the data set also contains suspected commercial or industrial electricity users or electricity thieves. So, such outliers can be ignored, as some minor or moderate outliers it is objectively present.

Removing them will certainly make the machine learning results more accurate, but this seemingly good result may be false and does not reflect the most realistic and objective results. So, a more sensible approach is to combine the results of the statistics with the analysis of objectively existing electricity consumption behaviour and daily electricity consumption characteristics (for example, actually comparing the data on the basis of the statistics to see if the average or peak interval of electricity consumption per household over 1034 days shows extremes with outlier multiples). Along these lines, all objectively existing outliers need to be retained and the true extreme outliers removed wherever possible.

After identifying the extreme outliers, the actual number of outliers to be removed is 235, which are characterised by deviations of several hundred or even a thousand times from the main data set. After the initial data cleaning, the data visualisation can be analysed and outlined using a more realistic data set.

4.3 Electricity Data Visualization

More detailed analysis of the data will allow machine learning predictions to be presented with better results. The visualisation of the data allows the underlying cyclical patterns and trends within the data set to be more clearly presented, and allows the difference between normal and theft users to be better differentiated. The visualisation process also provides more ideas for later data feature extraction and provides a better basis for predictive modelling. Here, the combination of Matplotlib and Seaborn libraries in Python can enhance the plotting effect.

Firstly, the data set contains two categories of normal users and theft users. Due to the sheer size of the data dimensions, representative data can be selected, the respective total average of the two categories of user data for analysis. The following figure 14 and 15 are available for both categories of users on a yearly and quarterly basis:

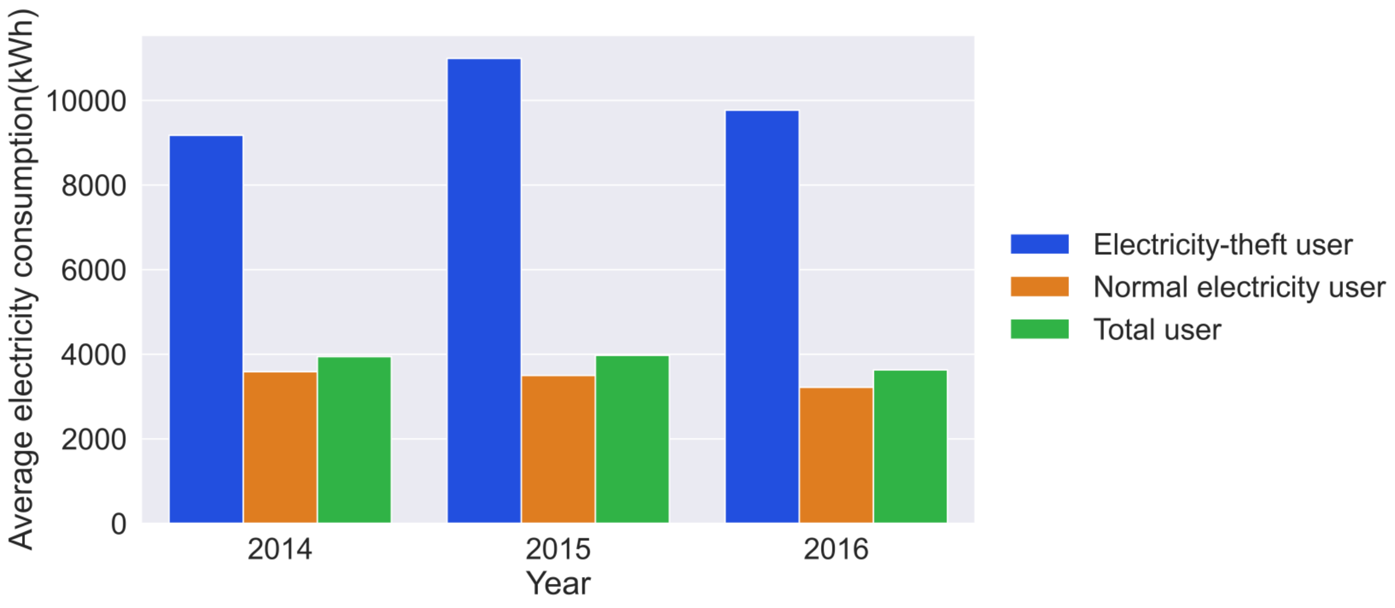


Figure 14: Average annual electricity consumption

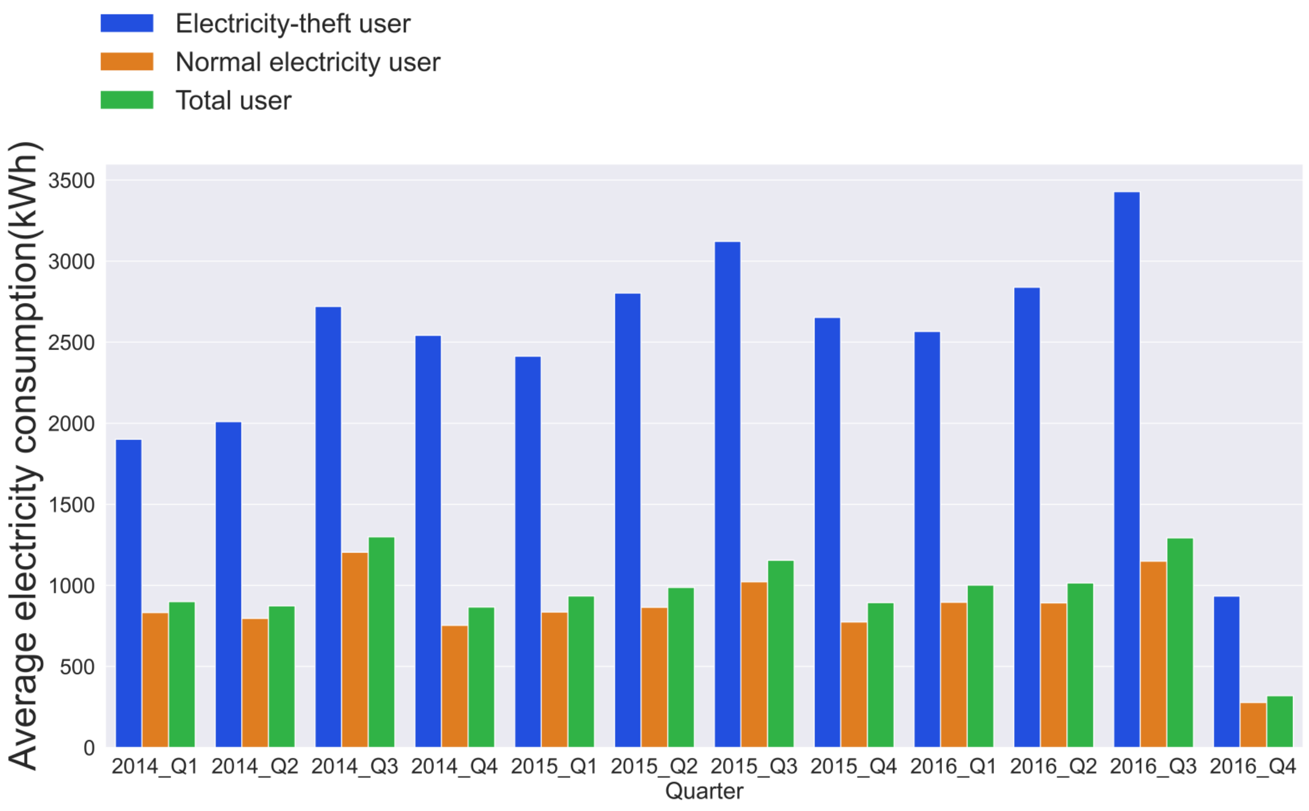


Figure 15: Quarterly average electricity consumption

By looking at the two graphs, it can first be seen that the average electricity consumption in the fourth quarter of 2016 was very small. This is because there is only one month of data for the fourth quarter of 2016, so this anomaly can be ignored. However, in the yearly graph it can be seen that the average electricity consumption of electricity thieves is more than twice that of normal users, while the fluctuations between normal and total users are small and have a stable ratio. The quarterly graphs also show that the fluctuations and ratios between normal and total customers are very regular. It is worth noting that in the third quarter of each year there is a peak in electricity consumption for both categories, which is very much in line with real life. This is because June to September is the summer months in China, the peak period for air conditioning [32]. However, electricity theft users still maintain a high average electricity consumption in the quarterly graph and the trend is upwards every year. This is in stark contrast to the smooth cyclical nature of normal users. This also reflects the fact that the behavioural characteristics of electricity theft users can make some difference.

Due to the complexity and concealment of electricity theft in reality, the ever-evolving ways in which electricity is stolen dictate that the final data collected on electricity consumption is diverse. The total amount of electricity consumed is sometimes not directly used to judge the existence of theft, but should be explored in more detail. The following graph compares the average daily electricity consumption of two types of customers over a period of 1034 days, as shown in figure 16:

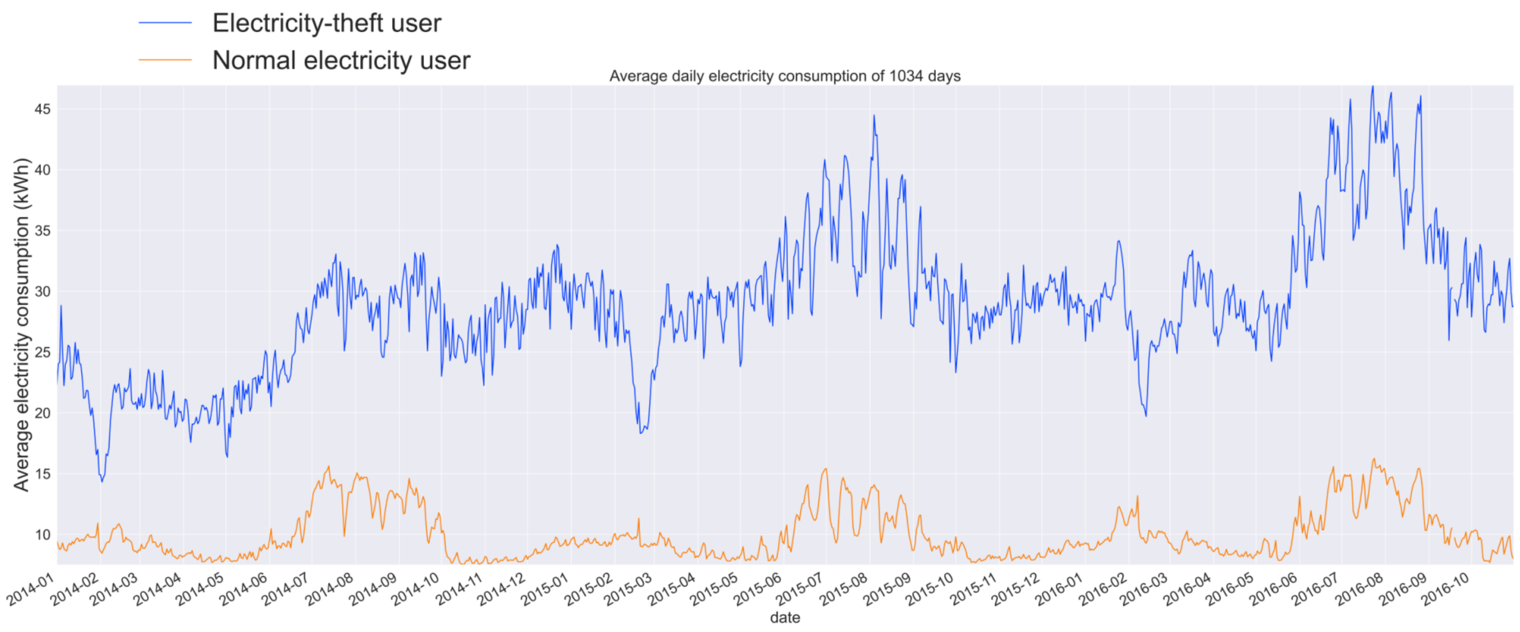


Figure 16: The average daily electricity consumption in 1034 days for the two types of users

The graph above clearly reflects that the average daily electricity consumption of electricity thieves is still greater than that of normal users. On the one hand, normal users have a smooth fluctuating and cyclical pattern, and seasonal specificity is also more regular. For example, the peak period of electricity consumption for air conditioning in summer. As well as a small peak during the Chinese New Year around March each year. On the other hand, in the curve of electricity theft users, the average electricity consumption fluctuates greatly and is not smoothly cyclical. A point worth noting is that electricity theft users also seem to have a certain seasonal pattern, with a peak in the summer months of each year as well. This point suggests that the behaviour of electricity theft users is deceptive and somewhat misleading. However, there is another curious phenomenon in the graph. Around February to March each year (Chinese New Year), there is a clear downward trend in electricity consumption by electricity theft customers and it is close to the average value of electricity consumption by normal customers. There are a number of reasons for this phenomenon, for example, during the Chinese New Year period the electricity thieves may not be stealing or for some reason the thieves need to reduce their electricity consumption during this period, etc.

After the annual and quarterly analysis has been completed, further data can be outlined for both types of users on a monthly and weekly basis. The sample data for the year 2015 can be selected, as shown in figure 17 and 18:

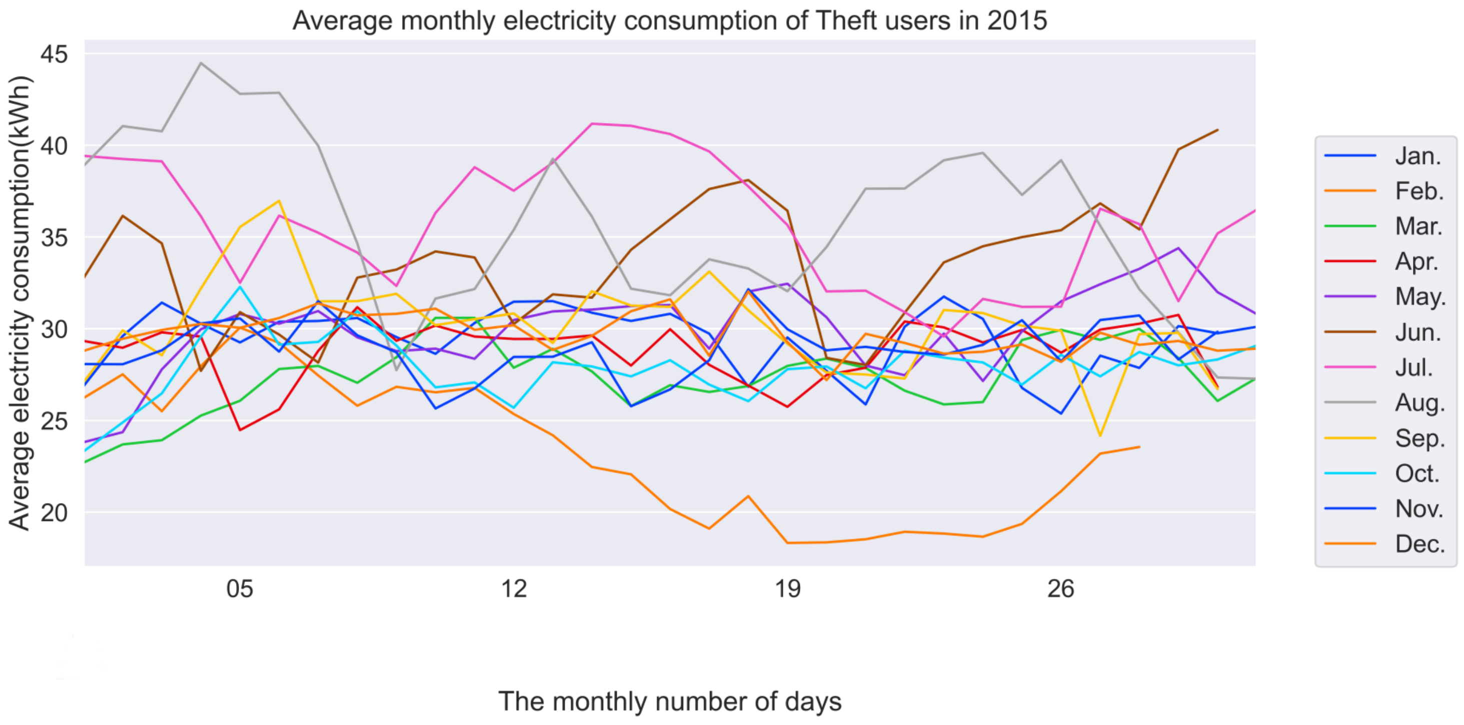


Figure 17: Average monthly electricity consumption of Theft users in 2015

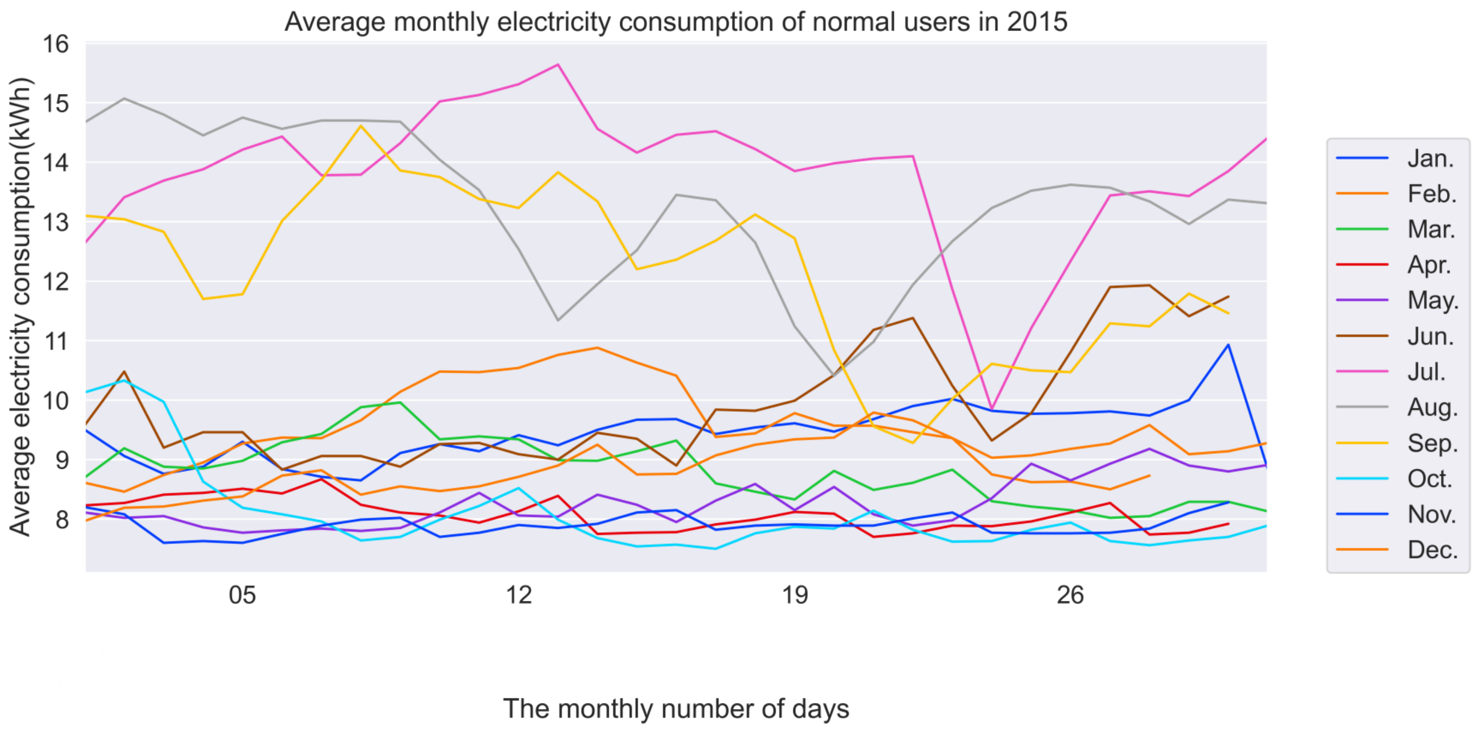


Figure 18: Average monthly electricity consumption of Normal users in 2015

By comparing the two types of customers it is clear that normal customers' electricity consumption data tends to be stable and less volatile in months other than summer, with July, August and September being significantly stronger than other months in terms of consumption and fluctuations. The data for electricity thieves, however, appears unusually chaotic and there is a very sharp decline in December, with the overall trend not conforming to natural patterns. The extraction of monthly features is also an aspect that needs to be considered.

In addition to this, according to Zheng et al. (2017) [12], data from four weeks can be extracted for further analysis, for example, plotting again the comparison of data between the two types of users, as well as plotting Pearson's correlation coefficient (PCC) and Autocorrelation function (ACF). These methods show correlations and potential regularities between the data in each of the two categories of users. The two categories of users can first be plotted again on a weekly basis, as shown in figure 19 and 20:

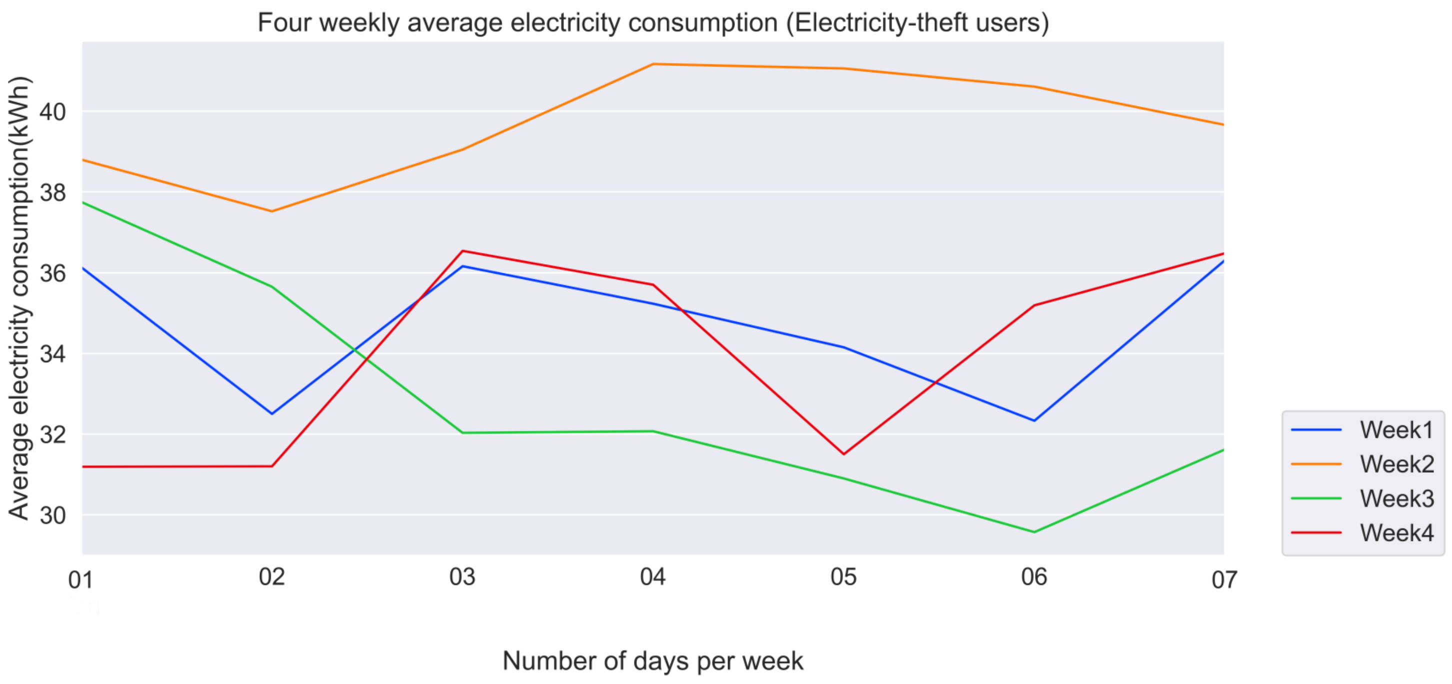


Figure 19: average daily electricity consumption every four weeks (Theft users)

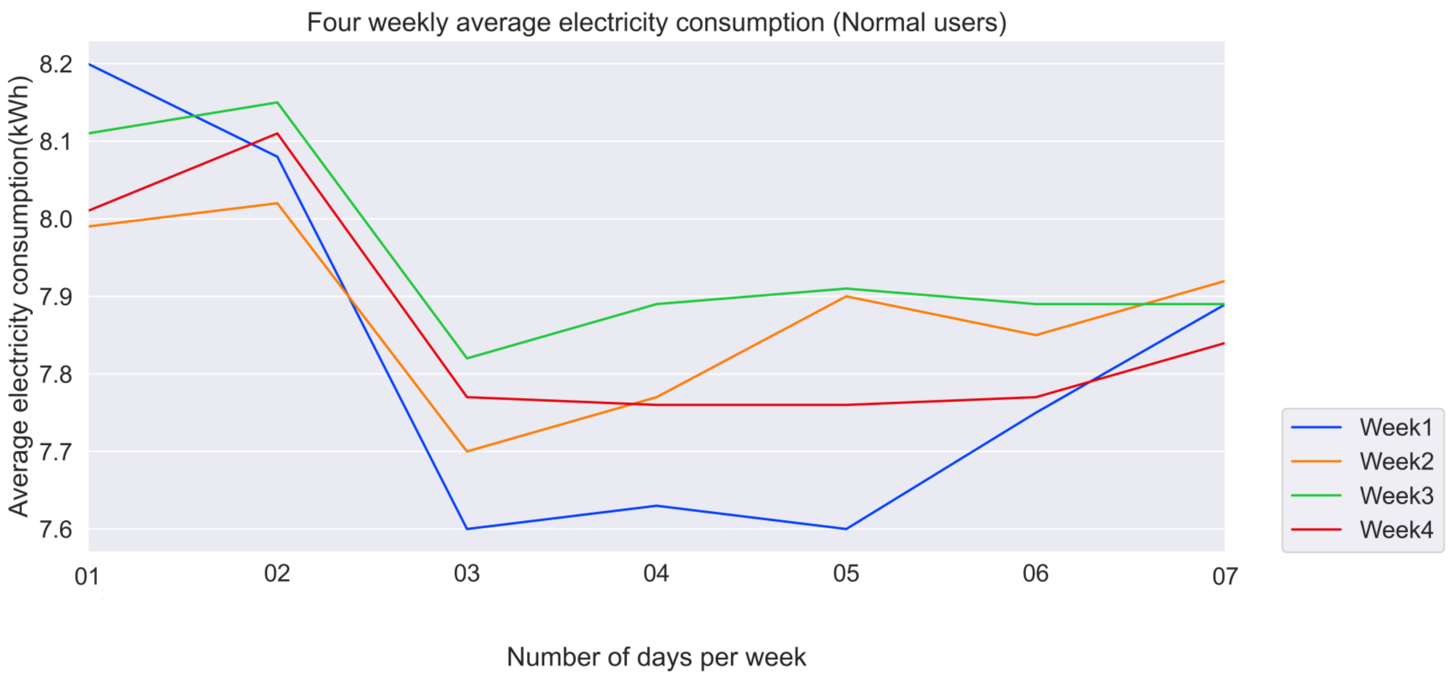


Figure 20: average daily electricity consumption every four weeks (Normal users)

From the graph above it can be clearly seen the regularity of weekly electricity consumption for normal customers. Mondays and Tuesdays are the peaks of electricity use, with Wednesdays falling into the weekly lows of electricity use and then continuing to fluctuate steadily. The electricity thieves, on the other hand, continue to show chaotic electricity consumption behaviour. In fact, while comparing the averages, it also can be carried out similar analyses for the other users in the sample. In general, normal users show good cyclical and seasonal patterns, but electricity theft users continue to have chaotic electricity usage characteristics mixed in. Thus, annual, quarterly, monthly as well as weekly and daily electricity usage characteristics can be used as a benchmark for extracting features.

Next, to explore the correlation between the data, the Pearson's correlation coefficient (PCC) for the two types of customers over the four weeks of data can be plotted, as shown in figure 21 and 22:

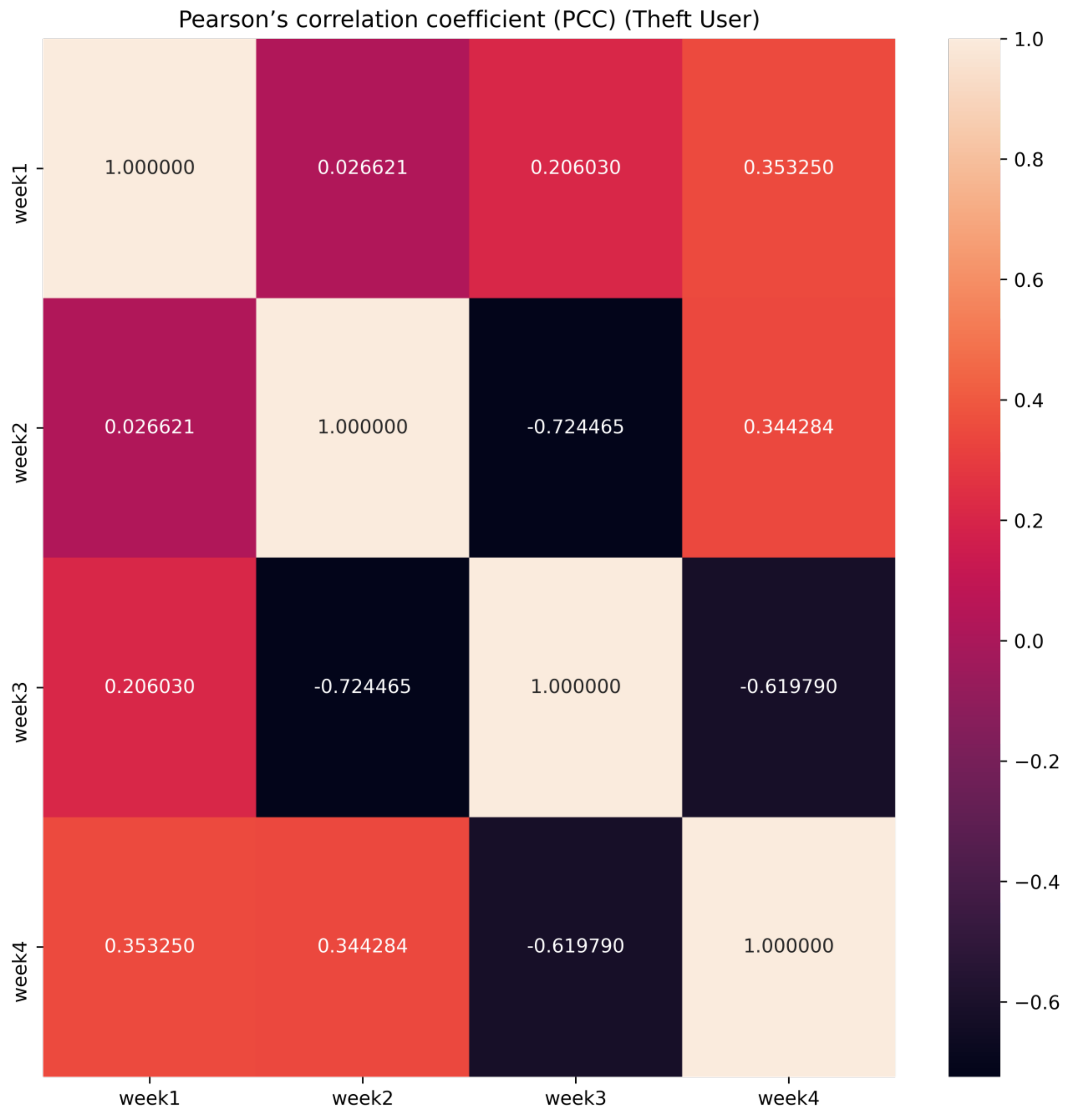


Figure 21: Pearson’s correlation coefficient (PCC) (Theft User)

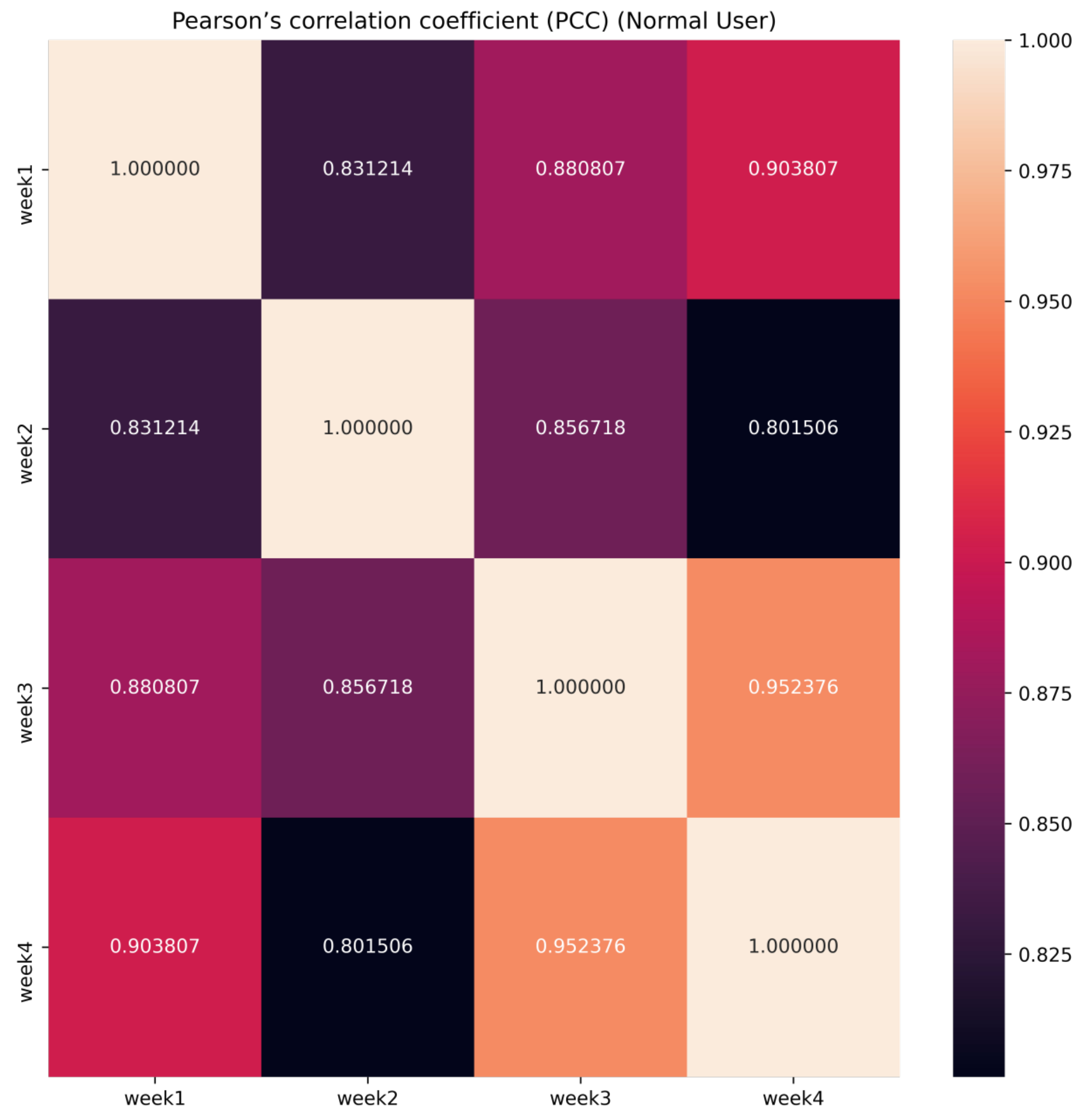
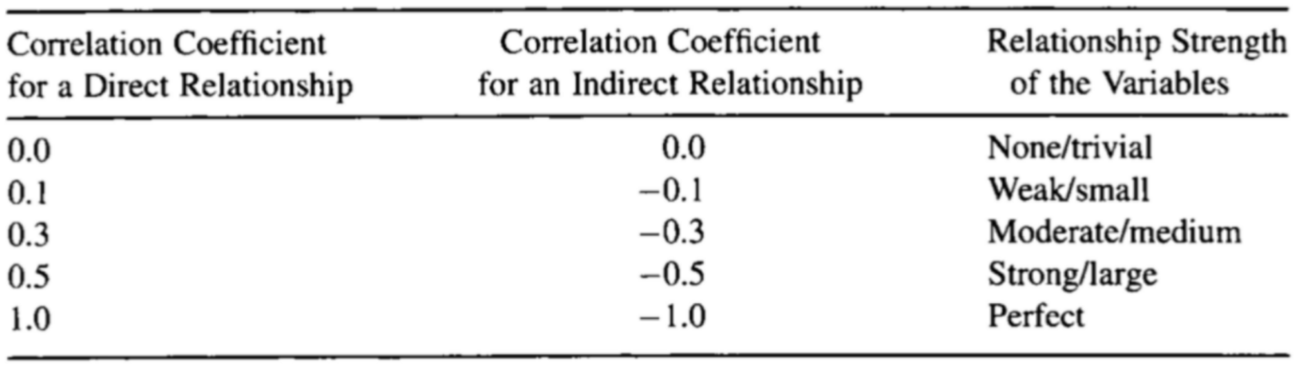


Figure 22: Pearson's correlation coefficient (PCC) (Normal User)

The two figures clearly show that the data correlation of normal users is much stronger than electricity theft user. The correlation coefficient for electricity theft customers does not exceed a maximum of 0.3 and has a certain negative correlation. However, the correlation coefficient for normal customers is generally higher than 0.8 and shows a strong positive correlation. In the PCC, values above 0.5 or below -0.5 represent a relatively significant correlation. Positive values closer to 1 indicate a stronger direct correlation. A negative value and closer to -1 represents a strong indirect correlation [33]. The following figure 23 shows that:



Correlation Coefficient for a Direct Relationship	Correlation Coefficient for an Indirect Relationship	Relationship Strength of the Variables
0.0	0.0	None/trivial
0.1	−0.1	Weak/small
0.3	−0.3	Moderate/medium
0.5	−0.5	Strong/large
1.0	−1.0	Perfect

Figure 23: Description of Pearson's correlation coefficient (PCC) [33]

Similarly, the daily electricity consumption of all normal customers and electricity theft customers for 1034 days can be compared separately, as shown in the following figure 24 and 25:

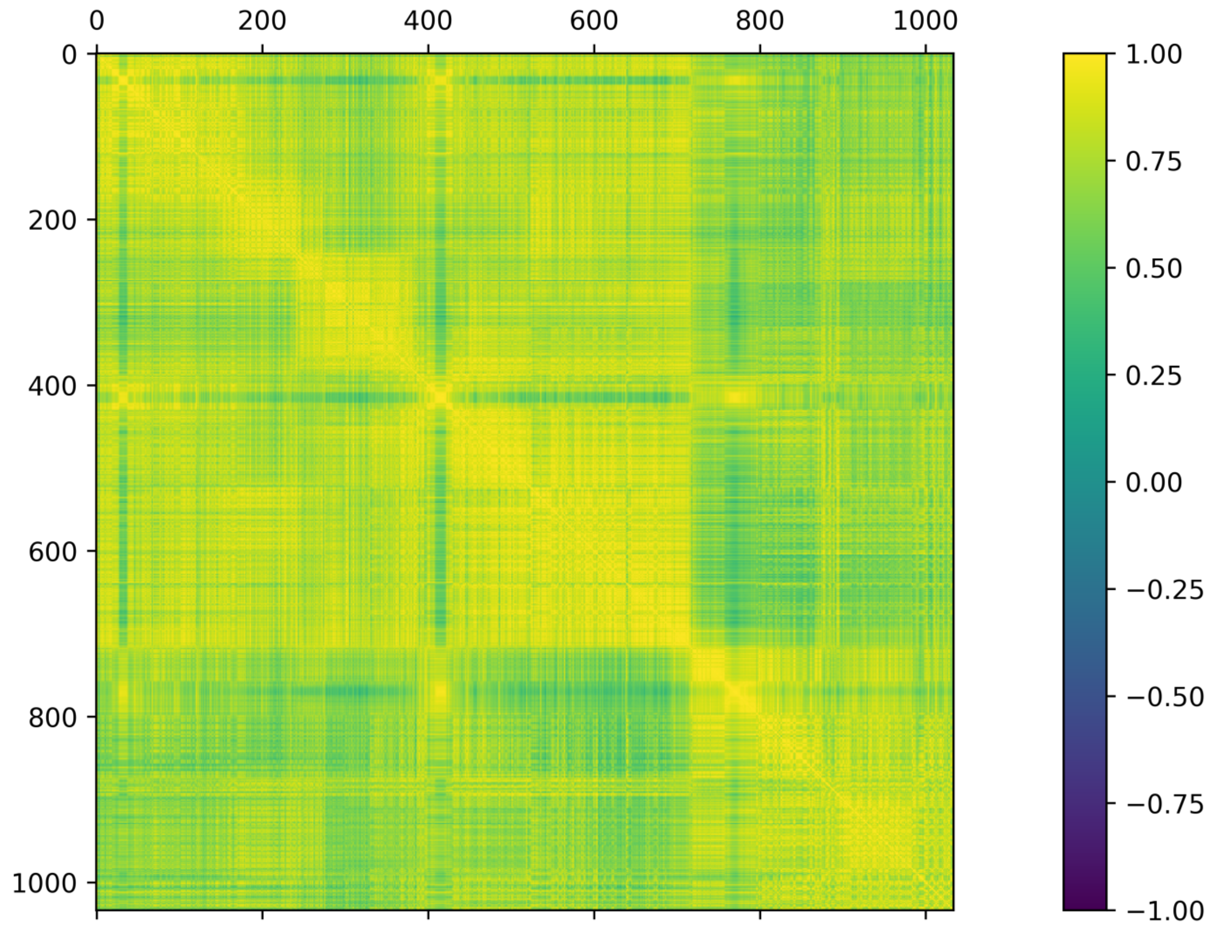


Figure 24: Pearson's correlation coefficient (PCC) (Theft User 1034 days)

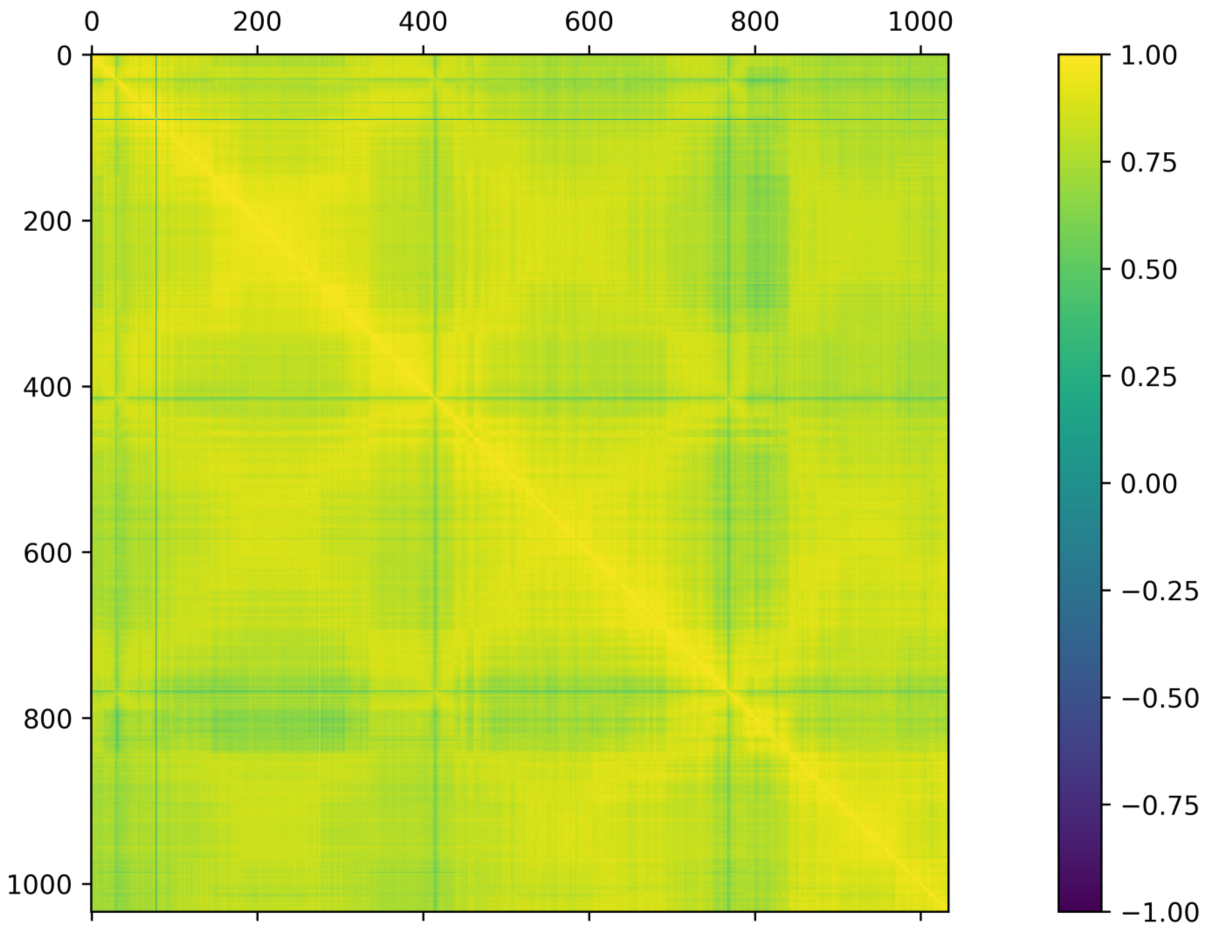


Figure 25: Pearson's correlation coefficient (PCC) (Normal User 1034 days)

By comparing on the raw data, it is also clear that the positive correlation between the daily electricity consumption of normal users is stronger than that of electricity thieves. Moreover, in the PCC diagram for normal customers, it can be seen cross-pointing with directionality. This indicates the potential cyclicality and regularity of the electricity consumption characteristics of normal customers.

In addition, in the Autocorrelation function (ACF) diagram [12] it can be also seen the cyclicality of the average daily electricity consumption of normal users over 4 consecutive weeks, which reinforces the previous analysis. In contrast, electricity theft users do not show strong cyclical characteristics. This is shown in figure 26 and 27:

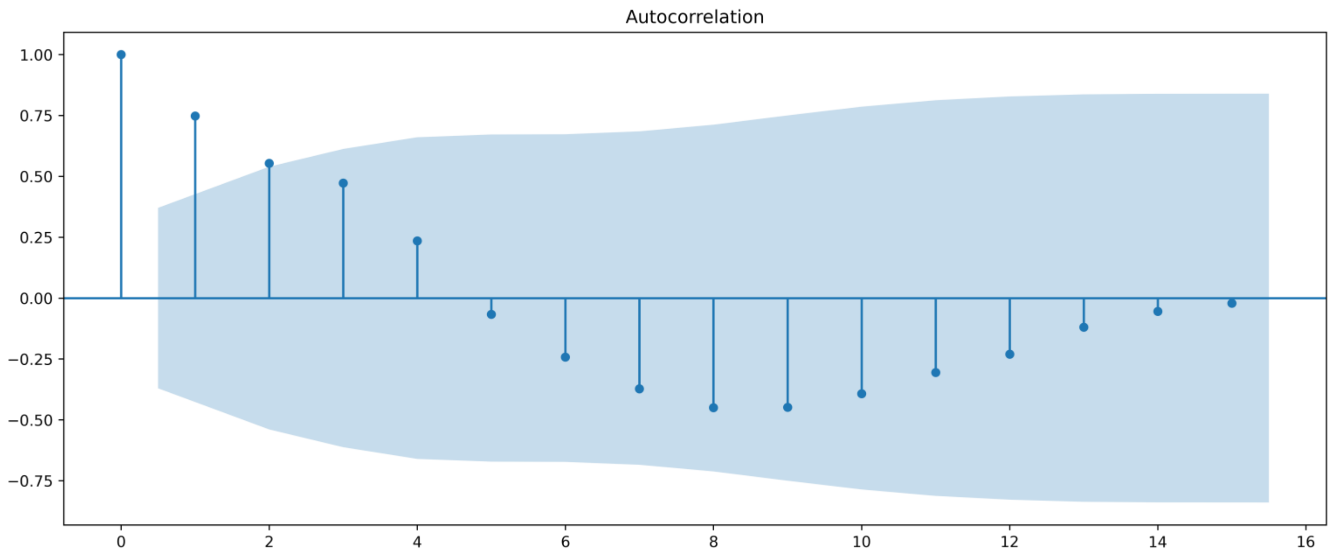


Figure 26: Autocorrelation function (ACF) of Theft User in 4 weeks

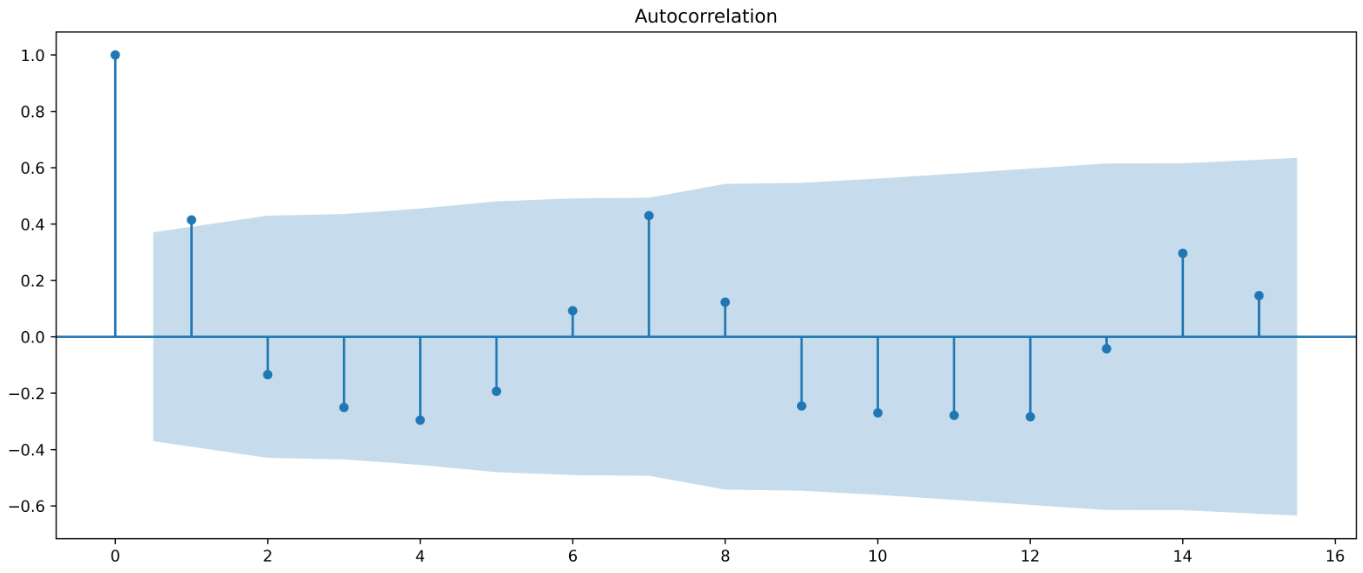


Figure 27: Autocorrelation function (ACF) of Normal User in 4 weeks

The data set has already been observed to have typical cyclical patterns, such as trend and seasonality. In simple terms, the SGCC data set is a seasonally trending time series. Here can use seasonal and trend decomposition using loess (STL decomposition), a time series decomposition method that uses robust locally weighted regression as a smoothing method and is based on LOESS (locally weighted regression) to decompose the time series into decomposition into trend components, seasonal components and residual terms [40]. In a way, the data and features determine the upper limit of machine learning, and the models and algorithms only approximate this upper limit [41]. In theory, an STL decomposition should be performed for each user. For presentation purposes, It can be continued to select the 1034 days of average daily electricity consumption for both types of users for STL decomposition. This is illustrated in figure 28 and 29:

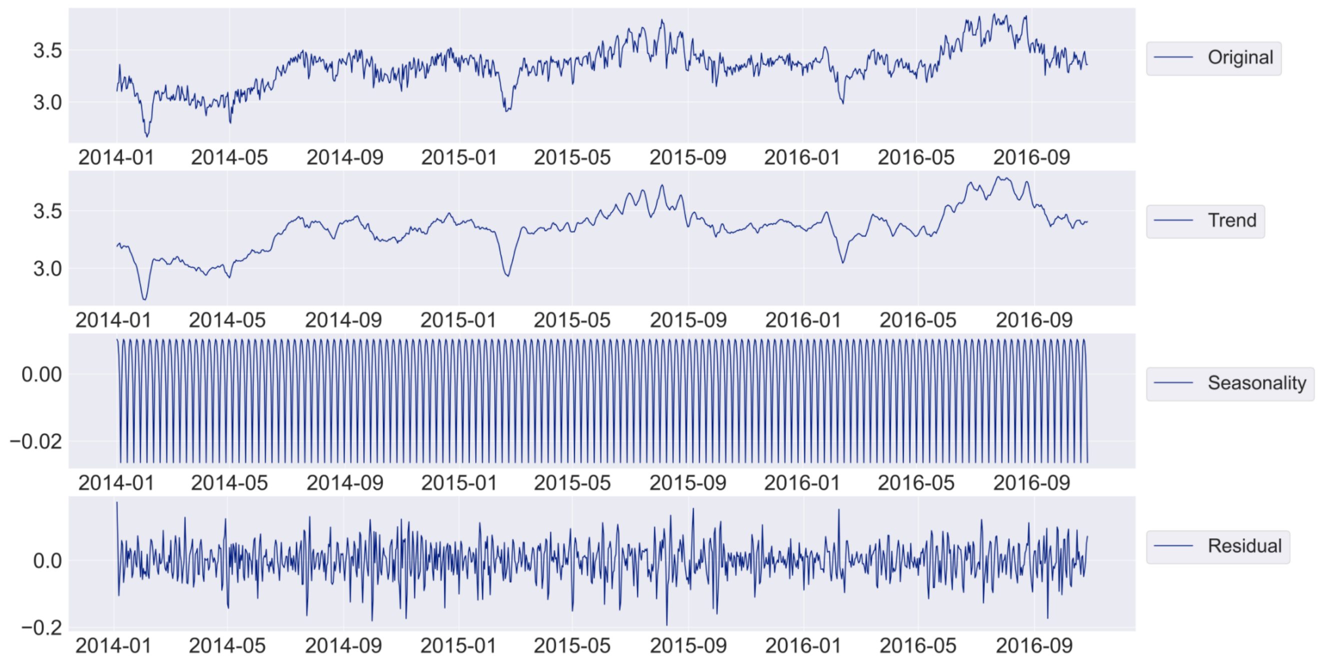


Figure 28: STL decomposition of average daily electricity consumption for 1034 days (Electricity-theft user)

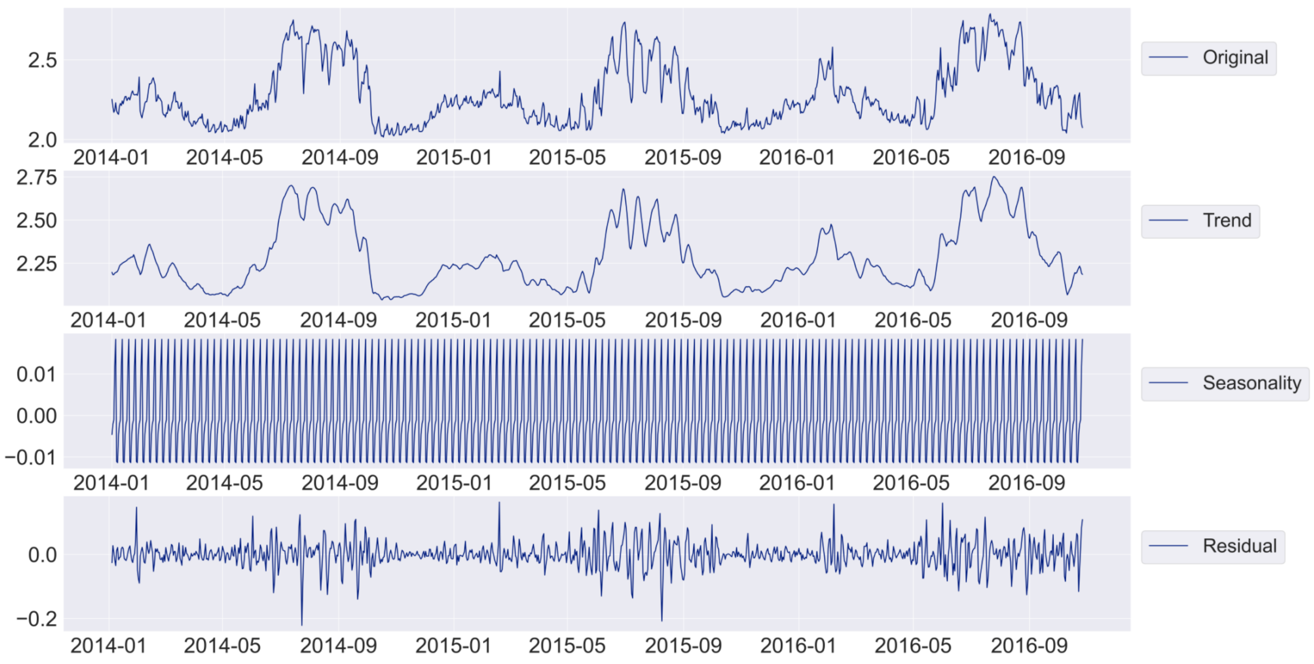


Figure 29: STL decomposition of average daily electricity consumption (Normal electricity user)

The STL decomposition diagram further shows that the trend for normal customers is cyclical, while the trend for electricity theft customers is upward. The two types of customers have very different seasonality. As for the residual data, the seasonality and trend have been removed and are more stable data. For the selection of features, the trend, seasonality and residual data can still be extracted according to different temporal characteristics.

Overall, the data set has been observed and analysed from a number of perspectives, and the distinctive characteristics and behaviour of the two types of customers have been broadly captured. In terms of temporal characteristics, annual, quarterly, monthly and weekly can all produce different potential rates of change. In terms of statistical data, means, minima, maxima and even variances, medians and so on can be used to outline the framework of characteristics for the model that follows. In addition to the raw data base, features can be extracted from the trend, seasonal and residual stable data through STL decomposition. These features, once extracted and transformed, can be better utilised by the model and algorithm. Before proceeding further with feature engineering, another key issue needs to be considered. As this project deals with a supervised learning binary classification problem and the raw data set inherently has an imbalance between the two types of users. In order to predict better results from the model, this needs to be dealt with first.

4.4 Imbalanced Classification Sorting

Imbalanced classification datasets tend to mislead the performance in machine learning. In contrast, different data sampling techniques can better balance the class distribution and train directly on the data set instead of directly modifying the original data [19]. Sampling techniques can be roughly divided into three types: over-sampling, under-sampling and combined sampling technique. The basic pattern is shown in figure 30:

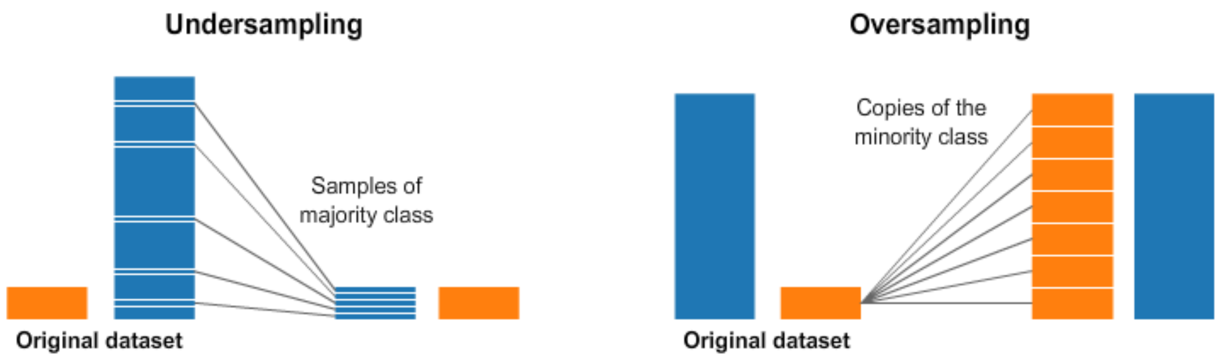
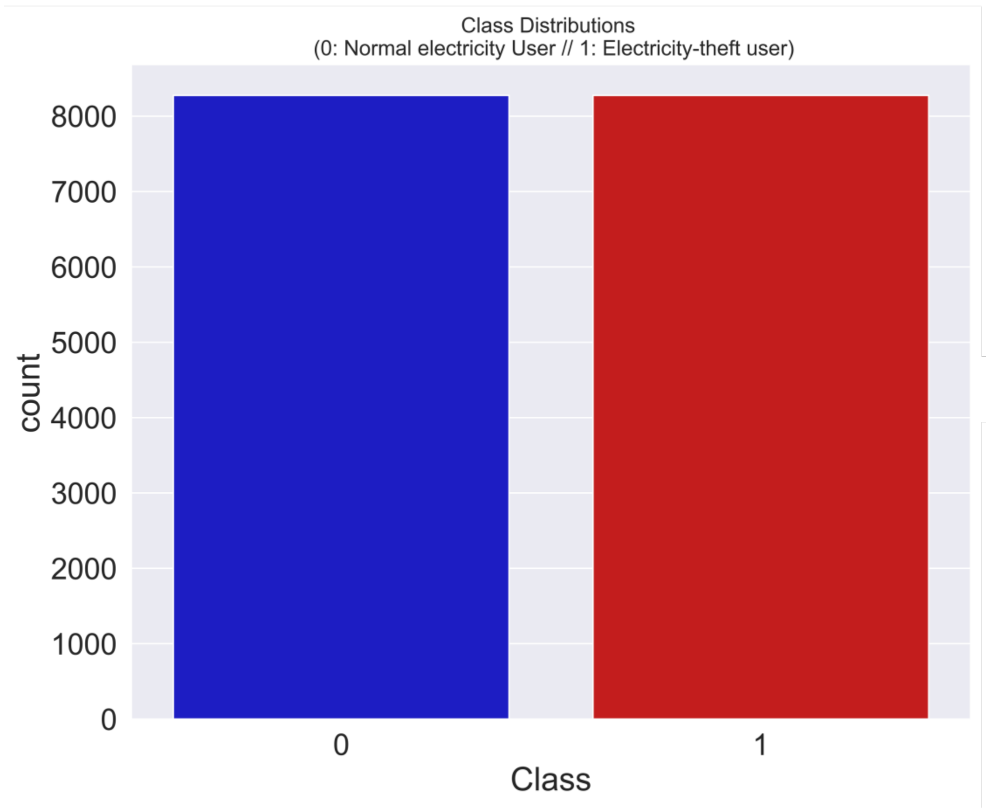


Figure 30: Example of basic sampling pattern [42]

The current imbalance status of the data set is known, with 558 electricity theft customers and 8,275 normal customers. The electricity theft customers represent approximately 6.3% of the total customers and are classified as severely unbalanced. In order to continue to maintain the authenticity of the data set, the oversampling technique is the first thing that needs to be considered. Hasan et al. (2019) [6] point out in their case that ordinary oversampling techniques can allow the model to develop an overfitting state due to the replication of data points. They adopted the SMOTE (Synthetic Minority Oversampling Technique) to generate electricity theft user of synthetic data using minority instances. However, a method called Borderline-SMOTE [43] has shown better performance than SMOTE in research. In simple terms, for example, there is a possibility of overlap between the minority and majority classes in the raw data set or statistical observations of electricity data. SMOTE may confuse the two classes of data, resulting in inaccurate classification data being produced. However, Borderline-SMOTE will classify observations in this minority class as noise points when the data adjacent to the minority class are all in the majority class, and ignore them when generating the data [43]. It is equivalent to creating boundaries in the vicinity of some outliers, which is more conducive to the accuracy of the generated data. The following figure 31 shows the state of the data set after using Borderline-SMOTE balance:



```
Counter({0: 8275, 1: 558})
Counter({1: 8275, 0: 8275})
```

Figure 31: The state of the data set after classification balance

Having obtained the 7,717 electricity theft customers produced by Borderline-SMOTE, the results can be compared with real electricity theft customers and normal customers. The average daily electricity consumption of the three categories of users for 1034 days is shown in figure 32:

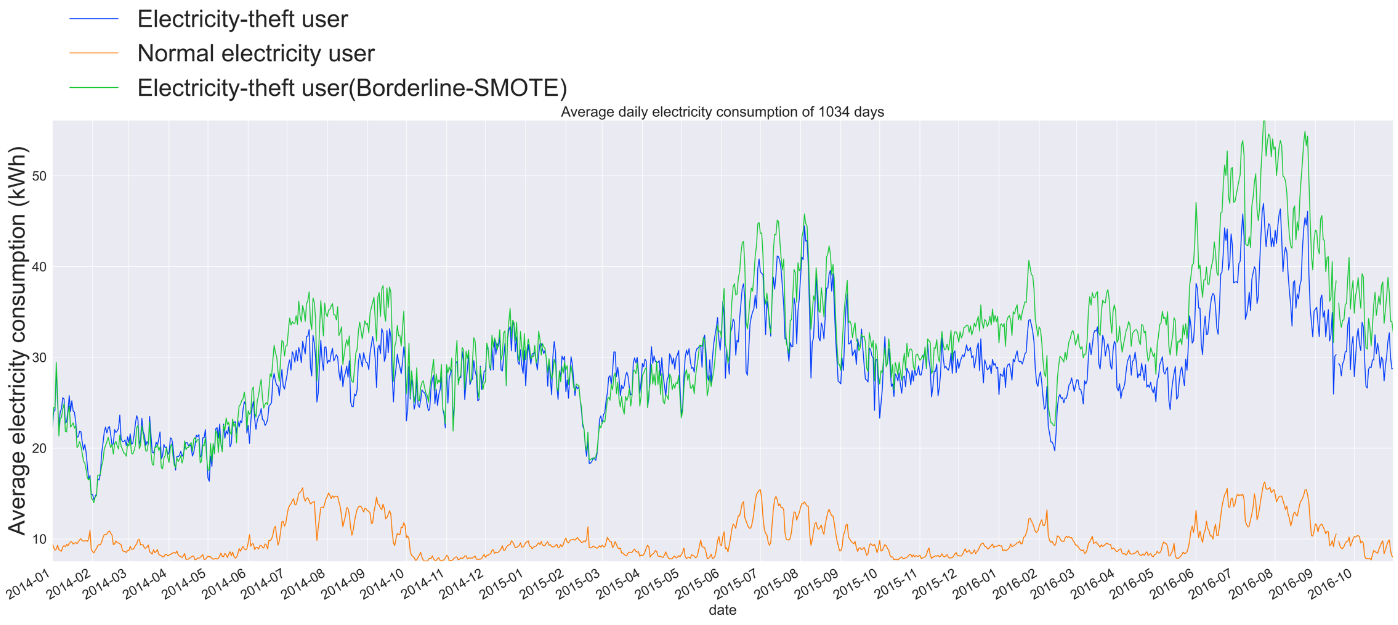


Figure 32: Comparison of electricity theft users produced by Borderline-SMOTE

From the graph it can be observed that the trend of electricity theft users produced by Borderline-SMOTE matches the trend of real electricity theft users and is closer to the electricity consumption data of real electricity theft users. And it does not show any significant overlap with normal electricity users.

4.5 Data Transformation

Normalization rescales the data from the original range so that all values are within the new range of 0 and 1 [17]. The study cases [12] [14] [15] [16] [18] used the 'MAX-MIN scaling method' to normalize the data set. This can be done to make the data fit the neural network better, since there is some numerical variation in the electricity consumption data. This also helps to optimize the core algorithm of the neural network and prevent overfitting. The formula is as follows:

$$f(x_i) = \frac{x_i - \min(x)}{\max(x) - \min(x)} \quad [12]$$

In addition, this project adds a novel data transformation method, batch-normalization, to the ConvLSTM model, which can make training deep neural networks more efficient. As stated by Ioffe & Szegedy (2015) ‘it can accelerate deep network training by reducing internal covariate shift’ [45]. In simple terms, batch-normalization can be added after each layer before the model output layer to optimize the efficiency of model operations.

5. Model Building

The processed data is now ready to be applied to the algorithms and models. Three models are built in this project, MLP (baseline), CNN-LSTM and ConvLSTM. Data modeling is a long and iterative process that first requires a uniform partitioning of the processed data into training, validation and test data sets. The training and validation data can be considered as exercises, while the test data is a way to truly validate the model.

The proportions of the three data sets in this project are, respectively, 64%, 16% and 20%. The data is then matched to the model for prediction through model definition and compilation. The three models belong to supervised models dealing with dichotomous problems, both by training the models and evaluating the performance of the models in identifying normal electricity users and electricity theft users. Since the data set was too large, in order to shorten the project time, 'earlystopping' [47] was applied to three models so that the modeling efficiency could be improved, and the network could be better generalized. It is also convenient to select the best model for prediction. The patience of all models is uniformly 50 and the monitor is val_loss. The modeling process can be divided into the following steps of figure 33:

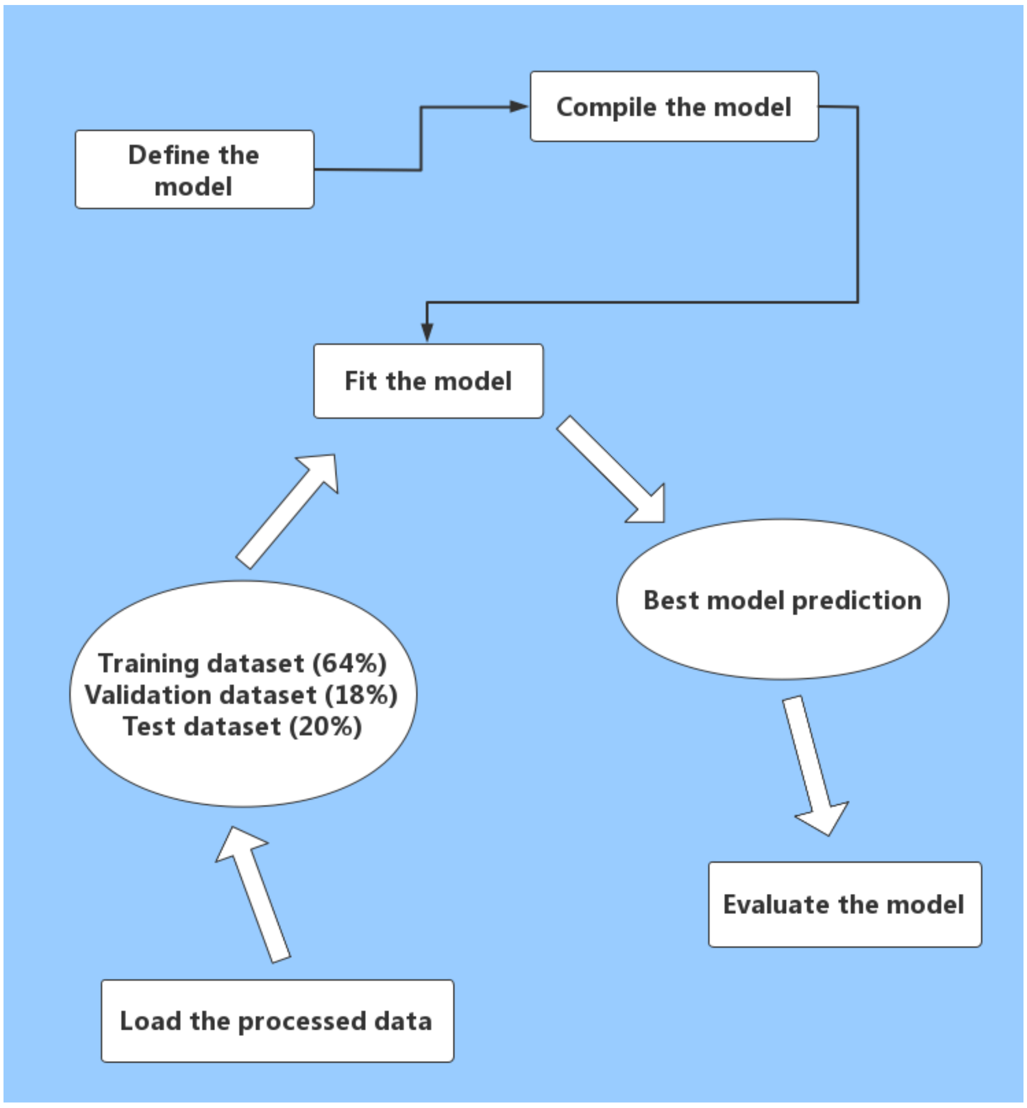


Figure 33: The modeling process

5.1 MLP

Although the MLP is the baseline model, dropout [48] is still used in order to ensure the optimal state of each model. It is used to perform adaptive regularization to prevent overfitting of the model. Since no time series are implicated in this model, the unit used for feature fitting is days. The model architecture and parameters are as following figure 34:

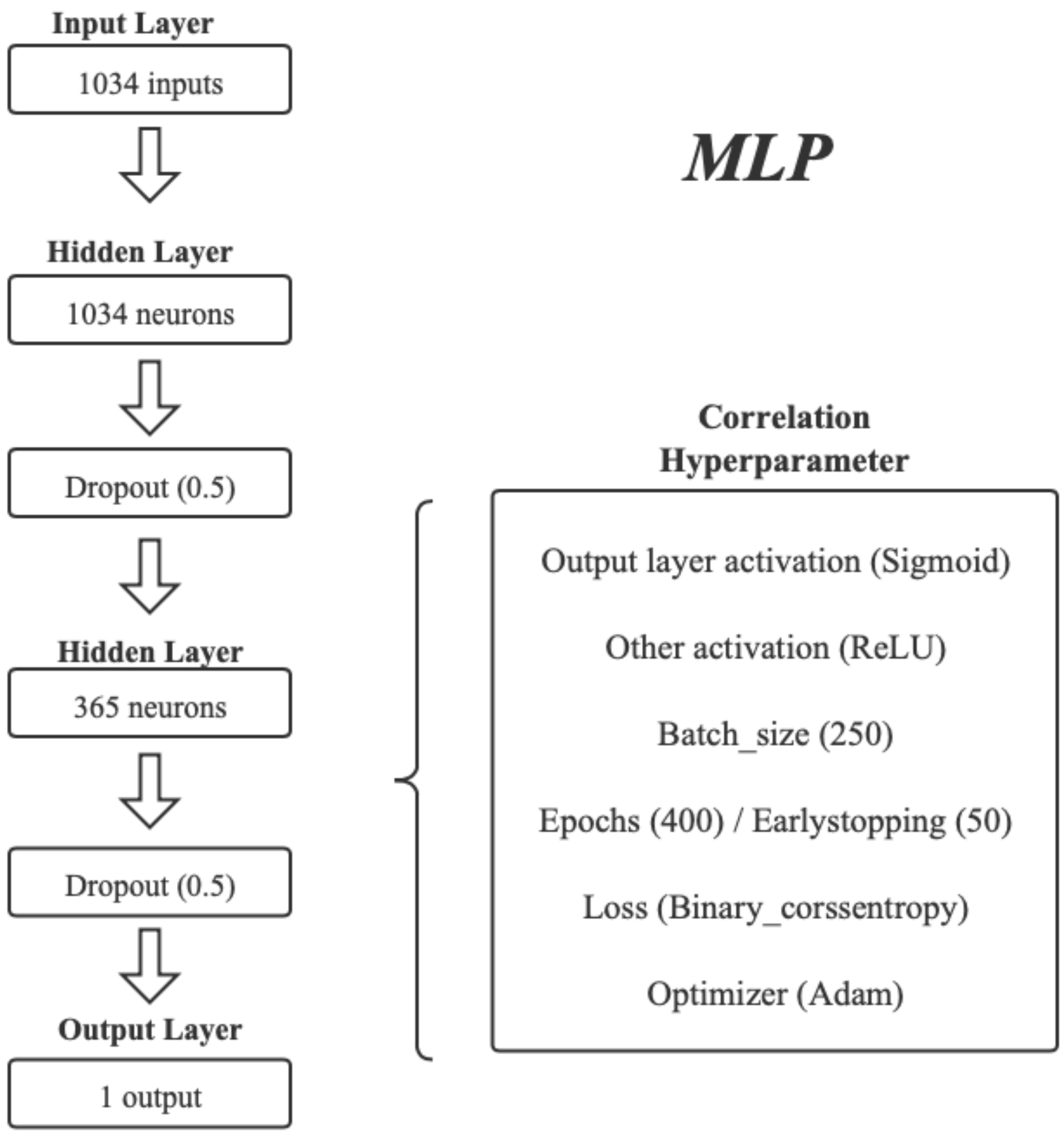


Figure 34: MLP model architecture and hyperparameter tuning

In terms of hyperparameters, the output layer activation function is sigmoid [49] and softmax is not chosen. Since the results of both are not very different in this model and sigmoid is more stable for scaled data in this project. Sigmoid is suitable for the binary classification prediction output of this project (0 for normal user label and 1 for electricity theft user label) because it exists between 0 and 1. While other activation functions are ReLU [49], which is currently used for almost all kinds of neural networks in deep learning. It is more conducive to back propagation and avoid problems such as gradient explosion or vanishing. As shown in figure 35:

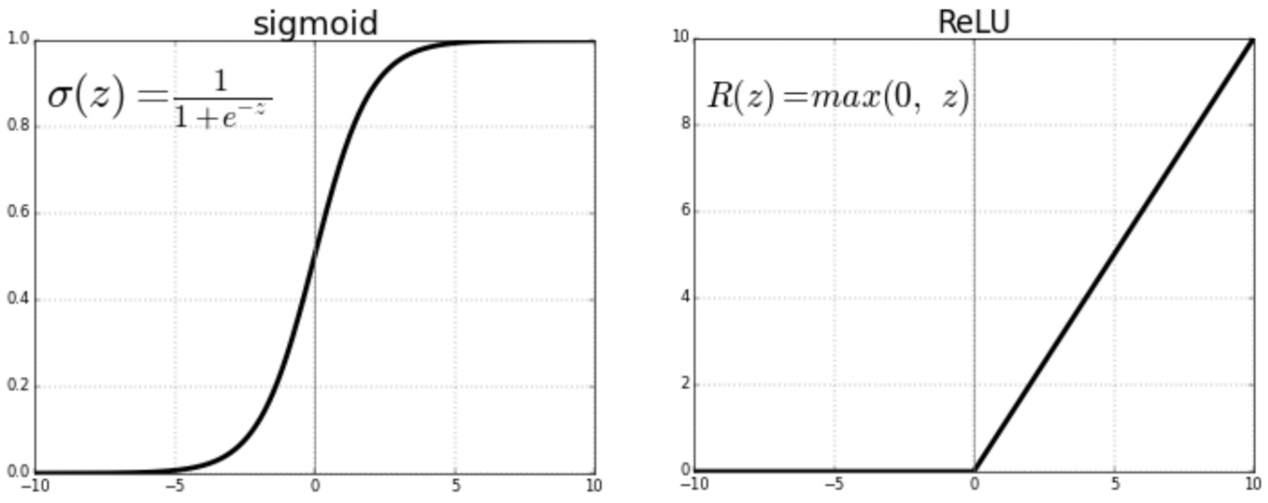


Figure 35: Sigmoid and ReLU Function [49]

In addition, logarithmic loss is also the first to deal with binary classification issue, namely the binary_crossentropy in Keras. Binary cross-entropy [50] compares each predicted probability with the actual category output, which can be either 0 or 1. It then calculates a score that penalizes the probability based on the distance from the expected value.

The optimizer is Adam [51], which is an optimization algorithm and can replace the classical stochastic gradient descent method to iteratively update the network weights in the training data. In short, Adam can adjust the learning rate of each network weight adaptively.

5.2 CNN-LSTM

The concept of the model is to use a CNN layer to extract features from the input time series electricity consumption data, i.e., the model extracts features within each sub-series (time period) in a windowed block pattern. Since the data set contains 1034 days of electricity consumption data, the time step can be divided into 11*94, i.e., each subsequence contains 3 months of electricity consumption data sequences. the CNN layer can be encapsulated in TimeDistributed [54] and the extracted features are flattened for use in the LSTM model. As shown in the figure below: As shown in following figure 36:

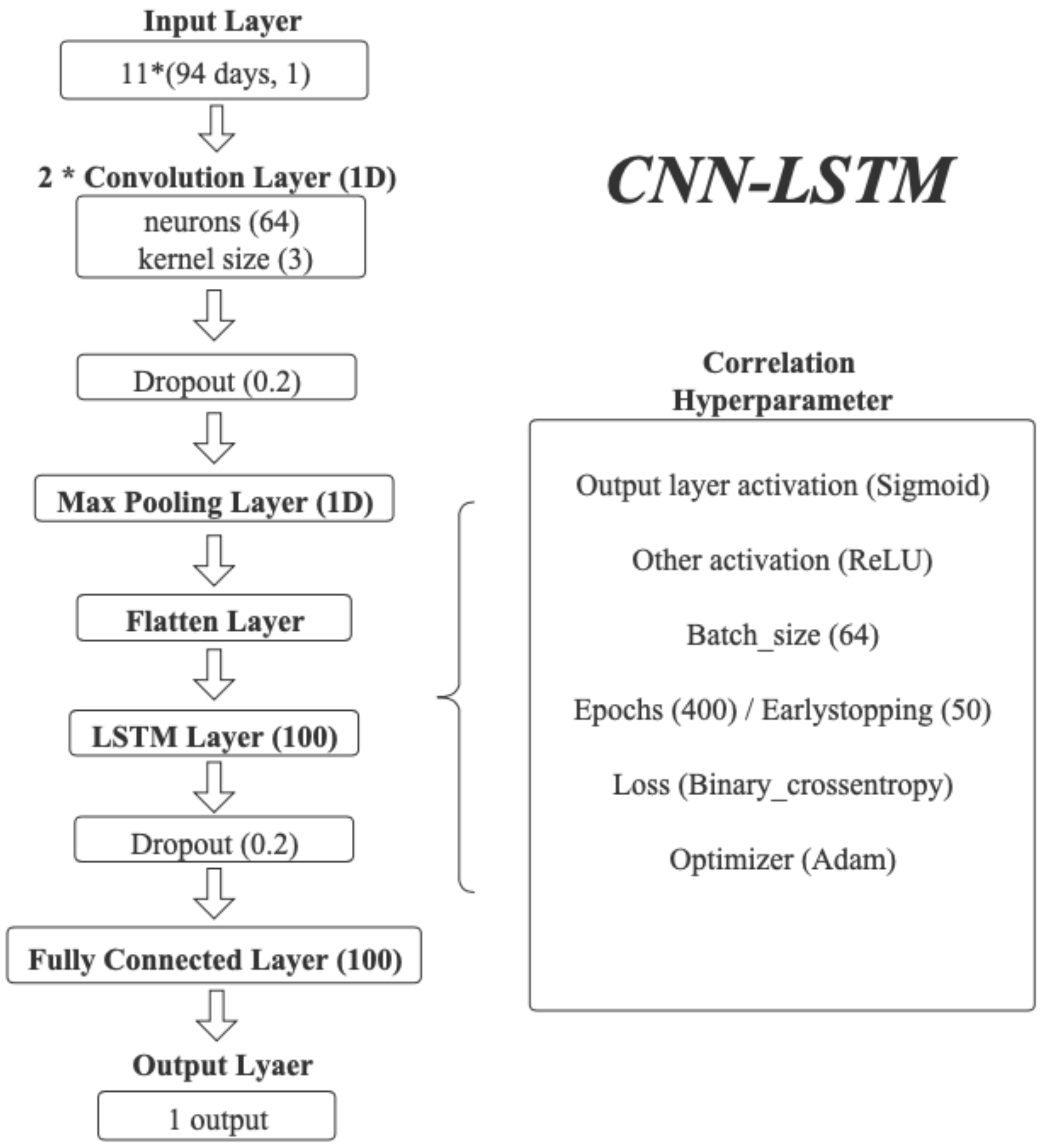


Figure 36: CNN-LSTM model architecture and hyperparameter tuning

The parameter settings of this model do not change much from the baseline model, and the maximum pooling layer following two consecutive CNN layers is also a more conventional model.

5.3 ConvLSTM

ConvLSTM can currently be applied in the basic computer vision domain [55] with good results. In addition, the classical 'human activity recognition' also uses this model architecture [56]. The central idea of this model in this project is to deal with the issue of binary classification of time series data. In essence, ConvLSTM is different from the first two models. CNN-LSTM passes the features extracted from CNN to LSTM, while ConvLSTM performs the convolutional operation in LSTM. This also involves another key difference, the input data of ConvLSTM is in three-dimensional (3-D) instead of the previous 1-D or 2-D data dimensions. The following figure 37 shows an example of the ConvLSTM structure:

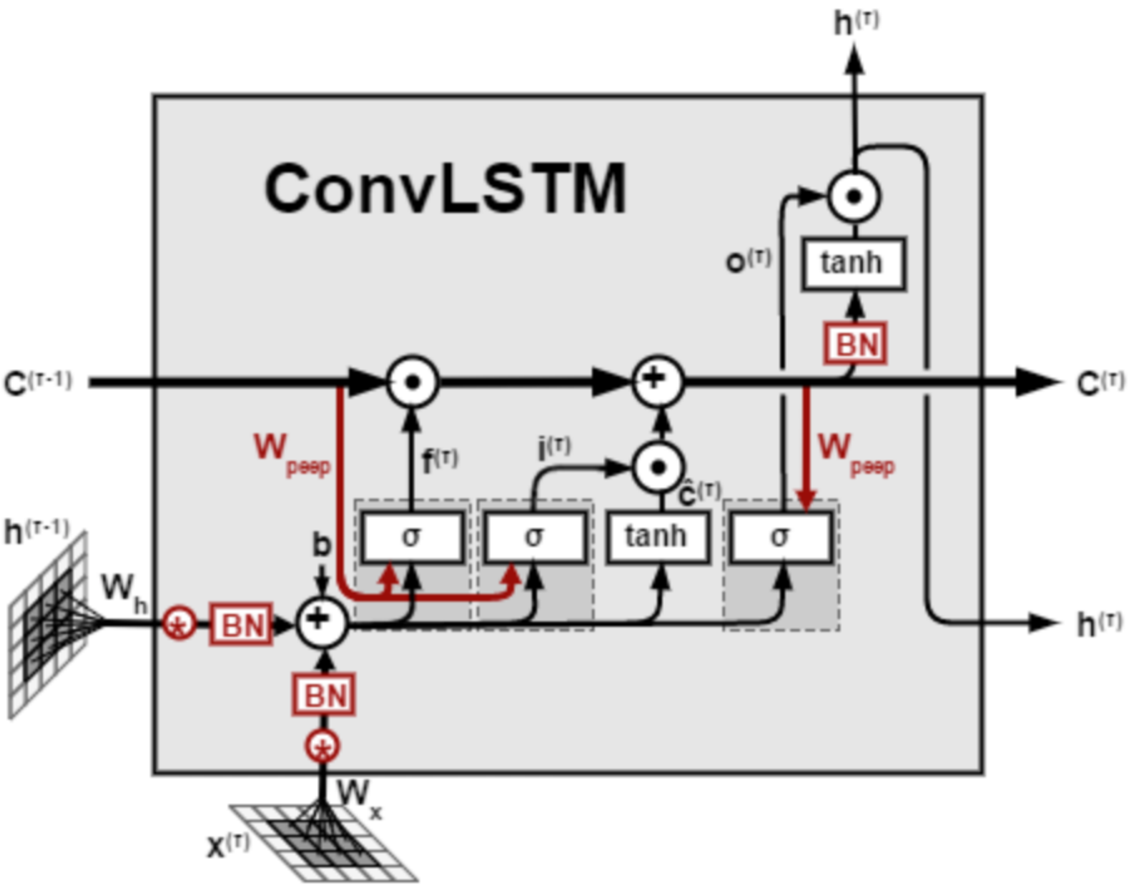


Figure 37: ConvLSTM cell structure [57]

Therefore, the first step of the model is the need to reshape the electricity consumption data into 3D dimensions, i.e., samples, time steps and features. In this model, the special ConvLSTM2D [58] is applied with the expected input dimensions: samples, time, rows, columns, channels, which can also be interpreted as the time step being decomposed into rows * columns of picture data points.

Here, 'time' is 11 and 'columns' is 94, in the same way as 1034 days were divided into 11*94 days in the last CNN-LSTM. ‘Row' is 1, because the original dimension of the electricity consumption data is 1D. In addition, 'channels' is also 1, because the data set does not contain other additional features.

Furthermore, in terms of data normalization, ConvLSTM in this project uses batch-normalization [45], which allows data set transformations to be compiled in the modeling, eliminating the time of separate transformations and providing model efficiency. The complete ConvLSTM structure becomes clearer and more concise, as shown in the following figure 38:

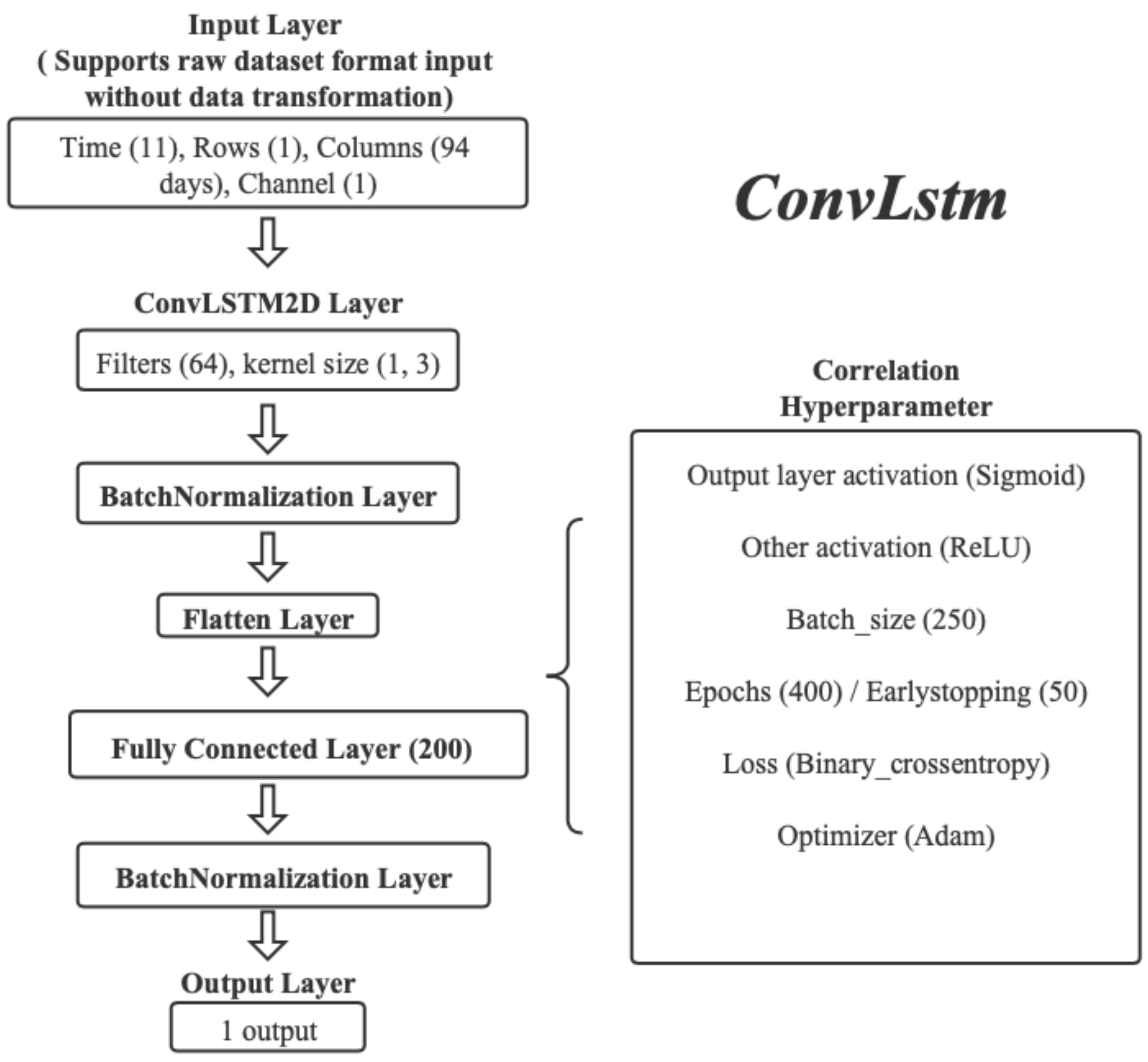

Input Layer
(Supports raw dataset format input without data transformation)
Time (11), Rows (1), Columns (94 days), Channel (1)
ConvLSTM2D Layer
Filters (64), kernel size (1, 3)
BatchNormalization Layer
Flatten Layer
Fully Connected Layer (200)
BatchNormalization Layer
Output Layer
1 output
ConvLstm
Correlation Hyperparameter
Output layer activation (Sigmoid)
Other activation (ReLU)
Batch_size (250)
Epochs (400) / Earlystopping (50)
Loss (Binary_crossentropy)
Optimizer (Adam)

Figure 38: ConvLSTM model architecture and hyperparameter tuning

A point worth noting is that the output still needs to be flattened into a long vector before the dense layer can be interpreted. After the model is complete, the next step is to evaluate the model with comprehensive performance metrics.

See Appendix B for relevant codes

6. Performance Metrics Analysis

6.1 Performance Metrics Description

For the model evaluation part, the project used a more comprehensive performance metric on the test data set (20%).

Classification accuracy

Accuracy is the prediction made by the model for each electricity user category in the test data set and compared to the user labels (0 and 1). Simply put accuracy is the percentage of correct examples predicted in the test set. Since the data set has been processed by balanced classification, the accuracy has a strong ability to prove. It is calculated as:

$$\text{Accuracy} = \frac{Correct\ Predictions}{Total\ Predictions}\ [59]$$

Loss (Binary cross-entropy/Log loss)

Both loss and accuracy should be a probability value between 0 and 1. In general, loss is the opposite of accuracy, with smaller values representing better model performance. It calculates the fraction of penalty probability based on the distance from the expected value, which means how close or far it is from the actual value [60]. This project is binary classification, so the loss is binary cross-entropy. As shown in the following figure 39:

$$\frac{1}{N}\sum_{i=1}^{N} -(y_i * \log(p_i) + (1-y_i) * \log(1-p_i))$$

Figure 39: Binary cross-entropy/Log loss formula [60]

Confusion Matrix

Confusion matrix is a technique applied to summarize the performance of classification algorithms, and it can show more intuitively the correctness and error types of the model [59]. In a way, it overcomes the limitation of relying solely on classification accuracy. Typical binary classification confusion matrix is shown in figure 40:

		Predicted	
		Positive	Negative
True	Positive	TP	FN
	Negative	FP	TN

Figure 40: Typical binary classification confusion matrix [59]

Assume that the value in each box is 10.

TP is True Positive, which means that the model correctly classifies 10 of the positive class data. In this project, it can be interpreted that the model correctly classifies 10 electricity theft user.

TN is True Negative, which means the model correctly classifies 10 negative classes of data. The same can be interpreted as the model correctly classifies 10 normal users.

FP is True Negative, which means that the model incorrectly classifies 10 negative classes as positive. This means that the 10 normal users are incorrectly predicted as electricity theft user.

FN is False Negative, which means that 10 positive data are incorrectly classified as negative by the model. This means that 10 electricity theft users are incorrectly predicted as normal users.

In addition, the accuracy can be calculated in a clearer way through the confusion matrix with the following formula:

$$\text{Accuracy} = \frac{TN + TP}{TN + FN + TP + FP} \text{ [59]}$$

Precision

Precision refers to the proportion of actual positive results in classified positive data, and represents the classification accuracy of classified positive data [13]. In this project, it is the classification accuracy of electricity theft users. Its value is between 0 and 1, the larger the better. The formula is as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \text{ [13]}$$

Recall

Recall is the proportion of classified positive results in the actual positive data, i.e., the classification accuracy in the actual positive data [13]. In this project, it represents the classification accuracy of actual electricity theft users, and its value is also between 0 and 1. The formula is as follows:

$$\text{Recall} = \frac{TP}{TP + FN} \text{ [13]}$$

F1-Score

F1-score captures the trend of precision and recall, thus making the model evaluation more comprehensive. The formula is as follows:

$$\text{F1} - \text{score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \text{ [6]}$$

Cohen’s kappa

Cohen's kappa can be used as a means of judging the strength of the model's classification predictions. The kappa value is more of a measure comparing the observed accuracy with the expected accuracy [61]. It is also a value between 0 and 1, the same as the previous metric, and the larger the value the better the performance of the model. The following is Cohen's kappa formula for the binary classification confusion matrix:

$$Cohen's\ kappa = \frac{2 \times \ (TP \times TN - FN \times FP)}{(TP + FP) \times (FP + TN) \times (TP + FN) \times (FN + TN)} \quad [62]$$

ROC-AUC

AUC is area under the curve and ROC-AUC can be derived by calculating area under the ROC curve [25]. On the AUC curve, the higher the ROC-AUC score the larger area under the curve, which has a value between 0 and 1. In fact, the ROC curve visualizes TP and FP in a trade-off manner.

PR-AUC

Similar to ROC-AUC, PR-AUC also defines the PR curve, which is a visualization of precision and recall. The PR-AUC also has a score between 0 and 1, with the higher the value the larger the curve area. The above two approaches are more often applied to the perfect evaluation of the model in the form of threshold intrinsic trade-offs [25].

6.2 Metric Comparison

The application of Python in machine learning makes the model evaluation precise and clear, and the performance metrics of all three target models have good performance, with ConvLSTM performing the best. This is also due to the proper pre-processing of the dataset.

	Accuracy	Loss	Precision	Recall	F1-Score	Cohen’s kappa	ROC-AUC	PR-AUC
MLP	0.964	0.162	0.969	0.960	0.964	0.927	0.985	0.983
CNN-LSTM	0.977	0.131	0.979	0.976	0.978	0.955	0.991	0.992
ConvLSTM	0.984	0.089	0.984	0.985	0.984	0.969	0.993	0.991

Table 3: Metrics comparison

In the AUC curves and PR curves, all three models performed very well. The curves all cover almost the entire AUC region, and the values are all very close to 1, which indicates the presence of at least one threshold that allows for excellent prediction. This is shown in figures 41 and 42:

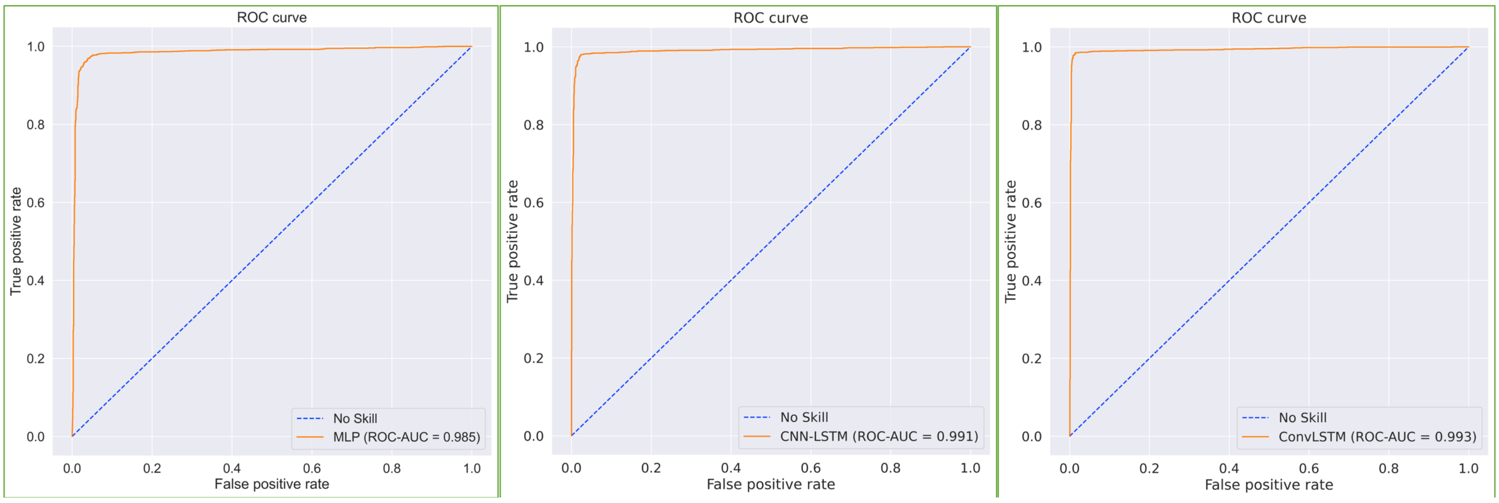


Figure 41: Comparison of ROC curves of MLP, CNN-LSTM and CovnLSTM

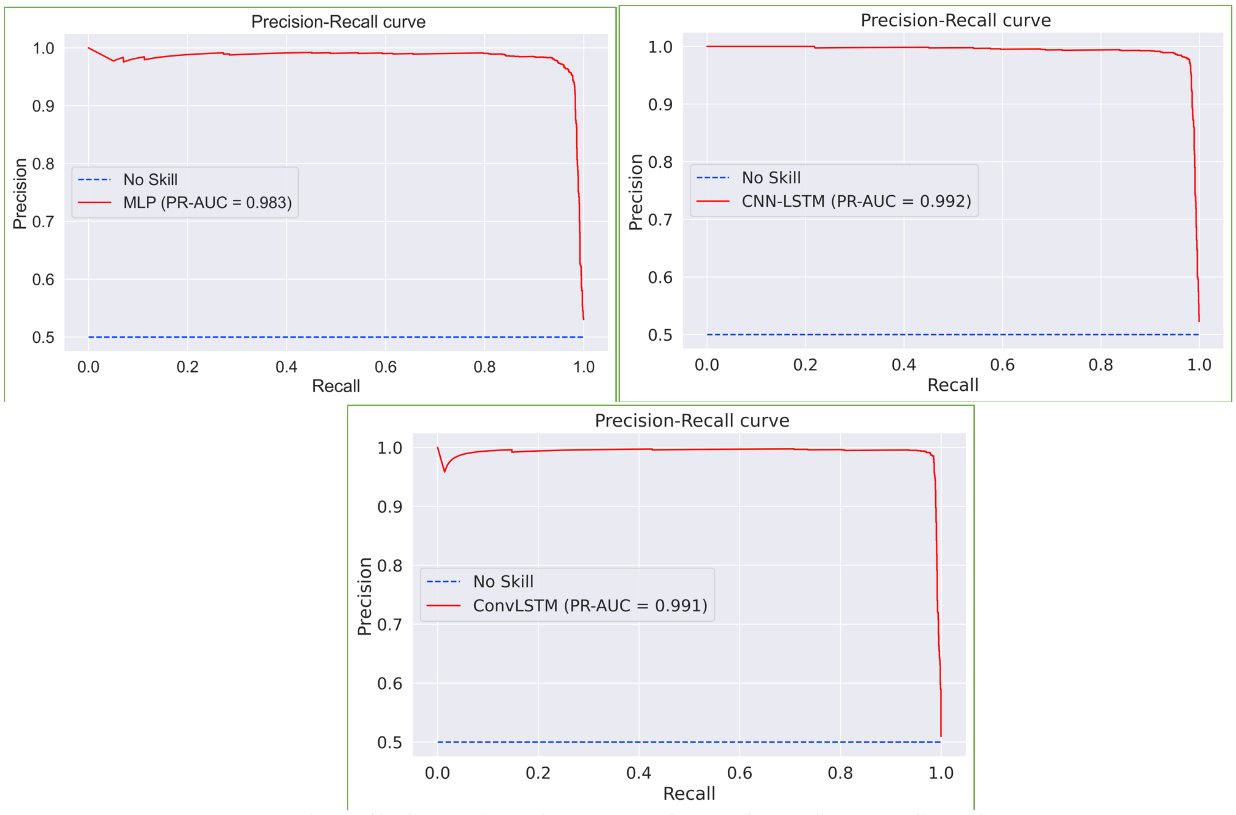


Figure 42: Comparison of PR curves of MLP, CNN-LSTM and CovnLSTM

For a more detailed comparison, the confusion matrix and the associated scores can also be reconfirmed with each other by means of pictures, as shown in following figure 43:

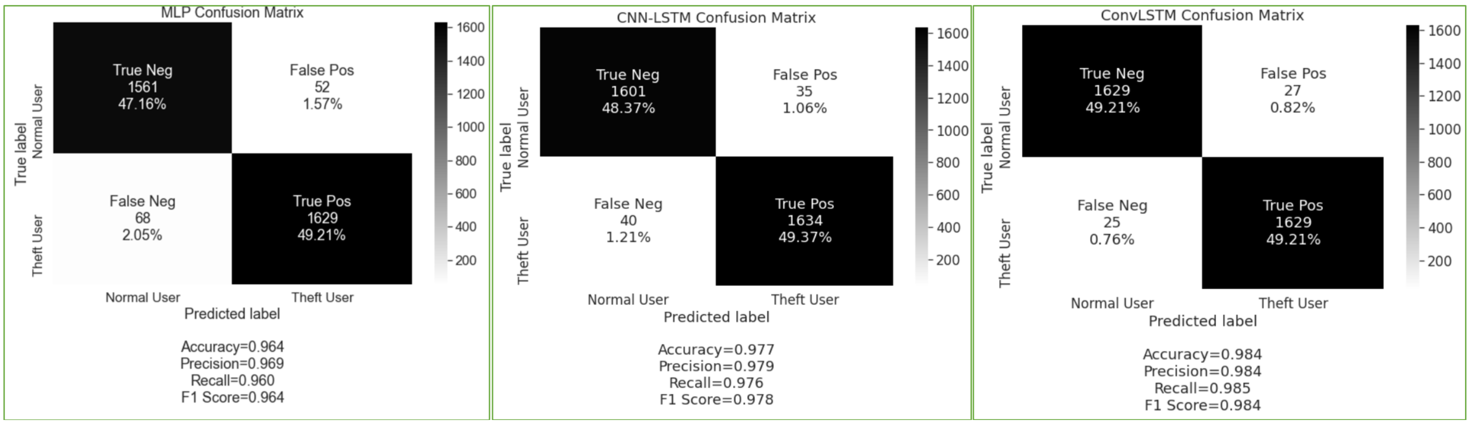


Figure 43: Comparison of Confusion Matrix of MLP, CNN-LSTM and CovnLSTM

In the previous PR curves, the PR-AUC of CNN-LSTM is slightly higher than that of ConvLSTM by 0.001. This can be observed in the confusion matrix image because CNN-LSTM is slightly more accurate than ConvLSTM in predicting the ture positive class, i.e., it is slightly more accurate in identifying the actual normal users. However, looking at all the predictions, the false positive and negative rate of ConvLSTM, and the accuracy of prediction of actual electricity theft users are stronger than CNN-LSTM. Combined with the previous comprehensive metrics comparison, the model robustness and prediction accuracy of ConvLSTM is stronger than CNN-LSMT and MLP.

6.3 Convergence Analysis

To better demonstrate the above conclusions, it is necessary to further analyze the convergence process [59] of the model, which is a visual analysis of the model's performance in the training and validation datasets. As shown in figure 44, 45 and 46:

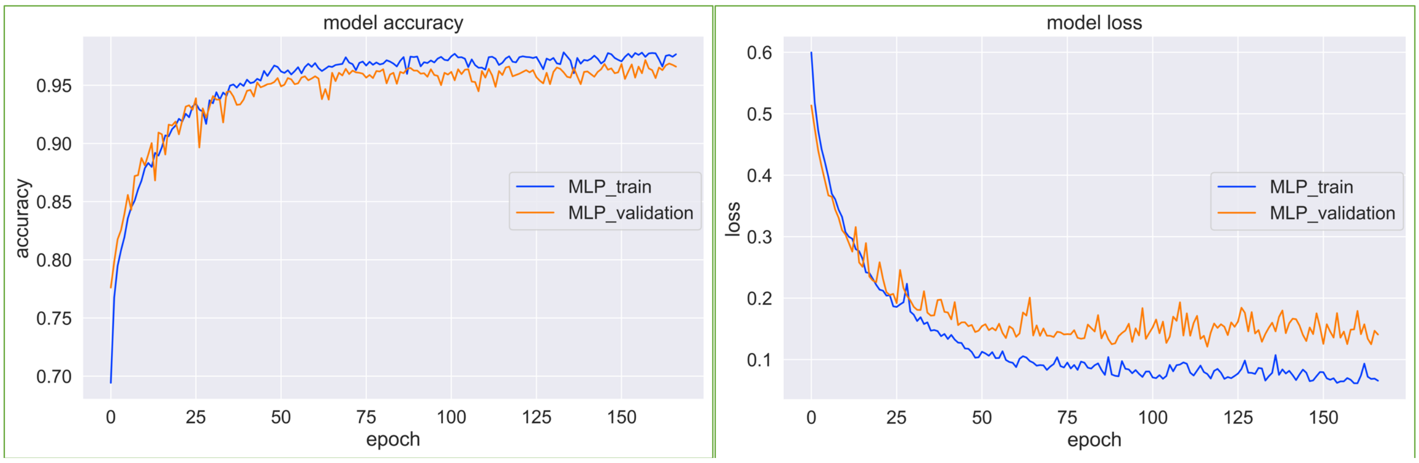


Figure 44: History of MLP Model

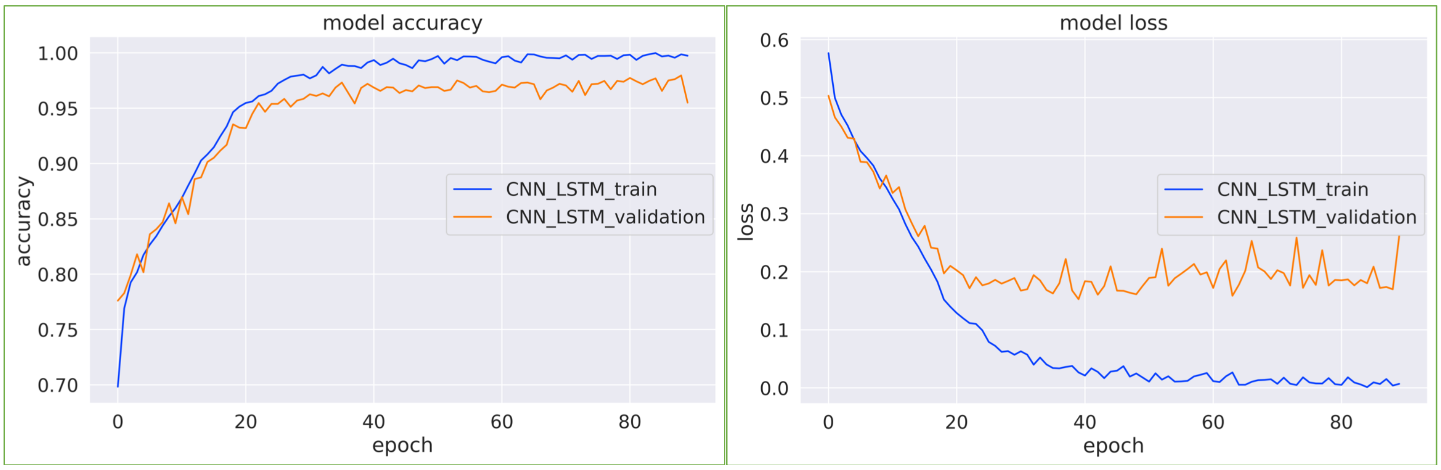
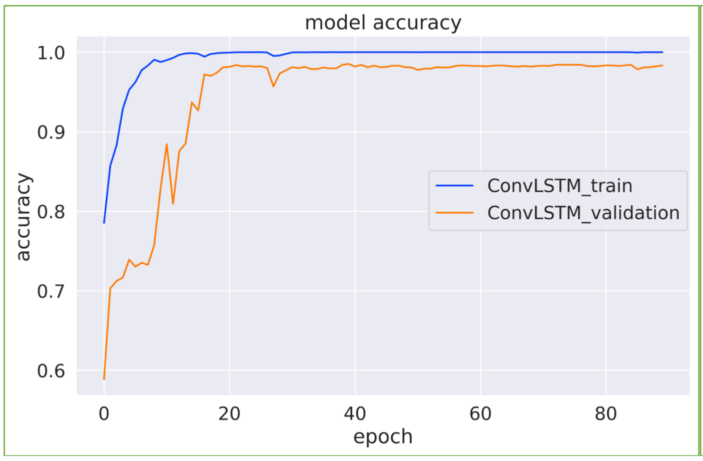


Figure 45: History of CNN-LSTM Model



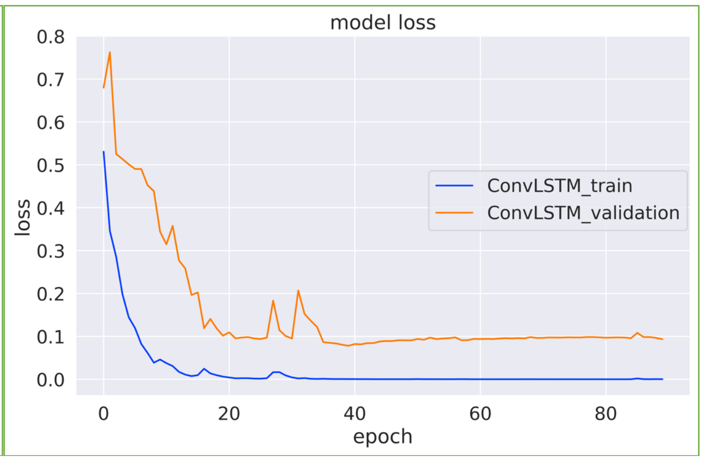


Figure 46: History of ConvLSTM Model

The above comparison reveals that the generalization gap [63] of all three models is relatively stable around 0.1. Both CNN-LSTM and ConvLSTM can reach the optimal model state before 90 epochs, while MLP needs around 150 epochs to reach it. In addition, MLP and CNN-LSTM reach smooth convergence around 60 epochs and 40 epochs, respectively, while ConvLSTM reaches the model convergence state well around 20 epochs. It is worth noting that ConLSTM performs better than the other two models in terms of noise control throughout the curve fluctuation state, which is also due to the application of batch-normalization in the model. Combined with the above analysis, ConvLSTM outperforms MLP and CNN-LSTM in terms of model convergence efficiency and model generalization ability, showing robustness of the model and prediction.

7. Conclusion & Future Work

7.1 Conclusion

In this project, three electricity theft detection models were successfully built and showed good results in electricity theft customer identification, where the novel ConvLSTM model outperformed MLP (baseline) and CNN-LSTM in aggregate. Data pre-processing techniques such as IQR, KNN imputation and Borderline-SMOTE are well utilized in the three models built in this project. ConvLSTM adopts more efficient batch-normalization in data transformation and outperforms MLP and CNN-LSTM on the test data set in terms of accuracy, loss, precision, recall, F1-score, Cohen's kappa, ROC-AUC, PR-AUC: 0.984, 0.089, 0.984, 0.985, 0.984, 0.969, 0.993, 0.991. In addition, it performs best in confusion matrix, ROC curve and PR curve. ConvLSTM also demonstrates robustness and outperforms the first two models in terms of model architecture, convergence efficiency and generalization ability. This can also show that ConvLSTM can improve the efficiency and flexibility of deploying machine learning ETDs for power utilities. It can adapt to the current complex and changing electricity theft environment by optimizing the complex core model structure and shortening the processing time of electricity consumption data on the basis of guaranteeing the accuracy of identifying electricity theft users. It enables power companies to make optimized deployments for different electricity theft behaviors in a shorter period of time.

7.2 Future Work

The hybrid LSTM model achieved good results in this project, however, the free cloud GPU still consumed a lot of training time during the implementation of the project due to the huge dataset and computation volume. Facing the huge electricity market, how to effectively solve the massive data and terminal matching problem is the key point that needs further expansion. For example, a regional power company can adopt more advanced computing equipment or deploy larger-scale cloud servers to the machine learning platform. Then it would be possible to collect, calculate and analyze the electricity consumption data and patterns of all customers in the region in real-time and simultaneously. Moreover, if the ETD model can be encapsulated to make it compatible with more convenient terminal platforms and transmission methods for operation, for example, the model can be ported to 5G communication mobile platforms. This will reduce the lag in detection of nascent electricity theft and also improves the foresight of changes in behavior of electricity theft.

Furthermore, GAN generation of electricity theft data, as well as merging STL decomposition data features, and even multi-feature modeling, such as adding weather and geographic location, are also issues that need further consideration. This can enable machine learning techniques to analyze and detect electricity theft in more dimensions, which is a way to combine objective factors to uncover potential electricity thieves.

Bibliography

[1] Fang, X., Misra, S., Xue, G., & Yang, D. (2011). *Smart grid—The new and improved power grid: A survey. IEEE communications surveys & tutorials, 14*(4), 944-980.

[2] Sreedevi, S. V., Prasannan, P., Jiju, K., & Lekshmi, I. I. (2020, January). Development of indigenous smart energy meter adhering indian standards for smart grid. In *2020 IEEE International Conference on Power Electronics, Smart Grid and Renewable Energy (PESGRE2020)* (pp. 1-5). IEEE.

[3] Zheng, J., Gao, D. W., & Lin, L. (2013, April). Smart meters in smart grid: An overview. In *2013 IEEE Green Technologies Conference (GreenTech)* (pp. 57-64). IEEE.

[4] Leite, J. B., & Mantovani, J. R. S. (2016). Detecting and locating non-technical losses in modern distribution networks. *IEEE Transactions on Smart Grid*, 9(2), 1023-1032.

[5] Taylor, Andy, Alexander, David, Finch, Amanda, & Sutton, David. (2013). *Information security management principles, second edition* (2nd ed.). Swindon, U.K: BCS Learning and Development.

[6] Hasan, M., Toma, R. N., Nahid, A. A., Islam, M. M., & Kim, J. M. (2019). Electricity theft detection in smart grid systems: A CNN-LSTM based approach. *Energies, 12*(17), 3310.

[7] Deng, L., & Yu, D. (2014). Deep learning: methods and applications. *Foundations and trends in signal processing*, *7*(3–4), 197-387.

[8] Dorffner, G. (1996). Neural networks for time series processing. *In Neural network world.*

[9] Yang, J., Nguyen, M. N., San, P. P., Li, X. L., & Krishnaswamy, S. (2015, June). Deep convolutional neural networks on multichannel time series for human activity recognition. In *Twenty-fourth international joint conference on artificial intelligence.*

[10] Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015, April). Long short term memory networks for anomaly detection in time series. In *Proceedings* (Vol. 89, pp. 89-94)..

[11] Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling* (Vol. 26, p. 13). New York: Springer.

[12] Zheng, Z., Yang, Y., Niu, X., Dai, H. N., & Zhou, Y. (2017). Wide and deep convolutional neural networks for electricity-theft detection to secure smart grids. *IEEE Transactions on Industrial Informatics, 14*(4), 1606-1615.

[13] Feng, X., Hui, H., Liang, Z., Guo, W., Que, H., Feng, H., ... & Ding, Y. (2020). A Novel Electricity Theft Detection Scheme Based on Text Convolutional Neural Networks. *Energies, 13(*21), 5758.

[14] Li, S., Han, Y., Yao, X., Yingchen, S., Wang, J., & Zhao, Q. (2019). Electricity theft detection in power grids with deep learning and random forests. *Journal of Electrical and Computer Engineering, 2019.*

[15] Chen, Z., Meng, D., Zhang, Y., Xin, T., & Xiao, D. (2020, February). Electricity theft detection using deep bidirectional recurrent neural network. In *2020 22nd International Conference on Advanced Communication Technology (ICACT)* (pp. 401-406). IEEE.

[16] Aslam, Z., Javaid, N., Ahmad, A., Ahmed, A., & Gulfam, S. M. (2020). A combined deep learning and ensemble learning methodology to avoid electricity theft in smart grids. *Energies, 13*(21), 5599.

[17] Bisong, E. (2019). Introduction to Scikit-learn. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform* (pp. 215-229). Apress, Berkeley, CA.

[18] Buzau, M. M., Tejedor-Aguilera, J., Cruz-Romero, P., & Gómez-Expósito, A. (2019). Hybrid deep neural networks for detection of non-technical losses in electricity smart meters. *IEEE Transactions on Power Systems, 35*(2), 1254-1263.

[19] Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., & Herrera, F. (2018). *Learning from imbalanced data sets* (Vol. 10, pp. 978-3). Berlin: Springer.

[20] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436-444.

[21] Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks*, 18(5-6), 602-610.

[22] Madhure, R. U., Raman, R., & Singh, S. K. (2020, July). Cnn-lstm based electricity theft detector in advanced metering infrastructure. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)* (pp. 1-6). IEEE.

[23] Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3156-3164).

[24] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research, 15*(1), 1929-1958..

[25] He, H., & Ma, Y. (Eds.). (2013). Imbalanced learning: foundations, algorithms, and applications. Somerset: Wiley.

[26] Networked Energy Services. (2020, June 17). *Energy Theft and Fraud Reduction*. N&S. Retrieved from https://www.networkedenergy.com/en/news-events/energy-theft-and-fraud-reduction?utm_medium=sei&utm_source=external&utm_campaign=article%20f&t

[27] Northeast Group, LLC. (2017, May). *Electricity Theft and Non-Technical Losses: Global Markets, Solutions, and Vendors.* Retrieved from http://www.northeast-group.com/reports/Brochure-Electricity%20Theft%20&%20Non-Technical%20Losses%20-%20Northeast%20Group.pdf

[28] Northeast Group, LLC. (2020, Jan). *Emerging Markets Smart Grid: Outlook 2020.* Retrieved from http://www.northeast-group.com/reports/Brochure-Emerging%20Markets%20Smart%20Grid%20Outlook%202020%20-%20Northeast%20Group.pdf

[29] Bhatia, G., & Gulati, M. (2004). Reforming the Power Sector: Controlling Electricity Theft and Improving Revenue. The World Bank, Washington, DC.

[30] Jamacia Public Service Company Ltd. (2019). *2014-2019 Tariff Application*. Retrieved from https://www.our.org.jm/ourweb/sites/default/files/documents/sector_documents/jps_rate_case_application_for_2019-2024_public_version.pdf

[31] Witten, Ian H, Frank, Eibe, Hall, Mark A, & Pal, Christopher J. (2016). *Data Mining (The Morgan Kaufmann Series in Data Management Systems).* San Francisco: Elsevier Science & Technology.

[32] Li, Y., Pizer, W. A., & Wu, L. (2019). Climate change and residential electricity consumption in the Yangtze River Delta, China. *Proceedings of the National Academy of Sciences, 116*(2), 472-477.

[33] Richardson, A. (2010). Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach. *International Statistical Review / Revue Internationale De Statistique, 78*(3), 451-452.

[34] Kazil, J., & Jarmul, K. (2016). *Data wrangling with Python: tips and tools to make your life easier*. " O'Reilly Media, Inc.".

[35] Efron, B. (1994). Missing data, imputation, and the bootstrap. *Journal of the American Statistical Association, 89*(426), 463-475.

[36] Beretta, L., & Santaniello, A. (2016). Nearest neighbor imputation algorithms: a critical evaluation. *BMC medical informatics and decision making, 16*(3), 197-208..

[37] Aggarwal, C., SpringerLink, & Elsevier. (2013). *Outlier Analysis*.

[38] Galarnyk, M. (2020, July 6). *Understanding Boxplots - Towards Data Science*. Medium. Retrieved from https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51

[39] Dawson, R. (2011). How significant is a boxplot outlier?. *Journal of Statistics Education, 19*(2).

[40] Bergmeir, C., Hyndman, R. J., & Benítez, J. M. (2016). Bagging exponential smoothing methods using STL decomposition and Box–Cox transformation. *International journal of forecasting, 32*(2), 303-312.

[41] Patel, A. (2018, July 20). *Chapter-6 How to learn feature engineering*? - ML Research Lab. Medium. Retrieved from https://medium.com/ml-research-lab/chapter-6-how-to-learn-feature-engineering-49f4246f0d41

[42] Stout, B. (n.d.). Undersampling and Oversampling Statistics Visual Example. Pinterest. Retrieved from https://www.pinterest.co.uk/pin/514958538641697615/

[43] Han, H., Wang, W. Y., & Mao, B. H. (2005, August). Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. In *International conference on intelligent computing* (pp. 878-887). Springer, Berlin, Heidelberg.

[44] Criddle, B. C. (2021, February 10). *Bitcoin consumes "more electricity than Argentina."* BBC News. Retrieved from https://www.bbc.co.uk/news/technology-56012952

[45] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.

[46] Brownlee, J. (2016). *Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras.* Machine Learning Mastery.

[47] Prechelt, L. (1998). Early stopping-but when?. In *Neural Networks: Tricks of the trade* (pp. 55-69). Springer, Berlin, Heidelberg.

[48] Wager, S., Wang, S., & Liang, P. S. (2013). Dropout training as adaptive regularization. *Advances in neural information processing systems*, 26, 351-359.

[49] Sharma, S. (2021, July 4). *Activation Functions in Neural Networks - Towards Data Science*. Medium. Retrieved from https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[50] Ramos, D., Franco-Pedroso, J., Lozano-Diez, A., & Gonzalez-Rodriguez, J. (2018). Deconstructing cross-entropy for probabilistic binary classifiers. *Entropy, 20*(3), 208.

[51] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[52] Phi, M. (2020, June 28). *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. Medium. Retrieved from https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[53] Gurucharan, M. K. (2021, April 3). *Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network*. UpGrad Blog. Retrieved from https://www.upgrad.com/blog/basic-cnn-architecture/

[54] Mutegeki, R., & Han, D. S. (2020, February). A CNN-LSTM approach to human activity recognition. In *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)* (pp. 362-366). IEEE.

[55] Song, H., Wang, W., Zhao, S., Shen, J., & Lam, K. M. (2018). Pyramid dilated deeper convlstm for video salient object detection. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 715-731).

[56] Singh, S. P., Sharma, M. K., Lay-Ekuakille, A., Gangwar, D., & Gupta, S. (2020). Deep ConvLSTM with self-attention for human activity decoding using wearable sensors. *IEEE Sensors Journal, 21*(6), 8575-8582.

[57] Xavier, A. (2019, April 22). *An introduction to ConvLSTM - Neuronio.* Medium. Retrieved from https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7

[58] Xingjian, S. H. I., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems* (pp. 802-810).

[59] Branco, P., Torgo, L., & Ribeiro, R. (2015). A survey of predictive modelling under imbalanced distributions. *arXiv preprint arXiv:1505.01658.*

[60] Saxena, S. (2021, March 3). *Binary Cross Entropy/Log Loss for Binary Classification*. Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/

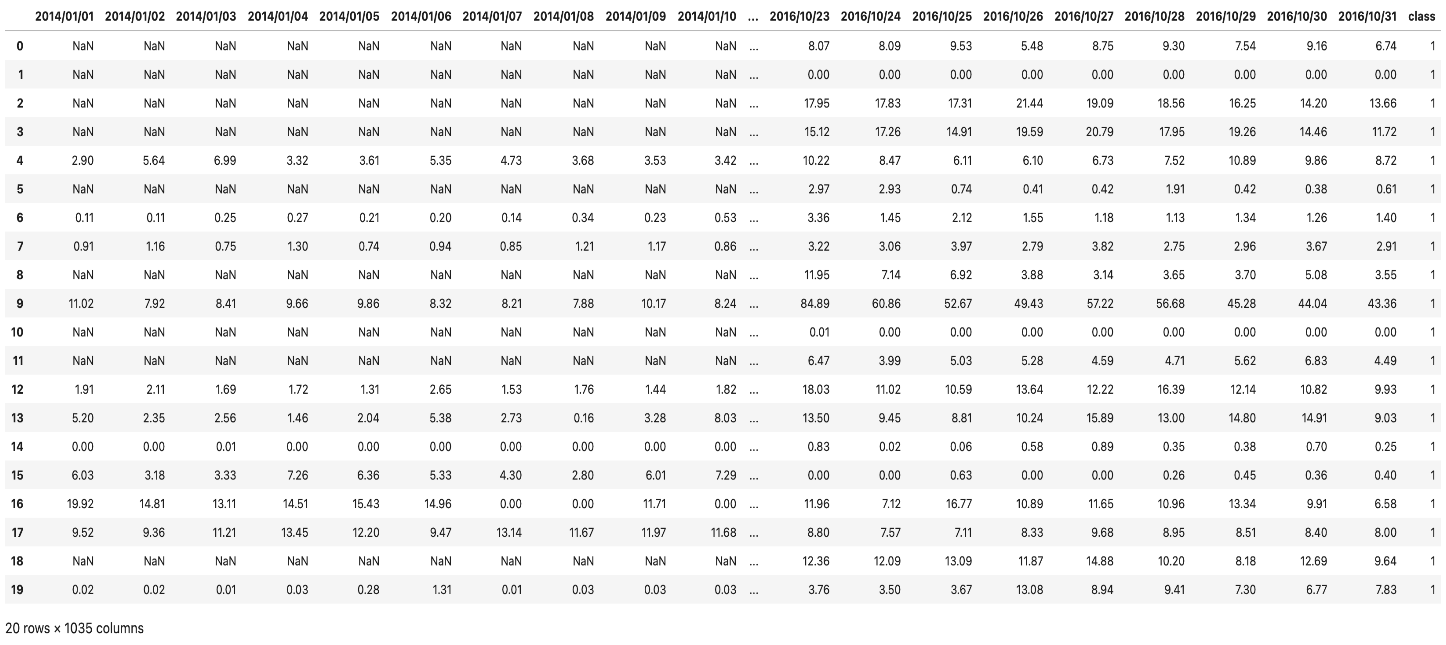
[61] McHugh, M. L. (2012). Interrater reliability: the kappa statistic. *Biochemia medica, 22*(3), 276-282.

[62] Chicco, D., Warrens, M. J., & Jurman, G. (2021). The Matthews correlation coefficient (MCC) is more informative than Cohen's Kappa and Brier score in binary classification assessment. *IEEE Access*.

[63] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836.*

Appendix A - Data set Description

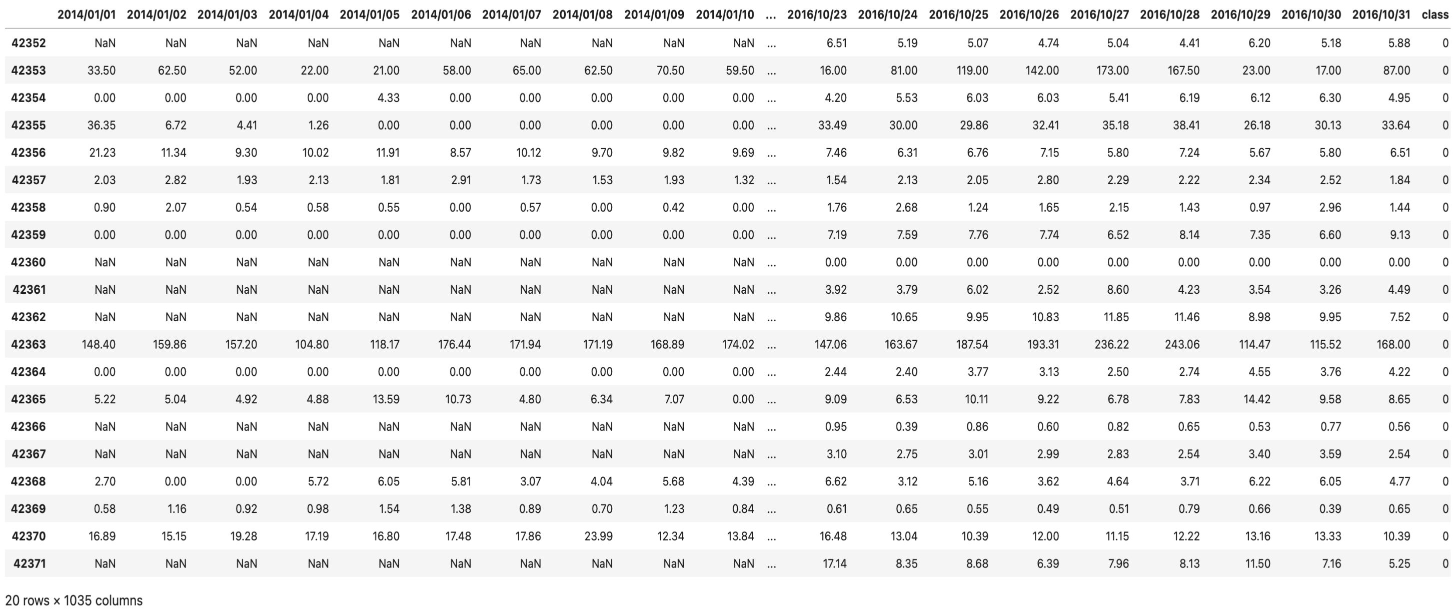
The data set comes from the link: https://github.com/henryRDlab/ElectricityTheftDetection [12]. The following is the content of this project:



	2014/01/01	2014/01/02	2014/01/03	2014/01/04	2014/01/05	2014/01/06	2014/01/07	2014/01/08	2014/01/09	2014/01/10	...	2016/10/23	2016/10/24	2016/10/25	2016/10/26	2016/10/27	2016/10/28	2016/10/29	2016/10/30	2016/10/31	class
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	8.07	8.09	9.53	5.48	8.75	9.30	7.54	9.16	6.74	1
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	17.95	17.83	17.31	21.44	19.09	18.56	16.25	14.20	13.66	1
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	15.12	17.26	14.91	19.59	20.79	17.95	19.26	14.46	11.72	1
4	2.90	5.64	6.99	3.32	3.61	5.35	4.73	3.68	3.53	3.42	...	10.22	8.47	6.11	6.10	6.73	7.52	10.89	9.86	8.72	1
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.97	2.93	0.74	0.41	0.42	1.91	0.42	0.38	0.61	1
6	0.11	0.11	0.25	0.27	0.21	0.20	0.14	0.34	0.23	0.53	...	3.36	1.45	2.12	1.55	1.18	1.13	1.34	1.26	1.40	1
7	0.91	1.16	0.75	1.30	0.74	0.94	0.85	1.21	1.17	0.86	...	3.22	3.06	3.97	2.79	3.82	2.75	2.96	3.67	2.91	1
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	11.95	7.14	6.92	3.88	3.14	3.65	3.70	5.08	3.55	1
9	11.02	7.92	8.41	9.66	9.86	8.32	8.21	7.88	10.17	8.24	...	84.89	60.86	52.67	49.43	57.22	56.68	45.28	44.04	43.36	1
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	6.47	3.99	5.03	5.28	4.59	4.71	5.62	6.83	4.49	1
12	1.91	2.11	1.69	1.72	1.31	2.65	1.53	1.76	1.44	1.82	...	18.03	11.02	10.59	13.64	12.22	16.39	12.14	10.82	9.93	1
13	5.20	2.35	2.56	1.46	2.04	5.38	2.73	0.16	3.28	8.03	...	13.50	9.45	8.81	10.24	15.89	13.00	14.80	14.91	9.03	1
14	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.83	0.02	0.06	0.58	0.89	0.35	0.38	0.70	0.25	1
15	6.03	3.18	3.33	7.26	6.36	5.33	4.30	2.80	6.01	7.29	...	0.00	0.00	0.63	0.00	0.00	0.26	0.45	0.36	0.40	1
16	19.92	14.81	13.11	14.51	15.43	14.96	0.00	0.00	11.71	0.00	...	11.96	7.12	16.77	10.89	11.65	10.96	13.34	9.91	6.58	1
17	9.52	9.36	11.21	13.45	12.20	9.47	13.14	11.67	11.97	11.68	...	8.80	7.57	7.11	8.33	9.68	8.95	8.51	8.40	8.00	1
18	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	12.36	12.09	13.09	11.87	14.88	10.20	8.18	12.69	9.64	1
19	0.02	0.02	0.01	0.03	0.28	1.31	0.01	0.03	0.03	0.03	...	3.76	3.50	3.67	13.08	8.94	9.41	7.30	6.77	7.83	1

20 rows × 1035 columns

Figure A1: Overview of raw data (first 20 rows)

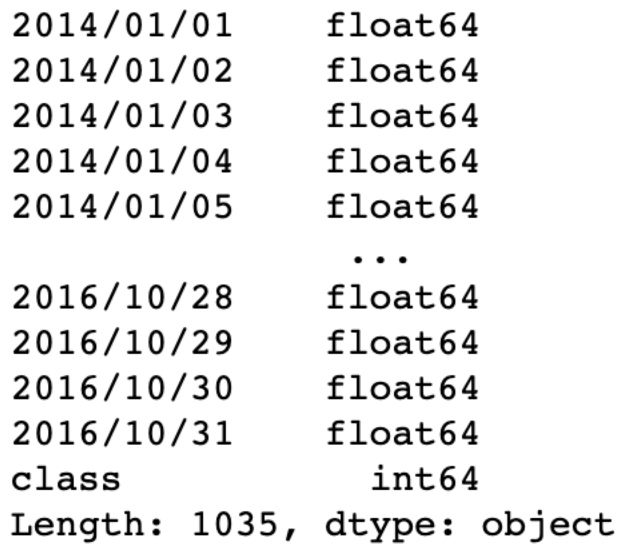


	2014/01/01	2014/01/02	2014/01/03	2014/01/04	2014/01/05	2014/01/06	2014/01/07	2014/01/08	2014/01/09	2014/01/10	...	2016/10/23	2016/10/24	2016/10/25	2016/10/26	2016/10/27	2016/10/28	2016/10/29	2016/10/30	2016/10/31	class
42352	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	6.51	5.19	5.07	4.74	5.04	4.41	6.20	5.18	5.88	0
42353	33.50	62.50	52.00	22.00	21.00	58.00	65.00	62.50	70.50	59.50	...	16.00	81.00	119.00	142.00	173.00	167.50	23.00	17.00	87.00	0
42354	0.00	0.00	0.00	0.00	4.33	0.00	0.00	0.00	0.00	0.00	...	4.20	5.53	6.03	6.03	5.41	6.19	6.12	6.30	4.95	0
42355	36.35	6.72	4.41	1.26	0.00	0.00	0.00	0.00	0.00	0.00	...	33.49	30.00	29.86	32.41	35.18	38.41	26.18	30.13	33.64	0
42356	21.23	11.34	9.30	10.02	11.91	8.57	10.12	9.70	9.82	9.69	...	7.46	6.31	6.76	7.15	5.80	7.24	5.67	5.80	6.51	0
42357	2.03	2.82	1.93	2.13	1.81	2.91	1.73	1.53	1.93	1.32	...	1.54	2.13	2.05	2.80	2.29	2.22	2.34	2.52	1.84	0
42358	0.90	2.07	0.54	0.58	0.55	0.00	0.57	0.00	0.42	0.00	...	1.76	2.68	1.24	1.65	2.15	1.43	0.97	2.96	1.44	0
42359	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	7.19	7.59	7.76	7.74	6.52	8.14	7.35	6.60	9.13	0
42360	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0
42361	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	3.92	3.79	6.02	2.52	8.60	4.23	3.54	3.26	4.49	0
42362	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	9.86	10.65	9.95	10.83	11.85	11.46	8.98	9.95	7.52	0
42363	148.40	159.86	157.20	104.80	118.17	176.44	171.94	171.19	168.89	174.02	...	147.06	163.67	187.54	193.31	236.22	243.06	114.47	115.52	168.00	0
42364	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	2.44	2.40	3.77	3.13	2.50	2.74	4.55	3.76	4.22	0
42365	5.22	5.04	4.92	4.88	13.59	10.73	4.80	6.34	7.07	0.00	...	9.09	6.53	10.11	9.22	6.78	7.83	14.42	9.58	8.65	0
42366	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.95	0.39	0.86	0.60	0.82	0.65	0.53	0.77	0.56	0
42367	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	3.10	2.75	3.01	2.99	2.83	2.54	3.40	3.59	2.54	0
42368	2.70	0.00	0.00	5.72	6.05	5.81	3.07	4.04	5.68	4.39	...	6.62	3.12	5.16	3.62	4.64	3.71	6.22	6.05	4.77	0
42369	0.58	1.16	0.92	0.98	1.54	1.38	0.89	0.70	1.23	0.84	...	0.61	0.65	0.55	0.49	0.51	0.79	0.66	0.39	0.65	0
42370	16.89	15.15	19.28	17.19	16.80	17.48	17.86	23.99	12.34	13.84	...	16.48	13.04	10.39	12.00	11.15	12.22	13.16	13.33	10.39	0
42371	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	17.14	8.35	8.68	6.39	7.96	8.13	11.50	7.16	5.25	0

20 rows × 1035 columns

Figure A2: Overview of raw data (last 20 rows)

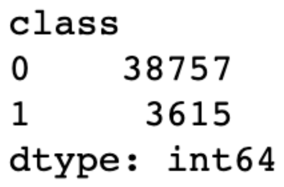
An observation of the above figure shows that the data set consists of 1034 columns of daily electricity consumption data and 1 column of categorically labelled data with 42,372 rows of users. In addition, the data set has a large number of missing values marked as NaN and many zero values. The large number of missing and zero values can directly lead to the failure of the subsequent visual data analysis and affect the accuracy of the model predictions.



```
2014/01/01    float64
2014/01/02    float64
2014/01/03    float64
2014/01/04    float64
2014/01/05    float64
                ...
2016/10/28    float64
2016/10/29    float64
2016/10/30    float64
2016/10/31    float64
class           int64
Length: 1035, dtype: object
```

Figure A3: The type of the data set variable

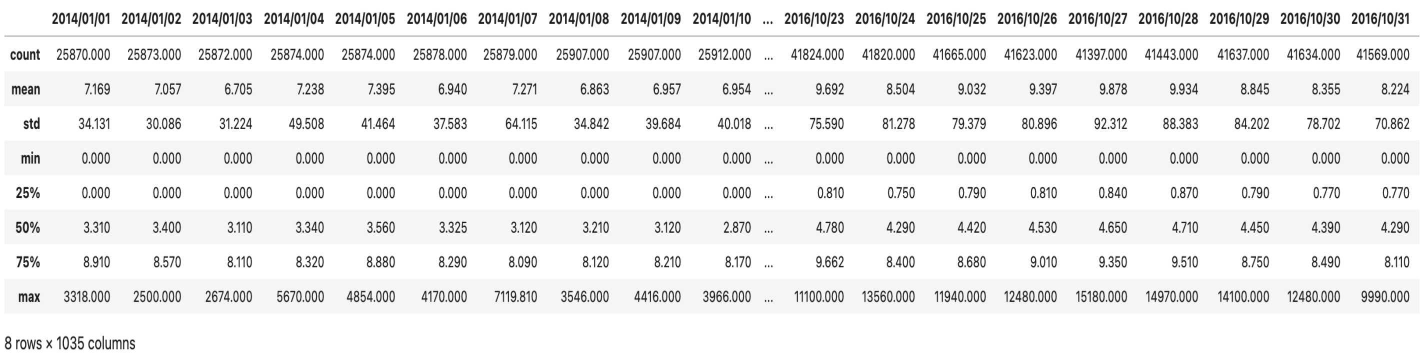
The data type matches the data set description, with the power type float64 and the category label int64.



```
class
0    38757
1     3615
dtype: int64
```

Figure A4: Class distribution

Tag 1 corresponds to 3615 electricity theft customers and tag 0 corresponds to 38757 normal customers. In total, there are 42,372 tags, which match the description of the dataset.



	2014/01/01	2014/01/02	2014/01/03	2014/01/04	2014/01/05	2014/01/06	2014/01/07	2014/01/08	2014/01/09	2014/01/10	...	2016/10/23	2016/10/24	2016/10/25	2016/10/26	2016/10/27	2016/10/28	2016/10/29	2016/10/30	2016/10/31
count	25870.000	25873.000	25872.000	25874.000	25874.000	25878.000	25879.000	25907.000	25907.000	25912.000	...	41824.000	41820.000	41665.000	41623.000	41397.000	41443.000	41637.000	41634.000	41569.000
mean	7.169	7.057	6.705	7.238	7.395	6.940	7.271	6.863	6.957	6.954	...	9.692	8.504	9.032	9.397	9.878	9.934	8.845	8.355	8.224
std	34.131	30.086	31.224	49.508	41.464	37.583	64.115	34.842	39.684	40.018	...	75.590	81.278	79.379	80.896	92.312	88.383	84.202	78.702	70.862
min	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
25%	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.810	0.750	0.790	0.810	0.840	0.870	0.790	0.770	0.770
50%	3.310	3.400	3.110	3.340	3.560	3.325	3.120	3.210	3.120	2.870	...	4.780	4.290	4.420	4.530	4.650	4.710	4.450	4.390	4.290
75%	8.910	8.570	8.110	8.320	8.880	8.290	8.090	8.120	8.210	8.170	...	9.662	8.400	8.680	9.010	9.350	9.510	8.750	8.490	8.110
max	3318.000	2500.000	2674.000	5670.000	4854.000	4170.000	7119.810	3546.000	4416.000	3966.000	...	11100.000	13560.000	11940.000	12480.000	15180.000	14970.000	14100.000	12480.000	9990.000

8 rows × 1035 columns

Figure A5: Raw data descriptive statistics

Anomalies in the data can be found by looking at the statistical information in the raw data:

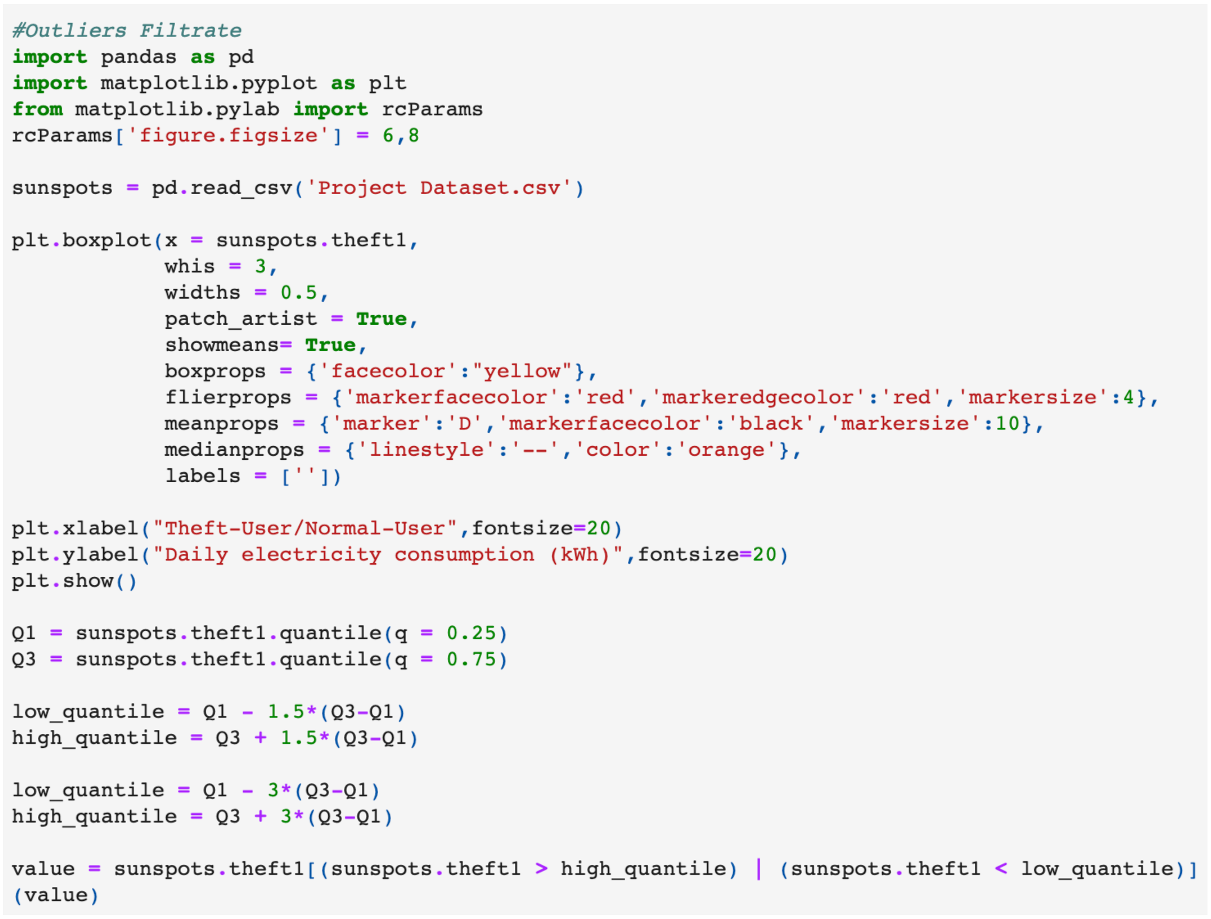
1. The Count of daily electricity consumption for all users varies greatly and some of the 25th Percentile is almost zero, indicating that the data set contains a large number of missing or zero values.

2. The maximum values of daily electricity consumption are unusual, with some of the maximum daily consumption exceeding 10,000 kWh, however, the data and averages of daily electricity consumption are very much in line with the habits of residential electricity consumption. It can be analysed that this data set should contain a small amount of information on industrial or commercial electricity consumption, or that the data collection is anomalous due to a malfunctioning energy metering device. The billing and data collection methods for industrial electricity consumption also differ from those for general electricity consumption, as do the patterns of behaviour and patterns of electricity consumption between the two [32]. However, these users will be retained in order to maintain the authenticity of the data set and to validate the compatibility of the model. This issue can be dealt with at a later date when the data is transformed.

3. The standard deviation fluctuates over time. This indicates that there are seasonal fluctuations in this electricity consumption, or that it is affected by outliers.

Appendix B - Related Project Code

The data set format of this project are CSV and XLSX, and the preliminary preparation is carried out through Microsoft Excel. Project data pre-processing, image generation, model establishment, training and result evaluation were all programmed in Python and completed on the Jupyter Notebook platform. On the hardware side, CNN-LSTM and ConvLSTM modeling is performed on a free cloud GPU due to the large data set and the enormous computing time involved. The rest is performed on a personal laptop. For details, please refer to 3.2 Related Configuration of Project Platform. The following is a code summary of the significant steps:



```python
#Outliers Filtrate
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 6,8

sunspots = pd.read_csv('Project Dataset.csv')

plt.boxplot(x = sunspots.theft1,
            whis = 3,
            widths = 0.5,
            patch_artist = True,
            showmeans= True,
            boxprops = {'facecolor':"yellow"},
            flierprops = {'markerfacecolor':'red','markeredgecolor':'red','markersize':4},
            meanprops = {'marker':'D','markerfacecolor':'black','markersize':10},
            medianprops = {'linestyle':'--','color':'orange'},
            labels = [''])

plt.xlabel("Theft-User/Normal-User",fontsize=20)
plt.ylabel("Daily electricity consumption (kWh)",fontsize=20)
plt.show()

Q1 = sunspots.theft1.quantile(q = 0.25)
Q3 = sunspots.theft1.quantile(q = 0.75)

low_quantile = Q1 - 1.5*(Q3-Q1)
high_quantile = Q3 + 1.5*(Q3-Q1)

low_quantile = Q1 - 3*(Q3-Q1)
high_quantile = Q3 + 3*(Q3-Q1)

value = sunspots.theft1[(sunspots.theft1 > high_quantile) | (sunspots.theft1 < low_quantile)]
(value)
```

Figure B1: Outlier filtering

```python
# KNN Imputation
from numpy import isnan
from pandas import read_csv
from sklearn.impute import KNNImputer
import pandas as pd

dataframe = read_csv('Project Dataset.csv', header=None, na_values='?')

data = dataframe.values
ix = [i for i in range(data.shape[1]) if i != 1034]
X, y = data[:, ix], data[:, 1034]

print('Missing: %d' % sum(isnan(X).flatten()))

imputer = KNNImputer()
imputer.fit(X)
Xtrans = imputer.transform(X)
print('Missing: %d' % sum(isnan(Xtrans).flatten()))
df = pd.DataFrame(Xtrans)
df.to_csv('KNN Project Dataset.csv')
```

Figure B2: KNN Imputation

```python
# Pearson’s correlation coefficient 4 weeks (Average normal users/electricity theft users)
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 15,6

data = pd.read_csv('Project Dataset 4 weeks.csv')

print("Is there any null value in the dataframe ?",data.isnull().values.any())
print(data.isnull().sum())
print("The number of entries in this dataframe :",len(data))

data.corr()

plt.subplots(figsize = (10,10))
sns.heatmap(data.corr(),annot=True,fmt="f").set_title("Pearson’s correlation coefficient (PCC)
                                                       (Normal/Electricity Theft User)")
plt.savefig('image.png',dpi=600, bbox_inches='tight')
plt.show()

# Pearson’s correlation coefficient 1034days (normal users/electricity theft users)
from matplotlib import pyplot
from pandas import read_csv
from matplotlib.pylab import rcParams
import matplotlib.pyplot as plt
rcParams['figure.figsize'] = 15,6

data = read_csv('Project Dataset 1034days.csv')
correlations = data.corr()


fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)

plt.savefig('image.png',dpi=600, bbox_inches='tight')
pyplot.show()
```

Figure B3: PCC normal users/electricity theft users (4 weeks average/1034 days)

```python
# ACF 4weeks average (nomral users/electricity theft users)
import numpy as np
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 15,6

data = pd.read_csv('Project Dataset 4 weeks average (normal/electricity theft.csv')
data.head()

data['datetime'] = pd.to_datetime(data.datetime)
data = data.set_index(data.datetime)
data.drop('datetime', axis = 1, inplace = True)
data.head()

ts = data['value1']
plot_acf(ts)
plt.savefig('image.png',dpi=600, bbox_inches='tight')
plt.show()
```

Figure B4: ACF normal users/electricity theft users (4 weeks average)

```python
# 1034days STL Decomposing (normal users/electricity theft users)
import numpy as np
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 30,18
sns.set_theme(style="darkgrid")
sns.set(font_scale=3,palette='dark')

data = pd.read_csv('Project Dataset (normal users/electricity theft users).csv')
data['datetime'] = pd.to_datetime(data.datetime)
data = data.set_index(data.datetime)
data.drop('datetime', axis = 1, inplace = True)
ts = data['value1']
ts_log = np.log(ts)

decomposition = seasonal_decompose(ts_log)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.subplot(411)
plt.plot(ts_log, label = 'Original')
plt.legend(bbox_to_anchor=(1.01, 0.5), loc=3, borderaxespad=0,fontsize=30)
plt.subplot(412)
plt.plot(trend, label = 'Trend')
plt.legend(bbox_to_anchor=(1.01, 0.5), loc=3, borderaxespad=0,fontsize=30)
plt.subplot(413)
plt.plot(seasonal, label = 'Seasonality')
plt.legend(bbox_to_anchor=(1.01, 0.5), loc=3, borderaxespad=0,fontsize=30)
plt.subplot(414)
plt.plot(residual, label = 'Residual')
plt.legend(bbox_to_anchor=(1.01, 0.5), loc=3, borderaxespad=0,fontsize=30)
plt.savefig('image.png',dpi=600, bbox_inches='tight')
```

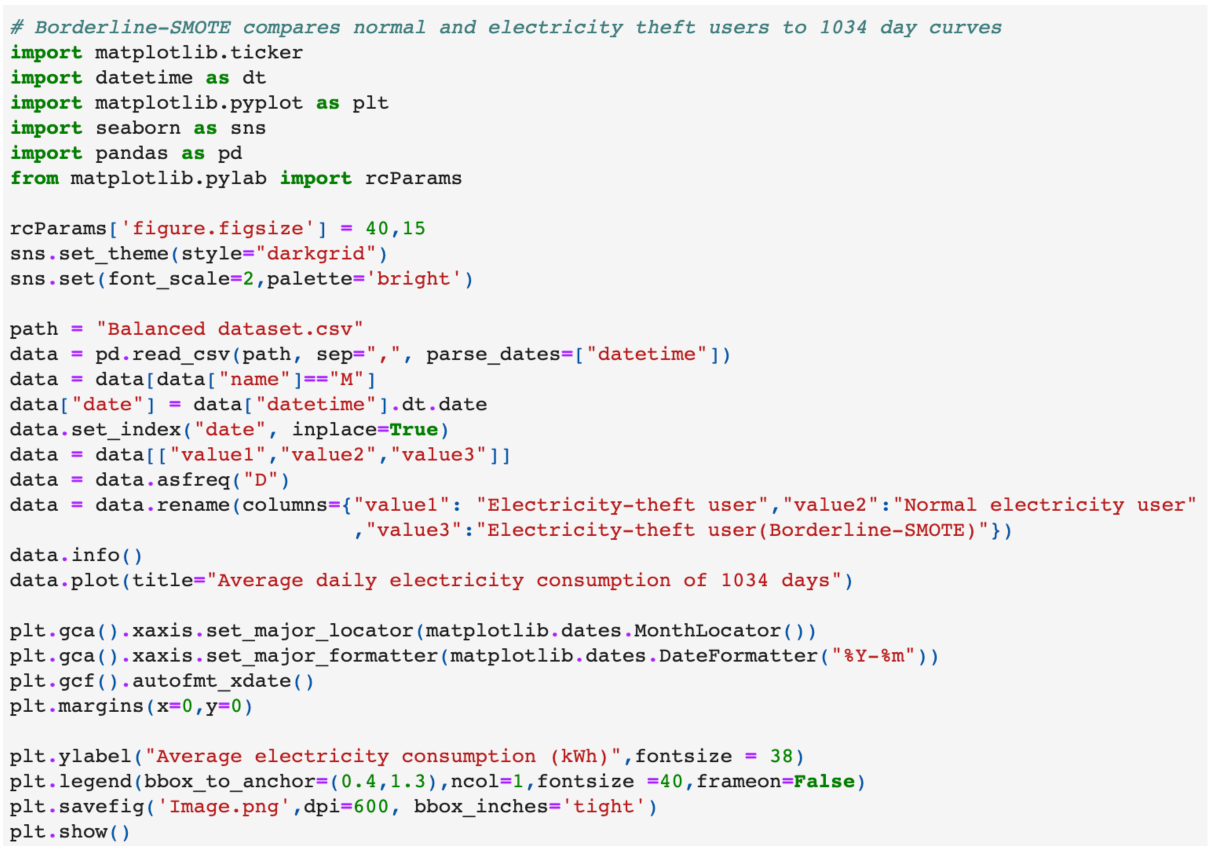
Figure B5: 1034days STL Decomposing (normal users/electricity theft users)

```python
# Borderline-SMOTE balance dataset classification
from imblearn.over_sampling import BorderlineSMOTE
from collections import Counter
from matplotlib import pyplot
from numpy import where
import matplotlib.pyplot as plt
%matplotlib inline
df = pd.read_csv('Project Dataset.csv')
X = df.drop('target', axis = 1)
y =df['target']
counter = Counter(y)
print(counter)
oversample = BorderlineSMOTE()
X, y = oversample.fit_resample(X, y)
counter = Counter(y)
print(counter)
data_res = np.concatenate((X, np.vstack(y)), axis = 1)
data_res = pd.DataFrame(data_res)
data_res.to_csv('Blanced Dataset.csv')

# Blanced dataset visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.patches as mpatches
from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 10,8
sns.set_theme(style="darkgrid")
sns.set(font_scale=1.8,palette='bright')

df = pd.read_csv('Blanced Dataset.csv')

colors = ["#0101DF", "#DF0101"]
sns.countplot('Class', data=df, palette=colors)
plt.title('Class Distributions \n (0: Normal electricity User // 1: Electricity-theft user)', fontsize=14)
plt.savefig('image.png',dpi=600, bbox_inches='tight')
```

Figure B6: Balanced data set classification and visualization

```python
# Data transformation(MinMAX)
import pandas as pd
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv('Project Dataset.csv', header=None)
array = df.values

X = array[:,0:1034]
Y = array[:,1034]
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)

set_printoptions(precision=3)
print(rescaledX[0:5,:])

df = pd.DataFrame(rescaledX)
df.to_csv('Preprocessed dataset.csv')
```

Figure B7: Data transformation(MinMAX)

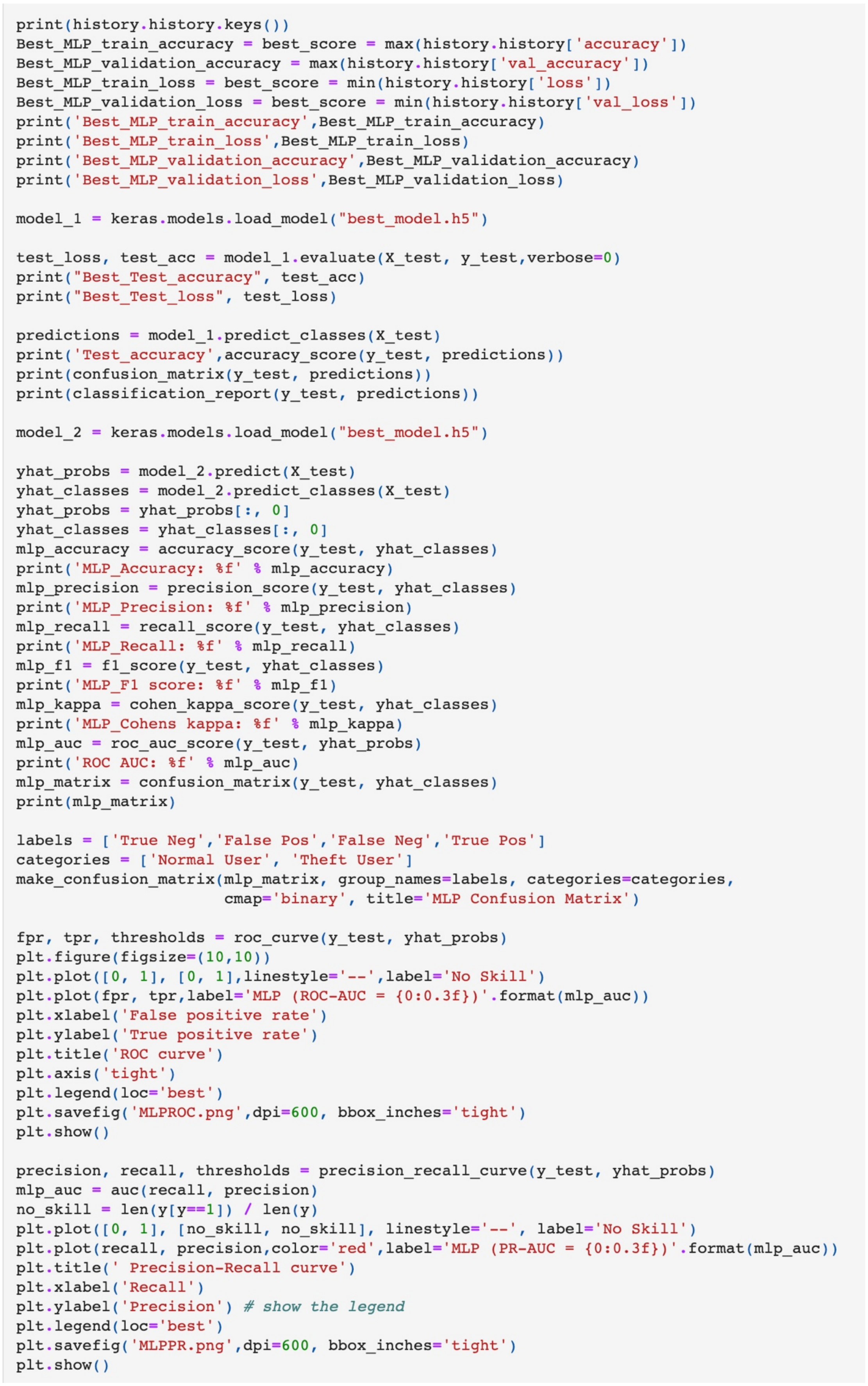


```python
# Borderline-SMOTE compares normal and electricity theft users to 1034 day curves
import matplotlib.ticker
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from matplotlib.pylab import rcParams

rcParams['figure.figsize'] = 40,15
sns.set_theme(style="darkgrid")
sns.set(font_scale=2,palette='bright')

path = "Balanced dataset.csv"
data = pd.read_csv(path, sep=",", parse_dates=["datetime"])
data = data[data["name"]=="M"]
data["date"] = data["datetime"].dt.date
data.set_index("date", inplace=True)
data = data[["value1","value2","value3"]]
data = data.asfreq("D")
data = data.rename(columns={"value1": "Electricity-theft user","value2":"Normal electricity user"
                            ,"value3":"Electricity-theft user(Borderline-SMOTE)"})
data.info()
data.plot(title="Average daily electricity consumption of 1034 days")

plt.gca().xaxis.set_major_locator(matplotlib.dates.MonthLocator())
plt.gca().xaxis.set_major_formatter(matplotlib.dates.DateFormatter("%Y-%m"))
plt.gcf().autofmt_xdate()
plt.margins(x=0,y=0)

plt.ylabel("Average electricity consumption (kWh)",fontsize = 38)
plt.legend(bbox_to_anchor=(0.4,1.3),ncol=1,fontsize =40,frameon=False)
plt.savefig('Image.png',dpi=600, bbox_inches='tight')
plt.show()
```

Figure B8: Comparison of electricity theft user data generated by Borderline-SMOTE

CNN-LSTM and ConvLSTM model training times on laptops ranged from a few hours to more than a dozen hours at a time. The training time on the free cloud GPU is also up to about half an hour. Since there is a time limit on the use of the free cloud GPU, the modeling code and the model evaluation code are integrated into one process in order to save time during modeling, which speeds up the efficiency of model tuning. The programming code is detailed in the following pictures, in the order of MLP, CNN-LSTM and ConvLSTM:

```
import numpy
import numpy as np
import matplotlib.ticker
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import callbacks
from keras.layers import Dropout
from keras.callbacks import ModelCheckpoint
from keras.layers import TimeDistributed
from keras.layers.convolutional import Conv1D
from keras.layers import Flatten
from keras.layers.convolutional import MaxPooling1D
from keras.layers import ConvLSTM2D
from keras.layers import LSTM
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.model_selection import train_test_split
from sklearn.metrics import average_precision_score
from matplotlib.pylab import rcParams
import sys
sys.path.insert(1, '/Users/gaoxiaoxin/zzzzz1')
import cf_matrix
from cf_matrix import make_confusion_matrix
sns.set_context('talk')

rcParams['figure.figsize'] = 10,6
sns.set_theme(style="darkgrid")
sns.set(font_scale=1.5,palette='bright')

dataset = numpy.loadtxt("Preprocessed dataset.csv", delimiter=",")
X = dataset[:,0:1034]
y = dataset[:,1034]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = Sequential()
model.add(Dense(1034, input_dim=1034,  activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(365, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,  activation='sigmoid'))
callbacks = [
    keras.callbacks.ModelCheckpoint(
        "best_model.h5", save_best_only=True, monitor="val_loss"
    ),
    callbacks.EarlyStopping(monitor="val_loss", patience=50, verbose=1),
]

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

history = model.fit(X_train, y_train, validation_split=0.2,epochs= 400, callbacks=callbacks,
                    batch_size= 250, verbose=1)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['MLP_train', 'MLP_validation'], bbox_to_anchor=(1.01,0.65),ncol=1)
plt.savefig('MLP accuracy.png',dpi=600, bbox_inches='tight')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['MLP_train', 'MLP_validation'], bbox_to_anchor=(1.01,0.65),ncol=1)
plt.savefig('MLPLOSS.png',dpi=600, bbox_inches='tight')
plt.show()
```



```
print(history.history.keys())
Best_MLP_train_accuracy = best_score = max(history.history['accuracy'])
Best_MLP_validation_accuracy = max(history.history['val_accuracy'])
Best_MLP_train_loss = best_score = min(history.history['loss'])
Best_MLP_validation_loss = best_score = min(history.history['val_loss'])
print('Best_MLP_train_accuracy',Best_MLP_train_accuracy)
print('Best_MLP_train_loss',Best_MLP_train_loss)
print('Best_MLP_validation_accuracy',Best_MLP_validation_accuracy)
print('Best_MLP_validation_loss',Best_MLP_validation_loss)

model_1 = keras.models.load_model("best_model.h5")

test_loss, test_acc = model_1.evaluate(X_test, y_test,verbose=0)
print("Best_Test_accuracy", test_acc)
print("Best_Test_loss", test_loss)

predictions = model_1.predict_classes(X_test)
print('Test_accuracy',accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))

model_2 = keras.models.load_model("best_model.h5")

yhat_probs = model_2.predict(X_test)
yhat_classes = model_2.predict_classes(X_test)
yhat_probs = yhat_probs[:, 0]
yhat_classes = yhat_classes[:, 0]
mlp_accuracy = accuracy_score(y_test, yhat_classes)
print('MLP_Accuracy: %f' % mlp_accuracy)
mlp_precision = precision_score(y_test, yhat_classes)
print('MLP_Precision: %f' % mlp_precision)
mlp_recall = recall_score(y_test, yhat_classes)
print('MLP_Recall: %f' % mlp_recall)
mlp_f1 = f1_score(y_test, yhat_classes)
print('MLP_F1 score: %f' % mlp_f1)
mlp_kappa = cohen_kappa_score(y_test, yhat_classes)
print('MLP_Cohens kappa: %f' % mlp_kappa)
mlp_auc = roc_auc_score(y_test, yhat_probs)
print('ROC AUC: %f' % mlp_auc)
mlp_matrix = confusion_matrix(y_test, yhat_classes)
print(mlp_matrix)

labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['Normal User', 'Theft User']
make_confusion_matrix(mlp_matrix, group_names=labels, categories=categories,
                      cmap='binary', title='MLP Confusion Matrix')

fpr, tpr, thresholds = roc_curve(y_test, yhat_probs)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1],linestyle='--',label='No Skill')
plt.plot(fpr, tpr,label='MLP (ROC-AUC = {0:0.3f})'.format(mlp_auc))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.axis('tight')
plt.legend(loc='best')
plt.savefig('MLPROC.png',dpi=600, bbox_inches='tight')
plt.show()

precision, recall, thresholds = precision_recall_curve(y_test, yhat_probs)
mlp_auc = auc(recall, precision)
no_skill = len(y[y==1]) / len(y)
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
plt.plot(recall, precision,color='red',label='MLP (PR-AUC = {0:0.3f})'.format(mlp_auc))
plt.title(' Precision-Recall curve')
plt.xlabel('Recall')
plt.ylabel('Precision') # show the legend
plt.legend(loc='best')
plt.savefig('MLPPR.png',dpi=600, bbox_inches='tight')
plt.show()
```

Figure B9: MLP

```python
import tensorflow as tf
import numpy
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
import matplotlib.ticker
import seaborn as sns
from tensorflow import keras
from keras import callbacks
from keras.callbacks import ModelCheckpoint
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras.layers import Dropout
from tensorflow.keras.layers import LSTM
from tensorflow.python.keras.layers import TimeDistributed
from tensorflow.python.keras.layers.convolutional import Conv1D
from tensorflow.python.keras.layers.convolutional import MaxPooling1D
from tensorflow.python.keras.layers import Flatten
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.model_selection import train_test_split
from matplotlib.pylab import rcParams
import sys
sys.path.insert(1, '/cf_matrix')
import cf_matrix
from cf_matrix import make_confusion_matrix
sns.set_context('talk')

rcParams['figure.figsize'] = 10,6
sns.set_theme(style="darkgrid")
sns.set(font_scale=1.5,palette='bright')
dataset = numpy.loadtxt("Preprocessed Dataset.csv", delimiter=",")

X = dataset[:,0:1034]
y = dataset[:,1034]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
y_test = np.reshape(y_test, (3310, 1))
X_test = np.reshape(X_test,(3310,1034,-1))
y_train = np.reshape(y_train, (13240, 1))
X_train = np.reshape(X_train, (13240, 1034, -1))

n_features, n_outputs = X_train.shape[2], y_train.shape[1]
n_steps, n_length = 11, 94
X_train = X_train.reshape((X_train.shape[0], n_steps, n_length, n_features))
X_test = X_test.reshape((X_test.shape[0], n_steps, n_length, n_features))

model = Sequential()
model.add(TimeDistributed(Conv1D(64, 3, activation='relu'),
                          input_shape=(None,n_length,n_features)))
model.add(TimeDistributed(Conv1D(64, 3, activation='relu')))
model.add(TimeDistributed(Dropout(0.2)))
model.add(TimeDistributed(MaxPooling1D()))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(100))
model.add(Dropout(0.2))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='sigmoid'))

callbacks = [
    keras.callbacks.ModelCheckpoint(
        "best_cnnlstm_model.h5", save_best_only=True, monitor="val_loss"
    ),
    callbacks.EarlyStopping(monitor="val_loss", patience=50, verbose=1),
]

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

history = model.fit(X_train, y_train, validation_split=0.2,epochs= 400, callbacks=callbacks,
                    batch_size= 64, verbose=1)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['CNN_LSTM_train', 'CNN_LSTM_validation'], bbox_to_anchor=(1.01,0.65),ncol=1)
plt.savefig('CNN_LSTM Accuracy.png',dpi=600, bbox_inches='tight')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['CNN_LSTM_train', 'CNN_LSTM_validation'], bbox_to_anchor=(1.01,0.65),ncol=1)
plt.savefig('CNN_LSTMLOSS.png',dpi=600, bbox_inches='tight')
plt.show()
```

```
print(history.history.keys())
Best_CNN_LSTM_train_accuracy = best_score = max(history.history['accuracy'])
Best_CNN_LSTM_validation_accuracy = max(history.history['val_accuracy'])
Best_CNN_LSTM_train_loss = best_score = min(history.history['loss'])
Best_CNN_LSTM_validation_loss = best_score = min(history.history['val_loss'])
print('Best_CNN_LSTM_train_accuracy',Best_CNN_LSTM_train_accuracy)
print('Best_CNN_LSTM_train_loss',Best_CNN_LSTM_train_loss)
print('Best_CNN_LSTM_validation_accuracy',Best_CNN_LSTM_validation_accuracy)
print('Best_CNN_LSTM_validation_loss',Best_CNN_LSTM_validation_loss)

model_1 = keras.models.load_model("best_cnnlstm_model.h5")

test_loss, test_acc = model_1.evaluate(X_test, y_test,verbose=0)
print("Best_Test_accuracy", test_acc)
print("Best_Test_loss", test_loss)

predictions = model_1.predict_classes(X_test)
print('Test_accuracy',accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))

model_2 = keras.models.load_model("best_cnnlstm_model.h5")

yhat_probs = model_2.predict(X_test)
yhat_classes = model_2.predict_classes(X_test)
yhat_probs = yhat_probs[:, 0]
yhat_classes = yhat_classes[:, 0]
cnnlstm_accuracy = accuracy_score(y_test, yhat_classes)
print('CNN_LSTM_Accuracy: %f' % cnnlstm_accuracy)
cnnlstm_precision = precision_score(y_test, yhat_classes)
print('CNN_LSTM_Precision: %f' % cnnlstm_precision)
cnnlstm_recall = recall_score(y_test, yhat_classes)
print('CNN_LSTM_Recall: %f' % cnnlstm_recall)
cnnlstm_f1 = f1_score(y_test, yhat_classes)
print('CNN_LSTM_F1 score: %f' % cnnlstm_f1)
cnnlstm_kappa = cohen_kappa_score(y_test, yhat_classes)
print('CNN_LSTM_Cohens kappa: %f' % cnnlstm_kappa)
cnnlstm_auc = roc_auc_score(y_test, yhat_probs)
print('CNN_LSTM ROC-AUC: %f' % cnnlstm_auc)
cnnlstm_matrix = confusion_matrix(y_test, yhat_classes)
print(cnnlstm_matrix)

labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['Normal User', 'Theft User']
make_confusion_matrix(cnnlstm_matrix, group_names=labels, categories=categories,
                      cmap='binary', title='CNN-LSTM Confusion Matrix')

fpr, tpr, thresholds = roc_curve(y_test, yhat_probs)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1],linestyle='--',label='No Skill')
plt.plot(fpr, tpr,label='CNN-LSTM (ROC-AUC = {0:0.3f})'.format(cnnlstm_auc))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.axis('tight')
plt.legend(loc='best')
plt.savefig('cnnlstmROC.png',dpi=600, bbox_inches='tight')
plt.show()

precision, recall, thresholds = precision_recall_curve(y_test, yhat_probs)
auc = auc(recall, precision)
no_skill = len(y[y==1]) / len(y)
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
plt.plot(recall, precision,color='red',label='CNN-LSTM (PR-AUC = {0:0.3f})'.format(auc))
plt.title(' Precision-Recall curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='best')
plt.savefig('cnnlstmPR.png',dpi=600, bbox_inches='tight')
plt.show()
```

Figure B10: CNN-LSTM

```
from tensorflow.python.keras.layers import BatchNormalization
import tensorflow as tf
import numpy
import numpy as np
import matplotlib.pyplot as plt
import tensorflow
import matplotlib.ticker
import seaborn as sns
from tensorflow import keras
from keras import callbacks
from keras.callbacks import ModelCheckpoint
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras.layers import Dropout
from tensorflow.keras.layers import LSTM
from tensorflow.python.keras.layers import TimeDistributed
from tensorflow.python.keras.layers.convolutional import Conv1D
from tensorflow.python.keras.layers.convolutional import MaxPooling1D
from tensorflow.python.keras.layers import ConvLSTM2D
from tensorflow.python.keras.layers import Flatten
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from sklearn.model_selection import train_test_split
from matplotlib.pylab import rcParams
import sys
sys.path.insert(1, '/cf_matrix')
import cf_matrix
from cf_matrix import make_confusion_matrix
sns.set_context('talk')
rcParams['figure.figsize'] = 10,6
sns.set_theme(style="darkgrid")
sns.set(font_scale=1.5,palette='bright')

dataset = numpy.loadtxt("Preprocessed Dataset.csv", delimiter=",")
X = dataset[:,0:1034]
y = dataset[:,1034]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
y_test = np.reshape(y_test, (3310, 1))
X_test = np.reshape(X_test,(3310,1034,-1))
y_train = np.reshape(y_train, (13240, 1))
X_train = np.reshape(X_train, (13240, 1034, -1))

n_timesteps, n_features, n_outputs = X_train.shape[1], X_train.shape[2], y_train.shape[1]
n_steps, n_length = 11, 94
X_train = X_train.reshape((X_train.shape[0], n_steps, 1, n_length, n_features))
X_test = X_test.reshape((X_test.shape[0], n_steps, 1, n_length, n_features))

model = Sequential()
model.add(ConvLSTM2D(64, (1,3), activation='relu', input_shape=(n_steps, 1, n_length,n_features)))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(200, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(n_outputs, activation='sigmoid'))

callbacks = [
    keras.callbacks.ModelCheckpoint(
        "best_ConvLSTM_model.h5", save_best_only=True, monitor="val_loss"
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor="val_loss", factor=0.5, patience=20, min_lr=0.0001
    ),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=50, verbose=1),
]

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

history = model.fit(X_train, y_train, validation_split=0.2,epochs= 400, callbacks=callbacks,
                    batch_size= 250, verbose=1)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['ConvLSTM_train', 'ConvLSTM_validation'], bbox_to_anchor=(1.01,0.65),ncol=1)
plt.savefig('ConvLSTM accuracy.png',dpi=600, bbox_inches='tight')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['ConvLSTM_train', 'ConvLSTM_validation'], bbox_to_anchor=(1.01,0.65),ncol=1)
plt.savefig('ConvLSTM.png',dpi=600, bbox_inches='tight')
plt.show()
```

```python
print(history.history.keys())
Best_ConvLSTM_train_accuracy = best_score = max(history.history['accuracy'])
Best_ConvLSTM_validation_accuracy = max(history.history['val_accuracy'])
Best_ConvLSTM_train_loss = best_score = min(history.history['loss'])
Best_ConvLSTM_validation_loss = best_score = min(history.history['val_loss'])
print('Best_ConvLSTM_train_accuracy',Best_ConvLSTM_train_accuracy)
print('Best_ConvLSTM_train_loss',Best_ConvLSTM_train_loss)
print('Best_ConvLSTM_validation_accuracy',Best_ConvLSTM_validation_accuracy)
print('Best_ConvLSTM_validation_loss',Best_ConvLSTM_validation_loss)

model_1 = keras.models.load_model("best_ConvLSTM_model.h5")

test_loss, test_acc = model_1.evaluate(X_test, y_test,verbose=0)
print("Best_Test_accuracy", test_acc)
print("Best_Test_loss", test_loss)

predictions = model_1.predict_classes(X_test)
print('Test_accuracy',accuracy_score(y_test, predictions))
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))

model_2 = keras.models.load_model("best_ConvLSTM_model.h5")

yhat_probs = model_2.predict(X_test)
yhat_classes = model_2.predict_classes(X_test)
yhat_probs = yhat_probs[:, 0]
yhat_classes = yhat_classes[:, 0]
convlstm_accuracy = accuracy_score(y_test, yhat_classes)
print('ConvLSTM_Accuracy: %f' % convlstm_accuracy)
convlst_precision = precision_score(y_test, yhat_classes)
print('ConvLSTM_Precision: %f' % convlst_precision)
convlst_recall = recall_score(y_test, yhat_classes)
print('ConvLSTM_Recall: %f' % convlst_recall)
convlst_f1 = f1_score(y_test, yhat_classes)
print('ConvLSTM_F1 score: %f' % convlst_f1)
convlst_kappa = cohen_kappa_score(y_test, yhat_classes)
print('ConvLSTM_Cohens kappa: %f' % convlst_kappa)
convlst_auc = roc_auc_score(y_test, yhat_probs)
print('ConvLSTM ROC-AUC: %f' % convlst_auc)
convlst_matrix = confusion_matrix(y_test, yhat_classes)
print(convlst_matrix)

labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['Normal User', 'Theft User']
make_confusion_matrix(convlst_matrix, group_names=labels, categories=categories,
                      cmap='binary', title='ConvLSTM Confusion Matrix')

fpr, tpr, thresholds = roc_curve(y_test, yhat_probs)
plt.figure(figsize=(10,10))
plt.plot([0, 1], [0, 1],linestyle='--',label='No Skill')
plt.plot(fpr, tpr,label='ConvLSTM (ROC-AUC = {0:0.3f})'.format(convlst_auc))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.axis('tight')
plt.legend(loc='best')
plt.savefig('ConvLSTMROC.png',dpi=600, bbox_inches='tight')
plt.show()

precision, recall, thresholds = precision_recall_curve(y_test, yhat_probs)
auc = auc(recall, precision)
no_skill = len(y[y==1]) / len(y)
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
plt.plot(recall, precision,color='red',label='ConvLSTM (PR-AUC = {0:0.3f})'.format(auc))
plt.title(' Precision-Recall curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='best')
plt.savefig('ConvLSTMPR.png',dpi=600, bbox_inches='tight')
plt.show()
```

Figure B11: ConvLSTM

Appendix C - Model Training History Log

The following are the model training history log for MLP, CNN-LSTM and ConvLSTM:

MLP

```
Epoch 1/400
43/43 [==============================] - 1s 16ms/step - loss: 0.6323 - accuracy: 0.6353 -
val_loss: 0.5136 - val_accuracy: 0.7761
Epoch 2/400
43/43 [==============================] - 1s 12ms/step - loss: 0.5292 - accuracy: 0.7698 -
val_loss: 0.4759 - val_accuracy: 0.7983
Epoch 3/400
43/43 [==============================] - 1s 12ms/step - loss: 0.4752 - accuracy: 0.7962 -
val_loss: 0.4405 - val_accuracy: 0.8176
Epoch 4/400
43/43 [==============================] - 1s 12ms/step - loss: 0.4475 - accuracy: 0.8048 -
val_loss: 0.4144 - val_accuracy: 0.8255
Epoch 5/400
43/43 [==============================] - 1s 12ms/step - loss: 0.4173 - accuracy: 0.8157 -
val_loss: 0.3908 - val_accuracy: 0.8395
Epoch 6/400
43/43 [==============================] - 1s 12ms/step - loss: 0.4028 - accuracy: 0.8336 -
val_loss: 0.3671 - val_accuracy: 0.8557
Epoch 7/400
43/43 [==============================] - 1s 13ms/step - loss: 0.3709 - accuracy: 0.8481 -
val_loss: 0.3651 - val_accuracy: 0.8429
Epoch 8/400
43/43 [==============================] - 1s 12ms/step - loss: 0.3497 - accuracy: 0.8566 -
val_loss: 0.3439 - val_accuracy: 0.8720
Epoch 9/400
43/43 [==============================] - 1s 13ms/step - loss: 0.3525 - accuracy: 0.8584 -
val_loss: 0.3307 - val_accuracy: 0.8727
Epoch 10/400
43/43 [==============================] - 1s 12ms/step - loss: 0.3228 - accuracy: 0.8767 -
val_loss: 0.3107 - val_accuracy: 0.8875
Epoch 11/400
43/43 [==============================] - 1s 12ms/step - loss: 0.3069 - accuracy: 0.8840 -
val_loss: 0.3034 - val_accuracy: 0.8810
Epoch 12/400
43/43 [==============================] - 0s 11ms/step - loss: 0.2991 - accuracy: 0.8841 -
val_loss: 0.2897 - val_accuracy: 0.8905
Epoch 13/400
43/43 [==============================] - 1s 12ms/step - loss: 0.3016 - accuracy: 0.8783 -
val_loss: 0.2757 - val_accuracy: 0.9003
Epoch 14/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2842 - accuracy: 0.8886 -
val_loss: 0.3157 - val_accuracy: 0.8682
Epoch 15/400
43/43 [==============================] - 0s 12ms/step - loss: 0.2789 - accuracy: 0.8829 -
val_loss: 0.2581 - val_accuracy: 0.9094
Epoch 16/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2682 - accuracy: 0.8947 -
val_loss: 0.2509 - val_accuracy: 0.9079
```

```
Epoch 17/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2492 - accuracy: 0.9056 -
val_loss: 0.2895 - val_accuracy: 0.8905
Epoch 18/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2487 - accuracy: 0.9034 -
val_loss: 0.2352 - val_accuracy: 0.9162
Epoch 19/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2333 - accuracy: 0.9094 -
val_loss: 0.2287 - val_accuracy: 0.9154
Epoch 20/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2248 - accuracy: 0.9140 -
val_loss: 0.2260 - val_accuracy: 0.9188
Epoch 21/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2144 - accuracy: 0.9223 -
val_loss: 0.2584 - val_accuracy: 0.9079
Epoch 22/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2122 - accuracy: 0.9209 -
val_loss: 0.2311 - val_accuracy: 0.9199
Epoch 23/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2128 - accuracy: 0.9191 -
val_loss: 0.2096 - val_accuracy: 0.9316
Epoch 24/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2069 - accuracy: 0.9228 -
val_loss: 0.2050 - val_accuracy: 0.9328
Epoch 25/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1806 - accuracy: 0.9371 -
val_loss: 0.2069 - val_accuracy: 0.9290
Epoch 26/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1841 - accuracy: 0.9350 -
val_loss: 0.1918 - val_accuracy: 0.9388
Epoch 27/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1779 - accuracy: 0.9338 -
val_loss: 0.2458 - val_accuracy: 0.8965
Epoch 28/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1939 - accuracy: 0.9249 -
val_loss: 0.2164 - val_accuracy: 0.9301
Epoch 29/400
43/43 [==============================] - 1s 12ms/step - loss: 0.2119 - accuracy: 0.9248 -
val_loss: 0.2039 - val_accuracy: 0.9222
Epoch 30/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1829 - accuracy: 0.9352 -
val_loss: 0.1962 - val_accuracy: 0.9316
Epoch 31/400
43/43 [==============================] - 1s 14ms/step - loss: 0.1758 - accuracy: 0.9337 -
val_loss: 0.1859 - val_accuracy: 0.9407
Epoch 32/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1578 - accuracy: 0.9464 -
val_loss: 0.1809 - val_accuracy: 0.9377
Epoch 33/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1668 - accuracy: 0.9387 -
val_loss: 0.1807 - val_accuracy: 0.9377
Epoch 34/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1559 - accuracy: 0.9456 -
val_loss: 0.2112 - val_accuracy: 0.9181
Epoch 35/400
```

```
43/43 [==============================] - 1s 12ms/step - loss: 0.1704 - accuracy: 0.9360 -
val_loss: 0.1764 - val_accuracy: 0.9441
Epoch 36/400
43/43 [==============================] - 1s 13ms/step - loss: 0.1528 - accuracy: 0.9477 -
val_loss: 0.1716 - val_accuracy: 0.9449
Epoch 37/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1480 - accuracy: 0.9518 -
val_loss: 0.1720 - val_accuracy: 0.9403
Epoch 38/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1503 - accuracy: 0.9471 -
val_loss: 0.1968 - val_accuracy: 0.9332
Epoch 39/400
43/43 [==============================] - 1s 12ms/step - loss: 0.1430 - accuracy: 0.9486 -
val_loss: 0.1975 - val_accuracy: 0.9335
Epoch 40/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1323 - accuracy: 0.9533 -
val_loss: 0.1776 - val_accuracy: 0.9377
Epoch 41/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1310 - accuracy: 0.9577 -
val_loss: 0.1766 - val_accuracy: 0.9452
Epoch 42/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1403 - accuracy: 0.9521 -
val_loss: 0.1654 - val_accuracy: 0.9460
Epoch 43/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1334 - accuracy: 0.9514 -
val_loss: 0.1935 - val_accuracy: 0.9403
Epoch 44/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1243 - accuracy: 0.9584 -
val_loss: 0.1561 - val_accuracy: 0.9524
Epoch 45/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1280 - accuracy: 0.9563 -
val_loss: 0.1605 - val_accuracy: 0.9483
Epoch 46/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1254 - accuracy: 0.9614 -
val_loss: 0.1607 - val_accuracy: 0.9494
Epoch 47/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1175 - accuracy: 0.9561 -
val_loss: 0.1543 - val_accuracy: 0.9509
Epoch 48/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1096 - accuracy: 0.9644 -
val_loss: 0.1563 - val_accuracy: 0.9513
Epoch 49/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1086 - accuracy: 0.9636 -
val_loss: 0.1448 - val_accuracy: 0.9528
Epoch 50/400
43/43 [==============================] - 0s 12ms/step - loss: 0.0941 - accuracy: 0.9689 -
val_loss: 0.1471 - val_accuracy: 0.9562
Epoch 51/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1177 - accuracy: 0.9581 -
val_loss: 0.1545 - val_accuracy: 0.9490
Epoch 52/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1065 - accuracy: 0.9626 -
val_loss: 0.1575 - val_accuracy: 0.9505
Epoch 53/400
```

```
43/43 [==============================] - 0s 11ms/step - loss: 0.1053 - accuracy: 0.9640 -
val_loss: 0.1478 - val_accuracy: 0.9562
Epoch 54/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1067 - accuracy: 0.9601 -
val_loss: 0.1515 - val_accuracy: 0.9551
Epoch 55/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1038 - accuracy: 0.9618 -
val_loss: 0.1474 - val_accuracy: 0.9509
Epoch 56/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1060 - accuracy: 0.9640 -
val_loss: 0.1583 - val_accuracy: 0.9517
Epoch 57/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1167 - accuracy: 0.9573 -
val_loss: 0.1435 - val_accuracy: 0.9566
Epoch 58/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0979 - accuracy: 0.9671 -
val_loss: 0.1351 - val_accuracy: 0.9577
Epoch 59/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0955 - accuracy: 0.9682 -
val_loss: 0.1540 - val_accuracy: 0.9543
Epoch 60/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0942 - accuracy: 0.9682 -
val_loss: 0.1503 - val_accuracy: 0.9558
Epoch 61/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0831 - accuracy: 0.9718 -
val_loss: 0.1371 - val_accuracy: 0.9577
Epoch 62/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1028 - accuracy: 0.9649 -
val_loss: 0.1425 - val_accuracy: 0.9558
Epoch 63/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0962 - accuracy: 0.9658 -
val_loss: 0.1766 - val_accuracy: 0.9381
Epoch 64/400
43/43 [==============================] - 1s 13ms/step - loss: 0.1159 - accuracy: 0.9562 -
val_loss: 0.1716 - val_accuracy: 0.9468
Epoch 65/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1005 - accuracy: 0.9645 -
val_loss: 0.2008 - val_accuracy: 0.9377
Epoch 66/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0934 - accuracy: 0.9654 -
val_loss: 0.1392 - val_accuracy: 0.9607
Epoch 67/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0862 - accuracy: 0.9694 -
val_loss: 0.1556 - val_accuracy: 0.9535
Epoch 68/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0937 - accuracy: 0.9665 -
val_loss: 0.1387 - val_accuracy: 0.9607
Epoch 69/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0908 - accuracy: 0.9693 -
val_loss: 0.1506 - val_accuracy: 0.9585
Epoch 70/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0861 - accuracy: 0.9735 -
val_loss: 0.1389 - val_accuracy: 0.9641
Epoch 71/400
```

```
43/43 [==============================] - 0s 11ms/step - loss: 0.0954 - accuracy: 0.9682 -
val_loss: 0.1387 - val_accuracy: 0.9596
Epoch 72/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0992 - accuracy: 0.9664 -
val_loss: 0.1368 - val_accuracy: 0.9626
Epoch 73/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0976 - accuracy: 0.9654 -
val_loss: 0.1451 - val_accuracy: 0.9611
Epoch 74/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0931 - accuracy: 0.9682 -
val_loss: 0.1442 - val_accuracy: 0.9607
Epoch 75/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0868 - accuracy: 0.9697 -
val_loss: 0.1409 - val_accuracy: 0.9600
Epoch 76/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0904 - accuracy: 0.9676 -
val_loss: 0.1415 - val_accuracy: 0.9566
Epoch 77/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0926 - accuracy: 0.9673 -
val_loss: 0.1412 - val_accuracy: 0.9588
Epoch 78/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0822 - accuracy: 0.9728 -
val_loss: 0.1478 - val_accuracy: 0.9562
Epoch 79/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0832 - accuracy: 0.9686 -
val_loss: 0.1355 - val_accuracy: 0.9622
Epoch 80/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0934 - accuracy: 0.9683 -
val_loss: 0.1339 - val_accuracy: 0.9619
Epoch 81/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1028 - accuracy: 0.9651 -
val_loss: 0.1348 - val_accuracy: 0.9637
Epoch 82/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0904 - accuracy: 0.9710 -
val_loss: 0.1570 - val_accuracy: 0.9517
Epoch 83/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0904 - accuracy: 0.9685 -
val_loss: 0.1513 - val_accuracy: 0.9592
Epoch 84/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0863 - accuracy: 0.9711 -
val_loss: 0.1457 - val_accuracy: 0.9607
Epoch 85/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0962 - accuracy: 0.9656 -
val_loss: 0.1723 - val_accuracy: 0.9513
Epoch 86/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0867 - accuracy: 0.9703 -
val_loss: 0.1345 - val_accuracy: 0.9615
Epoch 87/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0808 - accuracy: 0.9724 -
val_loss: 0.1469 - val_accuracy: 0.9603
Epoch 88/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0965 - accuracy: 0.9640 -
val_loss: 0.1332 - val_accuracy: 0.9641
Epoch 89/400
```

```
43/43 [==============================] - 0s 11ms/step - loss: 0.0807 - accuracy: 0.9725 -
val_loss: 0.1250 - val_accuracy: 0.9649
Epoch 90/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0741 - accuracy: 0.9751 -
val_loss: 0.1264 - val_accuracy: 0.9626
Epoch 91/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0732 - accuracy: 0.9738 -
val_loss: 0.1379 - val_accuracy: 0.9626
Epoch 92/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0948 - accuracy: 0.9682 -
val_loss: 0.1432 - val_accuracy: 0.9600
Epoch 93/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0903 - accuracy: 0.9680 -
val_loss: 0.1473 - val_accuracy: 0.9603
Epoch 94/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0903 - accuracy: 0.9659 -
val_loss: 0.1584 - val_accuracy: 0.9573
Epoch 95/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0773 - accuracy: 0.9721 -
val_loss: 0.1288 - val_accuracy: 0.9637
Epoch 96/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0838 - accuracy: 0.9686 -
val_loss: 0.1508 - val_accuracy: 0.9588
Epoch 97/400
43/43 [==============================] - 1s 13ms/step - loss: 0.0784 - accuracy: 0.9722 -
val_loss: 0.1392 - val_accuracy: 0.9581
Epoch 98/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0728 - accuracy: 0.9749 -
val_loss: 0.1837 - val_accuracy: 0.9502
Epoch 99/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0805 - accuracy: 0.9721 -
val_loss: 0.1337 - val_accuracy: 0.9607
Epoch 100/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0765 - accuracy: 0.9736 -
val_loss: 0.1434 - val_accuracy: 0.9592
Epoch 101/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0726 - accuracy: 0.9734 -
val_loss: 0.1524 - val_accuracy: 0.9615
Epoch 102/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0675 - accuracy: 0.9794 -
val_loss: 0.1652 - val_accuracy: 0.9543
Epoch 103/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0714 - accuracy: 0.9758 -
val_loss: 0.1428 - val_accuracy: 0.9637
Epoch 104/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0702 - accuracy: 0.9744 -
val_loss: 0.1617 - val_accuracy: 0.9596
Epoch 105/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0701 - accuracy: 0.9741 -
val_loss: 0.1271 - val_accuracy: 0.9634
Epoch 106/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0852 - accuracy: 0.9667 -
val_loss: 0.1374 - val_accuracy: 0.9637
Epoch 107/400
```

```
43/43 [==============================] - 0s 11ms/step - loss: 0.0769 - accuracy: 0.9736 -
val_loss: 0.1699 - val_accuracy: 0.9532
Epoch 108/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0905 - accuracy: 0.9671 -
val_loss: 0.1631 - val_accuracy: 0.9528
Epoch 109/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0912 - accuracy: 0.9649 -
val_loss: 0.1931 - val_accuracy: 0.9449
Epoch 110/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1062 - accuracy: 0.9602 -
val_loss: 0.1391 - val_accuracy: 0.9622
Epoch 111/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0891 - accuracy: 0.9658 -
val_loss: 0.1752 - val_accuracy: 0.9566
Epoch 112/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0826 - accuracy: 0.9717 -
val_loss: 0.1388 - val_accuracy: 0.9641
Epoch 113/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0768 - accuracy: 0.9716 -
val_loss: 0.1595 - val_accuracy: 0.9585
Epoch 114/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0767 - accuracy: 0.9745 -
val_loss: 0.1711 - val_accuracy: 0.9486
Epoch 115/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0914 - accuracy: 0.9679 -
val_loss: 0.1337 - val_accuracy: 0.9622
Epoch 116/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0780 - accuracy: 0.9700 -
val_loss: 0.1387 - val_accuracy: 0.9588
Epoch 117/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0764 - accuracy: 0.9721 -
val_loss: 0.1209 - val_accuracy: 0.9653
Epoch 118/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0730 - accuracy: 0.9738 -
val_loss: 0.1439 - val_accuracy: 0.9660
Epoch 119/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0710 - accuracy: 0.9739 -
val_loss: 0.1586 - val_accuracy: 0.9577
Epoch 120/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0915 - accuracy: 0.9659 -
val_loss: 0.1515 - val_accuracy: 0.9588
Epoch 121/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0836 - accuracy: 0.9718 -
val_loss: 0.1575 - val_accuracy: 0.9600
Epoch 122/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0693 - accuracy: 0.9752 -
val_loss: 0.1534 - val_accuracy: 0.9615
Epoch 123/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0739 - accuracy: 0.9734 -
val_loss: 0.1399 - val_accuracy: 0.9630
Epoch 124/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0719 - accuracy: 0.9728 -
val_loss: 0.1605 - val_accuracy: 0.9611
Epoch 125/400
```

```
43/43 [==============================] - 0s 11ms/step - loss: 0.0715 - accuracy: 0.9750 -
val_loss: 0.1534 - val_accuracy: 0.9630
Epoch 126/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0703 - accuracy: 0.9777 -
val_loss: 0.1620 - val_accuracy: 0.9569
Epoch 127/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0821 - accuracy: 0.9702 -
val_loss: 0.1843 - val_accuracy: 0.9539
Epoch 128/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1115 - accuracy: 0.9594 -
val_loss: 0.1762 - val_accuracy: 0.9517
Epoch 129/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0851 - accuracy: 0.9723 -
val_loss: 0.1460 - val_accuracy: 0.9607
Epoch 130/400
43/43 [==============================] - 1s 13ms/step - loss: 0.0744 - accuracy: 0.9743 -
val_loss: 0.1773 - val_accuracy: 0.9509
Epoch 131/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0738 - accuracy: 0.9736 -
val_loss: 0.1424 - val_accuracy: 0.9615
Epoch 132/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0847 - accuracy: 0.9697 -
val_loss: 0.1480 - val_accuracy: 0.9653
Epoch 133/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0897 - accuracy: 0.9662 -
val_loss: 0.1290 - val_accuracy: 0.9641
Epoch 134/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0671 - accuracy: 0.9777 -
val_loss: 0.1417 - val_accuracy: 0.9611
Epoch 135/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0655 - accuracy: 0.9785 -
val_loss: 0.1516 - val_accuracy: 0.9573
Epoch 136/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0803 - accuracy: 0.9705 -
val_loss: 0.1600 - val_accuracy: 0.9566
Epoch 137/400
43/43 [==============================] - 0s 11ms/step - loss: 0.1182 - accuracy: 0.9570 -
val_loss: 0.1459 - val_accuracy: 0.9649
Epoch 138/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0744 - accuracy: 0.9739 -
val_loss: 0.1693 - val_accuracy: 0.9577
Epoch 139/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0805 - accuracy: 0.9697 -
val_loss: 0.1798 - val_accuracy: 0.9509
Epoch 140/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0811 - accuracy: 0.9695 -
val_loss: 0.1430 - val_accuracy: 0.9615
Epoch 141/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0864 - accuracy: 0.9705 -
val_loss: 0.1591 - val_accuracy: 0.9619
Epoch 142/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0786 - accuracy: 0.9724 -
val_loss: 0.1665 - val_accuracy: 0.9596
Epoch 143/400
```

```
43/43 [==============================] - 0s 11ms/step - loss: 0.0672 - accuracy: 0.9756 -
val_loss: 0.1657 - val_accuracy: 0.9573
Epoch 144/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0744 - accuracy: 0.9728 -
val_loss: 0.1555 - val_accuracy: 0.9615
Epoch 145/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0816 - accuracy: 0.9690 -
val_loss: 0.1403 - val_accuracy: 0.9637
Epoch 146/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0837 - accuracy: 0.9680 -
val_loss: 0.1301 - val_accuracy: 0.9683
Epoch 147/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0654 - accuracy: 0.9777 -
val_loss: 0.1522 - val_accuracy: 0.9634
Epoch 148/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0628 - accuracy: 0.9777 -
val_loss: 0.1415 - val_accuracy: 0.9645
Epoch 149/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0767 - accuracy: 0.9724 -
val_loss: 0.1752 - val_accuracy: 0.9603
Epoch 150/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0792 - accuracy: 0.9724 -
val_loss: 0.1531 - val_accuracy: 0.9611
Epoch 151/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0779 - accuracy: 0.9720 -
val_loss: 0.1255 - val_accuracy: 0.9683
Epoch 152/400
43/43 [==============================] - 0s 12ms/step - loss: 0.0683 - accuracy: 0.9753 -
val_loss: 0.1574 - val_accuracy: 0.9554
Epoch 153/400
43/43 [==============================] - 0s 12ms/step - loss: 0.0747 - accuracy: 0.9740 -
val_loss: 0.1456 - val_accuracy: 0.9619
Epoch 154/400
43/43 [==============================] - 0s 12ms/step - loss: 0.0736 - accuracy: 0.9724 -
val_loss: 0.1358 - val_accuracy: 0.9679
Epoch 155/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0652 - accuracy: 0.9751 -
val_loss: 0.1756 - val_accuracy: 0.9566
Epoch 156/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0662 - accuracy: 0.9758 -
val_loss: 0.1357 - val_accuracy: 0.9660
Epoch 157/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0641 - accuracy: 0.9785 -
val_loss: 0.1455 - val_accuracy: 0.9603
Epoch 158/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0666 - accuracy: 0.9750 -
val_loss: 0.1252 - val_accuracy: 0.9717
Epoch 159/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0702 - accuracy: 0.9779 -
val_loss: 0.1487 - val_accuracy: 0.9645
Epoch 160/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0671 - accuracy: 0.9750 -
val_loss: 0.1491 - val_accuracy: 0.9630
Epoch 161/400
```

```
43/43 [==============================] - 0s 11ms/step - loss: 0.0600 - accuracy: 0.9788 -
val_loss: 0.1792 - val_accuracy: 0.9562
Epoch 162/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0723 - accuracy: 0.9737 -
val_loss: 0.1414 - val_accuracy: 0.9653
Epoch 163/400
43/43 [==============================] - 1s 13ms/step - loss: 0.0966 - accuracy: 0.9656 -
val_loss: 0.1573 - val_accuracy: 0.9630
Epoch 164/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0758 - accuracy: 0.9748 -
val_loss: 0.1340 - val_accuracy: 0.9671
Epoch 165/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0756 - accuracy: 0.9729 -
val_loss: 0.1249 - val_accuracy: 0.9687
Epoch 166/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0686 - accuracy: 0.9738 -
val_loss: 0.1469 - val_accuracy: 0.9675
Epoch 167/400
43/43 [==============================] - 0s 11ms/step - loss: 0.0688 - accuracy: 0.9757 -
val_loss: 0.1408 - val_accuracy: 0.9660
Epoch 00167: early stopping
```

CNN-LSTM

```
Epoch 1/400
166/166 [==============================] - 7s 30ms/step - loss: 0.6224 - accuracy: 0.6390
- val_loss: 0.5032 - val_accuracy: 0.7761
Epoch 2/400
166/166 [==============================] - 5s 28ms/step - loss: 0.5096 - accuracy: 0.7647
- val_loss: 0.4662 - val_accuracy: 0.7829
Epoch 3/400
166/166 [==============================] - 5s 27ms/step - loss: 0.4738 - accuracy: 0.7909
- val_loss: 0.4502 - val_accuracy: 0.7991
Epoch 4/400
166/166 [==============================] - 5s 28ms/step - loss: 0.4566 - accuracy: 0.8027
- val_loss: 0.4309 - val_accuracy: 0.8180
Epoch 5/400
166/166 [==============================] - 5s 28ms/step - loss: 0.4288 - accuracy: 0.8164
- val_loss: 0.4294 - val_accuracy: 0.8017
Epoch 6/400
166/166 [==============================] - 5s 28ms/step - loss: 0.4138 - accuracy: 0.8212
- val_loss: 0.3896 - val_accuracy: 0.8361
Epoch 7/400
166/166 [==============================] - 5s 28ms/step - loss: 0.3934 - accuracy: 0.8347
- val_loss: 0.3888 - val_accuracy: 0.8406
Epoch 8/400
166/166 [==============================] - 5s 28ms/step - loss: 0.3941 - accuracy: 0.8381
- val_loss: 0.3724 - val_accuracy: 0.8471
Epoch 9/400
166/166 [==============================] - 5s 28ms/step - loss: 0.3637 - accuracy: 0.8498
- val_loss: 0.3435 - val_accuracy: 0.8640
Epoch 10/400
166/166 [==============================] - 5s 28ms/step - loss: 0.3331 - accuracy: 0.8633
- val_loss: 0.3659 - val_accuracy: 0.8459
```

```
Epoch 11/400
166/166 [==============================] - 5s 28ms/step - loss: 0.3264 - accuracy: 0.8701
- val_loss: 0.3358 - val_accuracy: 0.8697
Epoch 12/400
166/166 [==============================] - 5s 28ms/step - loss: 0.3212 - accuracy: 0.8760
- val_loss: 0.3459 - val_accuracy: 0.8542
Epoch 13/400
166/166 [==============================] - 5s 28ms/step - loss: 0.2853 - accuracy: 0.8899
- val_loss: 0.3073 - val_accuracy: 0.8860
Epoch 14/400
166/166 [==============================] - 5s 28ms/step - loss: 0.2557 - accuracy: 0.9068
- val_loss: 0.2836 - val_accuracy: 0.8875
Epoch 15/400
166/166 [==============================] - 5s 28ms/step - loss: 0.2525 - accuracy: 0.9060
- val_loss: 0.2612 - val_accuracy: 0.9014
Epoch 16/400
166/166 [==============================] - 5s 28ms/step - loss: 0.2211 - accuracy: 0.9162
- val_loss: 0.2793 - val_accuracy: 0.9052
Epoch 17/400
166/166 [==============================] - 5s 28ms/step - loss: 0.1988 - accuracy: 0.9258
- val_loss: 0.2415 - val_accuracy: 0.9116
Epoch 18/400
166/166 [==============================] - 5s 29ms/step - loss: 0.1870 - accuracy: 0.9306
- val_loss: 0.2399 - val_accuracy: 0.9169
Epoch 19/400
166/166 [==============================] - 5s 28ms/step - loss: 0.1531 - accuracy: 0.9459
- val_loss: 0.1972 - val_accuracy: 0.9354
Epoch 20/400
166/166 [==============================] - 5s 28ms/step - loss: 0.1394 - accuracy: 0.9526
- val_loss: 0.2102 - val_accuracy: 0.9324
Epoch 21/400
166/166 [==============================] - 5s 28ms/step - loss: 0.1181 - accuracy: 0.9576
- val_loss: 0.2025 - val_accuracy: 0.9320
Epoch 22/400
166/166 [==============================] - 5s 28ms/step - loss: 0.1232 - accuracy: 0.9537
- val_loss: 0.1941 - val_accuracy: 0.9449
Epoch 23/400
166/166 [==============================] - 5s 28ms/step - loss: 0.1068 - accuracy: 0.9623
- val_loss: 0.1716 - val_accuracy: 0.9547
Epoch 24/400
166/166 [==============================] - 5s 28ms/step - loss: 0.1034 - accuracy: 0.9665
- val_loss: 0.1907 - val_accuracy: 0.9468
Epoch 25/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0919 - accuracy: 0.9692
- val_loss: 0.1766 - val_accuracy: 0.9539
Epoch 26/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0758 - accuracy: 0.9723
- val_loss: 0.1799 - val_accuracy: 0.9539
Epoch 27/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0634 - accuracy: 0.9789
- val_loss: 0.1861 - val_accuracy: 0.9585
Epoch 28/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0622 - accuracy: 0.9774
- val_loss: 0.1796 - val_accuracy: 0.9513
Epoch 29/400
```

```
166/166 [==============================] - 5s 28ms/step - loss: 0.0619 - accuracy: 0.9791
- val_loss: 0.1842 - val_accuracy: 0.9569
Epoch 30/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0514 - accuracy: 0.9826
- val_loss: 0.1892 - val_accuracy: 0.9585
Epoch 31/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0650 - accuracy: 0.9748
- val_loss: 0.1676 - val_accuracy: 0.9626
Epoch 32/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0509 - accuracy: 0.9821
- val_loss: 0.1699 - val_accuracy: 0.9611
Epoch 33/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0418 - accuracy: 0.9865
- val_loss: 0.1943 - val_accuracy: 0.9634
Epoch 34/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0602 - accuracy: 0.9800
- val_loss: 0.1851 - val_accuracy: 0.9607
Epoch 35/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0465 - accuracy: 0.9818
- val_loss: 0.1686 - val_accuracy: 0.9687
Epoch 36/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0396 - accuracy: 0.9870
- val_loss: 0.1627 - val_accuracy: 0.9732
Epoch 37/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0279 - accuracy: 0.9906
- val_loss: 0.1801 - val_accuracy: 0.9637
Epoch 38/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0271 - accuracy: 0.9915
- val_loss: 0.2220 - val_accuracy: 0.9543
Epoch 39/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0430 - accuracy: 0.9845
- val_loss: 0.1678 - val_accuracy: 0.9683
Epoch 40/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0274 - accuracy: 0.9906
- val_loss: 0.1527 - val_accuracy: 0.9721
Epoch 41/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0212 - accuracy: 0.9927
- val_loss: 0.1838 - val_accuracy: 0.9687
Epoch 42/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0242 - accuracy: 0.9921
- val_loss: 0.1827 - val_accuracy: 0.9656
Epoch 43/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0236 - accuracy: 0.9920
- val_loss: 0.1605 - val_accuracy: 0.9690
Epoch 44/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0179 - accuracy: 0.9935
- val_loss: 0.1753 - val_accuracy: 0.9687
Epoch 45/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0321 - accuracy: 0.9890
- val_loss: 0.2094 - val_accuracy: 0.9637
Epoch 46/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0225 - accuracy: 0.9915
- val_loss: 0.1675 - val_accuracy: 0.9664
Epoch 47/400
```

```
166/166 [==============================] - 5s 28ms/step - loss: 0.0274 - accuracy: 0.9904
- val_loss: 0.1672 - val_accuracy: 0.9653
Epoch 48/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0241 - accuracy: 0.9925
- val_loss: 0.1639 - val_accuracy: 0.9705
Epoch 49/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0225 - accuracy: 0.9933
- val_loss: 0.1612 - val_accuracy: 0.9683
Epoch 50/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0182 - accuracy: 0.9937
- val_loss: 0.1757 - val_accuracy: 0.9690
Epoch 51/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0113 - accuracy: 0.9968
- val_loss: 0.1894 - val_accuracy: 0.9690
Epoch 52/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0182 - accuracy: 0.9936
- val_loss: 0.1906 - val_accuracy: 0.9656
Epoch 53/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0191 - accuracy: 0.9931
- val_loss: 0.2398 - val_accuracy: 0.9668
Epoch 54/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0269 - accuracy: 0.9917
- val_loss: 0.1760 - val_accuracy: 0.9751
Epoch 55/400
166/166 [==============================] - 5s 27ms/step - loss: 0.0102 - accuracy: 0.9963
- val_loss: 0.1888 - val_accuracy: 0.9728
Epoch 56/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0085 - accuracy: 0.9971
- val_loss: 0.1966 - val_accuracy: 0.9687
Epoch 57/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0150 - accuracy: 0.9959
- val_loss: 0.2047 - val_accuracy: 0.9702
Epoch 58/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0214 - accuracy: 0.9935
- val_loss: 0.2134 - val_accuracy: 0.9653
Epoch 59/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0162 - accuracy: 0.9940
- val_loss: 0.1952 - val_accuracy: 0.9645
Epoch 60/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0235 - accuracy: 0.9907
- val_loss: 0.1991 - val_accuracy: 0.9656
Epoch 61/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0151 - accuracy: 0.9949
- val_loss: 0.1721 - val_accuracy: 0.9713
Epoch 62/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0112 - accuracy: 0.9968
- val_loss: 0.2048 - val_accuracy: 0.9694
Epoch 63/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0165 - accuracy: 0.9951
- val_loss: 0.2197 - val_accuracy: 0.9687
Epoch 64/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0222 - accuracy: 0.9930
- val_loss: 0.1587 - val_accuracy: 0.9728
Epoch 65/400
```

```
166/166 [==============================] - 5s 28ms/step - loss: 0.0052 - accuracy: 0.9992
- val_loss: 0.1775 - val_accuracy: 0.9732
Epoch 66/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0047 - accuracy: 0.9986
- val_loss: 0.2018 - val_accuracy: 0.9717
Epoch 67/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0081 - accuracy: 0.9979
- val_loss: 0.2532 - val_accuracy: 0.9581
Epoch 68/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0150 - accuracy: 0.9954
- val_loss: 0.2075 - val_accuracy: 0.9660
Epoch 69/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0096 - accuracy: 0.9972
- val_loss: 0.2006 - val_accuracy: 0.9687
Epoch 70/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0232 - accuracy: 0.9915
- val_loss: 0.1875 - val_accuracy: 0.9721
Epoch 71/400
166/166 [==============================] - 5s 27ms/step - loss: 0.0030 - accuracy: 0.9993
- val_loss: 0.2027 - val_accuracy: 0.9705
Epoch 72/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0158 - accuracy: 0.9943
- val_loss: 0.1977 - val_accuracy: 0.9649
Epoch 73/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0092 - accuracy: 0.9976
- val_loss: 0.1763 - val_accuracy: 0.9747
Epoch 74/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0033 - accuracy: 0.9989
- val_loss: 0.2588 - val_accuracy: 0.9619
Epoch 75/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0211 - accuracy: 0.9945
- val_loss: 0.1723 - val_accuracy: 0.9717
Epoch 76/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0093 - accuracy: 0.9971
- val_loss: 0.1942 - val_accuracy: 0.9721
Epoch 77/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0071 - accuracy: 0.9973
- val_loss: 0.1773 - val_accuracy: 0.9747
Epoch 78/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0093 - accuracy: 0.9967
- val_loss: 0.2371 - val_accuracy: 0.9671
Epoch 79/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0231 - accuracy: 0.9922
- val_loss: 0.1763 - val_accuracy: 0.9747
Epoch 80/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0048 - accuracy: 0.9984
- val_loss: 0.1859 - val_accuracy: 0.9739
Epoch 81/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0055 - accuracy: 0.9977
- val_loss: 0.1853 - val_accuracy: 0.9773
Epoch 82/400
166/166 [==============================] - 5s 27ms/step - loss: 0.0118 - accuracy: 0.9956
- val_loss: 0.1868 - val_accuracy: 0.9743
Epoch 83/400
```

```
166/166 [==============================] - 5s 28ms/step - loss: 0.0090 - accuracy: 0.9976
- val_loss: 0.1766 - val_accuracy: 0.9717
Epoch 84/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0063 - accuracy: 0.9982
- val_loss: 0.1858 - val_accuracy: 0.9747
Epoch 85/400
166/166 [==============================] - 5s 29ms/step - loss: 0.0011 - accuracy: 0.9999
- val_loss: 0.1802 - val_accuracy: 0.9770
Epoch 86/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0040 - accuracy: 0.9988
- val_loss: 0.2088 - val_accuracy: 0.9656
Epoch 87/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0072 - accuracy: 0.9971
- val_loss: 0.1722 - val_accuracy: 0.9751
Epoch 88/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0150 - accuracy: 0.9958
- val_loss: 0.1738 - val_accuracy: 0.9762
Epoch 89/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0044 - accuracy: 0.9985
- val_loss: 0.1698 - val_accuracy: 0.9796
Epoch 90/400
166/166 [==============================] - 5s 28ms/step - loss: 0.0038 - accuracy: 0.9988
- val_loss: 0.2613 - val_accuracy: 0.9551
Epoch 00090: early stopping
```

ConvLSTM

```
Epoch 1/400
43/43 [==============================] - 16s 306ms/step - loss: 0.6707 - accuracy: 0.7454
- val_loss: 0.6799 - val_accuracy: 0.5887
Epoch 2/400
43/43 [==============================] - 12s 289ms/step - loss: 0.3762 - accuracy: 0.8463
- val_loss: 0.7627 - val_accuracy: 0.7032
Epoch 3/400
43/43 [==============================] - 12s 289ms/step - loss: 0.2917 - accuracy: 0.8802
- val_loss: 0.5253 - val_accuracy: 0.7122
Epoch 4/400
43/43 [==============================] - 12s 290ms/step - loss: 0.2110 - accuracy: 0.9236
- val_loss: 0.5135 - val_accuracy: 0.7168
Epoch 5/400
43/43 [==============================] - 12s 289ms/step - loss: 0.1472 - accuracy: 0.9507
- val_loss: 0.5011 - val_accuracy: 0.7390
Epoch 6/400
43/43 [==============================] - 12s 290ms/step - loss: 0.1244 - accuracy: 0.9606
- val_loss: 0.4904 - val_accuracy: 0.7304
Epoch 7/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0873 - accuracy: 0.9752
- val_loss: 0.4902 - val_accuracy: 0.7353
Epoch 8/400
43/43 [==============================] - 13s 291ms/step - loss: 0.0637 - accuracy: 0.9829
- val_loss: 0.4525 - val_accuracy: 0.7326
Epoch 9/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0399 - accuracy: 0.9907
- val_loss: 0.4388 - val_accuracy: 0.7576
Epoch 10/400
```

```
43/43 [==============================] - 12s 290ms/step - loss: 0.0422 - accuracy: 0.9891
- val_loss: 0.3440 - val_accuracy: 0.8278
Epoch 11/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0371 - accuracy: 0.9917
- val_loss: 0.3147 - val_accuracy: 0.8844
Epoch 12/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0342 - accuracy: 0.9930
- val_loss: 0.3577 - val_accuracy: 0.8093
Epoch 13/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0166 - accuracy: 0.9975
- val_loss: 0.2772 - val_accuracy: 0.8758
Epoch 14/400
43/43 [==============================] - 12s 291ms/step - loss: 0.0120 - accuracy: 0.9986
- val_loss: 0.2584 - val_accuracy: 0.8852
Epoch 15/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0063 - accuracy: 0.9993
- val_loss: 0.1965 - val_accuracy: 0.9369
Epoch 16/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0071 - accuracy: 0.9989
- val_loss: 0.2024 - val_accuracy: 0.9267
Epoch 17/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0244 - accuracy: 0.9940
- val_loss: 0.1191 - val_accuracy: 0.9721
Epoch 18/400
43/43 [==============================] - 12s 291ms/step - loss: 0.0184 - accuracy: 0.9970
- val_loss: 0.1406 - val_accuracy: 0.9702
Epoch 19/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0089 - accuracy: 0.9987
- val_loss: 0.1188 - val_accuracy: 0.9743
Epoch 20/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0079 - accuracy: 0.9996
- val_loss: 0.1017 - val_accuracy: 0.9811
Epoch 21/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0040 - accuracy: 0.9996
- val_loss: 0.1094 - val_accuracy: 0.9815
Epoch 22/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0023 - accuracy: 0.9999
- val_loss: 0.0954 - val_accuracy: 0.9838
Epoch 23/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0022 - accuracy: 0.9999
- val_loss: 0.0973 - val_accuracy: 0.9823
Epoch 24/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0023 - accuracy: 1.0000
- val_loss: 0.0983 - val_accuracy: 0.9826
Epoch 25/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0017 - accuracy: 1.0000
- val_loss: 0.0952 - val_accuracy: 0.9819
Epoch 26/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0016 - accuracy: 1.0000
- val_loss: 0.0938 - val_accuracy: 0.9823
Epoch 27/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0017 - accuracy: 0.9999
- val_loss: 0.0971 - val_accuracy: 0.9800
Epoch 28/400
```

```
43/43 [==============================] - 12s 290ms/step - loss: 0.0109 - accuracy: 0.9969
- val_loss: 0.1834 - val_accuracy: 0.9569
Epoch 29/400
43/43 [==============================] - 12s 291ms/step - loss: 0.0165 - accuracy: 0.9954
- val_loss: 0.1145 - val_accuracy: 0.9732
Epoch 30/400
43/43 [==============================] - 12s 291ms/step - loss: 0.0086 - accuracy: 0.9982
- val_loss: 0.1008 - val_accuracy: 0.9773
Epoch 31/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0046 - accuracy: 0.9998
- val_loss: 0.0950 - val_accuracy: 0.9811
Epoch 32/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0021 - accuracy: 1.0000
- val_loss: 0.2069 - val_accuracy: 0.9800
Epoch 33/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0020 - accuracy: 0.9999
- val_loss: 0.1523 - val_accuracy: 0.9815
Epoch 34/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0014 - accuracy: 0.9999
- val_loss: 0.1363 - val_accuracy: 0.9789
Epoch 35/400
43/43 [==============================] - 12s 290ms/step - loss: 7.6161e-04 - accuracy:
1.0000 - val_loss: 0.1219 - val_accuracy: 0.9789
Epoch 36/400
43/43 [==============================] - 12s 290ms/step - loss: 0.0014 - accuracy: 0.9999
- val_loss: 0.0865 - val_accuracy: 0.9807
Epoch 37/400
43/43 [==============================] - 12s 289ms/step - loss: 8.3377e-04 - accuracy:
1.0000 - val_loss: 0.0850 - val_accuracy: 0.9796
Epoch 38/400
43/43 [==============================] - 12s 290ms/step - loss: 5.1040e-04 - accuracy:
1.0000 - val_loss: 0.0836 - val_accuracy: 0.9796
Epoch 39/400
43/43 [==============================] - 12s 289ms/step - loss: 5.7735e-04 - accuracy:
0.9999 - val_loss: 0.0805 - val_accuracy: 0.9841
Epoch 40/400
43/43 [==============================] - 12s 289ms/step - loss: 4.1574e-04 - accuracy:
1.0000 - val_loss: 0.0781 - val_accuracy: 0.9853
Epoch 41/400
43/43 [==============================] - 12s 290ms/step - loss: 5.1508e-04 - accuracy:
1.0000 - val_loss: 0.0822 - val_accuracy: 0.9819
Epoch 42/400
43/43 [==============================] - 12s 290ms/step - loss: 2.7000e-04 - accuracy:
1.0000 - val_loss: 0.0814 - val_accuracy: 0.9841
Epoch 43/400
43/43 [==============================] - 12s 290ms/step - loss: 5.1441e-04 - accuracy:
1.0000 - val_loss: 0.0843 - val_accuracy: 0.9811
Epoch 44/400
43/43 [==============================] - 12s 290ms/step - loss: 2.2395e-04 - accuracy:
1.0000 - val_loss: 0.0844 - val_accuracy: 0.9830
Epoch 45/400
43/43 [==============================] - 12s 290ms/step - loss: 1.9423e-04 - accuracy:
1.0000 - val_loss: 0.0881 - val_accuracy: 0.9811
Epoch 46/400
```

```
43/43 [==============================] - 12s 289ms/step - loss: 2.0816e-04 - accuracy:
1.0000 - val_loss: 0.0892 - val_accuracy: 0.9815
Epoch 47/400
43/43 [==============================] - 12s 289ms/step - loss: 2.1666e-04 - accuracy:
1.0000 - val_loss: 0.0892 - val_accuracy: 0.9830
Epoch 48/400
43/43 [==============================] - 12s 289ms/step - loss: 1.8244e-04 - accuracy:
1.0000 - val_loss: 0.0910 - val_accuracy: 0.9830
Epoch 49/400
43/43 [==============================] - 12s 290ms/step - loss: 1.5665e-04 - accuracy:
1.0000 - val_loss: 0.0908 - val_accuracy: 0.9811
Epoch 50/400
43/43 [==============================] - 12s 290ms/step - loss: 1.9227e-04 - accuracy:
1.0000 - val_loss: 0.0907 - val_accuracy: 0.9807
Epoch 51/400
43/43 [==============================] - 12s 289ms/step - loss: 3.1638e-04 - accuracy:
0.9999 - val_loss: 0.0940 - val_accuracy: 0.9777
Epoch 52/400
43/43 [==============================] - 12s 290ms/step - loss: 2.6351e-04 - accuracy:
1.0000 - val_loss: 0.0922 - val_accuracy: 0.9792
Epoch 53/400
43/43 [==============================] - 12s 289ms/step - loss: 2.2118e-04 - accuracy:
1.0000 - val_loss: 0.0972 - val_accuracy: 0.9792
Epoch 54/400
43/43 [==============================] - 12s 289ms/step - loss: 1.6434e-04 - accuracy:
1.0000 - val_loss: 0.0936 - val_accuracy: 0.9811
Epoch 55/400
43/43 [==============================] - 12s 289ms/step - loss: 1.3784e-04 - accuracy:
1.0000 - val_loss: 0.0950 - val_accuracy: 0.9807
Epoch 56/400
43/43 [==============================] - 13s 291ms/step - loss: 1.2349e-04 - accuracy:
1.0000 - val_loss: 0.0958 - val_accuracy: 0.9807
Epoch 57/400
43/43 [==============================] - 12s 289ms/step - loss: 1.2453e-04 - accuracy:
1.0000 - val_loss: 0.0976 - val_accuracy: 0.9826
Epoch 58/400
43/43 [==============================] - 12s 289ms/step - loss: 4.9190e-04 - accuracy:
1.0000 - val_loss: 0.0910 - val_accuracy: 0.9834
Epoch 59/400
43/43 [==============================] - 12s 290ms/step - loss: 1.4997e-04 - accuracy:
1.0000 - val_loss: 0.0913 - val_accuracy: 0.9830
Epoch 60/400
43/43 [==============================] - 12s 290ms/step - loss: 1.2631e-04 - accuracy:
1.0000 - val_loss: 0.0941 - val_accuracy: 0.9826
Epoch 61/400
43/43 [==============================] - 13s 294ms/step - loss: 1.1994e-04 - accuracy:
1.0000 - val_loss: 0.0938 - val_accuracy: 0.9826
Epoch 62/400
43/43 [==============================] - 12s 289ms/step - loss: 9.8613e-05 - accuracy:
1.0000 - val_loss: 0.0941 - val_accuracy: 0.9823
Epoch 63/400
43/43 [==============================] - 12s 289ms/step - loss: 8.9673e-05 - accuracy:
1.0000 - val_loss: 0.0938 - val_accuracy: 0.9830
Epoch 64/400
```

```
43/43 [==============================] - 12s 291ms/step - loss: 8.9802e-05 - accuracy:
1.0000 - val_loss: 0.0949 - val_accuracy: 0.9834
Epoch 65/400
43/43 [==============================] - 12s 291ms/step - loss: 1.0624e-04 - accuracy:
1.0000 - val_loss: 0.0958 - val_accuracy: 0.9830
Epoch 66/400
43/43 [==============================] - 12s 290ms/step - loss: 1.2228e-04 - accuracy:
1.0000 - val_loss: 0.0953 - val_accuracy: 0.9823
Epoch 67/400
43/43 [==============================] - 12s 289ms/step - loss: 8.3825e-05 - accuracy:
1.0000 - val_loss: 0.0960 - val_accuracy: 0.9819
Epoch 68/400
43/43 [==============================] - 12s 289ms/step - loss: 7.6459e-05 - accuracy:
1.0000 - val_loss: 0.0952 - val_accuracy: 0.9826
Epoch 69/400
43/43 [==============================] - 12s 291ms/step - loss: 1.5976e-04 - accuracy:
1.0000 - val_loss: 0.0983 - val_accuracy: 0.9819
Epoch 70/400
43/43 [==============================] - 12s 290ms/step - loss: 1.1244e-04 - accuracy:
1.0000 - val_loss: 0.0963 - val_accuracy: 0.9826
Epoch 71/400
43/43 [==============================] - 12s 289ms/step - loss: 9.7221e-05 - accuracy:
1.0000 - val_loss: 0.0967 - val_accuracy: 0.9830
Epoch 72/400
43/43 [==============================] - 12s 290ms/step - loss: 1.1543e-04 - accuracy:
1.0000 - val_loss: 0.0976 - val_accuracy: 0.9826
Epoch 73/400
43/43 [==============================] - 12s 290ms/step - loss: 6.5503e-05 - accuracy:
1.0000 - val_loss: 0.0973 - val_accuracy: 0.9841
Epoch 74/400
43/43 [==============================] - 12s 289ms/step - loss: 7.3900e-05 - accuracy:
1.0000 - val_loss: 0.0973 - val_accuracy: 0.9841
Epoch 75/400
43/43 [==============================] - 12s 290ms/step - loss: 6.5488e-05 - accuracy:
1.0000 - val_loss: 0.0978 - val_accuracy: 0.9841
Epoch 76/400
43/43 [==============================] - 12s 289ms/step - loss: 9.0457e-05 - accuracy:
1.0000 - val_loss: 0.0975 - val_accuracy: 0.9841
Epoch 77/400
43/43 [==============================] - 12s 290ms/step - loss: 7.1202e-05 - accuracy:
1.0000 - val_loss: 0.0976 - val_accuracy: 0.9841
Epoch 78/400
43/43 [==============================] - 12s 290ms/step - loss: 6.4039e-05 - accuracy:
1.0000 - val_loss: 0.0985 - val_accuracy: 0.9826
Epoch 79/400
43/43 [==============================] - 12s 290ms/step - loss: 6.0770e-05 - accuracy:
1.0000 - val_loss: 0.0984 - val_accuracy: 0.9823
Epoch 80/400
43/43 [==============================] - 12s 291ms/step - loss: 9.9124e-05 - accuracy:
1.0000 - val_loss: 0.0978 - val_accuracy: 0.9826
Epoch 81/400
43/43 [==============================] - 12s 289ms/step - loss: 4.6517e-05 - accuracy:
1.0000 - val_loss: 0.0969 - val_accuracy: 0.9834
Epoch 82/400
```

```
43/43 [==============================] - 12s 289ms/step - loss: 4.9163e-05 - accuracy:
1.0000 - val_loss: 0.0973 - val_accuracy: 0.9834
Epoch 83/400
43/43 [==============================] - 12s 289ms/step - loss: 5.6726e-05 - accuracy:
1.0000 - val_loss: 0.0976 - val_accuracy: 0.9826
Epoch 84/400
43/43 [==============================] - 12s 290ms/step - loss: 5.7549e-05 - accuracy:
1.0000 - val_loss: 0.0972 - val_accuracy: 0.9834
Epoch 85/400
43/43 [==============================] - 12s 290ms/step - loss: 5.9657e-05 - accuracy:
1.0000 - val_loss: 0.0959 - val_accuracy: 0.9841
Epoch 86/400
43/43 [==============================] - 12s 289ms/step - loss: 0.0026 - accuracy: 0.9990
- val_loss: 0.1082 - val_accuracy: 0.9785
Epoch 87/400
43/43 [==============================] - 12s 289ms/step - loss: 3.3738e-04 - accuracy:
1.0000 - val_loss: 0.0984 - val_accuracy: 0.9807
Epoch 88/400
43/43 [==============================] - 12s 290ms/step - loss: 1.6985e-04 - accuracy:
1.0000 - val_loss: 0.0985 - val_accuracy: 0.9811
Epoch 89/400
43/43 [==============================] - 13s 292ms/step - loss: 2.0327e-04 - accuracy:
1.0000 - val_loss: 0.0963 - val_accuracy: 0.9823
Epoch 90/400
43/43 [==============================] - 12s 290ms/step - loss: 1.5512e-04 - accuracy:
1.0000 - val_loss: 0.0935 - val_accuracy: 0.9834
Epoch 00090: early stopping
```