DEVELOPMENT OF LEARNING RULES FOR HIGH-ORDER CELLULAR NEURAL NETWORKS AND APPLICABILITY IN IMAGE PROCESSING

Duong Duc Anh[1*], Nguyen Quang Hoan[2], Nguyen Hong Vu[3], Nguyen Tai Tuyen[2], Nguyen Quang Tri[4]
[1]Vietnam Research Institute of Electronics, Informatics and Automation
[2]Posts and Telecommunications Institute of Technology, [3]The Radio and Electronics Association of Vietnam
[4]Hanoi University of Science and Technology

ARTICLE INFO

Received: 07/6/2023
Revised: 29/6/2023
Published: 29/6/2023

KEYWORDS

Cellular Neural Networks
Learning Rules
High-Order
Edge detection
Perceptron

ABSTRACT

This paper aims at modifying a learning algorithm, developed from Recurrent Perceptron Learning Algorithm (RPLA) and a Pattern Recognition Algorithm for High-Order Cellular Neural Networks (HOCNN). Our research methods are developing theory of learning for high-order cellular neural networks and experiment with modified algorithms. The research results include two proposed algorithms and software which was built to test the two mentioned algorithms. The obtained set of the weights from our developed algorithm (named as Second-Order Recurrent Perceptron Learning Algorithm: SORPLA) can be used as filters or kernels for problems in imaging processing. In conclusion, firstly, the paper has modified the RPLA algorithm, which adds templates **A** and high-level templates **B**; secondly, it has improved the PyCNN image processing algorithm; finally, the paper also proposes an applicability of SORPLA in edge detection of image using the obtained set of the weights from the developed algorithm for the High-Order Cellular Neural Networks.

PHÁT TRIỂN LUẬT HỌC DÙNG CHO MẠNG NƠ RON TẾ BÀO BẬC CAO VÀ KHẢ NĂNG ỨNG DỤNG TRONG XỬ LÝ ẢNH

Dương Đức Anh[1*], Nguyễn Quang Hoan[2], Nguyễn Hồng Vũ[3], Nguyễn Tài Tuyên[2], Nguyễn Quang Trí[4]
[1]Viện nghiên cứu Điện tử, Tin học, Tự động hóa, [2]Học viện Công nghệ Bưu chính Viễn thông
[3]Hội Vô tuyến – Điện tử Việt Nam, [4]Đại học Bách Khoa Hà Nội

THÔNG TIN BÀI BÁO

Ngày nhận bài: 07/6/2023
Ngày hoàn thiện: 29/6/2023
Ngày đăng: 29/6/2023

TỪ KHÓA

Mạng nơ ron tế bào
Luật học
Bậc cao
Dò cạnh
Perceptron

TÓM TẮT

Mục đích của bài viết này là cải tiến một thuật toán học, được phát triển từ thuật toán học Perceptron hồi quy và thuật toán nhận dạng mẫu (dành cho Mạng nơ ron tế bào bậc cao). Phương pháp nghiên cứu của chúng tôi là phát triển lý thuyết học trong mạng nơ ron tế bào bậc cao và thử nghiệm các thuật toán. Kết quả nghiên cứu là hai thuật toán được cải tiến và bộ trọng số, ảnh xử lý được bằng hai thuật toán đó. Tập hợp các trọng số thu được từ thuật toán đã phát triển (tên là Thuật toán học Perceptron hồi quy bậc hai: SORPLA) có thể được sử dụng làm bộ lọc hoặc hạt nhân cho các vấn đề trong xử lý ảnh. Kết luận của bài báo như sau: Thứ nhất, sửa đổi thuật toán RPLA, bổ sung các mẫu bậc cao **A** và các mẫu bậc cao **B**; Thứ hai, cải thiện thuật toán xử lý hình ảnh PyCNN. Ngoài ra, bài báo cũng đề xuất khả năng ứng dụng của SORPLA trong phát hiện biên ảnh bằng cách sử dụng tập các trọng số thu được từ thuật toán đã phát triển cho Mạng nơ ron tế bào bậc cao.

DOI: https://doi.org/10.34238/tnu-jst.8087

[*] Corresponding author. *Email: ddatdh1@gmail.com*

1. Introduction

Cellular Neural Network (CNN) is one of the Recurrent Artificial Neural Networks (RANN) family and they have successfully created the CNN Universal Machine. Currently, CNN processors can achieve up to 50,000 frames per second [1], and for certain applications such as flash detection, these microprocessors have outperformed a conventional supercomputer [2], [3]. It was known in 1999, C. GuKzelis proposed the Recurrent Perceptron Learning Algorithm (RPLA) for the first-order CNNs [4] used in image processing. In 2014, Ankit Aggarwal presented the open-source software PyCNN in Python [5] and applied for image pattern recognition using CNN [6]. In 2020, we proposed the Second-Order Cellular Neural Networks (SOCNN) architectures based on the standard CNN presented by Leon O. Chua [7] with the Second-Order polynomial inputs, feedback outputs to improve performance compared to standard cellular neural networks [8].

This paper consists of the four sections. In the second section, we present the methodology and research basis for the learning algorithm, including: architectures and learning rules of the High-Order Cellular Neural Networks (HOCNNs); the SORPLA to determine the set of weights for SOCNNs based on the original RPLA [9]; the third part presents the calculation method and results of the learning algorithm, then applied to the edge detection problem in image processing. The last section presents the conclusion.

2. Methodology

2.1. Architectures of the High-Order Cellular Neural Networks

Consider a *M*N* of CNN, with *M*N* cells sorted into *M* rows and *N* columns. We denote the cell *(i, j)* on the i^{th} row and the j^{th} column by *C(i, j).* Now let's determine the neighborhood of a cell in CNN [7] , [10], [11].

Definition 1: The *r–neighborhood, r* $\in$ *N** of a cell $C(i,j)$ in a CNN is defined:

$$N_r(i,j)=\left\{C(k,l)\middle|max\left\{|k-i|,|l-j|\right\}\le r\right\}\quad(1\le k\le M,1\le l\le N) \tag{1}$$

In this paper, we are using the Second-Order Cellular Neural Networks (SOCNNs) architecture as a representative of the HOCNN and also select *r =1*. Then the SOCNNs can be described through a set of equations as follows [8]:

State equation:

$$\begin{aligned}C\frac{dx_{ij}(t)}{dt}=&-\frac{1}{R}x_{ij}(t)+\sum_{(k,l)}\mathbf{A1}(i,j;k,l)y_{kl}(t)+\sum_{(k,l)}\mathbf{B1}(i,j;k,l)u_{kl}+\mathbf{I}\\&+\sum_{(k,l)}\sum_{(m,n)}\mathbf{A2}(i,j;k,l,m,n)y_{kl}(t)y_{mn}(t)+\sum_{(k,l)}\sum_{(m,n)}\mathbf{B2}(i,j;k,l,m,n)u_{kl}u_{mn}\end{aligned} \tag{2}$$

where, **A1***(.)*, **A2***(.)*, **B1***(.)*, **B2***(.)* is the first-order, second-order feedback matrices; first-order, second-order input matrices respectively; *R* is linear resistor usually chosen to be between 1*KΩ* and 1*MΩ* (as normalized unity); *C* is linear capacitor usually chosen to be 1 μ *F* (as normalized unity); *I* is cellular bias of the CNN cell; $u_{ij,}$ u_{mn}; $x_{ij}(t)$; $y_{ij}(t)$, $y_{mn}(t)$ is input; state; actual output of the CNN cell respectively.

Output equation: $$y_{ij}(t)=\frac{1}{2}\left[\left|x_{ij}(t)+1\right|-\left|x_{ij}(t)-1\right|\right] \tag{3}$$

Input equation: $$u_{ij}=E_{ij}=constant\qquad 1\le i\le M;1\le j\le N \tag{4}$$

Constrain equations: $$\left|x_{ij}(0)\right|\le 1;\left|u_{ij}\right|\le 1\qquad 1\le i\le M;1\le j\le N \tag{5}$$

Parameter assumptions:

$$\mathbf{A1}(i,j;k,l)=\mathbf{A1}(k,l;i,j);\mathbf{A21}(i,j;r,s)=\mathbf{A21}(r,s;i,j);..;\mathbf{A29}(i,j;r,s)=\mathbf{A29}(r,s;i,j);$$

$$\mathbf{A2}(i,j;k,l;m,n)=\mathbf{A2}(i,j;m,n;k,l)=\mathbf{A2}(m,n;i,j;k,l;)=\mathbf{A2}(k,l;i,j;m,n)=\mathbf{A2}(k,l;m,n;i,j)=\mathbf{A2}(m,n;k,l;i,j) \quad (6)$$

Equations from (2) to (6) can be represented in Figure 1.

Remarks:

- Matrix **B2***(i,j; k,l; m,n)* deployed into an equivalent coefficients with *k*l=m*n=3*3=9* corresponding to **B21** to **B29.**

- Matrix **A2***(i,j; k,l; m,n)* deployed into an equivalent coefficients with *k*l=m*n=3*3=9* corresponding to **A21** to **A29**.

- In **A21-A29** and **B21- B29** matrices, index 2 show the coefficients of second-order polynomial for SOCNN, indexes 1-9 indicate the number of neighboring cells (*r=1*).

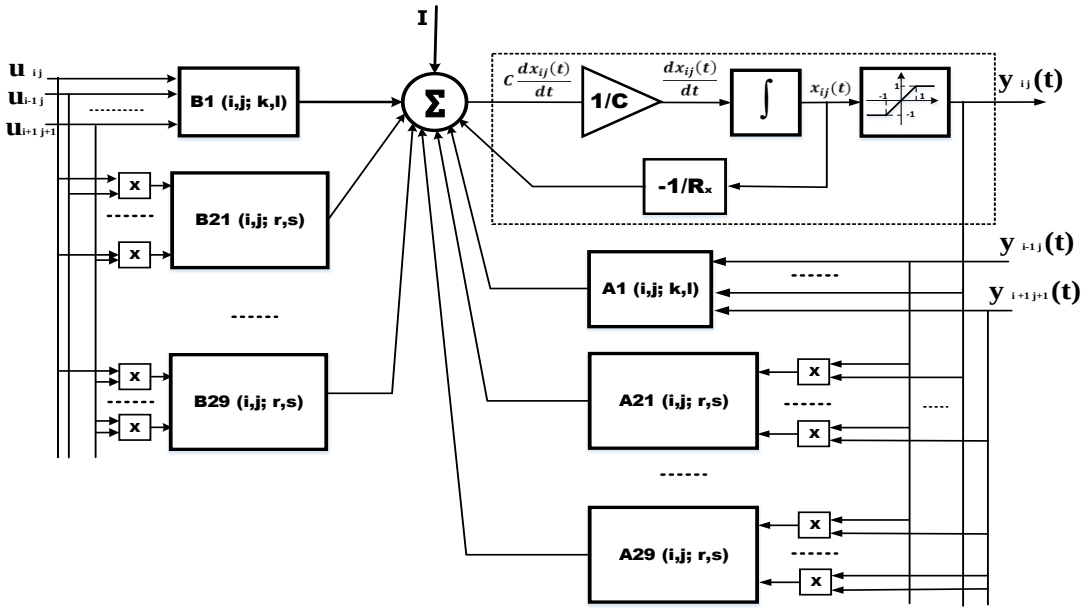

Figure 1. *Structure diagram of the SOCNNs*

2.2. Stability of the Second-Order Cellular Neural Networks

Since our SOCNN have feedback output signals, it can be leading to unstable for the system. One of the most effective tools to analyse the stability properties of the nonlinear dynamic system SOCNN is the Lyapunov method [12]. In [8] we give the theorem to prove SOCNNs stable with the symmetric conditions, indicated in (6) and it is a reason for selecting the initial symmetric weights used for SOCNN training in later. When the system is stable, it is completely operated [2]. The operation of SOCNN includes two phases: i) learning phase (determining the weights) and ii) running phase (the network can only operate when the weights are determined or learned).

2.3. Learning Rule for the High-Order Cellular Neural Networks

There have been many publications for learning methods for first-order neural networks [13]. C. GuKzelis [4] perfected the Recurrent Perceptron Learning Algorithm (RPLA) for first-order CNNs, in which, the **W**-weights include first-order output feedback weights **A1**, first-order input weights **B1**, and bias **I:**

$$\mathbf{W}=\left[\mathbf{A1}^T \quad \mathbf{B1}^T \quad \mathbf{I}\right]^T \quad (7)$$

Then, the standard CNNs are converted to the straight-forward Perceptron form, and the calculation of the newly converted weight **W** in (7) can be performed according to the RPLA.

With a similar approach, we have proposed a recurrent perceptron learning algorithm for SOCNNs. In this case, the new set of the weights is defined as follows [9]:

$$\mathbf{W} = \left[\mathbf{A}1^T \quad \mathbf{A}21^T \quad ... \quad \mathbf{A}29^T \quad \mathbf{B}1^T \quad \mathbf{B}21^T \quad .. \quad \mathbf{B}29^T \quad \mathbf{I} \right]^T \tag{8}$$

When SOCNNs reach steady state, states $x_{ij}(t)$ are constant, leading to $dx_{ij}(t)/dt = 0$, equation (2) can be rewritten as (9) (R_x=1$M\Omega$):

$$\begin{aligned} x_{ij}(t) = & \sum_{(k,l)} \mathbf{A1}(i,j;k,l) y_{kl}(t) + \sum_{(k,l)} \mathbf{B1}(i,j;k,l) u_{kl} + \mathbf{I} \\ & + \sum_{p=1}^{9} \sum_{(k,l)(m,n)} \mathbf{A2p}(i,j;k,l;m,n) y_{kl}(t) y_{mn}(t) + \sum_{p=1}^{9} \sum_{(k,l)(m,n)} \mathbf{B2p}(i,j;k,l;m,n) u_{kl} u_{mn} \end{aligned} \tag{9}$$

Set the total input matrix as follows [14]:

$$\mathbf{Y}_{ij}^s = \left[\left[y_{ij}^{s*}(\infty) \right]^T \quad \left[y_{kl}^{s*}(\infty) \right] \left[y_{ij}^{s*}(\infty) \right]^T \quad \left[u_{ij}^{s*} \right]^T \quad \left[u_{kl}^{s*} \right] \left[u_{ij}^{s*} \right]^T \quad 1 \right]^T \tag{10}$$

The stable Second - Order CNNs equation of state (9) is determined as follow:

$$x_{ij}^s(\infty) = \left[\mathbf{Y}_{ij}^s \right]^T * \mathbf{W} \tag{11}$$

then the kinematic equation of the stable SOCNNs is equivalent to a Perceptron networks by the formula (11). At that time, the weight calculation of SOCNNs can be performed according to the Trial-Error-Correction learning rule like Perceptron method. Assuming that the input and output signals of SOCNNs are polarized (values of +1 or -1), then the output interaction function of SOCNNs must be a sign function as follow:

$$y_{ij}^s(\infty) = sgn\left(x_{ij}^s(\infty) \right) = sgn\left[\left[Y_{ij}^s \right]^T * W \right] \tag{12}$$

SORPLA is built based on equation (2) when the network is stable with a split output. The stable output value is determined according to equation (3).

Learning rule:

To calculate the set of weights for the SOCNNs network at steady state, the SORPLA method is proposed with the learning algorithm below. The difference between the actual output and the desired output is calculated as follows [1]:

$$\varepsilon[w[n]] = \frac{1}{2} \sum_{i,j,s} y_{ij}^s(\infty) \times \left(y_{ij}^s(\infty) - d_{ij}^s \right) = \sum_{i,j \in D^+} y_{ij}^s(\infty) - \sum_{i,j \in D^-} y_{ij}^s(\infty) \tag{13}$$

$\varepsilon[w[n]]$: sum of error between desired stable output value and actual stable output value;

d_{ij}^s : desired stable output value of SOCNNs; *s*: number of training sample sets;

$y_{kl}^s(\infty)$: actual stable output value of SOCNNs;

D^- ; D^+ : the error output range is calculated based on the following principles:

$$D^+ = \begin{cases} y_{ij}^s(\infty) = 1 \\ d_{ij}^s = -1 \end{cases} \qquad D^- = \begin{cases} y_{ij}^s(\infty) = -1 \\ d_{ij}^s = +1 \end{cases} \tag{14}$$

The calculation of the weight value of the second-order CNNs is then done as follows:

$$W(n+1) = \left[W(n) - \Delta W(n) \right]; \Delta W(n) = \alpha * \left(\sum_{(i,j,s) \in D^+} Y_{ij}^s(n) - \sum_{(i,j,s) \in D^-} Y_{ij}^s(n) \right) \tag{15}$$

α : the learning rate of the network, usually chosen as a positive constant;

During the implementation of SORPLA algorithm, the weight vector set is adjusted considering the effect of each cell for all training samples. When the error between the actual

output and the desired output at each cell is zero, skip the weight update at that cell and move on to the next cell in a left-to-right and top-to-bottom rule. The algorithm ends only when the sum of the deviations of all pixels in the sample set reaches a value of 0.

*Input: i) Give SOCNN from (2) to (6); ii) t*emplates $\left(x_{ij}^{s}(0);u_{ij}^{s};d_{ij}^{s}\right)$ with size of *N*M; iii)* the initial weight matrix: $\mathbf{A1}^{0},\mathbf{B1}^{0},\mathbf{I}^{0}$; $\mathbf{A21}^{0},\mathbf{A22}^{0},...,\mathbf{A29}^{0}$; $\mathbf{B21}^{0},\mathbf{B22}^{0},...,\mathbf{B29}^{0}$; iv) learning rate: α;

v) sample time: τ (s)

Output: The final weight matrix: $\mathbf{A1}^{n},\mathbf{B1}^{n},\mathbf{I}^{n}$, $\mathbf{A21}^{n},\mathbf{A22}^{n},...,\mathbf{A29}^{n}$, $\mathbf{B21}^{n},\mathbf{B22}^{n},...,\mathbf{B29}^{n}$;

*Step 0: n=1 (First loop for SORPLA). For image size of M*N learning patterns.*

Step 1: Calculate $y_{ij}(t)$ following (3). Then D^{+}, D^{-} according to the formula (24)

Step 2: Calculate the second-order inputs and outputs of SOCNN;

Step 2.1: Calculate $u_{kl}u_{mn}$ then calculate $\mathbf{B2}(i,j;k,l;m,n)u_{kl}u_{mn}$ depend on Figure 1.

Step 2.2: Calculate $y_{kl}(t)y_{mn}(t)$ then calculate $\mathbf{A2}(i,j;k,l;m,n)y_{kl}(t)y_{mn}(t)$ depend on Figure 1.

Step 3: Calculate $\varepsilon[w[n]]$ following (13)

Step 3.1: If $\varepsilon[w[n]]\neq 0$ then compute $W(n+1)$ following (25), then compute $x_{ij}(n+1)$ of SOCNN following (21). After that return Step 2.

Step 3.2: If $\varepsilon[w[n]]=0$, then go to step 4;

Step 4: Return *W* following (15) for CNNs and actual output values following (3).

Remarks:

- Contribution of the authors in SOCNN is added at Step 2.1; Step 2.2
- The algorithm has 3 loops. The complexity of the algorithm is estimated O(n3).

3. Results and Discussion

3.1. Problem Statement

Give: i) the structure of the SOCNNs according to the equation (2) to (6); ii) the network training templates $\left[x_{ij}^{s}(0),u_{ij}^{s},d_{ij}^{s}\right]$ with size of *8*8*, 02 templates for learning algorithm follow Figure 2 below. Need to detection the edge of image. To solve the problem need to perform 02 phases: Learning phase and pattern recognition phase.

3.2. Learning Phase

In the learning phase, implement the learning algorithm in the Section 2.2.2. With the initial conditions as $\mathbf{A1}^{0}$, $\mathbf{B1}^{0}$ $\mathbf{I}^{0}$, $\mathbf{A21}^{0},..,$ $\mathbf{A29}^{0}$, $\mathbf{B21}^{0},..,$ $\mathbf{B29}^{0}$.

In the library of template for CNNs [15], we selected the set of initial weights, satisfying the conditions of stability for SOCNNs according to (6) and to (15) below:

$$\mathbf{A1}^{0}=\begin{pmatrix}0&0&0\\0&4&0\\0&0&0\end{pmatrix},\ \mathbf{A21}^{0}=\mathbf{A24}^{0}=\mathbf{A25}^{0}=\mathbf{A28}^{0}=\begin{pmatrix}0&0&0\\0&3&0\\0&0&0\end{pmatrix},\ \mathbf{A23}^{0}=\begin{pmatrix}0&1&0\\1&3&1\\0&1&0\end{pmatrix},\quad \mathbf{I}^{0}=0,$$

$$\mathbf{A22}^{0}=\mathbf{A26}^{0}=\mathbf{A27}^{0}=\mathbf{A29}^{0}=\begin{pmatrix}0&0&0\\0&4&0\\0&0&0\end{pmatrix},\ \mathbf{B1}^{0}=\begin{pmatrix}0&0&0\\0&0&0\\0&0&0\end{pmatrix},\ \mathbf{B21}^{0}=\mathbf{B22}^{0}=...=\mathbf{B29}^{0}=\begin{pmatrix}0&0&0\\0&0&0\\0&0&0\end{pmatrix}$$

The learning process uses SORPLA to determine the set of weights for the problem of edge detection of the image. We perform some case studies with different learning rates $\alpha_1=0.001,\alpha_2=0.01,$ and with the same time $\tau=0.01s$. With learning rate $\alpha=0.001$, time

$\tau = 0.01s$, after 40 loops, the final weights $\mathbf{A1}^{40}, \mathbf{B1}^{40}, \mathbf{I}^{40}$ $\mathbf{A21}^{40}, .., \mathbf{A29}^{40}, \mathbf{B21}^{40}, .., \mathbf{B29}^{40}$ were obtained with high accuracy ($approximately$ 100%). That means the actual output value $y_{ij}(t)$ is approximately equal to the desired output value d^s. This algorithm was implemented and installed on Desktop Computer (Core I5 7400, 16G RAM, 512GB SSD), using C# language in Visual Studio 2022 development environment with a runtime of about 60s. Regarding the calculation way, we used the traditional method Trial - Error - Correction applied for Perceptron networks. The Trial stage was given a set of initialization weights **A, B, I**. With that set of weights, it could lead to an Error $\varepsilon[w[n]]$ between the actual output and the desired output according to formula (13), so it is necessary to Correct that set of weights according to equation (15). That process was repeated until the error $\varepsilon[w[n]]$ was approximately 0 following step 3 in the SORPLA algorithm.

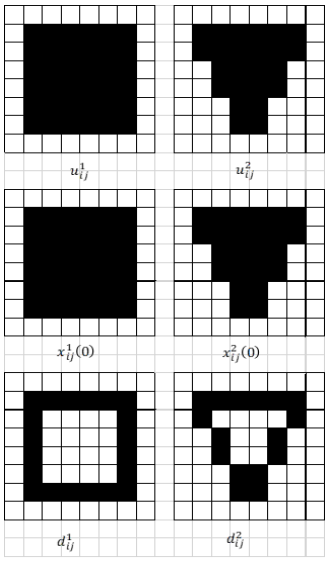

Figure 2. *Templates for image processing*

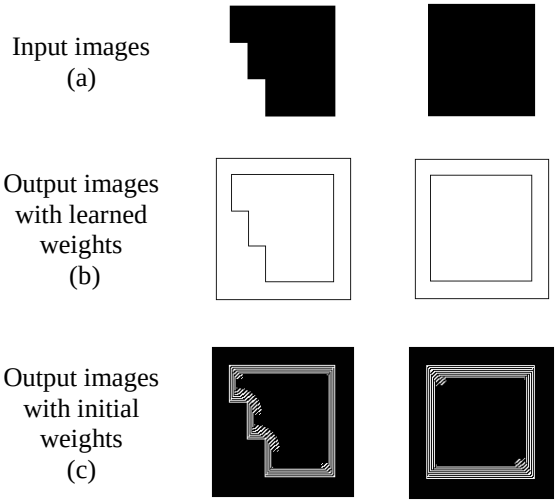

Figure 3. *Result of edge detection in image processing*

The first-order weight matrices:

$$\mathbf{A1}^{40} = \begin{pmatrix} -1.39 & -0.82 & -0.40 \\ -0.82 & 2.93 & -0.82 \\ -0.40 & -0.82 & -1.39 \end{pmatrix}, \quad \mathbf{B1}^{40} = \begin{pmatrix} 0.18 & -0.07 & -0.29 \\ -0.29 & 0.17 & -0.29 \\ -0.29 & -0.07 & 0.18 \end{pmatrix}, \quad \mathbf{I}^{40} = 18.71$$

The second-order feedback weight matrices:

$$\mathbf{A21}^{40} = \begin{pmatrix} -0.73 & -0.73 & -0.41 \\ -0.79 & 2.85 & -0.79 \\ -0.41 & -0.73 & -0.73 \end{pmatrix}, \quad \mathbf{A22}^{40} = \begin{pmatrix} -0.41 & -0.63 & -0.31 \\ -0.52 & 3.93 & -0.52 \\ -0.31 & -0.63 & -0.41 \end{pmatrix}, \quad \mathbf{A23}^{40} = \begin{pmatrix} -0.52 & 0.33 & -0.65 \\ 0.38 & 3.02 & 0.38 \\ -0.65 & 0.33 & -0.52 \end{pmatrix},$$

$$\mathbf{A24}^{40} = \begin{pmatrix} -0.73 & -0.17 & -0.57 \\ -0.28 & 3.28 & -0.28 \\ -0.57 & -0.17 & -0.73 \end{pmatrix}, \quad \mathbf{A25}^{40} = \begin{pmatrix} -0.04 & 0.34 & -0.05 \\ 0.32 & 3.18 & 0.32 \\ -0.05 & 0.34 & -0.04 \end{pmatrix}, \quad \mathbf{A26}^{40} = \begin{pmatrix} -0.44 & -0.09 & -0.40 \\ -0.18 & 4.25 & -0.18 \\ -0.40 & -0.09 & -0.44 \end{pmatrix},$$

$$\mathbf{A27}^{40} = \begin{pmatrix} -0.44 & -0.09 & -0.40 \\ -0.18 & 4.24 & -0.18 \\ -0.40 & -0.09 & -0.44 \end{pmatrix}, \quad \mathbf{A28}^{40} = \begin{pmatrix} -0.08 & -0.41 & -0.36 \\ -0.45 & 3.07 & -0.45 \\ -0.36 & -0.41 & -0.08 \end{pmatrix}, \quad \mathbf{A29}^{40} = \begin{pmatrix} -0.39 & -0.36 & -0.51 \\ -0.43 & 3.11 & -0.43 \\ -0.51 & -0.36 & -0.39 \end{pmatrix},$$

The second-order input weight matrices:

$$\mathbf{B21}^{40}=\begin{pmatrix}-0.31 & -0.60 & -0.67\\ -0.67 & -0.12 & -0.67\\ -0.67 & -0.60 & -0.31\end{pmatrix},\quad \mathbf{B22}^{40}=\begin{pmatrix}-0.06 & -0.36 & -0.41\\ -0.41 & -2.32 & -0.41\\ -0.41 & -0.36 & -0.06\end{pmatrix},\quad \mathbf{B23}^{40}=\begin{pmatrix}-0.79 & -0.83 & -0.90\\ -0.90 & -0.36 & -0.90\\ -0.90 & -0.83 & -0.79\end{pmatrix},$$

$$\mathbf{B24}^{40}=\begin{pmatrix}-0.11 & -0.20 & -0.46\\ -0.45 & 0.08 & -0.45\\ -0.46 & -0.20 & -0.11\end{pmatrix},\quad \mathbf{B25}^{40}=\begin{pmatrix}0.10 & -0.09 & -0.34\\ -0.35 & 0.19 & -0.35\\ -0.34 & -0.09 & 0.10\end{pmatrix},\quad \mathbf{B26}^{40}=\begin{pmatrix}-0.13 & -0.25 & -0.39\\ -0.38 & 0.01 & -0.38\\ -0.39 & -0.25 & -0.13\end{pmatrix},$$

$$\mathbf{B27}^{40}=\begin{pmatrix}-0.13 & -0.25 & -0.39\\ -0.38 & 0.01 & -0.38\\ -0.39 & -0.25 & -0.13\end{pmatrix},\quad \mathbf{B28}^{40}=\begin{pmatrix}0.04 & -0.22 & -0.42\\ -0.42 & 0.12 & -0.42\\ -0.42 & -0.22 & 0.04\end{pmatrix},\quad \mathbf{B29}^{40}=\begin{pmatrix}-0.63 & -0.56 & -0.75\\ -0.76 & -0.22 & -0.76\\ -0.75 & -0.56 & -0.63\end{pmatrix}.$$

With this final set of weight matrices, the actual output matrices similar to the desired output matrices. That means SORPLA finished for edge detection problem with desired accuracy.

3.3. Pattern Recognition Phase

3.3.1. Pattern Recognition Algorithm

To get the algorithm to receive pattern recognition, we modified open software PyCNN [5] become our algorithm PySOCNN by adding the second-order matrices in Python language using Google Colab Library. Here, we have performed a convolutional multiplication method between the input weights **B1, B21, B22, .., B29** (as Kernel or window or filter) of size *3x3* with the first-order input image, and the second-order input image of size ***M*N***. The process was carried out for each cell according to the rule from left to right, from top to bottom until the end of the image.

Input: SOCNN follow (2) to (6), some input images with size of ***M*N***, the learned weight matrix (Section 2.2.2): $\mathbf{A1}^n, \mathbf{B1}^n, \mathbf{I}^n$; $\mathbf{A21}^n, \mathbf{A22}^n, ..., \mathbf{A29}^n$; $\mathbf{B21}^n, \mathbf{B22}^n, ..., \mathbf{B29}^n$; number of loops: l

Output: The edge detection of images

Step 1: Initial SOCNNs include:

Step 1.1: Calculate the second-order inputs $u_{kl}u_{mn}$ then calculate $\mathbf{B2}(i,j;k,l;m,n)u_{kl}u_{mn}$ (Figure 1)

Step 1.2: Calculate the second-order outputs $y_{kl}(t)y_{mn}(t)$ then calculate $\mathbf{A2}(i,j;k,l;m,n)y_{kl}(t)y_{mn}(t)$ (Figure 1).

Step 1.3: Calculate the $\frac{dx_{ij}(t)}{dt}$ follow (2)

Step 1.4: Calculate $x_{ij}(t)$ by integral method $\frac{dx_{ij}(t)}{dt}$, Calculate $y_{ij}(t)$ follow (3)

Step 2: Give input image size of *M*N*

Step 3: Go to *step 1* to recalculate $\frac{dx_{ij}(t)}{dt}, x_{ij}(t), y_{ij}(t)$

Step 4: Return output image after τ time

Remarks:

- The our contribution is addition to *Step 1.1; Step 1.2* .
- The algorithm has 2 loops. The complexity of the algorithm is estimated $O(n^2)$.

3.3.2. Experimental results

In the Input of PySOCNN, we use: i) SOCNN from (2) to (6); ii) 02 input images with size of *64*64* as Figure 3; iii) The learned weight matrices, the initial weight matrix (Section 2.2.2);. At the output, we obtained an edge-separated image using proposed recognition algorithm after 50 epochs and about 90s. We performed the recognition algorithm on the improved PySOCNN, the actual output images is shown in Figure 3. In which, Figure 3b shows the output image using the set of computed weights, Figure 3c shows the output image using the initial set of weights.

Remarks:

- SOCNNs have the ability to apply to image processing problems. In this paper, we use the image processing method applied to the edge detection problem.
- With the same SOCNNs structure, input image sample and sampling time, then the contour of the object in the output image using a set of learnt weights has obvious results. In the other case, the border of the object is not clear, the boundary of the object is not really accurate. This can confirm the effectiveness of SORPLA algorithm for second-order cellular neural networks.

4. Conclusion

The main contribution of this paper is to improve the two algorithms. Firstly, it has modified the RPLA algorithm, by adding the high-order of templates **A** and templates **B**, and testing the weights set in C#. Secondly, it has improved the PyCNN image processing algorithm: Image Processing with High-Order Cellular Neural Networks in Python, which uses the weights learned by real math and adds high-order computational components. Boundary separation was carried out for several different cases with positive results.

The next research direction is to test the algorithm for many different scenarios, compare and evaluate. The results will be published in the following journals or conferences.

REFERENCES

[1] Wikipedia, "Cellular neural network," 2023. [Online]. Available: https://en.wikipedia.org/wiki/Cellular_neural_network. [Accessed June 4, 2023].

[2] A. Slavova, *Cellular Neural Networks: Dynamics and Modelling*, Springer Dordrecht, 2003.

[3] D. Liang, J. Zhang, S. Jiang, X. Zhang, J. Wu, and Q. Sun, "Mobile Traffic Prediction Based on Densely Connected CNN for Cellular Networks in Highway Scenarios," in *Proceedings of the 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, Xi'an, China, 2019, doi: 10.1109/WCSP.2019.8927980.

[4] C. GuKzelis, S. Karamamut, I0. Genc, "A Recurrent Perceptron Learning Algorithm for Cellular Neural Networks," *Springer-Verlag*, vol. 51, pp. 296-309, 1999.

[5] A. Aggarwal, "GitHub: Let's build from here," 2014. [Online]. Available: https://github.com/ankitaggarwal011/PyCNN. [Accessed June 2, 2023].

[6] A. Fülöp and A. Horváth, "Application of Cellular Neural Networks in Semantic Segmentation," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, Daegu, Korea, 2021, doi: 10.1109/ ISCAS51556.2021.9401249.

[7] L. O. Chua and L. Yang, "Cellular Neural Networks: Theory," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1257 - 1272, 1988.

[8] Q. H. Nguyen, T. T. Nguyen, and D. A. Duong, "Achitecture and Stability of the Second – Order Cellular Neural," *UTEHY Jounal of Science and Technology*, vol. 27, pp. 91-97, 2020.

[9] D. A. Duong, Q. H. Nguyen, T. T. Nguyen, T. V. Q. Lai, and T. D. Hoang, "Development of Recurrent Perceptron Learning Algorithm for Second-Order Cellular Neural Networks," *Measurement, Control, and Automation*, vol. 3, no. 3, pp. 64-72, 2022.

[10] T. T. Nguyen, "Development of Multi-Interaction Cellular Neural Networks and Applicability," PhD. thesis, Vietnam Research Institute of Electronics, Informatics and Automation, Hanoi, 2022.

[11] T. T. Nguyen, Q. H. Nguyen, and D. A. Duong, "Associative Memory Using Second-Order Cellular Neural Networks," *Journal of Science and Technology on Information and Communication,* vol. 1, no. 4, pp. 116-121, 2022.

[12] Q. H. Nguyen, "High-Order Hopfield Neural Network Stability and Applicability in Robot Control," PhD. Thesis, Institute Academy of Science and Technology, Hanoi, 1996.

[13] H. Mizutani, "A New Learning Method for Multilayered Cellular Neural Networks," in *Proceedings of the Third IEEE International Workshop on Cellular Neural Networks and Their Applications*, Rome, Italy, 1994, doi: 10.1109/CNNA.1994.381680.

[14] D. A Duong, T. T. Nguyen, T. T. Nguyen, and Q. H. Nguyen, "Modified Perceptron Learning Rule and Application Abilities for Cellular Neural Networks," *Scientific Journal of Ha Long Universities*, vol. 6, no. 6, pp. 1-7, 2022.

[15] K. Karacs, Gy. Cserey, Á. Zarándy, P. Szolgay, Cs. Rekeczky, L. Kék, and V. Szabó, "Software Library for Cellular Ware Computing Engines, Budapest, Hungary: Cellular Sensory and Wave Computing Laboratory of the Computer and Automation," 2014.