ABSTRACT  The detection of periodicity in a time series is considered a challenge in many research areas. 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. Different methods, including deep neural networks, have been proposed across many applications to find suitable solutions to the period length extraction problem. This article proposes a convolutional neural network (CNN) based period classification algorithm, named PCA, to detect the dataset periods. In particular, assuming that a data stream contains periodical features, the PCA utilizes historical labeled data as training material and classifies new instances accordingly based on their periods. Its performance has been tested on both synthetic and real-world periodic time series data (PTSD) with very encouraging results. In particular, we have observed that the PCA is capable of achieving 100% accuracy in the case of low noise PTSD. Even the training of the PCA is not economical if the data do not contain much noise, it still demonstrates high performance on both synthetic and real-world datasets. Besides, we have shown that our new algorithm can capture the relationship between the shape of the waves and the target period, which is significantly different from the classical methods that mainly focus on the wave’s amplitude.

INDEX TERMS  Convolutional neural networks, deep learning, period extraction, time series analysis.

I. INTRODUCTION

A. 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. For example, knowing the period length from general public health data can enable pharmaceutical organizations to choose the delivery season of their flu vaccines [1] in a timely manner. 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 their running budget. The ideal scenario is thus one in which the available resources match the demand with neither surplus nor shortage.

In the past decades, several decision optimization algorithms have been proposed [2]. 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 article 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. In this article, we limit our discussions to the dataset containing periodic features.

This article is an extended version of the preliminary conference article [3] in which we proposed the original period classification algorithm.

B. RELATED WORK

In the literature of machine learning, one of the most challenging problems has always be Time Series Classification (TSC) [4], [5]. In more recent years, the availability of real-world data has significantly increased [6]; hundreds of TSC algorithms have been proposed with different design philosophies [7]. As a result of lacking integrity, most TSC
problems require some level of human involvement [8]. Meanwhile, if a classification problem takes ordered data, it can be transformed into a TSC problem [9]. TSC problems are not only challenging but also have high diversity. The related problems include but not limited to:

1) electronic health records [10],
2) human activity recognition [9], [11],
3) acoustic scene classification [12],
4) cyber-security [13].

Researchers have proposed many algorithms to achieve required accuracy level for variant tasks [7]. A classical method is to train a K-th nearest neighbor (KNN) classifier using a distance function [14]. Specifically, the Dynamic Time Warping (DTW) distance function combined with a KNN classifier appears to be a very strong baseline [7]. Different distance functions have been tested empirically, however these studies showed no statistical significant differences between individual distance functions [14]. On the other hand, researchers have demonstrated that the ensemble of KNN classifiers using different distance measures outperforms an individual classifier. Therefore, recent advance is heavily focused on ensemble techniques [15]–[22]. Most of these works outperform the KNN-DTW approach [7] and share one common technique, transforming the raw data into another feature dimension (for instance, the shapelets transformation [17] can encode raw data into the feature dimension defined by DTW [20]).

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 model called COTE (Collective Of Transformation-based Ensembles) [15] which takes advantage from 35 independent classifiers. COTE ensembles not only different distance measurements but also different feature transformations. Then, Lines et al. [18], [23] 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 up rise of deep learning models for TSC [24].

The Fourier transform and DFT [25] have also 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.

C. MAIN CONTRIBUTION

The main contribution of this article involves the period length extraction problem. This article proposes the Period Classification Algorithm (PCA) which 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. Our empirical study shows the PCA performs quite well against both synthetic and real-world data. Moreover, the study also shows the algorithm can learn the relationship between the changing patterns of the waves and the period targets. This exclusive feature makes the algorithm quite different from the classical ones that mainly focus on the amplitudes of the waves.

D. ARTICLE ORGANIZATION

The remaining of this article is organized as follows. Section II first introduces the preprocessing algorithm. This algorithm converts the raw data samples to a red-green-blue (RGB) image. Then, Section III presents the proposed model in detail. Section IV presents the performance measurements, whereas Section V provides the training policy along with problems during training. Afterward, Section VI introduces the synthetic data generation and discusses the algorithm’s performance on them. Furthermore, Section VII contains insights that relate to the selected real-world data, and Section VIII demonstrates the PCA’s performance on real-world data. We draw the conclusion in Section IX and list the potential future work in Section X.

II. PREPROCESSING ALGORITHM

The PCA algorithms can be partitioned into two modules: one is the preprocessing algorithm and the other one is the neural network for classification. The second module is mainly implemented with the convolutional neural networks (CNN).

Considering that the CNN models were initially proposed for the algorithms that have images as inputs [26], Algorithm 1 is introduced to transform raw PTSDs into image-like representations. The algorithm has two super-parameters: the desired width and height of the output image representations, and it converts the raw PTSDs into the image of the required size.

The algorithm starts by marking the data points and connecting them using line segments on a white canvas. Notice that, given various PTSDs, the generated raw graphs may have different sizes as they may have different lengths of periods and amplitudes. To facilitate the latter’s processing, we pad the images to make them have the same sizes. The padded images will have most of the pixels white which contain no information. This will effect the latter neural network training and lower the data processing efficiency. To fix the problem, we use Algorithm 1 to partition rawImage into pieces of the same desiredWidth and desiredHeight. Then we stack all the pieces up by taking the sum of the pixel values mod 255. Note that the mod operation ensures the data validity by keeping the RGB values between 0 and 255.
TABLE 1. The i-th feature extraction layer (i = 1, 2, ⋅⋅⋅, 6).

| Layer Type          | Hyper-parameters                                      |
|---------------------|--------------------------------------------------------|
| Convolution         | $2^{i+1}$ kernels of size $(3, 1)$ and $2^{i+1}$ kernels of size $(1, 3)$ |
| LeakyReLU Activation| $\alpha = 0.1$                                        |
| Maximum Pooling     | Pooling size: $(2, 2)$                                 |
| Dropout             | Rate: 0.25                                             |

We add nonlinearities using Leaky ReLU function with $\alpha = 0.1$. All pooling layers have the pooling size $2 \times 2$, and we apply the dropout method with the dropout rate 0.25 to avoid the overfitting problem.

Note that, instead of applying a convolution layer of a $3 \times 3$ square kernel, we used two parallel layers with the kernels of sizes $3 \times 1$ and $1 \times 3$, respectively. The design, originally proposed by Szegedy et al. [27], has a similar performance as the classical CNN but contains fewer parameters. In our case, the kernels need $3 + 3 = 6$ parameters, which has a similar performance as the regular kernel implementation that requires $3 \times 3 = 9$ parameters. In principle, fewer parameters require less training samples and training iterations and result in a better generalization.

The leaky ReLU function simply returns the input value if it is greater than zero; otherwise, it returns $\alpha$ times the input. Compared to the traditional ReLU function, the design introduces nonlinearities without dropping information when the input is less than zero and maintains a high computation efficiency.

The introduction of pooling layers removes the irrelevant features and keeps the most important ones. A stack of conv-pooling, then, can extract low-level features, like edges and corners, while the more complicated features including heart rate diagrams periods are composed from those low-level features and will be captured by the layers closer to the output. The extracted features are sent to the classifier module consisting of two fully connected layers, which makes the period length prediction.

The dropout design removes the kernels of the CNN with the probability of the dropout rate. This forces the model prediction not to rely on any particular kernels and their outputs, which could cause serious overfitting problem [28].

Fig. 1 visualizes the part of the model that follows the stack of the conv-pooling layers. A well designed model should have sufficient independent features after the completion of the training process. Assuming that each neuron at the last conv-pooling layer represents a single feature (that is, whether a certain pattern appears in the input image), there is no need for us to keep the spatial information. Thus, we can flatten the outputs to meet the input size requirement of the classifier module without losing much information.

In the classifier module, we still apply the leaky ReLU function for nonlinearities. We also use dropout with dropout rate 0.25 to alleviate the overfitting problem.
TABLE 2. Deep convolutional model architecture.

| Layer Type                  | Hyper-parameters |
|-----------------------------|------------------|
| Six feature extraction layers | See Table 1      |
| Platten Layer               | N/A              |
| Fully Connected Layer       | # neurons: 256   |
| LeakyRelu Activation        | α: 0.1           |
| Dropout                     | Rate: 0.25       |
| Fully Connected Layer       | # neurons: 64    |
| LeakyRelu Activation        | α: 0.1           |
| Dropout                     | Rate: 0.25       |
| Softmax Activation          | N/A              |

From the math view, the fully connect layers (without activation functions) can be considered as an affine transformation:

\[ f_j : \mathbb{R}^{D_j} \rightarrow \mathbb{R} \quad \text{for} \quad j = 1, 2, \ldots, n, \]

where \( D_j \) is the output size of the previous layer and \( n \) is the size of the current one. Each \( f_j \) is defined as:

\[ f_j(\text{features}) = \sum_{i=1}^{D_j} (w_i \cdot \text{feature}_i + b_i). \quad (1) \]

and a fully connected layer is the concatenation of the functions \( f_1, f_2, \ldots, f_n \). Remark that in order to increase the model’s capacity, we also add the leaky ReLu functions for nonlinearities.

At last, the classification prediction is made in the output layer. The layer is similar to the fully connected layer but uses the softmax function \([29]\) (defined in Equation (2)) instead of the activation function. Notice that the output layer has the number of outputs same as the number of classes to be classified.

The output of the softmax function can be seen as the probability mass function of a multinomial distribution. When performing the prediction task, the class corresponding to the highest probability value is selected. We summarize the entire neural network model in Table 2 alone with the hyper-parameters we used for the experiments.

\[
\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \quad (2)
\]

The main task to be solved in this article is to classify the PTSDs according to the length of their period. Notice that, the value of the target labels solely depends on the period length. As a result, the choice of the unit time length is not a concern as long as all input data sets share the same unit.

Moreover, to facilitate the model implementation, the labels are converted into the one-hot representation instead of the integers. For instance, assuming that the possible period length choices are from zero to three, then \((1, 0, 0, 0)\) denotes the period length zero and \((0, 0, 1, 0)\) represents the length three.

IV. PERFORMANCE MEASUREMENT

The performance of deep models can be measured by loss and accuracy. The accuracy value measures deep models from a macro perspective. Its calculation is based on a set of instances, and the result simply tells the correctly classified percentage of the set. It provides a general understanding of how well or poor a model performs; however, it lacks details in errors made for each instance.

Unlike accuracy, loss is not a percentage measurement. Loss is a number telling how far a prediction was on a single instance. The same deep model may obtain different loss values by predicting two independent instances. So the goal is to find a set of weights and biases that minimize loss values for all possible instances.

We utilize cross entropy \([30]\) defined by Equation (3). 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 not designed to produce one certain single label but a possibility distribution among all possible labels. For a binary classification task, ground truth labels include positive \(<1.0, 0.0>\) and negative \(<0.0, 1.0>\). A CNN could produce a prediction like \(<0.7, 0.3>\). This signifies that the CNN is 70% certain the instance is positive. In case of the same CNN produced another prediction \(<0.9, 0.1>\) for another instance which shares the same ground truth label, then Equation (3) measures a greater loss value for the prediction on the earlier instance than the later one.

\[
\text{CE}(y_i, \hat{y}_i) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i \quad (3)
\]

V. 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 \( \text{model}(x_i, \theta) \), where \( \theta \) represents the parameters. Therefore, we can treat loss functions \( f_{\text{loss}}(y_i, \hat{y}_i) \) as objective functions \( \arg\min_{\theta} f_{\text{loss}}(y_i, \text{model}(x_i, \theta)) \). Thus, we require strategies to optimize \( \theta \) in order to minimize the loss value.

Optimizer Adam \([31]\) was utilized to train our deep model. The name “Adam” stands for adaptive moment estimation. It demonstrated state-of-the-art performance among many deep model optimizers, in terms of training speed and result.

Adam combines the innovation of Adagrad \([32]\) and Momentum \([33]\) into one optimizer. It takes two hyperparameters, \( \beta_1 \) for moment decay and \( \beta_2 \) for learning rate decay. The authors of Adam reported state-of-the-art performance on many application when \( \beta_1 = 0.9 \) and \( \beta_2 = 0.999 \).
Therefore, all experiments mentioned in Sections VI and VIII were trained by the Adam optimizer with the parameter configuration proposed by the authors of Adam.

Essentially, the training phase is a repeating process, and it can be broken down into iterations. In deep learning literature, one training iteration is usually called an epoch. Each epoch starts by randomly selecting one or several instances \( x_i \) from the training set. These selected \( n \) instances form a batch \( X \). Deep models then produce prediction labels for the first batch and sum their loss values. This part of computation also refers as forward propagation. Then the backward propagation is done by the optimizer’s parameter update policy. Accordingly, Adam calculates the following:

\[
g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta} \text{loss}(y_i, \text{model}(x_i, \theta))
\]

\[
m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t
\]

\[
v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2
\]

where the \( \theta \) represents the collection of all trainable parameters. Then \( m_t \) 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

\[
\hat{m}_t = \frac{m_t}{1 - \beta_1}
\]

\[
\hat{v}_t = \frac{v_t}{1 - \beta_2}
\]

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

\[
\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t
\]

These selected instances will not be used in the next batch of training. One epoch completes when every instance is selected. Afterward, all instances will be randomly shuffled, and we repeat the above procedure.

Besides the optimizer and the CNN architecture, other factors could affect the training result. For instance:

1) The noise component must not dominate the real PTSD trend. Specifically, the PTSD’s amplitude should be obvious for human eyes after adding the noise component.

2) The training set must have a strong connection with its labels. Otherwise, this situation could result in a large gap between training and testing evaluations. The model could perform very well on training instances by memorizing each of them; however, this does not guarantee that the model is generalized effectively at the evaluation stage, as reported by Zhang et al. [34].

3) Usually, a training set should include all possible labels from the problem domain. If a deep model is deployed to a working environment that contains unseen labels, it can only classify the instance as the closest label from the training set.

The listings above are essential but not complete. Some factors are not listed but can further push the training result away from the potential optimum.

VI. EXPERIMENTS ON SYNTHETIC DATASETS

The core problem in this article is to classify PTSDs by their period length, and experiments are designed around it. The experiments we performed in this section consist of two parts: 1) we test how resistant to the noise our algorithm is. 2) we show our algorithm can capture the relationship between the wave’s shape and the target period while most classical algorithms cannot.

Due to the nature of deep models, we decided to use the Graphic Processing Unit (GPU) instead of the traditional Central Processing Unit (CPU) for the experiments. More specifically, two Gigabyte GeForce GTX 1080 graphics cards accelerated the process tremendously. In terms of software, the implementation of the deep model was helped by Python library Keras version 2.2.2. Additionally, Keras was using TensorFlowGPU version 1.10.0 as its backend. As a result of limitations in computational power, all experiments are configured with \( \text{desiredWidth} = \text{desiredHeight} = 250 \) during the preprocessing phase (see Algorithm 1). We can set these values higher if extra computing resources are available. Interestingly, a higher resolution provides more periodic features when polynomial components dominate. Besides, the batch size was set to be 32 whereas the initial learning rate was 0.0003.

The first part of our experiments were aimed at determining the threshold of the PCA. The data were generated through Algorithm 2 with the Gaussian noise added. The parameter \( \text{periodLength} \) determines the desired length of a single period measured by time indices. Then, the parameter \( \text{window} \) represents the upper bound of time indices, whereas \( \text{sampleSize} \) decides the number of data samples for a single period. In addition, the callable parameter \( \text{noiseFunc()} \) can be any possible noise function.

The algorithm starts by filling the array \( X \) by spaced time indices between zero and \( \text{periodLength} \). They are representing time indices of the first period. Then temporary variable array \( Y_p \) is populated by samples from a standard normal distribution. The algorithm now has the first period and can keep copying \( Y_p \) and extended \( X \) until there is no enough room for another period.

The experiments had two possible ground truth labels: 2 and 7, and 2,000 images are generated for each label, respectively. Altogether 4,000 images were generated. After shuffling, 3,600 (90%) were selected as training instances, and the rest 400 (10%) were saved as testing instances. We did not use the same image multiple times across different experiment runs.

That is, we first attempted to determine the maximum noise for which the PCA could tolerate. So we tested the algorithm by setting the standard deviation \( \sigma \) of the Gaussian noise from...
Algorithm 2 Noisy Periodic Time Series Data Generation

Input: periodLength, window, sampleSize, NoiseFunc()
Output: array of time indices, array of PTSD samples

1. \( X, Y_p, Y := \text{empty array} \)
2. for \( i = 0; i < \text{samples}; i = i + 1 \) do
3. \( \text{Append} \ (\text{periodLength/samples} \times i) \) to \( X \)
4. \( \text{sample} := \text{StandardNormalDistribution()} \)
5. \( \text{Append} \ \text{sample} \) to \( Y_p \)
6. end for
7. \( \text{repeats} := \text{floor}(\text{window}/\text{periodLength}) \)
8. \( \text{nextPeriod} := \text{copy} \ X \)
9. for \( i = 0; i < \text{repeats}; i = i + 1 \) do
10. \( x := \text{the last element of} \ X \)
11. \( \text{increase every element of} \ \text{nextPeriod} \) by \( x \)
12. \( X := \text{concatenate}(X, \text{nextPeriod}) \)
13. \( Y := \text{concatenate}(Y, Y_p) \)
14. end for
15. for \( i = 0; i < \text{size}(X); i = i + 1 \) do
16. \( Y[i] += \text{NoiseFunc}(X[i]) \)
17. end for
18. return \( X, Y \)

FIGURE 2. The periodic function \( g(x) = 0.7 \times \text{mod}(x, 2) \) added with no noise.

FIGURE 3. The periodic function \( g(x) = 0.7 \times \text{mod}(x, 2) \) added to gaussian noise configured with \( \mu = 0 \) and \( \sigma = 1 \).

zero to ten. Every time we increased the deviation, we trained Algorithm 2 from scratch.

Fig 2 and Fig 3 present two raw PTSD generated by Algorithm 2. However, PTSD plotted on Fig 2 is not padded with any noise whereas Fig 3 is padded with Gaussian noise (with \( \mu = 0 \) and \( \sigma = 1 \)). Besides, PCA is deployed on the same generated dataset for ten times. Fig 4 demonstrates the averaged result from each run. This plot shows the PCA can tolerate high-level of Gaussian noise, which is the motivation for the next set of experiments.

The next set of experiments uses polynomial noise functions. Our goal here is to determine the highest polynomial degree as well as the highest coefficient variance for which the PCA could provide a correct answer. The listings below provide the degree values of polynomial noise functions along with the variance of the polynomial coefficient, respectively, for implementation of these experiments:

1) polynomial degree \{0, 1, 2, 3, 4\}
2) variance of polynomial coefficient \{0.5, 1.0, 1.5, 2.0, 3.0, 4.0\}

By choosing one from each listing and combine, we can obtain 30 configurations. For each configurations, we have performed 10 runs and averaged the results for performance measurement. As mentioned previously, images were randomly generated by the methods introduced in Algorithm 2.

Thus, the results were not biased by a random sample or by random deep model initialization.

Fig 5 presents the collected results of the second experimental set. The x-axis indicates the degree of the polynomial...
components, whereas the y-axis indicates the average out-of-sample accuracy from 10 identical runs. Each line represents a different coefficient variances configuration, as mentioned earlier. Specifically, the PCA showcased steady performance when polynomial degrees were below 3. The average accuracy reading was above 90%.

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’ periodic components were generated with an equal amplitude. Therefore, periodic features vanished in those PTSDs composed with higher degrees polynomial.

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. Since 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 of the waveform decreases, and the sawtooth is evident within the period of sawtooth wave as the target:

$$x_h(g(x)) = 0.005(x-10)(x-17)$$

Fig. 6 has noise polynomial $h(x) = 0.005(x-10)(x-17)$. 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. 6 plots a noisy polynomial with a degree of 3. Its periodic feature is not yet blurred. However, Fig. 7 demonstrates what can happen when the polynomial degree continues to increase. The coefficient variance of both waveforms are at the same level. In comparison with Fig. 6, the sawtooth is unrecognizable across the domain.

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.

![Figure 6](image1.png)

**FIGURE 6.** Periodic function $g(x) = 0.7 \times \text{mod}(x, 2)$ added to a polynomial noise function $h(x) = 0.005(x-10)(x-17)$. Periodic features are still recognizable when the polynomial degree is 3.

![Figure 7](image2.png)

**FIGURE 7.** Periodic function $g(x) = 0.7 \times \text{mod}(x, 2)$ added to a polynomial noise function $h(x) = 0.005(x-10)(x-17)(x-5)$. The periodic feature is less recognizable than the waveform from Fig. 6.

| Waves for mixtures | CNN Acc | Fourier Transform Acc |
|--------------------|---------|-----------------------|
| Sawtooth, Square   | 100.0%  | 58.0%                 |
| Sawtooth, Square, Reversed Sawtooth | 99.5% | 42.8% |
| Sawtooth, Square, Reversed Sawtooth, Sine | 99.6% | 35.3% |

The second part of the experiments are performed to show that our new algorithm is able to detect the period of the data with certain changing features. For instance, considering an input data which is a mixture of sawtooth, square and sine waves. Our algorithm can learn to detect the period of a sawtooth wave always, which is very different from those classical methods that often output the period of the wave having the greatest amplitude.

We trained and tested our algorithm over the dataset generated through the following procedure, which mixes the waves of the various shapes and the amplitudes and selects the period of sawtooth wave as the target:

**Algorithm 3** The Mixture of the Waves With Various Changing Patterns and Amplitudes

**Input:** periods, periodicFuncs, windowSize, sampleStride

**Output:** A composite periodic function plot

1. $X :=$ evenly spaced numbers over the closed interval $0$, windowSize with stride equal to sampleStride.
2. $Y :=$ an empty array.
3. for $i = 0$; $i < \text{length(periodicFuncs)}$; $i = i + 1$ do
4. $amp_i :=$ a random integer from closed interval $[1, 6]$ with respect to uniform distribution
5. $Y = amp_i \times \text{periodicFuncs}[i](X, \text{periods}[i])$
6. end for
7. image := plot($X, Y$)
8. return image

We listed the test results of our algorithm as well as the Fourier Transformation in Table 3. The table indicates that our algorithm successfully captures the relation between the shape of the waves and the target period. At the same time, the Fourier transformation can only output the period of the
wave having the greatest amplitude and continuously fails during the test.

VII. REAL-WORLD DATASETS SELECTION

As mentioned, Algorithm 2 provides datasets that have balanced classes but different noise levels or noise types. 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 [35] (HHWD, 12,960 images, 5 classes)
2) Weather Madrid 1997-2015 data [36] (LEMD, 1,539 images, 18 classes)
3) Entsoe hourly energy consumption [37] (ENTSOE, 21,240 images, 10 classes)
4) PJM hourly energy consumption [38] (PJMHEC, 96,135 images, 5 classes)
5) Rain in Australia [39] (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. 8 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 consumed during working days and less during the weekend.

Unlike the electricity usage data from the PJMEHCEC 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. 9 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.

VIII. EXPERIMENTS ON REAL-WORLD DATASET

Unlike generated datasets, real-world datasets usually have an unbalanced label distribution. 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.

We calls instances that are correctly classified with respect to the ground truth label true positives (TPs) and true negatives (TNs). Positive instances classified as negative are called false negatives (FNs), while negative instances classified as positive are called false positives (FPs).

As mentioned, the accuracy measurement is not sufficient for unbalanced datasets. Thus, we use the F1 score, which is defined as follows:

\[
\text{precision} = \frac{TP}{TP + FP} \quad (10)
\]
\[
\text{recall} = \frac{TP}{TP + FN} \quad (11)
\]
\[
\text{F1-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (12)
\]

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. 10 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. 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 of batches, and consequently, more model optimization.
TABLE 4. Test result of period classification algorithm.

|                | Test loss | Test accuracy |
|----------------|-----------|---------------|
| HHWD           | 1.13E-06  | 100.0%        |
| LEMD           | 2.64E-01  | 92.3%         |
| WeatherAUS     | 2.85E-01  | 92.4%         |
| ENTSOE         | 4.50E-03  | 99.8%         |
| PJMHEC         | 1.15E-03  | 99.9%         |

FIGURE 12. Validation accuracy of period classification algorithm on real-world datasets.

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 the 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. 11 and 12 present the results of applying the PCA on the validation dataset. Generally, Figs. 12 and 10 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. 11 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.

There are 10% of samples saved for the final testing of the PCA. Note that we have made the label distributions of the training and testing datasets the same so that their data distribution should be similar. Notice that, unlike a traditional configuration that usually chooses an 80%/20% dataset separation for training and testing, we have chosen a higher ratio 90%/10%. This configuration could make the training and testing data distribution less similar to each other and thus make the task more challenging. Readers should expect a better performance of our algorithm if 80%/20% dataset separation is adopted. Table 4 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. 11 and 12, which indicates that the model is generalized enough to fit both training and test data with acceptable performance.

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.

IX. CONCLUSION

Our experiments demonstrate that the PCA is suitable for real-world periodic classification tasks. Unlike those classical methods, we have shown that the PCA can perform well while the target period is given by the wave that changes in a particular pattern instead of having the largest amplitude. Its performance may vary according to the quantity of data and the noise component. If the noise component follows the Gaussian distribution, its effectiveness on the PCA's performance is limited. However, if the noise component is a polynomial function and the polynomial trend is too large, then the data may not appear periodic in the images. Moreover, we have observed that the PCA's performance approaches 100% as the training data size increases. This hints that our model has the enough capacity to fully capture the relationship between the input data and the target periods.

X. FUTURE WORK

Anticipated future work involves the following areas:

- **Applications**: We tested the PCA on the synthetic and real-world datasets. Performance analysis revealed that the PCA demonstrated high accuracy. We believe 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. The hyper-parameters used in this article can be further optimized for better performance.

REFERENCES

[1] S. Yang, M. Santillana, J. S. Brownstein, J. Gray, S. Richardson, and S. C. Kou, “Using electronic health records and Internet search information for accurate influenza forecasting,” *BMC Infectious Diseases*, vol. 17, no. 1, p. 332, Dec. 2017.

[2] K. Dey and S. Saha, “Influence of procurement decisions in two-period green supply chain,” *J. Cleaner Prod.*, vol. 190, pp. 388–402, Jul. 2018.

[3] H. Zhang, A. Nayak, and H. Lu, “Periodic time series data classification by deep neural network,” in *Proc. 26th Int. Conf. Telecommun. (ICT)*, Apr. 2019, pp. 319–323.

[4] Q. Yang and X. Wu, “10 challenging problems in data mining research,” *Int. J. Inf. Technol. Decis. Making*, vol. 5, no. 4, pp. 597–604, Dec. 2006.

[5] P. Esling and C. Agon, “Time-series data mining, 2012.” *ACM Comput. Surv.*, vol. 45, no. 1, p. 12, 2012.
[6] D. F. Silva, R. Giusti, E. Keogh, and G. E. A. P. Batista, “Speeding up similarity search under dynamic time warping by pruning unpromising alignments,” Data Mining Knowl. Discovery, vol. 32, no. 4, pp. 988–1016, Jul. 2018.

[7] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, “The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances,” Data Mining Knowl. Discovery, vol. 31, no. 3, pp. 606–660, May 2017.

[8] M. Längkvist, L. Karlsson, and A. Louifi, “A review of unsupervised feature learning and deep learning for time-series modeling,” Pattern Recognit. Lett., vol. 42, pp. 11–24, Jun. 2014.

[9] H. F. Nweke, Y. W. Teh, M. A. Al-garadi, and U. R. Alo, “Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges,” Expert Syst. Appl., vol. 105, pp. 233–261, Sep. 2018.

[10] A. Rajkomar et al., “Scalable and accurate deep learning with electronic health records,” NPJ Digit. Med., vol. 1, no. 1, p. 18, 2018.

[11] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, “Deep learning for sensor-based activity recognition: A survey,” Pattern Recognit. Lett., vol. 119, pp. 3–11, Mar. 2019.

[12] T. L. Nwee, T. H. Dat, and B. Ma, “Convolutional neural network with multi-task learning scheme for acoustic scene classification,” in Proc. Asia-Pacific Signal Inf. Process. Assoc. Annula Summit Conf. (APSIPA ASC), Dec. 2017, pp. 1347–1350.

[13] G. Susto, A. Cenedese, and M. Terzi, “Time-series classification methods: Review and applications to power systems data,” in Big Data Application in Power Systems. Amsterdam, The Netherlands: Elsevier, 2018, pp. 179–220.

[14] J. Lines and A. Bagnall, “Time series classification with ensembles of elastic distance measures,” Data Mining Knowl. Discovery, vol. 29, no. 3, pp. 565–592, May 2015.

[15] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, “Time-series classification with COTE: The collective of transformation-based ensembles,” IEEE Trans. Knowl. Data Eng., vol. 27, no. 9, pp. 2522–2535, Sep. 2015.

[16] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, “Classification of time series by shapelet transformation,” Data Mining Knowl. Discovery, vol. 28, no. 4, pp. 851–881, Jul. 2014.

[17] A. Bostrom and A. Bagnall, “Binary shapelet transform for multiclass time series classification,” in Proc. Int. Conf. Big Data Anal. Knowl. Discovery, Springer, 2015, pp. 257–269.

[18] J. Lines, S. Taylor, and A. Bagnall, “HIVE-COTE: The hierarchical vote collective of transformation-based ensembles for time series classification,” in Proc. IEEE 16th Int. Conf. Data Mining (ICDM), Dec. 2016, pp. 1041–1046.

[19] P. Schäfer, “The BOSS is concerned with time series classification in F. J. Harris, “On the use of windows for harmonic analysis with the discrete Fourier transform,” Proc. IEEE, vol. 66, no. 1, pp. 51–58, Jan. 1978.

[20] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. NIPS, 2012, pp. 1097–1105.

[21] Z. Wang, W. Yan, and T. Oates, “Time series classification from scratch with deep neural networks: A strong baseline,” in Proc. Int. Joint Conf. Neural Netw. (IJCNN), May 2017, pp. 1578–1585.

[22] F. J. Harris, “On the use of windows for harmonic analysis with the discrete Fourier transform,” Proc. IEEE, vol. 66, no. 1, pp. 51–58, Jan. 1978.

[23] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. NIPS, 2012, pp. 1097–1105.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in Proc. CVPR, 2016, pp. 2818–2826.

[25] V. Tinto, “Dropout from higher education: A theoretical synthesis of recent research,” Rev. Educ. Res., vol. 45, no. 1, pp. 89–125, Mar. 1975.

[26] A. Martinis and R. Astudillo, “From softmax to sparsemax: A sparse model of attention and multi-label classification,” in Proc. ICML, 2016, pp. 1614–1623.

[27] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[28] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. ICLR, 2015, pp. 1–15.

[29] J. Duch, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” J. Mach. Learn. Res., vol. 12, pp. 2121–2159, Jul. 2011.

[30] R. Sutton, “Two problems with back propagation and other steepest descent learning procedures for networks,” in Proc. 8th Annu. Conf. Cogn. Sci. Soc., 1986, pp. 823–832.

[31] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, “Understanding deep learning requires rethinking generalization,” in Proc. ICLR, 2017, pp. 1–5.

[32] S. Gene. (Dec. 2017). Historical Hourly Weather Data 2012–2017. [Online]. Available: https://www.kaggle.com/selfishgine/historical-hourly-weather-data

[33] J. Castro. (Sep. 2016). Weather Madrid 1997–2015. [Online]. Available: https://www.kaggle.com/juliannism/weather_madrid_lenmd_1997_2015.csv

[34] Entso-e. (2009). Entso-E Statistical Database. [Online]. Available: https://www.entsoe.eu/data/data-portal/

[35] R. Mulla. (Aug. 2018). Hourly Energy Consumption. [Online]. Available: https://www.kaggle.com/robiskube/hourly-energy-consumption

[36] J. Young. (Dec. 2018). Rain in Australia. [Online]. Available: https://www.kaggle.com/jophy/weather-dataset-rattle-package


HAOYUE LU (Member, IEEE) received the B.Sc. degree (joint) in computer science and mathematics and the master’s degree in computer science from the University of Ottawa, Canada, in 2017 and 2019, respectively. He joined the University of Ottawa in 2013, where he is currently a Research Associate. His research interests include artificial intelligence and network structures.

AMIYA NAYAK (Senior Member, IEEE) received the B.Math degree in computer science and combinatorics and optimization from the University of Waterloo, Canada, in 1981, and the Ph.D. degree in systems and computer engineering from Carleton University, Canada, in 1991. He is currently a Full Professor with the School of Electrical Engineering and Computer Science, University of Ottawa. His research interests include software-defined networking, mobile computing, wireless sensor networks, and vehicular ad hoc networks.

He has over 17 years of industrial experience in software engineering, avionics and navigation systems, and simulation- and system-level performance analysis. He has served on the Editorial Board of several journals, including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the International Journal of Parallel, Emergent and Distributed Systems, the Journal of Sensor and Actuator Networks, and the EURASIP Journal of Wireless Communications and Networking. He is currently on the Editorial Board of the IEEE INTERNET OF THINGS JOURNAL, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE Open Journal of the Computer Society, Future Internet, and the International Journal of Distributed Sensor Networks.