3-2018

VIRTUALIZED CLOUD PLATFORM MANAGEMENT USING A COMBINED NEURAL NETWORK AND WAVELET TRANSFORM STRATEGY

Chunyu Liu
California State University – San Bernardino, 005398220@coyote.csusb.edu

VIRTUALIZED CLOUD PLATFORM MANAGEMENT USING A COMBINED NEURAL NETWORK AND WAVELET TRANSFORM STRATEGY

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Chunyu Liu

March 2018

VIRTUALIZED CLOUD PLATFORM MANAGEMENT USING A COMBINED NEURAL NETWORK AND WAVELET TRANSFORM STRATEGY

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Chunyu Liu

March 2018

Approved by:

Dr. Tong Lai Yu, Committee Chair, School of Computer Science and Engineering

Dr. Kerstin Voigt, Committee Member

Dr. Yunfei Hou, Committee Member

ABSTRACT

This study focuses on implementing a log analysis strategy that combines a neural network algorithm and wavelet transform. Wavelet transform allows us to extract the important hidden information and features of the original time series log data and offers a precise framework for the analysis of input information. While neural network algorithm constitutes a powerful nonlinear function approximation which can provide detection and prediction functions. The combination of the two techniques is based on the idea of using wavelet transform to denoise the log data by decomposing it into a set of coefficients, then feed the denoised data into a neural network. The experimental outputs reveal that this strategy can have a better ability to identify the patterns among problems in a log dataset, and make predictions with a better accuracy. This strategy can help the platform maintainers to adopt corresponding actions to eliminate risks before the occurrence of serious damages.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Tong Lai Yu, for all his guidance and help during the project. His patient guidance, enthusiastic encouragement, and scholarly advice have helped me to a great extent to accomplish this research work. He is not only a professor for me, he is more like a close friend and a respected person that worth learning in life.

I would also like to extend my thanks to Dr. Yunfei Hou and Dr. Kerstin Voigt for being the committee members. Thank you for your valuable advice and support.

I am also very thankful for the help of the Department of Computer Science at California State University, San Bernardino. Especially, my graduate advisor Dr. Josephine Mendoza, who gave me great help during the graduate study.

Finally, I want to thank my family for the support and encouragement throughout my study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

STATEMENT OF PURPOSE

The primary objective of this project is to design a virtualized cloud platform management program that has a better pattern recognition, noise tolerance, and prediction accuracy. In this project, an attempt has been made to find an improved neural network model for log analysis by combining the wavelet transform and a neural network. This project results from a virtualized cloud platform, by which the experiment log data sets are collected. Based on the log dataset features, we design and implement the new model. The experimental results demonstrate the proposed model is very effective and accurate, it has a better ability for feature attraction and noise tolerance than conventional neural networks, which can help the cloud platforms to improve risks detection and prediction.

CHAPTER TWO

INTRODUCTION

Background

The rise of cloud computing has made the service maintenance more complex than ever. Servers of a cloud platform are facing various internal and external risks. An effective risk detection, analysis and prediction method can help the platform to locate risks, identify problems, and make predictions. To satisfy the requirements of cloud service maintenance, auditing logs in cloud servers is the most efficient way for diagnosing the system and software problems. However, for large systems, various components generate large amounts of log information in real time [1]. We have to extract useful information from huge amounts of data to identify problems.

Many attempts have been made to find the most appropriate way for log-based fault prediction. They are different from the ways of implementation. Here we list a part of existing methods.

PCRE (Perl Compatible Regular Expressions)

The PCRE technique is a library that supports special separator technique and regular expression, it is a normalizing technique used for extracting useful information from unstructured data [2].

FFDA (Field Failure Data Analysis)

The FFDA method is based on grouping system logs by checking specific contents in log messages. The key step is filtering out entries that are not useful

and redundant error entries from the log [3]. It attempts to group the error entries related to the same fault manifestation, and isolates accidental patterns.

Neural network

The usage of neural network methods helps to build an intelligent log analyzer that can detect known and unknown network intrusions automatically. The log analyzer is trained with multi-layer neural network algorithms for the prediction tasks [4].

Methods mentioned above have various limitations. For example, PCRE can only identify isolated problems, but not the correlations among them. The FFDA doesn't have the ability of self-learning and adaptability. A neural network, as a mature and effective tool, has many advantages, it has rigorous derivation process, solid theoretical basis, strong versatility, and clear physical concepts [5], it usually achieves good results in prediction compared with other methods mentioned above. However, the log time series usually contain non-linear, non-stationary, high noisy and chaotic characteristics, which may sharply weaken the training and prediction results of the neural network.

Wavelet transform has become a fast-developing and popular method in signal processing field since the 1980s. Because of its powerful feature extraction capability, it is often regarded as a "microscope" in mathematics [6] and a powerful tool for representing nonlinearities [7]. The concept of wavelet neural network (WNN) which combines the wavelet transform and neural network was first proposed by Q. Zhang in 1992 [8]. Wavelet neural networks have

demonstrated remarkable results in the prediction, classification, and modeling of different nonlinear signals [9].

This project implements a new log analysis strategy based on wavelet neural network. It uses wavelet transform to decompose the input log data into Approximation Coefficients (AC) and Detail Coefficients (DC), then perform thresholding on the coefficients, get the denoised data series and feed it to the neural network. Wavelet transform is an effective tool for the analyzing of a wide range of time-series data, which is especially suitable for log data analysis. Experimental results reveal that the new log analysis strategy has many advantages. For example, it has faster convergence, strong learning ability, and generalization capability, easily adapt to the new data, and has a better prediction result.

Project Overview

The datasets used in the experiments of this study are produced by a virtualized cloud platform, their features include create-time, host id, program id, and severity.

The software part is developed based on two frameworks: Deeplearning4J and JWave. Deeplearning4J implements the computational basis of the neural network, while JWave implements wavelet transform and the support of wavelets. This software combines the two frameworks by a series of steps.

Create a neural network model

- Use LSTM (Long Short-Term Memory) networks.

- Multi-layers model with hidden layers.

Create a wavelet transform denoising processor

- Select a wavelet and a level n, apply wavelet (or wavelet packet) transform to decompose the noisy data series into a set of approximation and detail coefficients.
- Calculate a threshold, and apply thresholding to the detail coefficients to remove noises.
- Recompose the coefficients to get a denoised data series.

Feed the denoised data series into the neural network model

- For every wavelet basis, train the model and make predictions.
- Try different wavelets basis and select the best one.

CHAPTER THREE

NEURAL NETWORK

The neural network is a computing model that consists of a number of simple, highly interconnected processing elements (called nodes or neurons), which abstractly emulates the structure and operation of the biologic nervous system [10].

Neuron

Each neuron is associated with a particular function called activation function. The connection between every two neurons (called weight) represents the signal strength when passing through the two neurons. Suppose we have a training dataset (x_i, y_i). Figure 1 shows a typical neuron.

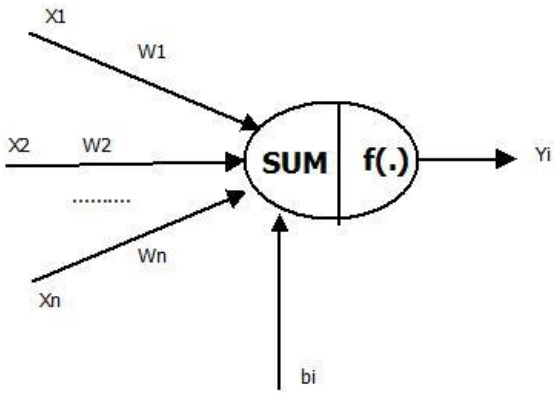

Figure 1. A Typical Neuron Model in Neural Network.

In Figure 1, x_j (j: 1->n) are the signals passed from other neurons to the current neuron i; w_{ij} is the weighted value that connects neuron j with neuron i; b_i is the bias. The output value of the neuron can be expressed as:

$$net_i = \sum_{j=1}^{n} w_{ij} x_j - b_i \qquad (1)$$

$$y_i = f(net_i) \qquad (2)$$

The function f is activation function, if we treat the bias as a special input x^0 with corresponding weight w^{i0}, then we can abbreviate the above expression to be:

$$net_i = \sum_{j=0}^{n} w_{ij} x_j \qquad (3)$$

$$y_i = f(net_i) \qquad (4)$$

If the net value is positive, the neuron is in the fire state, while negative, it is in the inhibitory state [11].

Neural Network Model

The neural network is a model that consists a number of neurons. The output of one neuron can be the input to the other. Figure 2 shows a simple neural network:

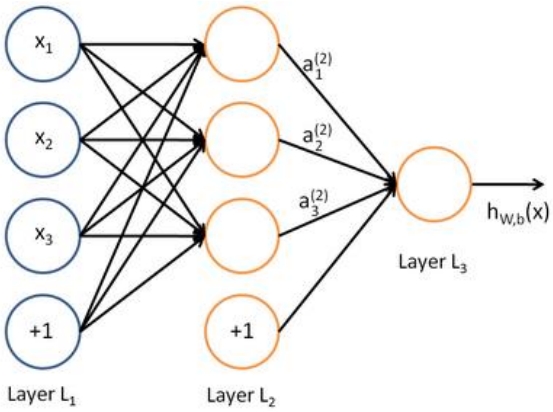

Figure 2. Three Layers Neural Network Model.

The leftmost layer is the input layer; the rightmost is the output layer, and the layers between them are hidden layers. The number of neurons in an input layer and an output layer is determined by the dimension of the actual input and output requirements [12]. The number of neurons in a hidden layer depends on the complexity of the problems, the more complex the problem is, the more neurons the hidden layer usually needs.

If we use $a_i^{(l)}$ to represent the activation value of the ith neuron of layer l, when l=1, $a_i^{(1)} = x_i$, that is the ith input value. For every given vector parameters weight W, and bias b, we can get the output value by function $h_{W,b}(x)$ as steps below:

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \quad (5)$$

$$a_2^{(2)} = f\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}\right) \quad (6)$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \quad (7)$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}) \quad (8)$$

If we use $z_i^{(l)}$ to represent the sum of weighted input values including the bias of the ith neuron in lth layer (e.g. $z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(2)}x_j + b_j^{(2)}$), then $a_i^{(l)} = f(z_i^{(l)})$, we can get a more concise expression:

$$z^{(2)} = W^{(1)}x + b^{(1)} \quad (9)$$

$$a^{(2)} = f(z^{(2)}) \quad (10)$$

$$z^{(3)} = W^{(3)}a^{(2)} + b^{(2)} \quad (11)$$

$$h_{W,b}(x) = a^{(3)} = f(z^{(3)}) \quad (12)$$

The calculation steps above are called forward propagation. With a similar architecture, we can create models with more hidden layers and more neurons in output layer according to actual problems.

When a neural network model is created, we can initialize the weights with random values. During the training phase, the model compares the output value with an expected value. If they are not equal, the model will use the difference to adjust the network parameters. This process is called backpropagation or BP. The BP algorithm divides the difference for every neuron backwardly from hidden layers to the input layer. In this way, neurons in every layer get the error signals, and adjust their weights according to corresponding error signals. This happens during the whole training phase. The weights are adjusted constantly until the difference between the output and the expected value is acceptable or the presetting training time is finished [13].

CHAPTER FOUR

WAVELET TRANSFORM

In this chapter, we will discuss wavelet transform, including the decomposing feature of wavelet and wavelet packet, and the idea to apply them into denoising.

Wavelet Transform

The wavelet transform was proposed to make up the shortages of Fourier transform and Short Time Fourier Transform [14] for the analysis and processing of non-stationary signals. Wavelet transform has the multi-resolution feature, and the ability to get locality in both time and frequency domains. It has been widely used in the signal processing field.

Wavelet transform can provide the time and frequency information through a series of basic transformation with an original wavelet called mother wavelet. To be specific, the time features can be obtained by the translation of a mother wavelet, while frequency features can be obtained by changing the scale of the mother wavelet. Using wavelet transform, we can decompose a signal into a series of wavelets with different scales and positions. These wavelets are dilated and translated forms of a mother wavelet:

$$\psi_{s,p}(t) = \frac{1}{\sqrt{s}}\psi(\frac{t-p}{s}) \qquad (13)$$

In the equation (13), s is the scale or dilation parameter and p is the shift or translation parameter. The value $\frac{1}{\sqrt{s}}$ is used for normalizing $||\Psi_{s,p}(t)|| = 1$. The scale value determines the stretch or compression level of the wavelet [15].

There are several types of wavelet transform: continuous wavelet transform (CWT), discrete wavelet transform (DWT), and wavelet packet transform (WPT).

Suppose f(t) is the signal. Ψ is the shifted wavelet with certain scale and position. CWT calculates the sum of the inner product between signal f(t) and mother wavelet Ψ in the whole signal existing period.

$$C_x(s,p) = \frac{1}{\sqrt{s}}\int_{-\infty}^{\infty} f(t)\Psi(\frac{t-p}{s})\,dt, s > 0 \qquad (14)$$

By adjusting the scale and position, CWT can get a series of coefficients C, which represent the relationship between the wavelet and corresponding part of the signal. The bigger value the C is, the more approximate degree they are.

Because the continuous variable of scale and translation may cause a computational problem, DWT is introduced. The DWT can decompose the signal into different frequency components, and it can also recompose them into a signal.

WPT is very similar to DWT, the differences are that DWT only decompose the approximation coefficients, while in WPT, both the approximation and detail coefficients are decomposed. Therefore, WPT has a better ability for complex and flexible analysis. Figure 3 shows a 3 levels WPT decomposition tree.

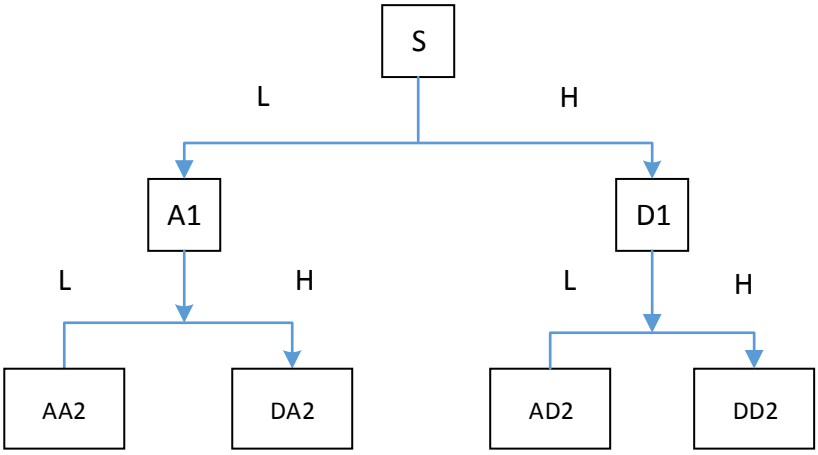

Figure 3. Wavelet Packet Decomposition Tree.

Wavelet Denoising

The objective of denoising is to remove the noisy part of the signal while keeping the original signal as much as possible. The noisy signal can be expressed as:

$$s(k) = f(k) + e(k) \qquad (15)$$

Where s(k) is the observed signal, f(k) is the effective signal, e(k) is noise. Recovering f from the noisy signal s is the objective of denoising.

Wavelet thresholding denoising [16] is an effective method to remove noise. The basic idea is: As the continuity feature of effective signal, the wavelet coefficients are usually large, while noise is randomness, its coefficients are usually smaller than effective signal. Besides, noise is mostly contained in the high-frequency range of a signal, that is, the detail of wavelet coefficients. Therefore, if we use a threshold to set the small coefficients to zero, most the noise will be removed, and only slightly damage the effective signal if we adopt a

reasonable threshold. After thresholding, we reconstruct the signal to get the denoised signal.

The process of wavelet denoising can be done in three steps: 1. Select a wavelet and a level, apply wavelet or wavelet packet decomposition to the noisy signal to get a set of coefficients. 2. Select an appropriate threshold value and apply thresholding to the detail part of coefficients. 3. Reconstruct the signal to get a denoised signal. Figure 4 shows the process.

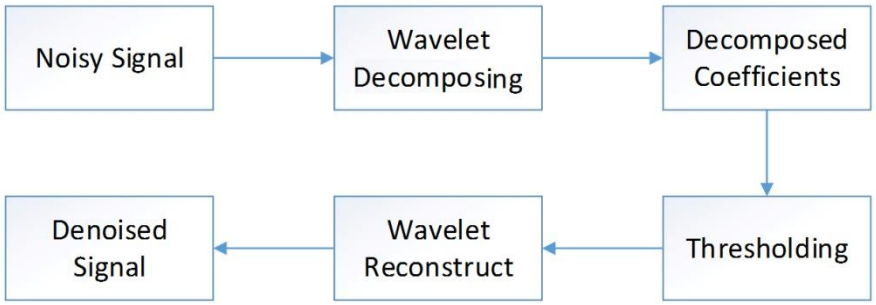

Figure 4. Wavelet Threshold Denoising Process.

In log data series, a number of wavelets can be used, such as Haar, Morlet, Daubechies, Coiflet, Symlets.

Wavelet Neural Network

As discussed above, wavelet transform has the ability to feature extraction and denoising, while neural network (NN) has self-adaptive, fault tolerance, robustness, and strong inference ability [17]. If we can combine the advantages of the two techniques, the new architecture will be more powerful and efficient in the predictions of log events. This new type of network model is usually called wavelet neural network [18]. There are different types of wavelet neural networks

according to the combination approaches. This study implements a model in an incompact way, that is, using wavelet transform to preprocess the input data series, then feed the processed data into a neural network.

CHAPTER FIVE

EXPERIMENT PREPARATION AND MODEL SELECTION

The experimental target of this study is a virtualized cloud platform, which serves a variety of applications. This platform consists of a number of hosts with different operating systems and programs running inside. The purpose is to analyze the historical log data of the platform, then make a prediction of the future possible events. To achieve this, the first step needs to do is to collect the data. Then design an appropriate network model according to the dataset, train the network and make predictions.

Data Collection

As the hosts and applications in the virtualized cloud platform are complex, the first step we need to do is identify what features we care about for the measurement and analysis afterward. When an event occurs, we need to identify its time, location, and event content. Therefore, for every data item, we need the features of occurred time, a host identifier, program (application) identifier, and severity. Besides, a neural network is a mathematical model, which needs the numerical representation of features, we need to convert all the qualitative values (text, characters, etc.) to quantitative values before fitting them to the model. So labeling all the hosts, programs, and severity names with numbers are necessary. We assign each of the three features with specific value domains according to the business requirements. For example, we define the severity value from 0 to 19,

the more the value is, the higher level it is. Similarly, according to the business relations, we assign the hosts from 0 to 4, and programs from 0 to 50.

With the values defined above, we changed the log code in virtualized cloud platform according to the format we defined. Then the platform keeps serving with those applications and handing with various user requests. After a period of time, we got a log dataset.

Table 1. Sample of Experiment Data

Time	Host ID	Program ID	Severity
1466536701	3	13	16
1466536704	3	13	16
1466536725	2	2	8
1466536726	2	2	8
1466536727	2	2	8
1466536729	3	2	18
1466536784	2	2	8
1466536785	2	2	8
1466536787	1	42	11
1466536810	1	42	11
1466536824	3	13	16

Table 1 shows some sample items of the log data, each item has 4 columns, listed in time incremental order.

Data Analysis and Model Selection

The log dataset we obtained from the platform have time sequence feature, the items are ordered by timestamps. This type of feature is called time series, which is a sequence of numerical data points that are listed in successive order. Unlike other types of dataset, the sequence order itself can also provide special information. Time series forecasting is the use of a model to predict future values

based on previously observed values [19]. RNN (Recurrent Neural Networks) is a kind of neural network for handling time series data forecasting. It can not only learn the current information but also take advantage of previous sequence dependence. The Long Short-Term Memory (LSTM) network is a type of RNN. It is suitable for analyzing and predicting longer time dependencies, and overcomes the drawback called vanishing gradient problem in RNN networks. LSTM is a powerful network to learn the most important previous information and understand whether this information is useful or not for making predictions.

This study will choose LSTM as the neural network model to incorporate with wavelet transform for the time series log dataset prediction.

Computational Tools

The implementation of both neural network and wavelet transform need a lot of work. For example, the neural network needs fast math libraries for various mathematical calculations. It also needs supporting training methods and adjusting network parameters. Wavelet transform needs various vector calculations and the support of wavelets. So both their implementations need a lot of work, which are not within the scope of this study. Instead, this study will use some existing computational tools to support the experiment.

Neural network framework

The implementation of the neural network is crucial for training and predicting effects. In recent years, many neural network frameworks have been developed for supporting training. For example TensorFlow, Deeplearning4j,

Theano, Caffe, and Torch. This study uses Deeplearning4j as neural network supporting framework. Deeplearning4j is a Java-based toolkit for building, training and deploying neural networks [20]. It is open source software developed by Skymind Company. Deeplearning4j can support well for the LSTM model we need in this study. And the version is 0.8.0.

Wavelet transform framework

JWave is an implementation of a series of algorithms, like Wavelet Packet Transform, Fast Wavelet Transform, and Discrete Fourier Transform, which are available in 1-D, 2-D, and 3-D calculations. The wavelet transform algorithms are using normalized orthogonal (orthonormal) wavelets like Haar, Coiflet, Daubechies, Symlets, and even some Bi-Orthogonal [21]. JWave is also based on Java language, so it can combine with Deeplearning4j well.

CHAPTER SIX

SYSTEM CONSTRUCTION

The part is the implementation of combining neural network with wavelet transform. It includes network design and implementation.

Network Design

As we discussed in the previous chapter, the log data has four columns: [time, host id, program id, severity]. We will treat time as a sequential identifier while using the other three columns for training and testing. Therefore, both the input layer and output layer have 3 nodes. The input log data is fed into the wavelet transform/Inverse wavelet transform processor module firstly, which is for log data series feature extracting and denoising. The number of hidden layer units is adjusted during the experiment according to the prediction precise. Between hidden layer and output layer, a dense layer is added for the change of dimensions and features extraction. Figure 5 shows the model architecture created in this study.

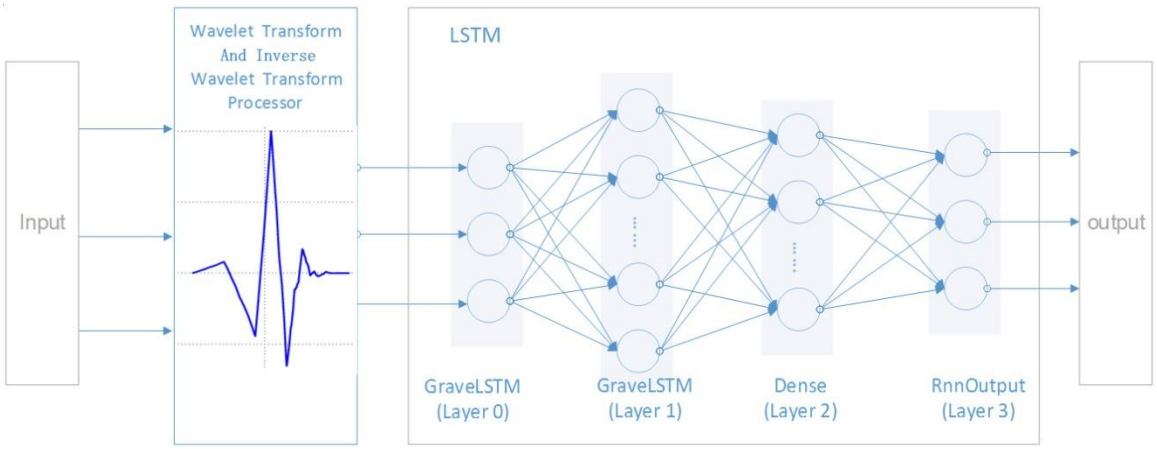

Figure 5. Neural Network Architecture in This Study.

According to the model tuning results, the parameters of the LSTM model is configured as Table 2.

Table 2. Parameters of the LSTM Model in This Study

Name	Value
Learning rate	0.02
Iterations	1
Hidden layer 1 nodes	256
Hidden layer 2 nodes	256
Dense layer nodes	32
Backprop type	TruncatedBPTT
TruncatedBPTT length	32

Due to Haar and Daubechies 3 are commonly used in time series dataset, this study will use these two wavelets to build up the wavelet transform processor module respectively.

Implementation

This part will introduce the implementation work, including some core classes design and algorithms.

LogPrediction

LogPrediction shown in Figure 6 is the main class for running log prediction process. This class integrates all the procedures, including data series preprocessing, wavelet denoising, training, testing, and prediction quality evaluation.

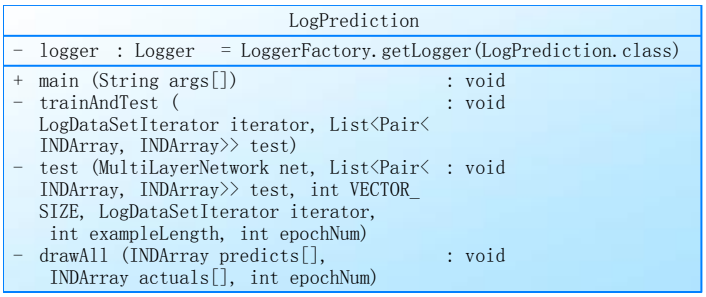

Figure 6. Training and Testing for Log Prediction.

There are two quality evaluations provided in this study: the subjective and the objective. We plot out the prediction and actual values in one chart, so we can compare the prediction accuracy between different models and algorithms. Besides, we also provide a quantify measurement: Root Mean Squared Error (RMSE).

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y(i) - y'(i))^2}{N}} \quad (16)$$

Where y(i) is the prediction value, y'(i) is the actual value, N is the dataset size for prediction. The smaller values of RMSE, the closer are the predicted values to the actuals values.

LogDataSetIterator

LogDataSetIterator provides the preprocessing functions. It also provides some methods used during the training and testing stage.

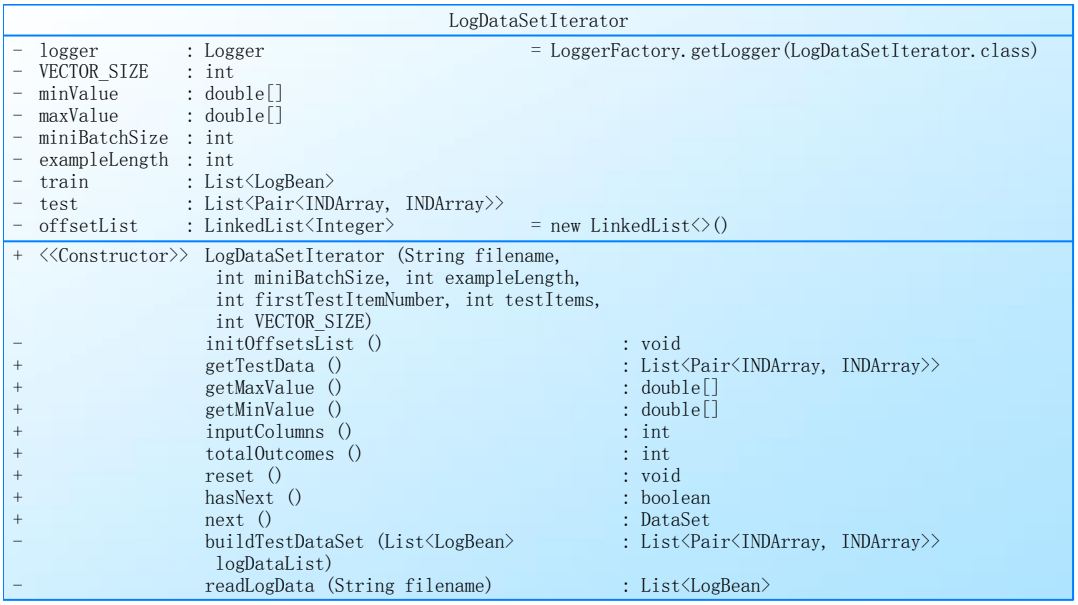

Figure 7. Dataset Iterator for Input Data.

The preprocessing in LogDataSetIterator includes reading a dataset from a file, splitting the dataset into test and training sets. It also provides normalization transformation which converts the dataset values into the range [0, 1]. The normalization follows the expression:

$$f: x \to y = \frac{x - x_{min}}{x_{max} - x_{min}} \qquad (17)$$

In equation (17), x_{min} is the minimum value of x in dataset, x_{max} is the maximum value in dataset, x, y ∈ R^n.

WaveletTransformProcessor

WaveletTransformProcessor shown in Figure 8 is the implementation of wavelet transform / Inverse wavelet transform processor. It provides a processor for wavelet denoising, which decomposes the input noisy data series into a set of coefficients with different levels, applies a threshold into the detail coefficients, and reconstruct the signal to get the denoised signal.

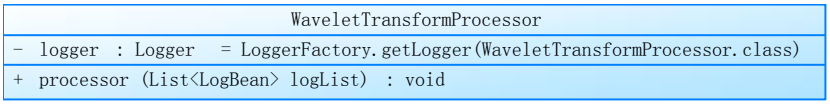

Figure 8. Wavelet Transform / Inverse Wavelet Transform Processor.

WaveletCoefHandler

WaveletCoefHandler provides thresholding algorithms, as shown in Figure 9, it can calculate the threshold value according to the coefficients feature and the decomposed level, and provide thresholding with both hard and soft ways.

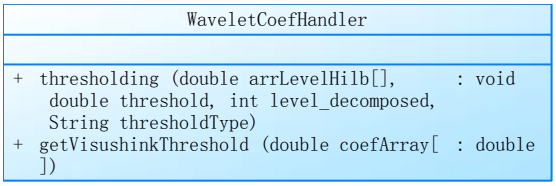

Figure 9. Wavelet Coefficient Handler.

The method getVisushinkThreshold is an algorithm to calculate the threshold value, it is implemented according to the Visushink method which is proposed by Donoho and Johnstone [22]. They proved from a statistic point of view that the noisy signal coefficients will be smaller than $\sigma\sqrt{2\log M}$ with high probability, where M is the log data series length, σ is the noise variance. For

calculation convenience, it is usually to set $\sigma = \frac{Medium(W)}{0.6745}$, where W is the high frequency wavelet coefficient array, and Medium(w) is the medium value of the array.

The method thresholding is applying the threshold value into coefficients, it provides two ways of thresholding: soft-thresholding and hard-thresholding. The hard-thresholding is to set the coefficients to 0 when they are less than the threshold value.

$$w_\lambda = \begin{cases} w, & |w| > \lambda \\ 0, & |w| \le \lambda \end{cases} \quad (18)$$

While soft-thresholding is not only to set the coefficients that are less than the threshold value but also to subtract the coefficients with the threshold value that greater than the threshold value.

$$w_\lambda = \begin{cases} sign(w) * (|w| - t), & |w| < \lambda \\ 0, & |w| \le \lambda \end{cases} \quad (19)$$

Where sign() is the positive or negative sign of a wavelet coefficient. The two thresholding methods will be tested and compared in the experiment.

LSTMNetwork

LSTMNetwork is the model class, it creates an LSTM model with the given parameters shown in Figure 10.

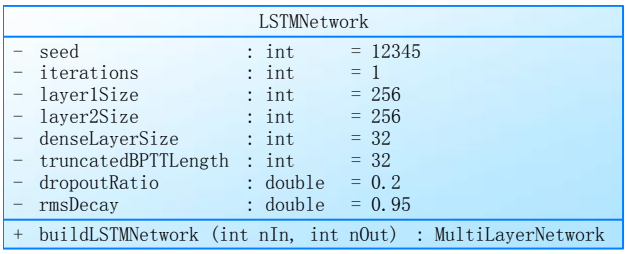

Figure 10. Neural Network Model Class.

DrawingTool

In order to get a visible output and the comparison between the prediction and actual values, a drawing class is designed, the class DrawingTool showed in Figure 11 provides the function for printing out the predictions with a visible line chart. It is easy to get to know the prediction accuracy and the trend by comparing the prediction line with an actual line in one chart.

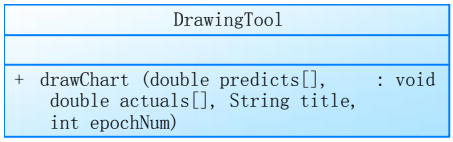

Figure 11. Drawing Tool for Visible Output.

CHAPTER SEVEN

EXPERIMENT

The experiments below are based on the datasets created in the preparation stage. The program splits the dataset into two parts, for example, the proportion of training part is 60%, and the testing part is 40%.

This study creates three types of neural networks: Haar wavelet network, Daubechies 3 (DB3) wavelet network, and conventional network. These three network models will be initialized with the same parameters and trained with the same datasets. The results will be compared.

This experiment uses the dataset collected from the virtualized cloud platform. In the following part, firstly we illustrate the denoising effects with Haar and DB3 wavelets, then compare the prediction results of the three models.

Haar Denoising

In this part, we show the denoising results for the three input original signals (severity, program, and host) with Haar wavelet.

Severity Denoising

Figure 12 shows the original noisy signal and the denoised severity signal with Haar.

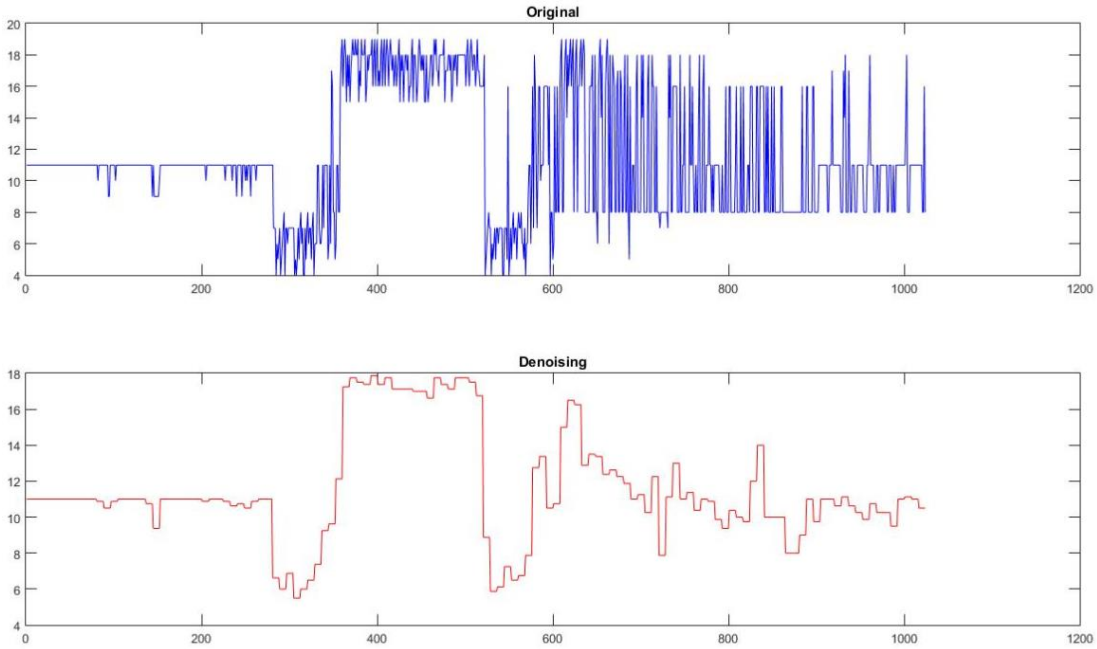

Figure 12. Original and Denoised (with Haar) Severity Signal.

For better comparison, as shown in Figure 13, we merge the two signals in one chart.

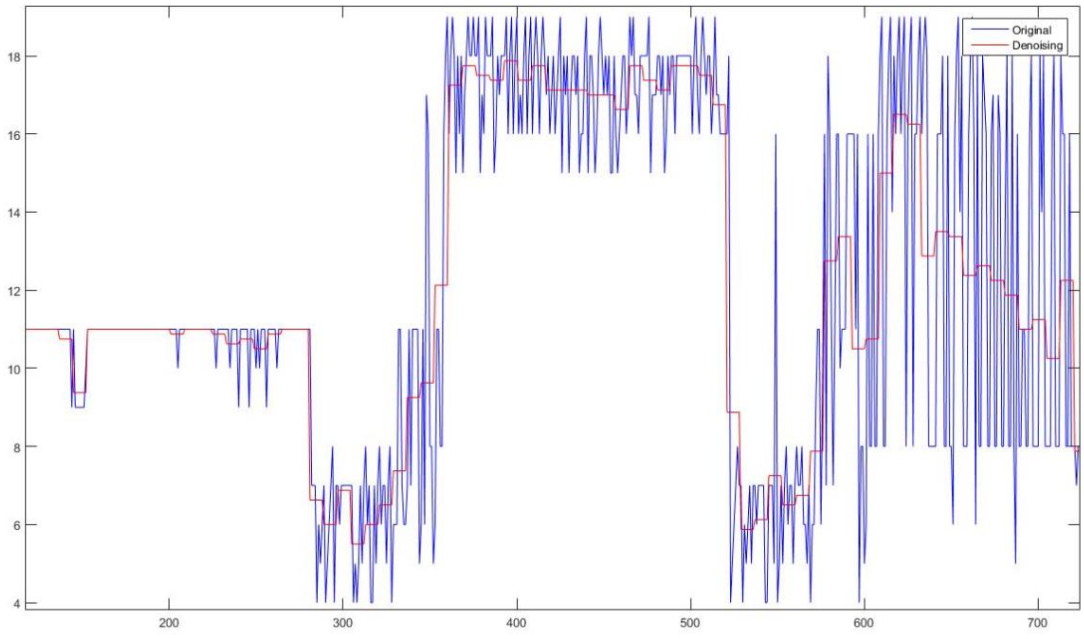

Figure 13. Original and Denoised (with Haar) Severity Signal in One Chart.

Program Denoising

Figure 14 and Figure 15 show the original noisy signal and the denoised program signal with Haar.

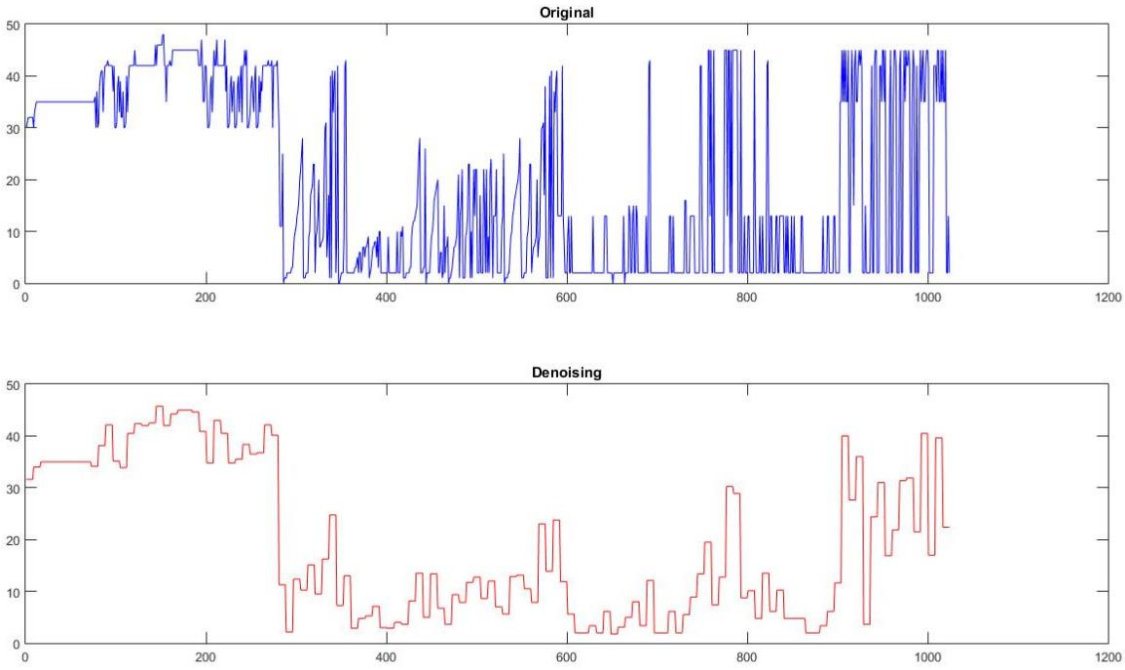

Figure 14. Original and Denoised (with Haar) Program Signal.

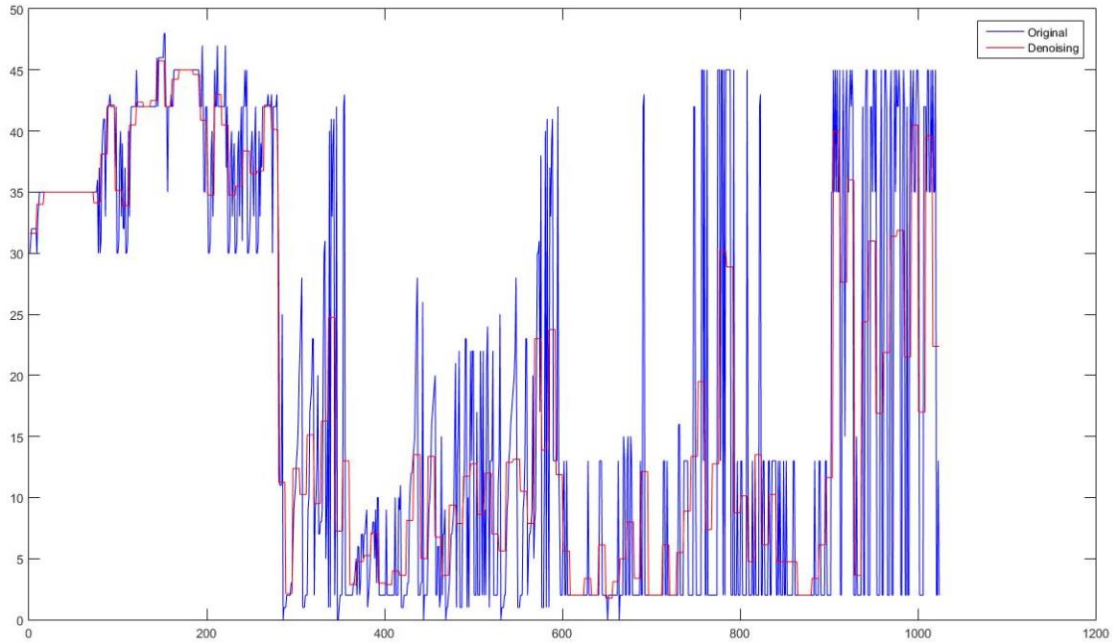

Figure 15. Original and Denoised (with Haar) Program Signal in One Chart.

Host Denoising

Figure 16 and Figure 17 show the original noisy signal and the denoised host signal with Haar.

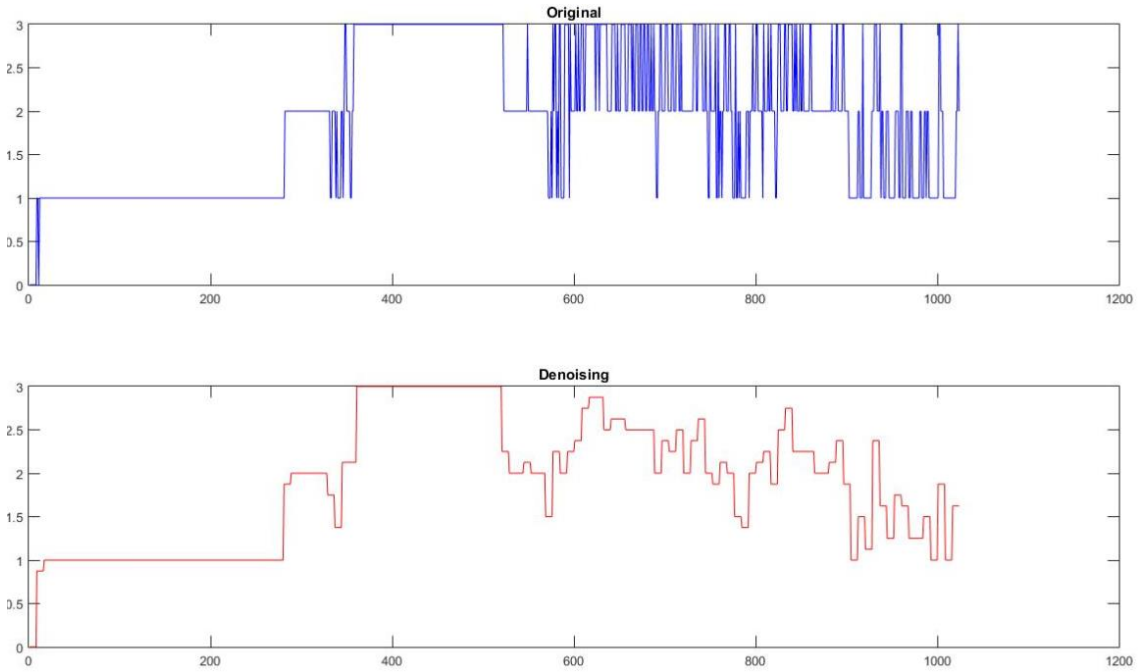

Figure 16. Original and Denoised (with Haar) Host Signal.

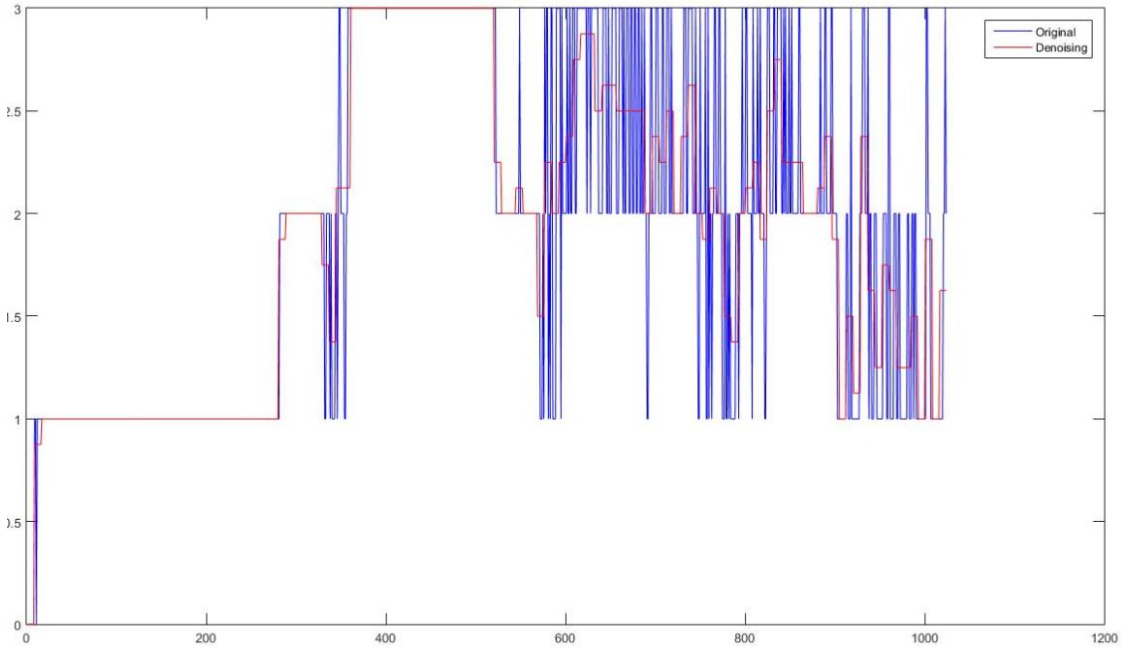

Figure 17. Original and Denoised (with Haar) Host Signal in One Chart.

Daubechies 3 Denoising

In this part, we show the denoising results for the three input original signals (severity, program, and host) with Daubechies 3 wavelet.

Severity Denoising

Figure 18 and Figure 19 show the original noisy signal and the denoised severity signal with Daubechies 3.

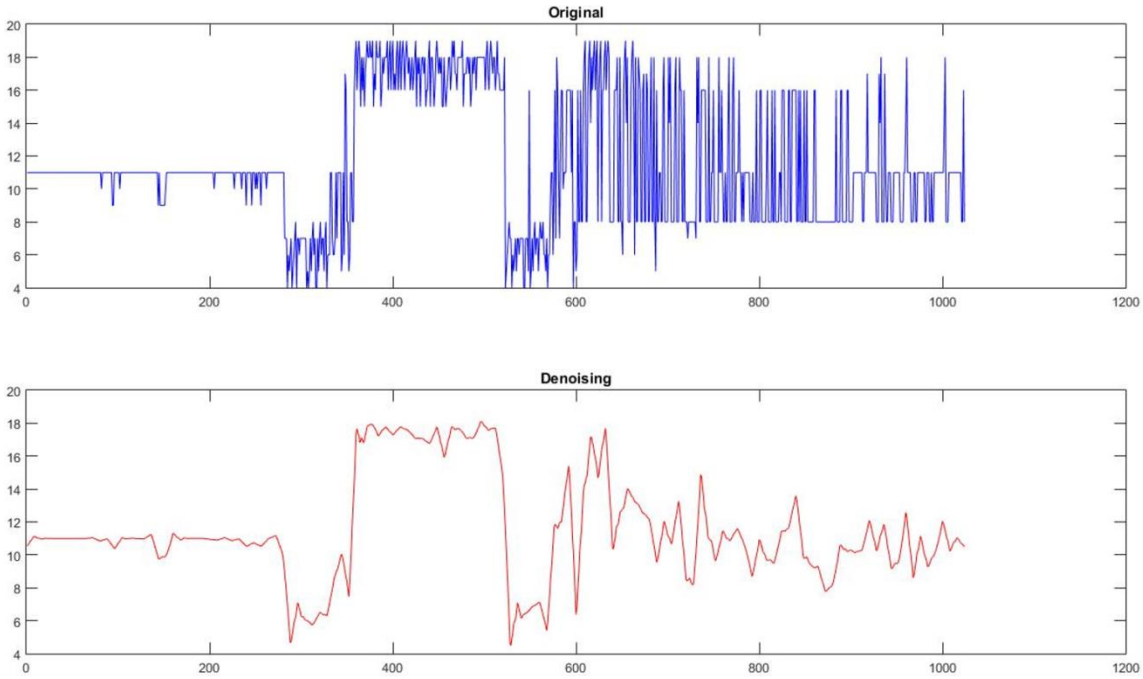

Figure 18. Original and Denoised (with DB3) Severity Signal.

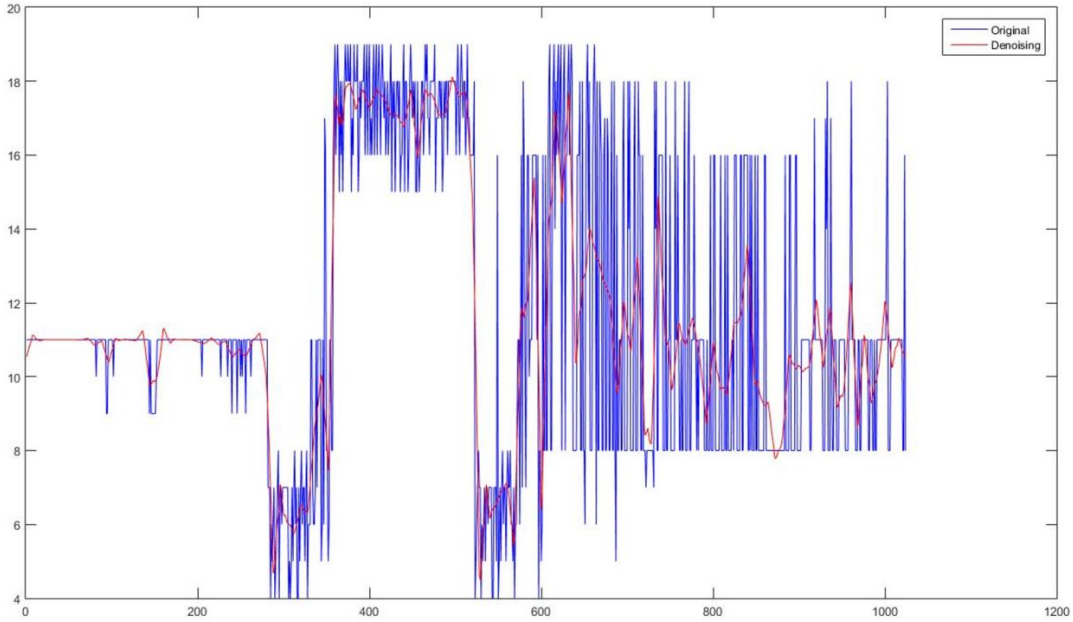

Figure 19. Original and Denoised (with DB3) Severity Signal in One Chart.

Program Denoising

Figure 20 and Figure 21 show the original noisy signal and the denoised program signal with Daubechies 3.

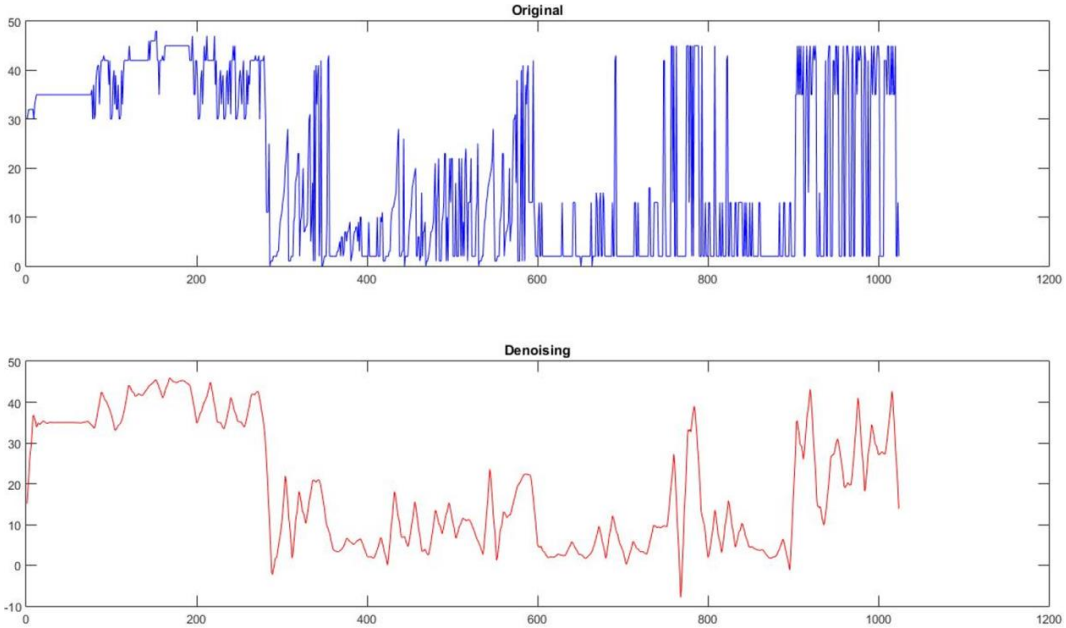

Figure 20. Original and Denoised (with DB3) Program Signal.

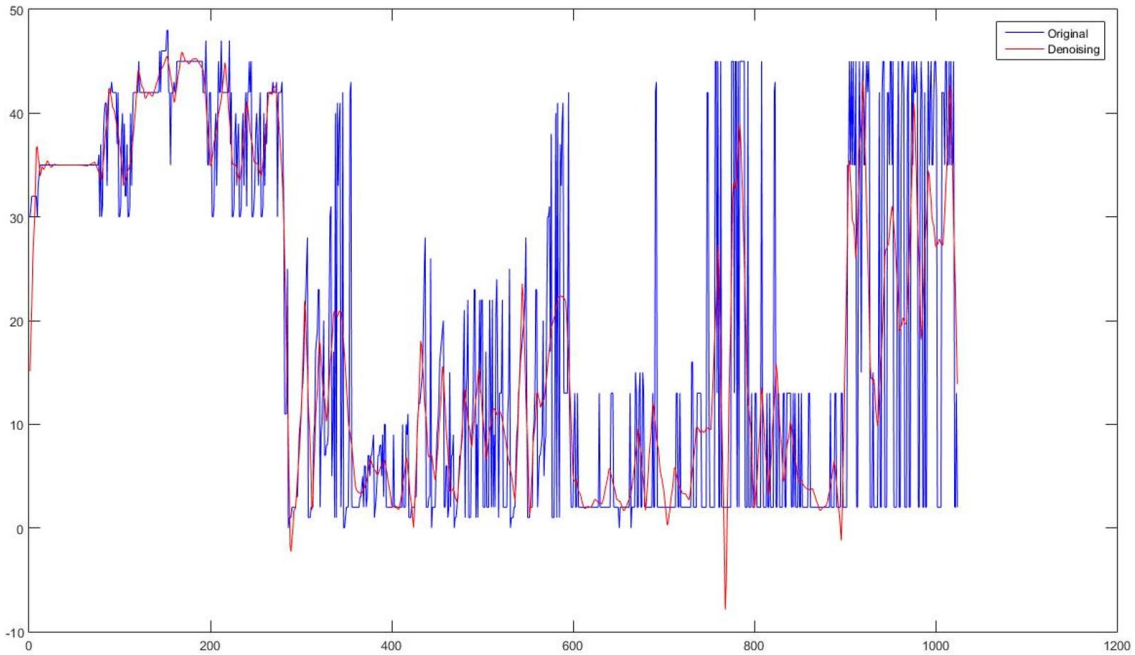

Figure 21. Original and Denoised (with DB3) Program Signal in One Chart.

Host Denoising

Figure 22 and Figure 23 show the original noisy signal and the denoised host signal with Daubechies 3.

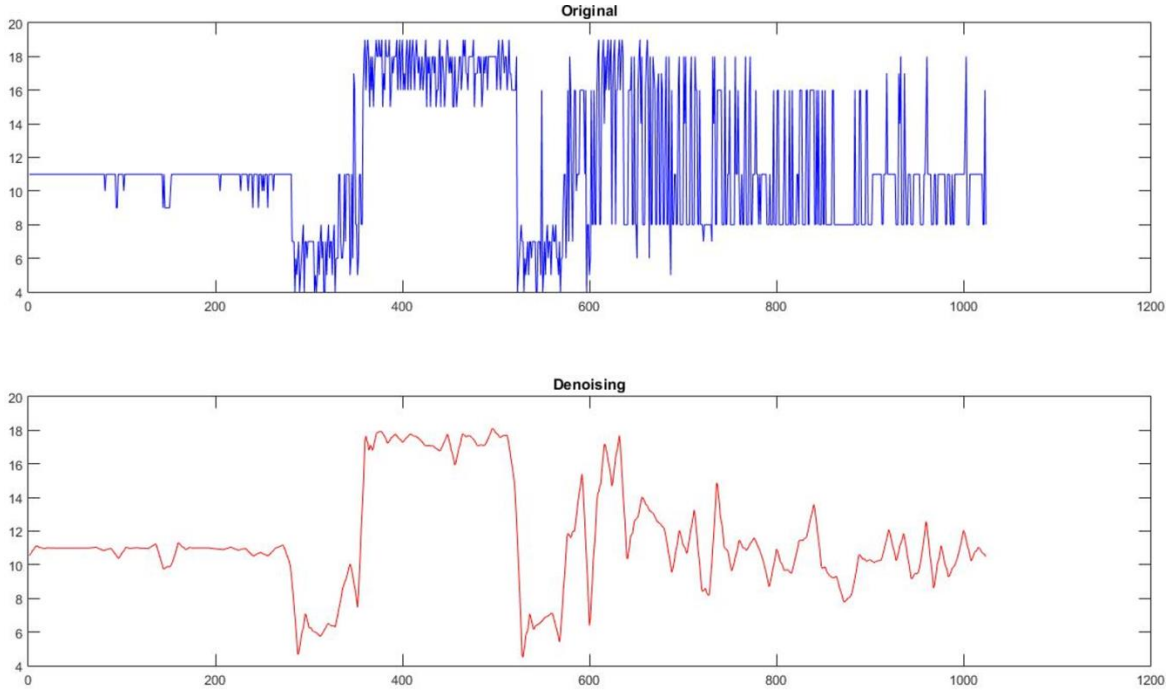

Figure 22. Original and Denoised (with DB3) Host Signal.

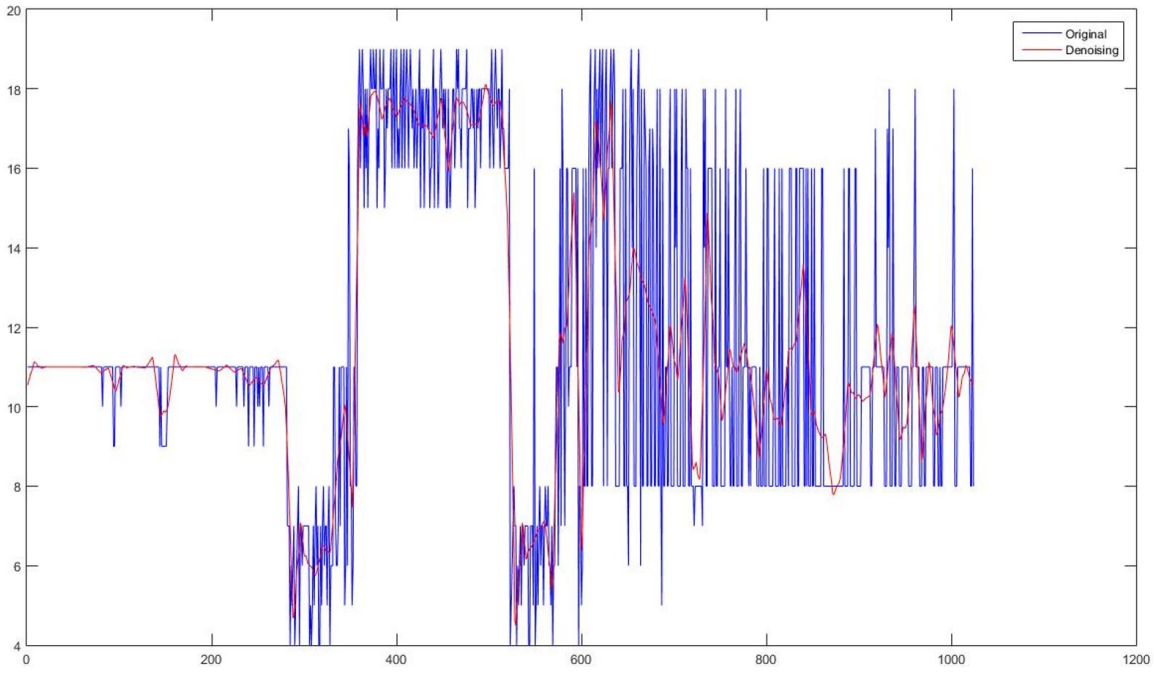

Figure 23. Original and Denoised (with DB3) Host Signal in One Chart.

From Figure 12-23, we can see wavelet denoising removes a considerable amount of the noise while preserving the sharp features in the signal, which means it can preserve important signal features while removing noise.

Prediction Results

All the LSTM parts of the three models are configured with the same structures and parameters. Train them with 60 epochs, we get prediction results of severity, program, and host by different network models. Each of the following figures shows the prediction results of Haar (top), DB3 (middle) and Conventional network (bottom) respectively.

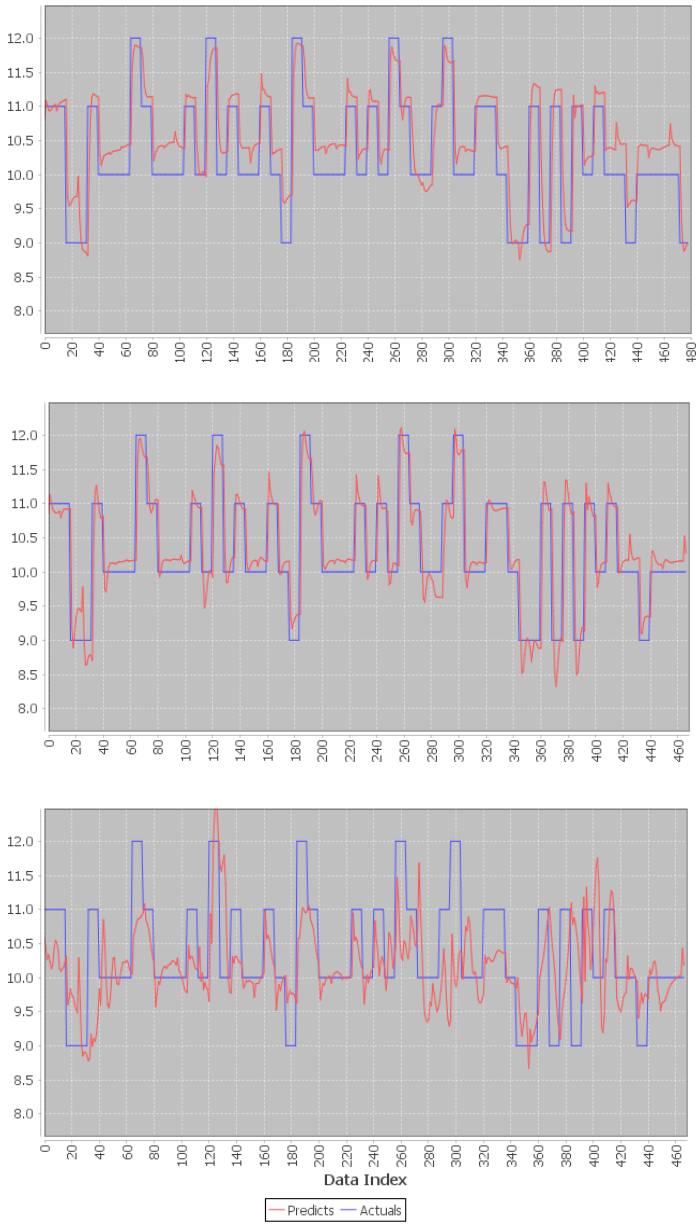

Figure 24. Severity Prediction by Haar, DB3 and Conventional NN.

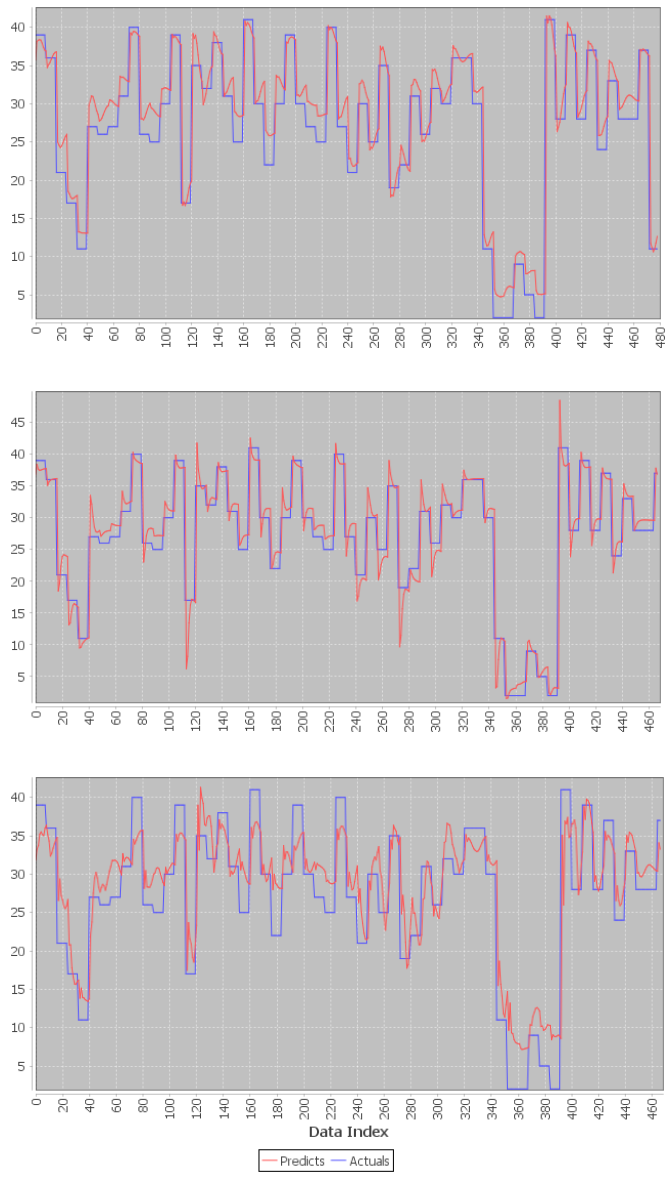

Figure 25. Program Prediction by Haar, DB3 and Conventional NN.

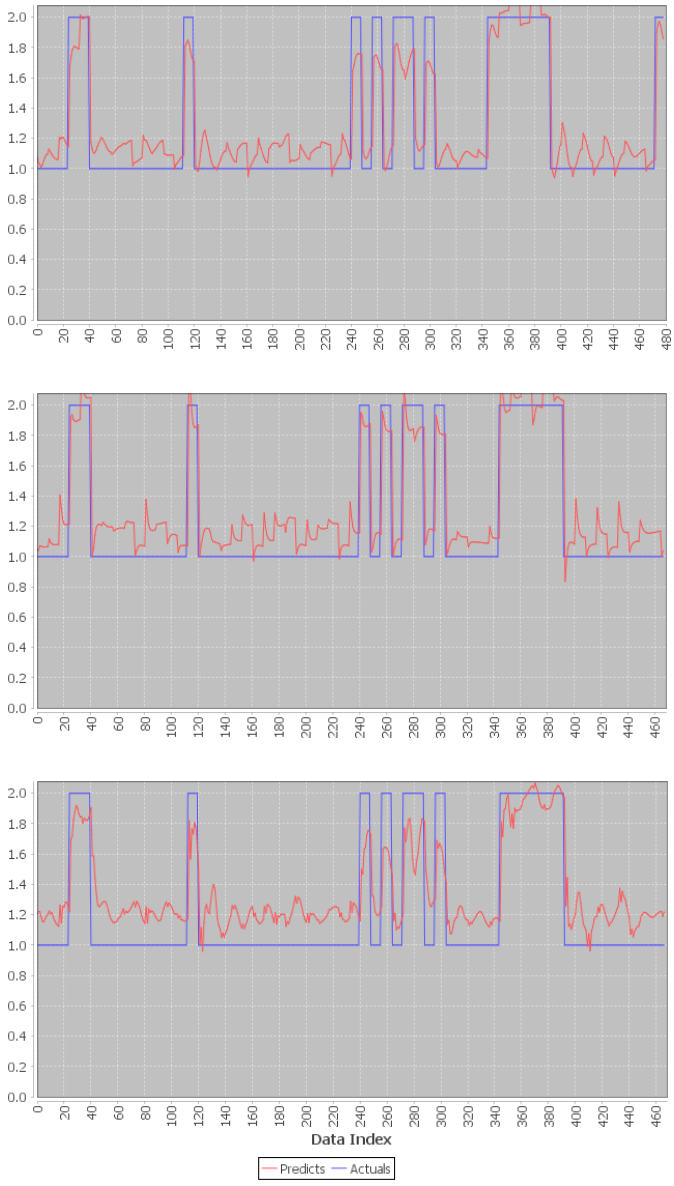

Figure 26. Host Prediction by Haar, DB3 and Conventional NN.

We can see, from Figure 24 to 26, both the Haar (top chart) and DB3 (middle chart) wavelet neural networks can provide more accurate predictions than the conventional (bottom chart) neural network in the same condition.

The RMSE values of severity, program, and host of the three models are shown in table 3.

Table 3. RMSE Values of the Three Models

Neural Network Model	RMSE		
	Severity	Program	Host
Haar	0.5319	4.0112	0.2031
DB3	0.4499	4.0420	0.2137
Conventional NN	0.7533	4.9809	0.2557

It is clear that lower RMSE values for wavelet neural network models when compared to a conventional network. Take DB3 for example, the MSE values of severity, program, and host predictions are 40.3%, 18.9%, and 16.4% less than the corresponding values of conventional neural network respectively.

From the results shown above. It is evident that the performance wavelet neural network model (both Haar and DB3) is remarkably better than the conventional neural network model. In the same condition, they have a better ability of pattern recognition, faster convergent speed, and high precision.

CHAPTER EIGHT

FUTURE WORK

This study tries two different wavelets to create wavelet neural networks for log dataset analysis and prediction, but the wavelets only limit to those supported by JWave package. The future work will further expand the wavelet library to find more suitable wavelets for log datasets.

The future work also needs to try other ways for the combination of wavelet transform and neural network, for example, the way to replace the activation function of hidden neurons with wavelet function, the translation and dilation of wavelet along with weights are modified in accordance with some training algorithm.

The study only used the datasets created by a virtualized cloud platform. More datasets from more platforms are needed to strengthen the conclusion.

CHAPTER NINE

CONCLUSIONS

This study results from the virtualized platform management method. The purpose is to try a new model by combining wavelet transform with a neural network. This new type of neural network was proposed for time series log datasets. At the beginning of the study, the data demands were analyzed, then a dataset was collected for the experiment usage. According to the time series feature of platform log dataset, LSTM model was introduced as a neural network framework with Haar or DB3 as the wavelet processor. After the design of the network architecture and classes, the wavelet neural network was implemented.

The experiment constructs three type of neural networks: Haar wavelet neural network, DB3 wavelet neural network, and conventional neural network. They share the same LSTM network architecture and configurations. Then feed these models with same training and testing datasets. The experiment results reveal that the proposed model provides much better prediction results than existing methods. It also has a significantly better ability to recognize the data patterns, and train with a more rapid convergence. Therefore, the work demonstrates the feasibility of combining wavelet transform and neural network in virtualized cloud platform to achieve accurate prediction.

APPENDIX A

CODE OF CRITICAL PARTS

This section will introduce some critical parts of the program, including some algorithms and implementation code. For example, the algorithms of training and testing, data building, wavelet denoising, wavelet coefficients handling, and the implementation of network model building.

Training and Testing Algorithms in Class LogPrediction

```
    /**
     * train and test the network model, save or load the model
     */
      private static void trainAndTest(LogDataSetIterator iterator,
List<Pair<INDArray, INDArray>> test) throws IOException {
            // build lstm network
            logger.info("build LSTM networks...");
            MultiLayerNetwork net = null;
            net = LSTMNetwork.buildLSTMNetwork(iterator.inputColumns(),
iterator.totalOutcomes());
            int epochs = PropertiesUtil.getEpochs();

            String fileName = "LogLSTM_" + PropertiesUtil.getWaveletType() +
".zip";
            File locationToSave = new File("savedModels/" + fileName);
            // if not use saved model, train new model
            if(!PropertiesUtil.getUseSavedModel()) {
                  logger.info("starting to train LSTM networks with "
+PropertiesUtil.getWaveletType()+ " wavelet...");
                  // train
                  for (int i = 0; i <= epochs; i++) {
                        logger.info("training at epoch "+i);
                        DataSet dataSet;
                        while (iterator.hasNext()) {
                              dataSet = iterator.next();
                              net.fit(dataSet);
                        }
                        iterator.reset(); // reset iterator
                        net.rnnClearPreviousState(); // clear previous state
                  }
                  // save model to file
                  logger.info("saving trained network model...");
                  ModelSerializer.writeModel(net, locationToSave, true);
            } else {
                  logger.info("loading network model...");
                  net =
ModelSerializer.restoreMultiLayerNetwork(locationToSave);
            }
            // testing
```

```
            test(net, test, PropertiesUtil.getVectorSize(), iterator,
PropertiesUtil.getExampleLength(), epochs);

        logger.info("Both the training and testing are finished, system is
exiting...");
        System.exit(0);
    }

    /**
     * test the model, plot the prediction charts and calculate the MSE values
     */
      private static void test(MultiLayerNetwork net, List<Pair<INDArray,
INDArray>> test, int VECTOR_SIZE, LogDataSetIterator iterator, int
exampleLength, int epochNum){
             // Testing
             logger.info("testing...");
        INDArray max = Nd4j.create(iterator.getMaxValue());
        INDArray min = Nd4j.create(iterator.getMinValue());
        INDArray[] predicts = new INDArray[test.size()];
        INDArray[] actuals = new INDArray[test.size()];

        double[] mseValue=new double[VECTOR_SIZE];
        for (int i = 0; i < test.size(); i++) {
            predicts[i] =
net.rnnTimeStep(test.get(i).getKey()).getRow(exampleLength -
1).mul(max.sub(min)).add(min);
            actuals[i] = test.get(i).getValue();
            // Calculate the MSE of results
            mseValue[0]+=Math.pow((actuals[i].getDouble(0,0) -
predicts[i].getDouble(0,0)), 2);
            mseValue[1]+=Math.pow((actuals[i].getDouble(0,1) -
predicts[i].getDouble(0,1)), 2);
            mseValue[2]+=Math.pow((actuals[i].getDouble(0,2) -
predicts[i].getDouble(0,2)), 2);
        }
        double mseHost = Math.sqrt(mseValue[0]/test.size());
        double mseProgram = Math.sqrt(mseValue[1]/test.size());
        double mseSeverity = Math.sqrt(mseValue[2]/test.size());
        logger.info("MSE for [severity, program id, host] is:
["+mseSeverity+", "+mseProgram+", "+mseHost + "]");

        logger.info("Starting to print out values.");
             // draw chart for prediction and actual values
               for (int i = 0; i < predicts.length; i++)
                     logger.info("Prediction="+predicts[i] + ", Actual=" +
actuals[i]);
               logger.info("Drawing chart...");
               drawAll(predicts, actuals, epochNum);
               logger.info("Finished drawing...");
          }
```

Part of Data Building Algorithms in Class LogDataSetIterator

```
    /**
     * get the next data for training
     * @return DataSet
     */
    public DataSet next() {
        if (offsetList.size() == 0) throw new NoSuchElementException();
        int actualMiniBatchSize = Math.min(miniBatchSize, offsetList.size());
        INDArray input = Nd4j.create(new int[] {actualMiniBatchSize,
VECTOR_SIZE, exampleLength}, 'f');
        INDArray label = Nd4j.create(new int[] {actualMiniBatchSize,
VECTOR_SIZE, exampleLength}, 'f');
        for (int index = 0; index < actualMiniBatchSize; index++) {
            int startIdx = offsetList.removeFirst();
            int endIdx = startIdx + exampleLength;
            LogBean curData = train.get(startIdx);
            LogBean nextData;
            for (int i = startIdx; i < endIdx; i++) {
                nextData = train.get(i + 1);
                int c = i - startIdx;
                input.putScalar(new int[] {index, 0, c}, (curData.getHostID()
- minValue[0]) / (maxValue[0] - minValue[0]));
                input.putScalar(new int[] {index, 1, c},
(curData.getProgramID() - minValue[1]) / (maxValue[1] - minValue[1]));
                input.putScalar(new int[] {index, 2, c},
(curData.getSeverity() - minValue[2]) / (maxValue[2] - minValue[2]));

                label.putScalar(new int[] {index, 0, c}, (nextData.getHostID()
- minValue[0]) / (maxValue[0] - minValue[0]));
                label.putScalar(new int[] {index, 1, c},
(nextData.getProgramID() - minValue[1]) / (maxValue[1] - minValue[1]));
                label.putScalar(new int[] {index, 2, c},
(nextData.getSeverity() - minValue[2]) / (maxValue[2] - minValue[2]));
                curData = nextData;
            }
            if (offsetList.size() == 0)
             break;
        }
        return new DataSet(input, label);
    }

    /**
     * build test data with input and label
     * @param logDataList
     * @return
     */
    private List<Pair<INDArray, INDArray>> buildTestDataSet (List<LogBean>
logDataList) {
       int window = exampleLength + 1; // window is for predicting the
following items as a time series
       List<Pair<INDArray, INDArray>> test = new ArrayList<>();
```

```
        for (int i = 0; i < logDataList.size() - window; i++) {
                INDArray input = Nd4j.create(new int[] {exampleLength,
VECTOR_SIZE}, 'f');
                for (int j = i; j < i + exampleLength; j++) {
                        LogBean log = logDataList.get(j);
                        input.putScalar(new int[] {j - i, 0}, (log.getHostID() -
minValue[0]) / (maxValue[0] - minValue[0]));
                        input.putScalar(new int[] {j - i, 1}, (log.getProgramID()
- minValue[1]) / (maxValue[1] - minValue[1]));
                        input.putScalar(new int[] {j - i, 2}, (log.getSeverity() -
minValue[2]) / (maxValue[2] - minValue[2]));
                }

                INDArray label = Nd4j.create(new int[] {VECTOR_SIZE}, 'f');
                LogBean log = logDataList.get(i + exampleLength);
                label.putScalar(new int[] {0}, log.getHostID());
                label.putScalar(new int[] {1}, log.getProgramID());
                label.putScalar(new int[] {2}, log.getSeverity());

                test.add(new Pair<>(input, label));
        }
        return test;
    }

    /**
     * read data from file
     * @param filename
     * @return
     */
        private List<LogBean> readLogData (String filename) {
          List<LogBean> logDataList = new ArrayList<>();
          try {
                logger.debug("Reading data items..");
              @SuppressWarnings("resource")
                        List<String[]> list = new CSVReader(new
FileReader(filename)).readAll();
              for (int i = 0; i < maxValue.length; i++) {
                  maxValue[i] = Double.MIN_VALUE;
                  minValue[i] = Double.MAX_VALUE;
              }
              for (String[] arr : list) {
                  double[] nums = new double[VECTOR_SIZE];
                  for (int i = 0; i < arr.length; i++) {
                      nums[i] = Double.valueOf(arr[i]);
                      if (nums[i] > maxValue[i])
                            maxValue[i] = nums[i];
                      if (nums[i] < minValue[i])
                            minValue[i] = nums[i];
                  }
                  logDataList.add(new LogBean(nums[0], nums[1], nums[2]));
              }
          } catch (IOException e) {
```

```
            e.printStackTrace();
        }
        return logDataList;
    }
```

Wavelet Denoising Algorithms in Class WaveletTransformProcessor

```
/**
 * Wavelet feature extraction, decompose, thresholding, denoising, and
 * reconstruction
 *
 * @author Liucy
 *
 */
public static void processor(List<LogBean> logList) {
      Logger.info("Strating to denoise log data series with " +
PropertiesUtil.getWaveletType());
        int len = logList.size();
        int level_decomposed = 3;
        // create arrays for wavelet transform
        double[] hostArr = new double[len];
        double[] programArr = new double[len];
        double[] severityArr = new double[len];
        for(int i=0; i<len; i++)
        {
             LogBean lb = logList.get(i);
             hostArr[i] = lb.getHostID();
             programArr[i] = lb.getProgramID();
             severityArr[i] = lb.getSeverity();
        }
        Wavelet waveletType = null;
        if("DB3".compareToIgnoreCase(PropertiesUtil.getWaveletType())==0)
             waveletType=new Daubechies3();
        else
if("Haar".compareToIgnoreCase(PropertiesUtil.getWaveletType())==0)
             waveletType=new Haar1();

        Transform transform=new Transform(new
WaveletPacketTransform(waveletType));

        // denoise
        double[][] hostArrHilb2D = transform.decompose(hostArr);
        double[][] programArrHilb2D = transform.decompose(programArr);
        double[][] severityArrHilb2D = transform.decompose(severityArr);

        // Get threshold by details coefficient from level 1
        double hostThreshold =
WaveletCoefHandler.getVisushinkThreshold(hostArrHilb2D[1]);
        double programThreshold =
WaveletCoefHandler.getVisushinkThreshold(programArrHilb2D[1]);
        double severityThreshold =
```

```
WaveletCoefHandler.getVisushinkThreshold(severityArrHilb2D[1]);

        // Use threshold to denoise
        WaveletCoefHandler.thresholding(hostArrHilb2D[level_decomposed],
hostThreshold, level_decomposed, "hard");
        WaveletCoefHandler.thresholding(programArrHilb2D[level_decomposed],
programThreshold, level_decomposed, "hard");
        WaveletCoefHandler.thresholding(severityArrHilb2D[level_decomposed],
severityThreshold, level_decomposed, "hard");

        double [] hostArrReco = transform.recompose(hostArrHilb2D,
level_decomposed);
        double [] programArrReco = transform.recompose(programArrHilb2D,
level_decomposed);
        double [] severityReco = transform.recompose(severityArrHilb2D,
level_decomposed);

        for(int i=0; i<len; i++)
        {
             LogBean lb = logList.get(i);
             lb.setHostID(hostArrReco[i]);
             lb.setProgramID(programArrReco[i]);
             lb.setSeverity(severityReco[i]);
        }

        logger.info("Host Data Series Origianl: " + Arrays.toString(hostArr));
        logger.info("Host Data Series Denoised: " +
Arrays.toString(hostArrReco));
        logger.info("");
        logger.info("Program Data Series Origianl: " +
Arrays.toString(programArr));
        logger.info("Program Data Series Denoised: " +
Arrays.toString(programArrReco));
        logger.info("");
        logger.info("Severity Data Series Origianl: " +
Arrays.toString(severityArr));
        logger.info("Severity Data Series Denoised: " +
Arrays.toString(severityReco));
        logger.info("");
      }
```

```
	/**
	 * Apply the threshold into coefficients, support hard and thresholds
	 * @param arrLevelHilb
	 * @param threshold
	 */
	public static void thresholding(double[] arrLevelHilb, double threshold,
int level_decomposed, String thresholdType) {
		int approximationNum =(int) (arrLevelHilb.length/Math.pow(2,
level_decomposed));
		// hard threshold
		if(thresholdType.equalsIgnoreCase("hard"))
		{
			// use threshold only for detail part, exclude from
approximation part
			for (int i = approximationNum; i < arrLevelHilb.length;
i++) {
				if (Math.abs(arrLevelHilb[i]) <= threshold) {
					arrLevelHilb[i] = 0;
				}
			}
		} else if(thresholdType.equalsIgnoreCase("soft")) {
			// soft threshold
			for (int i = approximationNum; i < arrLevelHilb.length;
i++) {
				if (Math.abs(arrLevelHilb[i]) <= threshold) {
					arrLevelHilb[i] = 0;
				} else {
					if(arrLevelHilb[i] < 0)
						arrLevelHilb[i] = threshold -
Math.abs(arrLevelHilb[i]);
					else
						arrLevelHilb[i] =
Math.abs(arrLevelHilb[i] - threshold);
				}
			}
		}
	}

	/**
	 * Calculate the threshold value, Implement this method according to
	 * 'Visushink threshold method' (by Donoho and Johnstone)
	 *
	 * @param coefArray
	 * @return
	 */
	public static double getVisushinkThreshold(double[] coefArray) {
		double threshold=0;
		double sigma = 0;

		double[] tempArray = Arrays.copyOfRange(coefArray, 0,
```

```
coefArray.length/2);
            int len = tempArray.length;

            for(int i=0; i<len; i++)
            {
                  tempArray[i] = Math.abs(tempArray[i]);
            }
            Arrays.sort(tempArray);
            if(len%2==0 && len>=2)
                  sigma =(tempArray[len/2-1] + tempArray[len/2])/2/0.6745;
            else
                  sigma = tempArray[len/2]/0.6745;

            threshold = sigma * Math.sqrt(2.0 * Math.log(len));

            return threshold;
      }
```

LSTM Model Builder in Class LSTMNetwork

```
      /**
       * build LSTM model with parameters, and use UIServer to get the
       * visualized monitoring information when training the model
       *
       * @param nIn: node size of input
       * @param nOut: node size of output
       * @return LSTM Model
       */
    public static MultiLayerNetwork buildLSTMNetwork (int nIn, int nOut) {
        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
                .seed(seed)
                .iterations(iterations)
                .learningRate(PropertiesUtil.getLearningRate())
.optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
                .weightInit(WeightInit.XAVIER)
                .updater(Updater.RMSPROP)
                .rmsDecay(rmsDecay)
                .regularization(true)
                .l2(l2Value)
                .list()
                .layer(0, new GravesLSTM.Builder()
                        .nIn(nIn)
                        .nOut(layer0Size)
                        .activation(Activation.TANH)
                        .gateActivationFunction(Activation.HARDSIGMOID)
                        .dropOut(dropoutRatio)
                        .build())
                .layer(1, new GravesLSTM.Builder()
                        .nIn(layer1Size)
                        .nOut(layer2Size)
```

```
                        .activation(Activation.TANH)
                        .gateActivationFunction(Activation.HARDSIGMOID)
                        .dropOut(dropoutRatio)
                        .build())
                .layer(2, new DenseLayer.Builder()
                          .nIn(layer2Size)
                          .nOut(denseLayerSize)
                          .activation(Activation.RELU)
                          .build())
                .layer(3, new RnnOutputLayer.Builder()
                        .nIn(layer3Size)
                        .nOut(nOut)
                        .activation(Activation.IDENTITY)
                        .lossFunction(LossFunctions.LossFunction.MSE)
                        .build())
                .backpropType(BackpropType.TruncatedBPTT)
                .tBPTTForwardLength(truncatedBPTTLength)
                .tBPTTBackwardLength(truncatedBPTTLength)
                .pretrain(false)
                .backprop(true)
                .build();

        // instantiate the net
        MultiLayerNetwork net = new MultiLayerNetwork(conf);
        net.init();

       // for monitoring the information when training
        // instantiate the UIserver
        UIServer uiServer = UIServer.getInstance();

        StatsStorage statsStorage = new InMemoryStatsStorage();

        // add StatsListener to collect info
        net.setListeners(new StatsListener(statsStorage));

        // for the visualization of the statsStorage content
        uiServer.attach(statsStorage);

        return net;
    }
```

REFERENCES

[1] Zou, D., Qin, H., & Jin, H. (2016). Uilog: Improving log-based fault diagnosis by log analysis. *Journal of Computer Science and Technology*, 31(5), 1038-1052.

[2] Jeon, K., Park, S., Chun, S., & Kim, J. (2016). A study on the big data log analysis for security. *International Journal of Security and Its Applications*, 10(1), 13-20.

[3] Pecchia, A., Cotroneo, D., Kalbarczyk, Z., & Iyer, R. (2011). Improving log-based field failure data analysis of multi-node computing systems. *Dependable Systems & Networks (DSN)*, 2011 IEEE/IFIP 41st International Conference on, 97-108.

[4] Frayssinet, D., Thiria, S., Badran, F. & Briqueu, L. (2000). Use of neural networks in log's data processing: prediction and rebuilding of lithologic facies. *Petrophysics meets Geophysics*, Paris, 6-8.

[5] Wang, G., Guo, L., & Duan, H. (2013). Wavelet Neural Network Using Multiple Wavelet Functions in Target Threat Assessment. *The Scientific World Journal*, 2013, 7.

[6] Cao, L., Hong, Y., Fang, H., and He, G. (1995). “Predicting chaotic time series with wavelet networks.” *Physica*, Ser D, 85, 225–238.

[7] Fang, Y., and Chow, T.W. S. (2006). “Wavelets based neural network for function approximation.” *Lecture Notes in Computer Science*, 3971, 80–85.

[8] Zhang, Q., & Benveniste, A. (1992). Wavelet networks, *IEEE Transactions on Neural Networks*, 3(6), 889-898.

[9] Bakshi, B., & Stephanopoulos, G. (1993). Temporal representation of process trends for diagnosis and control. *Control Engineering Practice*, 1(3), 549.

[10] Balabin, Safieva, & Lomakina. (2008). Wavelet neural network (WNN) approach for calibration model building based on gasoline near infrared (NIR) spectra. *Chemometrics and Intelligent Laboratory Systems*, 93(1), 58-62.

[11] (2017, May 15). An introduction to neural network [Online]. Available: http://www.cnblogs.com/HEAAD/ARCHIVE/2011/03/07/1976443.HTML.

[12] Alexandridis, Antonios K. (2013). Wavelet neural networks: A practical guide. Neural Networks, 42, 1-28.

[13] Goyal, Manish. (2014). Monthly rainfall prediction using wavelet regression and neural network: An analysis of 1901-2002 data, Assam, India. *Theoretical & Applied Climatology*, 118(1/2), 25-35.

[14] Shao, Cai, & Pan. (1999). Wavelet transform and its applications in high performance liquid chromatography (HPLC) analysis. *Chemometrics and Intelligent Laboratory Systems*, 45(1), 249-256.

[15] Alexandridis, Antonios K, Zapranis, Achilleas D. (2014). Wavelet Neural Networks: With Applications in Financial Engineering, Chaos, and Classification. Hoboken: Wiley, 10-17.

[16] Donoho, D.L. (1995) De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, Vol. 41(3), 613-627.

[17] (2017, May 17). Introduction to artificial neural networks [Online]. Available: http://blog.csdn.net/fengbingchun/article/details/50274471.

[18] Hung, S., Huang, C., & Wen, C., USING WAVELET NEURAL NETWORK FOR THE IDENTIFICATION OF A BUILDING STRUCTURE FROM EXPERIMENTAL DATA. *13th World Conference on Earthquake Engineering Vancouver*, B.C., Canada August 1-6, 2004 Paper No. 241.

[19] (2017, Jun 5). Time Series [Online]. Available: http://www.investopedia.com/terms/t/timeseries.asp.

[20] (2017, Jun 5). What Is DeepLearning4j [Online]. Available: https://deeplearning4j.org/overview.

[21] (2017, Jun 7). JWave - Open source Java implementation of orthogonal and bi-orthogonal wavelets [Online]. Available: https://github.com/cscheiblich/JWave.

[22] Donoho, D. L., & Johnstone, I. M. (1994). Ideal spatial adaptation via wavelet shrinkage. *Biometrika*, 1994, 81: 425-455.