Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2017) 25: 2363 – 2374

doi:10.3906/elk-1510-87

Bayesian estimation of discrete-time cellular neural network coefficients

Hakan Metin ÖZER[1], **Atilla ÖZMEN**[1,*], **Habib ŞENOL**[2]
[1]Department of Electrical and Electronics Engineering, Faculty of Engineering and Natural Sciences, Kadir Has University, İstanbul, Turkey
[2]Department of Computer Engineering, Faculty of Engineering and Natural Sciences, Kadir Has University, İstanbul, Turkey

Received: 12.10.2015 • **Accepted/Published Online:** 20.09.2016 • **Final Version:** 29.05.2017

Abstract: A new method for finding the network coefficients of a discrete-time cellular neural network (DTCNN) is proposed. This new method uses a probabilistic approach that itself uses Bayesian learning to estimate the network coefficients. A posterior probability density function (PDF) is composed using the likelihood and prior PDFs derived from the system model and prior information, respectively. This posterior PDF is used to draw samples with the help of the Metropolis algorithm, a special case of the Metropolis--Hastings algorithm where the proposal distribution function is symmetric, and resulting samples are then averaged to find the minimum mean square error (MMSE) estimate of the network coefficients. A couple of image processing applications are performed using these estimated parameters and the results are compared with those of some well-known methods.

Key words: Bayesian learning, cellular neural networks, Metropolis–Hastings, estimation

1. Introduction

Since the introduction of cellular neural networks (CNNs) by Chua and Yang [1, 2], they have been an active field of research especially in image processing [3–5]. A CNN has a set of parameters (also called coefficients or synaptic weights) by its design. These parameters need to be evaluated before any processing of input data can be achieved [6]. Various methods to find these parameters have been proposed throughout the last couple of decades. Each of them has advantages and disadvantages. Choosing a suitable algorithm for both the type of CNN and type of processed data is important. The type of training algorithm may also affect the performance. A proposed method to train a CNN is to use a multilayer perceptron (MLP) [7]. Here the dynamic behavior of a CNN when converged to a fixed point is reproduced with an MLP using the adequate number of input, hidden, and output layers. Then the similarities and parameter relations between the CNN and MLP are used to find the template coefficients. However, the computational burden of the resulting CNN-MLP structure increases as the size of the CNN increases. The recurrent perception learning algorithm (RPLA) [8] is another supervised algorithm for obtaining the template coefficients in completely stable CNNs. The RPLA resembles the perceptron learning algorithm for neural networks. The RPLA suffers from the local minimum problem like its MLP version. In [9] a special case of supervised learning called decentralized asynchronous learning is demonstrated, in which each cell of the CNN learns in a spatially and temporally distributed environment. Some heuristic algorithms such as genetic algorithm and ant colony optimization algorithm [10, 11] are also

*Correspondence: aozmen@khas.edu.tr

used to train CNNs. In these methods, the difference between the settled output and the desired output is used to evaluate the CNN templates. A stable learning algorithm that is based on the input-to-state stability theory is proposed in [12].

In this work, CNN parameters are determined using Bayesian learning method. First for the unknown CNN parameters prior distribution is specified and then the posterior distribution is computed for the parameters using the observed (training) data. The posterior distribution is obtained by combining the prior distribution with the likelihood for the parameters given the training data. Obtaining a sample from the posterior distribution is more difficult than obtaining a sample from the prior distribution. Therefore generating random samples that correspond to posterior distribution is a potential problem in Bayesian learning. Markov chain Monte Carlo (MCMC) methods can be used to accomplish this task. The method proposed by Metropolis [13] and further developed later by Hastings [14] that falls under the MCMC methods is suitable for generating random samples from a multidimensional distribution. Unlike other random sample generation methods, the Metropolis–Hasting algorithm also allows one to generate a great number of random samples easily and rapidly even from a probability distribution that is not commonly known.

This paper is organized as follows. A brief theory of DTCNN is presented in Section 2. Bayesian estimation of DTCNN coefficients is given in Section 3. Generation of a template sample using the Metropolis algorithm is presented briefly in Section 4. Image processing applications are given in Section 5. Finally the advantages and drawbacks of the proposed method and future research are discussed in the Discussion and conclusions section.

2. Discrete-time CNNs

The dynamics of a single discrete-time CNN (DTCNN) cell can be defined with the following state and output equations, respectively:

$$x_{ij}(n+1) = \sum_{C(k,l)\in S_r(i,j)} A(i,j;k,l)v_{kl}(n) + \sum_{C(k,l)\in S_r(i,j)} B(i,j;k,l)u_{kl} + z, \tag{1}$$

and

$$v_{ij}(n) = f(x_{ij}(n)) = \frac{1}{2}|x_{ij}(n)+1| - \frac{1}{2}|x_{ij}(n)-1|, \tag{2}$$

where u, v, and x denote input, output, and state, respectively. State variable x is initialized with an initial value before input data are processed. A and B in the state equation Eq. (1) are called the cloning template and control template, respectively. Each element of a template represents the synaptic weights between a cell and its neighbors. z is the cell bias and is constant for every cell. S_r is the neighborhood of cell C with radius r and r determines how many neighbor cells are interconnected with each other. The output equation states that output v is a function of state x; thus the system is recursive.

A DTCNN can be of any $M \times N$ size. Size of a DTCNN usually matches the size of its input. Output of a stable DTCNN, which is denoted by f_∞, converges to $\{+1, -1\}$, which is called the binary output property. The number of iterations in the recursive structure of a DTCNN to satisfy this convergence is indefinite and usually depends on the application and size of input data. It is known that a DTCNN is stable if the cloning template weights are symmetric. Furthermore, a symmetric CNN has the binary output property if the self feedback parameter of cloning template A is greater than 1 [15].

3. Bayesian estimation of CNN templates

3.1. Bayesian learning

Bayesian learning is the process of updating the probability estimate for a hypothesis as additional prior knowledge is gathered and it is an important probabilistic technique used in various disciplines. Bayesian learning is used in image processing [16], machine learning [17], mathematical statistics [18], engineering [19], philosophy [20], medicine [21], and law [22].

The Bayesian framework [23] is distinguished by its use of probability to express all forms of uncertainty. Bayesian learning results in a probability distribution of system parameters that yield how likely the various parameter values are. Before applying Bayesian learning, a prior distribution in terms of prior information must be defined. After observations are made and using the prior distribution, a posterior distribution is obtained with respect to the Bayes theorem. The resulting posterior distribution contains information from both the observation of parameters (likelihood) and the background knowledge about the parameters (prior).

Computational difficulties arise when the resulting posterior density function is difficult or impossible to sample with procedural methods. That is where MCMC methods come to the rescue. MCMC methods are used to overcome the computational problems in Bayesian learning.

3.2. System model

To make use of probability theory, the DTCNN system is transformed into an estimation problem by adding noise to the output of the system. This results in the following system model:

$$y[n] = f_{\infty}^{[n]}(\boldsymbol{u}, \boldsymbol{\theta}) + \omega[n] \quad , n = 0, 1, ..., N-1, \tag{3}$$

where each $u[n]$ and $y[n]$ corresponds to a pixel of the input image and its output in the output image, respectively, as images are considered as input and output. In vector notation for all input–output pairs Eq. (3) can be given as follows: [23–25]:

$$\boldsymbol{y} = f_{\infty}(\boldsymbol{u}, \boldsymbol{\theta}) + \boldsymbol{\omega}, \tag{4}$$

where $\boldsymbol{\theta}$ is the vector consisting of the network coefficients in Eq. (1). The network coefficients in Eq. (1) are chosen as shown in Eq. (5) for reducing the computational cost. Each of $\mathbf{A}$ and $\mathbf{B}$ has a 3×3 template size and contains only 5 unknown coefficients as follows:

$$\mathbf{A} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_4 \\ \theta_3 & \theta_2 & \theta_1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \theta_6 & \theta_7 & \theta_8 \\ \theta_9 & \theta_{10} & \theta_9 \\ \theta_8 & \theta_7 & \theta_6 \end{bmatrix}, \quad z = \theta_{11} \tag{5}$$

where z represents the bias coefficient. Therefore the coefficient vector $\boldsymbol{\theta} \in \mathbf{R}^{11\times 1}$ can be defined as

$$\boldsymbol{\theta} = [\theta_1 \ \ \theta_2 \ \ \theta_3 \ \ \theta_4 \ \ \theta_5 \ \ \theta_6 \ \ \theta_7 \ \ \theta_8 \ \ \theta_9 \ \ \theta_{10} \ \ \theta_{11}]^{\top}. \tag{6}$$

Additive Gaussian noise ω with a distribution $\mathcal{N}(0, \sigma_{\omega}^2)$ transforms the problem of finding the network coefficients into an estimation problem [25].

To obtain the posterior PDF, likelihood and prior PDFs must be acquired first. Likelihood $p(\boldsymbol{y}|\boldsymbol{\theta})$ is a Gaussian distribution with mean $f_\infty(\boldsymbol{u},\boldsymbol{\theta})$ and variance σ_ω^2,

$$\boldsymbol{y}|\boldsymbol{\theta} \sim \mathcal{N}(f_\infty(\boldsymbol{u},\boldsymbol{\theta}), \sigma_\omega^2 \boldsymbol{I}) \tag{7}$$

$\{y(n)|\boldsymbol{\theta}\}_{n=0}^{N-1}$ are independent from each other; therefore $p(\boldsymbol{y}|\boldsymbol{\theta})$ can be written as follows:

$$p(\boldsymbol{y}|\boldsymbol{\theta}) = \prod_{n=0}^{N-1} p(y(n)|\boldsymbol{\theta}), \tag{8}$$

and the resulting $p(\boldsymbol{y}|\boldsymbol{\theta})$ is written as follows:

$$p(\boldsymbol{y}|\boldsymbol{\theta}) = \prod_{n=0}^{N-1} \frac{1}{\sqrt{2\pi\sigma_\omega^2}} e^{-\frac{(y[n]-f_\infty^{[n]}(\boldsymbol{u},\boldsymbol{\theta}))^2}{2\sigma_\omega^2}} \tag{9}$$

The output of function $\mathbf{f}_\infty$ is a vector and stands for $\mathbf{f}_\infty^{[\mathbf{n}]}$ nth value of function $\mathbf{f}_\infty$.

Prior distribution of $\boldsymbol{\theta}$ is also assumed to be $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta)$, where the mean vector and the covariance matrix are given as

$$\boldsymbol{\mu_\theta} = [\mu_1, \mu_2, ..., \mu_{11}]^\top, \tag{10}$$

and

$$\boldsymbol{\Sigma}_\theta = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{11}^2 \end{bmatrix}, \tag{11}$$

respectively. Since the parameter vector $\boldsymbol{\theta}$ has independent elements the prior PDF of $\boldsymbol{\theta}$ is written as

$$p(\boldsymbol{\theta}) = \frac{1}{\sqrt{(2\pi)^{11}|\boldsymbol{\Sigma}_\theta|}} e^{-\frac{(\boldsymbol{\theta}-\boldsymbol{\mu_\theta})^\top \boldsymbol{\Sigma}_\theta^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu_\theta})}{2}}. \tag{12}$$

Applying the Bayes theorem, the posterior PDF is obtained with respect to likelihood Eq. (9) and prior Eq. (12) PDFs as follows:

$$\begin{aligned} p(\boldsymbol{\theta}|\boldsymbol{y}) &\propto p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \\ &= \prod_{n=0}^{N-1} \frac{1}{\sqrt{2\pi\sigma_\omega^2}} e^{-\frac{(y[n]-f_\infty^{[n]}(\boldsymbol{u},\boldsymbol{\theta}))^2}{2\sigma_\omega^2}} \\ &\quad \times \frac{1}{\sqrt{(2\pi)^{11}|\boldsymbol{\Sigma}_\theta|}} e^{-\frac{(\boldsymbol{\theta}-\boldsymbol{\mu_\theta})^\top \boldsymbol{\Sigma}_\theta^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu_\theta})}{2}} \end{aligned} \tag{13}$$

The posterior PDF of Eq. (13) is not differentiable. Thus to obtain an estimation of $\boldsymbol{\theta}$, samples are drawn from the posterior PDF of Eq. (13) using the Metropolis algorithm and then averaged to obtain an MMSE estimation of $\boldsymbol{\theta}$.

The resulting sample sequences of network coefficients are in the following form:

$$\boldsymbol{\theta}(k)|\mathbf{y} = [\theta_1(k)|\mathbf{y}, \theta_2(k)|\mathbf{y}, ..., \theta_{11}(k)|\mathbf{y}]^\top, \quad k = 0, 1, ..., K-1, \tag{14}$$

where K is the total number of samples generated. The MMSE estimator of $\boldsymbol{\theta}$ vector, which is defined as the posterior expectation of $\boldsymbol{\theta}$, can be approximately obtained by averaging the samples as follows:

$$\hat{\boldsymbol{\theta}}_{MMSE} = E[\boldsymbol{\theta}|\boldsymbol{y}] \approx \frac{1}{K}\sum_{k=0}^{K-1} \boldsymbol{\theta}(k)|\mathbf{y}. \tag{15}$$

The resulting estimation of $\boldsymbol{\theta}$ in vector form is

$$\hat{\boldsymbol{\theta}}_{MMSE} = \left[\hat{\theta}_1, \hat{\theta}_2, ..., \hat{\theta}_{11}\right]^\top \tag{16}$$

and these coefficients are then used for testing after elements of $\boldsymbol{\theta}$ in Eq. (6) are distributed back to their places in $\mathbf{A}$, $\mathbf{B}$, and z in Eq. (5).

4. Generation of a template sample using the Metropolis algorithm

The Metropolis algorithm [13], a special case of the Metropolis–Hastings algorithm [14], is one of the MCMC techniques to generate samples from a desired distribution where it is hard or impossible to generate samples by using standard techniques. This method uses a symmetric proposal density to generate new samples starting with an initial point. These generated new sample candidates are given an acceptance ratio by using the desired distribution PDF and accepted according to this acceptance ratio.

Before the samples are generated, an initial point is chosen. This point may be predicted according to the prior information,

$$\boldsymbol{\theta}(0) = [\theta_0(0), \theta_1(0), ..., \theta_{11}(0)]^\top, \tag{17}$$

where $\boldsymbol{\theta}(0)$ is the vector of initial points for each network coefficient.

After the initial points are determined, a new sample vector $\boldsymbol{\theta}_{new}$ is generated according to the proposal density. For the Metropolis algorithm the proposal density function must be a symmetric distribution. Therefore in this work the proposal density function is chosen to be a Gaussian distribution function. For the new sample vector to take its place in the sample sequence, it is trailed using the posterior distribution of Eq. (13). To manage this, the following proportion, which is denoted by α, is evaluated:

$$\alpha = \frac{p(\boldsymbol{\theta}_{new}|\boldsymbol{y})}{p(\boldsymbol{\theta}(k-1)|\boldsymbol{y})} \tag{18}$$

where $p(\boldsymbol{\theta}_{new}|\boldsymbol{y})$ is the posterior distribution evaluated with the new sample vector and $p(\boldsymbol{\theta}(k-1)|\boldsymbol{y})$ is the posterior distribution evaluated with the previous sample vector. The constants in the nominator and denominator of Eq. (13), which does not involve any variables, cancel out in the proportion of Eq. (18) and the products of $p(\boldsymbol{y}|\boldsymbol{\theta})$ can be written as a sum in the exponent.

If $\alpha \geq 1$ then the new sample vector is inserted directly into the sample sequence. If $\alpha < 1$ then it is accepted with α probability. To manage this, a random number denoted by α' is generated from a uniform distribution of $U(0,1)$ and then α is compared with α'. In the case where $\alpha \geq \alpha'$, the new sample vector is inserted into the sample sequence. Otherwise the preceding sample vector is inserted into the sample sequence. After the new sample candidate or the previous one is inserted into the sample sequence, the loop starts again by producing a new sample vector by using the proposal density and continues until the intended number of samples is generated. More details can be found in [26, 27].

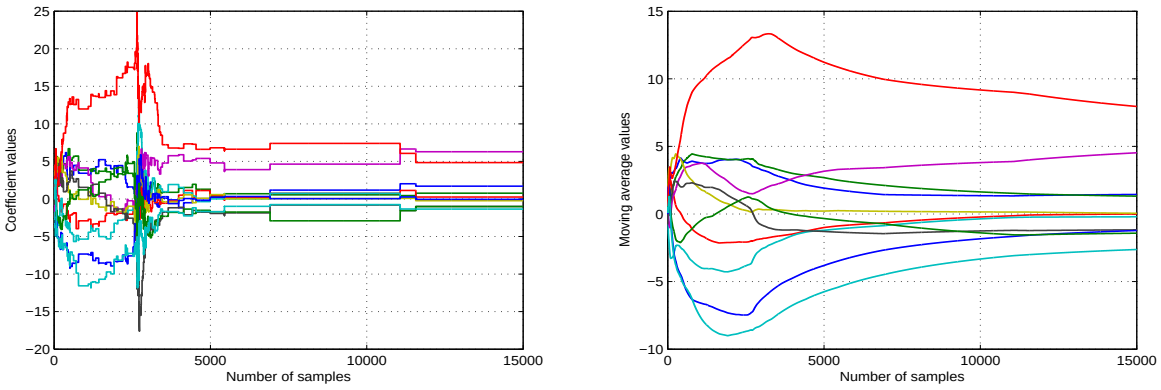
5. Applications

In the training steps, for the edge detection, noise removal, and hole filling applications that are demonstrated below, Eq. (1) and Eq. (2) are used to generate $f_\infty(\boldsymbol{u}, \boldsymbol{\theta}(k))$ for each sample. 15,000 Samples are used for training and during training the number of iterations for each sample is limited according to the application and input image size whether the DTCNN converges or not. This is because the stability criteria of the DTCNN do not guarantee the number of iterations required for the convergence of the DTCNN. The initial value of the state x is selected according to the application. For each application, first the synthetic training images are used to estimate DTCNN coefficients and then some test images are applied to the DTCNN using estimated coefficients to find output. It must be noted that the input and output images used in training are different from those used in testing. Initial values of the coefficients are drawn from a Gaussian distribution $\mathcal{N}(0, 0.64)$ and during the inference the variance of the proposal distribution in Metropolis is kept constant.

5.1. Edge detection

For an edge detection application, the edges in the input image are detected using the estimated coefficients and the results are compared with Canny and Sobel edge detectors. Coefficients $\mathbf{A}$, $\mathbf{B}$, and z are given below. The changes and moving average convergence of the coefficients are also shown in Figure 1a and Figure 1b, respectively. Initial values for state x are selected as zeros for this application.

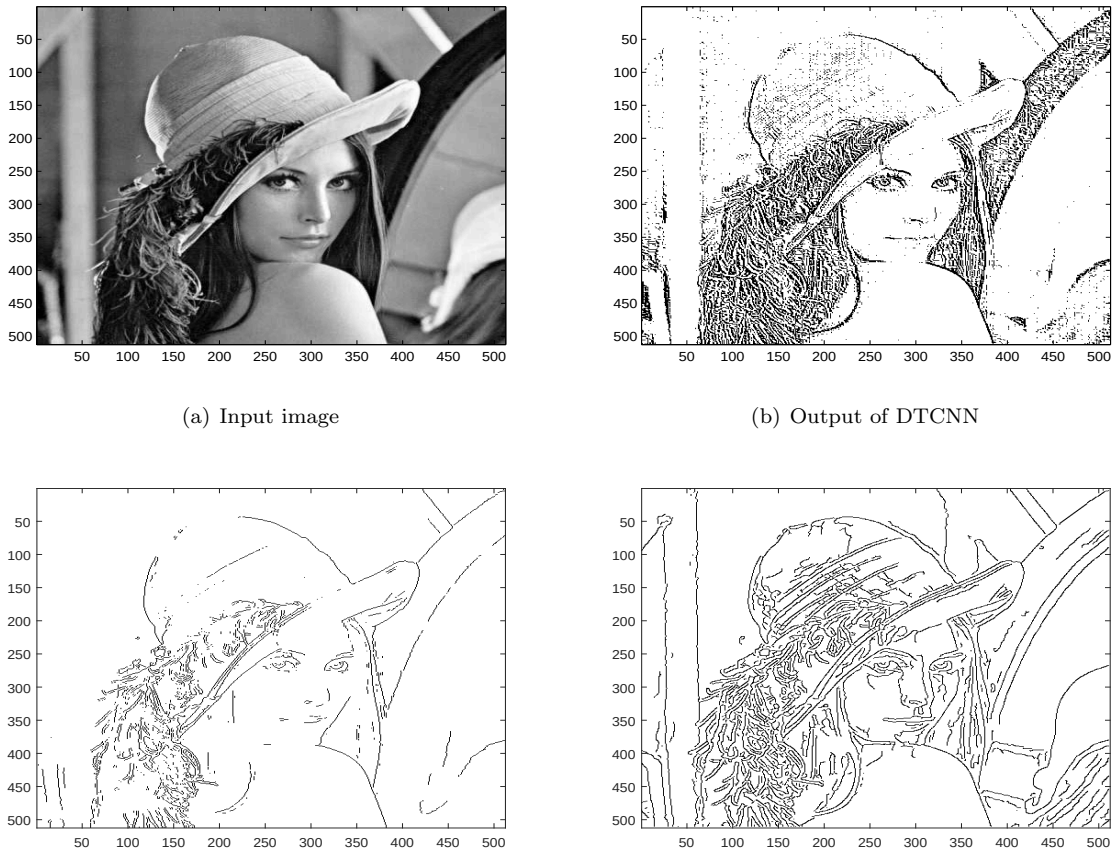
$$\mathbf{A} = \begin{bmatrix} 1.4461 & 1.3190 & 0.0012 \\ -0.2081 & 4.5272 & -0.2081 \\ 0.0012 & 1.3190 & 1.4461 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0.0546 & -1.1733 & -1.2234 \\ -1.4206 & 7.9548 & -1.4206 \\ -1.2234 & -1.1733 & 0.0546 \end{bmatrix}, z = -2.6278.$$



(a) Changes of the coefficients

(b) Moving average

Figure 1. Convergence of the coefficients for the edge detection application.



(a) Input image

(b) Output of DTCNN

(c) Output of Sobel edge detector

(d) Output of Canny edge detector

Figure 2. Results of edge detection.

Input and output images are shown in Figure 2. When compared with Sobel and Canny edge detectors, as can be seen in Figure 2, the CNN filter gives finer details. The Sobel edge detector suffers from the continuity problem. Like other classical edge detectors based on gradient estimation, the Canny edge detector suffers from aliasing.

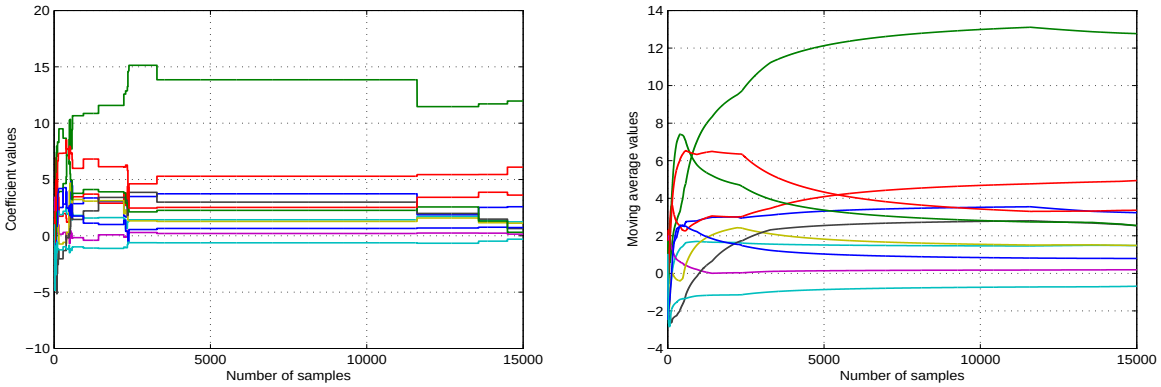
5.2. Noise removal

In this application, CNN coefficients are obtained using a Gaussian noise image with SNR value of -10 dB. Then, using the estimated coefficients, the system is tested with a Gaussian noise test image with SNR value of -5 dB and salt and pepper noise test image with 50% noise. The estimated coefficients $\mathbf{A}$, $\mathbf{B}$, and z are given below.

$$\mathbf{A}=\begin{bmatrix}3.2314 & 12.7699 & 4.9418\\ 1.4786 & 0.1876 & 1.4786\\ 4.9418 & 12.7699 & 3.2314\end{bmatrix},\ \mathbf{B}=\begin{bmatrix}1.4838 & 2.5393 & 0.7922\\ 2.5469 & 3.3612 & 2.5469\\ 0.7922 & 2.5393 & 1.4838\end{bmatrix},\ z=-0.6867$$

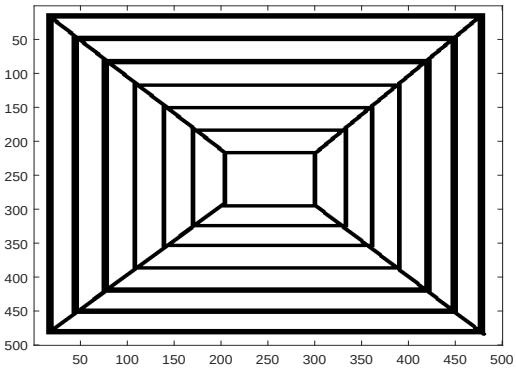
The changes and moving average convergence of all coefficients are also shown in Figure 3a and Figure 3b, respectively.

Initial values for state x are selected as zeros for this application. The original image is shown in Figure 4a. The noisy images are shown in Figure 4b and Figure 4c, respectively. Obtained results are compared with those of the Wiener and median filters [28] in terms of peak signal-to-noise ratio (PSNR). Filtered outputs are

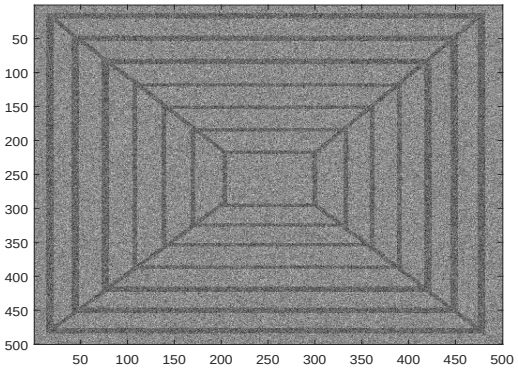


(a) Changes of the coefficients (b) Moving average

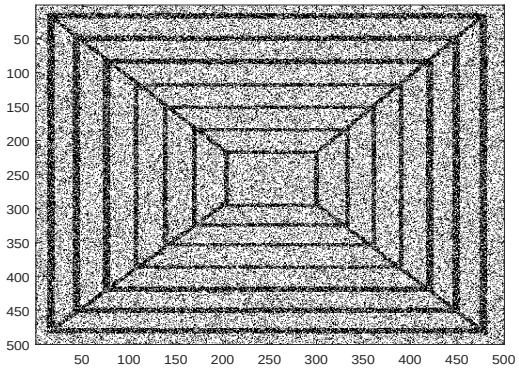
Figure 3. Convergence of the coefficients for the noise removal application.



(a) Original image



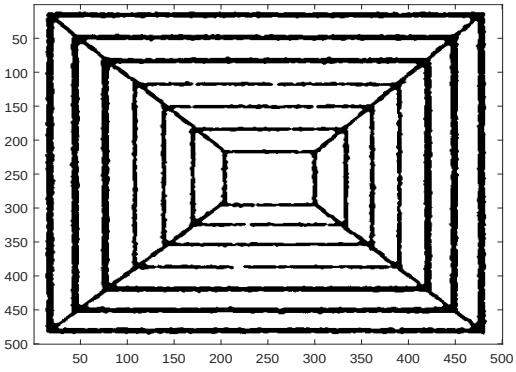
(b) Gaussian noise image



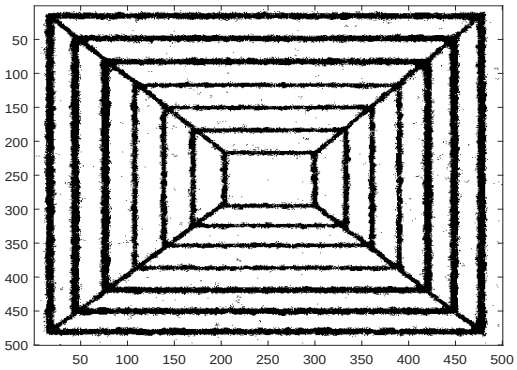
(c) Salt and pepper noise image

Figure 4. Original and noisy images.

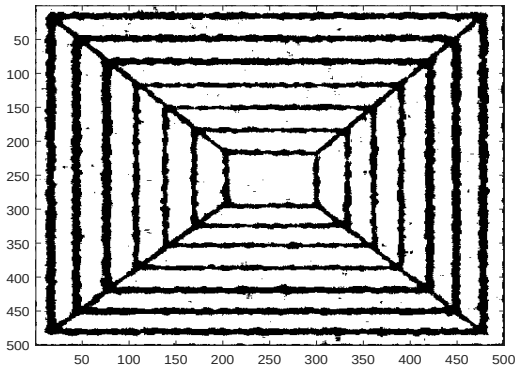
shown in Figure 5 and Figure 6. For a fair comparison windows size is selected as 7×7 for each filter since the maximum PSNR value is detected for only 7×7 windows size. For the Gaussian noise image, CNN, Wiener, and median filter PSNR values are obtained as 13.22 dB, 12.05 dB, and 11.68 dB, respectively. For the salt and pepper noise image PSNR values are obtained as 14.72 dB, 12.33 dB, and 12.70 dB, respectively. In the Gaussian noise image, the maximum PSNR value is obtained with the Wiener filter. In the salt and pepper noise image the maximum PSNR value is obtained with the CNN filter. In general, the results are close to each other, but Wiener and median filters suffer from finding the optimum windows size.



(a) Output of DTCNN



(b) Output of Wiener filter



(c) Output of Median filter

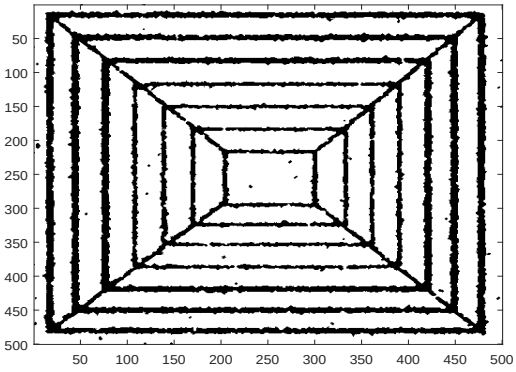
Figure 5. Results of Gaussian noise removal.

5.3. Hole filling

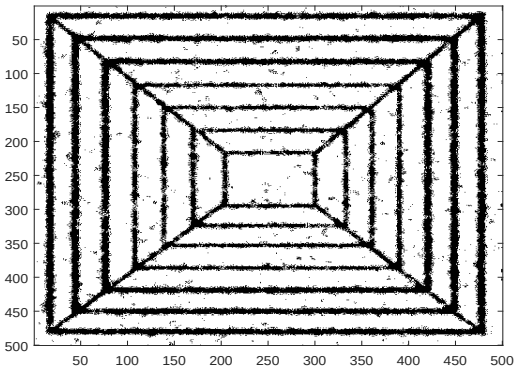
A hole filling application where the objects in the input image are filled is performed using the estimated coefficients for $\mathbf{A}$, $\mathbf{B}$, and z below,

$$\mathbf{A}=\begin{bmatrix} 1.5125 & 0.9413 & -0.1290 \\ 3.0592 & -0.0642 & 3.0592 \\ -0.1290 & 0.9413 & 1.5125 \end{bmatrix}, \mathbf{B}=\begin{bmatrix} -0.7045 & 1.2622 & 1.1628 \\ 0.1881 & 5.6201 & 0.1881 \\ 1.1628 & 1.2622 & -0.7045 \end{bmatrix}, z=1.7897$$

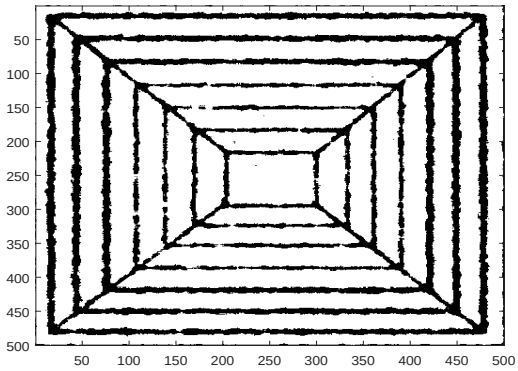
Initial values of the state x are selected as ones for this application. Changes and moving average convergence of the coefficients are shown in Figure 7a and Figure 7b, respectively. The input and output images are shown



(a) Output of DTCNN



(b) Output of Wiener filter



(c) Output of Median filter

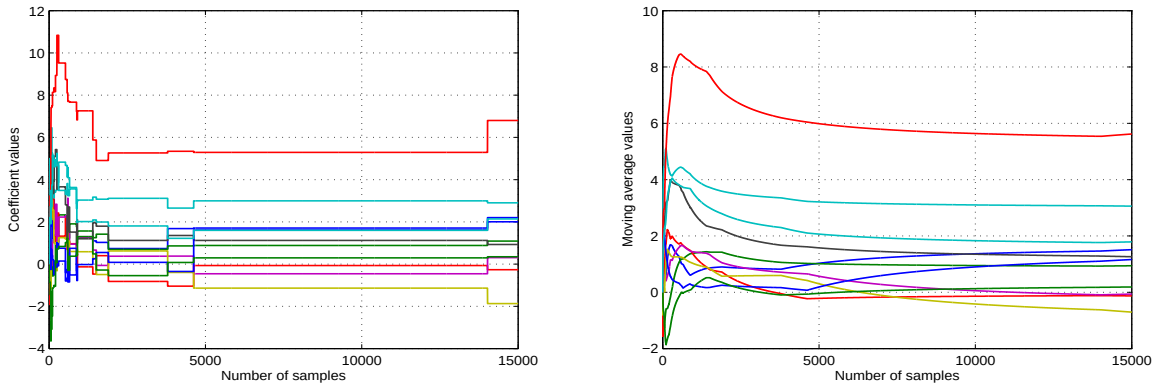
Figure 6. Results of salt and pepper noise removal.

in Figure 8a and Figure 8b. In this application the result is compared with the *MATLAB imfill* function and the same output is obtained. However, the *imfill* function has superiority over the CNN when compared in terms of calculation time. While the *imfill* function uses an algorithm based on morphological reconstruction, the CNN uses only basic two-dimensional filter techniques recursively.

6. Discussion and conclusions

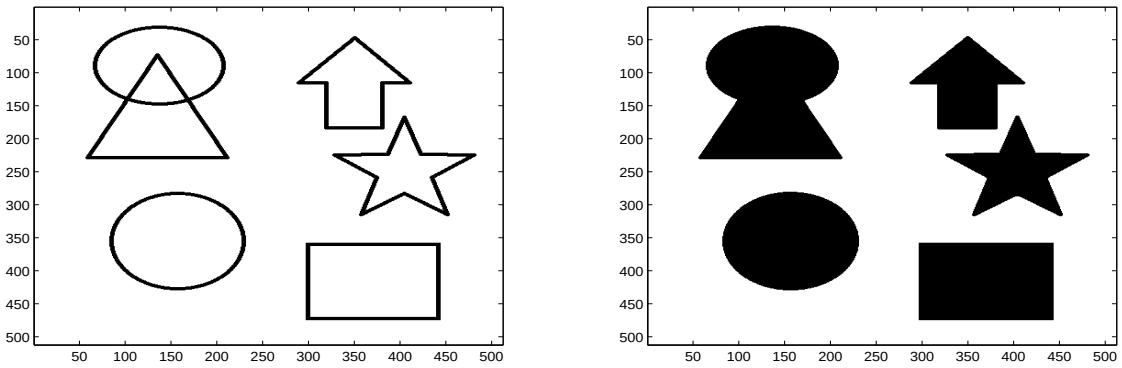
The suggested Bayesian estimation method to find the coefficients for a CNN is a probabilistic method in its design. Prior information about the network parameters, if available, can give better estimation results for these parameters.

In this study, at each step of the Metropolis algorithm, division of two posterior distributions, α, is calculated from Eq. (18) and then decision making is done for acceptation and rejection with respect to α. In the RPLA algorithm, a set of difference equations is defined to obtain DTCNN coefficients. The algorithm renews the value of the template coefficient vector according to the RPLA update rule. As seen in [8], the update rule contains only simple operations such as a few multiplications and additions. In the MLP, the gradient vector that is used to update NN coefficients is calculated by employing the backpropagation algorithm. As compared with the RPLA update rule and the Metropolis algorithm, the backpropagation algorithm requires much more multiplication and addition operations, resulting in higher computational complexity.



(a) Changes of the coefficients (b) Moving average

Figure 7. Convergence of the coefficients for the hole filling application.



(a) Input image (b) Output image

Figure 8. Results of hole filling.

One disadvantage is the correlated samples resulting from the Metropolis–Hastings algorithm. Correlation of the samples requires more samples to be taken into account to overcome the correlation of two or more consequent samples. This also results in a slower convergence for the network parameters. Throwing away the majority of samples and taking only every nth sample can reduce correlation of samples. Moreover, the current implementation uses the Gaussian PDF as a symmetric distribution. Using an asymmetric distribution may increase the convergence speed of the estimated parameters. When compared in terms of calculation time, because of the iterative structure of the DTCNN, the proposed method has no superiority over most of the other methods. However, while each of the other methods uses different algorithms for each of the applications, the DTCNN uses only simple and the same two-dimensional filter techniques for different applications.

References

[1] Chua LO, Yang L. Cellular neural networks: Theory. IEEE T Circuits Syst 1988; 35:1257-1272.

[2] Chua LO, Yang L. Cellular neural networks: Applications. IEEE T Circuits Syst 1988; 35: 1273-1290.

[3] Kim K, Lee S, Kim JY, Kim M, Yoo HJ. A configurable heterogeneous multicore architecture with cellular neural network for real-time object recognition. IEEE T Circ Syst Vid 2009; 19-11: 1612-1622.

[4] Costantini G, Casali D, Carota M. CNN-Based unsupervised pattern classification for linearly and non linearly separable data sets. WSEAS Transactions on Circuits and Systems 2005; 4-5: 448-452.

[5] Kananen A, Paasio A, Laiho M, Halonen K. CNN applications from the hardware point of view: Video sequence segmentation. Int J Circ Theor App 2002; 30-(2-3): 117-137.

[6] Nossek JA. Design and learning with cellular neural networks. In: Third IEEE International Workshop on Cellular Neural Networks and their Applications, CNNA-94; 18–21 December 1994; Rome, Italy. pp. 137-146.

[7] Vinyoles-Serra M, Jankowski S, Szymanski Z. Cellular neural network learning using multilayer perceptron. In: 20th European Conference on Circuit Theory and Design, ECCTD2011; 29–31 August 2011; Linkping, Sweden. pp. 214-217.

[8] Güzeliş C, Karamahmut S. Recurrent perceptron learning algorithm for completely stable neural networks. In: Cellular Neural Networks and their Applications, CNNA-94; 18–21 December 1994; Rome, Italy. pp. 177-182.

[9] Luitel B, Venayagamoorthy GK. Decentralized asynchronous learning in cellular neural networks. IEEE T Neural Netw Learn Syst 2012; 23:1755-1766.

[10] Kozek T, Roska T, Chua LO. Genetic algorithm for CNN template learning. IEEE T Circuits Syst 1993; 40-6: 392-402.

[11] Ünal M, Onat M, Bal A. Cellular neural network training by ant colony optimization algorithm. In: 2010 IEEE 18th Signal Processing and Communications Applications Conference, SIU2010; 22–24 April 2010; Diyarbakır, Turkey. pp. 1661-1666.

[12] Moreno-Armendariz MA, Pazienza GE, Yu W. Training cellular neural networks with stable learning algorithm. Lect Notes Comput Sc 2006; 3971: 558-563.

[13] Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equations of state calculations by fast computing machines. J Chem Phys 1953; 21: 1087-1092.

[14] Hastings WK. Monte Carlo sampling methods using Markov chains and their applications. Biometrika 1970; 57: 97-109.

[15] Arik S. Stability analysis of dynamical neural networks. PhD, South Bank University, London, UK, 1997.

[16] Ding X, He L, Carin L. Bayesian robust principal component analysis. IEEE T Image Process 2011; 20-12: 3419-3430.

[17] Tipping ME. Sparse Bayesian learning and the relevance vector machine. J Mach Learn Res 2001; 1: 211-244.

[18] Ding M, He L, Dunson D, Carin L. Nonparametric Bayesian segmentation of a multivariate inhomogeneous space-time poisson process. Bayesian Anal 2012; 7-4: 813-840.

[19] Fox EB, Sudderth EB, Jordan MI, Willsky AS. Bayesian nonparametric inference of switching dynamic linear models. IEEE T Signal Proces 2011; 59-4: 1569-1585.

[20] Hartmann S, Sprenger J. Bayesian epistemology. Routledge Companion to Epistemology 2010; 609-620.

[21] Kononenko I. Inductive and Bayesian learning in medical diagnosis. Appl Artif Intel 1993; 7-4: 317-337.

[22] Parisi F, Scharff RL. The role of status quo bias and Bayesian learning in the creation of new legal rights. J Law Econ 2006; 7-10: 1-27.

[23] Neal RM. Bayesian learning for neural networks. PhD, University of Toronto, Toronto, Canada, 1995.

[24] Freitas JFG. Bayesian methods for neural networks. PhD, University of Cambridge, Cambridge, UK, 1999.

[25] Özer HM. Bayesian learning for cellular neural networks. MSc, Kadir Has University, İstanbul, Turkey, 2013.

[26] MacKay DJC. Information Theory, Inference and Learning Algorithms. Cambridge, UK: Cambridge University Press, 2003.

[27] Bishop CM. Pattern Recognition and Machine Learning. New York, NY, USA: Springer, 2006

[28] Lim JS. Two Dimensional Signal and Image Processing. Upper Saddle River, NJ, USA: Prentice Hall, 1990.