Periodic Time Series Data Analysis by Novel Machine Learning Methodologies

by

Haolong Zhang

Thesis submitted in partial
fulfillment of the requirements for the
Master of Computer Science degree

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Period length extraction is considered a challenge in many research fields. To solve this problem, different methods have been proposed across many applications. For instance, supply chain management is an area that can greatly benefit from precise periodic information. In addition, periodic information on physiological data can provide insights into individuals' health conditions, which is the motivation of this thesis.

The difficulty of period length extraction involves the varying noise levels among working environments. A system that performs well in one environment may not be accurate in another. In this work, we explore two machine learning approaches, each of which attempts to solve the problem at a different noise level. The first algorithm, the **p**eriod **c**lassification **a**lgorithm (PCA), utilizes historical labeled data as training material and classifies new instances. The PCA demonstrates robustness to both generated and natural noise. However, the training of the PCA is not economical if the data do not contain much noise.

The second algorithm, the **p**eriod **d**etection **a**lgorithm (PDA), is used when the noise level is not very high. It does not require historical data, but rather detects the period length directly from the data stream. The PDA cannot tolerate as much noise as the PCA; however, it is more efficient and simpler to deploy.

By investigating both algorithms on artificial and real-world datasets, we determined that they have advantages under different circumstances. In particular, the PDA outperforms the PCA when the system is noise-free, while it fails on real-world datasets, which usually contain a large amount of noise. In contrast, given that the training material is representative of test datasets, the PCA demonstrates high performance on both artificial and real-world datasets.

Acknowledgements

I would like to thank my family and all of my friends who supported me both physically and mentally. Additionally, I am grateful to my senior colleague, Haoye Lu, who gave me advice when I needed it most. Finally, I express my deepest appreciation toward Prof. Amiya Nayak. He provided me with guidance, support, advice, and ideas throughout my thesis work. I would not have been able to complete my program if it were not for him.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and objective

The analysis of periodic time series datasets (PTSDs) is relevant to many research fields. In addition, precisely classifying the period length of PTSDs is useful for industries, such as health care. A determined period length can help pharmaceutical companies decide the release time of their products (e.g., flu vaccines) [77], for which analyzing general public health data is helpful. Consumers and patients generally hope that there are sufficient resources to meet their demands. However, companies do not wish to have idle resources, which would increase the running budget. The ideal scenario is thus one in which the available resources match the demand with neither surplus nor shortage.

In the past several decades, several decision optimization algorithms have been proposed [14]. Through examining their approaches, we observed that the majority of supply chain optimization algorithms require predetermined period information. The vast quantity of data makes manual analysis infeasible; however, because the tasks are simple but tedious, they are suitable for artificial intelligence (AI) programs. The motivation of this thesis began with this problem. Currently, many AI applications are integrated into our

daily life, and we believe that it is possible to develop an AI agent that can free business managers, medical doctors, and customers from manually extracting period information.

1.2 Periodic time series dataset and period extraction

Trend prediction applications originally stemmed from extrapolation in statistics. Extrapolation is a process that uses known function values within some range to estimate values outside of that range [50]. In the case of PTSDs, because it is a periodic function in the time domain, the extrapolation becomes a prediction. Time series analysis is another classical method for making predictions. It assumes that future events depend on past events. The spectral theorem demonstrates that all time series can be divided into periodic and polynomial components [7]; that is, the two components can be analyzed independently of each other.

In this work, two design methodologies were tested for both real-world and generated PTSDs. According to the spectral theorem, generation methods produce two independent functions individually: periodic and polynomial. Real-world datasets were carefully selected that possessed sufficient periodic information with noise between each period. The thesis proposes two different machine learning algorithms for extracting period information. The first is a supervised algorithm called the **period classification algorithm (PCA)**. This algorithm requires prior knowledge of the potential length of the period along with manually labeled data to train. The second algorithm is an unsupervised algorithm called the **period detection algorithm (PDA)**. This algorithm was designed to perform fast processing without the learning phase; however, it tolerates less noise than the PCA.

1.3 Convolutional neural network and supervised learning

In 2009, Raina et al. [52] discovered that graphics processing units (GPUs) can significantly accelerate the training process of AI. Since then, this field has continued to advance. Krizhevsky et al. reported that the AlexNet model exhibits human-like performance on image classification [30]; however, its strength in period detection problems has not been tested.

According to the spectral theory of time series [7], any PTSD can be decomposed into periodic and polynomial components. Traditional methods, such as Fourier transformation, cannot find the period length if there is a polynomial component in the dataset. In this paper, we implement a period classification algorithm based on convolutional neural networks (CNNs), which can overcome the aforementioned problem.

The fundamental building blocks of CNNs are layers. More specifically, they are input layers, convolution layers, pooling layers, fully connected layers, and output layers. These layers can recognize both local features, such as edges and corners, and abstract features, such as animals and humans.

As humans, we can easily extract periodic features by viewing a function graph. More impressively, our brains can handle various types of noise that are independent of polynomial components. We believe that this is because human brains are well designed for analyzing visual patterns. A two-dimensional (2D) CNN is designed to simulate and surpass this intelligence; thus, we believe that CNNs can fulfill the task of extracting periodic features.

To use a 2D CNN to classify the period length, we must first preprocess the PTSDs. In particular, we plot the raw datasets on canvases of the same size, which can be achieved

by adding white pads surrounding the original images. We then partition the images into square graphs, followed by stacking them together. It should be noted that because the original images mostly consist of white pixels, the stacking operation saves GPU resources without concealing the periodic features.

1.4 Learning automata and unsupervised learning

Learning automata (LAs) were initially introduced by Tsetlin [69] and have been used in various domains and applications. LAs have generally been studied as a model that can maximize the reward probability without knowledge of the reward function. The reward function can be viewed as an abstract environment for a specific task. However, the environment can change, and the same query can result in a different outcome at different time steps.

Because the purpose of LAs is to maximize the reward probability in an unknown environment, the training of LAs does not require labels. This training strategy is called unsupervised learning. In the regime of LAs, during the training phases, the environment does not instruct the LA with correct labels. Instead, the environment will give penalty or reward to inform the LA how bad its action is. Based on the responses, the LA modifies its strategy to take actions in order to maximize the overall probability of getting a reward.

1.5 Main contribution

The main contribution of the thesis involves the period length extraction problem. The thesis proposes two algorithms, each of which has unique advantages under different circumstances.

The PCA takes advantage of historical data, using them to tolerate noise. This algorithm requires pre-labeled data, and raw data streams must be converted into images prior to period extraction. In contrast, the PDA can be applied directly to data streams. The only requirement for the PDA to perform well is that the noise level must not be very high.

A comparison of the two algorithms yields the following conclusions in this thesis. The PCA and the PDA take different machine learning approaches. Thus, they are not meant to outperform each other, but to function in their respective domains. This thesis suggests using the PCA when the working environment is noisy. When the environment is artificial or the noise level is low, the PDA is more efficient and economical.

1.6 Thesis organization

The remainder of the thesis is organized as follows. Chapter 2 reviews related work, including period detection methods and general machine learning techniques. Chapter 3 introduces the PCA based on the CNN approach. Then, Chapter 4 introduces the unsupervised learning PDA, which is based on LA. Finally, the thesis is concluded in Chapter 5.

Chapter 2

Related work

Periodic classification has been a major challenge in many research fields. Period information is not only valuable and profitable for corporations, but also lifesaving for individuals. Consequently, various methods have been proposed across many applications. Supply chain management, for example, is an area that can benefit significantly from precise periodic information. The reason is that many supply chain optimization methods are proposed based on a pre-determined period length [14, 47, 51, 72]. Period information on physiological data can also provide insight into individuals' health conditions. This chapter reviews existing period classification and detection methods.

The chapter is organized as follows. Section 2.1 reviews the Fourier transform, while Section 2.2 compares the differences between supervised and unsupervised machine learning algorithms. Section 2.3 introduces several machine learning classification algorithms, while Section 2.4 surveys the state-of-the-art performance achieved by deep learning methods. Lastly, two applications related to period extraction are discussed.

2.1 Fourier transform and Fourier series

Fourier transformation extracts frequency information from a function of time. For example, music or sound can be described by their volume and notes. In this case, volume represents the amplitude of the sound wave, while notes represent the wavelength. The shorter the wavelength, the higher the frequency.

The Fourier transform originated with Fourier series. Using Fourier series, a complex waveform can be represented by a sum of simple sine and cosine waves with variant parameters. The Fourier transform extends this concept to approach infinity. Generally, the Fourier transform of function f is written as $\hat{f}$. The formal definition is as follows:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx \tag{2.1}$$

A Fourier transform is used to represent a general, non-periodic function by a continuous superposition or integral of complex exponentials.

In contrast, a Fourier series is the expansion of a periodic function. For a periodic function $f(x)$ on interval $[-\pi, \pi]$, the Fourier series of $f(x)$ is given by

$$f(x) = \frac{1}{2} a_0 + \sum_{n=1}^{\infty} a_n cos(nx) + \sum_{n=1}^{\infty} b_n sin(nx), \tag{2.2}$$

where $a_n,\ b_n$ are referred to as Fourier coefficients:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) dx \tag{2.3}$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) cos(nx) dx \tag{2.4}$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) sin(nx) dx \tag{2.5}$$

In general, a Fourier transform is the limit of the Fourier series of a function with the period approaching infinity. The limits of integration then change from one period to $(-\infty, \infty)$.

Usually, the discrete Fourier transform (DFT) [23] is used to extract period information from a finite sequence. Unlike the Fourier transform, which works directly on the function definition, the DFT knows N individual data points. These data points are sampled with an equal time interval T, namely, $f[0], f[1], \cdots, f[k], \cdots, f[N-1]$. Then, because the integrand exists only at the data points, Eq 2.1 becomes the following:

$$F(\xi) = \int_0^{(N-1)} f(x) e^{-2\pi i x \xi} dt \tag{2.6}$$

$$= \sum_{k=0}^{N-1} f[k] e^{-\xi k T} \tag{2.7}$$

The Fourier transform and DFT have been widely used in fields such as single analysis and sound analysis. After the computation is completed, the period information can be easily obtained. However, if the original function $f(x)$ is unknown or sampled with noise, the resulting function may have multiple peaks, which would lead to confusion.

As discussed, using the Fourier transform and related approaches to obtain period information is powerful when the periodic functions are clearly defined and noise-free. However, when the target period is not the most dominating period with the highest Fourier coefficient, then the Fourier transform will have difficulty processing it. More importantly, if the data samples are noisy enough to offset the real distribution, or if the noise is also periodic, the Fourier transform will produce incorrect period information.

2.2 Supervised and unsupervised machine learning algorithms

The majority of practical machine learning problems are solved by supervised learning techniques. The reason is that supervised learning techniques are usually guided by human experience. Formally, supervised learning involves identifying an approximated function $\hat{f}$ that maps input variables (x) to an output variable (Y) through iteration. The resulting function is not necessarily generalized for all possible input cases; therefore, the goal is to minimize the prediction differences when new inputs arrive.

The approach is called supervised learning because the approximation process is performed in a supervised manner. The training process uses a pre-labeled dataset as a teacher that supervises the learning algorithm. The algorithm iterates through the dataset, and the learner makes predictions based on the features and is corrected by these labels. This process halts when the learner's performance is deemed acceptable. That is the training loss is lower than some certain threshold.

Supervised learning problems can also be categorized into classification and regression problems. The two categories can be determined by the label of the learning problem. If the label variable of the learning problem is discrete, then this problem is treated as a classification learning problem. For example, if we wish to train an agent that can classify computer images into human descriptions (e.g., shirt, dress, jeans, etc) after training, then the learning problem is a classification problem. Otherwise, if the dataset is labeled by continuous variables, then the learning problem is a regression problem. For instance, the unemployment rate is a continuous value between 0 and 1. Wu at el. [75] reported how to forecast this rate using a machine learning algorithm with web information.

In contrast to supervised learning algorithms, unsupervised learning algorithms ad-

dress problems that only have input data (X) and no corresponding labels. The principle is to generalize hidden structures or distributions in the data that may not be obvious to humans. A major difference between unsupervised and supervised learning algorithms is the presence of labels. Unsupervised algorithms have no guidance from labels but are also free from human bias.

There are two major categories of unsupervised learning algorithms: clustering and association algorithms. The goal of clustering algorithms is to reveal undiscovered groupings in data. For example, customers with similar ages tend to have similar grocery bills. In contrast, association algorithms look for relevance between data features. For instance, if someone buys a cooking guide, he or she is likely to buy fresh greens in the near future.

2.3 Machine learning algorithms

As mentioned in Section 2.2, there are two major categories of machine learning algorithms: supervised and unsupervised algorithms. Classification and regression are supervised learning approaches, while clustering and association are unsupervised learning approaches.

The purpose of classification is to learn from historical data and be able to classify new observations in the future. This can be a binary or multi-classification problem; that is, there can be two or more possible classes for each observation. For instance, a medical diagnosis that can result in either a positive or negative conclusion is a binary classification problem because there are only two classes. In contrast, classifying a picture of a random animal involves more than two possible classes.

A number of learning algorithms have been proposed, including the following:

- Probability-based (Bayesian network [19])
- Distance-based algorithms (k-nearest neighbor (KNN) [48])
- Tree-based (decision tree [57])
- Ensemble (random forest[11])
- Linear classifiers (logistic regression [15], support vector machine [11], neural network [20])

Different algorithms can outperform others depending on the task and the dataset. For instance, when each feature of the dataset is independent of other features, the Bayesian network prevails. However, the independence assumption can sometimes be difficult to verify because that training datasets are not usually collected by machine learning experts. As a result, other algorithms that were not built on this assumption may have superior performance. Several well-known machine learning algorithms are introduced below.

K-nearest neighbor: The KNN algorithm takes a set of labeled points from a hyperplane and simply memorizes them. When a new point arrives, the algorithm utilizes a distance measurement (e.g., city block distance, Euclidean distance) to select the k^{th} closest points from the memorized data points. The label of the new point is then predicted by the majority of the selected points, which means that each new point is labeled by peers that are closer to it. This algorithm is highly dependent on the training data. If the training data have clear clusterings with clear boundaries, then the trained model is very robust and reliable. However, when the number of features and the quantity of data points increase, this algorithm needs time to compute the distance between the new points and each training point. Thus, this algorithm is rarely applied on image datasets directly.

Decision tree: Decision tree algorithms tend to learn branching factors that divide the training data into less chaotic manner. The algorithms recursively search for rules that can increase the purity of each divided sub-ground. These algorithms generally function better with discrete data features because they have finite possible sub-grounds. At each recursive step, the data are divided into smaller subgroups, and the tree is developed incrementally. Once a stopping criterion is met, the resulting tree contains only decision nodes and leaf nodes, and new instances can be quickly classified by examining the decision nodes. Decision trees usually require a longer training time than a KNN; however, the resulting model is significantly lighter. However, both algorithms have overfitting tendencies, which signifies that the algorithms cannot detect bias from historical data points. Once the data are biased, the algorithm is likely to fall into a pit and produces incorrect predictions. For example, if a company wishes to use the CVs of its current employees to train a decision tree or KNN model, and currently males constitute the majority of employees, the model may ignore all female applicants, resulting in a biased model.

Random forest: Random forest, or random decision forest, is an algorithm based on decision tree algorithms. This algorithm is designed to counter the overfitting tendency of decision trees by growing several trees with different training materials. Then, in principle, each resulting tree should be overfitted to different aspects depending on how the training materials are segmented. Prediction is then performed by the voting of all grown trees. Random decision forests thus correct for decision trees' tendency to overfit to their training set.

Neural network: A neural network consists of units (neurons) that are arranged in layers and convert an input vector into an output one. Each unit takes an input vector, applies an often non-linear function to it, and then passes it to the next layer. The

networks are generally defined to be feed-forward; that is, a unit feeds its output to all units in the next layer, but there is no feedback to the previous layer. Weights are applied to signals passing from one unit to another, and these weights are tuned in the training phase to adapt a neural network to a particular problem. In recent years, scholars discovered that the training process of neural networks can be accelerated by GPUs. This allows scholars to train networks with more layers and more neurons. It was then demonstrated that neural networks with deeper layers tend to perform better on some tasks, but are more difficult to train. A family of new models and training algorithms has been developed around this core and continues to evolve. This family of algorithms is referred to as deep learning.

2.4 Deep learning and image classification

As discussed earlier, machine learning technology powers many aspects of modern society, such as web searches, content filtering on social networks, and recommendations on e-commerce websites, and it is becoming increasingly present in consumer products, such as cameras and smartphones. However, more complex tasks, such as identifying objects in images, transcribing speech into text, matching news items or products with users' interests, and selecting relevant search results, are usually performed by deep learning.

Traditional machine learning techniques lack the ability to process raw data, which have a large quantity of features. Constructing a machine learning system that can recognize image patterns usually requires careful engineering to achieve an ad hoc design. The design usually preprocesses raw pixel data into a suitable internal representation or feature vector that is then fed into a sub-learning system. The sub-learning system learns to classify or recognize image patterns in the input.

Deep learning methods follow the same path of representation learning, as they take

raw data and automatically produce representation maps through training. Importantly, the same process can be applied recursively, which then allows the learning system to learn representations of representations. Similarly, humans learn simple concepts as babies, while advanced concepts are learned later in life. Deep learning systems simulate this stacking non-linear transformation. Each layer enriches the complexity of the representation. From a global point of view, the entire deep learning system learns a complex mapping system from raw input data to their labels. For example, a deep system trained to classify visual objects uses its lower layers to detect simple geometric elements. Then, higher layers evaluate whether there are enough simple elements to form a human face. If the label of this input data is positive for containing a face, then the learning machine correctly classifies it. The key is that these behaviors are not engineered by humans, but are purely learned by a computer program with a general proposed model.

For computer scientists, image classification is an important field of computer vision. Formally, image classification can be defined as an automatic labeling algorithm that tags images with predefined class labels. Many real-world applications are related to this field, including self-driving vehicles [5, 25, 42], face IDs [28, 49, 62], facial expression recognition [1], image denoising [67] and similar image retrieval [59, 70, 79]. Such tasks generally involve a large quantity of data and real-time processing.

For humans, image classification is a fundamental skill for survival, and humans have been naturally selected to sharply process images. However, the logic of the vision sense is difficult to generalize, and difficulty arises when objects have high variability within the same class [10]. Traditionally, image classification tasks have been performed with a two-step approach. First, features are extracted using feature descriptors and are then used as inputs to a classifier. This approach is problematic when the selected descriptors are not useful for the classification task [32]. For instance, a cloud descriptor does not

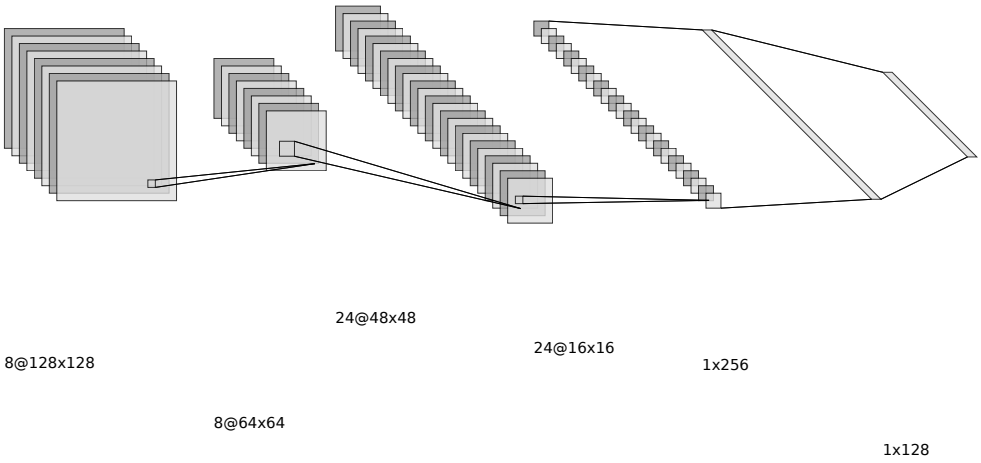

Figure 2.1: General structure of convolutional neural network. This instance has four convolutional layers and two fully connected layers. The last layer makes a prediction among 128 classes.

produce useful features for a self-driving vehicle. Arguably, a pedestrian descriptor is more relevant; however, pedestrians appear different in different clothing.

As other scholars [33, 61, 66], we believe that deep learning will succeed in the near future because it requires very little manual engineering and can thus take advantage of the increase computation resources and data. New learning algorithms and architectures that are currently being developed for deep neural networks will accelerate this progress.

2.5 Time series classification

In the last two decades, Time Series Classification (TSC) was treated as one of the most challenging problems in the field of data mining [18, 76]. In more recent years, the availability of real-world data has significantly increased [58]; hundreds of TSC algorithms have been proposed with different design philosophies [3]. As a result of lacking integrity, most TSC problems require some level of human involvement [31]. Meanwhile, if a classification problem takes ordered data, it can be transformed into a TSC problem [46]. TSC problems are not only challenging but also have high diversity. The related problems include but not limited to:

1. electronic health records [53],
2. human activity recognition [46, 71],
3. acoustic scene classification [45],
4. cyber-security [63].

Researchers have proposed many algorithms to achieve required accuracy level for variant tasks [3]. A classical method is to train a Kth nearest neighbor (KNN) classifier using a distance function [36]. Specifically, the Dynamic Time Warping (DTW)

distance function combined with a KNN classifier appears to be a very strong baseline [3]. Different distance functions have been tested empirically, which indicate that no single distance measure significantly outperforms DTW [36]. Researchers have also demonstrated that the ensemble of KNN classifiers using different distance measures outperforms an individual classifier. Therefore, recent advance is heavily focused on ensemble techniques [2, 4, 6, 13, 24, 26, 37, 56]. Most of these works outperform the KNN-DTW approach [3] and share one common technique, transforming the raw data into a new feature space (for instance, using shapelets transform [6] to encode raw data into the feature space defined by DTW [26]).

Researchers believe that the ensemble KNNs performing various distance measurements can outperform a system only using one distance measurement. This view motivated the development of an ensemble of 35 classifiers named COTE (Collective Of Transformation-based Ensembles) [2]. COTE ensembles not only different distance measurements but also different feature transformations. Then, Lines et al. [37, 38] upgraded COTE to HIVE-COTE using a Hierarchical Vote system. Recently, scholars believe the deep learning models can extract hierarchical features from raw data with very little human involvements. We believe it was a huge motivation for the recent utilization of deep learning models for TSC [73].

Chapter 3

Periodic Time Series Data Classification By Supervised Deep Convolutional neural network

This chapter introduces the PCA based on the CNN architecture. Briefly, PCA takes as input a PTSD with a set of possible periods (denoted L). After processing the dataset, the PCA predicts the period length by selecting the most probable period from L.

The chapter first introduces the preprocessing algorithm in Section 3.1. This algorithm converts the raw data samples to a red-green-blue (RGB) image. Then, Section 3.2 presents the proposed model in detail. Section 3.3 presents the performance measurements, while Section 3.4 provides the training policy along with problems during training.

3.1 Preprocessing algorithm

Because CNN models were originally designed for image classification problems [30], to use this model, we introduce an algorithm to convert raw PTSDs into their image repre-

sentations. We then describe the design of CNNs in detail. By combining the conversion algorithm with a CNN, we obtain the PCA. Finally, we discuss the assumptions of the PCA.

Because CNN architecture was originally designed for image classification tasks, Algorithm 1 is implemented as a preprocessing method. It takes as parameters the raw input datasets, the desired width, and the desired height.

The algorithm begins by plotting the data on a white canvas followed by connecting them by line segments. It should be noted that the images plotted from the two datasets may have different heights or widths because they may not have the same period length or amplitude. To make their dimensions identical, Algorithm 1 adds white padding around the images. At this stage, the pixels of $rawImage$ are mostly white, which implies a low density of information and thus damages the performance of the CNN. To fix this problem, Algorithm 1 segments $rawImage$ into $pieces$ with identical $desiredWidth$ and $desiredHeight$. Then, the stack process takes the sum of the pixel values modulo 255. Notice that RGB records the strengths of red, green and blue by numbers ranging from zero to 255. Thus, the module operation ensures the validity of the data.

Algorithm 1: PTSD conversion algorithm

Inputs : $rawdataset$, $desiredWidth$, $desiredHeight$
Output: $stackedImage$

1 plot $rawdataset$ to $rawImage$;
2 compute the closest multiple mul, where $mul \times desiredWidth \geq rawImage.width$;
3 put white surrounding padding on $rawImage$ to make the size equal to $(desiredHeight, desiredWidth \times mul)$;
4 segment $rawImage$ into small $pieces$ with size equal to $(desiredHeight, desiredWidth)$;
5 stack each $pieces$ on top of each other to generate $stackedImage$;
6 return $stackedImage$;

Although the curve from the raw image does not have intersections, the stacking

process may place segments of the curve across from each other. Pixels from these intersections create misleading information. However, the stacking process does not modify the true correlations between pixels everywhere. In addition, the white backgrounds do not provide useful information for CNN models. Replacing them boosts the information density and saves valuable GPU resources, which also facilitates the convergence of the CNN.

3.2 Deep convolutional model design

The design of the CNN model used in this thesis contains the following fundamental building blocks that are described below:

1. One input layer
2. Five convolution-pooling (conv-pooling) blocks
3. One flatten layer
4. Two fully connected layers
5. Output layers

The CNN model has one input layer that considers every pixel as input. The model processes images as 3D tensors, where the first two dimensions are the coordinates of the pixels, and the third dimension is the RGB value.

The following five mini-networks are combined with the convolutional layers, activation function, pooling layers, and dropout function. The ith mini-network is configured as follows:

1. Convolutional layer with 2^{4+i} kernels of size 3×1

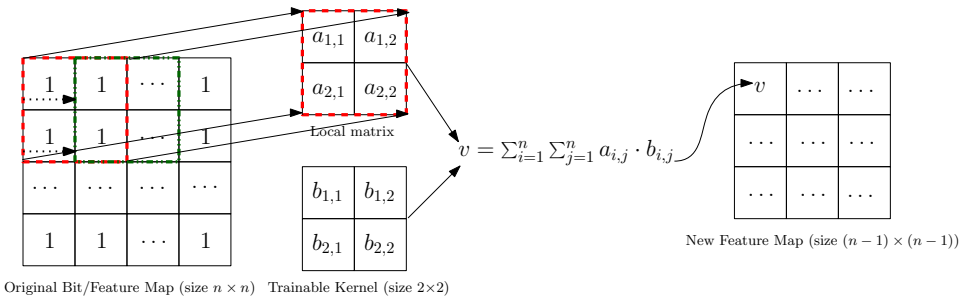

Figure 3.1: The 2D convolution operation scans feature maps from the previous layer row by row and column by column. Every iteration takes an equal-sized window of the kernel and applies the dot product operation. The results are collected as the new feature maps.

2. Convolutional layer with 2^{4+i} kernels of size 1×3

3. Leaky ReLu activation function with $\alpha = 0.1$

4. Pooling layer with $pool\_size = 2 \times 2$

5. Dropout layer with dropout rate = 0.25

Fig. 3.1 provides a conceptual example of 2D convolution operations. To construct each convolutional layer, two fine-tuning hyper-parameters are provided. The first parameter is a pair of integers that represent the size of a matrix. The matrices are called convolution kernels. The matrix in the center of Fig. 3.1 is an example of a convolution kernel with size 2×2. Each convolution kernel describes a local image feature. As illustrated in the figure, convolutional layers take the dot product of each 2×2 local matrix from the previous layers and the kernels. This operation generates a new feature map, which is the right matrix in Fig. 3.1. Each convolutional layer can generate more than one feature map, which corresponds to a kernel feature. The number of output feature maps is governed by the second parameter. These feature maps are then fed to the next layer.

In the configuration of this thesis, instead of one convolutional layer with a kernel size of 3×3, two convolutional layers are used. The first layer has a kernel size of 3×1, while the second one has a kernel size of 1×3. This architecture was originally introduced by the Inception V3 network [65], which has a similar approach as the classical CNN but has fewer parameters. In out case, convolutional layers with a kernel size of 3×3 have nine parameters per kernel, whereas Inception blocks require $3 + 3 = 6$ parameters only. Thus, less training time is required to optimize the design.

The leaky ReLu activation function [39] outputs the input value when it is greater than zero. When the input is less than zero, the leaky ReLu function outputs a constant ($alpha$) times input. This function brings nonlinearities to the CNN model without a high computational cost.

The leaky ReLu activation function is followed by a pooling layer that can remove irrelevant features without affecting relevant ones [60]. By stacking conv-pooling blocks, the CNN model obtains low-level features, such as an edge descriptor, by layers that are close to the input layers. High-level features, such as a healthy heart rate diagram or upward trending internet use, are learned by layers closer to the output. These high-level features are fed to the same structure of a multilayer perceptron (MLP) to perform classification tasks.

The dropout layers are the last operation of each mini-network. These layers remove some kernels with respect to the dropout rate. Thus, CNN models would not rely on particular kernels, which would result in poor out-of-sample performance [68].

Fig. 3.2 illustrates the network design, which follows the conv-pooling blocks. At the end of the conv-pooling process, the network is assumed to have sufficient independent features. Then, the flatten layer is placed to simplify future computation. This layer reshapes the stacked feature maps with a size of $(height, width, channel)$ to a one-

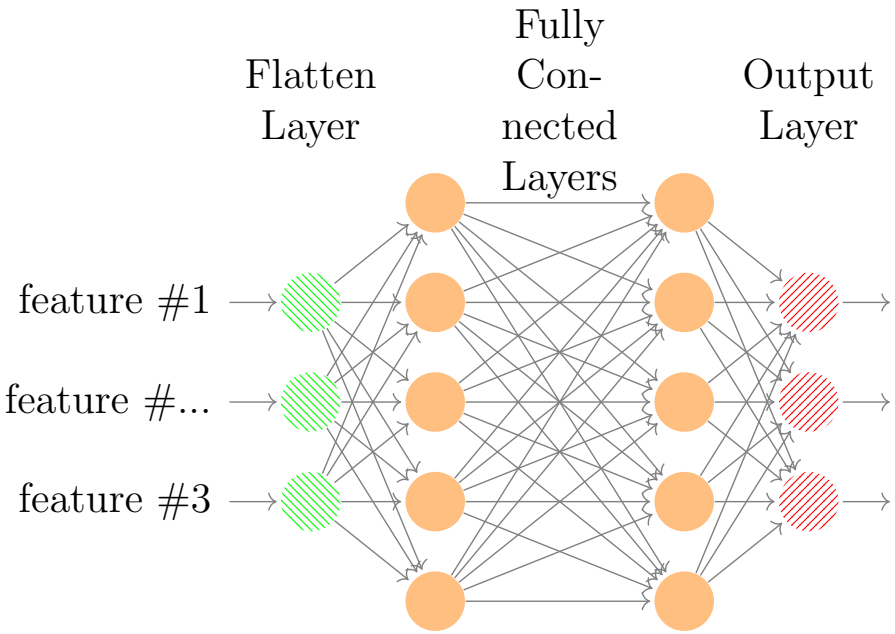

Figure 3.2: Structure of convolutional neural network design after conv-pooling blocks (also known as the multilayer perceptron). A different number of neurons is displayed here than in the actual design. Fully connected layers process input features and provide analysis results to the next layer. The output layer performs classification at the end.

dimensional vector with size ($height \times width \times channel$). The vector is then used as input to the fully connected layers. Each of two fully connected layers employs all of the following functions:

1. One leaky ReLu activation, where $\alpha = 0.1$
2. One dropout layer, where the dropout rate = 0.25

Each fully connected layer takes a hyper-parameter that indicates how many dense functions associated to the node of the next layer. Mathematically, these dense functions can be considered high-dimensional functions:

$$f_j : \mathbb{R}^{D_i} \to \mathbb{R},$$

where D_i is the number of inputs from the previous layer and $j = 1, 2, \ldots, n$ (where n is the number of nodes of the next layer). Function f_j is defined as follows:

$$f_j(features) = \sum_{i=1}^{D_i} (w_i \cdot feature_i + bias_i). \tag{3.1}$$

A fully connected layer is structured as a collection of functions defined by (3.1). Identical leaky ReLu activation functions and the dropout functions are also included in the fully connected layers.

Finally, the output layer produces the classifications. This layer is similar to the fully connected layers but replaces the dropout function by the softmax [41] function defined as (3.2). The output layer always contains a number of dense functions equal to the number of classes. The dense function outputs a vector with a size equal to the number of classes. After the leaky ReLu activation function, the softmax function converts the vector into a multinoulli distribution. Ultimately, the position with the largest value

indicates the classification result.

$$softmax(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \tag{3.2}$$

In this work, the main focus is to classify PTSDs by their period length. This implies that the target labels are deterministic integers in terms of time indices. As long as the sample size of one period is satisfied, the unit of the time indices is not a great concern. In addition, single integer labels are converted into vector labels. For example, instead of taking a period length equal to 2 or 3 as a target label, the label is written as $(0, 0, 1, 0)$ or $(0, 0, 0, 1)$. This representation is referred to as one-hot encoding [9]. If 2 is the target label from the previous example, the result of the CNN model is correct if the third element has the greatest value.

3.3 Performance measurement

The deep learning community normally focuses on two performance indicator: accuracy and loss. The accuracy of a model measures how well the classifier performs in general, and is computed by counting the correctly classified instances and dividing by the size of the test sets. In this case, CNN models do not have to output the exact labels. Assigning the highest probability to the target class is sufficient to be considered a correct classification.

The loss value measures the confidence level of the classifier regarding the result of a single instance. In general, a high loss value indicates a poor model performance, whereas a low loss value indicates a high performance. In this work, we use the loss value to describe and guide the optimization of our model.

Through the development of machine learning and deep learning, many loss functions

have been introduced and studied. The **m**ean **s**quared **e**rror (MSE) [34] is a loss function that is widely used in many classical machine learning models. It is formally defined in Definition 1. The MSE simply calculates the squared distance between each prediction and label, and then takes the average as the loss value. Because the distance is squared, the loss value stays positive for optimization. However, because we use softmax to compute the final prediction, the distance between the predictions and labels will not be greater than one, which is not very descriptive of the models' performance. Thus, MSE is not used in this work.

Definition 1 (Mean squared error)**.**

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.3}$$

The MSE is one of many loss functions that are not used in our model. Other loss functions, such as the mean absolute error [74] (MAE, a variant of MSE) and hinge [54], are not used because their function curves are not similar to an accuracy curve.

We utilize **cross entropy** [22] defined by (3.4). Cross entropy was introduced to quantify the difference between two probability distributions. When measuring the performance of a CNN model, we take the ground truth label as the target distribution y, and the model predicts another distribution, $\hat{y}$. The ground truth label should be definite; however, CNNs are generally not certain about their results. For example, a classification result (l_i) may be $< 0.3,\ 0.7 >$. This signifies that the model is 70% certain that the ith image is second class. A related ground truth label ($\hat{l}_i$) may be $< 1.0,\ 0.0 >$. It is clear that (3.4) produces a greater loss value for $< 0.3,\ 0.7 >$ than for another classification result, $< 0.9,\ 0.1 >$, which has the same ground truth.

Definition 2 (Cross entropy loss)**.**

$$f_{CEL}(y_i, \hat{y}_i) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log\ \hat{y}_i \tag{3.4}$$

3.4 Training algorithms

After measuring the performance of the CNN models, we require a strategy to adjust the convolutional features and coefficients of the density functions. As mentioned earlier, loss functions measure the performance of a deep model, where a lower loss value indicates a better performance. Predictions $\hat{y}_i$ are produced by $model(x_i, \theta)$, where θ represents the parameters. Therefore, we can treat loss functions $f_{loss}(y_i, \hat{y}_i)$ as objective functions $argmin_\theta\ f_{loss}(y_i, model(x_i, \theta))$. Thus, we require strategies to optimize θ in order to minimize the loss value.

Many strategies to lower loss functions have been introduced. However, due to deep models' complexity, the **gradient descent** family [27] has gained popularity, which has a guarantee that a local minimum of the loss function in terms of the model parameters can be reached. It does not require a clear understanding of the underlying function, which is one of the main reasons for its popularity.

The principle of gradient descent is to iteratively navigate on a hyperplane specified by a loss function. The navigation takes the steepest descent step at each iteration. To achieve this, for each trainable parameter $\theta_j \in \theta$ we update it as follows:

$$\theta^{(j)} = \theta^{(j)} - \eta \frac{\delta}{\delta \theta^{(j)}} f_{loss}(y_i, model(x_i, \theta)) \tag{3.5}$$

In (3.5), η represents the learning rate, which is used to control the speed of the parameter update. If the learning rate is too high, the model may fail to converge.

However, if the learning rate is too low, the model may be trapped at a local minimum point.

Unsurprisingly, the start location, or initialization, of θ plays an important role in a gradient descent-based algorithm. In other words, this optimizer is sensitive to weight initialization. Besides, the data sample x_i is as important as weight initialization. Outliers exist in datasets, and a random outlier may mislead the optimizer and result in poor model performance. Therefore, the original gradient descent is largely abandoned recently.

Mini-batch gradient descent [35] and **batch gradient descent** [55] were introduced to address the outlier problem. Instead of computing the gradient for one training instance at each iteration, a deep model processes a group of instances and computes the average of their gradient. If the group size is configured as the size of the training dataset, it is called batch gradient descent. Otherwise, if the batch size is less than the size of the dataset, it is called mini-batch gradient descent. Because the updating of parameters only occurs once per batch, this can be very slow and resource-consuming if the batch size is too large. However, with a fine-tuned configuration, this optimizer is less sensitive to outliers. Thus, mini-batch gradient descent is used in our training strategies.

Mini-batch gradient descent and batch gradient descent can be considered as the variants of gradient descent rather than its successors. Additional strategies have been proposed to address various challenges.

All gradient-based optimizers would not have an acceptable performance when the fastest descent direction on the loss hypersurface does not point to the global minimum. It is like a deep valley where the gradient is very steep at the cliffs. More importantly, when the weight is initialized on the cliffs, the gradient will be large and not aimed toward

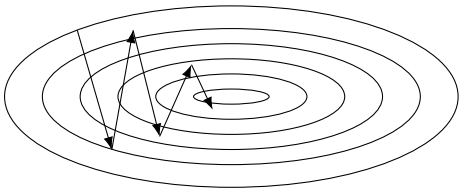

Figure 3.3: Gradient descent without momentum, configured with a high learning rate. Each arrow represents one training iteration, while the ellipses are contour plots of the loss surface. Because the optimizer is not affected by the previous training step, the model is not directed to the bottom of the valley, but to the other cliff.

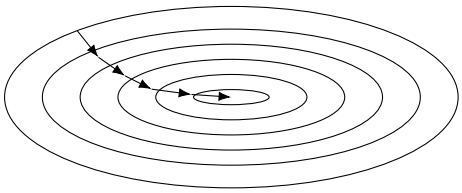

Figure 3.4: Gradient descent without momentum, configured with a low learning rate. In comparison with Fig. 3.3, arrows are shorter because of a lower learning rate. The optimizer takes a smaller step at each training iteration, and the model thus smoothly converges to the local minimum.

the local minimum. Fig. 3.3 demonstrates this situation. This situation is problematic because the gradient is too large, and we will land on another cliff. However, the case demonstrated in Fig. 3.3 is not severe, and the model converges to a local minimum after six training iterations. If the situation is severe enough, the model can fail to converge. We can bypass this problem with an extremely low learning rate. However, this requires additional training steps, convergence time, and training data. Fig. 3.4 demonstrates what can happen on the same loss surface with a lower learning rate. For this specific example, Fig. 3.4 does not require extra resources than Fig. 3.3. However, loss surfaces can be more complex. At the same time, if the model is initialized at a position far from the local minimum, six learning iterations may not be sufficient to achieve the same result.

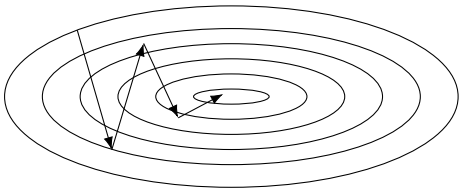

Figure 3.5: Gradient descent with momentum, configured with a high learning rate. Starting from the second iteration (left to right, arrows), the direction of optimization is pushed toward the local minimum. In comparison with Fig. 3.3, the optimizer takes one less step to reach the local minimum.

Momentum [64] is a method used to accelerate the convergence speed. The method achieves this by taking a fraction γ of the previous gradient into account. Equation (3.6) defines the momentum term that will be used in parameter updating. The parameter update rule is defined by (3.7). Comparing (3.7) with (3.5) reveals that the only difference is the momentum term, γv_{t-1}. The optimizer thus considers not only the current location on the loss surface, but also how it arrived there. This is similar to pushing a ball down a hill, where gravity is constantly and cumulatively affecting the direction and velocity of the ball. Thus, using momentum results in faster convergence and requires fewer training samples. Fig. 3.5 presents an example of optimizing the model in Fig. 3.3 with momentum. Differences appear after the first iteration. Due to the momentum term, the direction of optimization is pushed toward the local minimum.

$$v_t = \gamma v_{t-1} + \eta \frac{\delta}{\delta \theta^{(j)}} f_{loss}(y_i, model(x_i, \theta)) \tag{3.6}$$

$$\theta^{(j)} = \theta^{(j)} - v_t \tag{3.7}$$

In general, each learning iteration takes a batch of datasets, and deep models are usually trained on thousands of batches. Therefore, faster convergence can potentially

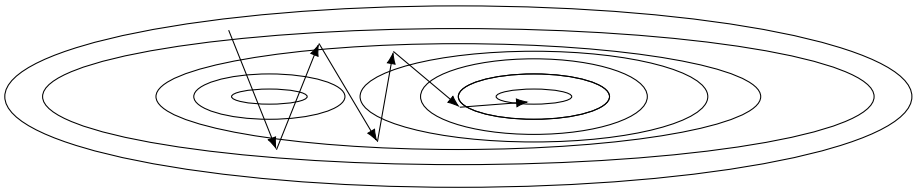

Figure 3.6: Gradient descent on a complex loss surface with momentum, configured with a high learning rate. The model is initialized on a surface with two local minima. The closer local minimum is shallower than the farther one. With a high learning rate, the model can avoid the shallow local minimum and converge to a deeper local minimum.

reduce the dependence level of high-quality data. In this example (Fig. 3.5 and Fig. 3.3), the momentum optimizer requires five iterations to reach the bottom of the valley. The original optimizer requires six iterations. This difference in the iteration number can play a crucial role when training on a small dataset. If we are required to use a small learning rate, then the model may be stuck at a shallow minimum. However, large updates can jump over small pits.

Fig. 3.6 presents a situation in which the model is initialized close to two local minima. The local minimum on the left side is shallower than the one on the right. If the learning rate is as low as in Fig. 3.4, the optimizer will be trapped at the left local minimum. Fig. 3.7 demonstrate the behavior, where the training process ends at the shallow local minimum point. However, with a higher learning rate, the optimizer can avoid the left local minimum and converge to a deeper one on the right.

In addition to the deep valley problem discussed earlier, the parameter update rate is another major challenge for training a deep model. A problem occurs when some parameters are overtrained while some are underemphasized. Gradient descent involves simply taking a partial derivative with respect to an individual parameter, and some parameters receive a larger gradient signal than others. Through learning iterations, the deep model is repeatedly trained to map the training data to the training target. This

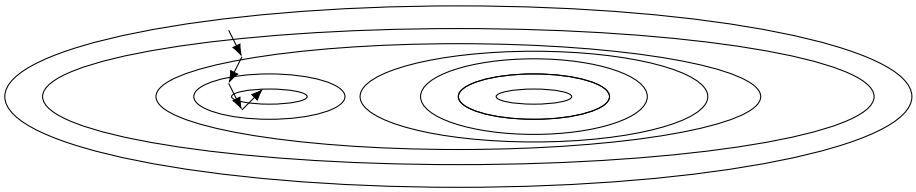

Figure 3.7: Gradient descent on a complex loss surface with momentum, configured with a low learning rate. In contrast to the case in Fig. 3.6, the model is trapped by the shallow minimum due to the low learning rate.

signifies that some parameters may be overtrained while others are barely affected by the optimizer.

Adagrad [16] is another gradient-based optimization algorithm that tracks the update rate of each parameter. Using this information, Adagrad performs large updates on parameters that are less well trained and small updates on others. Dean et al. [12] reported that Adagrad greatly improved the robustness of deep models. For PTSD classification, different waveforms may not occur as frequently as others. Thus, an adaptive learning rate is helpful. Unlike the optimizers we have discussed, Adagrad assigns an independent learning rate to each parameter $\theta^{(j)}$. To do so, Adagrad first estimates the gradient in each training iteration t:

$$g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta f_{loss}(y_i, model(x_i, \theta)) \tag{3.8}$$

The key of this algorithm is hidden in matrix G_t, as defined by (3.9). After performing matrix multiplication between the current gradient vector and its transpose, we obtain a matrix with size $j \times j$. G_t is the sum of the product of gradients until training iteration t, and is defined as follows:

$$G_t = \sum_{\gamma}^{t} g_\gamma g_\gamma^\top \tag{3.9}$$

Then, the diagonal of G_t is obtained and summed with a number close to zero, ϵ. The square root of the summation is used as the divisor of the learning rate η. The term ϵ helps to avoid division by zero. In addition, the individual parameter $\theta^{(j)}$ has its own learning rate decay. The greater the gradient it receives, the more its learning rate decays at that iteration. Formally, it can be defined as (3.10):

$$\begin{bmatrix} \theta_{t+1}^{(1)} \\ \theta_{t+1}^{(2)} \\ \vdots \\ \theta_{t+1}^{(j)} \\ \vdots \\ \theta_{t+1}^{(m)} \end{bmatrix} = \begin{bmatrix} \theta_{t}^{(1)} \\ \theta_{t}^{(2)} \\ \vdots \\ \theta_{t}^{(j)} \\ \vdots \\ \theta_{t}^{(m)} \end{bmatrix} - \eta \left(\begin{bmatrix} \epsilon & 0 & 0 & \cdots & 0 \\ 0 & \epsilon & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \epsilon & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \epsilon \end{bmatrix} + \begin{bmatrix} G_i^{(1,1)} & 0 & 0 & \cdots & 0 \\ 0 & G_i^{(1,1)} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & G_i^{(1,1)} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & G_i^{(1,1)} \end{bmatrix} \right)^{\frac{1}{2}} \cdot \begin{bmatrix} g_t^{(1)} \\ g_t^{(1)} \\ \vdots \\ g_t^{(j)} \\ \vdots \\ g_t^{(m)} \end{bmatrix} \tag{3.10}$$

For simplicity, (3.10) can be written as (3.11).

$$\begin{bmatrix}\theta_{t+1}^{(1)}\\ \theta_{t+1}^{(2)}\\ \vdots\\ \theta_{t+1}^{(j)}\\ \vdots\\ \theta_{t+1}^{(m)}\end{bmatrix} = \begin{bmatrix}\theta_{t}^{(1)}\\ \theta_{t}^{(2)}\\ \vdots\\ \theta_{t}^{(j)}\\ \vdots\\ \theta_{t}^{(m)}\end{bmatrix} - \eta \left(\begin{bmatrix}\epsilon & 0 & 0 & \cdots & 0\\ 0 & \epsilon & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & \cdots & \epsilon & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & \epsilon\end{bmatrix} + \begin{bmatrix}G_i^{(1,1)} & 0 & 0 & \cdots & 0\\ 0 & G_i^{(2,2)} & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & \cdots & G_i^{(j,j)} & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & G_i^{(m,m)}\end{bmatrix} \right)^{\frac{1}{2}} \cdot \begin{bmatrix}g_t^{(1)}\\ g_t^{(1)}\\ \vdots\\ g_t^{(j)}\\ \vdots\\ g_t^{(m)}\end{bmatrix}$$

$$\begin{bmatrix}\theta_{t+1}^{(1)}\\ \theta_{t+1}^{(2)}\\ \vdots\\ \theta_{t+1}^{(j)}\\ \vdots\\ \theta_{t+1}^{(m)}\end{bmatrix} = \begin{bmatrix}\theta_{t}^{(1)}\\ \theta_{t}^{(2)}\\ \vdots\\ \theta_{t}^{(j)}\\ \vdots\\ \theta_{t}^{(m)}\end{bmatrix} - \eta \left(\begin{bmatrix}\frac{1}{\sqrt{\epsilon+G_i^{(1,1)}}} & 0 & 0 & \cdots & 0\\ 0 & \frac{1}{\sqrt{\epsilon+G_i^{(2,2)}}} & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{1}{\sqrt{\epsilon+G_i^{(j,j)}}} & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & \frac{1}{\sqrt{\epsilon+G_i^{(m,m)}}}\end{bmatrix} \right) \cdot \begin{bmatrix}g_t^{(1)}\\ g_t^{(1)}\\ \vdots\\ g_t^{(j)}\\ \vdots\\ g_t^{(m)}\end{bmatrix}$$

$$\begin{bmatrix}\theta_{t+1}^{(1)}\\ \theta_{t+1}^{(2)}\\ \vdots\\ \theta_{t+1}^{(j)}\\ \vdots\\ \theta_{t+1}^{(m)}\end{bmatrix} = \begin{bmatrix}\theta_{t}^{(1)}\\ \theta_{t}^{(2)}\\ \vdots\\ \theta_{t}^{(j)}\\ \vdots\\ \theta_{t}^{(m)}\end{bmatrix} - \begin{bmatrix}\frac{\eta}{\sqrt{\epsilon+G_i^{(1,1)}}} & 0 & 0 & \cdots & 0\\ 0 & \frac{\eta}{\sqrt{\epsilon+G_i^{(2,2)}}} & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{\eta}{\sqrt{\epsilon+G_i^{(j,j)}}} & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & \frac{\eta}{\sqrt{\epsilon+G_i^{(m,m)}}}\end{bmatrix} \cdot \begin{bmatrix}g_t^{(1)}\\ g_t^{(1)}\\ \vdots\\ g_t^{(j)}\\ \vdots\\ g_t^{(m)}\end{bmatrix}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\epsilon I + diag(G_t)}} \cdot g_t$$

However, we did not use these optimizers directly. Instead, another optimization algorithm called **Adam** [29] was utilized to train our deep model. Adam stands for

adaptive moment estimation and has been reported as state-of-the-art among many deep learning algorithms. Adam combines the innovation of Adagrad and Momentum into one optimizer. It takes two hyperparameters, β_1 for moment decay and β_2 for learning rate decay. Specifically, Adam calculates the following:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.11}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{3.12}$$

where m_t and v_t are estimates of the first moment and second moment of the gradients, respectively. If $m_0,\ v_0$ are initialized as vectors of zeros, and $\beta_1,\ \beta_2$ are close to 1, the authors of Adam observed that $m_0,\ v_0$ are biased toward zero. Thus, instead of taking $m_0,\ v_0$ directly, the bias-corrected first and second moment estimates are computed as

$$\hat{m}_t = \frac{m_t}{1 - \beta_1} \tag{3.13}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2} \tag{3.14}$$

The bias-corrected terms are then used to update the parameters as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{3.15}$$

The authors demonstrated state-of-the-art performance when $\beta = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. Furthermore, cross entropy loss provides the necessary information for Adam to optimize CNN models. All experiments mentioned in Sections 3.6 and 3.8 were trained by the Adam algorithm with the parameter configuration proposed by the authors of Adam.

During the training process, the CNN model randomly takes several instances from

the training set and computes the mean of their loss. These instances form a training batch and are not reused before the end of one training iteration. The Adam algorithm then uses this information to minimize the cross entropy loss. This process is repeated until every instance has been selected and one training epoch is completed.

To apply the CNN design to PTSD classification problems, there are assumptions that must be satisfied, such as the following:

1. The amplitude of the PTSD must be larger than the white noise. Otherwise, the real trend may haven been overwritten during the sampling phase.

2. The training datasets must have a strong correlation between the sample data and target label. If this condition is not satisfied, the trained model may perform well during the training process; however, this does not guarantee that the model is generalized effectively at the evaluation stage, as reported by Zhang et al. [80].

3. The content of the training sets must cover all possible features with respect to the difference between periods. For example, if the model is trained on sample datasets that only contain PTSDs of a period length of 2, the model is capable of minimizing the training error. However, it would not perform well on evaluation datasets that only contain PTSDs of a period length of 7.

The above conditions are necessary but not sufficient. In other words, there are additional factors that may affect the overall performance; however, they are not discussed in this thesis.

3.5 Periodic time series dataset generation

Both the training and testing datasets are generated by Algorithm 2 and Algorithm 3. The only difference between the two algorithms is the noise sampling function. Algo-

rithm 2 uses a random polynomial function while Algorithm 3 uses the Gaussian noise. Ninety percent of the datasets was used for training, while the remaining 10% are used for testing. The process took five inputs: *periodLength*, *window*, *sampleSize*, $polyDegree$, and $polyVariance$.

The parameter *periodLength* indicates how long a single period lasts in terms of the time index. *window* is the length of the time slice that multiple periods can occupy, while *sampleSize* indicates how many data points are sampled for a single period.

At the beginning of the generation, spaced time indices between zero and *periodLength* were computed. We randomly sampled Y_i from a standard normal distribution, and copied Y and extended X until the window could not fit another period.

Algorithm 2 takes two parameters to control the random generation of polynomial components. The first parameter is the maximum degree, while the second parameter is the variance of the coefficients. The degree of each polynomial component is an integer that is randomly sampled from a uniform distribution between zero and the maximum degree. Once the degree is defined, the coefficients are sampled from a standard normal distribution and then multiplied by the variance. This operation is performed in line 16 of Algorithm 2. Every PTSD thus has a newly generated polynomial component that completes the generation of one sample. On the other hand, Algorithm 3 utilizes the zero mean with different standard deviation σ Gaussian distribution as its noise function.

3.6 Training and testing PCA using generated dataset

The core problem in this thesis is to classify two or more PTSDs by their period length. In terms of hardware, two Gigabyte GeForce GTX 1080 graphics cards were utilized to accelerate the training and testing of deep models. The deep models were implemented by Keras version 2.2.2 using the TensorFlow version 1.10.0 backend. These experiments

Algorithm 2: Periodic time series data generation with polynomial noise

Inputs : *periodLength*,*window*,*sampleSize*, *polyDegree*, *polyVariance*
Output: array of time index and corresponding data

$X,\ Y_p,\ Y$:= empty array;
for $i = 0;\ i < samples;\ i = i + 1$ **do**
    Put $(periodLength/samples \times i)$ at the end of X;
    Put the value sampled from the standard normal distribution at the end of Y_p
end for
$repeats = \text{floor}(window/periodLength)$;
$nextPeriod$:= copy X;
for $i = 0;\ i < repeats;\ i = i + 1$ **do**
    x_{end} := the last element of X;
    increase every element of $nextPeriod$ by x_{end};
    $X := X$ concatenate $nextPeriod$;
    $Y := Y$ concatenate Y_p;
end for
$polyFunc := randPoly(polyDegree,\ polyVariance)$;
for $i = 0;\ i < size(X);\ i = i + 1$ **do**
    $Y[i] += polyFunc(X[i])$
end for
return $X,\ Y$;

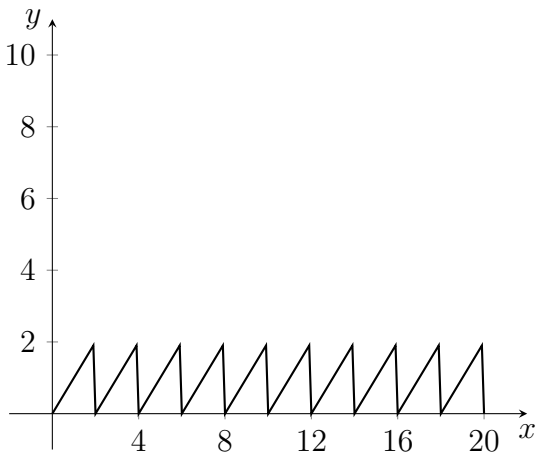

Figure 3.8: The periodic function $g(x) = 0.7 \times mod(x, 2)$ added with no noise.

Algorithm 3: Periodic time series data generation with gaussian noise

Inputs : *periodLength*,*window*,*sampleSize*, *gaussianSigma*
Output: array of time index and corresponding data

$X,\ Y_p,\ Y :=$ empty array;
for $i = 0;\ i < samples;\ i = i + 1$ **do**
  Put $(periodLength/samples \times i)$ at the end of X;
  Put the value sampled from the standard normal distribution at the end of Y_p
end for
$repeats = \text{floor}(window/periodLength)$;
$nextPeriod :=$ copy X;
for $i = 0;\ i < repeats;\ i = i + 1$ **do**
  $x_{end} :=$ the last element of X;
  increase every element of $nextPeriod$ by x_{end};
  $X := X$ concatenate $nextPeriod$;
  $Y := Y$ concatenate Y_p;
end for
for $i = 0;\ i < size(X);\ i = i + 1$ **do**
  $Y[i] += gaussianNoise(\mu = 0, \sigma = gaussianSigma)$
end for
return $X,\ Y$;

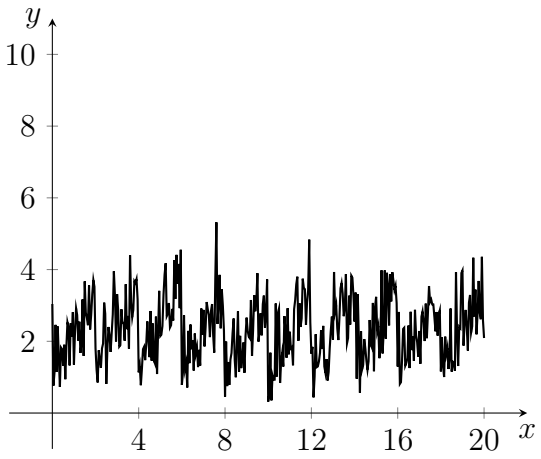

Figure 3.9: The periodic function $g(x) = 0.7 \times mod(x, 2)$ added to gaussian noise configured with $\mu = 0$ and $\sigma = 1$.

were aimed at determining the threshold of the PCA. The experiments had two possible ground truth labels: 2 and 7, respectively. Each ground truth label had $2,000$ images associated with it. Thus, in total, there were $4,000$ images generated, where $3,600$ (90%) were for training and 400 (10%) were for testing. We did not use the same image multiple times during the training process. That is, we first attempted to determine the maximum Gaussian σ for which the PCA could provide a correct answer. The Gaussian σ varied from zero to ten. Every time we increase the Gaussian σ configuration, Algorithm 3 is executed again. Fig 3.8 and Fig 3.9 present two raw PTSD generated by Algorithm 3. However, PTSD plotted on Fig 3.8 is not padded with any noise while Fig 3.9 is padded with Gaussian noise (with $\mu = 0$ and $\sigma = 1$). Besides, PCA is deployed on the same generated dataset for ten times. Fig 3.10 demonstrates the averaged result from each run. The results show that PCA managed to stay above 90 percent accuracy. This plot shows the PCA can tolerate strong Gaussian noise, which is the motivation for the next set of experiments.

For next set of experiments use polynomial noise functions. Our goal here is to determine the maximum polynomial degree and coefficient variance for which the PCA could provide a correct answer. The following two lists provide the values of the maximum polynomial degree and the variance of the polynomial coefficient, respectively, for implementing the experiments:

1. maximum polynomial degree {0, 1, 2, 3, 4}

2. variance of polynomial coefficient {0.5, 1, 1.5, 2, 3, 4}

Each combination was performed 10 times to obtain a general performance measurement. For every new experiment, images were randomly generated by the methods introduced in Section 3.5. Thus, the results were not biased by a random sample or by random deep model initialization.

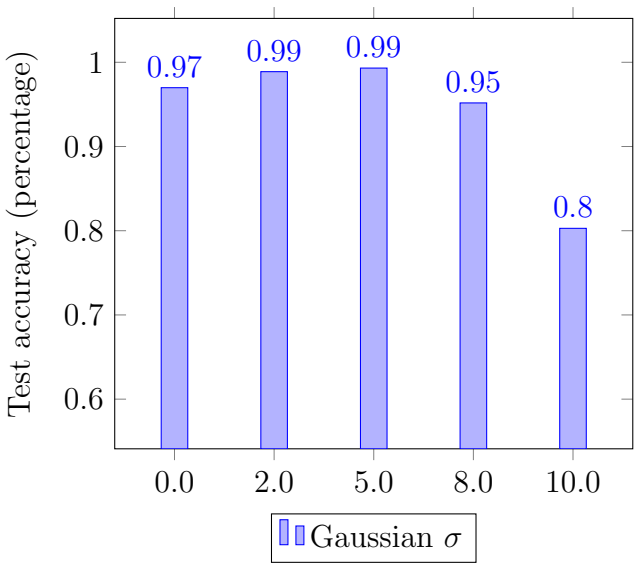

Figure 3.10: Accuracy result of period classification algorithm (PCA) classifying 400 unseen test images after being trained on 3,600 images. Period 2 and Period 7 instances are equally distributed among both training and testing sets. The test cases were padded with different Gaussian noise function. For the different Gaussian σ configurations, the PCA performed stably with high accuracy.

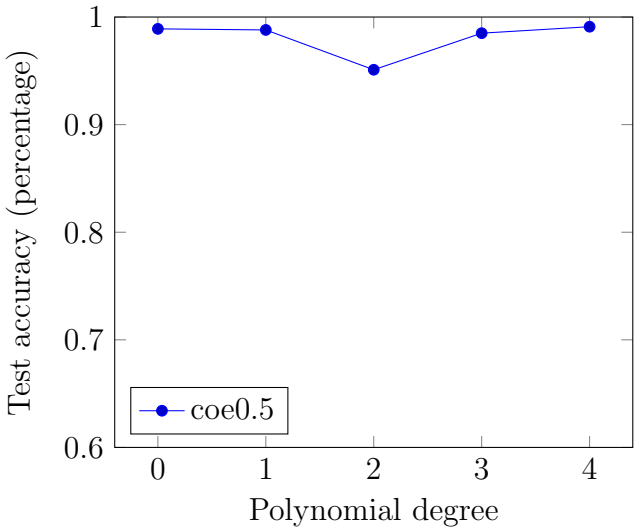

Figure 3.11: Accuracy result of period classification algorithm (PCA) classifying 400 unseen test images after being trained on 3,600 images. Period 2 and Period 7 instances are equally distributed among both training and testing sets. The test case was configured with a polynomial coefficient variance of 0.5. For the different maximum polynomial degrees, the PCA performed stably with high accuracy.

Due to limitations in computational power, we implemented the experiments by setting $desiredWidth = desiredHeight = 250$ in the PTSD conversion algorithm (see Algorithm 1). We can set the values higher if extra computing resources are avaliable. Interestingly, a higher resolution provides more periodic features when polynomial components dominate. Figs. 3.8–3.13 store the results of the first experimental set. The x-axis indicates the degree of the polynomial components, while the y-axis indicates the average out-of-sample accuracy from 10 identical experiments. The lines have different coefficient variances, as mentioned earlier.

Fig. 3.11 demonstrates how the PCA performed in the easiest experimental configuration, in which the polynomial coefficient variance was equal to 0.5. In this experiment, the PCA performed well for different polynomial degrees. The plot reveals that when the polynomial coefficients are low, higher polynomial degrees are not more effective than

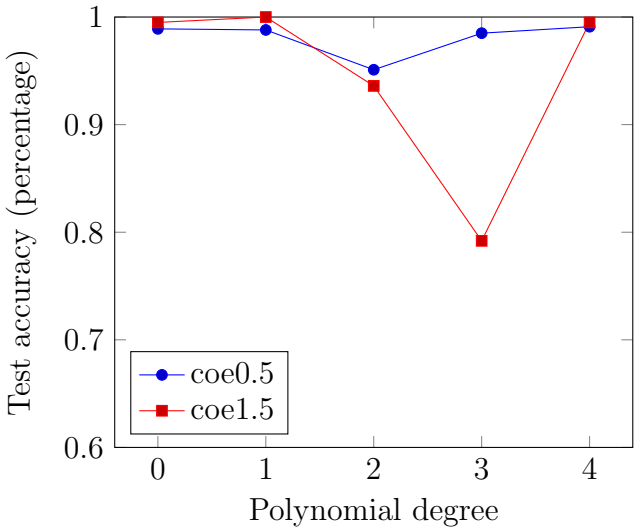

Figure 3.12: The same number of training and testing images were generated with a polynomial coefficient variance of 1.5. This is the dataset for which the period classification algorithm displayed the poorest performance.

lower polynomial degrees. Experimentally, this demonstrates that the PCA is capable of extracting different waveforms and counting the number of repeating graphic patterns.

Fig. 3.12 plots the results when the polynomial coefficient variance is 1.5. The plot indicates that the PCA performed relatively the same when the polynomial degrees were lower than 3. However, the PCA suffered greatly when the degree was equal to 3. We believe that this was because the polynomial components blurred the periodic features. However, the performance of the PCA returned to above 90% when the degree increased. We believe that this was because the data generator sampled coefficients from a normal distribution.

However, some waveforms became unrecognizable not with increasing polynomial noise, but with decreasing polynomial noise. A sawtooth wave is a typical example in which the amplitude increases within each period. Both Figs. 3.13 and 3.14 are examples of the same sawtooth wave ($g(x) = 0.7 \times mod(x, 2)$) with different polynomial noise.

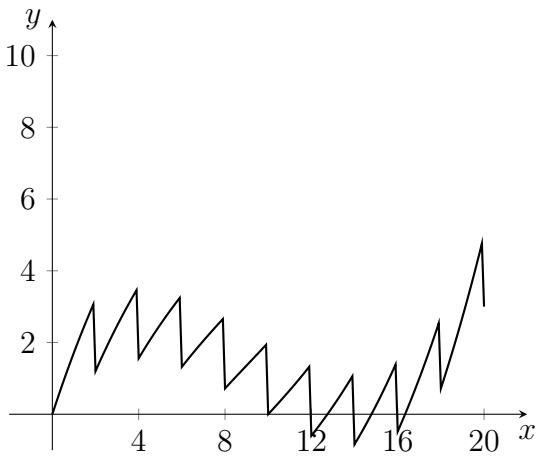

Figure 3.13: Periodic function $g(x) = 0.7 \times mod(x, 2)$ added to a polynomial noise function $h(x) = 0.005(x)(x - 10)(x - 17)$. Periodic features are still recognizable when the polynomial degree is 3.

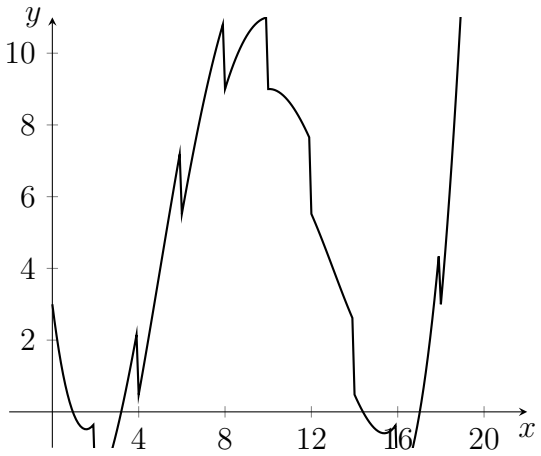

Figure 3.14: Periodic function $g(x) = 0.7 \times mod(x, 2)$ added to a polynomial noise function $h(x) = 0.005(x)(x-18)(x-13)(x-5)$. The periodic feature is less recognizable than the waveform from Fig. 3.13.

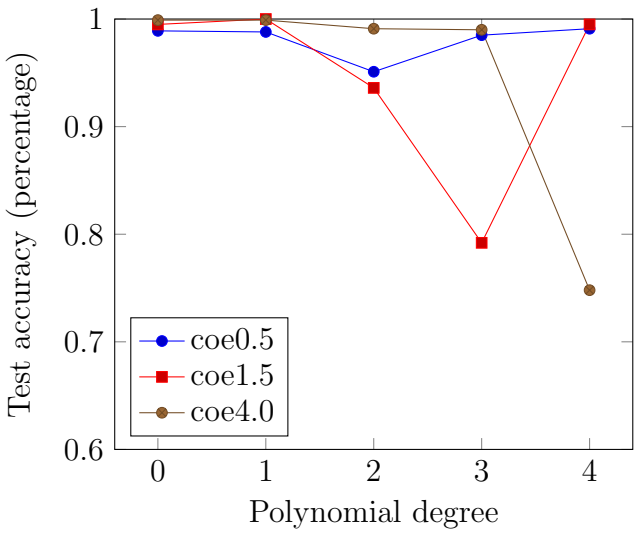

Figure 3.15: To further test the capability of the period classification algorithm, we increased the polynomial coefficient variance to 4.0. The model performed relatively well with a low degree of polynomial noise. However, the performance decreased sharply when the degree rose to 4.

Fig. 3.13 has noise polynomial $h(x) = 0.005(x)(x-10)(x-17)$. When $4 < x < 12$, the trend of the waveform decreases, and the sawtooth is evident within the interval. The sawtooth is still recognizable when $x < 4$. However, it fades when x surpasses 13.

Fig. 3.13 plots a noisy polynomial with a degree of 3. Its periodic feature is not yet blurred. However, Fig. 3.14 demonstrates what can occur when the polynomial degree continues to increase. The coefficient variance of both waveforms are at the same level. In comparison with Fig. 3.13, the sawtooth is unrecognizable across the domain.

Lastly, Fig. 3.16 demonstrates that the PCA remained above 90% test accuracy when the polynomial degrees were less than 3. Once the polynomial degrees reach 3, the performance for some test cases decreased accordingly. This observation is expected, as all training and testing data were plotted to equal-sized images. As mentioned earlier, all PTSDs were generated with period components that had the same amplitude. Therefore, PTSDs with higher degrees were vertically composed, and periodic features

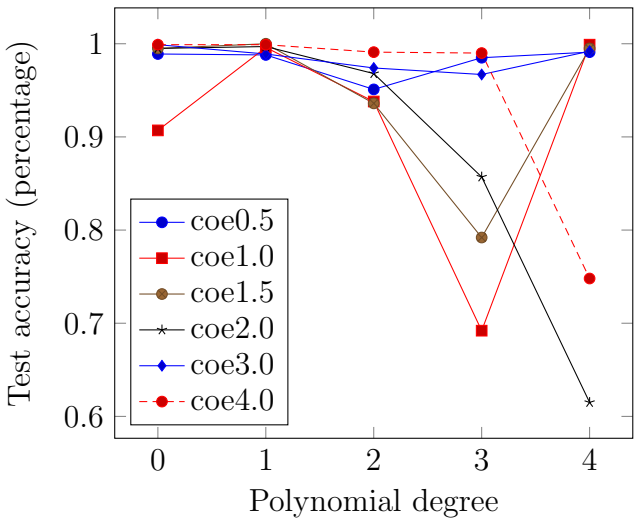

Figure 3.16: Test using 400 images equally distributed between period 2 and period 7 classes. The CNN was trained using $1,800$ images for both classes.

vanished. Furthermore, two coefficient test cases rose above 90% again, and two others decreased when the degree was equal to 4. In addition, the lower bound of the accuracy decreased even further, which demonstrates that the generation of datasets became more random. Therefore, it was more likely for polynomial components to become dominant. In addition, these plots did not contain enough periodic information for the PCA to extract. The waves looked more like curves than waves. However, this problem was not present in the selected real-world datasets. Waveforms across periods looked similar to others.

3.7 Real-world datasets

As mentioned, Algorithm 2 provides datasets that have balanced classes but different noise levels. The experiments demonstrated the PCA's capacity for binary classification tasks. For a real-world dataset, we used a different approach.

The PTSD generation algorithm provides all possible situations to fully test the PCA. However, the real-world datasets described below were collected in addition to the generated datasets. To simulate the working conditions of the PCA, these real-world datasets are more suitable than generated ones. Most importantly, they provide more insights into the performance of the PCA in a real-world setting.

Theoretically, Algorithm 2 can generate all possible noise on the amplitude axis. However, there are other aspects that can affect the performance, such as a phase shift or inconsistent sample rate caused by internet delay. Thus, we are still interested in how PCA performs on real-world datasets. After careful examination, we selected the following datasets:

1. Historical hourly weather data [21] (HHWD, 12,960 images, 5 classes)
2. Weather Madrid 1997-2015 data [8] (LEMD, 1,539 images, 18 classes)
3. Entsoe hourly energy consumption [17] (ENTSOE, 21,240 images, 10 classes)
4. PJM hourly energy consumption [44] (PJMHEC, 96,135 images, 5 classes)
5. Rain in Australia [78] (WeatherAUS, 2,536 images, 8 classes)

We did not use the entire datasets above because some fields did not contain periodic features. Some fields from the HHWD dataset, such as *windDirection*, appeared completely random across the time index.

Fields selected from the datasets were manually verified, including temperature data and electricity usage. Electricity usage appeared to be similar on the same days in different weeks. Thus, several weeks of data were used to plot periodic waveforms. Fig. 3.17 presents two examples from the PJMHEC dataset that reveal the nature of the dataset. Within each period, the five high peaks indicate that more electricity is

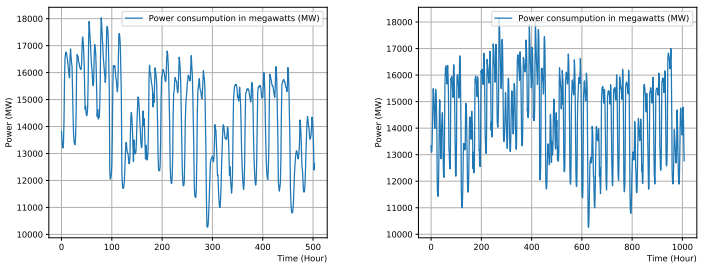

Figure 3.17: Two examples from PJME hourly energy consumption dataset. The left plot has three periods while the right plot has six periods.

consumed during working days and less during the weekend. Unlike the electricity usage data from the PJMEHEC dataset, the temperature data from the HHWD and LEMD datasets contain both short-term and long-term periods. In short-term periods, such as different days of the same week, they may have similar temperature readings. In addition, temperature readings in different years within a decade appear to be periodic as well. However, the shorter the time window is, the noisier the waveform appears. Short-term prediction is also less challenging than long-term prediction. Therefore, the number of years was used as labels for temperature data.

Fig. 3.18 presents three examples from the HHWD dataset. The plots all have a different number of periods; however, the images are the same size. The left plot has only one period, the middle plot has three periods, and the right plot has five periods. Comparing the three plots demonstrates that a larger number of periods plotted in an image leads to plots that are less clear. In addition, there are differences between each period, which signifies that noise exists within the dataset.

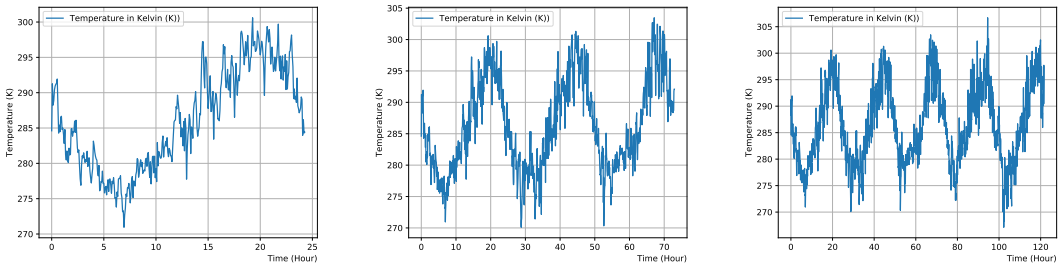

Figure 3.18: Examples from historical hourly weather dataset. The left plot has one period, the middle plot has three periods, and the right plot has five periods. The sizes of all three images are the same.

3.8 Training and testing PCA using real-world dataset

Unlike generated datasets, real-world datasets are unbalanced. This signifies that if 70% of instances of a dataset are from one class, then the model simply maps everything to that class, which will result in 70% accuracy. Thus, accuracy and loss alone are not sufficient to justify the performance of PCA, and extra measurements must be deployed.

Fig. 3.19 presents a result template of binary classifications, namely, a confusion matrix. The matrix calls instances that are correctly classified with respect to the ground truth label **t**rue **p**ositives (TPs) and **t**rue **n**egatives (TNs). Positive instances classified as negative are called **f**alse **n**egatives (FNs), while negative instances classified as positive are called **f**alse **p**ositives (FPs).

From the matrix, accuracy is defined as

$$accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{3.16}$$

As mentioned, the accuracy measurement is not sufficient for unbalanced datasets. Thus,

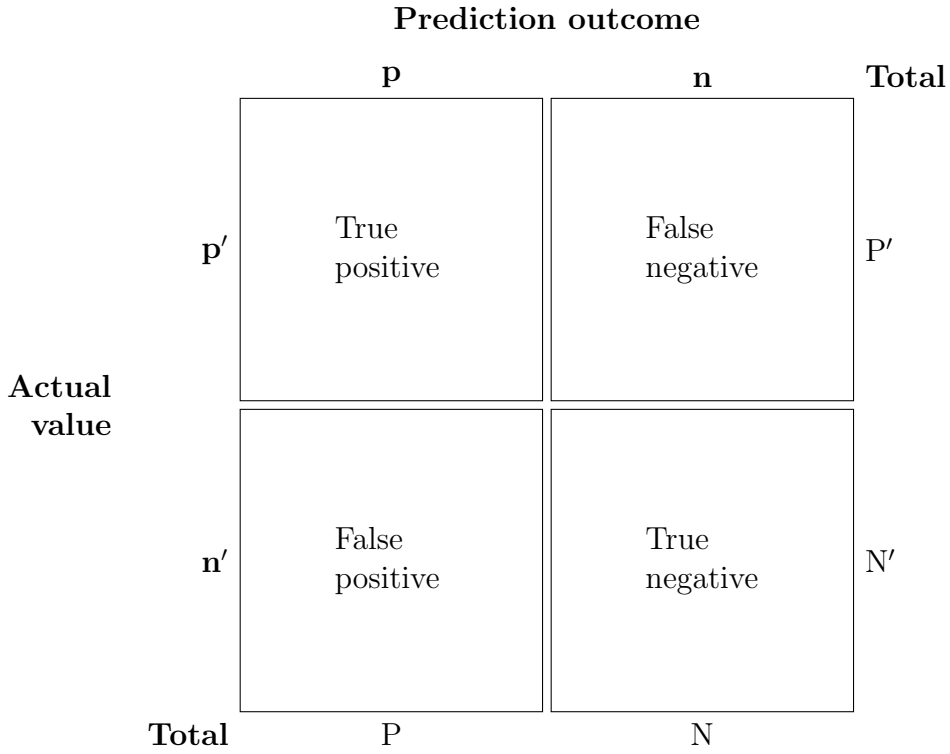

Figure 3.19: Example of confusion matrix for binary classification.

we use the F1 score, which is defined as follows:

$$precision = \frac{TP}{TP + FP} \tag{3.17}$$

$$recall = \frac{TP}{TP + FN} \tag{3.18}$$

$$\text{F1-score} = 2 \times \frac{precision \times recall}{precision + recall} \tag{3.19}$$

The F1 score combines two other measurements: **precision** and **recall**. Precision represents the proportion of predicted positive instances that are actually from the positive class. That is, when the classifier reports that an instance is positive, precision is the probability that the classifier is correct. Recall calculates the fraction between true positive instances and the total actual positive instances. In other words, it represents the percentage of actual positive instances that are correctly classified. Precision and recall are not entirely independent, and it is sometimes difficult to select one. This is one of the reasons for using the F1 score, which is a combination of the two. The F1 score is similar to the arithmetic mean but is not identical, and it is always between the precision and recall. However, the arithmetic mean gives the same weight to every element, whereas the F1 score gives larger weights to lower values. For example, if a classifier produces 100% recall and 0% precision, the F1 score is 0%. However, the arithmetic mean is 50%.

The experiments were configured as follows. In order to prevent possible over-fitting, the algorithm had 20 epochs to learn features from each dataset, and then the network will be deployed on testing datasets right before re-initialization. The result is obtained through averaging performance measurements over multiple runs. Fig. 3.20 presents the F1 score of the PCA during its training on a real-world dataset. The plot illustrates that the PCA achieved an F1 score of 95% for the HHWD, ENTSOE, and PJMHEC datasets. After the first epoch, the PCA hardly learned anything new from the data.

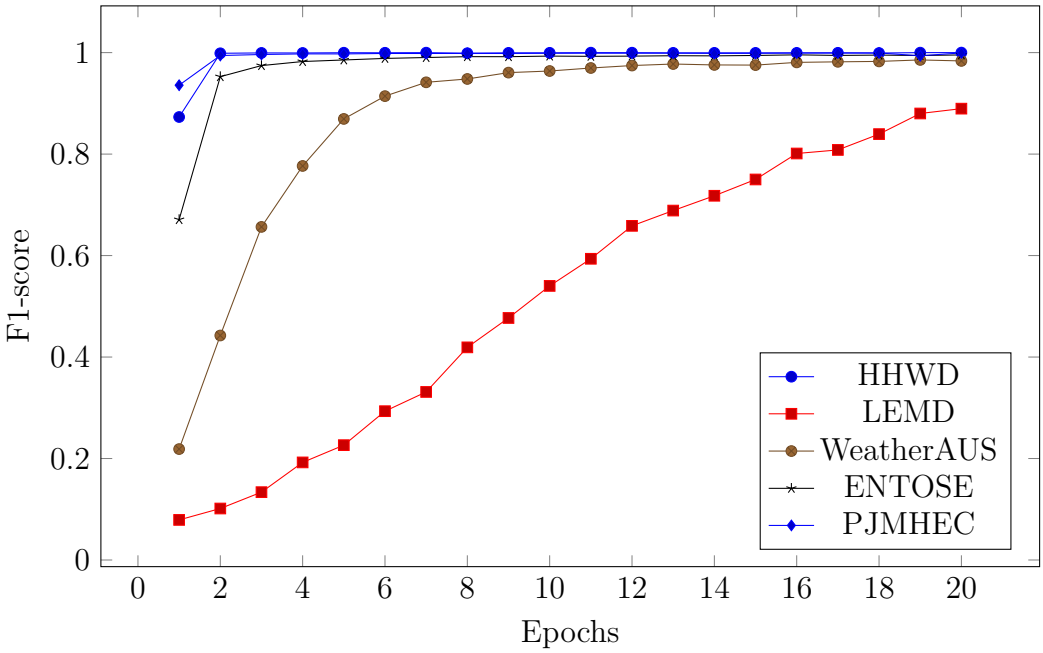

Figure 3.20: F1 score of period classification algorithm (PCA) during training. The HHWD dataset was the simplest for the PCA. After the first epoch, the PCA was able to perform good classification, and there was no new information for the PCA to learn. The ENTSOE and PJMHEC datasets experienced the same result. The WeatherAUS dataset required eight epochs for the PCA to learn. Overall, the LEMD dataset was the most difficult for the PCA. After 20 epochs, the F1 score was still approximately 0.9 for the LEMD dataset.

However, the PCA struggled on the WeatherAUS dataset and even more so on the LEMD dataset. The PCA trained for 14 epochs on the WeatherAUS dataset to achieve the same performance for which it trained for one epoch on the HHWD dataset. In addition, the PCA completed 20 epochs on the LEMD dataset with an F1 score of over 80%. We believe that the difference was caused by the number of instances in each dataset. HHWD, ENTSOE, and PJMHEC had the largest number of images, which signifies that the PCA had more training material within each epoch. Because we trained the PCA by the Adam optimizer, weight updates occurred at the end of each batch. This means that if the batch size remained the same, a larger dataset would produce a larger number

Test result of period classification algorithm

	Validation loss	Validation accuracy
HHWD	1.13E-06	100%
LEMD	2.44E-01	92.1875%
WeatherAUS	2.85E-01	92.4107%
ENTSOE	4.50E-03	99.7948%
PJMHEC	1.15E-03	99.9756%

of batches, and consequently, more model optimization.

Now, we know that the PCA can fit the training data well; however, this does not guarantee that it can fit the test data successfully. Ten percent of the training data were saved for later validation. The validation phase occurred at end of each epoch, and these data were never used for optimization. By applying the model on the validation data, we can gain insights about whether the PCA overfits the training data.

Figs. 3.21 and 3.22 present the results of applying the PCA on the validation dataset. Generally, Figs. 3.22 and 3.20 exhibit the same trends. However, for the WeatherAUS dataset, the PCA performed more poorly on the validation data than on the training data. In addition, Fig. 3.21 demonstrates whether the PCA actually learned concepts or was simply biased by the training data. The plot illustrates that the optimizer did not overtrain the PCA across the training procedure. The reason for this is that the validation loss does not increase for the HHWD, ENTSOE, and PJMHEC datasets. However, there are peaks in the plots, and overall, the validation loss decreases.

At least 10% of the entire datasets was saved for the final testing of the PCA. Table 3.1 presents the test results. Every experiment was performed three times; thus, the numbers in the table are the mean of three values. Values from the table are close to the right end of Figs. 3.21 and 3.22, which indicates that the model is generalized enough to fit both training and test data with acceptable performance.

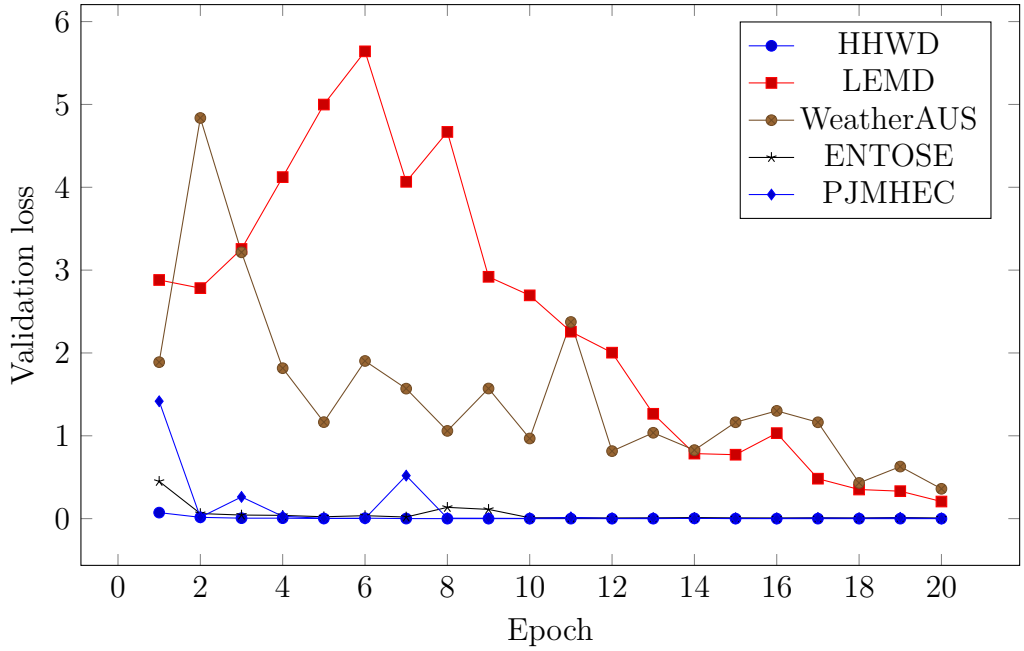

Figure 3.21: Validation loss of period classification algorithm on real-world datasets.

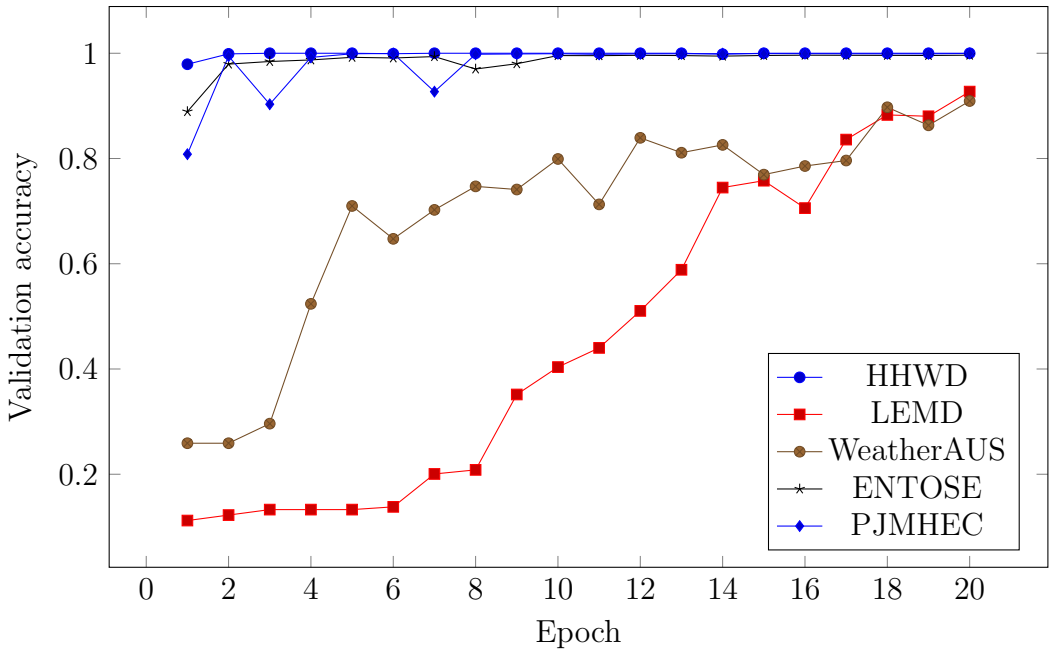

Figure 3.22: Validation accuracy of period classification algorithm on real-world datasets.

Overall, the PCA performed better on real-world datasets than on the generated dataset. The PCA achieved over 90% for all five datasets. However, its performance dropped to below 70% for some generated datasets.

Our experiments demonstrate that the PCA is suitable for real-world periodic classification tasks. Its performance may vary according to the quantity of data and polynomial trends. If the polynomial trend is too large, then the data may not appear periodic in the images. However, the PCA can potentially achieve 100% accuracy with sufficient data.

Chapter 4

Period Length Detection Within Controlled Environment By Learning Automata

CNNs require manually labeled training data in advance; however, such datasets are not always publicly available. Thus, this chapter introduces two other algorithms that can detect the period length without predetermined labels. These two algorithms are referred as PDAs, and the functionality is powered by learning automata (LAs). LAs are described in detail in Section 4.1. Section 4.2 provides the original and naive version of the algorithm with analysis, where the second algorithm is an enhanced version of the first algorithm with the same design. Section 4.3 illustrates how the enhanced algorithm tolerates controlled noise.

4.1 Learning automata in controlled environment

LAs are a type of machine learning method that adjusts a subsequent decision according to the feedback from the environment. LAs are formally defined as follows.

Definition 1. LAs require five parameters $(\alpha, \Phi, \beta, F, G)$ as follows:

1. α is a set containing all possible actions that LAs can take at each step. Each LA must take one and only one action for each step.
2. $\Phi = \{\phi_1, \phi_2, ..., \phi_s\}$ holds the internal states of the system.
3. β is the set of possible feedback from the environment.
4. $F : \Phi \times \beta \to \Phi$ defines the system transition map of LAs where feedback β is provided by the environment at each step.
5. $G : \Phi \to \alpha$ is the map for LAs to select the next action based on the current system state.

Definition 2. The environment is another part of the system. After an LA actions, the environment responds with feedback. The feedback can be either positive (a reward) or negative (a penalty).

The environment is assumed to be stationary, which means that for $\alpha_i \in \alpha$, the probability of receiving a penalty is fixed at P_i, while the probability of receiving a reward is $1 - P_i$.

PTSDs can be modeled as an environment that constantly provides the next data sample along the time axis to the LA. PTSDs satisfy the stationary assumption due to their periodic features, and can be viewed as a data stream D formed by repeating atoms S, the shortest repeating substring. Importantly, the length of atom S is the

period length of the data stream of the PTSD. PTSDs can then be treated as sequence "$SSS...$".

All LAs attempt to minimize the probability of receiving negative feedback from the environment. At each step, an LA learns from the environment's feedback. If the feedback is positive, the LA gains confidence in its previous action. Otherwise, if the feedback is negative, the LA adjusts its subsequent action. Thus, LAs behave in a specific environment. For PTSD classification, the environment is a sequence of data samples that has periodic features in the time domain.

4.2 Unsupervised learning algorithm for period detection

This section introduces a naive periodic detection algorithm based on LAs. This algorithm assumes that the input sequence repeats more than once and is periodic, and that the data samples are noise-free. The principle is simple: deploy an LA and begin feeding data samples one by one. Each time the LA receives a data sample, it first determines whether it is another element from the same period or whether the LA has seen this data sample before. The sequence then starts repeating. The algorithm thus does not require labeled training. Before describing the algorithm in greater detail, it is important to understand the question from the perspective of the LAs.

Lemma 1. An LA verifies a sequence formed by n distinct elements $e_0, e_1, ..., e_{n-1}$ with stride m. Before the LA observes the same e_i twice, all n distinct elements are observed exactly once if and only if m and n are coprime.

If m and n are coprime, an LA observes a repeating sequence with all but shuffled elements after n steps. However, the LA observes a shorter repeating sequence if m and

n are not coprime.

Example 1. A repeating sequence S is formed by $n = 3$ distinct elements $\{e_0, e_1, e_2\}$. If S is e_0, e_1, e_2, ..., and an LA is deployed at $i = 2$ also configured with stride $m = 2$, then it observes the following:

- $S[(2 + 0 \times 2)] = S[(2 + 0 \times 2) \bmod 3] = e_2$
- $S[(2 + 1 \times 2)] = S[(2 + 1 \times 2) \bmod 3] = e_1$
- $S[(2 + 2 \times 2)] = S[(2 + 2 \times 2) \bmod 3] = e_0$
- $S[(2 + 3 \times 2)] = S[(2 + 3 \times 2) \bmod 3] = e_2$
- ...

It can be seen that the observed sequences repeats after n steps. Meanwhile, the repeating subsequence is composed of the same distinct elements but the order is shuffled. However, the same LA observes a repeating sequence with a shorter period length if n and m are not coprime.

Example 2. A repeating sequence S is formed by $n = 4$ distinct elements $\{e_0, e_1, e_2, e_3\}$. If S is e_0, e_1, e_2, e_3, ..., and an LA is deployed at $i = 2$ also configured with stride $m = 2$, then it observes the following:

- $S[(2 + 0 \times 2)] = S[(2 + 0 \times 2) \bmod 4] = e_2$
- $S[(2 + 1 \times 2)] = S[(2 + 1 \times 2) \bmod 4] = e_0$
- $S[(2 + 2 \times 2)] = S[(2 + 2 \times 2) \bmod 4] = e_2$
- $S[(2 + 3 \times 2)] = S[(2 + 3 \times 2) \bmod 4] = e_0$
- ...

In this case, the LA still observes a repeating subsequence; however, its period length is clearly shorter than the original period length. The observed repeating sequence is "$e_2, e_0, \ldots$" which has a period length of 2, while the original repeating sequence is "e_0, e_1, e_2, e_3".

It should be noted that Lemma 1 holds as long as the elements of the sequence are distinct. Thus, Lemma 1 can be generalized as follows:

Theorem 1. For a sequence S composed of n distinct elements where $n \in \mathbb{Z}^+$, a data stream D is defined as the concatenation of repeating S. Let $m \in \mathbb{Z}^+$ and $m \leq n$, an LA pops m elements from the head of D at a time and observes the first element from the pop. If and only if m and n are coprime, then the LA observes a new repeating S', which is the shuffled version of S.

Proof of Lemma 1 is sufficient to prove Theorem 1.

Proof of Lemma 1 : ($\Rightarrow$) Assume (towards contradiction) that if m, n are coprime, then the LA does **NOT** observe all numbers from e_1 to e_n exactly once before it observes a second i. From this assumption, we know that the LA can observe the same element more than once within n observations, which means that $\exists a, b \in \mathbb{N}, 1 \leq a < b \leq n$ such that $i+am \equiv i+bm \pmod{n}$. Rearranging the equation, we obtain $(b-a)m \equiv 0 \pmod{n}$. Because m and n are coprime, we obtain $n|(b-a)$. In other words, $(b-a)$ is a multiple of n. From the assumption, we know that $(b-a) < n$, which contradicts that $(b-a)$ is a multiple of n. Therefore, if m, n are coprime, then the LA **DOES** observe all elements from e_1 to e_n exactly once before it observes a second i.

($\Leftarrow$) Assume (towards contraposition) that if m, n are **NOT** coprime, then the LA does **NOT** observe all numbers from e_1 to e_n exactly once before it observes a second i. Let $\gcd(m, n) = c$, $m = pc$, $n = qc$, where $p, q, c \in \mathbb{N}_{\geq 1}$. Then, $qm = q \times pc =$

$p \times qc = pn$ holds. In other words, n is a divisor of qm. Moreover, $i + qm \equiv i \pmod{n}$. According to $n = qc$, we know that $q < n$. In addition, there are fewer elements in $|\{i + q_i \times m_0| \leq q_i \leq q\}|$ than n.

Therefore, the contraposition of the statement is true. □

The contraposition of Theorem 1 is as follows.

Corollary 1. For a sequence S composed of n distinct elements, a data stream D is defined as SSS.... An LA observes every m element of D starting from i. The observed stream is formed by repeating a subset of S but not S if and only if m and n are **not** coprime.

Examples 1 and 2 not only reveal the perspective of LAs when m and n are coprime, but also reveal what occurs when they are not coprime. Thus, the following can also be deduced.

Theorem 2. In the same context of Theorem 1, suppose that data stream D is composed of a repeating sequence S with length n and an LA deployed to observe every m element of D. If m is a divisor of n, then the observed repeating sequence has a length of $\frac{n}{m}$.

Proof. Let $n = mp,\ p \in \mathbb{N}_{\geq 1}$ since m is a divisor of n.

For all e_i provided by stream D, we have $i + mp \equiv i \pmod{n}$, which implies that $e_{i+mp} = e_i$. This signifies that after the pth observation, the stream tends to repeat itself. To prove that p is the length of the atom, assume that there exists q such that $0 < q \leq p$, and that the observed stream has a repeating component of length q. The assumption implies that $i + mq \equiv i \pmod{n}$; therefore, $n|mq$ and mq are either larger than or equal to n.

By combining them, we obtain $(mq \geq n = mp) \rightarrow (q \geq p)$. Recall that q is assumed to be less than or equal to p.

Therefore, $q = p$, where p is the length of the shortest repeating component. □

Inspired by Corollary 1 and Theorem 2, Algorithm 4 can determine whether m is a divisor of the data stream's period length when S contains only unique elements.

Algorithm 4: Verification of non-one divisor of period length

Inputs : integer m, data stream D, index i
Output: boolean value: whether m is coprime with period length of D
1 Initialize LA_1 and LA_2;
2 Employ LA_1 to iterate over D and save unique elements in list l_1;
3 Employ LA_2 to iterate every mth element of D starting from position i. The unique elements are saved in list l_2;
4 **return** whether all elements in l_1 do **NOT** also exist in l_2;

Hypothetically, the length of the data stream can be indefinite. However, PTSDs are typically sampled from a specific time window. Thus, Algorithm 4 does not place early terminal conditions on the iterations of LA_1 and LA_2. Ideally, if the length of repeating sequence S is n, then after the nth iteration, both LA_1 and LA_2 should have seen every unique element. As mentioned at the beginning of this chapter, the period length is unknown a priori. In addition, it is possible for a PTSD to have only one period. Thus, the iteration number is bounded by the length of the data stream.

There are cases that Algorithm 4 cannot handle well. This is because Algorithm 4 is defined under the assumption that elements within a single period are unique. Example 3 illustrates the result when this assumption is not satisfied.

Example 3. Data stream D is formed by the repeating sequence $S :=$ "$a,\ a,\ b,\ a$". Then, D appears as "$\ldots\ a,\ a,\ b,\ a,\ a,\ a,\ b,\ a\ \ldots$" whose period length $n = 4$. Algorithm 4 is used to determine whether $m = 2$ is a divisor of n. By calling Algorithm 4(m, D, $i = 0$), two LAs observe $l_1 = [a,\ b]$ and $l_2 = [a,\ b]$, which contain the same elements. The algorithm returns TRUE, indicating that m and n are coprime. This is not what is excepted because $m = 2$ is not coprime with $n = 4$.

The problem in Example 3 does not always occur. If we modify parameter i, then

the problem does not appear. Suppose that $i = 1$, but everything else remains as they are in Example 3, resulting in $l_1 = [a,\ b]$ and $l_2 = [a]$. Because element b is missing from l_2, the algorithm declares that m and n are not coprime.

It would be computational costly for LA_2 to search all possible i using brute force. However, Example 3 demonstrates that LAs can extract periodic features with the desired configurations. Analyzing the failure of Algorithm 4 reveals that LA_2 is biased because it does not learn from the environment and sometimes requires a shift of view (different i).

After modifications, the new algorithm, Algorithm 5, is able to detect the period length of the input data stream. Algorithm 5 first initializes an LA with $i = 0$ and $m = 1$. It then begins traversing the input data stream. The algorithm initializes another LA whenever the existing LA observes more than one unique element. The most recently initialized LA begins traversing one index after the previous index. Before the LAs start traversing the data stream again, we increment the stride m by 1 for all LAs.

If the period length n of the data stream is equal to the number of LAs deployed, then all LAs should observe one and only one unique element from the data stream.

Example 4. For input data stream D composed of the repeating sequence $S =$ "1, 1, 1, 2". Algorithm 5 executes the following:

1. Initialize LA_0 with $m = 1$ and $i = 1$.
2. If the LA only observes 2, then Algorithm 5 deploys LA_1 with $m = 1$ and $i = 2$.
3. Increment m by 1 for LA_0 and LA_1.
4. When $LAList.length = 2$, the second LA observes 2 with stride $m = 2$.
5. A new LA, LA_2, is deployed with $m = 2$ and $i = 3$.

Algorithm 5: Naive learning automata (LA) period detection

Inputs : data stream: D
Output: integer: period length of D

Let $M = 1$ /* the stride to iterate D*/;
Let $I = 0$ /* the index of D that an LA will be deployed on*/;
Initialize LA_0 with $LA_0.m = M$ and $LA_0.i = I$;
Initialize Boolean value $done = FALSE$;
Push LA_0 onto an empty list $LAList$ /* Now the list contains one LA which deploy at index 0 of D with stride equal to 1 */;
while *NOT* $done$ **do**
    for $j = 0;\ j < LAList.length;\ j = j + 1$ **do**
        Employ LA_j to iterate every $LA_j.m$th element of D starting from position $LA_j.i$;
        if LA_j *observes more than one unique element* **then**
            $I = I + 1$;
            Initialize a new LA_M with $LA_M.m = M$ and $LA_M.i = I$;
            Push LA_M onto $LAList$;
            Increment m for all LA by 1;
            $M = M + 1$;
            Break;
        end if
        $done = TRUE$
    end for
end while
return $LAList.length$;

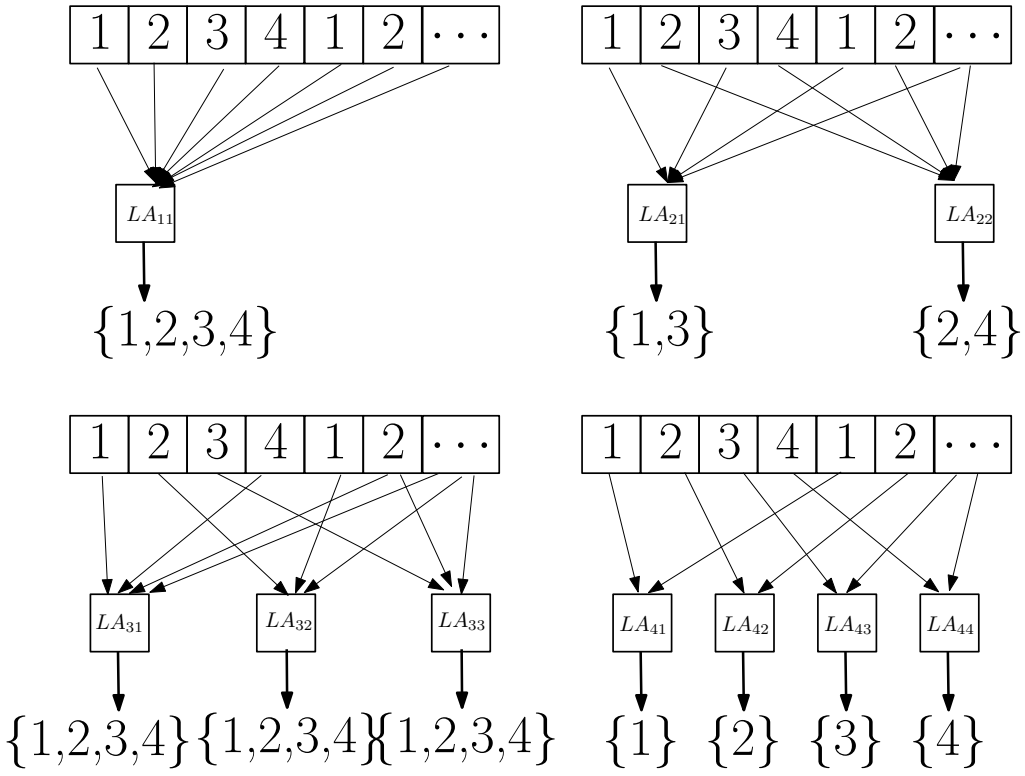

Figure 4.1: Naive PDA deals with a data stream of period four composed of unique elements.

6. Increment m by 1 for all LAs.
7. When $LAList.length = 3$, the most senior LA observes 2.
8. The fourth LA, LA_3, is deployed with $m = 3$ and $i = 4$.
9. Increment m by 1 for all LAs.
10. All LAs iterate through the data stream without observing more than one unique element.

Algorithm returns 4, which is equal to the length of S.

Example 4 is not a specific case, but can be generalized.

Theorem 3. In the same context of Theorem 1, for integer $l \in \mathbb{Z}^+$, suppose that l LAs are deployed to the same data stream D, which has a period length of n. If all LAs share the same stride m, and $m = l = n$, then all l LAs observe one and only one unique element.

Proof. The ith LA observes every $(i + al)$th element from the input data stream, where $a \in \mathbb{N}_{\geq 1}$. Because $l = n$, $i + al \equiv i \pmod{n}$ for all a.

Therefore, all LAs observe one and only one unique element. □

4.3 Enhanced period detection algorithm with noise tolerance

Section 4.2 introduces a period detection algorithm that requires the same number of LAs as the period length. Therefore, Algorithm 5 can be computationally expensive if the period length n is not bounded. Moreover, the current implementation cannot

tolerate any noise. Algorithm 5 is thus a naive approach and can be improved further. This section introduces an enhanced period detection algorithm based on Algorithm 5.

Theorem 2 demonstrates that if m divides n, then the LA observes a shorter repeating sequence with a length of $\frac{n}{m}$. This idea can be applied recursively until the observed repeating sequence contains only one unique element. However, Example 3 demonstrates that it is difficult for LAs to reject m divides n, but they can easily confirm it.

Lemma 2. Suppose that an LA observes data stream D with period length n. The LA is configured with stride m, where $1 < m \leq n$. If the LA observes fewer unique elements than the original data stream contains, then the greatest common divisor of m and n is greater than 1.

Proof. Assume (towards contradiction) that the LA observes fewer unique elements, and m is coprime with n.

Suppose that the LA is deployed at position i, and element $e_j \in S$ is not observed. By the assumption, m and n are coprime, and Theorem 1 indicates that the LA should visit every element at least once regardless of whether it is unique. In addition, e_j should also be observed, which is a contradiction. □

Theorem 4. In the same context of Theorem 1, suppose that m is the least positive integer that makes an LA observe fewer unique elements than the original data stream D contains. Then, $m|n$, where n is the period length of D.

Proof. Assume (towards contradiction) that m is not a divisor of n.

According to Lemma 2, because the LA observes fewer unique elements, $gcd(m,\ n) > 1$. Suppose that $gcd(m,\ n) = \epsilon$, where $\epsilon \in \mathbb{N}$ and $\epsilon < m$. Then, the LA should already have a stride equal to ϵ; however, the observed sequence contains the same number of unique elements. However, the LA observes at most $\frac{n}{\epsilon}$ unique elements, and at least

one element is not observed when the stride is equal to m. Let us call this element $S[i+q\epsilon]$, where $q \in \mathbb{N}^+$ and $q\epsilon \neq m$. We know that $S[i+q\epsilon] = S[i+q\epsilon \times lcm(m,n)]$ and $lcm(m,n) = \frac{n \times m}{\epsilon}$. By rearranging the equations, we obtain the following:

$$S[i+q\epsilon] = S[i+q\epsilon \times lcm(m,n)] = S[i+q\epsilon \times \tfrac{n\times m}{\epsilon}] = S[i+q \times n \times m]$$

Thus, every element is observed when the stride is equal to ϵ, and should also be observed when the stride is equal to m. In other words, either m is not the least positive integer that makes the LA observe fewer unique elements, or $\epsilon = m$. Therefore, m divides n. □

As mentioned, different starting positions i may result in different observation results. This suggests deploying m LAs at positions $i = 1, 2, ..., m$. It should be noted that all m LAs may not conclude $m|n$, even it is true. The following example demonstrates such a situation.

Example 5. Suppose that data stream D is composed of the repeating sequence $S =$ "1, 2, 2, 1". The repeating sequence has a length of $n = 4$, and we attempt to determine whether $m = 2$ is a divisor of n. By deploying LA_1 at $i = 1$ and LA_2 at $i = 2$, both LAs observe two unique elements, which is the same number of elements as can be found in the original data stream. However, when $m = 4$, all LAs observe one and only one unique element.

As illustrated in Example 5, if the repeating sequence contains fewer unique elements than the period length n, we cannot rely on prime numbers as factors of period lengths. In Example 5, $m = 2$, which is a prime factor of $n = 4$, and the two LAs do not capture this information. The next two prime numbers are 3 and 5, and neither of them will capture a stream containing fewer unique elements. By summarizing Theorem 4 and Example 5, the LAs cannot skip non-prime numbers (including n) as a potential factor

of n. Therefore, Example 5 indicates that the worst-case scenario is up-bounded if we iterate through $1 < m \leq n$. The following procedure is capable of identifying the least non-one divisor of unknown period length n.

Algorithm 6: Obtain factor

Inputs : data stream: D
Output: int: a factor of period length of D

```
1  Initialize LA_0;
2  LA_0.setStride(1);
3  LA_0.deploy(1);
4  N = LA_0 obtains number of unique elements from D;
5  for m = 2; m ≤ D.length; m = m + 1 do
6      Initialize m number of LAs;
7      for i = 1; i ≤ m; i = i + 1 do
8          LA_i.setStride(m);
9          LA_i.deploy(i);
10         N' = LA_i obtains number of unique elements from D;
11         if N' < N then
12             return N'
13         end if
14     end for
15 end for
16 return D.length
```

Algorithm 6 finds the least non-one divisor m_0 of period length n. During the process, Theorem 1 demonstrates that the LAs observe a shorter repeating sequence if $m|n$. It is possible that the observed stream contains only one unique element, but $\frac{n}{m} > 1$. In other words, the terminal condition cannot be as simple as the one in which the LA observes only one unique element.

Example 6. Suppose that data stream D has period length $n = 4$ and is composed of $S =$ "3, 5, 3, 1". Algorithm 6 would return 2 as the least non-one factor that is correct. However, our goal is to determine n. Now, we have two new streams, one observed by each LA. If $D =$ "3, 5, 3, 1, 3, 5, 3, 1, ..." then LA_1 observes $D_1 =$ "3, 3, 3, 3, ..." and

LA_2 observes D_1 = "5, 1, 5, 1, . . . ". Furthermore, if we recursively apply Algorithm 6 on D_1 only, the algorithm returns 1, although we have not reached our goal.

Algorithm 7: Obtain period length

Inputs : data stream: D
Output: int: period length of D

```
1  m = GetFactor(D);
2  N = 1;
3  for i = 1; i ≤ m; i = i + 1 do
4  |   Initialize LA_i with stride = m and position at D[i];
5  |   D_i = observed stream by LA_i;
6  |   N' = number of unique elements in D_i;
7  |   if N' > 1 then
8  |   |   branch = GetPeriodLength(D_i);
9  |   |   N = max(N, branch);
10 |   end if
11 end for
12 return N × m
```

To perform the period detection task, all LAs must observe only one unique element. Example 6 and Theorem 3 suggest that we allow all LAs to finish their observations and then determine the next move. The following procedure determines the unknown period length of a data stream. Fig. 4.2 demonstrates that a data stream composed of non-unique elements and two branches of LAs have different depths. Algorithm 7 uses the result obtained by the deepest branch.

This approach traverses the data stream more times than the previous approach when N is small (less than 10). However, for some applications, such as weather prediction and electric power consumption, the targeted period may be 1 year. In other words, one period may last for $365 \times dailySampleRate$ as many time indices. Algorithm 7 can be useful in this scenario. Including real-world datasets raises the issue of noise tolerance. Using weather prediction as an example, the average day temperatures on the same date of two different years may not be the same, and the units of various climatic measurement

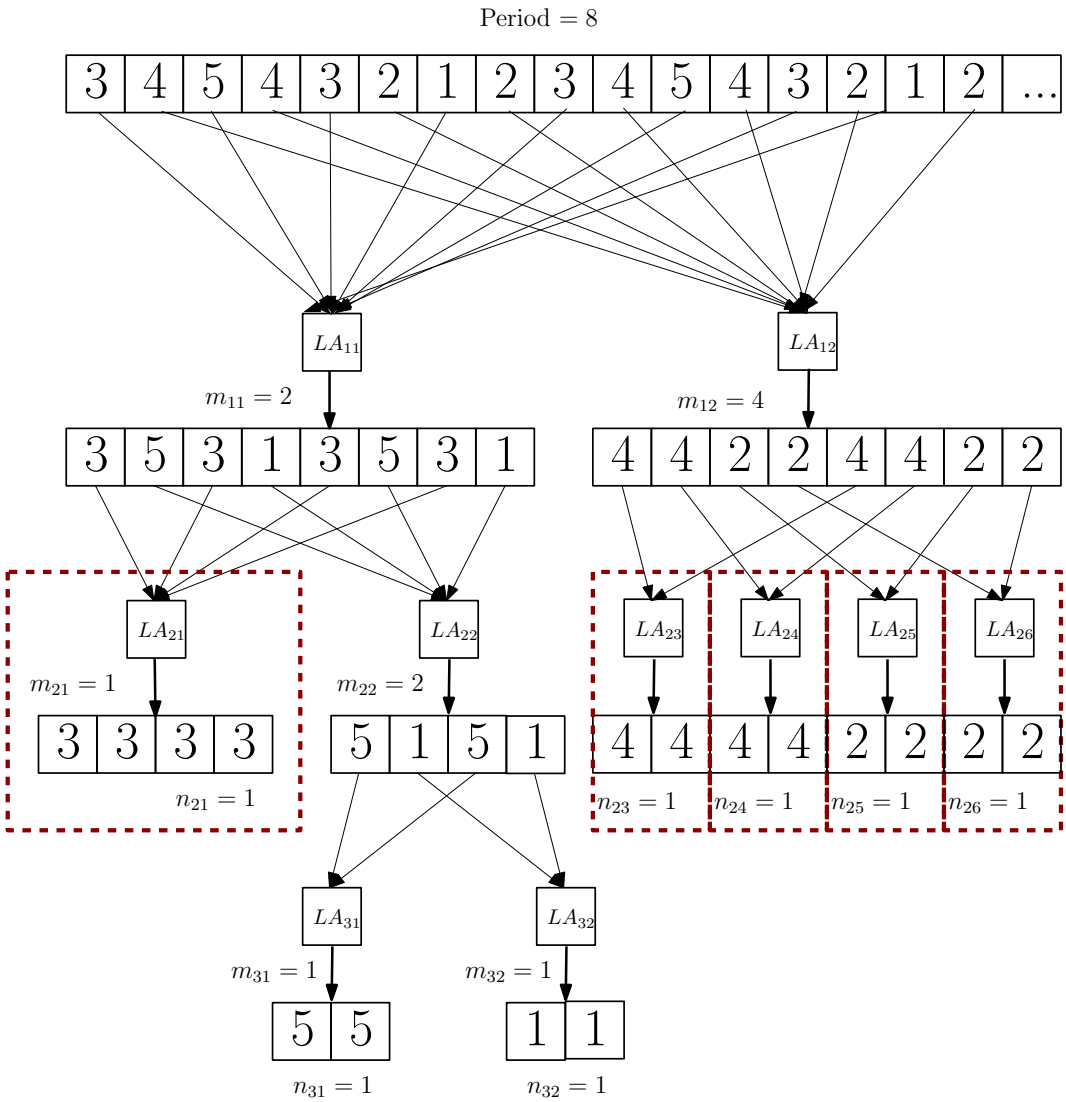

Figure 4.2: Two branches of learning automata (LA) deal with a data stream composed of non-unique elements.

are not integers.

The current design of LAs takes each observation precisely. This signifies that even if the stride is equal to the period length, the LA will not take a noise-padded sample as the same element. This then causes both approaches to malfunction. To tolerate natural noise, LAs must be informed a priori of the maximum unit variances. This is not ideal, but can be a solution in certain cases.

4.4 Experiments of PDA

Previous examples present possible situations in which PDAs operate in a scope it is not designed for. To fully justify our theoretical analysis of PDAs, we performed various experiments. More specifically, we introduced the following functionality of PDAs and enhanced PDAs:

1. A naive PDA can detect the period length when repeating sequences of the data stream are composed of unique elements, and noise is **not** implemented into the system.

2. A naive PDA can detect the period length when repeating sequences of the data stream are composed of unique elements, and noise is implemented into the system.

3. An enhanced PDA can detect the period length when repeating sequences of the data stream are composed of random elements, and noise is **not** implemented into the system.

4. An enhanced PDA cannot determine the period size from the real-world dataset described in Section 3.8.

Experiments 1 and 2 demonstrate that the naive PDA is successful in its designed scope. In addition, Experiments 3 and 4 demonstrate that the enhanced PDA functions in a more general environment.

The data streams used in Experiments 1, 2, and 3 were composed of a repeating sequence with only unique elements. To ensure this, we generated an array that contained all shuffled integers from 1 to the target period size. Thereafter, the testing data streams were simulated by repeatedly concatenating the generated sequence. The concatenation process occurred at least twice for each data stream. Otherwise, Algorithm 6 may have malfunctioned when the target period size was much larger than the repeating times of the sequence. This is because the stream was too short for the LA to explore all possible unique elements. For example, for a data stream containing only unique elements (repeating sequence before concatenation), an LA is placed at the first element with an m that does not divide the length of the period. Before the LA explores all potential unique elements, the stream ends, and the LA claims that it determined an m that indicates fewer unique elements. Algorithm 6 would not function as excepted with such misleading information. Algorithm 8 provides the details of the above discussion in a different form.

Algorithm 8: Data stream generation (unique)

Inputs : int: $periodSize$, int: $repeatNum$
Output: data stream: D

```
S = range(1, periodSize);
S = shuffle(S);
D = empty data stream;
for i = 1; i ≤ repeatNum; i = i + 1 do
    S = S + noise;
    D = D.concatenate(S);
end for
return D
```

In Experiment 4, a strategy to generate a sequence with a non-unique element is not

only necessary but crucial for later experiments. We cannot simply generate a random sequence with a desired length that is not composed of other shorter repeating sequences. For example, a random generated sequence with a length of 4 could be "a, b, a, b". Algorithm 7 will return 2 as the period length when the labeled ground truth is 4. Due to this potential problem, we propose the following concept that eliminates this problem.

The updated generation algorithm works as follows. First, we randomly select an integer that is less than the desired period length and is coprime with it. The integer is called the sub-period length. Then, we randomly select all positive integers that are less than *subperiod length* to build part of the repeating sequence. This part may contain non-unique elements because all integers less than the sub-period length are sampled with equal probability. Then, the second half of the repeating sequence is padded with all integers from range $[sub\ period\ length + 1, desired\ period\ length]$. Because *sub-period length* is coprime with the desired period length, there does not exist an integer α such that $\alpha \times sub\ period\ length = desired\ period\ length$. Thus, each generated data stream may contain non-unique elements, and the period length is always the desired length.

Table 4.1 illustrates our results and serves as additional information for the proofs in Section 4.3. Experiments 2, 4, 5 and 6 involve noise-padded data streams. As mentioned in Section 4.3, LAs are informed of the expected noise level. However, in real-world situations, the expected noise level may be over- or under-estimated. To simulate this phenomenon, the noise variance and expected noise variance were controlled individually. We discretized continuous noise into three different levels. Specifically, if the data samples are not padded with any noise, its noise level is labeled as "None". Besides, if noise is padded, but the variance is less than half the unit scale, we consider it is a "Low" noise level. At last, more severe noise is considered as "High" noise. In addition, the period

Algorithm 9: Data stream generation (non-unique)

Inputs : int: $periodSize$, int: $repeatNum$
Output: data stream: D

```
subPeriodLength = randomInt(2, periodSize);
isCoPrime = coPrime(subPeriodLength, periodSize);
while not isCoPrime do
    subPeriodLength = randomInt(2, periodSize);
    isCoPrime = coPrime(subPeriodLength, periodSize);
end while
firstHalfS = range(1, subPeriodLength);
firstHalfS = shuffle(firstHalfS);
S = firstHalfS.concatenate(range(subPeriodLength + 1, periodSize));
D = empty data stream;
for i = 1; i ≤ repeatNum; i = i + 1 do
    S = S + noise;
    D = D.concatenate(S);
end for
return D
```

Table 4.1: Experiments of PDA

Exp	Algorithm	Noise Level	Elements	Correctness
1	Naive PDA	None	Unique	Always
2	Naive PDA	Low	Unique	**Not Always**
3	Enhanced PDA	None	Unique	Always
4	Enhanced PDA	Low	Unique	Always
5	Enhanced PDA	Low	Non-unique	Always
6	Enhanced PDA	High	Non-unique	**Not Always**

size of the data streams was also controlled in all experiments. The period size was iterated from 3 to 74.

Before the second experiment, the PDA was expected to perform well when the noise level was less severe than reported. However, the results indicate that the PDA could not correctly determine the period length when the noise level reached the expected level. Importantly, we discovered that the longer the period size was, the less noise the PDA could tolerate.

Lastly, the enhanced PDA was tested using real-world datasets, such as HHWD and LEMD. The results indicated that the enhanced PDA failed to determine the period size. The reason is that the real-world datasets were too noisy for the enhanced PDA, and the length of the period was too long. These datasets had a sample rate in minutes, and each period could last for more than 100 minutes. The aforementioned four experiments demonstrated that a longer period makes the enhanced PDA more vulnerable to noise.

4.5 Runtime and efficiency

In contrast to PCA, the PDA does not require preprocessing, labeling, or training. The PDA's running time is $length\ of\ data\ stream \times number\ of\ non\ unique\ prime\ factors$. The proof for this follows Fig. 4.2. The longest path for the recursive calling tree is the number of non-unique prime factors. In the example, Fig. 4.2, the longest path is constructed by the three non-unique factors of 8: $\{2, 2, 2\}$. During this process, if the PDA cannot determine any factor of the period size, it attempts to deploy more LAs until it has the same number of LAs as $length\ of\ data\ stream$. Each deployed LA will not traverse the entire data stream; however, the number of elements are up-bounded by the length of the data stream. Denoting the length of the data stream as len, the above analysis can be formulated as follows:

$$\mathcal{O}_{PDA}(len) = \sum_{i=1}^{len} i \times len = \mathcal{O}_{PDA}(len^3) \tag{4.1}$$

In practice, this can be more efficient because each LA can perform on its own computational core. In addition, if noise is not implemented into the system, then larger period sizes are less likely to be a prime number [43]. This signifies that it is less likely for the PDA to perform in its worst case as the period size increases.

4.6 Memory usage

In terms of memory usage, both the PCA and PDA require additional memory space. The PCA uses additional memory space to store the trained CNN parameters. In addition, it requires more memory space to store the training graphs and their labels.

In contrast, the PDA requires much less memory space. The reason for this is that each LA only remembers the unique elements that it observes. In the worst case, there are *len* number of LAs, and each of them stores *len* number of unique elements. This signifies that in the worst case, the PDA requires

$$\mathcal{O}_{PDA}(len) = len^2 \tag{4.2}$$

amount of memory. As mentioned Section 5.1, if noise does not exist in the system, then the worst case is unlikely to occur.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis proposes two different machine learning algorithms for solving the period extraction problem. These algorithms are designed for two different working scenarios. The PCA is a supervised learning algorithm that requires a large quantity of labeled data. In contrast, the PDA is an unsupervised approach; however, it cannot tolerate much noise.

Table 5.1 presents an overall comparison between classical statistics (CS) approaches, the discrete Fourier transform (DFT) and the two proposed algorithms. In terms of

Table 5.1: Algorithm comparisons

Measure \ Approach	CS	DFT	PCA	PDA
Prior Knowledge	Required	**Not Required**	Required	**Not Required**
Human Involvement	High	High	High	**Low**
Running Time	Long	**Short**	Long	Medium
Memory Usage	High	**Low**	High	**Low**
Noise Tolerance	**Robust**	Fragile	**Robust**	Medium

prior knowledge required from the different algorithms, PCA requires labeled training data and CS usually requires experts who have domain knowledge. At the same time, it is possible for the DFT to extract more than one possible period length. In this case, human involvement is necessary to decide which one is the desired one.

For running time, in order to perform the CS, researchers take years to train and learn relevant domain backgrounds. Besides, it is necessary to collect training data for PCA, which could take years. For example, a climate period is usually longer than one year. For this reason, DFT and PDA clearly outperform the other two algorithms.

To the extend of memory usage, the PCA requires additional memory to store trainable feature maps and decision making weights. Along with the CS, researchers have to remember the necessary domain knowledge to proceed with their analysis. But, the DFT requires constant additional memory space, and the PDA takes significantly less extra space than the PCA.

To test the noise tolerance of the PCA and PDA, both proposed algorithms were tested on artificial and real-world datasets. In general, the PDA cannot tolerate as much noise as the PCA. As discussed in Section 4.4, the PDA failed to determine the period sizes of real-world datasets. On the same datasets, the PCA achieved an accuracy of over 90%. Because the PCA operates on the plotted graph of raw data, the noise must blur the graphic period patterns to affect the PCA's performance. That is, if the graphic pattern is recognizable by humans, then the PCA is able to classify the period length with a sufficient amount of training. At the same time, the PDA can be easily effected by not only white noise but also a sample rate mismatch. The previous experiments demonstrated that the PDA can fail in case of a high or unexpected noise level. When the PDA processes a data stream with a long period size, its tolerance decreases. For the synthetic datasets, when padded noise remains low or follows a Gaussian distribution,

both the PCA and PDA extract period information successfully. The PDA fails to detect any useful period information when the noise is above a threshold. However, the PCA has a more random performance if the noise increases in the dataset.

As previously mentioned, the PCA requires labeled data as a priori to perform training. In contrast, the PDA does not require labeled data, which leads the two algorithms to have different working scopes. If the data stream is noisy and its period pattern is stable, the PCA can perform at the human level after a period of training. However, if the data stream changes quickly but is less noisy, the PDA can be used to determine the period size in a bounded time.

5.2 Future work

Anticipated future work involves the following areas:

- **Applications**: We tested the PCA and PDA on the generated datasets as the well as limited real-world datasets. Performance analysis revealed that the PCA demonstrated high accuracy. We believe that various applications can utilize this model to improve their user experience. For instance, internet traffic usage prediction can be more accurate with precise period information.

- **Efficiency**: The deep convolutional model used in this work is relatively complex for the given task, and we believe that it is possible to train a shallower network to achieve the same performance. In addition, the necessary quantity of training data remains untested. Because shallower deep models have a smaller parameter space, they may result in less training data required to train the model. A model with fewer parameters results in higher portability, less energy consumption, and faster computation.

- **Algorithm**: Our analysis and experiments of the PDA revealed that it worked well when the noise was under the tolerance threshold, but failed to detect the period length when the noise exceeded half of the unit difference. To address this problem, researchers can consider **reinforcement learning** [40], a successor of LAs that can also be applied for the period detection task.

References

[1] Abhinav Agrawal and Namita Mittal. Using cnn for facial expression recognition: a study of the effects of kernel size and number of filters on accuracy. *The Visual Computer*, 36(2):405–412, 2020.

[2] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with cote: the collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, 2015.

[3] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

[4] Mustafa Gokce Baydogan, George Runger, and Eugene Tuv. A bag-of-features framework to classify time series. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2796–2802, 2013.

[5] Mariusz Bojarski, Davide Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[6] Aaron Bostrom and Anthony Bagnall. Binary shapelet transform for multiclass time series classification. In *International conference on big data analytics and knowledge discovery*, pages 257–269. Springer, 2015.

[7] Peter Brockwell and Richard Davis, editors. *Spectral Analysis*, pages 111–136. Springer New York, New York, NY, 2002.

[8] Julian Castro. Weather madrid 1997 - 2015, Sep 2016. URL https://www.kaggle.com/juliansimon/weather_madrid_lemd_1997_2015.csv.

[9] Patricio Cerda, Gaël Varoquaux, and Balázs Kégl. Similarity encoding for learning with dirty categorical variables. *Machine Learning*, pages 1–18, 2018.

[10] Dan Ciresan, Ueli Meier, Jonathan Masci, Luca Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[11] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[13] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153, 2013.

[14] Kartick Dey and Subrata Saha. Influence of procurement decisions in two-period green supply chain. *Journal of cleaner production*, 190:388–402, 2018.

[15] Stephan Dreiseitl and Lucila Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of biomedical informatics*, 35(5-6):352–359, 2002.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[17] Entso-e. Entso-e statistical database, 2009. URL https://www.entsoe.eu/data/data-portal/.

[18] Philippe Esling and Carlos Agon. Time-series data mining, 2012. *ACM Computing Surveys*, 45(1):12.

[19] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[20] Matt Gardner and Stephen Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

[21] Selfish Gene. Historical hourly weather data 2012-2017, Dec 2017. URL https://www.kaggle.com/selfishgene/historical-hourly-weather-data.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[23] Fredric Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[24] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28(4):851–881, 2014.

[25] Hyojin Jeon, Soosun Cho, et al. Brief paper: Drivable area detection with region-based cnn models to support autonomous driving. *Journal of Multimedia Information System*, 7(1):41–44, 2020.

[26] Rohit Kate. Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery*, 30(2):283–312, 2016.

[27] Jack Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3):462–466, 09 1952. URL https://doi.org/10.1214/aoms/1177729392.

[28] Sangyeob Kim, Juhyoung Lee, Sanghoon Kang, Jinsu Lee, and Hoi-Jun Yoo. A power-efficient cnn accelerator with similar feature skipping for face recognition in mobile devices. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67 (4):1181–1193, 2020.

[29] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[31] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature

learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42: 11–24, 2014.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436, 2015.

[34] Erich Lehmann and George Casella. *Theory of point estimation*. Springer, 2007.

[35] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

[36] Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29(3):565–592, 2015.

[37] Jason Lines, Sarah Taylor, and Anthony Bagnall. Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1041–1046. IEEE, 2016.

[38] Jason Lines, Sarah Taylor, and Anthony Bagnall. Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data*, 12(5), 2018.

[39] Andrew Maas, Awni Hannun, and Andrew Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, volume 30, page 3, 2013.

[40] Patrick Mannion, Jim Duggan, and Enda Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In *Autonomic Road Transport Support Systems*, pages 47–66. Springer, 2016.

[41] Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *ICML*, pages 1614–1623, 2016.

[42] Xiaoyu Mo, Yang Xing, and Chen Lv. Interaction-aware trajectory prediction of connected vehicles using cnn-lstm networks. *arXiv preprint arXiv:2005.12134*, 2020.

[43] Hugh Montgomery and Robert Vaughan. *Multiplicative number theory I: Classical theory*, volume 97. Cambridge university press, 2007.

[44] Rob Mulla. Hourly energy consumption, Aug 2018. URL https://www.kaggle.com/robikscube/hourly-energy-consumption.

[45] Tin Nwe, Tran Dat, and Bin Ma. Convolutional neural network with multi-task learning scheme for acoustic scene classification. In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1347–1350. IEEE, 2017.

[46] Henry Nweke, Ying Teh, Mohammed Al-Garadi, and Uzoma Alo. Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges. *Expert Systems with Applications*, 105:233–261, 2018.

[47] Eren Özceylan, Neslihan Demirel, Cihan Çetinkaya, and Eray Demirel. A closed-loop supply chain network design for automotive industry in turkey. *Computers & industrial engineering*, 113:727–745, 2017.

[48] Leif Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[49] Puja S Prasad, Rashmi Pathak, Vinit Kumar Gunjan, and HV Ramana Rao. Deep learning based representation for face recognition. In *ICCCE 2019*, pages 419–424. Springer, 2020.

[50] William Press. *Numerical recipes in Fortran 90: the art of parallel scientific computing: volume 2 of Fortran numerical recipes*, volume 1. Cambridge Univ. Press, 1996.

[51] Zahra Rafie-Majd, Seyed Hamid Reza Pasandideh, and Bahman Naderi. Modelling and solving the integrated inventory-location-routing problem in a multi-period and multi-perishable product supply chain with uncertainty: Lagrangian relaxation algorithm. *Computers & Chemical Engineering*, 109:9–22, 2018.

[52] Rajat Raina, Anand Madhavan, and Andrew Ng. Large-scale deep unsupervised learning using graphics processors. In *ICML*, pages 873–880, 2009.

[53] Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew Dai, Nissan Hajaj, Michaela Hardt, Peter Liu, Xiaobing Liu, Jake Marcus, Mimi Sun, et al. Scalable and accurate deep learning with electronic health records. *NPJ Digital Medicine*, 1(1):18, 2018.

[54] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, 2004.

[55] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[56] Patrick Schäfer. The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, 2015.

[57] Helmut Schmid. Probabilistic part-ofispeech tagging using decision trees. In *New methods in language processing*, page 154, 2013.

[58] Diego Silva, Rafael Giusti, Eamonn Keogh, and Gustavo Batista. Speeding up similarity search under dynamic time warping by pruning unpromising alignments. *Data Mining and Knowledge Discovery*, 32(4):988–1016, 2018.

[59] Biju Venkadath Somasundaran, Rajiv Soundararajan, and Soma Biswas. Robust image retrieval by cascading a deep quality assessment network. *Signal Processing: Image Communication*, 80:115652, 2020.

[60] Víctor Suárez-Paniagua and Isabel Segura-Bedmar. Evaluation of pooling operations in convolutional architectures for drug-drug interaction extraction. *BMC Bioinformatics*, 19(8):209, 2018.

[61] Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Jiancheng Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 2020.

[62] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In *Advances in neural information processing systems*, pages 1988–1996, 2014.

[63] Gian Susto, Angelo Cenedese, and Matteo Terzi. Time-series classification methods: Review and applications to power systems data. In *Big data application in power systems*, pages 179–220. Elsevier, 2018.

[64] Richard Sutton. Two problems with back propagation and other steepest descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*, pages 823–832, 1986.

[65] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826, 2016.

[66] Chunwei Tian, Yong Xu, Zuoyong Li, Wangmeng Zuo, Lunke Fei, and Hong Liu. Attention-guided cnn for image denoising. *Neural Networks*, 124:117–129, 2020.

[67] Chunwei Tian, Yong Xu, and Wangmeng Zuo. Image denoising using deep cnn with batch renormalization. *Neural Networks*, 121:461–473, 2020.

[68] Vincent Tinto. Dropout from higher education: A theoretical synthesis of recent research. *Review of educational research*, 45(1):89–125, 1975.

[69] Mikhail Tsetlin. Finite automata and models of simple forms of behaviour. *Russian Mathematical Surveys*, 18(4):1–27, 1963.

[70] Ji Wan, Dayong Wang, Steven Chu Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, and Jintao Li. Deep learning for content-based image retrieval: A comprehensive study. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 157–166. ACM, 2014.

[71] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119: 3–11, 2019.

[72] Xuerong Wang, Mingming Leng, Jingpu Song, Chunlin Luo, and Sunyuen Hui. Managing a supply chain under the impact of customer reviews: A two-period game analysis. *European Journal of Operational Research*, 277(2):454–468, 2019.

[73] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from

scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[74] Cort Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[75] Wei Xu, Tingting Zheng, and Ziang Li. A neural network based forecasting method for the unemployment rate prediction using the search engine query data. In *2011 IEEE 8th International Conference on e-Business Engineering*, pages 9–15. IEEE, 2011.

[76] Qiang Yang and Xindong Wu. 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, 5(04):597–604, 2006.

[77] Shihao Yang, Mauricio Santillana, John Brownstein, Josh Gray, Stewart Richardson, and SC Kou. Using electronic health records and internet search information for accurate influenza forecasting. *BMC infectious diseases*, 17(1):332, 2017.

[78] Joe Young. Rain in australia, Dec 2018. URL https://www.kaggle.com/jsphyg/weather-dataset-rattle-package.

[79] Xianxian Zeng, Yun Zhang, Xiaodong Wang, Kairui Chen, Dong Li, and Weijun Yang. Fine-grained image retrieval via piecewise cross entropy loss. *Image and Vision Computing*, 93:103820, 2020.

[80] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.