Applications of Neural Networks for Anomalous Energy Consumption Detection

B.E (Electrical), Batch 2016-17

Final Year Project Report

Prepared By:

Muhammad Waleed Hasan: EE-16177

Saad Mashkoor Siddiqui: EE-16163

Faiq Siddiqui: EE-16164

Syed Abdul Haseeb Qadri: EE-16194

INTERNAL ADVISOR
Dr. Mirza Muhammad Ali Baig
Assistant Professor
Department of Electrical Engineering
NEDUET

EXTERNAL ADVISOR
Mr. Shahzeb Anwar
Infrastructure Analyst
Eni Pakistan Ltd.

DEPARTMENT OF ELECTRICAL ENGINEERING

N.E.D UNIVERSITY OF ENGINEERING & TECHNOLOGY
KARACHI-75270

Acknowledgments

As electrical engineering undergraduates with little to no formal coursework in machine learning, artificial intelligence, or software development, this final year project was an intellectually rigorous, but rewarding endeavour. Through this project, the authors have witnessed themselves grow from fledgling machine learning students to aspiring machine learning practitioners with a sound understanding of the field's many tools, algorithms, and best practices.

This transformation would not have been possible without the unwavering support of the project's internal supervisor, Dr. M. M. Ali Baig. Dr. Baig was always available to provide insightful and incisive advice about both technical challenges as well as project management. He has inculcated within us the ability to think independently and critically about machine learning problems, for which we are exceedingly grateful.

The authors would also like to take this opportunity to thank the project's external supervisor, Mr. Shahzeb Anwar, for providing both access to and assistance with Amazon Web Services. With his support, we were able to explore the nuances of an industry standard platform for designing and deploying production level machine learning, which was an invaluable learning experience for us.

We are also thankful to Usama Altaf Zahid, EE-17024 from the Department of Electrical Engineering, NEDUET, for suggesting the use of sigmoid focal crossentropy as a loss function to address the class imbalance present in our dataset.

No words can express the debt of gratitude we owe to our families for their unconditional love, support, and encouragement. It is through their commitment to our education that we are able to

Finally, we are grateful to Allah Almighty, for blessing us with the health, fortitude, and intellectual vitality needed to see this project to its completion.

ABSTRACT

Electricity theft in electrical grids is a pressing issue for distribution companies (DISCOs), resulting in estimated losses of PKR 45 billion in Pakistan during 2019 [8]. Prevalent methods of electricity theft detection in Pakistan rely on manual energy meter inspections which are expensive, ineffective, and fail to scale with increasing number of consumers.

This project aims to address these issues by assessing the feasibility of artificial intelligence (AI) techniques for anomalous energy consumption detection. Specifically, anomalous energy consumption detection is framed as a supervised binary classification machine learning problem to which both shallow and deep learning approaches are applied.

This report establishes anomalous energy consumption detection as a complex technical problem of substantial socioeconomic importance along with the relevance and scope of the project detailed herein. After summarizing existing research on AI-based solutions for energy theft detection, the report outlines the methodology followed for the design and development of shallow and deep learning models for electricity theft detection. Data preprocessing of the SGCC dataset is discussed, followed by a description of the training and tuning of learning algorithms applied to the binary classification task in question: logistic regression, support vector machine, random forest, wide neural network, and convolutional neural network, and a wide and deep convolutional neural network.

The report also discusses various techniques that were investigated and applied to address the dataset's intrinsic class imbalance as well as the implications of the same for false positives and false negatives.

Test set performance of these learning algorithms is then compared and contrasted with each other and with benchmarks identified through literature review. The report concludes with a brief discussion areas and techniques that future researchers interested in the project may wish to explore.

Table of Contents

Table Of Figures

List of Tables

CHAPTER 1
INTRODUCTION

1. INTRODUCTION

1.1. Brief Introduction

Electricity has become a necessity of life and losses in electricity occur due to many reasons. These losses reduce the revenue of the power companies, previous studies shows that the losses due to electricity theft costs millions of dollars each year [2]. The abnormal pattern in energy consumption can be a good indicator of electricity theft. Nowadays, the abnormality in the energy consumption pattern have brought good attention to the data driven electricity theft detection approaches [3]. In this report we will discuss the abnormality found in the energy consumption of consumer's kWh data provided by state grid corporation of China (SGCC) [3] and classify thieves and normal consumer using machine learning tools and libraries.

1.2. Problem Statement

Theft of electricity is an alarming problem for any state. This crime can be committed either by tampering meter or by using electricity without being recorded by energy meters. Our aim in this regard is to use machine learning techniques to identify the anomalies in power consumption pattern of consumers and also to verify the results using different toolkits of machine learning (Keras, Scikit Learn) for this purpose. Moreover, to classify the users (from thief and regular consumers) we have built different machine learning architectures Random Forest, SVM (Support vector machine), logistic regression model and Wide Neural Network.

1.3. Objectives

1.3.1. To explore the feasibility of artificial intelligence-based methods for electricity theft detection.

One of the objective of this project is to explore if electricity thieves can be identified using Artificial Intelligence based methods, hence to carry out this investigation we will study previous theft detection techniques.

1.3.2. To identify the optimal machine and deep learning algorithm for AI-based electricity theft detection.

The second objective of this project is to devise most efficient AI-based method for electricity theft detection among the existing methods and assess their selectivity (Accuracy, speed of computation, cost). On the basis of these measures we will decide the best AI-based method for electricity theft detection.

1.4. Scope of work

1.4.1. Peer Review of an IEEE Journal Article

The project will verify the results of machine learning-based approaches to electricity theft detection presented in Zheng et. al's research paper. Having been cited by 25 other IEEE papers [3], Zheng et. al's research is a canonical work in the field of AI-based electricity theft detection. By systematically assessing and analyzing the assumptions, techniques, and results presented in the paper, the project will be an exercise in peer review which is a cornerstone of the academic research process.

1.4.2. Proof of concept of AI-based Electricity Theft Detection

The project will also serve as a proof-of-concept of deep learning-based solutions for residential electricity theft detection in smart grids. The project will establish that given a sufficiently large, labeled dataset of kWh values recorded over a time interval T, it is possible to train both shallow and deep learning models to identify potential electricity theft occurring within T with reasonable accuracy and reliability.

1.5. Significance of research

This project holds significance for two primary stakeholders in an electrical distribution system, namely consumers and distribution companies (DISCOs).

1.5.1. Consumers

Presently, DISCOs have a two-pronged approach to deterring electricity theft among residential consumers. On a micro level, consumers who are identified to have engaged in illegal abstraction of electrical energy have their connections terminated and are required to pay fines to the DISCO [17]. On a macro level, communities and locale s with substantial occurrences of electricity theft are more likely to face load shedding during periods of increased demand or shortfall [5] [6]. In the latter case, consumers who have paid their dues are also penalized along with thieves, thereby eliminating the incentive not to engage in electricity theft. On a second level, Jamil and Ahmad [7] have found that increasing electricity prices have a strong positive correlation with electricity theft, suggesting increase in electricity bills to compensate for non-technical losses in areas with widespread electricity theft tends to exacerbate, rather than mitigate, the issue. Blanket penalties for electricity theft detection are therefore both unnecessary and ineffective in preventing electricity theft.

The neural network presented in this paper is a first step towards incisive, accurate, and targeted electricity theft deterrence. It will empower DISCOs to remotely pinpoint specific consumers engaging in electricity theft without the need for manual, site inspection-based detection. This means DISCOs will be less likely to penalize the entire locale or neighborhoods' for the actions of the few consumers that are practicing electricity theft, thereby eliminating the idea of collective penalization of regular consumers.

1.5.2. DISCOs

A neural network capable of identifying potential electricity thieves using a window of kWh readings is of great operational and monetary value to DISCOs. The neural network developed in this project is intended to be an autonomous, intelligent, and scalable solution for electricity theft detection that can form the core of a smart grid's theft deterrence operations. Prevalent methods of electricity theft detection rely on manual energy meter inspections which are often inaccurate, time consuming, and scale poorly with growing number of electricity consumers often found in urban centers. In contrast, a neural network-based theft detection system will lead to more accurate, reliable, and actionable identification of instances of energy theft, and will minimize costs associated with manual energy meter inspections, improving operational efficiency.

From a monetary perspective, the primary benefit of a neural network-based electricity theft detection procedure will be the mitigation of the non-technical losses associated with electricity theft, which were recently estimated to be as much as PKR 45 billion in 2019 [8]. On a second level, the neural network is capable of adding value to energy consumption data that is already being logged and collected by DISCOs. The neural network(s) developed as part of this project can either be used as a transfer learning solution for tuning parameters of a similar, proprietary network developed by the DISCO, or can be trained on the DISCO's data from scratch.

In either case, the network effectively monetizes electrical consumption data by deriving actionable insights about potential electricity thieves which can then be used to both minimize instances of theft as well as decrease costs associated with identifying such occurrences through manual inspections. Furthermore, the network may be modified to be an online learning system, enabling it to continually learn from new instances of electricity thieves' consumption data identified during operation, and thus continually improve its performance, and improving return on investment after deployment.

CHAPTER 2
LITERATURE REVIEW

2. LITERATURE REVIEW

Electrical energy generation and distribution is a lucrative business that plays an integral part in building a country's economy. Electricity is now considered an essential part of our lives. With the demand for electrical energy increasing at a rate of 5.3% each year since World War II [4], it has become vital that the energy delivery network must be kept in proper working condition and any problems occurring in it must be addressed immediately.

In the energy sector the losses occurring in the generation, transmission and distribution is the foremost problem that degrades the efficiency of the system and causes utility companies 100 million Canadian dollars every year [2]. These losses fall into two main categories, namely Technical Losses (TLs) and Non-Technical Losses (NTLs). Technical losses are comparatively easier to resolve as the core cause and the location of the loss is known. This way they can be predicted and addressed by means of using better equipment. On the other hand NTLs are not only harder to solve but are nearly impossible to detect using manual inspections and other conventional methods.

Electricity theft is considered as one of the chief NTLs that plagues the distribution network in every energy delivery system. Especially in developing countries, the poor economic conditions encourages people to steal electricity causing huge revenue losses for the utility company and at the same time causing public safety hazard, like in case a fire or an electric shock occurs. Energy theft may occur primarily due to a number of reason which includes, physical tampering of the energy meter, sidestepping the energy meter (results in lower KWhs registered), false data dispatch from the host, malfunctions and misconfigurations in the communication scheme and improper synchronization of the meter after a scheduled maintenance job [2].

Now with the advent of smart girds and Advance Metering Infrastructure (AMI) the distribution sector is evolving. It is now possible to measure even very minute quantities of energy consumed by a customer and report them remotely using a smart meter. This data could serve as the basis for reducing NTLs, especially theft as the anomalous energy consumption trend can be reflected in the data transmitted by the smart meter. [2] highlights a technique that detects electricity theft based on data received by the energy meter. The 3 main features on which the mentioned system classifies a thief from a normal consumer are a) host based intrusion detection, that detects manipulating the internal software and hardware of the meter, b) on-meter anti tampering devices, and c) Non-intrusive load monitoring. The proposed model named AMIDs, integrated intrusion detection solution, is able to detect 62% of the cases as mentioned by the author.

In [10] devises a classification technique that uses Artificial Intelligence based methods to distinguish between thieves from non-thieves. An Artificial Neural network was proposed that learns on the basis of 7 different datasets and is able to recognize useful patterns leading to the final classification. The datasets include the socio-economic background of consumer, history of scrutiny, ownership exchange, debits and meter reading. These data bases are cleaned and fed into the neural network which then uses modern deep learning algorithms to map the input features to the desired output. The accuracy of the neural net achieved as quoted by the author is 87.17%, while the precision of classifying true energy thieves from normal consumer bumped up from 40% to 65.03%.

[11] proposes an innovative and fast hybrid algorithm named that comprises of Harmony Search in conjunction with a Optimum-path forest classifier to analyze patterns in consumption data to identify NTLs. Just like approaches discussed before the author of this article have showed it superiority over other methods for energy theft detection. The feature selection was performed on a dataset provided Brazilian electric power company with both commercial and industrial consumers. Accuracies of 92.60% were achieved using this method for the commercial consumers, while accuracy of 96.5% was published for industrial consumers. An advantage of this approach was that it was 9 times faster to train, in comparison to Particle Swarm optimization (PSO) algorithm, for a given dataset and identify appropriate features.

In [3] Zheng et al. propose a novel neural network architecture comprising wide and deep convolutional neural networks (WCNN and DCNN respectively). These CNNs are trained in tandem as a single neural network to identify electricity theft using a labeled, publicly available dataset from the State Grid Corporation of China (SGCC), which provides daily kWh consumption data of 42,372 residential consumers over 1,035 days.

2.1. Over View of “Wide and Deep Convolutional Neural Networks for Electricity-Theft Detection”

The authors first establish electricity theft detection as a practically and economically pertinent problem, citing BC Hydro Tech Inc. in stating that electricity theft losses cost ~CAD 100 million every year, and lead to severe electrical faults and hazards. The authors then proceed to review existing work on electricity theft detection, discussing both hardware and software-based anti-theft methods, the latter preferred by the authors for making use of data made available by the proliferation of Advanced Metering Infrastructure (AMI). Specifically, the authors have discussed the application of game theory, anomaly detection, and machine learning models such as support vector machines (SVMs) and neural networks (NNs) in the domain of electricity theft detection. In assessing these approaches, the authors found hardware-based methods to be expensive to operate, secure, and maintain. Machine learning approaches were analyzed to be more effective than other software alternatives. However, the authors note that these models either require artificial feature extraction or fail to take advantage of the inherent periodicity in the weekly kWh consumption patterns of regular consumers - a feature which the authors then proceed to establish to be absent in kWh consumption patterns of electricity thieves. Zheng et al.'s model aims to overcome this shortcoming by combining a WCNN and DCNN in a single neural network to simultaneously identify trends across all consumers on a single day and exploiting weekly periodicity in a single consumer's kWh consumption pattern. In this regard, Zheng et al. have provided a high-level overview of the WDCNN architecture along with analytical expressions for the custom convolutional kernels used in the DCNN component.

2.1.1. Data Preprocessing

The paper discusses how the SGCC data was preprocessed for use with machine learning models. Specifically how, Zheng et al. have replaced missing kWh values and scaled abnormal values present in the consumption distribution by applying a variant of the three-sigma rule and then finally mapping all kWh values to a [0, 1] range using min-max feature scaling.

2.1.2. Methodology and Results

Preprocessed data is then used to train various shallow learning models such as logistic regression (LR), SVMs, and random forest (RF) classifiers along with standalone WCNN and DCNNs for training ratios of 50%, 60%, 70, and 80%. These models are then evaluated on the basis of the Mean Average Precision (MAP) and the Area Under the receiver-operator characteristics Curve (AUC) over 100 and 200 test set samples. Hyperparameters for the final models are provided, although the methodology for deriving them has not been discussed. The WDCNN model is trained and evaluated in the same way, and is shown to outperform all other approaches by all metrics across all training ratios, reaching a peak MAP of 96.86%, as summarized in table 1 [3]. Hyperparameter tuning for the WDCNN model is then discussed at length, with trends in both MAP and AUC used to identify optimal values for WDCNN's layer counts, layer sizes, and epochs for all training ratios.

Table 1: Performance Comparison with Conventional Schemes

Training ratio = 80%		
Methods	**AUC**	**Optimal Hyperparameters**
LR	0.7060	C = inverse Regularization Strength =1.0, L2 Regularization
SVM	0.7413	Kernel = RBF, Penalty for Errors: 1.0

RF	0.7385	Trees= 200, Splitting Quality function = Gini impurity
Wide	0.6965	Neurons = 90, Epochs = 20, Relu Activation
CNN	0.7797	Various Combinations

2.1.3. Future Work

Zheng et al.'s results show that while a WDCNN does indeed outperform all other approaches, exploration of the DCNN is a cheaper, less computationally intensive solution which may be worthwhile, as it offers comparable performance as the WDCNN. The paper's methodology for identification of outliers is also worth re-evaluating, as the three sigma rule is more likely to identify outliers correctly when applied on a consumer, rather than daily, basis. Furthermore, despite identifying the kWh values as sequence data, Zheng et al. have not compared the performance of sequence models such as Recurrent Neural Networks (RNNs) with that of the WDCNN, which warrants further investigation.

2.2. Dimensionality Reduction

2.2.1. Dimensionality Reduction – Literature

Dimensionality reduction is a set of techniques designed to mitigate problems such as overfitting, computational intractability, and lack of visual interpretability in machine learning models. Geron [13] describes the curse of dimensionality as the phenomenon of a feature space becoming increasingly difficult to model as its dimensionality increases. This is because higher dimensional feature spaces are likely to be very sparse, with feature vectors of the same class often being very far away from each other.

It follows that as the dimensionality of a dataset increases, so too does the number of training examples required to populate dense regions of feature spaces corresponding to individual classes (in the context of a classification problem). Geron notes that it is often impractical to find datasets of the orders of magnitude required to create meaningful hyperplanes in high dimensional spaces.

Furthermore, the computational complexity of some machine learning algorithms scales poorly with an extremely large number of features. This can make such models both slow and expensive to train, validate, and use in production applications.

2.2.2. Principal Component Analysis (PCA)

Part of the solution to infamous dimensionality problems lies in the fact that data corresponding to real world problems aren't necessarily high dimensional and hence there exists a correlation among the features [13]. As a specific problem may generate only narrowly scoped data, like for our project KWh values at each day, so this stands to reason that every feature will have a correlation to every other feature. This insight gives birth to projection, which allows us to project our data to a lower dimensional hyperplane, with a penalty of some information loss. Like every other premise, there are exceptions to correlated feature assumption and for such cases the projection approach may not be a viable solution, which takes us to manifold learning explained in the upcoming section.

The goal is to choose the right hyperplane for our projections, so as to incur minimum information loss, hence only those axes chosen which account for the greatest amount of variance in the ascending order. These axes are defined by their unit vectors knows as Principal Components that are usually orthogonal to each other. Principal Component Analysis (PCA) works on the projection method thereby defining principal components for every dimension in the dataset and computing the variance as:

$$\frac{1}{N}\sum_{n=1}^{n}\{\mathrm{u}_1^T x_n - u_1^T \bar{x}\} = u_1^T S u_1$$

Where; u = unit vector for each dimension

X = sample set

Afterwards the training set can be projected onto the newly defined hyperplane by simply performing the matrix multiplication between the sample set and the principal components placed inside a matrix

$$X_{d-proj} = XW_d$$

Where, X is old sample set

X_d is the projected set

W_d is the principal components matrix

Linear PCA can be modified for identifying principal components using the kernel trick. Kernels are a set of algorithms that help define boundaries or learn patterns in the very high dimensional datasets. They make use of the kernel trick, which computes the inner products of the dimensions and mutually correlated features. This trick of identifying principal components is computationally cheaper. Once the components that define the hyperplane are identified, a linear PCA equation defined above is used for projecting down the samples on the lower dimensions.

2.2.3. Locally Linear Embeddings (LLE)

The principal component analysis is centered around the assumption of, there existing highly correlated features among the datasets. Likewise, manifold hypothesis holds that datasets with higher dimensions can be reduced to lower dimensional manifolds [13]. Depending on the dataset, there may be a complex decision boundary among the samples which when projected to lower dimensional manifolds result in complex or higher order decision boundaries.

This paves way for another dimensionality reduction technique known as Local Linear Embedding (LLE) that identifies correlation between a sample and its neighbors and chooses those dimensions that minimize the square distance between the manifold and the original dimension. Computed as

$$\hat{Z} = \underset{z}{argmin} \sum_{i=1}^{m} (z^{(i)} - \sum_{j=1}^{m} \widehat{w}_{i,j} z^{(j)})^2$$

2.3. Overfitting

Machine learning models are said to overfit their training data when the process of gradient descent begins to tune their learnable parameters to identify stochastic patterns or noise in the training data. A model that overfits the training data essentially memorizes patterns that map the training features to the training labels, instead of identifying meaningful patterns that will allow the model to generalize to unseen data. As such, an overfitted model will have low loss on the training set but a disproportionately higher loss on the validation set.

Tibshirani [18] has also defined overfitting as a problem of high variance. A model is said to have high variance when its predictions vary substantially based on the sample of population used for training. Ideally, a machine learning model should capture the underlying distribution of the population the training sample was drawn from. Causes of overfitting, as described by [24], [12], and [13] include

1. Poor quality data
2. Having too little data
3. Models with large memorization capacity or too many learnable parameters

2.4. Regularization

Regularization is an umbrella term that is used to describe techniques to mitigate overfitting in machine learning models. The ones that have been explored as part of this project are described below.

2.4.1. L1 and L2 Regularization

Also known as lasso and ridge regression respectively, these regularization techniques are designed to prevent model weights from becoming too large. Weights that are too large can indicate instability in a network, as even small changes in the features can lead to larger magnitudes of activation functions or hypotheses functions. These minute changes in input features are often attributable to noise or similar stochastic processes, which will thus be amplified by the model and skew its results. L1 and L2 regularization involve the addition of a coefficient term to the cost function for a given machine learning model. This constrains the optimization of weights via gradient descent to minimise loss while simultaneously placing an upper bound on the magnitude of learnable parameters.

In L1 regularization, a regularization factor λ is used to scale the Manhattan norm (L1 norm) i.e. the sum of all weights in the model, which tends to produce sparser weight matrices. L2 regularization scales the sum of squares of all weights (L2 norm) by the same regularization factor.

The higher the regularization factor, the greater the incentive for the gradient descent process to minimise the magnitudes of weights, and thus the stronger the regularization.

$$L1\ Regularization: J(\theta) = L(\theta) + \lambda \times \Sigma|W|$$

$$L2\ Regularization: J(\theta) = L(\theta) + \lambda \times \Sigma W^2$$

Here, $J(\theta)$is a cost function defined in terms of some parameter vector θ. $L(\theta)$ quantifies the cumulative loss on a set of training examples, while W is the weight matrix.

2.4.2. Dropout

A regularization technique devised specifically for neural networks [12], Dropout also aims to minimize spurious or irrelevant patterns from being propagated through deep learning models. Stochastic processes may cause activations of certain neurons in a neural network layer to be higher or lower than they should be given a set of input data. As these activations may then form the input for successive layers in computing their activations, the errors introduced by these patterns are propagated in the forward pass and thus affect the manner in which weights are tuned by backpropagation.

Dropout prevents this propagation of erroneous activations and the resulting erroneous weight updates by turning off a proportion of randomly chosen neurons in each layer during training. The activations of these neurons are thus zero for the rest of the forward pass through the neural network. This has a two-pronged effect on the ability of the neural network to generalize

1. The chances of erroneous activations being propagated through the network are reduced.
2. With some neurons in the previous layer(s) switched "off", neurons in the current layer are "incented" to identify combinations of the existing activations that are more strongly associated with the optimization metric.

While dropout causes neuron activations to become zero during training, it is important to note that this mechanism does not apply during validation or inference. All neurons in a neural network with dropout will compute their activations. However, neurons that had been turned off during training will naturally have smaller weight magnitudes, as their weights will not have been modified substantially during backpropagation. In this way, dropout can help mitigate overfitting in neural networks.

2.4.3. Batch Normalization

Batch normalization also has the effect of improving neural network performance for a given set of input features. As such, the authors have chosen to include a description of this technique in this section. **However, it is imperative to understand that batch normalization is not a regularization technique in the conventional sense**: it does not attempt to constrain the weights of a model in the same manner as L1 or L2 regularization or dropout.

Batch Normalization is best thought of as the concept of feature scaling applied to the inputs of each neural network layer. Normalizing or standardizing the inputs to a machine learning model is considered a good practice, as it prevents features with larger magnitudes from skewing the output of a hypothesis function relative to those with lower magnitudes.

If the conventional neural network is thought of as a series of stacked, multidimensional logistic regression classifiers [19], then the concept of batch normalization makes intuitive sense. Batch normalization scales the output or activations of every layer in the neural network such that activation magnitudes occupy roughly the same range of values before they are fed as inputs to the next layer. In doing so, batch normalization prevents successive activations from being skewed by activations with abnormally large magnitudes. For every batch of activations that is fed to a neural network layer, a batch mean and standard deviation are calculated and used to standardize the raw activation values as described below [20]:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \mathit{Batch\ mean}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \qquad \mathit{Batch\ variance}$$

$$\overline{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$E_x = \frac{1}{m}\sum_{i=1}^{j} \mu_B^{(i)} \qquad \mathit{Inference\ mean}$$

$$Var_x = \left(\frac{m}{m-1}\right)\frac{1}{m}\sum_{i=1}^{j} \sigma_B^{2(i)} \qquad \mathit{Inference\ variance}$$

$$y = \frac{\gamma}{\sqrt{Var_x + \epsilon}} x + \left(\beta + \frac{\gamma E_x}{\sqrt{Var_x + \epsilon}}\right) \qquad \mathit{Inference\ scaling/shifting}$$

$$y_i = \sqrt{x_i} + \beta$$

2.5. Class Imbalance

The ideal supervised binary classification problem will have a dataset that has the same number of training examples for both the positive and negative classes. This is not the case for many real world ML applications, where training examples of one class (often the target class) are few and far in between, whereas those of another class are abundant. The class with the fewer samples is called the minority class and the one whose samples are in abundance is called the majority class. Examples of such real world use cases include spam email classification, credit card fraud detection, and indeed the problem of electricity theft detection in the context of our FYP.

All of these use cases will inherently have a class imbalance: thieves are less common than regular consumers, spam email is less common than non-spam email, etc. We define the class imbalance ratio to be the ratio of the number of training examples of the minority class to the majority class. Ideally, it should be 1. The lower this ratio, the more imbalanced our dataset. In our dataset, we have a 91.5-8.5 split between examples of the negative class (regular consumers) and positive class (electricity thieves). The class imbalance ratio in this case is ~0.09, suggesting that the dataset is extremely skewed towards the majority class (regular consumers).

2.5.1. Effect of Class Imbalance

Class imbalance presents a problem for any ML model: it has fewer examples from which to learn the associations between a set of features and the target class. As a result, the model will often learn to classify examples of the majority class very well but be exceptionally poor in classifying examples of the minority class.

This can be problematic depending on the context of our ML problem: if all the model sees are regular consumers, it will classify many thieves as regular consumers too. It simply will not be able to differentiate between the two classes well enough. This increases False Positives (FPs) or False Negatives (FNs) in the confusion matrix of the model, and makes it unreliable.

2.5.2. Techniques for Addressing Class Imbalance

Several techniques have been developed to address class imbalances in statistical learning problems. A few of them are described below.

2.5.2.1. Weighted Loss

In a standard machine learning model, the cost function quantifies the “error” made by a model in terms of some loss metric that quantifies the difference between the ground truth and prediction for a given example. In a balanced dataset, a misclassification of examples of one class should increment the loss by the same amount as that of another class: it is equally “expensive” for the model to misclassify samples of both classes. The learning algorithm will thus tune its weights to ensure it learns to classify both classes, instead of favoring one over the other.

However, in imbalanced datasets, this approach does not work. Because the model assigns the same weight to misclassifications of the majority and minority classes, the models' weights are skewed by the majority class. This is because during training, the model sees more majority class samples than minority class samples and thus is incented to tune its parameters to accurately classify the majority class at the expense of the minority class. Weighted loss attempts to rectify this: because the few examples of the minority class are sparse, they are more valuable to the learning algorithm: it must learn a representation that allows them to be classified correctly. Thus the cost incurred by misclassifying a minority class sample is weighted to be substantially higher than that incurred by misclassifying a majority class sample.

In this way, the model is incented to learn representations that are more balanced: the fewer examples of the minority class are more expensive for the model to misclassify. Thus the model's weights are prevented from being tuned solely to classify the majority class and are instead balanced to classify both classes correctly → balanced model.

2.5.2.2. Undersampling

Undersampling the majority class is one of the methods to deal with imbalance classes of data by reducing the bias associated with it. This technique works best for large datasets. To undersample the majority class random undersampling can be used which randomly removes the data points from the majority class until a preset threshold is reached.

In randomly removing the data points for undersampling the user doesn't need to make any decision neither have to specify which data points are important or not. Previous studies show that the random removal of data for undersampling performs as well as where specific data points were removed deliberately from dataset. However, one of the disadvantages of the random undersampling is that important data points could also be removed. This problem usually arises when data is not smooth and has small features [21].

2.5.2.3. Oversampling

Oversampling the majority class is another way to deal with imbalances in the training set. This technique doesn’t work best with huge datasets as it involves adding/replicating the data points already present in the minority class to increase its weightage which can lead to a dataset that’s too large to use classifiers which have bad time complexity such as SVM. Although this technique is effective to tune the target class distribution but for a massively unbalanced dataset it may cause overfitting of minority class, this is one of the reasons why random oversampling gives worse performance on the test dataset but performs better on the training dataset. Random over sampler is used to randomly selecting a data point and creating its duplicates then repeating this process until a preset threshold value is reached [21].

2.5.2.4. SMOTE and ADASYN

Synthetic Minority Oversampling Technique (SMOTE) is a statistical technique developed specifically to address class imbalances in statistical learning problems. The technique generates new or synthetic samples of the minority class in the input feature space. Considering a sample x_i, a new sample x_{new} will be generated using its k nearest-neighbors. For instance, the 3 nearest-neighbors are included in the blue circle as illustrated in the figure below. Then, one of these nearest-neighbors x_{zi} is selected and a sample is generated as follows [22]:

$$x_{new} = x_i + \lambda \times (x_{zi} - x_i)$$

Where;

λ is the random number in the range of [0,1]. This interpolation will create a sample on the line between x_i and x_{zi} as illustrated in the image below:

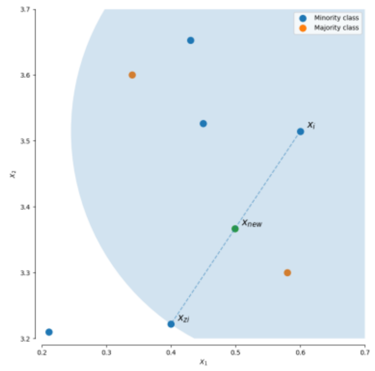


Figure 1: SMOTE

According to [22], SMOTE works best when combined with an undersampling of the majority class. Several variants of SMOTE have been developed. Chief amongst them is Borderline SMOTE. While the conventional SMOTE algorithm generates synthetic samples using minority class samples that may be present anywhere in the feature space, borderline SMOTE specifically chooses minority class samples that are present on a possible classification boundary between the minority and majority classes. The boundary itself is determined using the K-nearest neighbours algorithm. In doing so, borderline SMOTE can improve classification performance as it generate synthetic samples that are similar to those of the majority class, but actually belong to the minority class. It is posited that such synthetic generation helps classifiers learn better representations and distinguish between majority and minority classes with a higher degree of granularity.

2.5.2.5. ADASYN

ADASYN is a synthetic sample generation algorithm similar to SMOTE and its variants that is more selective about the region in the feature space where it creates new training examples. Specifically, the number of synthetic training examples of a class A generated by ADASYN in a particular region of the feature space (or neighborhood) is proportional to the number of examples from classes other than A. This “proportional” version of SMOTE thus has the potential to create training examples of the minority class in regions that would otherwise have been misclassified by a classification boundary. [23]

2.5.2.6. Focal Loss

Facebook AI [44]'s Focal Loss (also known as sigmoid focal crossentropy) is a loss function designed specifically to improve classification performance on highly imbalanced datasets, specifically in the context of computer vision. The canonical use case for the focal loss function is pixel-level segmentation masks of foreground objects relative to the background in dense backgrounds. This is analogous to a highly imbalanced classification problems as pixels of foreground objects are often few in number relative to those of the background, and thus constitute the minority class. All training examples considered difficult to classify or those that are incorrectly classified in a particular epoch of training are assigned higher weights in the next epoch, determined by focal loss-specific hyperparameters.

2.6. Convolutional Neural Network

2.6.1. Generalizability

A multilayer perceptron or feedforward neural network such as the WNN processes its input tensor as a 1-dimensional data structure. [3] and [14] posit that while such networks are certainly capable of learning specific interactions of features associated with the target class, they do not generalize well to inputs that have not occurred in the training data. As such, the WNN is limited in its ability to generalize to new sequences of kWh consumption since it does not transform this input further to extract higher level representations of features associated with electricity theft.

2.6.2. Overview – Convolutional Neural Networks

The Convolutional Neural Network (CNN) is a landmark neural network topology that was built specifically to address this limitation. One of the great successes of the deep learning revolution of the 2010s, the CNN was designed for hierarchical feature extraction from multidimensional input tensors such as images [45].

2.6.3. Convolution

CNNs iteratively parse 2-dimensional patches or windows of a potentially multichannel input tensor as standalone matrices. These matrices are then convolved with a convolutional kernel or filter, which is a matrix of learnable parameters within the CNN that attenuates or amplifies specific features within the input tensor. The output of the convolutional of an input tensor and convolutional kernel is a new tensor in which the features encoded by a filter are “activated” or amplified, whereas features not associated with the filter are attenuated. A single convolutional layer in a CNN is a collection of many such filters or kernels, each of which extracts a different feature from the same patch of the input tensor.

2.6.4. Stride

A CNN iteratively parses different, often overlapping, patches of the input tensor by sliding the convolution window from left to right and top to bottom within an 2D input tensor. The distance between the centers of adjacent convolutional windows in this operation is called the **stride**, and is an important hyperparameter of the CNN topology.

2.6.5. Padding

Convolutional windows are usually (1, 1), (3, 3), (5, 5) or (7, 7) dimensional. However, it is not always possible to extract convolutional window patches of elements from the input tensor, as their shapes can vary considerably. For instance, a (3, 3) convolutional window centered at the top-left pixel (5) will not have the elements required to convolve with filter elements in positions 1, 2, 3, 4, and 7 since there are no input tensor values associated with these positions in the original input tensor. However, if the positions with ? are padded with zeros, they can be convolved with the kernel without affecting the result of the convolution, as 0.X = 0 in the output activation map.

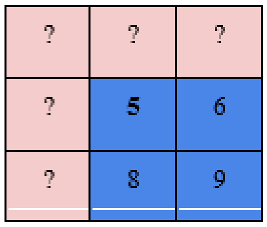
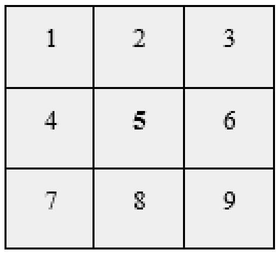
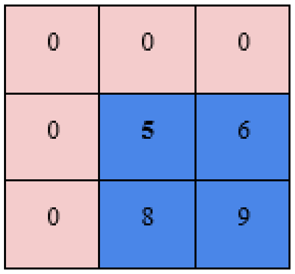


Figure 2: Input tensor patch in convolutional window



1	2	3
4	5	6
7	8	9

Figure 3: Convolutional Kernel



0	0	0
0	5	6
0	8	9

Figure 4: Zero padded input tensor patch

2.6.6. Continuous and Discrete Convolution

In the context of machine learning, the process of convolution is discretized as represented by the following equation. It is, in essence, a dot product between elements of the input tensor and the corresponding elements of the convolutional filter. Every element in the output activation map corresponds to the dot product of a convolutional window centered at the corresponding element of the input tensor and a convolutional filter. The dimension of the output tensor of a single convolution is shown below

$$output\ size\ = \left(\frac{n+2p-f}{s}+1\right)\times\left(\frac{n+2p-f}{s}+1\right). \quad [24]$$

$n \times n$ image

$f \times f$ filter

Padding p

Stride s

2.6.7. Pooling

The key to the generalizability of a CNN lies in its use of pooling operations and larger convolutional window sizes. Activation maps from the output of a convolutional layer are fed into a pooling layer that retains a specific type of activation amongst a collection of activations. With max pooling, only the numerically largest or strongest activation within a convolutional window is retained, while other convolutional values are discarded. This process of maximum pooling has two advantages

1. It identifies the "strongest" feature extracted by a convolutional over a patch of the input tensor.
2. It reduces the dimensionality of the input patch being processed without losing information about the most important feature within the patch.
3. Summary of pooling

   If input size is $n_H \times n_w \times n_c$

   Then output size will be [25]: $\left(\frac{n_H - f}{s} + 1\right) \times (\frac{n_w - f}{s} + 1) \times n_c$

   Where;

   $n_H$is the size of the height

   $n_W$is the size of the width

   $n_C$is the number of channels

2.6.8. Hierarchical Feature Extraction

Both of these aspects of the convolutional-pooling operation are associated with hierarchical feature learning, which is the essence of a CNN's generalizing power. By examining successively larger convolutional windows over input tensors that are made successively smaller with pooling, the CNN is able to extract hierarchical features without becoming computationally intractable. This means that shallower layers in a CNN may extract low level features such as edges or diagonal lines that are evident only when examining smaller patches of an input tensor such as an image. As the features are condensed into smaller patches by pooling and the convolutional window size increases, many such edges or diagonals are convolved with new filters to identify shapes such as triangles or squares. Many such shapes are then used to construct representations of textures, which in turn can be used to extract highly abstract features such as faces, humans, or vehicles.

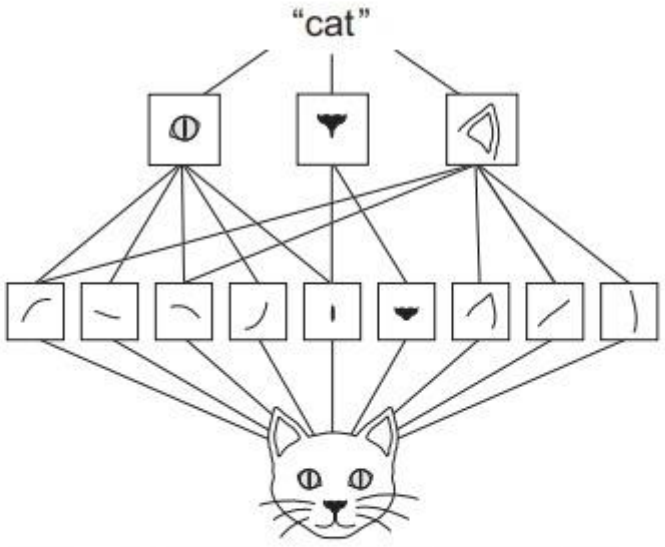


Figure 5: Hierarchical Feature Extraction

2.7. Wide and Deep Neural Network

2.7.1. Motivation

According to Cheng et. al [15], WNNs are capable of **memorizing** cross-feature interactions that are associated with successful identification of the target class, but do not generalize well to the feature spaces that they are used to process. Not only do they perform poorly when it comes to identifying cross-feature interactions not present in the training data, they may also fail to transform these interactions into more general, abstract features without many layers of transformations.

CNNs, on the other hand, excel at hierarchical feature extraction which, in turn, makes them capable of **generalizing** to the feature spaces formed by their inputs. However, CNNs are limited in their ability to memorize specific cross-feature interactions that may be associated with the target class.

	Good At	**Bad At**
WNN	Memorizing	Generalizing
CNN	Generalizing	Memorizing

Overview Table 2: Feature Extraction capability: CNN vs WNN

2.7.2.

Both Zheng [3] and Cheng [15] argue that WNNs and CNNs have complementary strengths and weaknesses, as summarized in the table above. They propose a hybrid neural network topology called the wide and deep convolutional neural network (henceforth abbreviated as WDNN) that combines the memorization capacity of a WNN with the generalizability of CNN. These sources claim that such a network is greater than the sum of its parts, and is capable of outperforming both standalone WNNs and CNNs on a host of binary classification problems, including that of electrical energy theft detection. The WDNN has been demonstrated to outperform its constituents on the canonical Machine Learning problems of wine quality prediction [26] and predicting housing prices in California [27], the latter of which is shown below.

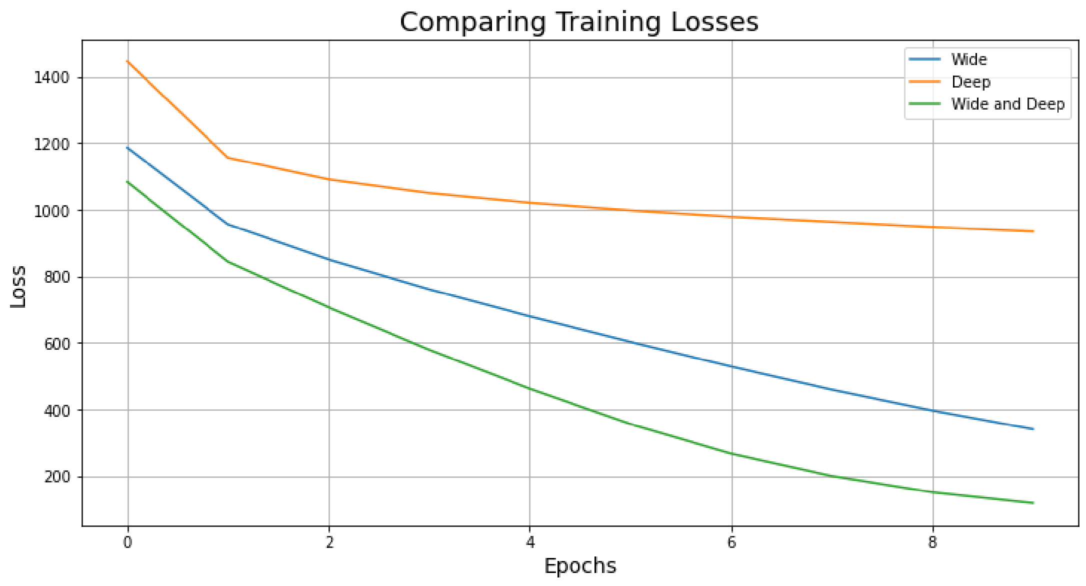


Figure 6: Loss comparison of WDNN and wide and deep networks

2.7.3. Topology

In a WDNN, the wide and deep convolutional components are trained in tandem and their respective outputs are added before being fed to a sigmoid classification unit, which outputs the final prediction of the network using features learnt from both the wide and deep networks. The weights assigned to the wide and deep outputs are learnable parameters that are tuned to minimize a loss function such as binary crossentropy via backpropagation. It is important to note that the WDNN is not an ensemble learning model. If this were the case, the wide and deep components would have been trained in isolation as standalone networks, with their predictions fed as inputs to a meta learning model such as a voting classifier [13].

Instead, two copies of the same input tensor are fed to the WDNN at the same time: a 1-D tensor to the wide component and the same 1-D tensor reshaped to a 2D tensor (or 3D tensor for multiple channels) for the convolutional component. The weights of both components are updated via the same backpropagated loss after each forward pass through the network.

In a way, the WDNN can be considered a multimodal input model as a 1D input tensor for the WNN component often represents a different modality of data than a 2D input tensor, which often encodes different data such as an image.

2.7.4. Hyperparameters

All the hyperparameters of the wide and deep convolutional neural networks are also relevant in the context of a WDNN, making this a difficult to tune owing to the high dimensionality of hyperparameter space to explore.

2.7.5. Combining Outputs from Individual networks

The method of combining the outputs of the wide and deep network is also a significant hyperparameter, as this, in part, determines the dimensionality of the final layers of the constituent networks. If the outputs of the WNN and CNN are simply to be concatenated, then the output dimensionality of either network can be arbitrary. However, using an arithmetic combination of the two outputs such as addition or subtraction will require both network outputs to have the same dimensions.

Based on literature review, concatenation seems to be a popular choice [28], [26]. However, Zheng [3] suggests using addition for the problem of electricity theft detection, possibly due to an assumption that the outputs of the two constituent networks represent the same quantities or features in the context of the classification problem. However, this assumption has yet to be substantiated in any of Zheng’s papers.

CHAPTER 3
METHODOLOGY/OBSERVATION & CALCULATION

3. METHODOLOGY/OBSERVATION AND CALCULATION

3.1. Data Preprocessing

Data preprocessing is an important step before training and testing data on the models. This technique is used to convert raw data into a more efficient and useful format. The data preprocessing steps for the dataset of SGCC are as follows.

1. Loading raw data
2. Checking for negative kWh values
3. Sorting the data
4. Replacing missing values
5. Identifying and replacing outliers
6. Feature scaling
7. Storing finalized data

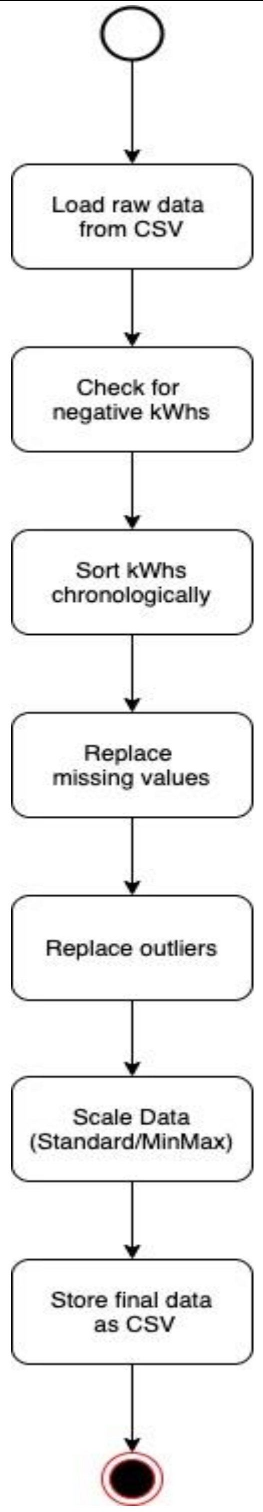


Figure 7:Data preprocessing Flow Chart

3.1.1. Loading Raw data from CSV

In the first step the data is read into pandas data frame to determine the number of columns and rows in the dataset, results show that the given dataset contains 42,372 rows and 1,036 columns, the first two columns consist of flag and consumer number remaining 1,034 columns contains the kilo watt hour reading of consumers for 1034 days.

3.1.2. Checking negative values

Since the energy consumption of a consumer can’t be negative therefore we check for negative values in kWh programmatically. The dataset consists of large number of kWhs, checking negative values by inspection of head is intractable. After checking the results show that there are no negative kWh values in the entire dataset.

3.1.3. Sorting data

The dates in the header of each column containing daily KWh readings are not in chronological order. Without sorting the data in chronological order the actual consumption pattern of a consumer with time can’t be determined and even the normal consumption pattern can be identified as abnormal, therefore it is necessary to sort data in chronological order. For this purpose following steps are taken for sorting the data in chronological order.

- The columns containing kWh values are separated from the column containing FLAGs and Consumer numbers.
- The columns in the kWh dataframe are converted from string to datetime objects.
- After converting to datetime objects the kWhs dataframe is sorted in chronological order.
- After sorting data in chronological order the sorted kWhs dataframe is joined back with the columns containing consumer numbers and FLAGs.

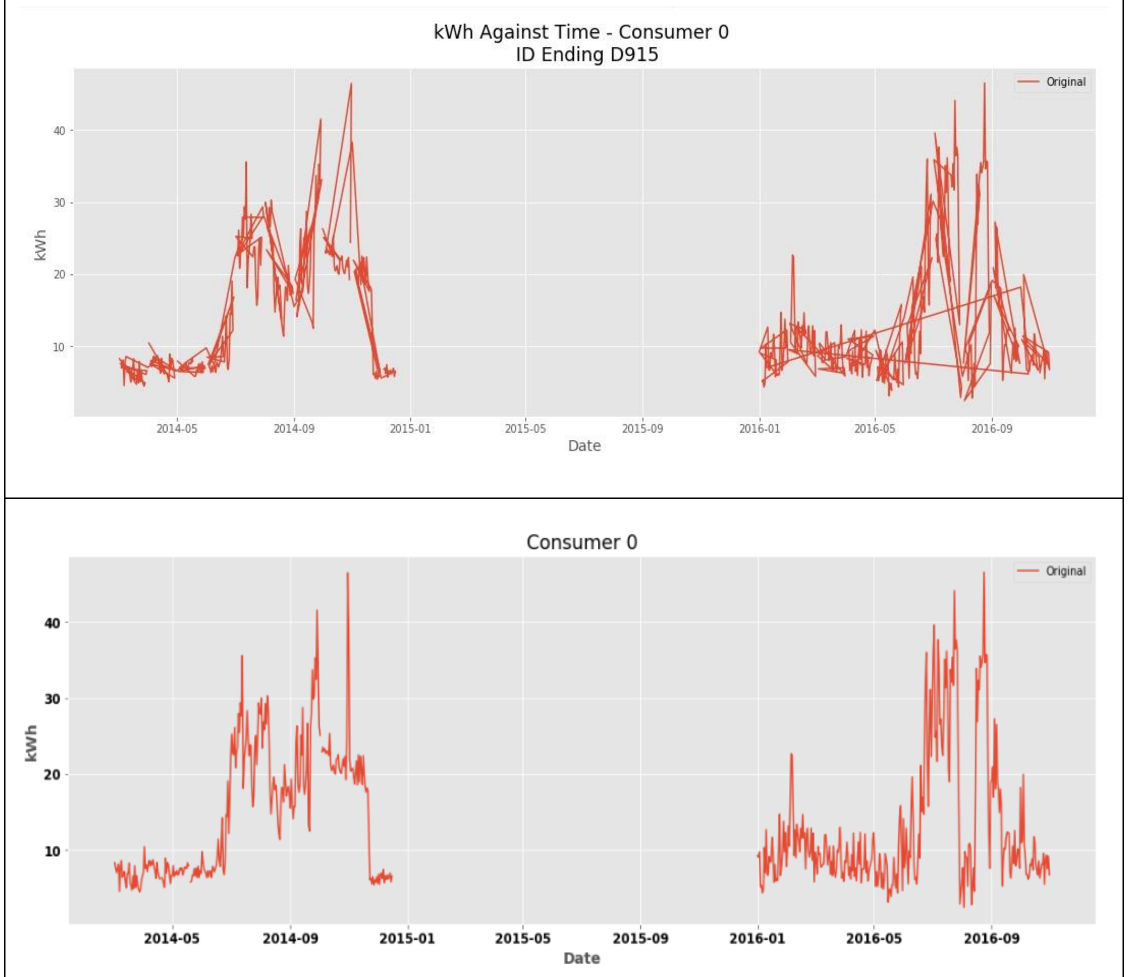


Figure 8:kWh Vs time before sorting (upper) and after sorting (lower)

3.1.4. Visualizing missing values

Since there are more than 1,000 columns in the dataset, using head to visualize missing values in each column is intractable. Instead of creating a new dataframe of non-null values in each column of the dataset, we have plotted it to show the trend in the number of missing values in the dataset. The figure given below shows the number of non-null values per day.

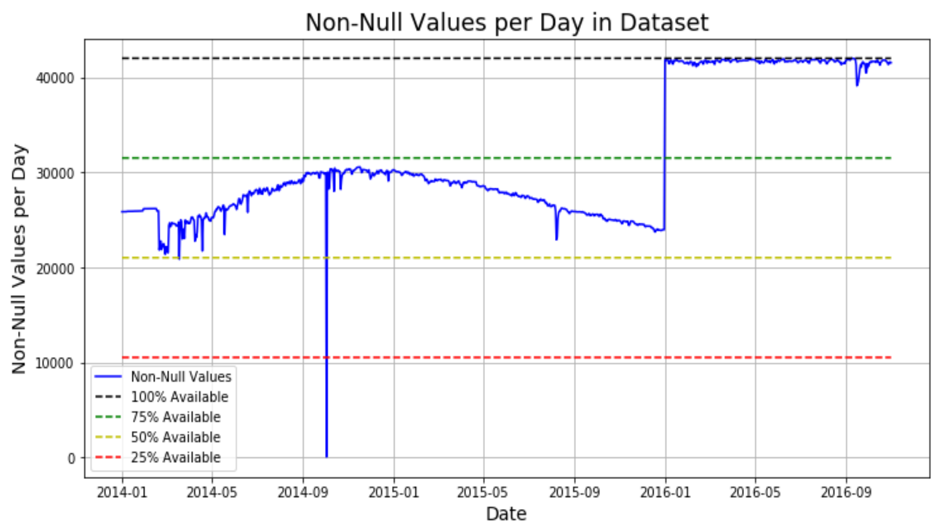


Figure 9: Non-Null/Non-missing values per day

From the figure shown it is clear that there is one day when all the kWh values are missing because number of non-null values dropped to zero. On most of the days, the number of non-null values lies in between 50% and 75% of the total number of consumers in the dataset and there is no missing values in the dataset after towards the end of the dataset i.e during the year 2016.

3.1.5. Replacement of missing values

The missing values are treated as NaN (Not a Number), to replace these values with a numeric value following techniques have been used in [3].

- If the value preceding the missing value and next to missing value are non-missing/non-null then it is replaced by the average of the KWhs consumed on the next and previous days [3].
  If $x_{i-1}, x_{i+1} \notin$ NaN, then

$$x_i = \frac{x_{i-1} + x_{i+1}}{2} \qquad Equation\ (3.1)$$

- If either of the next or previous days' kWhs are undefined, the current day’s kWhs are assumed to be 0. Since the kWh value before first day and after the last day remains unidentified so the missing values in the first and last columns are replaced by 0 [3].
  if $x_{i-1}, x_{i+1} \in$ NaN, then

$$x_i = 0$$

  Where;
  - x_{i-1} is the kWh value on the day before the missing value’s day.
  - x_{i+1} is the kWh value on the day after missing value’s day.
  - x_i is the day on which missing value exists

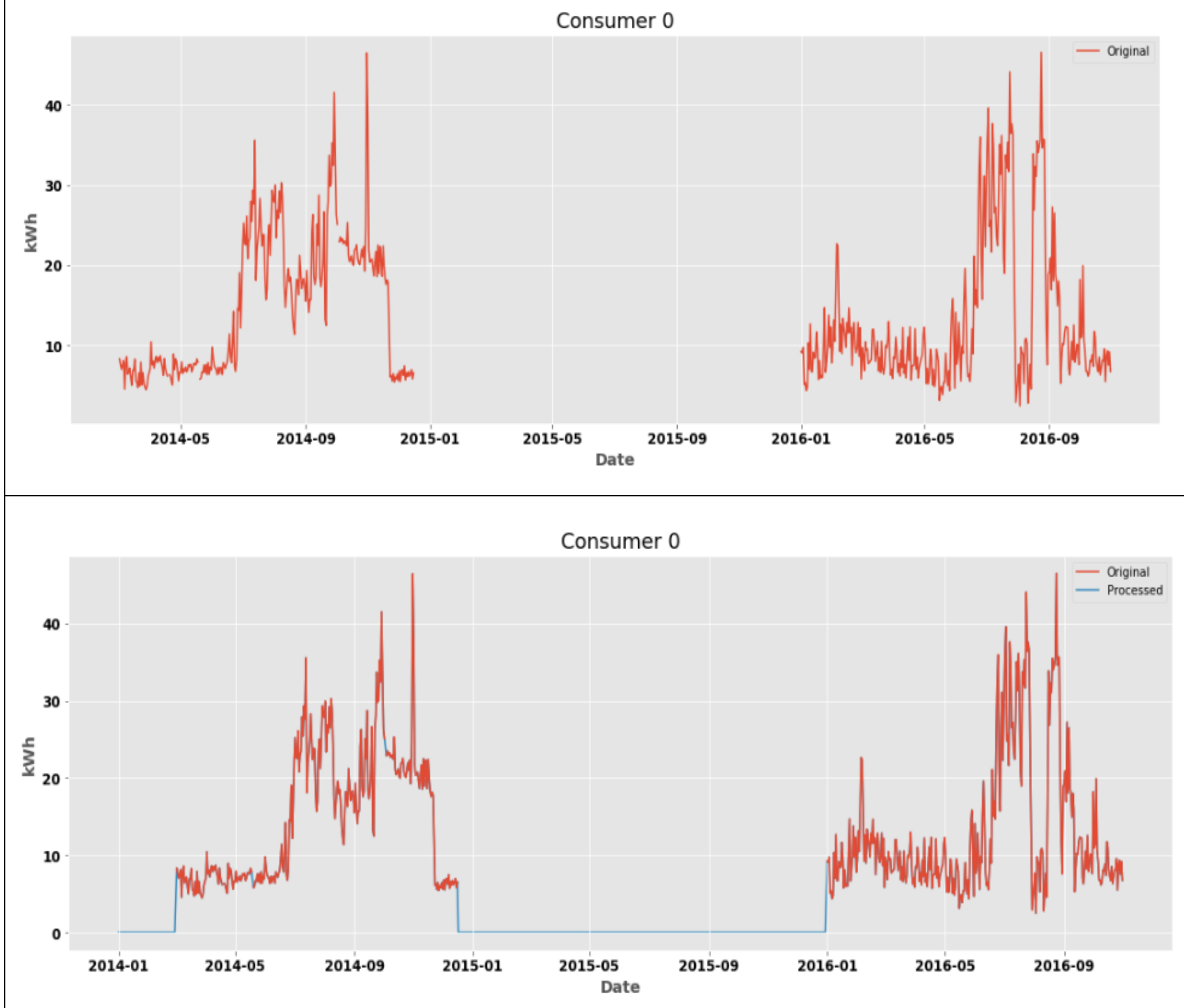


Figure 10:kWh Vs time before replacement (upper) and after replacement (lower) of NaN

3.1.6. Identification of outliers

Outliers are erroneous values that differs from other values in the data significantly. According to three sigma rule of thumb [16], any kWh value which is more than sum of average value and twice the standard deviation is considered as an outlier. Only positive deviations from the mean should be considered while identifying outliers because electricity consumption of each user is always greater than 0 after analyzing the electricity consumption data of 1,034 days.

Concretely, the kWh consumption on the i^{th} day x_i is considered to be an outlier if

$$x_i > avg(X) + 2 \times std(X) \qquad Equation\ (3.2)$$

where X is the vector containing all kWh consumption values of a single consumer.

3.1.7. Replacement of outlier with threshold values

The outliers have been replaced with the threshold values that is sum of average kWh value and twice the standard deviation [3].

$$f(x_i) = \begin{cases} avg(x) + 2std(x) & if\ x_i > avg(x) + 2std(x), \\ x_i & otherwise, \end{cases} \qquad Equation\ (3.3)$$

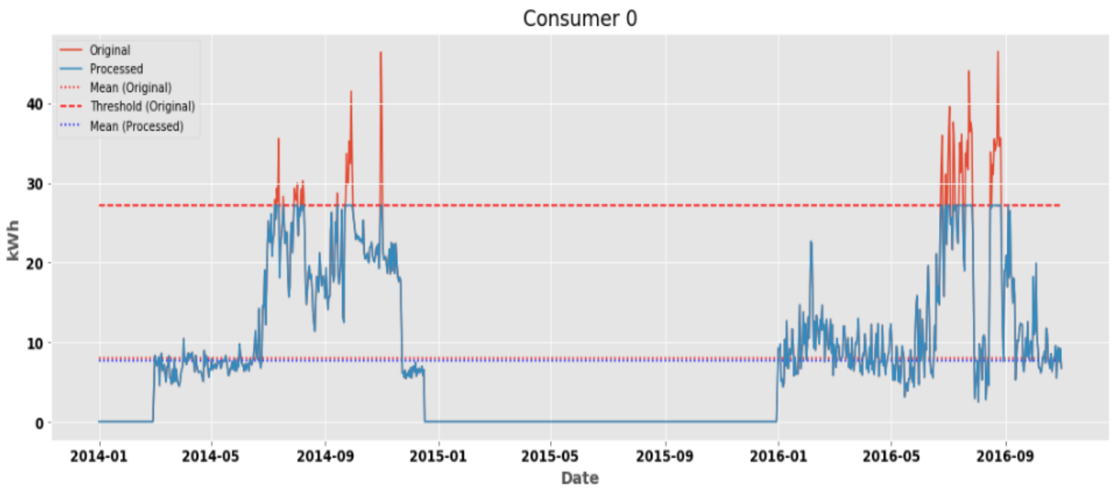


Figure 11:Consumption pattern of consumer number 0 after replacement of outliers

3.1.8. Feature Scaling

a. <u>MinMax Scaling</u>

`MinMaxScaler` is a built-in function which transforms each value in the column proportionally within the range [0,1]. It has been used to normalize the features for an individual consumer, concretely,

$$f(x_i) = \frac{x_i - \min(x)}{\max(x) - \min(x)} \qquad \textit{Equation}\ (3.4)$$

- x_i is the kWh consumption of a single consumer on the i^{th} day.
- x is a vector x_i day-by-day
- $\min(x)$ and $\max(x)$ are the minimum and maximum values of x_i for that consumer

`MinMaxScaler` scales along the column axis i.e. it will find the minimum and maximum values in a single column and use them for scaling. Since we want to scale according to min and max values amongst all kWhs of a single consumer (row axis) and not the min/max values amongst all consumers on a single day (column axis), therefore the data is first transposed before scaling.

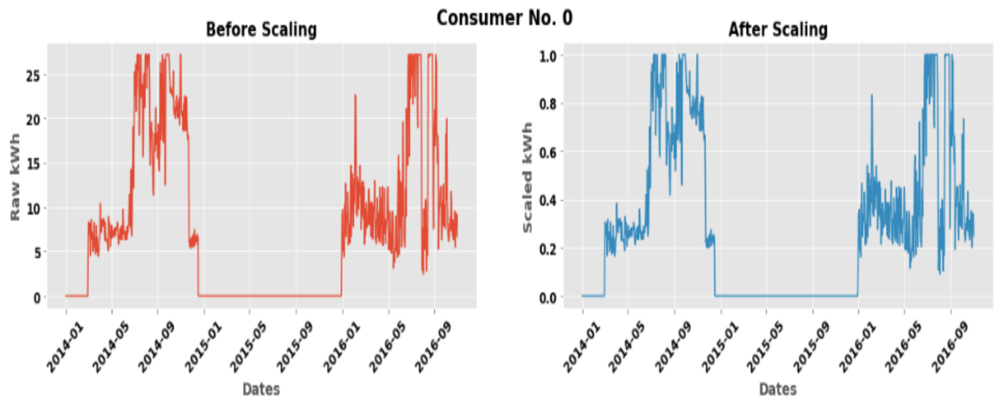


Figure 12: Consumer number 0 before and after minmax scaling

b. Standard Scaler

Standard scaler is another built-in function used for feature scaling. It removes the mean and scales the data to unit variance. The standard score of a sample x_i can be calculated as:

$$f(x_i) = \frac{x_i - mean(x)}{std(x)} \qquad Equation\ (3.5)$$

Where

- x_i is the kWh consumption of a single consumer on the i^{th} day.
- x is a vector x_i day-by-day
- $mean(x)$ is the mean of the training samples.
- $std(x)$ is the standard deviation.

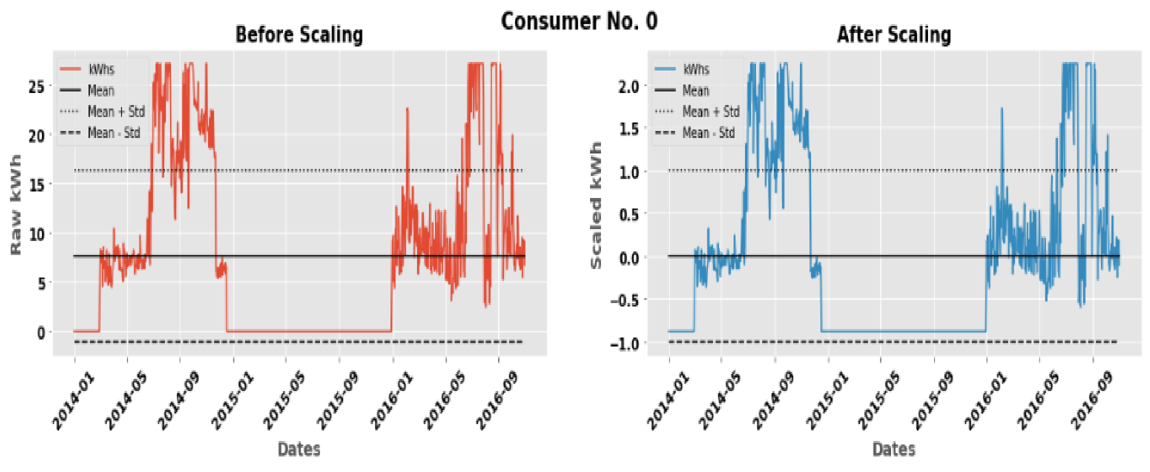


Figure 13:Consumer number zero before and after standard scaling

3.2. Shallow Learning Models

3.2.1. Logistic Regression (LR)

It is the most basic and effective learning algorithm for binary classification problems which has been used in this project for the classification of thieves and normal consumers of electricity by analyzing the preprocessed daily kWh consumption data. Python-based Deep Learning Libraries Scikit learn and Keras are used to develop, train and test this model and assess its selectivity (True positive rate). The steps involved in carrying out Logistic regression techniques are described below.

a. Train test Split

The preprocessed data produced by standard scaler is imported from the drive and split into training and test samples with the training ratio of 80% since LR gives best results at 80% training ratio in previous studies [3]. The proportion of electricity thieves and normal consumers is same in both training and test dataset for stratified sampling in train-test data split. The graph shown below confirms the proportion of thieves and non-thieves in the train, test and overall data.

3.2.2. Support Vector Machines (SVM)

It is another shallow learning model used for discriminative classification and regression problems. To train this model we have used the data produced by standard scaler at the training ratio of 20%, 60% and 80%. The computational speed of this model is the slowest among the tested models. At the training ratio of 80% the model took ~43 minutes to train. Following are the hyperparameters used for this model.

- Penalty parameter of error term is set to 1.0.
- Kernel: `rbf` – Radial Basis Function.

3.2.3. Random Forest (RF)

The third shallow learning model used for electricity theft detection problem. The data produced by standard scaler has been used to train this model with the training ratio of 80%. This model is expected to outperform SVM and Logistic Regression, specified hyperparameters are,

- Number of trees: 200
- `gini` Function is used to measure quality of split

3.3. Dimensionality Reduction

As will be discussed in (section 4.2) The support vector machine (SVM) classifier demonstrated promising classification performance in terms of ROC AUC. However, it was considered unsuitable for runtime evaluation or inference because it took exceedingly long to both train and generate predictions on the test set. This was attributed to the SVM algorithm’s time complexity: it scales quadratically or cubically with the number of features in the dataset. The authors wished to explore solutions for minimising training and prediction time of the SVM algorithm without substantially compromising its ROC AUC performance. To this end, a host of dimensionality reduction techniques were applied to the training data.

Before figuring out the principal components that conveyed the dimensions of newly formed hyperplane, the data had to be scaled or normalized. Hence, after the dataset was loaded, the features were extracted and the data was standardly scaled using the ScikitLearn preprocessing library. For implementing the PCA we again made use Scikit Learn whose decomposition feature allowed for easy Principal components extraction. While extracting components those dimensions of hyperplane were chosen which could preserve 95% variance of the original dataset. After projection of the features on the newly defined hyperplane the 1034 features in the original dataset were to reduce to almost half, at about 648 features per sample. Upon further analysis the feature value that exited at 924th place, in the original data, attributed to about 62% of the variance of the entire dataset. Cumulative Sum plots generated also indicated the same.

With the reduced data, our first goal was to train on shallow learning classifier like SVM with higher time complexity to experience and improvement in the training job. To our surprise the same classifier with the RBF kernel that took 38 mins to train on the original standard scaled data now took 45 minutes with the reduced data. Performance wise the AUC on the test data reduced to 0.75 from 0.77 obtained in the previous results. This lead us to believe that component analysis wasn't fit for our data and there may exist little to no correlation among the features, or they may be so spread out dimensionality wise, that their projections on the reduced hyperplane incurred huge information loss, hence the poor performance of the SVM on the reduced data. Manifold learning also lead to the same problems, as it could not classify feature with distinct boundaries and unwrap the manifold into reduced dimensions. All in all this route of reducing training times lead to no fruitful results.

3.4. Deep Learning Models

3.4.1. Wide Neural Network

Wide Neural network is a deep learning model, the model has been used for theft detection with the training ratio of 80%. Python-based Deep Learning Libraries scikit learn and keras are used to develop, train and test this model and assess its selectivity on the data produced by `standardscaler, minmaxscaler` and `maxabs`.

- **Number of Neurons**: In [3] the optimal number of neurons is 50. However, the highest AUC is obtained when the number of neurons is closer to 90.
- **Activation functions**: `Relu` in the first layer and `sigmoid` in the output layer

3.4.2. Convolutional Neural Network

For this project, we have used the Keras Sequential API to implement a convolutional neural network (CNN) based on the design specified by Zheng at al [3]. As discussed earlier, the CNN processes 2-D tensors of weekly kWh data rather than 1-D tensors of daily kWh data.

Therefore, in addition to stratification and feature scaling techniques applied to inputs of all other models, the following preprocessing steps further modify the kWh data for use with the CNN

1. Zero Padding
2. Reshaping 1D → 2D

3.4.2.1. Zero Padding

Daily kWh data for 1,034 days represents ~147.7 weeks of data. Since the CNN can only process a whole number of weeks, the 1,034-dimensional kWh vector for each consumer has been appended with two zeroes to bring the dimensionality of a single kWh vector to 1036, which is exactly equal to 148 weeks.

Zero padding is consistent with the kWh imputing technique used earlier in the data processing. Based on Zheng's recommendation, if the kWh data for the day before or after a particular day is unavailable or unknown, then the kWh data for that day is assumed to be 0.0 kWh. Since the data for days 1035 and 1036 is unknown, it can be safely imputed as 0.0 kWh for each consumer for greater consistency with the rest of the missing values in the data.

A custom ZeroPadder class was written for this purpose which extends the functionality of the ScikitLearn Transformer API. This allows the zero padding operation to be performed as part of a Scikit-Learn Data Pipeline.

3.4.2.2. Reshaping

Once the 1D kWh data has been appended with the requisite number of zeroes, it needs to be reshaped into a 2D tensor. This is again achieved with a customer Transformer class called Reshaper_2D that uses NumPy's highly optimized, massively parallelizable reshaping function to cast a (1, 1034) dimensional kWh vector into a (148, 7, 1)-dimensional tensor. The third axis of this tensor represents the channels axis which is necessary for a Keras Convolutional layer to process the tensor. For all intents and purposes, the data for a single consumer can be treated as a (148, 7) matrix. However, had the data also included similar timeseries information about other electrical power system quantities such as power factor, line voltage, and line current, the channels axis could be extended to accommodate them.

3.4.2.3. Zheng's Model

The first step in the investigation was to verify Zheng's models' performance. The models' topology is shown in the figure below. This model has 2.19M trainable parameters, several successive convolutional layers with the same, a single MaxPooling layer, and a densely connected classifier. The only difference in the implementation is the convolutional kernels. Zheng et al have engineered hardcoded convolutional kernel functions that look at (3, 7) windows of the input tensor at any given time.

Given that DL is more than capable of learning its own kernel functions to minimise loss, the authors felt it more prudent not to implement these custom convolutional kernels.

Where possible, we have used the optimal hyperparameters defined by Zheng et al for this topology. A list of these hyperparameters is as follows:

Hyperparameter	**Value**
Number of Convolutional Layers	5
Convolutional Filters per Layer	18 → 288

Convolutional Layer Activation Function	ReLU
Convolutional Window Size	(3, 3). Zheng assumed this to be (3, 7)
Pooling Window Size	Undefined. Assumed to be (5, 5)
Dense Layer Activation Units	60
Dense Layer Activation Function	ReLU
Padding	Undefined. Assumed to be same to maintain input tensor size.

Table 3 Optimal Parameters by Zheng

The densely connected classifier fed into single sigmoid activation unit. The loss function for this network was binary crossentropy. The neural network was trained on both standardized and min-max normalized stratified training data after an 80-20 split between training and test data. The ROC AUC and loss on both training and validation data were examined to identify any overfitting that may have been occurring during training. The network was then evaluated on the test set to achieve a benchmark against which future model design work could be compared.

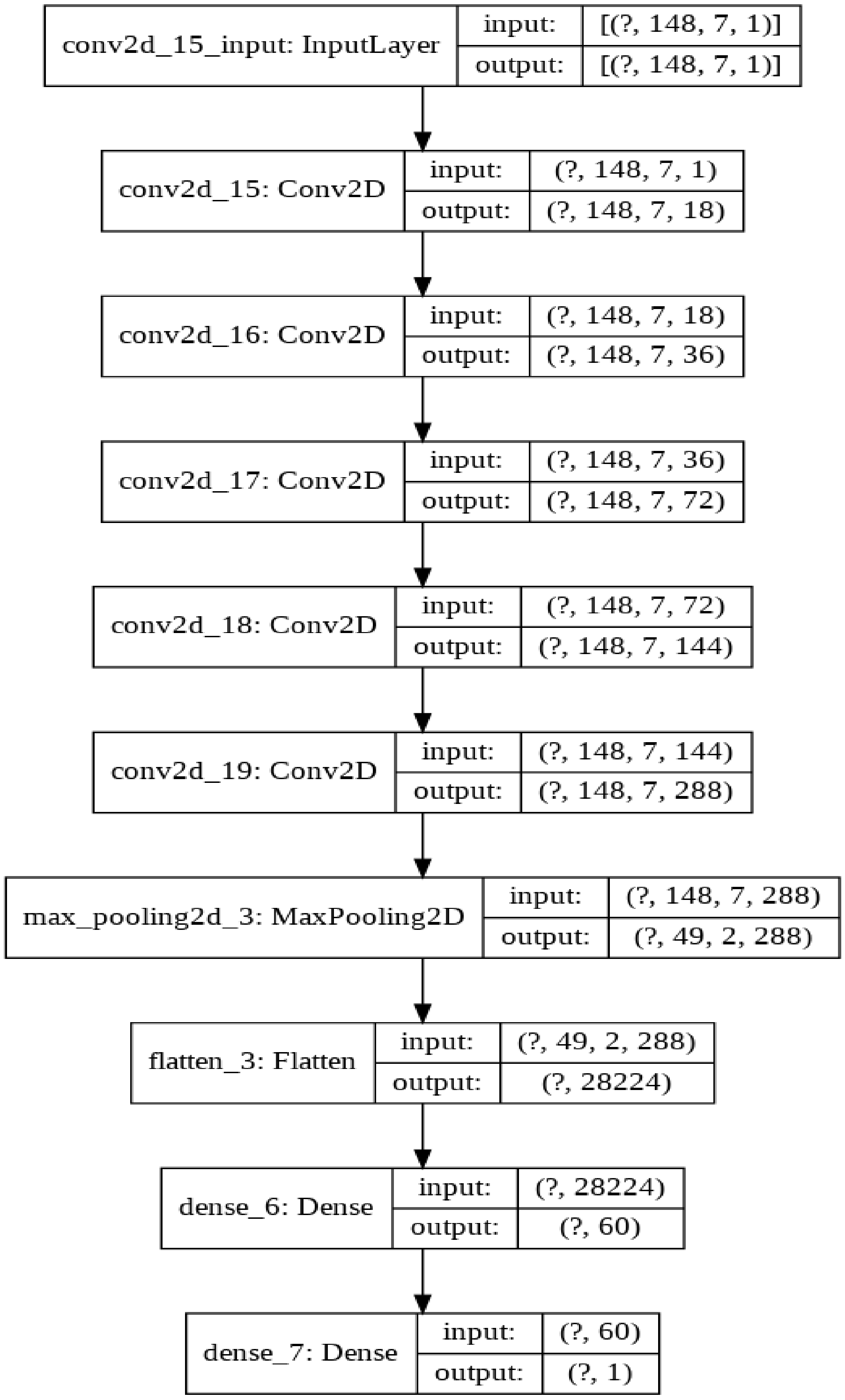


Figure 14 CNN Model Structure

3.4.3. Wide and Deep Neural Network (WDNN)

3.4.3.1. Keras Functional API

As discussed in section (reference to section where WDNN theory was discussed), the WDNN is a multimodal input model. In Keras, such models cannot be constructed with the Sequential API, which treats a neural network as a sequence or stack of successive transformations that modify the outputs of previous layers. The WDNN was therefore constructed using the Keras functional API, which creates a neural network as an acyclic graph of tensor transformations, where each layer is a standalone transformation function whose input and output is a tensor. In this way, a single input tensor can be routed through multiple transformations within the same network

.

3.4.3.2. WDNN Model Structure

Figure 13 shows the structure of the WDNN as described by Zheng et. al [3]. For each training example, the same 1034-dimensional kWh vector is used as inputs to layers of the wide and deep components. In case of the latter, the 1034-dimensional vector is zero-padded to be a 1036-dimensional vector, which is then reshaped into a (148, 7, 1) dimensional tensor. This tensor represents a single consumers' kWh readings on a weekly basis. Since we are only providing kWh readings for each consumer, the channels axis is set to 1 for compatibility with Keras Conv2D layers, which expect a channels axis by default.

The same kWh data is thus propagated along the network in two different shapes in the forward pass, and the loss incurred by the classification of this example is backpropagated to update weights of both the wide and deep components in tandem.

The outputs of the wide and deep components are then added before being fed to a densely connected classifier that combines the weighted sum of the 60-dimensional outputs from the constituent networks into an input for a sigmoid activation function.

The final output of the network is thus a probability of a particular consumer being an electricity thief.

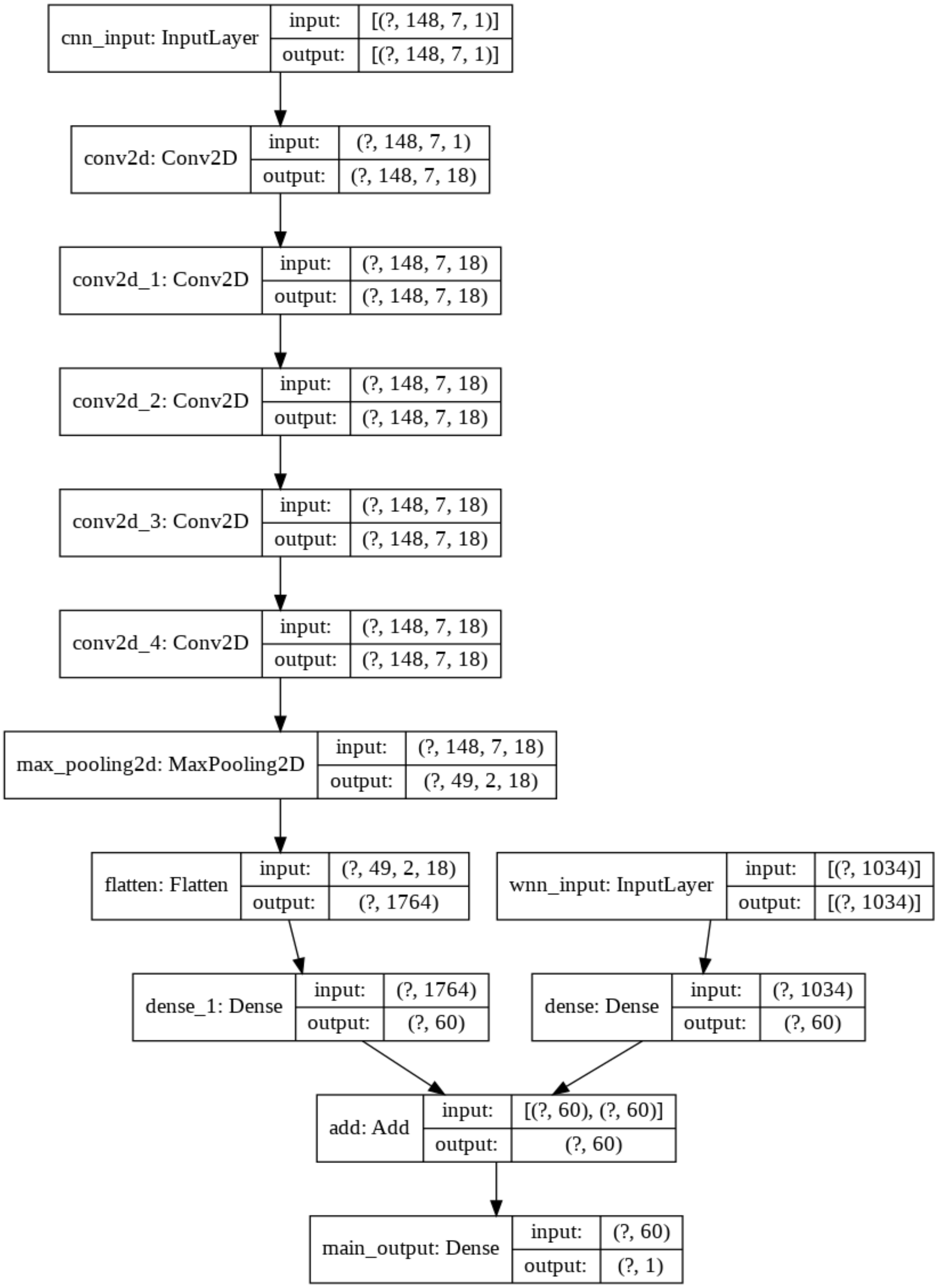


Figure 15: WDNN Model Structure

3.4.3.3. Hyperparameter Tuning

Keras models built using the Sequential API can be cast to Scikit-Learn estimators using a KerasWrapper object. This allows hyperparameter tuning of these models using the Scikit-Learn GridSearchCV API, which automatically explores a hyperparameter space for the optimal values of hyperparameters for a given model. Since the WDNN was built with the Functional API, it was incompatible with the hyperparameter tuning workflow described above. To accommodate this, the authors developed their own hyperparameter tuning module. The module used a function to instantiate a WDNN with the functional API and custom hyperparameters. This build function was used with nested list comprehensions to instantiate models with all possible combinations of hyperparameters possible in the hyperparameter space defined by the individual hyperparameter lists.

A for loop was then used to iterate over the list of models and save their training and validation history objects into a single dictionary of training histories. Custom modules were then used to extract and plot variations in the models' training and validation losses and AUCs for all combinations of hyperparameters explored. By automating and modularizing the process of training and evaluating models, the following hyperparameters were explored systematically.

3.4.3.4. Activation – CNN

To select the activation function, the model's performance was evaluated at different activation functions. From the trends shown in the figure given below it's clear that the AUC is higher (0.78) for SeLU as compared to ReLU and tanh. Therefore SeLU was selected as an activation function of CNN.

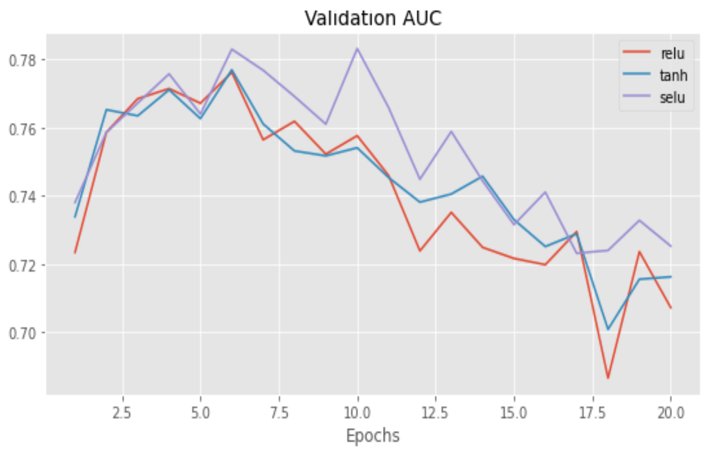


Figure 16: AUC by tuning CNN Activation

3.4.3.5. Filters

An optimal value for the filters were also selected via hyperparameter tuning, the model's performance was evaluated on certain hyperparameter values such as 8, 12, 16 and 20. From the trends shown in figure given below the AUC value for the 8 filters were higher for the most of the iterations as compared to other filter values. Therefore the 8 filters were selected for the CNN classifier.

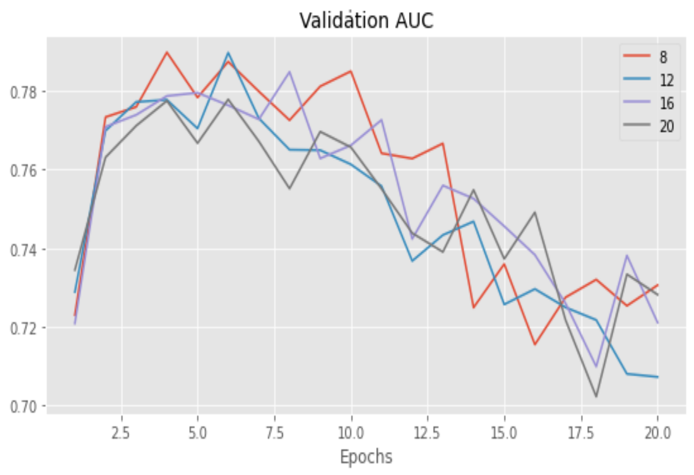


Figure 17: AUC by tuning Filters

3.4.3.6. Kernel Size

Similarly, an optimal value of kernel size was selected by evaluating the performance of model at each possible value from the range of possibilities as shown in the figure. The AUC is higher (0.77) for kernel size of (3, 3) as compared to rest of the values.

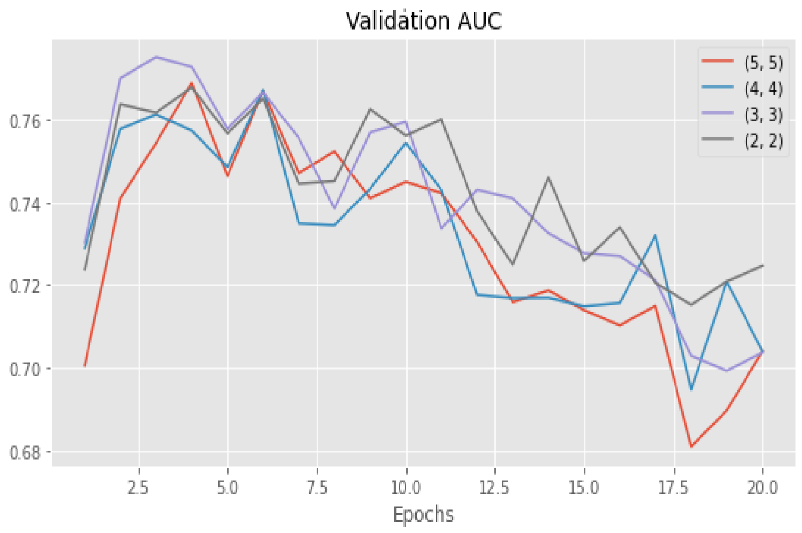


Figure 18: AUC by tuning kernel size

3.4.3.7. Dense Units – CNN

In order to select an optimal value for dense units, the classifier's performance was gauged at different possible values such as 34, 44, 54 and 64. The trends are close to each other but still the highest AUC value achieved by 54 dense units is clearly visible which is slightly greater than 0.78. Therefore 54 units were selected for CNN classifier.

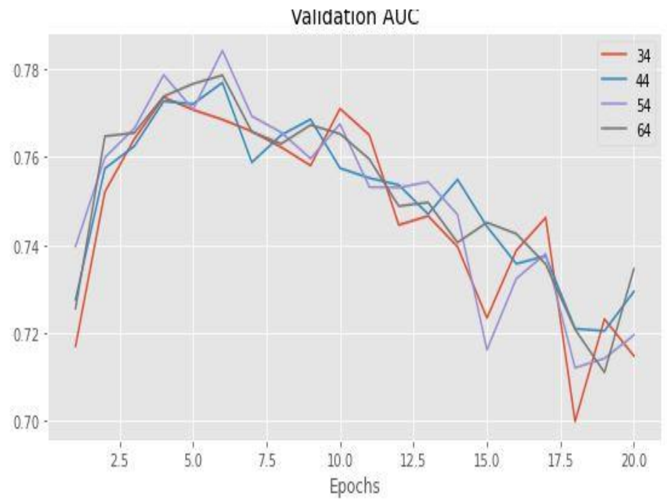


Figure 19: AUC by tuning CNN Units

3.4.3.8. Regularizer

With our limited data and a great number of learning parameters there was a chance of severe overfitting. To negate the effect, we employed dropout regularization with varying strengths, seen if figure below, to find an ideal value that would prevent our model from being too specific on the trained data. From the results obtained, a drop out strength of 0.25 was chosen, which obtained the highest AUC value of slightly less than 0.78, for our final model.

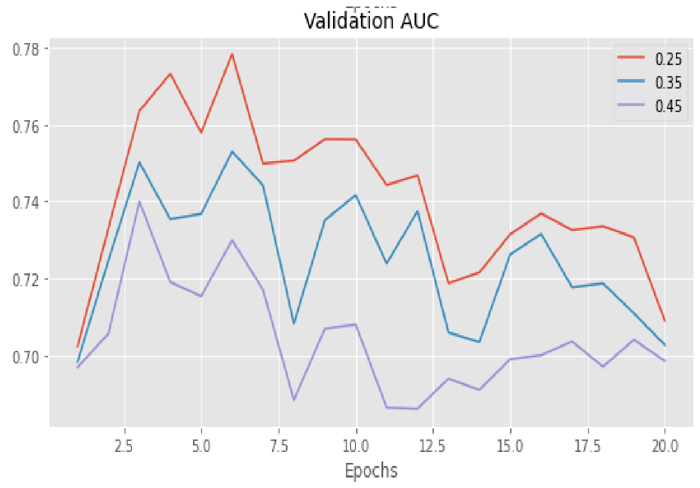


Figure 20: AUC by tuning Regularizer

3.4.3.9. Activation - WNN

To select the activation function for the WNN, the model's performance was evaluated at different activation functions. From the trends shown in the figure given below it's clear that the highest AUC value approx. (0.79) was achieved for 'Softmax'. Whereas the second highest AUC was achieved by 'tanh' approx. (0.78). Therefore 'Softmax' was selected as an activation function of WNN.

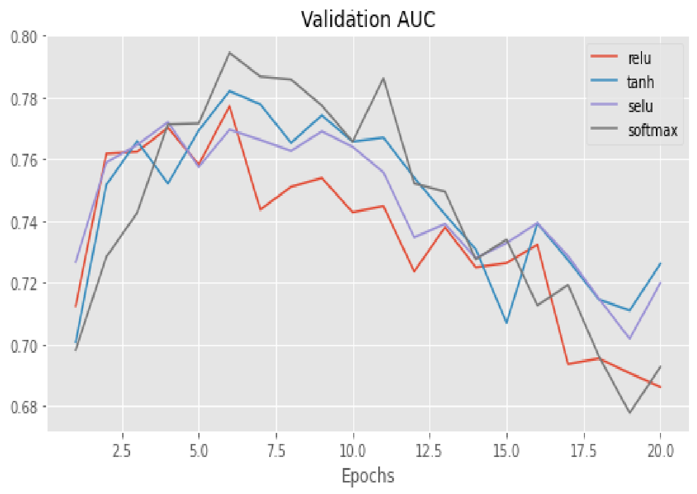


Figure 21: AUC by tuning WNN Activation

3.4.3.10. Summary of Hyperparameter Tuning

Subnet	**Parameter**	**Zheng**	**Optimal Value**	**AUC (Optimal)**
CNN	Regularizer	NA	DO OR BN	~0.79
	Activation	ReLU	SeLU	~0.76
	No. of Filters	18 ~ 20	8	~0.77 - ~0.78
	Kernel Size	(3, 3)	(3, 3)	~0.77 - ~0.78
	Dense Units	50	54	~0.78
	Dense Activ.	ReLU	SeLU	~0.78 - 0.79
	Pool Size	Max (3, 7)	Max (3, 7)	Not tuned

WNN	Units	64 (60% TR)	54	0.78
	Activation	ReLU	Softmax	0.78

Table 4: Tuned Hyperparameters in comparison with Zheng's

3.5. Class Imbalance

As part of this project, the following strategies and techniques were investigated as potential ways to address the class imbalance in the dataset.

- Weighted loss
- Focal loss
- Undersampling the majority class
- Oversampling the minority class
- SMOTE and ADASYN

With the exception of the first, the viability of each technique was first assessed on the two shallow learning models with the lowest training time, namely Logistic Regression and Random Forest. All hyperparameters for these models were set to those specified by [3] as described in (Sections on LogReg, RF). A confusion matrix normalized by true labels was then used to quantify the proportion of false predictions for both the positive and negative class. Only those techniques that were found to significantly decrease the models' false positives and false negatives in the confusion matrix were then applied to the WDNN.

3.5.1. Weighted Loss

Scikit-Learn’s compute_class_weight function was used with the `balance` argument [42] that implicitly assigns costs for each class such that the cumulative costs of all misclassifications in the training data will be roughly equal. That is, the weights for the majority and minority classes kmajority and kmajority are calculated using their proportions relative to the total training examples.

kmajoritynmajorityn=kminoritynminorityn≅0.5

This technique was tested with logistic regression, support vector machine, random forest, and wide and deep neural network classifiers.

For our dataset, classes 0 (regular consumers) and 1 (thieves) were assigned weights kmajority= 0.547 and kminority=5.860. Given the 91.5-8.5 split between the proportions of these classes, it is trivial to verify that

$$0.547\times0.915\cong5.860\times0.085\cong0.499$$

These class weights were then combined with class labels in a dictionary that was then passed to the fit function’s class_weights arguments for both Scikit-Learn and Keras models. In other words, these models assigned a ~10.7 times higher penalty for the misclassification of minority samples compared to majority samples.

3.5.2. Focal Loss

Having been proposed in 2018, focal loss has not yet been implemented in stable releases of many deep learning libraries such as Tensorflow. However, an implementation of the loss function was found in the addons module for the Tensorflow library, specifically as tensorflow.addons.sigmoid_focal_crossentropy

[29]. An object of this class was instantiated and passed to the loss argument when compiling a Keras model.

Given that this function has been designed to work only with neural networks, it was tested only with the wide and deep neural network.

3.5.3. Undersampling and Oversampling

The imbalanced-learn library [30] was used to implement both undersampling and oversampling. Unlike the loss functions, these techniques were not directly applied to the models themselves but rather to the training data used to modify the models' weights. A sampler object was instantiated that was then fit to the training examples. The RandomUnderSampler object randomly selected $n_{majority}$ training examples with label = 0 such that $n_{majority} = n_{minority\_original}$. This meant that the training set size was essentially reduced to $2 \times n_{minority\_original}$. A similar RandomOverSampler was instantiated and fitted to the training data. This object sampled minority class examples with replacement such that the $n_{minority} = k \times n_{majority}$, where $k \in [0.5, 1]$. While this increased the training set size, it also introduced a lot of redundancy in the training set owing to resampled minority data. Logistic Regression and Random Forest models were fit to training sets that were generated after undersampling and oversampling.

3.5.4. SMOTE and ADASYN

The imbalanced-learn library was also used to instantiate SMOTE, Borderline SMOTE, and ADASYN objects in much the same was as RandomUndersampler and

RandomOversampler. These objects with sampling strategies similar to those defined for RandomUndersampler and RandomOversampler to create synthetic minority training examples. SMOTE was also combined with majority undersampling, as [31] suggests this often leads to better model performance than standalone SMOTE. The synthetic dataset was then used to train logistic regression, random forests, and wide neural networks.

3.6. Final Model – definitive product of this study

The prior investigations had established the following facts

1. The WDNN outperformed all other models investigated as part of this project.
2. The optimal hyperparameters for the WDNN, as described in the table below
3. Weighted loss was the best strategy to address class imbalance

Combining all these facts, a new wide and deep neural network was instantiated on a standard scaled, stratified training set that consisted of 80% of the examples in the original dataset. Loss weights, as identified in (Section on Weighted Loss) were assigned to the classes during the fitting process.

The final model was evaluated by extracting the predicted probabilities of a given test set example belonging to the positive class. These probabilities were then compared with a classification threshold ϵ such that

$$\widehat{y_i} = \begin{cases} 1\ if\ p(\widehat{y_i}) = 1 > \epsilon \\ 0\ otherwise \end{cases}$$

Where $\widehat{y_i}$ is the predicted label for the i^{th} test set example.

This classification threshold was varied between $[0.5, 0.82]$ iteratively. For each classification threshold, the model's confusion matrix (normalized by true labels) was plotted to identify the proportion of examples of each class that were classified correctly.

We found that increasing the threshold makes it difficult for the model to correctly classify thieves, increasing the False negatives but reducing the false positive rate. Hence The classification threshold was thus tuned to achieve the best tradeoff possible between high true positive and true negatives while maintaining low false positives and false negatives.

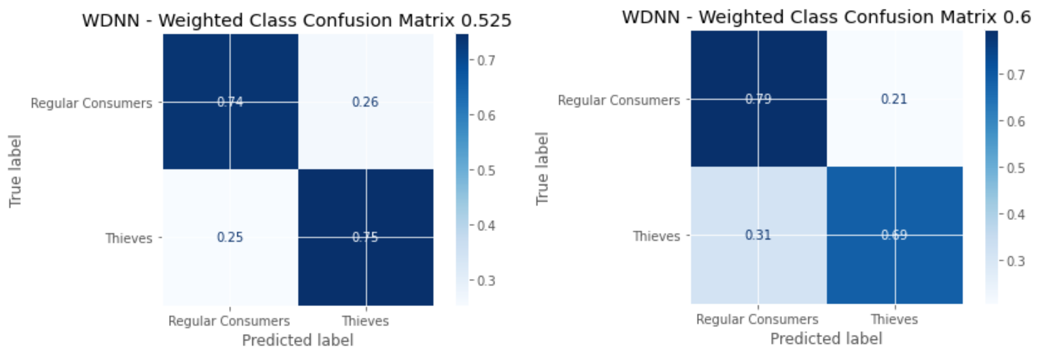


Figure 22: Visualizing effect of Increasing Threshold value

3.7. Model Deployment

In this day and age, Machine Learning (ML) models are useful to gain key insights from raw data that may give off an inconsequential vibe to a human eye. Minute adjustments in model parameters can lead to future defining business predictions that are essentially based on solid data collected in the past. But, ascending into an era, guided by ML models is still a far fetch idea for people who are not acquainted with proper data science skills. Fabricating ML models still requires experienced data scientists that help develop custom models for specific needs and wants. This lengthens the production process until the model has matured enough to make useful predictions, thereby adding no value to a potential client in the meantime. Hence for this purpose ML models are ***deployed*** by data scientists.

The deployment process can be formally defined as, to launch a pre-trained ML model on a platform that can be used for inferencing by a client. Deploying a ML model is analogous to shipping a software, except the ML model sits on a web or hardware based serving platform and this user only generates valuable predictions on his data. A great advantage to deploying models is that as a client adds new data to his storage he can make real-time predictions each day without the hassle of maintaining the code. When ML models are deployed they require a computing platform for churning out useful predictions as soon as a request is made. This platform is more often than not a web service hosted by a cloud based server to which data can be sent for inferencing. The data sent on the server is passed through the ML pipelines coded into the model and then fed into an estimator to generate predictions. Like a web server that is essentially a virtual computing machine, ML models can also be used for inferencing on a hardware based platforms like Intel OpenVINO compute stick [32]. This remote piece of hardware contains essential packages and toolkits that are designed to assist data scientists in advancing computer vision through AI, by providing plug-n-play operation. Like a web server, it can also be used to generate predictions on-the-fly.

After a model is deployed it needs to be monitored for the resources it uses and the expected range of data that can be processed to make a prediction. Furthermore, occasional updates in the libraries and dependencies should also have to be made to keep the model as efficient and optimized as possible.

3.7.1. Amazon Web Services

The final stage of our project required us to deploy our ML model on a platform that can be used for inferencing by a client by posting a http request on the cloud server. For this purpose we employed Amazon Web Services (AWS) which is a cloud computing platform serving 34% of the cloud market all over the world [33]. It provides on demand computing as well as custom API interactions to its customers without the hassle or a huge capital of setting up a server computing machine. AWS gives the essence of real computers with all its complexities like a storage device, GPUs, RAM and runtime optimization fundamentally available through the internet. Depending on your budget, you can have multiple computers working in a cluster on the cloud at your ease, fully managed by AWS.

3.7.2. Sagemaker

One of the services provided by AWS is ***Amazon Sagemaker*** which is a fully managed cloud ML environment that can be used by developers to build, train and deploy ML models with ease [33]. It was intended to accelerate the conventional process of ML model building by ramping up the production phase through cloud computing. Hundreds of data science enthusiasts can access the compute force as well as the fully integrated IDE known as ***Sagemaker Studio*** for building their custom models. Sagemaker Studio helps keep track of ML projects with a user friendly interface, guiding us through the treacherous paths of becoming a data scientist. It provides pre built ML model images that can be custom tuned for specific needs. With an addition of ***Sagemaker Autopilot*** feature, the model building process has taken a huge leap. Autopilot allows Sagemaker to decide which ML model is the best fit for your data by evaluating on a spectrum of algorithms and comparing their prediction performance. This feature adds another layer of abstraction between developing complete end to end ML models and optimizing code for your data.

Sagemaker Studio also helps to keep track of training jobs performed in the past and maintains an evaluation score that can be used to assess the model's performance. Another advantage of Sagemaker is that it allows you to code in multiple languages like python, R, Julia, GO etc.

3.7.3. S3 Bucket

Like any computer, a cloud based compute instance also requires a storage system to handle data flow. In AWS's terms this storage is a bucket or more specifically a S3 bucket that can be linked when a compute instance is launched. S3 bucket is actually a fast speed SSD storage in the cloud that is logically partitioned and set up based on the user's needs. All the data that resides in this bucket is uniquely identified by a key and a prefix pair that can be called from a notebook runtime environment to load that data. When a python based Jupyter notebook is launched in the Sagemaker Studio we have to link this bucket to the notebook to access data. An advantage of cloud based storage systems is that large amounts of data can be stored securely and accessed immediately by the compute instance without any latency issues between the server and the client.

3.7.4. Notebook Instance

Amazon Sagemaker gives access to independent Jupyter notebook instances that are essentially isolated computing environments running on a compute instance on the cloud. When you launch a notebook instance you can configure the following things:

- Notebook instance name
- S3 bucket, for storing and retrieving data
- Compute instance type, for running training jobs
- IAM role, that attaches a permissions policy for security purposes

This notebook can then be fundamentally used as a Jupyter notebook with its core functionality of running individual code snippet style programming. For making use of other amazon services like fast speed S3 storage, multiple compute instances, and endpoint creation we have to set up Sagemaker runtime that allows its internal architecture to communicate with the notebook. These notebooks can also be used for setting up model endpoints that can be invoked for making predictions, as will be discussed in the sections that follow.

3.7.5. AWS Lambda

Lambda is yet another cloud based computing service offered by AWS with the difference that it is quite essentially serverless for the user [34]. The user doesn’t have to indulge in launching or managing a server instead he is handed by the cloud. AWS lambda is very helpful in launching APIs or web applications as it has the capability to instantly relocate resources in response to a client request or some other event. It only bills for the compute resources used to run the code packaged and sent as a server request.

3.7.6. Deployment Setup

To deploy our model to the cloud we make full use of AWS cloud platform's ***free tier service*** that provides us free of cost cloud computing service for smaller compute instances and a basic storage service (***S3 bucket***) for a period of one year [35]. The process of deploying a model on AWS is mainly broken into 4 main components that need to be set up individually and should be configured to work in unison to be able to handle web requests and make predictions on the go. A brief description of all these components with their high level configuration settings are provided below.

3.7.6.1. Sagemaker Endpoint

To be able to generate inferences from a model by sending data over the web requires us to maintain an endpoint. An endpoint is essentially one end or a node in a communication network that can be accessed by other interfaces upon request [36]. A Sagemaker model endpoint is basically an entire trained model sitting on one end of a bidirectional communication channel that can be invoked for making predictions. This can be thought of a locally trained model with all its weights and artifacts just separated by a web server. This endpoint of course runs on a compute instance that is specified when deploying the endpoint. A Sagemaker endpoint can make predictions utilizing only a single compute instance or can make use of a cluster of instances running parallel jobs for extensive tasks. This endpoint is fully managed by AWS and is unique for each notebook instance.

3.7.6.2. Lambda Function

As discussed above AWS lambda is a serverless compute service that allows us to make web applications and handles http requests. Lambda function is fundamentally like any other function that takes input arguments and performs predefined tasks or they may be used to channel data to the function body where it performs an inference job on the inbound data. For our project we used a custom lambda function that can provide us with the functionality of data delivery to the model endpoint. The lambda function handler in python takes 2 arguments: *event* and *context* [37]. The context argument is for configuring Sagemaker runtime environment while the event argument is for pay loading data. The body of our lambda function contains the *invoke_endpoint* method that makes a call to our Sagemaker endpoint. In this way we can either send individual dictionaries containing features of a test sample for making predictions or a csv file for making batch predictions.

3.7.6.3. API Gateway

An Application Programming Interface (API) is a tool that allows applications to interact with a computing interface for accessing services or data that may reside on another system [38]. An API Gateway acts as a door that separates a web based client from backend services and defines rules for exchanging information between the two. APIs are useful in ways that they can be configured to manage tasks such as handling web requests, monitoring API access points, authentication service, limiting overuse of the computing instance and, for our case, making predictions by handling a lambda function.

Our project defines a custom Amazon API Gateway that is configured to handle requests between a web server and the lambda function. The API can handle http requests and uses POST method for sending requests to the model endpoint. For setting up the API it needs to reside in the same AWS cloud region [39] where the endpoint is deployed. It makes use of our custom lambda that we defined earlier and is deployed using the AWS API service.

3.7.6.4. Web Client

A web client is essentially a user's browser that serves as the second end in our communication channel, whose workflow is defined after this section. To access the web service or in this case our Sagemaker model endpoint the user sends a request from his browser to the API gateways that is processed and forwarded to the model for making predictions. This request can be made directly from a browser or by using an API client software like ***POSTMAN*** that makes it easy for testing the API in the development phase.

3.7.7. Cloud Processing Workflow

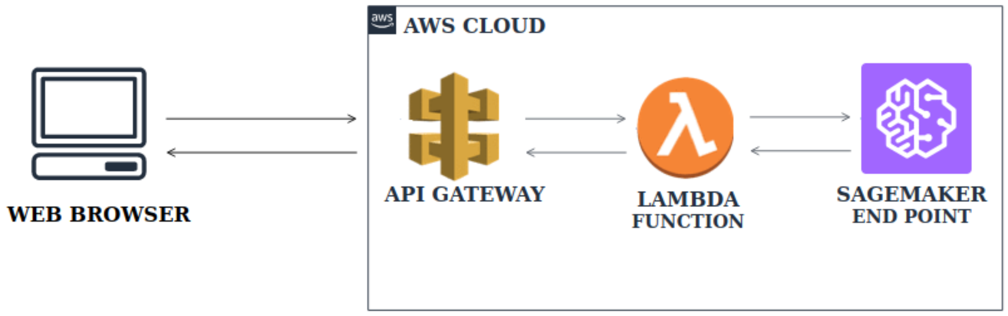
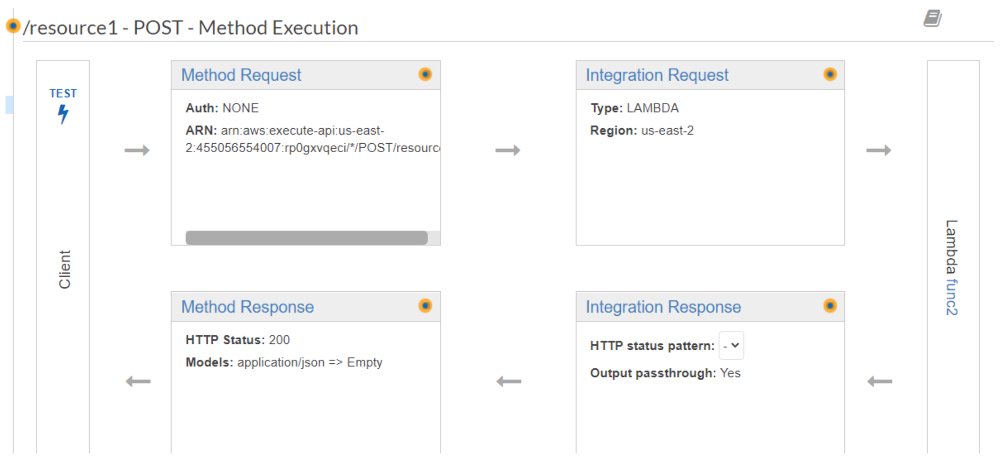


Figure 23: Cloud Processing Workflow

The API interface as vitalized by AWS API dashboard:


/resource1 - POST - Method Execution

Method Request
Auth: NONE
ARN: arn:aws:execute-api:us-east-2:455056554007:rp0gxvqeci/*/POST/resourc

Integration Request
Type: LAMBDA
Region: us-east-2

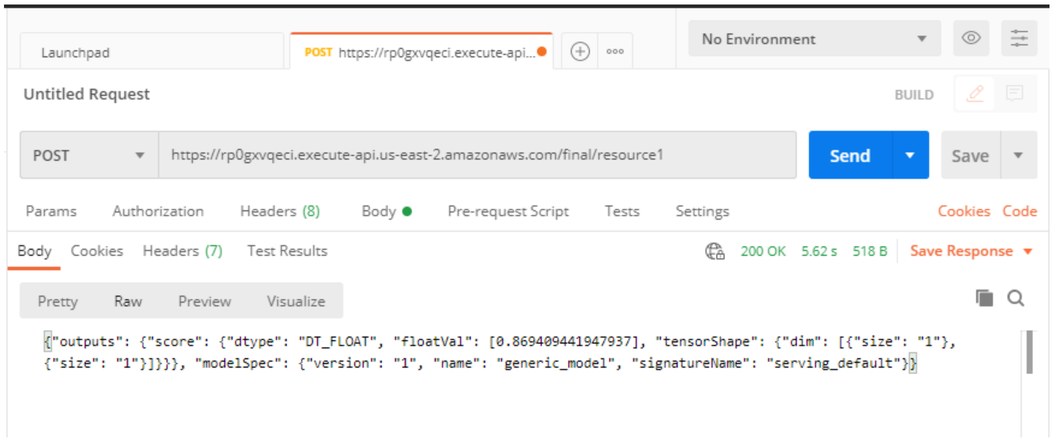
Method Response
HTTP Status: 200
Models: application/json => Empty

Integration Response
HTTP status pattern: -
Output passthrough: Yes

Figure 24: API Structure

An overview of the workflow of model deployment and invoking and endpoint for making predictions can be seen in the figure above. After configuring all the components the user sends requests via his browser or any other API client application to the Amazon API Gateway. The API Gateway communicates between the Lambda functions and the user and exchanges the data coming from the user. In our Lambda functions the data is parsed to the ***lambda_handler*** method that stores the features in its event argument in the form of either a dictionary, csv file or a json tree. This argument is then refined within the fiction’s body and a call to the ***invokeendpoint*** method is made that defines the compute instance for running the prediction job and the data forwarded by the client. Finally the prediction made by the model endpoint is stored in a json file and passed back to the user’s browser via the API Gateway.

The response generated by invoking the endpoint URL from postman is a JSON tree visualized below:



```
https://rp0gxvqeci.execute-api.us-east-2.amazonaws.com/final/resource1
```

```
{"outputs": {"score": {"dtype": "DT_FLOAT", "floatVal": [0.869409441947937], "tensorShape": {"dim": [{"size": "1"},
{"size": "1"}]}}}, "modelSpec": {"version": "1", "name": "generic_model", "signatureName": "serving_default"}}
```

Figure 25: Response Generated by POSTMAN

CHAPTER 4
RESULTS

4. RESULTS

Table 5: Preliminary Results of Models Trained

	Training Ratio = 80%			
Methods	**Scaling**	**AUC from our experimentation**	**Scaling**	**AUC from Zheng's Models**
Logistic Regression	Standard	0.71	Min-Max	0.706
SVM	Min-Max	0.77	Min-Max	0.7413
Random Forest	Min-Max	0.76	Min-Max	0.7385
Wide Neural Network	Standard	0.74	Min-Max	0.6965
CNN	Standard	0.77	Min-Max	0.7797
WDNN	Standard	0.82	Min-Max	0.7815

4.1. Logistic Regression

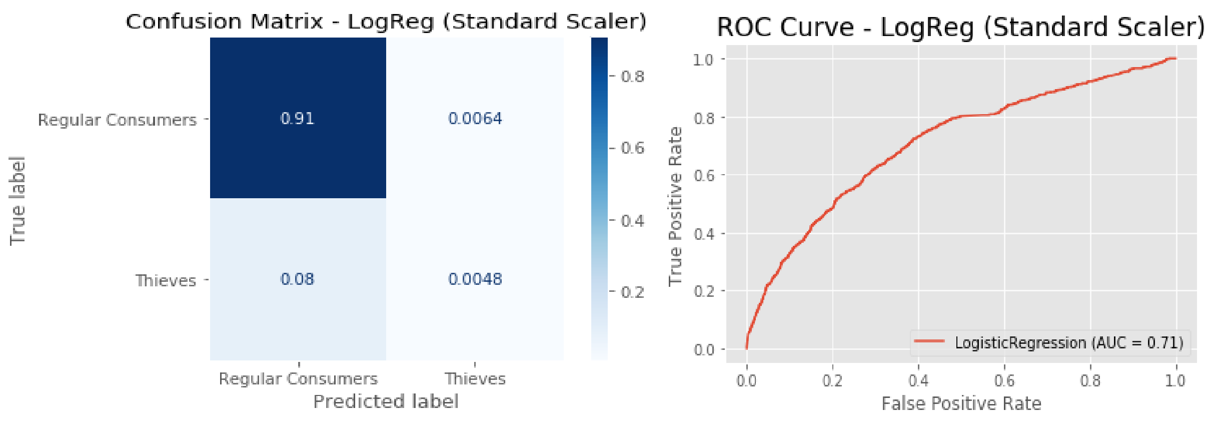


Figure 26: Confusion matrix and ROC Curve of Logistic Regression

Extensive experimentation on logistic regression was performed as the results obtained were not satisfactory and lagged far behind results obtained using hyperparameters mentioned in [3]. With 500 iteration, ‘liblinear’ solver and an L2 penalty, the very first modeled trained on min-max scaled data gave an AUC of **0.52.** Similar hyperparameters but with a SGDC Classifier gave an AUC of **0.706**. In addition to this a grid search was also performed on the inverse regularization parameter ‘c’ which didn’t improve the model much. Finally the standard scaled data was used in hopes of improving the AUC which had a marked effect on the classifier and bumped the AUC to **0.71.**

4.2. SVM

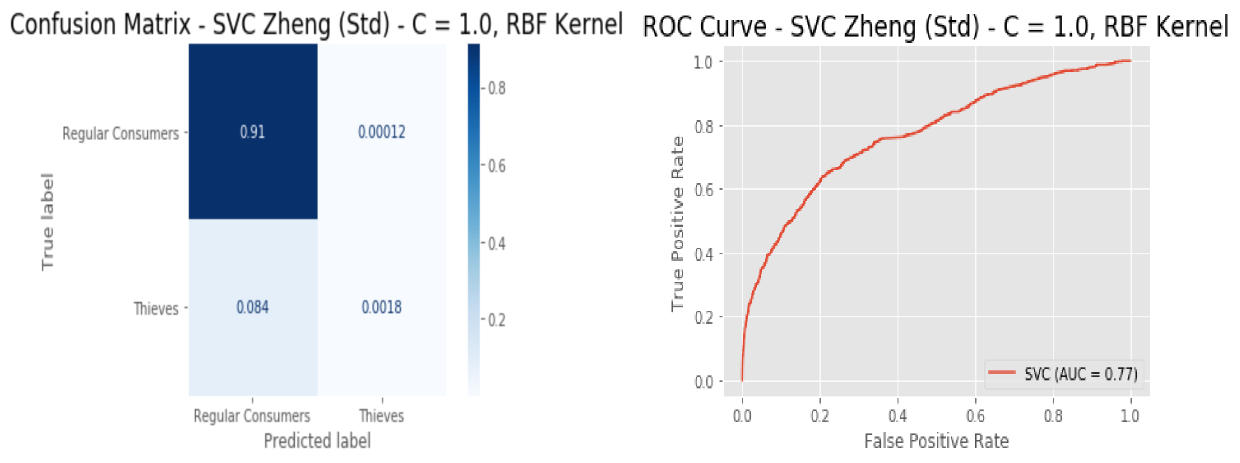


Figure 27:Confusion matrix and ROC curve of Support Vector Classifier

The Support Vector Classifier was trained using the standard scaled data. A very common Radial Basis Function (RBF) kernel was used with a penalty parameter C = 1.0. These hyperparameters were selected following previous investigations on SVMs for classifying electricity thieves from normal consumers [3]. A larger value of penalty parameter defines that the classification boundary opt by the SVM would be considerably small hence would lead to a better classification of the features. The results of SVM classifier for the purpose of theft detection as illustrated in [3] gave us an AUC of **0.7413**. This result was obtained by scaling the data according to Min-Max method. A self-experimentation of the same model with the hyperparameters mentioned in [3] improved the AUC to **0.77** with a training ratio of 80% and using the same scaling procedure for the features. Further investigation in hopes of improving the AUC led us to train the model with standard scaled data as it was proven to slightly outperform the models trained on min-max scaled data. But after training the model it gave us an gave us an AUC of **0.75** with standard scaling and 80% training ratio.

4.3. Random Forest

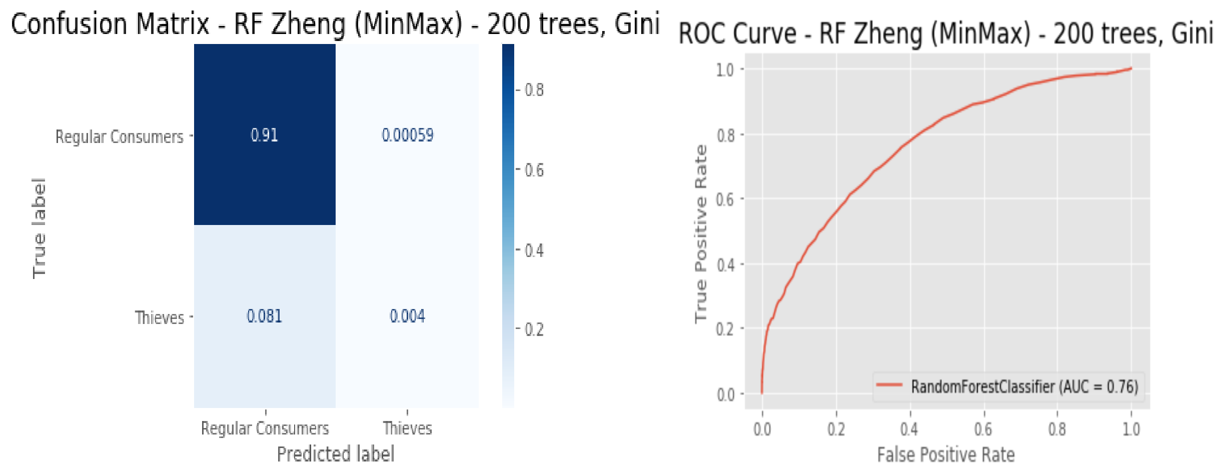


Figure 28:Confusion matrix and ROC curve of Random Forest Classifier

For this experimentation 2 separate models were trained using the standard scaled and Min-Max scaled data. Other hyperparameters include 200 decision trees working on ‘Gini’ criterion which were selected as per previous investigations in this regard [3]. Note that the max_depth hyperparameter was not initialized in any training of the model. The model trained using the Mix-max scaled data gave an AUC of **0.76** while the same hyperparameters gave a marginally better AUC of **0.73** with standard scaled data. Just like the SVC classifier the Random Forest model performed slightly better on min-max scaled data. Increasing the number of estimators (trees) may produce better results but with a risk of overfitting the training data and a considerable increase in training time. With a satisfactory set of metrics obtained with 200 estimators, no parameter search was performed.

4.4. Wide Neural Network

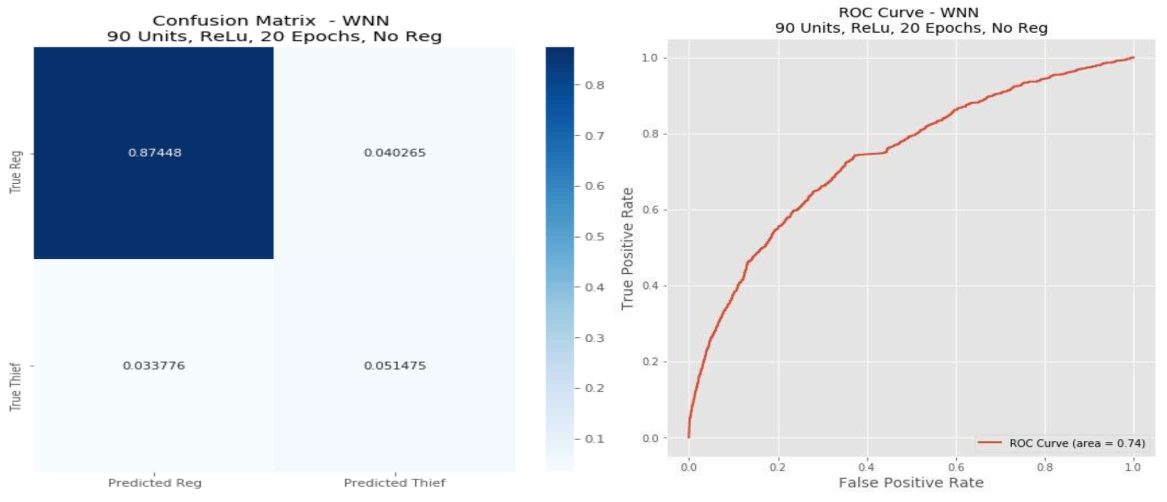


Figure 29: Confusion matrix and ROC curve of Wide Neural Network

A wide Neural Network with hyperparameters based on previous investigation [3] was also trained that outperformed other classifiers. The first dense layer had 90 neurons with a 'Relu' activation that fed a sigmoid classifier which performed the binary classification between thieves and normal consumers. The model was trained for 20 epochs only as there was no dropout layer in the initial models that would prevent overfitting of the model on the data. The AUC achieved using the above described model was **0.74** which was far better than the results produced by other shallow classifiers. Although the same standard scaled data was used for this experiment as well.

4.5. Convolutional Neural Network

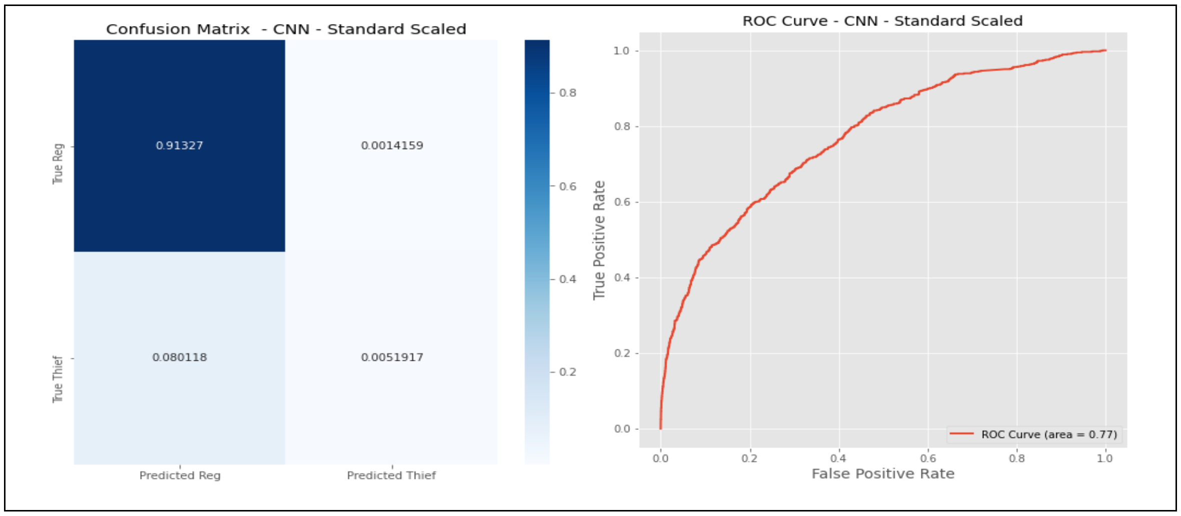


Figure 30: Confusion matrix and ROC curve of Convolutional Neural Network

A convolutional Neural Network was trained after selecting the optimal parameters mentioned in [3]. The results depicts that CNN clearly outperforms the shallow learning classifiers previously trained, although the same standard scale data with similar training ratio was used for the classifier. The model was trained for 20 epochs with 54 dense units and ‘Selu’ activation that fed a sigmoid classifier which performed the binary classification. The AUC achieved was 0.77 which was comparable the AUC achieved by the previous studies i.e. 0.75 for the same classifier [3]. Even though the same AUC was achieved for Support Vector Classifier but due to lengthy training time and the lower TP rate associated with SVM, the CNN model is still preferred.

4.6. Wide and Deep Convolutional Neural Network (WDNN)

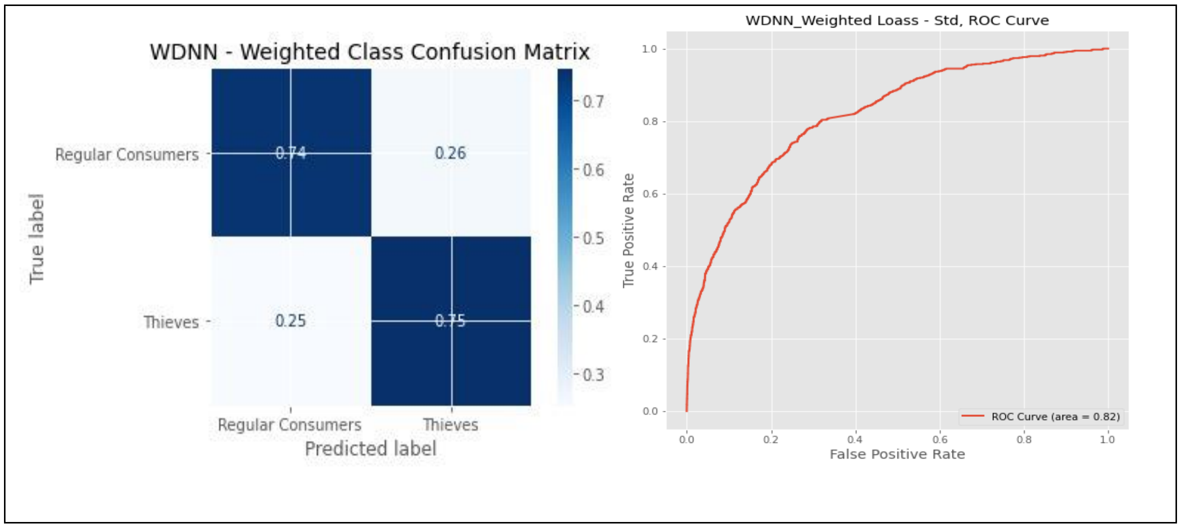


Figure 31: Confusion matrix and ROC curve of Wide and Deep Convolutional Neural Network

Finally the Wide and Deep Convolutional neural network was trained with the specified hyperparameters in table 3. The model outperforms all the existing classifiers for this project with the highest AUC value of 0.82, the model was trained on standard scaled data for 20 epochs with 54 dense units for both WNN and CNN components, whereas ‘Softmax’ was chosen as activation function for WNN component and SeLU for CNN component. Certain Hyperparameter values selected for WDNN through hyperparameter tuning were different as compared to the hyperparameter values for the same classifier in the previous studies [3]. The balanced dataset by weighted loss with better optimization of hyperparameter values has enabled the WDNN to achieve an AUC of **0.82** which is far better than the AUC achieved by the same classifier in the previous studies which was **0.7815** [3]. Hence WDNN was proved to be the best model for the classification of thieves and regular consumer of electricity.

CHAPTER 5
CONCLUSION, FUTURE RECOMMENDATIONS

5. CONCLUSIONS, FUTURE RECOMMENDATIONS

5.1. Successful Proof of Concept

This investigation concludes that shallow and deep learning-based approaches for electricity theft detection data are both viable and effective. All models have demonstrated the ability to extract meaningful transformations of kWh consumption patterns from 1D timeseries data in order to distinguish electricity thieves from regular consumers. Experimental results demonstrate that all models, with the exception of Random Forest classifiers, offer better classification performance when trained with standardized data rather than the min-max normalized data recommended by [3]. This difference in feature scaling techniques is also why experimental results of this investigation are approximately 1 - 5% higher than those presented in [3]'s research paper for the same training ratio.

5.2. Deep Learning Outperforms Shallow Learning

Deep neural models have been observed to outperform their shallow learning counterparts. Their ability to apply successive non-linear transformations to input data allows them to learn richer and more meaningful representations of the input data. Of the shallow learning models, only the random forest classifier has comparable classification and runtime performance to DL models, owing to its ensembling of multiple decision trees.

5.3. Improvement on Zheng's Benchmark

The ROC AUCs from our models are higher than those of Zheng's for all learning algorithms with the exception of the convolutional neural network. This goes to show that we have improved on benchmark set by Zheng et al through a more rigorous investigation of feature scaling techniques and extensive hyperparameter tuning. This is illustrated in the figure below:

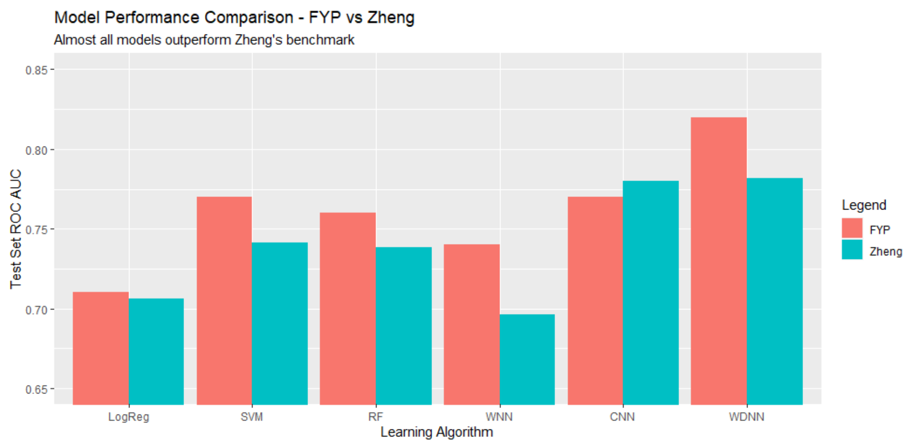


Figure 32: Model Performance Comparison

The greatest percentage improvement over benchmark results is observed in the case of the WNN, followed by the WDNN and then the SVM. The decrease in performance relative to the benchmark observed in case of the CNN may be the result of lack of hyperparameter tuning, or the model overfitting the data due to a large number of filters. Our implementation of the CNN increased the number of filters in each convolutional layer from 18 → 288, which may have substantially increased the network's memorization capacity and thus its tendency to overfit the data.

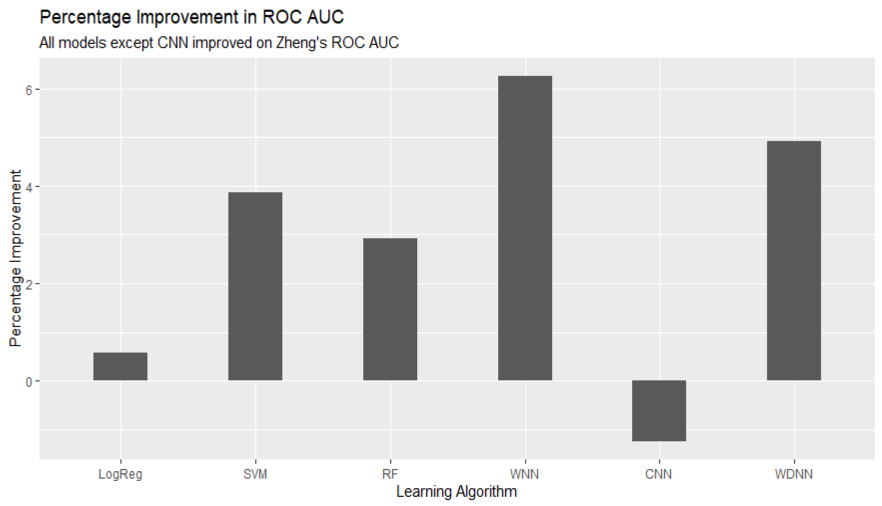


Figure 33: Improvement in ROC over Zheng's Study

5.4. WDNN as the Best Model

We conclude that the wide and deep convolutional neural network is indeed the best model for this supervised binary classification task. It has the highest ROC AUC score of all models on the test set, and also has the lowest proportions of both false positives and false negatives. While other models such as the WNN, RF, and LR may have somewhat comparable AUCs and higher true negative (TN) rates, their FPs are simply too high for us to use them in the context of electricity theft detection. This is why the WDNN was chosen to be model that is deployed on AWS as a deliverable for this project.

5.5. No Optimal Classification Threshold

In the context of the WDNN, increasing the classification threshold ϵ increases the proportion of false negatives (FNs) and decreases the proportion of true negatives (TNs). However, this is also accompanied by a decrease in the proportion of FPs.

As such, we conclude that there is no single optimal value for the WDNN classification threshold. A client using the model may wish to choose a higher or lower threshold depending on the balance the client wishes to strike between identifying the maximum number of thieves (maximizing TPs) and minimizing the number of regular consumers mistakenly classified as thieves (minimizing FPs).

5.6. Class Imbalance Techniques

We conclude that loss computation weighted by class weights is the best technique for addressing the problem of class imbalance in this dataset. This technique was observed to bring about the largest decrease in FPs and FNs in both LR and WDNN (the simplest and most complex models respectively). It is computationally inexpensive and compatible with most Machine Learning toolkits available today, which makes it an excellent choice for addressing class imbalance in production.

SMOTE and ADASYN seemed promising but failed to mitigate the impact of class imbalance. We attribute this to issues in data quality. The labels used for generating class weights for weighted loss remained unchanged throughout this project. As such, it is highly likely that they represent the true distribution of regular consumers and thieves in the dataset. However, the features used by SMOTE and ADASYN were heavily imputed due to a large number of missing values. Synthesizing new training examples from data that was already, in part, synthetic thus limited the efficacy of these techniques.

5.7. Data Quality

The most valuable lesson learnt through the course of this project was about the importance of having good quality data. We are confident that our techniques and models are capable of much better performance on this classification task, but are limited by the quality of the data used to train them. Our models could have performed much better with more training examples, fewer missing data points, and additional timeseries data for each consumer (e.g. power factors or line voltages).

5.8. Dimensionality Reduction

Since 95% of explained variance in the dataset can be attributed to ~660 features, we conclude a lot of kWh data is redundant. However, given that our experiments failed to yield substantial improvements in time complexity of SVMs with a subset of the original kWh data, we conclude that none of the dimensionality reduction techniques are suitable for incorporating into a production level theft detection model.

5.9. On Specific Models

A WNN trained on standardized data with the same hyperparameters as those presented in [3]'s work has the second highest ROC AUC score among all investigated models. Trends in the model's validation set loss and ROC AUC scores show evidence of overfitting which, counterintuitively, has not been mitigated by the use of conventional regularization techniques such as Dropout, L1, and L2 kernel regularizers. This suggests the model may have too large a memorization capacity [12], which may be mitigated by lowering the number of neurons or further hyperparameter tuning.

Results also indicate that Random Forest (RF) classifiers are the best shallow learning approach to the problem. Despite not requiring the computational overhead of feature scaling common to almost all other models, RF classifiers have the highest ROC AUC score across all investigated models. However, this is likely due to the fact that RF classifiers are an example of ensemble learning [14]: specifically, a collection of Decision Tree classifiers learning and predicting independently on the same data. Therefore, any comparison between the performance of standalone models such as logistic regression (LR) and support vector machines (SVM)s must account RFs being an ensemble learning solution.

While SVMs demonstrate incrementally better performance on this classification problem than the Wide Neural Network (WNN), this improvement is far outweighed by the disproportionately large training and prediction times. SVM time complexity of $O(n \times m^2) - O(n \times m^3)$ [15] means a theft detection model based on an SVM classifier is highly unlikely to scale well with training examples of a real-world distribution system, although it may find use as an online learning classifier that improves its weights incrementally using individual samples [15].

Logistic Regression offers the worse performance on this classification task, which is a consequence of the fact that the model has high bias: it assumes the kWh data of consumers and regular consumers is linearly separable. Since the logistic regression model is based on a linear combination of features and weights, its hypothesis space is far too limited to capture all the non-linearities necessary for successfully distinguishing a regular consumer from a thief. That being said, its ROC AUC score is still of the same order as all other models, and the model itself is far more interpretable than WNN and SVM classifiers.

5.10. Future Work

Due to time and resource constraints, the authors were unable to explore the following techniques in as detailed a manner as desired. However, we encourage future researchers interested in pursuing the problem of applying machine learning to electricity theft detection to explore the following areas.

5.10.1. Sequence Models

Sequence models: All models investigated as part of this project treat the 1034-dimensional vector ofse kWhs of a single consumer as a collection of individual features that do not necessarily form a sequence. While transformations of these features made by neural networks may indeed result in linear combinations, these linear combinations are not likely to preserve or exploit the inherent chronology of kWh consumption patterns.

We encourage researchers interested in carrying this project forward to explore the use of sequence models: neural networks that are designed to exploit the temporal relationship between successive input features to extract meaningful, long or short term dependencies and patterns in time series data.

Sequence models such as 1D convolutional neural networks and long short term memory models (LSTMs) can be used to process the kWh data for individual consumers as chronological sequences of information. Doing so will likely allow these models to learn both the periodicity that Zheng describes as a distinguishing criterion for thieves as well as other temporal features that are associated with electricity theft that are more abstract than simple periodicity. This is because unlike neural networks discussed in this project, sequence models have the functionality of memory or state: the individual recurrent units within sequence models use techniques such as attention and memory to retain long-term dependences between inputs across different timesteps.

In particular, we recommend future researchers to explore a combination of a 1D CNN and bidirectional LSTM as a hybrid sequence model as suggested by Hasan et. al [46]. 1D convolutions can extract generic temporal features from the 1034-dimensional sequence vector which can drastically reduce the dimensionality of the input to a recurrent network such as an LSTM. The bidirectional nature of the LSTM will allow the network to examine temporal changes in the feature patterns extracted by the CNN from both past to present and from present to past. Doing so can often help neural networks learn richer and more contrastive representations of the input data which, in turn, can help the networks distinguish between positive and negative classes.

5.10.2. Anomaly Detection

It may also be useful to frame this investigation not as a supervised binary classification problem, but rather as an anomaly detection problem. The 8.5% of thieves in the dataset are perhaps better treated as anomalous training examples amongst a majority of regular consumers. Anomaly detection algorithms, whose implementations are readily available in toolkits such as Scikit-Learn, are semi-supervised or unsupervised in nature, which means future researchers will no longer be limited to using fully labeled datasets in their investigation [47]. Anomaly detection algorithms will attempt to classify kWh consumption values in the dataset that deviate “sufficiently” from the majority of the training examples in the feature space. Based on our investigation in this project, it is safe to assume that the difference between kWh feature vectors of regular consumers and thieves will be sufficiently distinguishable by anomaly detection algorithms [48]

5.10.3. AWS Batch Transform

The API deployed for this model is a simple proof-of-concept that shows a pre-trained neural network can be made available for inference on a cloud platform like AWS. However, its processing is sequential in nature: a single request sent to the API can only contain features for a single example. As such, inference for multiple examples requires sending multiple requests to the API.

This can be improved by modifying the way inference is performed on the SageMaker instance. Specifically, we recommend future researchers to use AWS’s Batch Transform suite of inference methods, which are specifically designed to perform inference on large datasets [49]. Batch Transform will allow the API to use concurrency and parallel processing to simultaneously generate predictions for multiple training examples, taking full advantage of the scalability of neural network models.

7. REFERENCES

[1] R. Jiang, R. Lu, Y. Wang, J. Luo, C. Shen, and X. S. Shen, "Energy-theft detection issues for advanced metering infrastructure in smart grid," *Tsinghua Science and Technology*, vol. 19, no. 2, pp. 105-120, 2014.

[2] S. McLaughlin, B. Holbert, A. Fawaz, R. Berthier, and S. Zonouz, "A multi-sensor energy theft detection framework for advanced metering infrastructures," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 7, pp. 1319-1330, 2013.

[3] Z. Zheng, Y. Yang, X. Niu, H.-N. Dai, and Y. Zhou, "Wide and deep convolutional neural networks for electricity-theft detection to secure smart grids," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1606-1615, 2017.

[4] R. M. Rotty, "Growth in global energy demand and contribution of alternative supply systems," *Energy*, vol. 4, no. 5, pp. 881-890, 1979.

[5] The News, "Areas of power theft will not be exempted of load shedding: Kh Asif", 2017. [Online]. Available: https://www.thenews.com.pk/latest/196271-Areas-of-power-theft-will-not-be-exempted-of-load-shedding-Kh-Asif [Accessed: 25-Feb-2020]

[6] Pakistan Today, "'Power theft' areas to brave load shedding, says minister", 2019. [Online] Available: https://www.pakistantoday.com.pk/2019/06/08/power-theft-areas-to-brave-load-shedding-says-minister/ [Accessed: 25-Feb-2020]

[7] F. Jamil and E. Ahmad, "An Empirical Study of Electricity Theft from Electricity Distribution Companies in Pakistan", Ph.D, National University of Sciences and Technology, Islamabad, 2019. [Online]. Available: https://www.pide.org.pk/psde/pdf/AGM29/papers/Faisal%20jamil.pdf [Accessed: 25-Feb-2020]

[8] ARY News, "Power theft causes financial loss of more than Rs45 bln: NEPRA", 2019. [Online]. Available: https://arynews.tv/en/power-theft-financial-loss/ [Accessed: 25-Feb-2020]

[9] "Smart meters help reduce electricity theft", Bchydro.com, 2020. [Online]. Available: https://www.bchydro.com/news/conservation/2011/smart_meters_energy_theft.html. [Accessed: 26- Feb- 2020].

[10] B.C.Costa, B. Alberto, A. M. Portela, M. W and E. O.Eler, "Fraud Detection in Electric Power Distribution Networks using an Ann-Based Knowledge-Discovery Process", *International Journal of Artificial Intelligence &*

Applications, vol. 4, no. 6, pp. 17-23, 2013. Available: 10.5121/ijaia.2013.4602.

[11] C. Ramos, A. Souza, G. Chiachia, A. Falcão and J. Papa, "A novel algorithm for feature selection using Harmony Search and its application for non-technical losses detection", *Computers & Electrical Engineering*, vol. 37, no. 6, pp. 886-894, 2011. Available:10.1016/j.compeleceng.2011.09.013.

[12] F. Chollet, *Deep learning with Python*. New York: Manning Publications Co., 2018, pp. 108-110.

[13] A. Géron and R. Demarest, *Hands-on machine learning with Scikit-Learn and TensorFlow*. Sebastopol (Clif.) [etc.]: O'Reilly, 2019, pp. 163-164.

[14] H. Cheng, "Wide & Deep Learning: Better Together with TensorFlow", *Google AI Blog*, 2016. .

[15] H. Cheng et al., "Wide & Deep Learning for Recommender Systems", *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 2016. Available: https://arxiv.org/pdf/1606.07792.pdf. [Accessed 25 February 2020].

[16] V. Chandola, A. Banerjee and V. Kumar, "Anomaly detection", *ACM Computing Surveys*, vol. 41, no. 3, pp. 1-58, 2009. Available: 10.1145/1541880.1541882.

[17] "K-Electric’s Drive Against Defaulters and Power-theft Continues Full Throttle - K-Electric", *K-Electric*, 2020. [Online]. Available: https://www.ke.com.pk/2019/01/03/k-electrics-drive-against-defaulters-power-theft-continues-full-throttle/. [Accessed: 28- Feb- 2020].

[18] G. James, D. Witten, T. Hastie and R. Tibshirani, *An introduction to statistical learning*. New York: Springer Science+Business Media, 2017.

[19] A. Géron and R. Demarest, *Hands-on machine learning with Scikit-Learn and TensorFlow*. Sebastopol (Clif.) [etc.]: O'Reilly, 2019, pp. 100.

[20] F. Peccia, "Batch normalization: theory and how to use it with Tensorflow", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/batch-normalization-theory-and-how-to-use-it-with-tensorflow-1892ca0173ad. [Accessed: 12- Sep- 2020].

[21] S. Glen, "Undersampling and Oversampling in Data Analysis - Statistics How To", *Statistics How To*, 2020. [Online]. Available: https://www.statisticshowto.com/undersampling/#:~:text=Undersampling%20attempts%20to%20reduce%20the,used%20to%20fit%20a%20model). [Accessed: 12- Sep- 2020].

[22] G. Lemaitre, F. Nogueira, D. Victor and C. Aridas, "2. Over-sampling — imbalanced-learn 0.5.0 documentation", *Imbalanced-learn.readthedocs.io*, 2020. [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html#mathematical-formulation. [Accessed: 12- Sep- 2020].

[23] G. Lemaitre, F. Nogueira, D. Victor and C. Aridas, "2. Over-sampling — imbalanced-learn 0.5.0 documentation", *Imbalanced-learn.readthedocs.io*, 2020. [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html#smote-variants. [Accessed: 12- Sep- 2020].

[24] A. Ng, "Padding - Foundations of Convolutional Neural Networks | Coursera", *Coursera*, 2020. [Online]. Available: https://www.coursera.org/learn/convolutional-neural-networks/lecture/hELHk/pooling-layers. [Accessed: 12- Sep- 2020].

[25] A. Ng, "Pooling Layers - Foundations of Convolutional Neural Networks | Coursera", *Coursera*, 2020. [Online]. Available: https://www.coursera.org/learn/convolutional-neural-networks/lecture/hELHk/pooling-layers. [Accessed: 12- Sep- 2020].

[26] [8]"Prediction of quality of Wine", *Kaggle.com*, 2020. [Online]. Available: https://www.kaggle.com/vishalyo990/prediction-of-quality-of-wine. [Accessed: 12- Sep- 2020].

[27] H. Pandey, "House Price Prediction (Regression) with Tensorflow— Keras", *Medium*, 2020. [Online]. Available: https://medium.com/analytics-vidhya/house-price-prediction-regression-with-tensorflow-keras-4fc49fae7123. [Accessed: 12- Sep- 2020].

[28] A. Géron and R. Demarest, *Hands-on machine learning with Scikit-Learn and TensorFlow*. Sebastopol (Clif.) [etc.]: O'Reilly, 2019, pp. 100.

[29] "tfa.losses.sigmoid_focal_crossentropy | TensorFlow Addons", *TensorFlow*, 2020. [Online]. Available: https://www.tensorflow.org/addons/api_docs/python/tfa/losses/sigmoid_focal_crossentropy. [Accessed: 12- Sep- 2020].

[30] G. Lemaitre, F. Nogueira, D. Victor and C. Aridas, "Welcome to imbalanced-learn documentation! — imbalanced-learn 0.5.0 documentation", *Imbalanced-learn.readthedocs.io*, 2020. [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/. [Accessed: 12- Sep- 2020].

[31] N. Chawla, K. Bowyer, L. Hall and W. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique", *Journal of Artificial Intelligence*

Research, vol. 16, pp. 321-357, 2002. Available: 10.1613/jair.953 [Accessed 12 September 2020].

[32] "Intel® Distribution of OpenVINO™ Toolkit", *Intel*, 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html. [Accessed: 12- Sep- 2020].

[33] "Amazon Web Services", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Amazon_Web_Services. [Accessed: 12- Sep- 2020].

[34] [14]"AWS Lambda – Serverless Compute - Amazon Web Services", *Amazon Web Services, Inc.*, 2020. [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 12- Sep- 2020].

[35] "AWS Free Tier", *Amazon Web Services, Inc.*, 2020. [Online]. Available: https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc. [Accessed: 12- Sep- 2020].

[36] [14]"Communication endpoint", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Communication_endpoint. [Accessed: 12- Sep- 2020].

[37] "AWS Lambda function handler in Python - AWS Lambda", *Docs.aws.amazon.com*, 2020. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html. [Accessed: 12- Sep- 2020].

[38] "What does an API gateway do?", *Red Hat*, 2020. [Online]. Available: https://www.redhat.com/en/topics/api/what-does-an-api-gateway-do#:~:text=An%20API%20gateway%20is%20an,and%20return%20the%20appropriate%20result. [Accessed: 12- Sep- 2020].

[39] "Global Infrastructure", *Amazon Web Services, Inc.*, 2020. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/. [Accessed: 12- Sep- 2020].

[40] M. Hasan, R. Toma, A. Nahid, M. Islam and J. Kim, "Electricity Theft Detection in Smart Grid Systems: A CNN-LSTM Based Approach", *Energies*, vol. 12, no. 17, p. 3310, 2019. Available: 10.3390/en12173310.

[41] S. Li, "Anomaly Detection for Dummies", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/anomaly-detection-for-dummies-15f148e559c1. [Accessed: 12- Sep- 2020].

[42] "2.7. Novelty and Outlier Detection — scikit-learn 0.23.2 documentation", *Scikit-learn.org*, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/outlier_detection.html. [Accessed: 12- Sep- 2020].

[43] "sklearn.utils.class_weight.compute_class_weight — scikit-learn 0.23.2 documentation", *Scikit-learn.org*, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html. [Accessed: 13- Sep- 2020].

[44] T. Lin, P. Goyal, R. Girshick, K. He and P. Dollar, "Focal Loss for Dense Object Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318-327, 2020. Available: https://arxiv.org/pdf/1708.02002.pdf. [Accessed 12 September 2020].

[45] 10]Y. LeCun, Y. Bengio and G. Hinton, "Deep learning", Nature, vol. 521, no. 7553, pp. 436-444, 2015. Available: 10.1038/nature14539 [Accessed 13 September 2020].

[46] M. Hasan, R. Toma, A. Nahid, M. Islam and J. Kim, "Electricity Theft Detection in Smart Grid Systems: A CNN-LSTM Based Approach", *Energies*, vol. 12, no. 17, p. 3310, 2019. Available: 10.3390/en12173310.

[47] S. Li, "Anomaly Detection for Dummies", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/anomaly-detection-for-dummies-15f148e559c1. [Accessed: 04- Oct- 2020].

[48] "2.7. Novelty and Outlier Detection — scikit-learn 0.23.2 documentation", *Scikit-learn.org*, 2020. [Online]. Available: https://scikitlearn.org/stable/modules/outlier_detection.html. [Accessed: 04- Oct- 2020].

[49] "Get Inferences for an Entire Dataset with Batch Transform - Amazon SageMaker", *Docs.aws.amazon.com*, 2020. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-batch.html. [Accessed: 04- Oct- 2020].