國 立 交 通 大 學

電機與控制工程學系

博 士 論 文

模糊細胞神經網路整合系統

A Fuzzy Cellular Neural Network Integrated System

研 究 生：張 俊 隆

指導教授：林 進 燈

中 華 民 國 九 十 五 年 一 月

模糊細胞神經網路整合系統

A Fuzzy Cellular Neural Network Integrated System

研 究 生：張俊隆　　　　Student：Chun-Lung Chang

指導教授：林進燈 博士　　　Advisor：Dr. Chin-Teng Lin

國 立 交 通 大 學

電 機 與 控 制 工 程 學 系

博 士 論 文

A Dissertation
Submitted to Department of Electrical and Control Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
Electrical and Control Engineering
January 2006
Hsinchu, Taiwan, Republic of China

中華民國九十五年一月

模糊細胞神經網路整合系統

摘要

無論從應用或生物觀點，使用一組平行的細胞神經網路(cellular neural networks, CNN)以完成高階資訊處理與推論能力已被廣泛的接受。如此的整合型細胞神經網路可以解決比較複雜的問題。本論文提出一種以遞迴式模糊神經網路（recurrent fuzzy neural network）的架構以自動建構CNN整合系統。這個稱為RFCNN/RFCCNN (recurrent fuzzy CNN/ recurrent fuzzy coupled CNN)的整合系統可以同時自動的學習CNN的網路結構與參數。其網路結構學習包括模糊規則（fuzzy rules）與CNN數目的建立；其參數學習包括模糊歸屬函數與CNN模板（template）參數的學習。在RFCNN/RFCCNN中，每個模糊規則對應一個CNN。一個新的線上適應獨立元件分析混合模型（on-line adaptive ICA (independent component analysis) mixture-model technique）技術被提出以做為RFCNN的結構學習；另外，次序微分（ordered-derivative）可做為RFCNN/RFCCNN的參數學習。本論文所提出RFCNN/RFCCNN對於既存CNN整合系統同時地建立模糊規則與學習CNN模板參數的兩難提供一個解決方案。本論文最後並以工業上瑕疵檢測的問題做為說明RFCNN/RFCCNN系統的能力，實驗結果顯示所提出的方法是有效且有潛力的。

A Fuzzy Cellular Neural Network Integrated System

Student: Chun-Lung Chang Advisor: Chin-Teng Lin
Department of Electrical and Control Engineering
National Chiao-Tung University

Abstract

It is widely accepted that using a set of cellular neural networks (CNNs) in parallel can achieve higher-level information processing and reasoning functions either from application or biologics points of views. Such an integrated CNN system can solve more complex intelligent problems. In this thesis we propose two novel frameworks for automatically constructing a multiple-CNN integrated neural system in the form of a recurrent fuzzy neural network. The systems, called recurrent fuzzy CNN (RFCNN) and recurrent fuzzy coupled CNN (RFCCNN), can automatically learn its proper network structure and parameters simultaneously. The structure learning includes the fuzzy division of the problem domain and the creation of fuzzy rules and CNNs. The parameter learning includes the tuning of fuzzy membership functions and CNN templates. In the RFCNN/RFCCNN, each learned fuzzy rule corresponds to a CNN. Hence, each CNN takes care of a fuzzily separated problem region, and the functions of all CNNs are integrated through the fuzzy inference mechanism. A new on-line adaptive ICA (independent component analysis) mixture-model technique is proposed for the structure learning of RFCNN/RFCCNN, and the ordered-derivative calculus is applied to derive the recurrent learning rules of CNN templates in the parameter-learning phase. The proposed RFCNN/RFCCNN provides a solution to the current dilemma on the decision of templates and/or fuzzy rules in the existing integrated (fuzzy) CNN systems. The capability of the proposed RFCNN and RFCCNN are demonstrated and compared on the real-world defect inspection problems. Experimental results show that the proposed scheme is effective and promising.

誌謝

首先感謝指導教授林進燈院長多年來的指導。無論是專業上或是生活上的教導，都使我受益良多。林進燈教授的學識淵博、做事衝勁十足、熱心待人誠懇與風趣幽默等特質，都是非常值得學習的地方。對於本論文的完成，除了林教授的指導以外，也非常感謝諸位口試委員寶貴的意見，使得本論文更加完備。

在家人方面，首先感謝我的妻小：淑娟與皓恩；多年來因為學業的關係使得陪你們的時間相對的減少，日後當會好好的補償你們。此外感謝雙親張建皇與蕭秀鳳多年以來的支持與岳父岳母邱乃乾與傅美蘭的鼓勵。由於您們長年以來的支持與鼓勵，使得我無後顧之憂的專心於學業方面。

在學校方面，感謝鶴章博士、文昌博士、群立、世茂、朝暉等同學在學業上與生活上的幫忙與照顧。多年來，因為有你們的參與，使得我在交通大學的求學過程中更加的多采多姿。

在公司方面，感謝工研院機械所長官、部門經理與部門同事多年來的支持，得以使我能夠順利完成本論文。

謹以本論文獻給我的家人與關心我的師長與朋友們。

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

The two-dimensional inputs and outputs of the Cellular Neural Networks (CNN) [1], [2], make it very suitable for image processing. A single CNN can solve a basic task such as thresholding and filtering, etc. However, either from application or biologics points of views, several CNNs in parallel or in series can solve more complex intelligent problems, such as edge detection with impulse noise, the detection of fuzzy boundary, and features extraction, etc. To solve more complex problems, several CNNs can be integrated to solve specific problem. In this thesis, we propose a novel framework to integrate a set of CNNs in parallel in order to solve more complex intelligent problems.

1.2. Cellular Neural Network

The CNN first introduced as one that is able to implement alternative to fully connected neural networks, has evolved into a paradigm for these types of arrays. The block diagram of the CNN is shown in Fig. 1.1, and its dynamics is described by the following differential equations:

$$\frac{d}{dt}x_{i,j}(t) = -x_{i,j}(t) + \sum_{k,l\in N_r} a_{k,l}y_{i+k,j+l}(t) + \sum_{k,l\in N_r} b_{k,l}u_{i+k,j+l}(t) + z_{i,j} \tag{1.1}$$

$$N_r(i,j) = \{C(k,l)\,|\,\max\{|k-i|,|l-j|\} \le r\} \tag{1.2}$$

with output nonlinearity

$$y_{i,j}(t) = \frac{1}{2}(|x_{i,j}(t)+1| - |x_{i,j}(t)-1|), \tag{1.3}$$

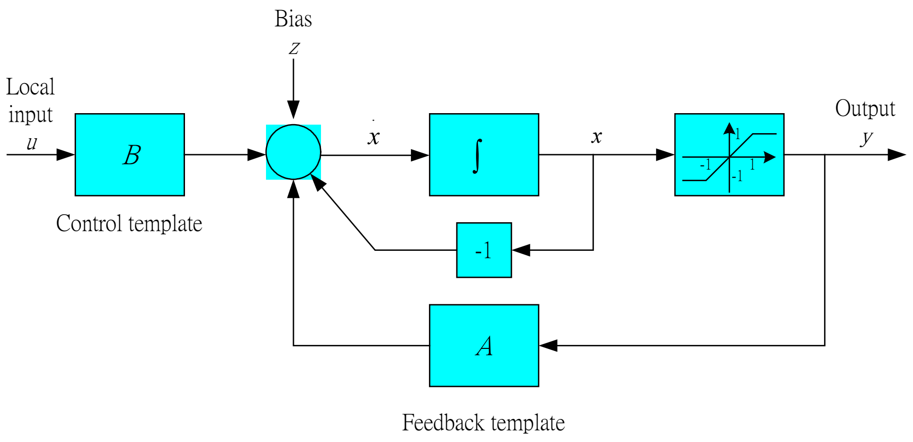


Figure 1.1 Block diagram of a CNN.

where $x_{i,j}$ is the internal state of a cell, $y_{i,j}$ is the output, $u_{i,j}$ is external input and $z_{i,j}$ is a local value called bias, (*i, j*) is a grid point associated with a cell on the 2-D grid, and (*k, l*) is a grid point in the neighborhood within a radius *r* of the cell (*i, j*). That is, *C*(*k, l*) is a cell in the neighborhood within a radius *r* of the cell *C*(*i, j*) and $N_r(i, j)$ is a set including all *C*(*k, l*) associated with a cell *C*(*i, j*). The *A* and *B* are two generic parametric functions called feedback template and control template, respectively. A CNN has a space invariant local interconnection structure associated with 19 free parameters (neighborhood within a radius *r* = 1), which exclusively determines the dynamic behavior of the CNN.

The CNN possesses some important characteristics such as efficient real-time processing capability and feasible VLSI implementation. Some applications requiring high-speed processing include real-time object recognition and tracking, high-speed visual inspection of manufacturing processes, etc.

1.3. CNN Integrated System

Besides some basic image processing tasks, the CNN has been used to mimic the

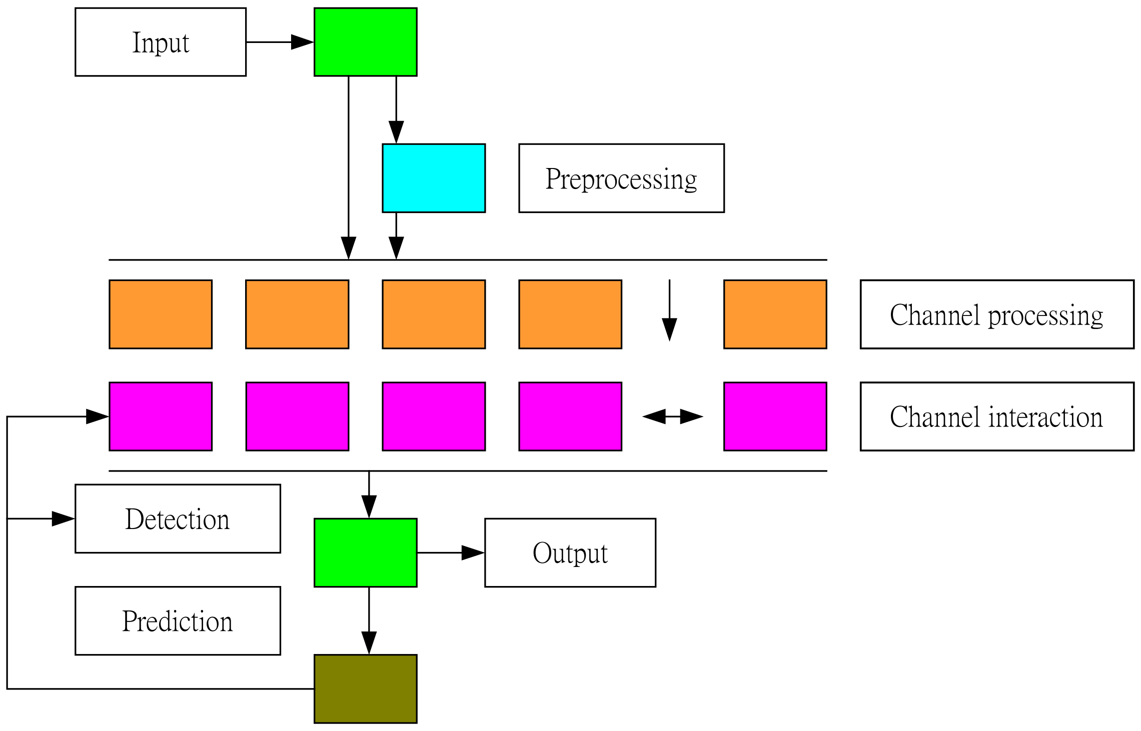


Figure 1.2 The framework of the multi-channel adaptive CNN algorithm.

local function of biological neural circuits, especially the human visual pathway system [3]. According to a current biological study [4], mammalian visual systems process the world through a set of separate parallel channels. As shown in Fig. 1.2, each sub-channel can be regarded as a unique CNN. The output of these sub-channels is then combined to form the new channel responses. One key point is to define the global channel interaction and result in a unique binary image flow. As a result, it is widely accepted that using a set of CNNs in parallel can achieve higher-level information processing and reasoning functions either from biologics or application points of views. Such an integrated CNN system can solve more complex intelligent problems.

For designing an integrated CNN system, in addition to the determination of a set of templates, another kernel problem is the way of integration. To solve this problem, the fuzzy inference system (FIS) is gaining attention. The FIS is a popular computing framework based on the concept of fuzzy set theory, fuzzy if-then rules, and fuzzy reasoning. With crisp inputs and outputs, fuzzy inference system implements a nonlinear mapping from its input space to output space by a number of if-then rules. It is very useful in image processing when it is difficult to specify, in a crisp mathematical form, the operation that is needed to yield a satisfying result from a complex image. For example, the boundary detection of different regions strongly depends on a subjective decision, especially in medical image. It cannot be clearly defined what is an edge-like and what is a noise-like pattern. In many cases both statements might be true; therefore a fuzzy-type linguistic description of all patterns is better than a crisp set approach. Therefore, FIS can play an important role to integrate a set of CNNs into a system.

1.3.1. Existing Fuzzy-based CNN Models and CNN Integrated Systems

To make a CNN or a set of CNNs have the ability of reasoning functions, several fuzzy-based CNN models were proposed [5]-[9], which are fuzzy cellular neural network (FCNN) proposed by Yang et al [5], [6], fuzzy reasoning implemented on CNN proposed by Balsi et al [7], [8]. To make a set of CNNs in parallel achieve higher-level information processing, several integrated CNN systems are proposed [9]-[11], which are cellular neuro-fuzzy networks (CNFNs) proposed by Colodro [9], and fuzzy-type CNN proposed by Rekeczky [10], [11] and Szatmári et al. [4]. In the following, we will survey these related papers.

1.3.1.1. Existing Fuzzy-based CNN Models

Yang et al. [5], [6] first proposed a FCNN model in 1996. Such architecture had the same structure as a CNN with nonlinear connections, but the connection functions were stated in terms of fuzzy logic operators, so that the model departed from the trend towards standardization and simplification of connection functions to be realized in future CNN universal machine (CNN-UM) chips. The authors gave an example for edge detection. The characteristics of Yang's FCNN are to integrate fuzzy logic into the structure of traditional CNN and maintains local connection among cells, but its drawback is that it is too complex to implement in the short term.

Balsi et al. [7], [8] proposed a fuzzy reasoning method implemented on CNN-UM in 1999. Such architecture has the same structure as a conventional CNN. The authors showed that standard fuzzy logic could be straightforwardly implemented in the CNN-UM framework without any architectural modifications. The authors showed several examples for edge detection and noise removal. One of them concerned the edge detection in the presence of impulse noise. Experimental result showed the edge could be detected even impulse noise existed with appropriate fuzzy rules. The characteristic of Balsi's FCNN is to map a standard Sugeno-style fuzzy-rule-based image processing algorithm into a standard CNN-UM analogic (analog and logic) algorithm. However, the fuzzy rules must be obtained by domain experts.

Yang et al. [5], [6] and Balsi et al. [7], [8] were devoted to make a CNN or a set of CNNs have the ability of fuzzy reasoning. However, the other authors [4], [9]-[11] were devoted to make a set of CNNs in parallel achieve higher-level information processing, which is the main research subject in this thesis. The related papers are described in the following subsection.

1.3.1.2. Existing CNN Integrated Systems

In 1996, Colodro et al. [9] proposed a new class of cellular networks called cellular neuro-fuzzy networks (CNFNs), which the linear combination and piece-wise linear function of a CNN were replaced by an arithmetic fuzzy-logic unit. To demonstrate the capabilities of the proposed CNFN, the authors gave an application example for edge detection. The example used eight fuzzy rules and its CNFN templates were well-known Sobel masks to detect edge. The characteristic of Colodro's CNFN is to provide a new architecture based on CNN and FIS to solve problem. Its drawbacks are the templates cannot be learned and the fuzzy rules must be obtained by domain experts. Though Colodro et al. showed a simple example for edge detection, they presented an approach to integrate different CNN template sets to solve problem.

Rekeczky et al. [10], [11] developed a common CNN framework for various adaptive non-linear filters [10]. Their experimental results indicated that impulsive noise elimination will be more robust if both the pixel intensity and the edge-like local property is taken into consideration and exploited in a fuzzy-type decision. Rekeczky et al. [11] also proposed a CNN-based spatio-temporal approach to find the endocardial (inner) boundary of the left ventricle from a sequence of echocardiographic images. The kernel of the left ventricle was located and the boundary was found using a fuzzy-adaptive technique. Boundary dislocation, area and smoothness constraints were transformed into the transient length of the CNN while the a priori knowledge about the heart morphology was built into the spatial template parameters. The authors showed the architecture of the processing steps of a fuzzy-type CNN analogic algorithm. As observed by Rekeczky et al. [11], it was not necessary to use a specialized CNN model [5], [6] in order to exploit fuzzy logic concepts. The reasons were as follows. First, elementary fuzzy-type computations, such as the min and max operator, are already defined in CNN-based gray-scale morphology and require only simple non-linear templates with sigmoid-type nonlinear interactions. Second, higher level fuzzy strategies can also be synthesized using 'conventional' linear and non-linear CNN templates. The characteristic of Rekeczky's fuzzy-type CNN analogic algorithm is to use FIS to integrate different CNNs to detect fuzzy boundary of a given object. Similarly, its drawbacks are the corresponding templates cannot be learned and must be assigned in advance, and the fuzzy rules must be obtained by domain experts.

In 2003, Szatmári et al. [4] proposed an image flow processing mechanism for visual exploration systems. The goal of this multi-channel topographic approach was to produce decision maps for salient feature localization and identification. According to a current biological study, mammalian visual systems process the world through a set of separate parallel channels. Each sub-channel can be regarded as a unique CNN. The output of these sub-channels is then combined to form the new channel responses. In the core of the algorithm crisp or fuzzy logic strategies define the global channel interaction and result in a unique binary image flow. Experimental results were shown for terrain exploration environment based on multiple feature extraction. The characteristic of Szatmári's method is to use FIS to integrate different CNNs to extract features of a given object. Similarly, its drawbacks are the corresponding templates cannot be learned and the fuzzy rules must be obtained by domain experts.

1.3.2. The Proposed CNN Integrated System

Two common characteristics are observed in the representative works of integrated CNN systems such as Colodro et al. [9] and Roska et al. [4], [11]. First, they all used many CNNs in parallel to solve a complex problem such as edge detection with impulse noise, the detection of fuzzy boundary, and features extraction, etc. Second, they all used FIS to make a final decision. For building a FIS, we have to specify the fuzzy sets, fuzzy operators and the knowledge base. However, the existing methods [4], [9], [11] all need to manually take the fuzzy rules by domain experts, which is difficult, even for domain experts, to examine all the input–output data from a complex system to find a number of proper fuzzy rules. In addition, they all need to assign the corresponding templates of CNNs in advance (i.e., templates cannot be learned with fuzzy rules simultaneously). Although according to Nossek's survey [12], the template coefficients of a CNN can be found by design [12], [13] or by learning [12], [14], these techniques cannot be applied to the design or learning of an integrated CNN system directly.

To cope with these drawbacks, we introduce a novel framework for automatically constructing a multiple-CNN integrated neural system in the form of a recurrent fuzzy neural network (FNN) called recurrent fuzzy CNN (RFCNN). Figure 1.3 shows the structure of the RFCNN and its functional correspondence with a typical fuzzy inference system (FIS) [15]-[17], where the fuzzifier is to transform crisp measured data into suitable linguistic values, which are then matched with the fuzzy rules in the rule base by performing fuzzy approximate reasoning to achieve desired linguistic outputs, which are then transformed back to final crisp outputs through the defuzzifier. The RFCNN shown in Fig. 1.3 can automatically learn its proper network structure and parameters simultaneously. The structure learning includes the fuzzy division of the problem domain and the creation of fuzzy rules and CNNs. The parameter learning includes the tuning of fuzzy membership functions and CNN templates. In the RFCNN, each learned fuzzy rule corresponds to a CNN. Hence, each CNN takes care of a fuzzily separated problem region, and the functions of all

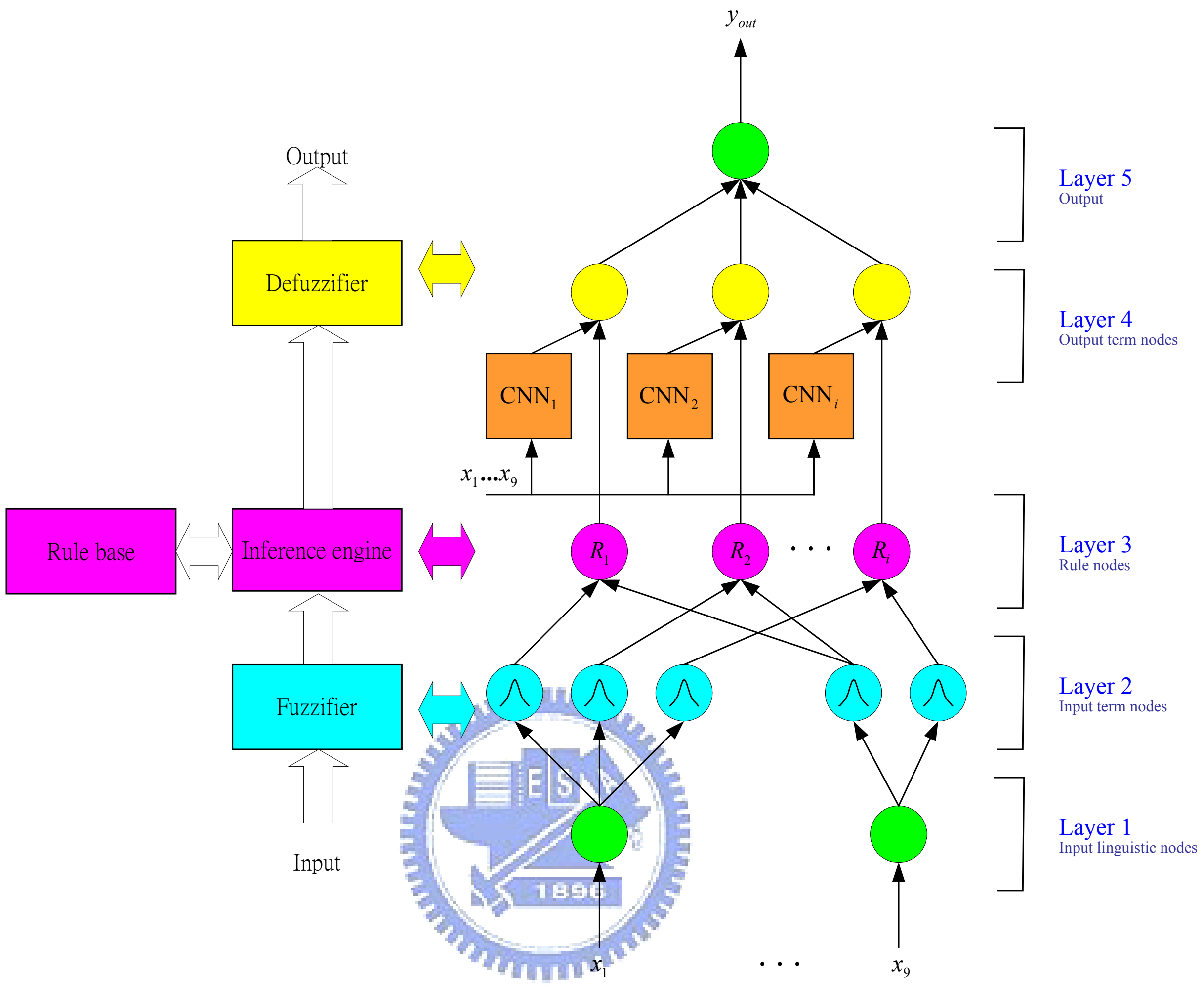


Figure 1.3 The schematic diagram of the RFCNN and FIS.

CNNs are integrated through the fuzzy inference mechanism.

The RFCNN is constructed in the form of a recurrent FNN. Two important learning tasks of a FNN are the structure identification and the parameters identification [15]-[19]. The structure identification is the partition of the input-output space [20]-[23], which influences the number of generated fuzzy rules, each corresponding to a CNN. Efficient partition of input-output data will result in faster convergence and better performance for FNN. In the parameter learning of the

RFCNN, the ordered-derivative calculus is applied to derive the recurrent learning rules due to the recurrent structure of the RFCNN inherited from CNNs. The derived rules can learn the CNN templates and other parameters in the RFCNN efficiently. The proposed RFCNN provides a solution to the current dilemma on the decision of templates and/or fuzzy rules in the existing integrated (fuzzy) CNN systems. It has been applied to solve the real-world defect inspection. This is an application for the defect inspection of color filter, which contains multiple types of defects (faults) with different features on a single image. Experimental results, shown in Sections 4 of Chapter 2 and Chapter 3, successfully demonstrate that the introduced scheme is very effective and promising.

1.4. Concluding Remarks

In this chapter, several fuzzy-based CNN integrated system are presented. Since those methods suffered from two problems, i.e., they all need to assign the corresponding templates of CNNs in advance (i.e., templates cannot be learned) and they all need to take the fuzzy rules manually by domain experts. To cope with these drawbacks, we proposed a novel framework for automatically constructing a multiple-CNN integrated neural system in the form of a recurrent fuzzy neural network (FNN). This system, called recurrent fuzzy CNN (RFCNN) [13], can automatically learn its proper network structure and parameters simultaneously. Each CNN takes care of a fuzzily separated problem region, and the functions of all CNNs are integrated through the fuzzy inference mechanism. For learning algorithm, the details of structure-learning algorithm based on adaptive ICA (independent component analysis) mixture-model technique are described in Section 3.1 of Chapter 2; the details of parameter-learning algorithm are described in Sections 3.2 of Chapter

2 and Chapter 3, respectively. For experimental results and discussions, they are described in Sections 4 of Chapter 2 and Chapter 3, respectively. Finally, conclusions and perspectives are described in the last chapter.

2. A Recurrent Fuzzy Cellular Neural Network System with Automatic Structure and Template Learning

In this chapter, we propose a novel framework for automatically constructing a multiple-CNN integrated neural system in the form of a recurrent fuzzy neural network. This system, called recurrent fuzzy CNN (RFCNN), can automatically learn its proper network structure and parameters simultaneously. The structure learning includes the fuzzy division of the problem domain and the creation of fuzzy rules and CNNs. The parameter learning includes the tuning of fuzzy membership functions and CNN templates. In the RFCNN, each learned fuzzy rule corresponds to a CNN. Hence, each CNN takes care of a fuzzily separated problem region, and the functions of all CNNs are integrated through the fuzzy inference mechanism. The new on-line adaptive ICA (independent component analysis) mixture-model technique, proposed in Section 3.1 of this chapter, is used for the structure learning of RFCNN, and the ordered-derivative calculus is applied to derive the recurrent learning rules of CNN templates in the parameter-learning phase. The proposed RFCNN provides a solution to the current dilemma on the decision of templates and/or fuzzy rules in the existing integrated (fuzzy) CNN systems. The capability of the proposed RFCNN is demonstrated on the real-world defect inspection problems. Experimental results show that the proposed scheme is effective and promising.

2.1. Introduction

The CNN has been used to mimic the local function of biological neural circuits, especially the human visual pathway system [3]. According to a current biological study [4], mammalian visual systems process the world through a set of separate parallel channels. Each sub-channel can be regarded as a unique CNN. The output of these sub-channels is then combined to form the new channel responses. As a result, it is widely accepted that using a set of CNNs in parallel can achieve higher-level information processing and reasoning functions either from biologics or application points of views. Such an integrated CNN system can solve more complex intelligent problems.

For designing an integrated CNN system, in addition to the determination of a set of templates, another kernel problem is the way of integration. To solve this problem, the fuzzy inference system (FIS) can play an important role to integrate a set of CNNs into a system. As mentioned in Section 3 of Chapter 1, to make a set of CNNs in parallel achieve higher-level information processing, several integrated CNN systems are proposed [4]-[7]. They have two common characteristics. First, they all used many CNNs in parallel to solve a complex problem. Second, they all used FIS to make a decision. The common drawbacks of these approaches are that they all need to assign the corresponding templates of CNNs in advance (i.e., templates cannot be learned) and they all need to take the fuzzy rules manually by domain experts. Although according to Nossek's survey [8], the template coefficients of a CNN can be found by design [8], [9] or by learning [8], [10], these techniques cannot be applied to the design or learning of an integrated CNN system directly.

To cope with these drawbacks, we proposed a novel framework for automatically constructing a multiple-CNN integrated neural system in the form of a recurrent fuzzy neural network (FNN) [11], [12]. This system, called recurrent fuzzy CNN (RFCNN) [13], can automatically learn its proper network structure and parameters simultaneously. The structure learning includes the fuzzy division of the problem domain and the creation of fuzzy rules and CNNs. The parameter learning includes the tuning of fuzzy membership functions and CNN templates. In the RFCNN, each learned fuzzy rule corresponds to a CNN. Hence, each CNN takes care of a fuzzily separated problem region, and the functions of all CNNs are integrated through the fuzzy inference mechanism.

As mentioned in Section 3 of Chapter 1, the RFCNN is constructed in the form of a recurrent FNN, which includes two important learning tasks: the structure identification and the parameters identification [15]-[19]. In this chapter, the new on-line adaptive ICA (independent component analysis) mixture-model technique, described in Section 3.1, is used for the structure learning of the RFCNN. Basically, ICA finds directions in the input space which lead to independent components instead of just uncorrelated ones, as PCA (principle component analysis) does [24], [25], so it reduces not only the number of rules (i.e., CNN) but also the number of membership functions under a pre-specified accuracy requirement dynamically. In the parameter learning of the RFCNN, the ordered-derivative calculus is applied to derive the recurrent learning rules due to the recurrent structure of the RFCNN inherited from CNNs [1], [2]. The derived rules can learn the CNN templates and other parameters in the RFCNN efficiently. The proposed RFCNN provides a solution to the current dilemma on the decision of templates and/or fuzzy rules in the existing integrated (fuzzy) CNN systems. It has been applied to solve the real-world defect inspection problems, which contain multiple types of defects (faults) with different features on a single image. Experimental results successfully demonstrate that the proposed scheme is very effective and promising.

This chapter is organized as follows. Section 2 describes the structure and functions of the proposed RFCNN. Section 3 describes the on-line structure and parameters learning algorithm for the RFCNN. Section 4 gives experimental results and discussions. Finally, conclusions are summarized in the last section.

2.2. Structure of the RFCNN

In this section, the structure of the proposed RFCNN shown in Fig. 2.1 is introduced. For clarity, we consider a CNN, with time constant = 1, time step = 1, and neighborhood within a radius = 1, which is characterized by the following templates:

$$A^i = \begin{bmatrix} 0 & 0 & 0 \\ 0 & a^i{}_{0,0} & 0 \\ 0 & 0 & 0 \end{bmatrix},\ B^i = \begin{bmatrix} b^i{}_{-1,-1} & b^i{}_{-1,0} & b^i{}_{-1,1} \\ b^i{}_{0,-1} & b^i{}_{0,0} & b^i{}_{0,1} \\ b^i{}_{1,-1} & b^i{}_{1,0} & b^i{}_{1,1} \end{bmatrix},\ I^i = z^i, \tag{2.1}$$

where A^i, B^i, and z^i is feedback template, control template, and bias of the *i*th CNN, respectively. By defining a CNN as above, the six-layered RFCNN network will realize a fuzzy model of the following form:

Rule i: IF x_1 is M_1^i and $\ldots x_j$ is $M_j^i \ldots$ and x_9 is M_9^i

$$\text{THEN } y_i(t+1) \text{ is } f^{'}(A^i y_i(t) + B^i u(t) + z^i(t)) \tag{2.2}$$

or

Rule i: IF x_1 is M_1^i and $\ldots x_j$ is $M_j^i \ldots$ and x_9 is M_9^i

$$\text{THEN } y_i(t+1) \text{ is } f^{'}(a^i_{0,0} y_i(t) + b^i{}_{-1,-1} x_1(t) + b^i{}_{-1,-0} x_2(t) + \ldots + b^i{}_{1,1} x_9(t) + z^i(t)), \tag{2.3}$$

where the current input vector is $u = \mathbf{x}_t = [x_1, \ \ldots, \ x_9]^T$, $A^i y_i(t)$ is $a^i_{0,0} y_i(t)$, $B^i u(t)$ is $\sum b^i u = b^i{}_{-1,-1} x_1(t) + b^i{}_{-1,-0} x_2(t) + \ldots + b^i{}_{1,1} x_9(t)$, $f^{'}$ is a sigmoid function, M_j^i is a fuzzy set, and $a^i_{0,0}$, $b^i{}_{k,l}$, and z^i are consequent parameters representing

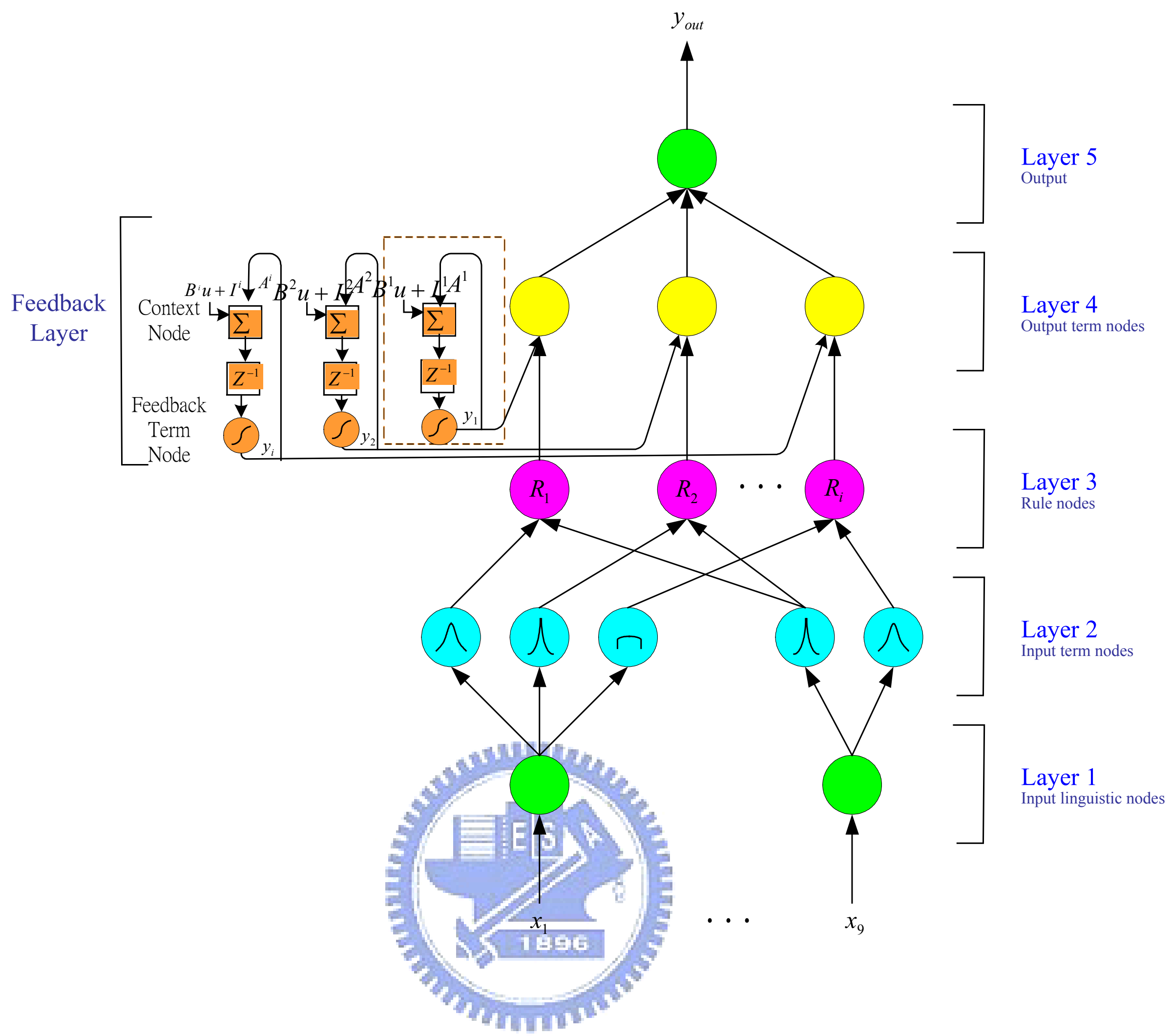


Figure 2.1 Structure of the proposed RFCNN.

feedback template, control template, and bias of the *i*th CNN, respectively. The number of input dimension of the RFCNN will be $(2r+1)^2$ if the neighborhood of a CNN cell is within a radius = *r*. As shown in (2.3), we focus on uncoupled CNN cells in this chapter. With this six-layered network structure of the RFCNN, we shall define the function of each node and use the proposed on-line ICA mixture model described in the next section to construct the structure of the RFCNN.

The RFCNN consists of nodes, each of which has some finite "fan-in" of connections represented by weight values from other nodes and "fan-out" of connections to other nodes. Associated with the fan-in of a node is an integration function f, which serves to combine information, activation, or evidence from other nodes. This function provides the net input for this node

$$node-input = f[u_1^{(k)}, u_2^{(k)}, \ldots, u_p^{(k)}; w_1^{(k)}, w_2^{(k)}, \ldots, w_p^{(k)}], \tag{2.4}$$

where $u_1^{(k)}$, $u_2^{(k)}$, ..., $u_p^{(k)}$ are inputs to this node and $w_1^{(k)}$, $w_2^{(k)}$, ..., $w_p^{(k)}$ are the associated link weights. The superscript (k) in (2.4) indicates the layer number. This notation will also be used in the following equations. A second action of each node is to output an activation value as a function of its net-input

$$node-output^{(k)} = o_i^{(k)} = a^{(k)}(node-input) = a^{(k)}(f), \tag{2.5}$$

where $a(\cdot)$ denotes the activation function. We shall next describe the functions of the nodes in each of the six layers of the RFCNN, which include five feedforward layers and one feedback layer.

Layer 1: No computation is done in this layer. Each node in this layer, which corresponds to one input variable, only transmits input values to the next layer directly. That is

$$f = u_i^{(1)},$$

and

$$o_i^{(1)} = a^{(1)}(f) = f. \tag{2.6}$$

From the above equation, the link weight in layer one $[w_i^{(1)}]$ is unity.

Layer 2: Each node in this layer corresponds to one linguistic value (small, large, etc.) of one of the input variables in Layer 1. In other words, the membership value which specifies the degree to which an input value belongs a fuzzy set is calculated in Layer 2. There are many choices for the types of membership functions for use, such as triangular, trapezoidal, or Gaussian ones. In this chapter, the membership functions are determined by the on-line ICA mixture model, which are either super-Gaussian function or sub-Gaussian function. It is noted that the output $\mathbf{x}$ from Layer 1 is projected into the independent axes obtained by the on-line ICA mixture model (as shown in Fig. 2.2) such that

$$\mathbf{s}_i = \mathbf{B}_i \mathbf{x}, \tag{2.7}$$

where $\mathbf{B}_i$ is the basis matrix determined by the on-line ICA mixture model, $i = 1,\ 2,\ ...,\ J$, and J is the number of clusters. That is, if the input data are classified into J clusters, the number of rules will be J.

With the choice of non-Gaussian membership function, the operation performed in this layer is

$$\mathbf{u}_i^{(2)} = \mathbf{s}_i,$$

$$f[u_{ij}^{(2)}] = p(u_{ij}^{(2)}),$$

where

$$p(u_{ij}) \propto \exp(-|u_{ij}|), \quad \text{for super-Gaussian}$$

$$p(u_{ij}) \propto \exp(-\log(\cosh(u_{ij})) - \frac{u_{ij}^2}{2}), \quad \text{for sub-Gaussian}$$

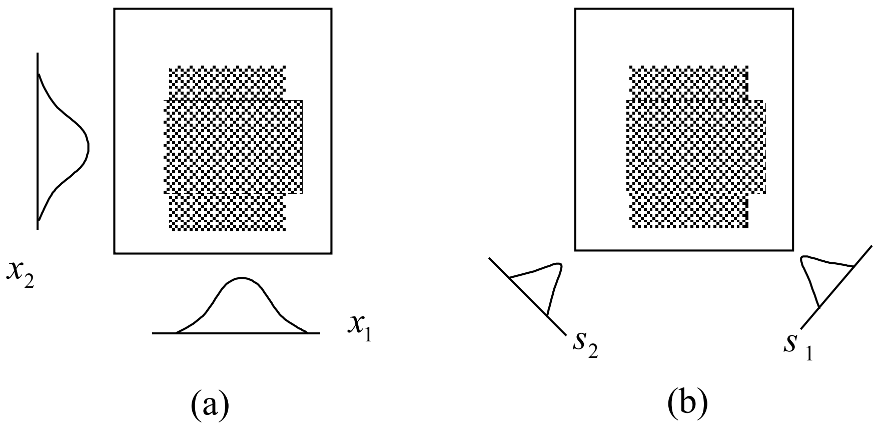


Figure 2.2 Transformation by the on-line ICA mixture model for the proposed RFCNN. (a) The regions covered by the original axes. (b) The covered regions by the independent axes obtained by the on-line ICA mixture model transformation.

and

$$o_i^{(2)} = a^{(2)}(f) = f, \tag{2.8}$$

where u_{ij} is the transformed value of the jth term of the ith input variable $\mathbf{x}_i$. The transformation can be regarded as a change of input coordinates, where the parameters of each membership function are kept unchanged, i.e., the center and the width of each membership function on the new coordinate axes are the same as the old ones.

Layer 3: A node in this layer represents one fuzzy logic rule and performs precondition matching of a rule. Here, we use the following AND operation for each Layer-3 node

$$f[u_i^{(3)}] = \prod_i u_i^{(3)},$$

and

$$o_i^{(3)} = a^{(3)}(f) = f. \tag{2.9}$$

The link weight in the Layer 3 $[w_i^{(3)}]$ is unity. The output f of a Layer-3 node represents the firing strength of the corresponding fuzzy rule.

Layer 4: This layer is called the consequent layer. Different nodes in Layer 3 may be connected to the same node in Layer 4, meaning that the same consequent fuzzy set is specified for different rules. One of the inputs to each node is the output delivered from Layer 3 (firing strength) and the other inputs are CNN related inputs $a^{(6)}$, which are the output of feedback term node. The feedback term node will be described in the feedback layer part in this section. Combining the two kinds of inputs in Layer 4, we obtain the whole function performed by this layer as

$$f[u_i^{(4)}] = u_i^{(4)},$$

and

$$o_i^{(4)} = a^{(4)}(f) = a^{(6)} \cdot f = sigmoid(A^i y_i(t) + B^i u(t) + z^i(t)) \cdot f \,, \tag{2.10}$$

where A^i, B^i, z^i is feedback template, control template, bias of the *i*th CNN, respectively, as defined in (2.1), and $sigmoid()$ is a sigmoid function, as defined in (2.13).

Layer 5: Each node in this layer corresponds to one output variable. The node integrates all the actions recommended by Layer 4 and acts as a defuzzifier with

$$f[u_i^{(5)}] = \sum_i u_i^{(5)} \,,$$

and

$$y_{out} = o^{(5)} = a^{(5)}(f) = f \quad . \tag{2.11}$$

Feedback Layer: As shown as Fig. 2.1, this self-feedback layer characterizes the consequents of the RFCNN as a CNN template. Two types of nodes are used in this layer, the square node named as *context node* and the circle node named as *feedback term node,* where each context node is associated with a feedback term node. The number of context nodes (and thus the number of feedback term nodes) is the same as that of output term nodes in layer 4. The inputs to a context node are from its corresponding output term nodes ($y_i(t)$), the input variables from Layer 1 ($u(t) = \mathbf{x}_t(t) = [x_1, \ ..., \ x_9]^T$), and template bias ($z^i(t)$). The output of its associated feedback term node is fed to the original node in layer 4. The context node functions as the state (the summation of input part) of the *i*th CNN

$$x^i(t+1) = A^i y_i(t) + B^i u(t) + z^i(t) \,. \tag{2.12}$$

As to the feedback term node, the membership function $f(u) = 2/(1+e^{-2u}) - 1$ is used to approximate piece-wise linear function used in CNN. With this choice, the feedback term node evaluates the output by

$$o_i^{(6)} = a^{(6)} = \frac{2}{1+e^{-2x^i}} - 1 \quad . \tag{2.13}$$

This output is connected to its corresponding node in layer 4, which characterizes the consequents of the RFCNN as a CNN template.

2.3. Learning Algorithm for the RFCNN

Two types of learning, structure and parameter learning, are used concurrently for the RFCNN. The structure learning includes both the precondition and consequent structure identification of a fuzzy if-then rule. In the RFCNN, the structure learning includes the fuzzy division of the problem domain (precondition structure identification), and the creation of fuzzy rules and CNNs (consequent structure identification). The precondition structure identification corresponds to the input-space partitioning and can be formulated as a combinational optimization problem with the following two objectives: to reduce the number of rules generated and to reduce the number of fuzzy sets on the universe of discourse of each input variable. As to the consequent structure identification, the main task is to decide when to generate a new consequent term (or a new CNN) for the output variable. In our system, we propose an on-line ICA mixture model to realize the precondition and consequent structure identification part of the RFCNN.

For the parameter learning, the parameters of each CNN template in the consequent parts are adjusted by the ordered derivative algorithm to minimize a given cost function. The parameters in the precondition part are adjusted by the on-line ICA mixture model algorithm. The RFCNN can be used for normal operation at any time during the learning process without repeated training on the input-output patterns when on-line operation is required. There are no rules (i.e., no nodes in the network except the input-output nodes) in this network initially. They are created dynamically

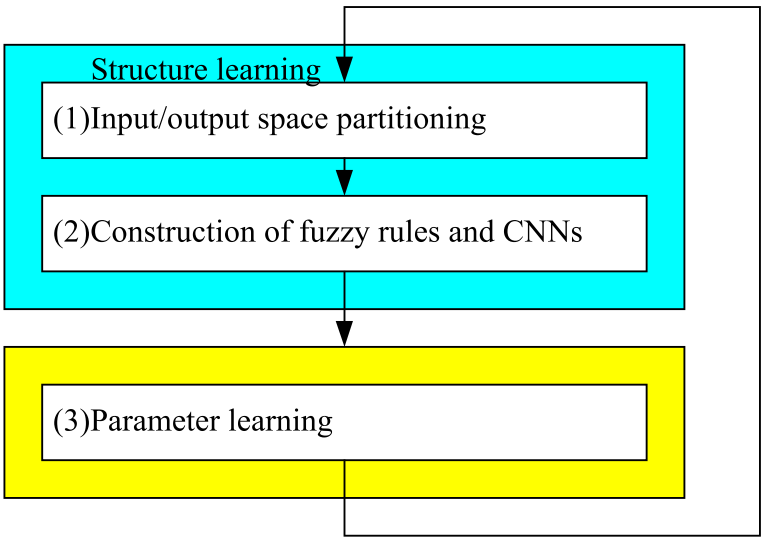

Structure learning
(1)Input/output space partitioning
(2)Construction of fuzzy rules and CNNs
(3)Parameter learning

Figure 2.3 Flowchart of the learning algorithm for the proposed FNN.

as learning proceeds upon receiving on-line incoming training data by performing the following learning processes simultaneously (see Fig. 2.3).

As shown in Fig. 2.3, learning processes (1) and (2) belong to the structure learning phase and (3) belongs to the parameter learning phase. The details of these learning processes are described in the rest of this section.

2.3.1. Structure Learning Algorithm of RFCNN

2.3.1.1 Input/Output Space Partitioning

Efficient partition of input-output data will result in faster convergence and better performance for the RFCNN. The most direct way is to partition the input space into grid types and each grid represents a fuzzy if-then rule (see Fig. 2.4(a)). This is called grid-based partitioning. The major problem of such kind of partition is that the number of fuzzy rules (and thus the number of CNNs) increases exponentially if the number of input variables or that of partition increases. A flexible partition method, the clustering-based approach which clusters the input training vectors in the input space, will reduce the rule and CNN numbers [20]-[23]. In fact, by observing the projected membership functions in Fig. 2.4(c), although the number of membership functions in Fig. 2.4(d) is more than that in Fig. 2.4(b), there are only 5 rules in Fig. 2.4(d); however, there are 9 rules in Fig. 2.4(b). By observing the projected membership functions in Fig. 2.4(c), we find that some membership functions projected from different clusters have high similarity degrees. These highly similar membership functions can be checked and merged by similarity measure [23]. In the rest of this subsection, we will introduce some clustering methods such as C-means, ISODATA, and a new on-line ICA mixture model to provide a proper partition of the input-output space for the RFCNN [25]. These clustering algorithms will be briefly described in the following subsections.

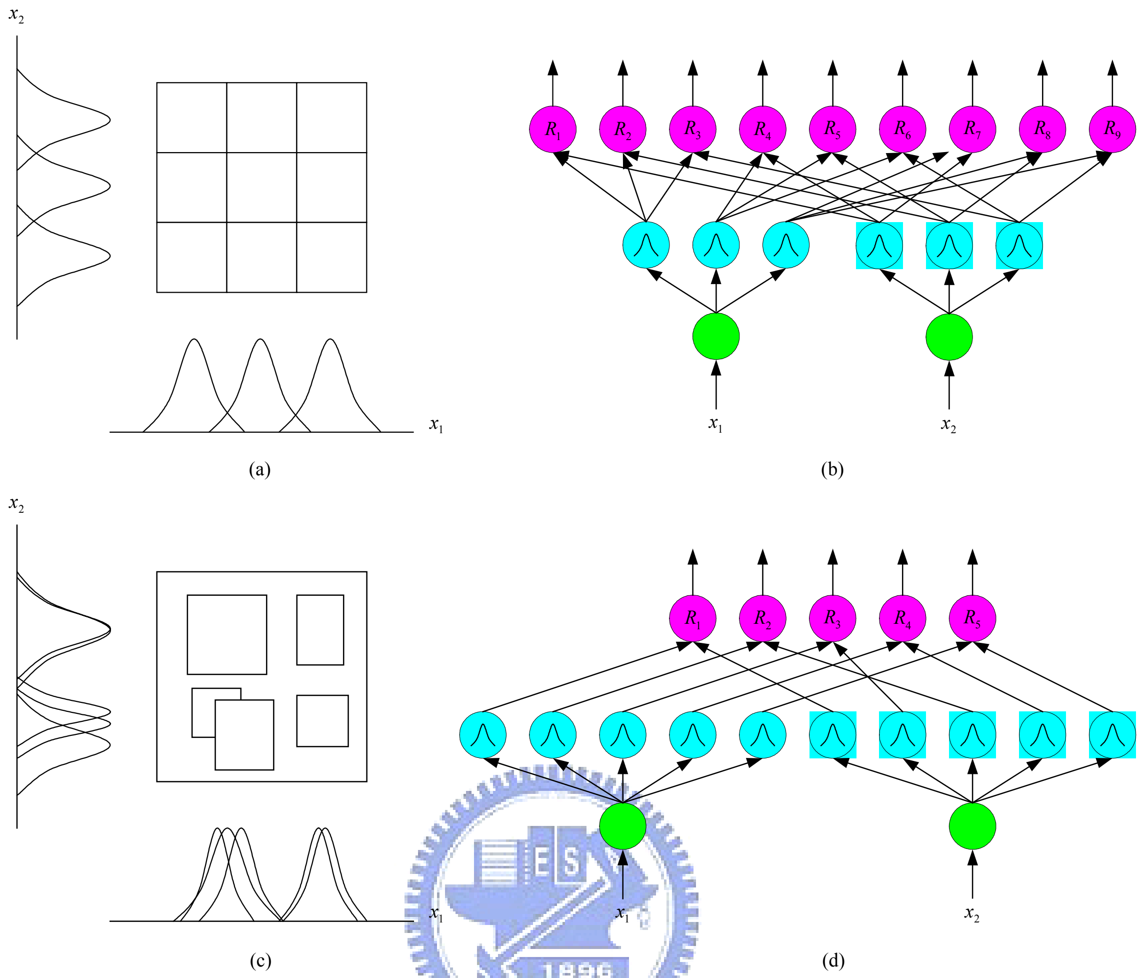

x_2
x_1
R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8 R_9
x_1 x_2
(a)
(b)
x_2
x_1
R_1 R_2 R_3 R_4 R_5
x_1 x_2
(c)
(d)

Figure 2.4 Fuzzy partitions of two-dimensional input space. (a) Grid-based partitioning. (b) If-then rules based on grid-based partitioning (c) Clustering-based partitioning. (d) If-then rules based on clustering-based partitioning.

2.3.1.1.1 C-means Clustering

In the C-means clustering algorithm, the sum of the squared distances from all of the points in the cluster to the cluster center is used as the criterion for grouping input data samples. The algorithm can be described as follows:

Step 1. Choose C initial cluster centers. For the first iteration, the starting values for cluster centers are chosen randomly.

Step 2. Assign unknown samples to corresponding clusters. For each sample, the Euclidean distance from each cluster center is calculated, and the sample is assigned to the cluster with the minimum distance.

Step 3. Compute new cluster centers, which are calculated by the mean value of the points in the same cluster.

Step 4. Check for the condition of convergence, which is that no cluster center has changed its value during Step 3.

The C-means algorithm is a simple and efficient scheme to find proper input/output partitioning for the RFCNN, and then determine the proper fuzzy rules (i.e., CNNs). However, it assumes that the number of clusters is known in advance, which is exactly what we have to know from the clustering algorithm to obtain the number of fuzzy rules (i.e., CNNs) in the RFCNN in some real-world applications. As a result, the C-means algorithm cannot be used to *on-line* determine the structure of the RFCNN if there is such a need.

2.3.1.1.2 ISODATA

In the ISODATA (Iterative Self-Organizing Data Analysis Technique) clustering algorithm [26], like the C-means algorithm, cluster centers are updated iteratively. The ISODATA algorithm is a good method to determine the structure of the RFCNN though it is more complex, because some heuristic procedures are incorporated into it. Three additional procedures that the algorithm performs are as follows: (1) if a cluster contains a small number of samples, that cluster is discarded; (2) if a cluster contains a large number of samples and the standard deviation is large, then the cluster is split into two; and (3) if the distance between two cluster centers is small, then the clusters are merged into one. Essentially, what is done in the ISODATA algorithm is that very small clusters are discarded, very large clusters are split, and very close clusters are merged. Again, like the C-means algorithm, the proper cluster number should be known in advance before applying the ISODATA algorithm. This also hinders the use of this clustering algorithm on the on-line structuring learning of RFCNN in some real-world problems.

2.3.1.1.3 On-line ICA Mixture Model for Dynamic Clustering

1) ICA Mixture Model

Several methods for input space partition have been proposed to cluster the input training vectors in the input space, such as Kohonen learning rule, hyperbox method, product-space partitioning, fuzzy c-mean method, EM algorithm, etc. [26]-[29]. Those methods are usually based on Gaussian membership functions. In general, the observed data can be categorized into several mutually exclusive classes [30]. When the data in each class are modeled as multivariate Gaussian, it is called a Gaussian mixture model (GMM) which is widely used throughout the fields of machine learning and statistics. One major drawback of GMMs is that if the dimension d of the problem space increases, the size of each covariance matrix, d^2, becomes prohibitively large. This problem has been solved by Tipping and Bishop [31] who replaced each Gaussian with a probabilistic principal component analysis (PCA) model. This allowed the dimensionality of each covariance to be effectively reduced while maintaining the richness of the model class. Independent component analysis (ICA) [24] is a technique that exploits higher-order statistical structure of the data, which has recently gained attention due to its successful applications to signal processing problems including speech enhancement, discrete signal processing and image processing, etc. The goal of ICA is to linearly transform the data such that the transformed variables are as statistically independent from each other as possible.

Basically, it finds direction in the input space which lead to independent components instead of just uncorrelated ones as PCA does, so it can be used to reduce not only the number of rules but also the number of membership functions under a pre-specified accuracy requirement dynamically.

Another drawback of GMMs is that it is based on Gaussian function. In some situation, it could not be separated from each other. It is generalized by assuming the data in each class are generated by a linear combination of independent non-Gaussian source [33]. This model is called the ICA mixture model. This allows modeling of classes with non-Gaussian structure such as platykurtic or leptokurtic probability density functions, and the model uses the gradient ascent method to maximize the log-likelihood function. In previous applications, this approach showed improved performance in data classification problems [34] and learning efficient codes for representing different types of images [25]. The advantage of this model is that the input data with increasing numbers of classes can provide greater flexibility in modeling structure and in finding more features compared with Gaussian mixture models or standard ICA algorithms. Although the ICA mixture model has many advantages, its cluster number should be given beforehand and the learning scheme is only suitable for off-line instead of on-line operation. Therefore, in the following section, we shall propose an on-line ICA mixture model to provide better dynamic partitioning of the input-output space for the proposed RFCNN.

2) On-line ICA Mixture Model for Dynamic Clustering

The proposed on-line ICA mixture model is derived from the conventional ICA mixture model. To enable the on-line operation, we will define a criterion to determine whether the number of clusters should be increased or not for any incoming training pattern. For each incoming pattern to the RFCNN, the resulting firing strength of a fuzzy rule can be interpreted as the degree that the incoming pattern belongs to the corresponding cluster. This likelihood can be represented as

$$F^{j}(\mathbf{x}_t) = \ln p(\mathbf{x}_t | C_j), \tag{2.14}$$

where $\mathbf{x}_t$ denotes the incoming pattern at time t, and $\ln p(\mathbf{x}_t | C_j)$ is the log likelihood value indicating the degree that the input data, $\mathbf{x}_t$, belongs to the *j*th cluster for $j \in [1, J]$.

Now, we assume that the number of clusters at time t is $J(t)$. Then, the total probability at time t is

$$p(\mathbf{x}_t) = \sum_{j=1}^{J(t)} p(\mathbf{x}_t | C_j) p_t(C_j). \tag{2.15}$$

Therefore, the posterior probability is:

$$p(C_j | \mathbf{x}_t) = \frac{p(\mathbf{x}_t | C_j) p_{t-1}(C_j)}{p(\mathbf{x}_t)}, \tag{2.16}$$

where $p_{t-1}(C_j)$ is the prior probability at preceding time, which can be obtained by former calculation result of the *j*th cluster. Hence, the probability $p_t(C_j)$ at this moment can be calculated by the following formula:

$$\begin{aligned} p_t(C_j) &= \frac{1}{t}\sum_{i=1}^{t} p(C_j | \mathbf{x}_i) = \frac{1}{t}\left[\sum_{i=1}^{t-1} p(C_j | \mathbf{x}_i) + p(C_j | \mathbf{x}_t)\right] \\ &= \frac{1}{t}\left[(t-1) \cdot p_{t-1}(C_j) + p(C_j | \mathbf{x}_t)\right] \end{aligned}. \tag{2.17}$$

Then the posterior probability $p(C_j | \mathbf{x}_t)$ in (2.16) can be obtained.

Based on the above derivation, we can obtain the following criterion for the generation of a new fuzzy rule (i.e., a new CNN). Let $\mathbf{x}_t$ be the newly incoming pattern at time t. Defining

$$F^{J_{\max}}(\mathbf{x}_t) = \max_{1 \le j \le J(t)} F^{j}(\mathbf{x}_t), \tag{2.18}$$

If $F^{J_{\max}}(\mathbf{x}_t) < F$, then a new rule is generated, where F is a pre-specified threshold value that decays during the learning process. Once a new rule is generated, the next step is to assign initial values of the corresponding membership functions. If $F^{J_{\max}}(\mathbf{x}_t) \geq F$, a new incoming data is added to an existed cluster and we have to update the parameters of each cluster such as mean ($\mathbf{M}_j$), covariance matrix ($\mathbf{Cov}_\mathrm{j}$), and the criterion of data distribution ($k_{j,p}$) that determines if the distribution of data is super-Gaussian or sub-Gaussian with the previous calculation results. They are defined as follows:

$$\mathbf{M}_j(t) \cong \frac{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^{t} p(C_j|x_i)} = \frac{\sum_{i=1}^{t-1} p(C_j|\mathbf{x}_i)\mathbf{x}_i + p(C_j|\mathbf{x}_t)\mathbf{x}_t}{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)} \tag{2.19}$$

$$\begin{aligned}
\mathbf{Cov}_j(t) &= \frac{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)(\mathbf{x}_i - \mathbf{M}_j(t))(\mathbf{x}_i - \mathbf{M}_j(t))^T}{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)} \\
&\cong \frac{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)\mathbf{x}_i\mathbf{x}_i}{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)} - \mathbf{M}_j(t)\mathbf{M}_j(t)^T \\
&= \frac{\sum_{i=1}^{t-1} p(C_j|\mathbf{x}_i)\mathbf{x}_i\mathbf{x}_i^T + p(C_j|\mathbf{x}_i)\mathbf{x}_t\mathbf{x}_t^T}{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)} - \mathbf{M}_j(t)\mathbf{M}_j(t)^T \qquad (2.20) \\
&= \frac{1}{tp_t(C_j)}\Big[(t-1)p_{t-1}(C_j)\mathbf{Cov}_j(t-1) + (t-1)p_{t-1}(C_j) \\
&\quad \mathbf{M}_j(t-1)\mathbf{M}_j(t-1)^T + p(C_j|\mathbf{x}_t)\mathbf{x}_t\mathbf{x}_t^T\Big] - \mathbf{M}_j(t)\mathbf{M}_j(t)^T
\end{aligned}$$

$$k_{j,p}(t) = sign\left\{ \frac{\left[\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)\operatorname{sech}^2(y_{j,p}(i))\right]\left[\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)(y_{j,p}(i))^2\right]}{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)} - \frac{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i) y_{j,p}(i)\cdot\tanh^2(y_{j,p}(i))}{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)} \right\} \tag{2.21}$$

In the above, the function $k_{j,p}(t)$ is defined as the function of criterion which allows for automatic switching between super-Gaussian and sub-Gaussian models and (2.21) can be further derived as

$$k_{j,p}(t) = sign\left\{ \frac{T_1(t)T_2(t)}{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)} - \frac{T_3(t)}{\sum_{i=1}^{t} p(C_j|\mathbf{x}_i)} \right\} \tag{2.22}$$

where

$$\begin{aligned} T_1(t) &= \sum_{i=1}^{t} p(C_j|\mathbf{x}_i)\operatorname{sech}^2(y_{j,p}(i)) \\ &= T_1(t-1) + p(C_j|\mathbf{x}_t)\operatorname{sech}^2(y_{j,p}(t)), \\ T_2(t) &= \sum_{i=1}^{t} p(C_j|\mathbf{x}_i)(y_{j,p}(i))^2 \\ &= T_2(t-1) + p(C_j|\mathbf{x}_t) y_{j,p}(t), \\ T_3(t) &= \sum_{i=1}^{t} p(C_j|\mathbf{x}_i) y_{j,p}(i)\cdot\tanh^2(y_{j,p}(i)) \\ &= T_3(t-1) + p(C_j|\mathbf{x}_t) y_{j,p}(t)\tanh^2(y_{j,p}(t)). \end{aligned} \tag{2.23}$$

Finally, the independent axes $\mathbf{B}_j(t)$, representing the axis of the jth cluster, can be obtained by the following formulations:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{B}_j}\ln p(\mathbf{x}_t) &= p(C_j|\mathbf{x}_t)\cdot\frac{\partial}{\partial \mathbf{B}_j}\ln p(\mathbf{x}_t|\mathbf{B}_j(t-1),\ \mathrm{M}_j(t-1)) \\ &= p(C_j|\mathbf{x}_t)\cdot\{[\mathbf{I} - k_{j,p}(t)\cdot g(\mathbf{y}_j(t))\cdot \mathbf{y}_j^T(t)]\cdot \mathbf{B}_j(t-1)\}. \end{aligned} \tag{2.24}$$

and

$$\mathbf{B}_j(t) = \mathbf{B}_j(t-1) + \frac{\partial}{\partial \mathbf{B}_j} \ln p(\mathbf{x}_t). \tag{2.25}$$

In (2.24), the function $g(x)$ is called component-wise nonlinearity function. If the distribution of data is appearing the super-Gaussian distribution, then it will be defined as $g(x) = -2\tanh(x)$. Otherwise, if the distribution of data is appearing the sub-Gaussian distribution, then it will be defined as $g(x) = \tanh(x) - x$.

Since the algorithm of on-line ICA mixture model can automatically determine the number of clusters according to new incoming data, it solves the problem of conventional ICA mixture model that the number of clusters has to be given beforehand.

2.3.1.2. Structure Learning Algorithm of RFCNN with On-line ICA Mixture Model

The way the input space is partitioned determines the number of rules extracted from training data as well as the number of fuzzy sets on the universal of discourse of each input variable. We will define a criterion to determine whether a new cluster (i.e., a new fuzzy rule or a new CNN) should be added or not. Let $\mathbf{x}_t$ of cluster j be the newly incoming pattern at time t. Defining

$$F^{J_{\max}}(\mathbf{x}_t) = \max_{1 \le j \le J(t)} F^j(\mathbf{x}_t), \tag{2.26}$$

where $F^j(\mathbf{x}_t) = \ln p(\mathbf{x}_t | C_j)$ is the log likelihood value indicating the degree that input data, $\mathbf{x}_t$, belongs to the *j*th cluster, and the superscript $J_{\max}$ is a maximum log likelihood value among all log likelihood values. If $F^{J_{\max}}(\mathbf{x}_t) \ge \bar{F}$, the number of cluster is not increased, where $\bar{F}$ is a pre-specified threshold value that decays during the learning process. In this case, the new incoming pattern is added to an existed cluster and the parameters of this cluster will be updated properly. Oppositely, if $F^{J_{\max}}(\mathbf{x}_t) < \bar{F}$, the number of cluster will be increased. The threshold value $\bar{F}$ is determined by experiments.

The whole algorithm for the generation of new fuzzy rules as well as fuzzy sets in each input variable is shown in Fig. 2.5 step by step. In PART 2 of Fig. 2.5, the threshold F_{in} determines how many rules will be generated, where F_{in} should be negative since it is taken in natural log. For a lower value of F_{in}, more rules will be generated. Similarly, F_{out} determines how many output clusters will be generated and a lower value of F_{out} will result in more output clusters. For the output space partitioning, the same approach in (2.14) is used. The generation of a new output cluster corresponds to the generation of a new CNN. Suppose a new input cluster is formed after the presentation of the current input-output training pair $(\mathbf{x}, \mathbf{d})$; then the consequent part is constructed by the algorithms shown in Fig. 2.6.

The above algorithm is based on the fact that different precondition of different rules may be mapped to the same consequent term, i.e., CNN. Since only the center of each output membership function is used for defuzzification, the consequent part of each rule may simply be regarded as a singleton. Compared to the general fuzzy rule-based models with singleton output where each rule has its own individual singleton value, fewer parameters are needed in the consequent part of the RFCNN, especially for the case with a large number of rules.

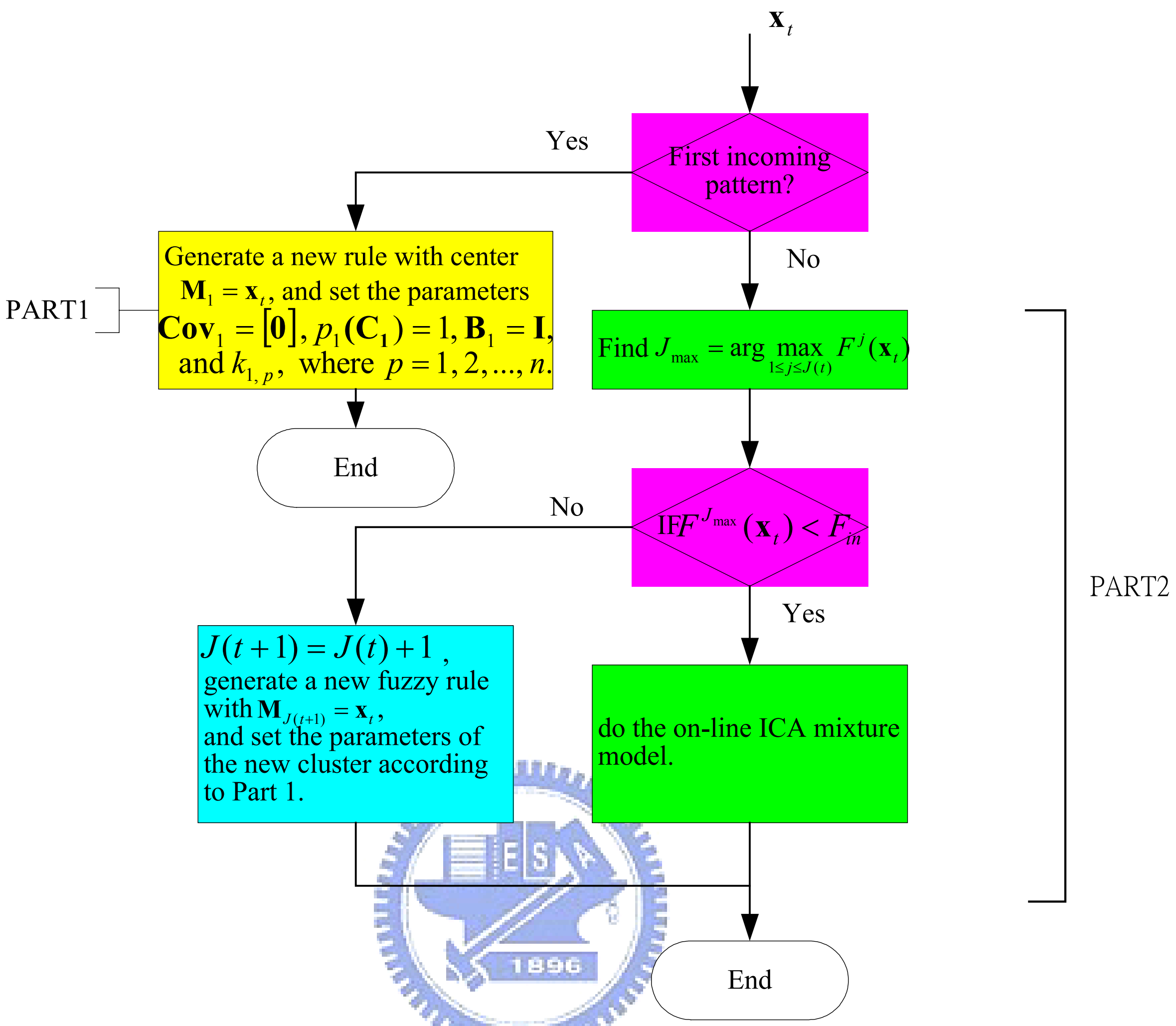
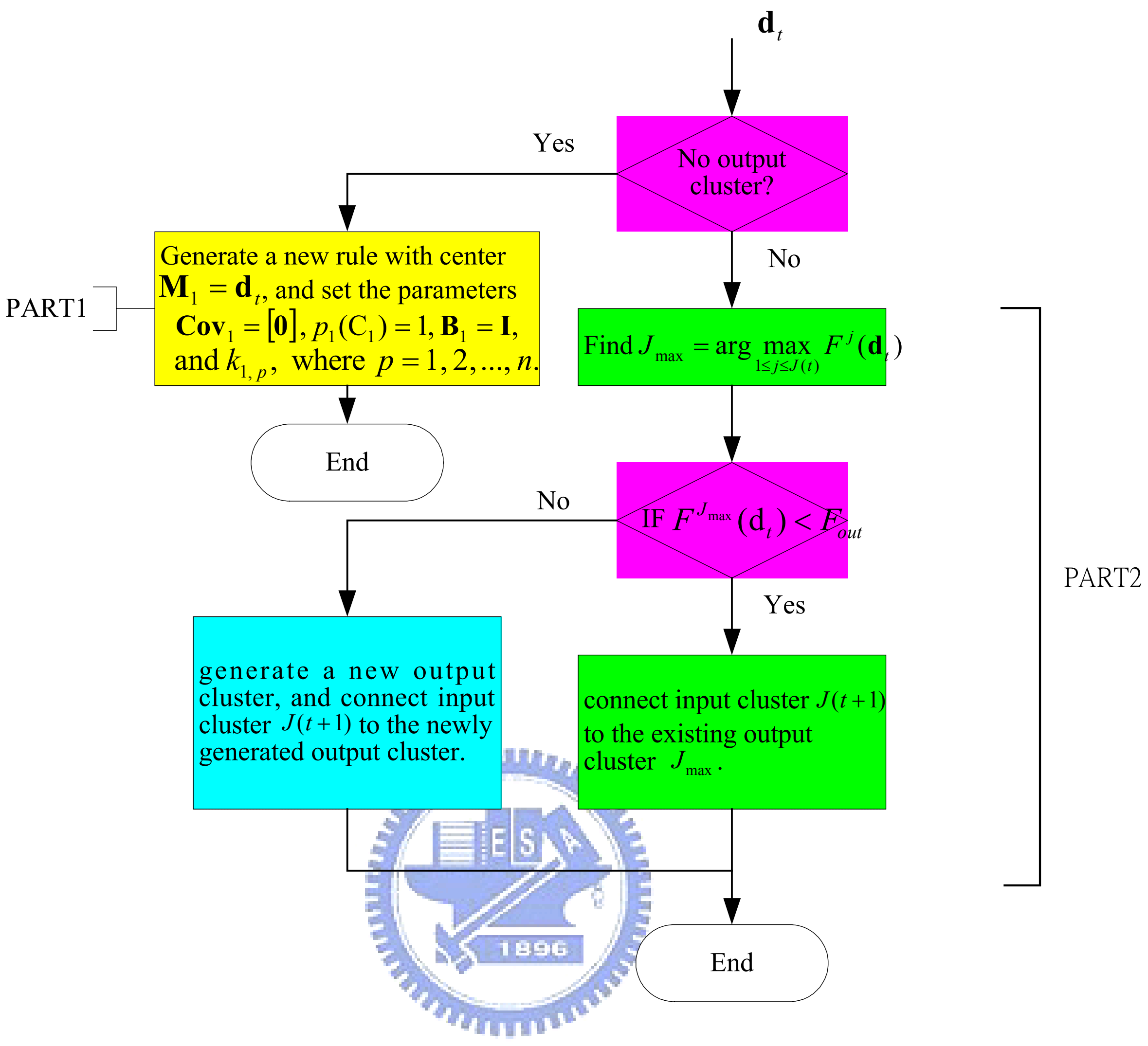

$\mathbf{x}_t$
First incoming pattern?
Yes
No
Generate a new rule with center $\mathbf{M}_1 = \mathbf{x}_t$, and set the parameters $\mathbf{Cov}_1 = [\mathbf{0}], p_1(\mathbf{C}_1) = 1, \mathbf{B}_1 = \mathbf{I}$, and $k_{1,p}$, where $p = 1, 2, ..., n$.
PART1
End
Find $J_{\max} = \arg\max_{1 \le j \le J(t)} F^j(\mathbf{x}_t)$
IF $F^{J_{\max}}(\mathbf{x}_t) < F_{in}$
No
Yes
$J(t+1) = J(t) + 1$, generate a new fuzzy rule with $\mathbf{M}_{J(t+1)} = \mathbf{x}_t$, and set the parameters of the new cluster according to Part 1.
do the on-line ICA mixture model.
PART2
End

Figure 2.5 Algorithm of input space partitioning.



$\mathbf{d}_t$
No output cluster?
Yes
No
Generate a new rule with center $\mathbf{M}_1 = \mathbf{d}_t$, and set the parameters $\mathbf{Cov}_1 = [\mathbf{0}], p_1(C_1) = 1, \mathbf{B}_1 = \mathbf{I}$, and $k_{1,p}$, where $p = 1, 2, ..., n$.
PART1
Find $J_{\max} = \arg \max_{1 \le j \le J(t)} F^j(\mathbf{d}_t)$
End
IF $F^{J_{\max}}(\mathrm{d}_t) < F_{out}$
No
Yes
PART2
generate a new output cluster, and connect input cluster $J(t+1)$ to the newly generated output cluster.
connect input cluster $J(t+1)$ to the existing output cluster $J_{\max}$.
End

Figure 2.6 Algorithm of output space partitioning.

2.3.2. Parameter Learning Algorithm of RFCNN by Ordered Derivative Calculus

After the network structure is adjusted according to the current training pattern, the network then enters the parameter identification phase to adjust the parameters of the network optimally based on the same training pattern. Notice that the following parameter learning is performed on the whole network after structure learning; no matter whether the nodes (links) are newly added or are existent originally. Since the

RFCNN is a dynamic system with feedback connections, the backpropagation learning algorithm cannot be applied to it directly. Also, due to the on-line learning property of the RFCNN, the off-line learning algorithms for the recurrent neural networks, like backpropagation through time and time-dependent recurrent backpropagation [17], cannot be applied here. Instead, the ordered derivative [34], which is a partial derivative whose constant and varying terms are defined using an ordered set of equations, is used to derive our learning algorithm. The ordered set of equations, described in Section 2 in each layer, is summarized in (2.28)-(2.33). Our goal is to minimize the error function

$$E(t+1)=\frac{1}{2}[y_{out}(t+1)-y_{out}^{d}(t+1)]^2=\frac{1}{2}\varepsilon(t+1)^2, \tag{2.27}$$

where $y_{out}^{d}(t+1)$ is the desired output, $y_{out}(t+1)$ is the current output, and $\varepsilon(t+1)$ is $(y_{out}(t+1)-y_{out}^{d}(t+1))$. For each training data set, starting at the input nodes, a forward pass is used to compute the activity levels of all the nodes in the network to obtain the current output $y_{out}(t+1)$. In the followings, dependency on time will be omitted unless emphasis on temporal relationships is required.

Summarizing the node functions defined in Section 2, the function performed by the network is

$$y_{out}(t+1)=\sum_i u_i^{(5)} \tag{2.28}$$

$$u_i^{(5)}=o^{(4)}=o^{(6)}\cdot h^i, \tag{2.29}$$

where

$$h^i=f[u_i^{(3)}]=\prod_i u_i^{(3)} \tag{2.30}$$

$$o^{(6)}=\frac{2}{1+e^{-2x^i}}-1 \tag{2.31}$$

$$x^i(t+1)=A^i y_i(t)+B^i u(t)+z^i(t), \tag{2.32}$$

and (2.1) is redefined as the following equation for clarity:

$$A^i = [0,0,0;0,a^i,0;0,0,0],\ B_j^i = [b_1^i, b_2^i, b_3^i; b_4^i, b_5^i, b_6^i; b_7^i, b_8^i, b_9^i]. \tag{2.33}$$

With the above formula and the error function defined in (2.27), we can derive the update rules for the free parameters in the RFCNN as follows.

Update rule of a^i (the parameter of feedback template of the *i*th CNN) is

$$a^i(t+1) = a^i(t) - \eta \frac{\partial^+ E(t+1)}{\partial a^i} \tag{2.34}$$

$$\begin{aligned} \frac{\partial^+ E(t+1)}{\partial a^i} &= \frac{\partial E\ (t+1)}{\partial a^i} + \sum_k \frac{\partial E\ (t+1)}{\partial y_{out,k}(t+1)} \frac{\partial^+ y_{out,k}(t+1)}{\partial a^i} \\ &= \frac{\partial E\ (t+1)}{\partial y_{out}(t+1)} \frac{\partial^+ y_{out}(t+1)}{\partial a^i} \end{aligned} \tag{2.35}$$

where

$$\frac{\partial E\ (t+1)}{\partial y_{out}(t+1)} = \varepsilon(t+1), \tag{2.36}$$

and

$$\frac{\partial^+ y_{out}(t+1)}{\partial a^i} = \sum_k \frac{\partial y_{out}\ (t+1)}{\partial o_k^{(4)}} \frac{\partial^+ o_k^{(4)}}{\partial a^i} = \frac{\partial y_{out}\ (t+1)}{\partial o_i^{(4)}} \frac{\partial^+ o_i^{(4)}}{\partial a^i}. \tag{2.37}$$

where

$$\frac{\partial y_{out}(t+1)}{\partial o_i^{(4)}} = \frac{\partial}{\partial o_i^{(4)}} \sum_k o_k^{(4)}(t+1) = 1, \tag{2.38}$$

and

$$\begin{aligned} \frac{\partial^+ o_i^{(4)}}{\partial a^i} &= \frac{\partial}{\partial a^i}[o_i^{(6)}(A_j^i y_i(t) + B_j^i u(t) + z^i(t))]h^i \\ &= h^i(1+o_i^{(6)})(1-o_i^{(6)})[y_i(t) + a^i \frac{\partial y_i(t)}{\partial a^i}] \end{aligned} . \tag{2.39}$$

Hence, the parameter a^i is updated by

$$a^i(t+1) = a^i(t) - \eta\varepsilon(t+1)\{h^i(1+o_i^{(6)})(1-o_i^{(6)})[y_i(t) + a^i \frac{\partial y_i(t)}{\partial a^i}]\}. \tag{2.40}$$

Similarly, the parameter b_j^i (the parameters of control template of the *i*th CNN) is updated by

$$b_j^i(t+1) = b_j^i(t) - \eta\varepsilon(t+1)[h^i(1+o_i^{(6)})(1-o_i^{(6)})x_j(t)] \tag{2.41}$$

and the parameter z_i (the bias of the *i*th CNN) is updated by

$$z^i(t+1) = z^i(t) - \eta\varepsilon(t+1)[h^i(1+o_i^{(6)})(1-o_i^{(6)})]. \tag{2.42}$$

As shown in (2.37) to (2.39), the update rules are in recursive form. The value $\partial^+ y/\partial a$ is equal to zero initially. For the rest free parameters in the RFCNN, they are obtained during the structure-learning phase by the on-line ICA mixture model algorithm proposed in the last section. Notice that according to the real time recurrent learning (RTRL) scheme [35], we can also obtain the same parameter learning rules for the RFCNN. Of course, other existing on-line learning algorithms [36], [37] for tuning the weights of recurrent neural networks can be possibly adopted for tuning the RFCNN, too.

2.4. Experimental Results and Discussions

The capability of the proposed RFCNN is demonstrated on the real-world defect inspection problems. Automatic defect inspection systems are becoming more and more important in industrial production lines. Especially in electronics industry, an attempt is often made to achieve almost 100% quality control of all components and final goods. Here we are interested in the defect inspection of color filter, which is one of components in TFT-LCD module and gives each pixel of LCD its own color. The difficulties in the defect inspection of color filter are its complex texture and need for high-speed processing. For the reason of high-speed processing, the CNN is a good way to achieve defect inspection. Besides, different kinds of defects in color filter need different CNN templates and some complex defects cannot be detected by a single CNN. Therefore, the proposed RFCNN is a good alternative to detect defect of color filter images. To train the RFCNN, as shown in Fig. 2.7, we use a 3x3 window to get the system inputs and set the whole image as the inputs of the RFCNN. The 3x3 window covers the central pixel and its 8 connected neighbors. The training image and corresponding desired output are shown in Figs. 2.8(a) and 2.8(b). We set the threshold $F_{in} = F_{out} = -50$ and learning rate as $\eta = 0.001$ for the clustering algorithm. As mentioned in Section 3, there are no rules (and no CNNs) in the RFCNN initially. They are created dynamically as learning proceeds upon receiving on-line incoming training data by performing the learning processes shown in Fig. 2.3. When the learning processes are done, three clusters (three fuzzy rules and CNN templates) were obtained. For the example of color filter, it takes about 1 minute to learn the structure (interconnection set) and 2 minutes to learn the parameters with Pentium IV 2.0G PC. However, the training can be done off-line, so it is not a problem for the on-line processing of CNN, which causes just little time.

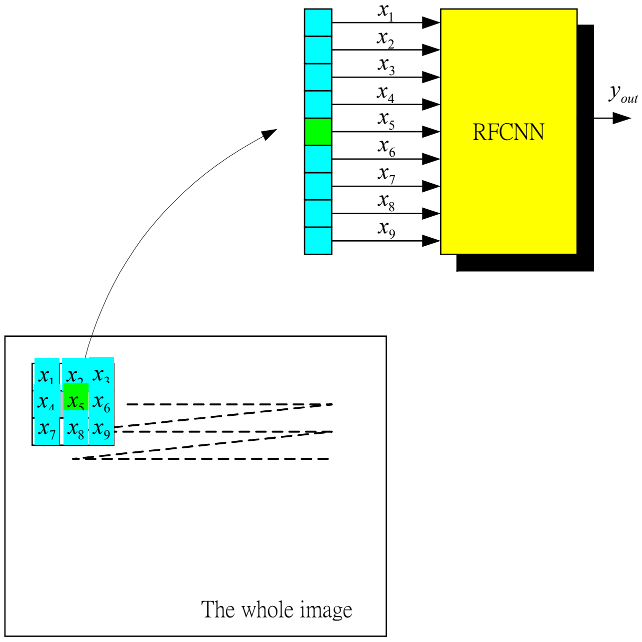
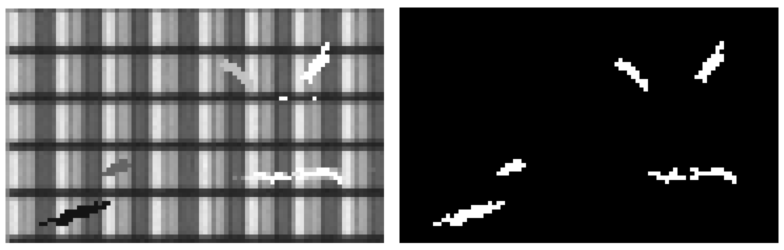


Figure 2.7 The training schematic diagram of the RFCNN.



(a) (b)

Figure 2.8 Training images. (a) Input image. (b) Desired output.

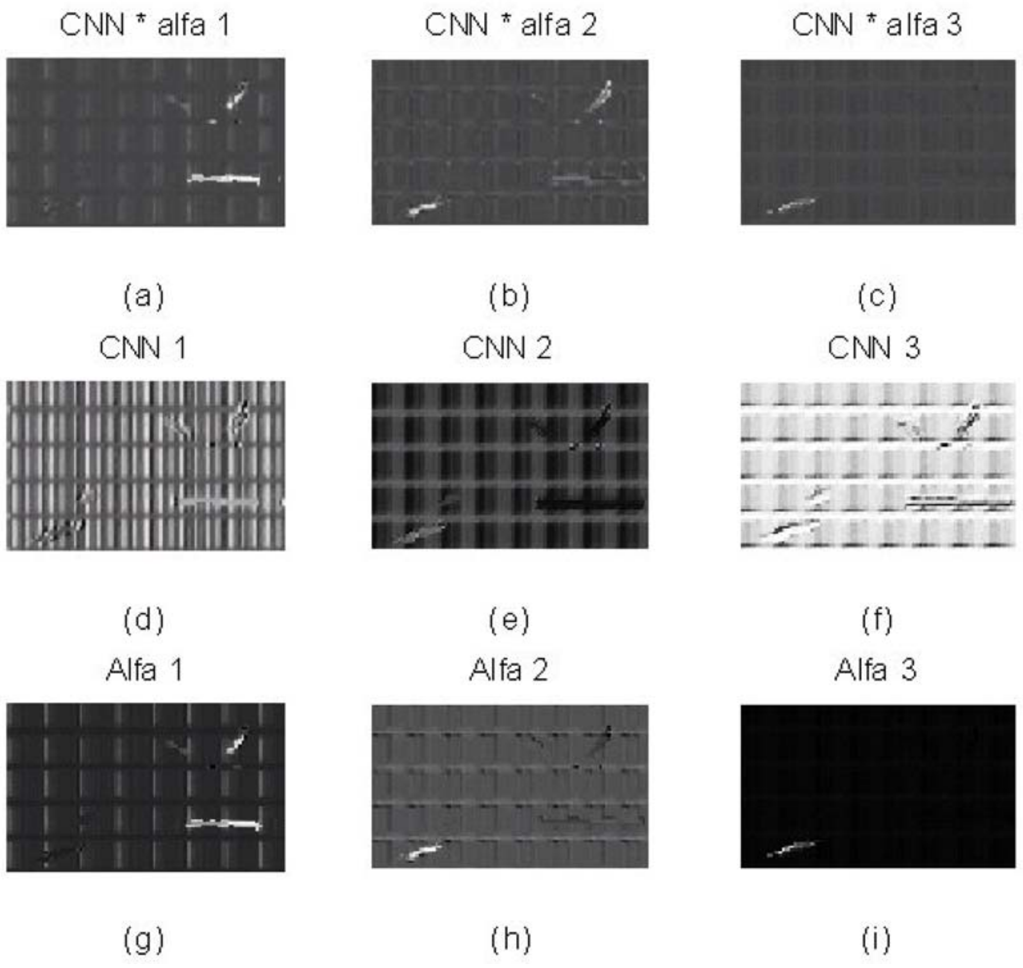


Figure 2.9 The outputs of Layer 3, 4, and Feedback Layer for the training image. (a)~(c) The outputs of the three Layer-4 nodes, respectively. (d)~(f) The outputs of the three CNNs in the Feedback Layer, respectively. (g)~(i) The outputs of the three Layer-3 nodes, respectively (firing strength of each rule).

Figure 2.9 shows the outputs of Layer 3, 4, and Feedback Layer for the training image. Figures 2.9(a) to 2.9(c) show the outputs of the three Layer-4 nodes, respectively, i.e., the outputs of the three CNNs in the Feedback Layer multiplied by the outputs of the three Layer-3 nodes (i.e., firing strength of each rule), respectively. Figures 2.9(d) to 2.9(f) show the outputs of the three CNNs in the Feedback Layer, respectively. Figures 2.9(g) to 2.9(i) show the outputs of the three Layer-3 nodes, respectively (firing strength of each rule). The sum of the outputs of the three Layer-4 nodes (i.e., Figs. 2.9(a) to 2.9(c)) forms the RFCNN final output. From Figs. 2.9(a) to 2.9(c), we can see that CNN 1 takes care of the defect texture on the right side of the training image, and CNNs 2 and 3 mainly take care of the defect textures on the left side of the training image. The template of each learned CNN is given as follows:

$$A^1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.64 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B^1 = \begin{bmatrix} 0.27 & -0.20 & 0.12 \\ 1.58 & -2.45 & 1.29 \\ 0.29 & -0.58 & 0.47 \end{bmatrix}, z^1 = -0.02.$$

$$A^2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.83 & 0 \\ 0 & 0 & 0 \end{bmatrix}, B^2 = \begin{bmatrix} 0 & -0.17 & -0.68 \\ 0.20 & -0.65 & 0.40 \\ -0.11 & 0.08 & -0.15 \end{bmatrix}, z^2 = 0.37.$$

$$A^3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.53 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B^3 = \begin{bmatrix} 0.09 & 1.26 & -1.22 \\ -1.58 & -1.70 & -0.57 \\ 0.59 & 0.50 & 1.54 \end{bmatrix}, z^3 = 1.78.$$

Based on the learned structure and parameters of the RFCNN, we test several images and three of those images as shown in Fig. 2.10. Figs. 2.10(a), 2.10(c), and 2.10(e) are the testing images and Figs. 2.10(b), 2.10(d), and 2.10(f) are the corresponding results of defect inspection. From Figs. 2.10(a) to 2.10(f), we can see that the learned structure and CNN templates of the RFCNN are well suited to detect the defects of color filer images. It has also been tested that detection results are still good if the images are shifted, that is because that the RFCNN only considers the central pixel and its 8 connected neighbors and they are still regular patterns after

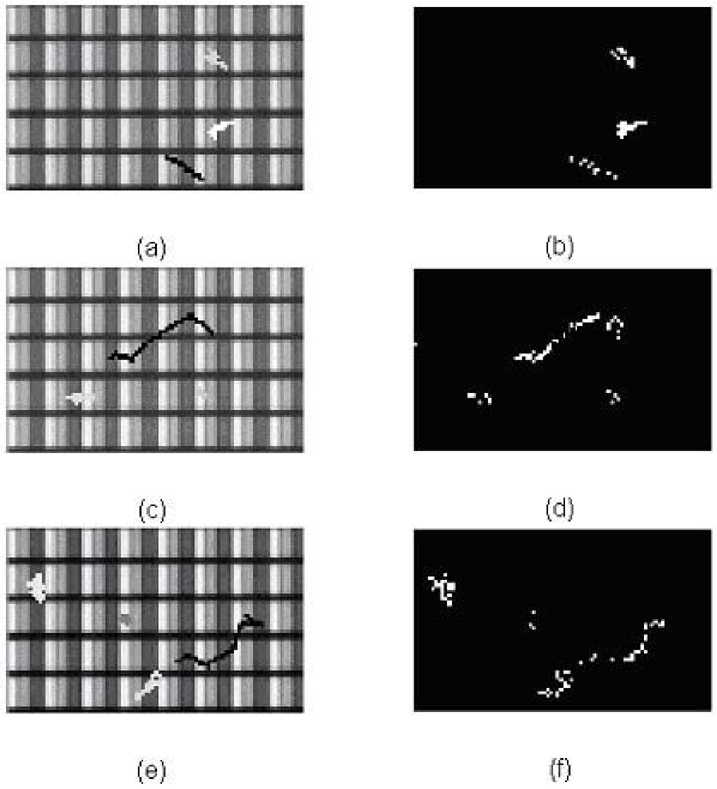


Figure 2.10 Experimental (Testing) results of the learned RFCNN. (a), (c), and (e) are input testing images. (b), (d), and (f) are corresponding detection results.

images are shifted. Therefore if the images are shifted, we need not re-teach the network.

The conventional methods using CNN for defect inspection [38]-[41] are using one or a set of CNN templates, which can be obtained by experiential engineers or learned by examples, to detect defect. To compare the RFCNN with conventional methods, we performed some experiments using a single CNN whose template is learned by genetic algorithm (GA). We find that the training image, shown in Fig. 2.8(a), cannot be learned well by using only a single CNN. However, if we have the training images and corresponding desired outputs as shown in Figs. 2.11(a) to 2.11 (d), the CNN template can be learned well by GA (A GA-based template learning for defect inspection is described in Appendix A.1). This fact implies that different kinds of defects in color filter need different CNN templates. That is, we can first identify the categories of defects and make each CNN template of defect category learned by GA. However, this will cause related questions as follows. First, how many defect categories, which determine how many CNN templates, should be classified? Second, how can we be sure which defects belong to the same category? In other words, what is the corresponding desired output for the uncategorized defects of color filter? Therefore it is difficult to manually use the divide-and-conquer principle to learn the templates of CNNs by GA. For the dilemma mentioned above, the proposed RFCNN provides a good alternative to solve this kind of problem.

To make the RFCNN converge more quickly during learning, GA can be used to learn some CNN templates to initialize the consequent part of the RFCNN. Though this experiment focuses on defect inspection of color filter, the proposed RFCNN can be also applied to those images with regular pattern, such as texture webs.

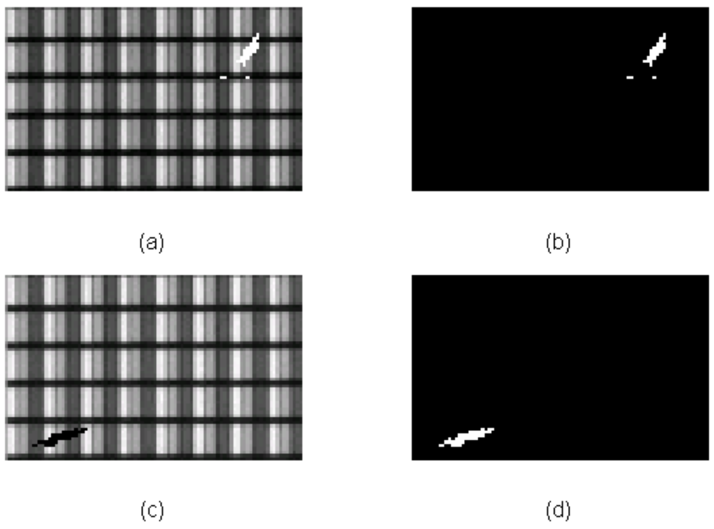


Figure 2.11 Training images by GA. (a) and (c) are input images. (b) and (d) are corresponding desired outputs.

The main idea of the proposed RFCNN is an integrated system of FIS and CNNs, which can construct fuzzy rules and CNN templates automatically. The example for the defect inspection of color filter has been demonstrated to verify the capability of the RFCNN. In addition to the defect inspection of color filter, we believe such an integrated CNN system, the RFCNN, has potential to solve more complex intelligent problems such as biological phenomena or other applications. Since CNN bears the characteristic of high-speed processing based on analog circuit realization, it will be very useful to realize the RFCNN by analog circuits. As studied in [7, 11], the elementary fuzzy-logic computations, such as the min, max, and fuzzification operator in a fixed neighborhood, have already been designed in CNN. Therefore, it is very promising and feasible to implement the RFCNN in the future work. An implementation scheme to realize the RFCNN includes the two following steps. First, use the RFCNN to learn the fuzzy rules and CNN templates. Second, construct a FIS based on the learned fuzzy rules and CNN templates.

For taking into account the non-idealities or mismatch due to the manufacturing, there are some ways can be done as follows:

1. We may add constraints with upper bound and lower bound to the learned parameter in learning algorithm.
2. The interval parameter learning is also available in [42] such that a tolerant range of parameters (weights) deviation can be achieved.
3. Since the proposed RFCNN can learn the structure and parameters automatically, we can increase the number of CNN and other nodes automatically to achieve the required accuracy if the target accuracy has not been satisfied.

2.5. Concluding Remarks

In this chapter we propose a novel framework, called RFCNN, for automatically constructing a multiple-CNN integrated neural system. This CNN-based fuzzy neural network can automatically learn its proper network structure and parameters simultaneously. The structure learning includes the creation of fuzzy rules and CNNs with a new on-line adaptive ICA mixture-model technique. The parameter learning includes the tuning of fuzzy membership functions and CNN templates based on the ordered derivative calculus. The proposed RFCNN provides a solution to the current dilemma on the decision of templates and/or fuzzy rules in the existing integrated (fuzzy) CNN systems. In order to verify the capability of the RFCNN, a real-world defect inspection problem has been demonstrated. The experimental results show that the proposed scheme is effective and promising. An extended framework to the RFCNN including the coupled CNNs is described in next chapter.

3. A Recurrent Fuzzy Coupled Cellular Neural Network System with Automatic Structure and Template Learning

In this chapter, we extend our previously proposed multi-CNN integrated system, called recurrent fuzzy CNN (RFCNN) which considers uncoupled CNNs only, to automatically learn the proper network structure and parameters simultaneously of coupled CNNs, which is called RFCCNN (recurrent fuzzy coupled CNN). The proposed RFCCNN provides a solution to the current dilemma on the decision of templates and/or fuzzy rules in the existing integrated (fuzzy) CNN systems. For comparison, the capability of the proposed RFCCNN is demonstrated on the same defect inspection problems. Simulation results show that the proposed RFCCNN outperforms the RFCNN.

3.1. Introduction

Since the RFCNN, described in Chapter 2, only considers the learning of uncoupled CNN templates, its ability to solve more complex high-level machine vision problems is quite limited. In this chapter we extend the RFCNN to automatically learn the proper network structure and parameters simultaneously of coupled CNNs. Due to the inclusion of coupled CNNs, the RFCCNN has shown more powerful abilities in detecting the fine detailed defects of color filters, compared to the previous RFCNN, in the simulations.

Since a new architecture of the RFCCNN, extended from the previous RFCNN, is proposed, the derivation of equations and learning algorithm, simulation results and discussions are novel. In short, there are two main differences. First, we propose a more complete architecture than the previous RFCNN. Second, simulation results show that the proposed RFCCNN outperforms the RFCNN.

The chapter is organized as follows. Section 2 describes the structure and functions of the proposed RFCCNN. Section 3 briefly describes the on-line structure and parameters learning algorithm for the RFCCNN. Section 4 gives simulation results and discussions. Finally, conclusions are summarized in the last section.

3.2. Structure of the RFCCNN

In this section, the structure of the proposed RFCCNN shown in Fig. 3.1 is introduced. For clarity, we consider a CNN, with time constant = 1, time step = 1, and neighborhood within a radius = 1, which is characterized by the following templates:

$$A^i = \begin{bmatrix} a^i{}_{-1,-1} & a^i{}_{-1,0} & a^i{}_{-1,1} \\ a^i{}_{0,-1} & a^i{}_{0,0} & a^i{}_{0,1} \\ a^i{}_{1,-1} & a^i{}_{1,0} & a^i{}_{1,1} \end{bmatrix}, B^i = \begin{bmatrix} b^i{}_{-1,-1} & b^i{}_{-1,0} & b^i{}_{-1,1} \\ b^i{}_{0,-1} & b^i{}_{0,0} & b^i{}_{0,1} \\ b^i{}_{1,-1} & b^i{}_{1,0} & b^i{}_{1,1} \end{bmatrix}, I^i = z^i \tag{3.1}$$

where A^i, B^i, and z^i is feedback template, control template, and bias of the *i*th CNN, respectively. By defining a CNN as above, the six-layered RFCCNN network will realize a fuzzy model of the following form:

Rule i: IF x_1 is M_1^i and $\ldots x_j$ is $M_j^i \ldots$ and x_9 is M_9^i

$$\text{THEN } y_i(t+1) \text{ is } f'(A^i y_i(t) + B^i u(t) + z^i(t)) \tag{3.2}$$

or

Rule i: IF x_1 is M_1^i and $\ldots x_j$ is $M_j^i \ldots$ and x_9 is M_9^i

$$\text{THEN } y_i(t+1) \text{ is } \begin{matrix} f'(a^i{}_{-1,-1} y_{i,1}(t) + a^i{}_{-1,0} y_{i,2}(t) + \ldots + a^i{}_{1,1} y_{i,9}(t) + \\ b^i{}_{-1,-1} x_1(t) + b^i{}_{-1,0} x_2(t) + \ldots + b^i{}_{1,1} x_9(t) + z^i(t)) \end{matrix}, \tag{3.3}$$

where the current input vector is $u = \mathbf{x}_t = [x_1, \ \ldots, \ x_9]^T$, $A^i y_i(t)$ is $a^i{}_{-1,-1}y_{i,1}(t) + a^i{}_{-1,0}y_{i,2}(t) + \ldots + a^i{}_{1,1}y_{i,9}(t)$, $B^i u(t)$ is $b^i{}_{-1,-1}x_1(t) + b^i{}_{-1,0}x_2(t) + \ldots + b^i{}_{1,1}x_9(t)$, $f^{'}$ is a bipolar sigmoid function, M_j^i is a fuzzy set, and $a_{k,l}^i$, $b^i{}_{k,l}$, and z^i are consequent parameters representing feedback template, control template, and bias of the *i*th CNN, respectively. The RFCCNN is a six-layered network structure with one feedback layer and can automatically learn its proper network structure (the creation of fuzzy rules and CNNs) and parameters (the tuning of fuzzy membership functions and CNN templates) simultaneously. In this chapter, as defined in (3.1) to (3.3), we extend the RFCNN to RFCCNN including the structure and parameter learning of coupled CNN cells. The six-layered network structure of the RFCCNN is mostly the same as that of the RFCNN, except for the feedback layer. In the feedback layer of the RFCCNN, the feedbacks are from coupled CNNs, i.e., $A^i y_i(t)$ is $a^i{}_{-1,-1}y_{i,1}(t) + a^i{}_{-1,0}y_{i,2}(t) + \ldots + a^i{}_{1,1}y_{i,9}(t)$, but in the feedback layer of the original RFCNN, the feedbacks are from uncoupled CNNs only, i.e., $A^i y_i(t)$ is $a^i{}_{0,0}y_i(t)$. The details of the function of each node of the RFCNN are described in Section 2 of Chapter 2.

3.3. Learning Algorithms for the RFCCNN

Similarly, two types of learning, structure and parameter learning, are used concurrently for the RFCCNN. The structure-learning algorithm of the RFCCNN is the same as those of the RFCNN. For the details of structure-learning algorithm of RFCCNN based on on-line ICA mixture model are described in Section 3.1 of Chapter 2. For parameter-learning algorithm of RFCCNN, they are described in the

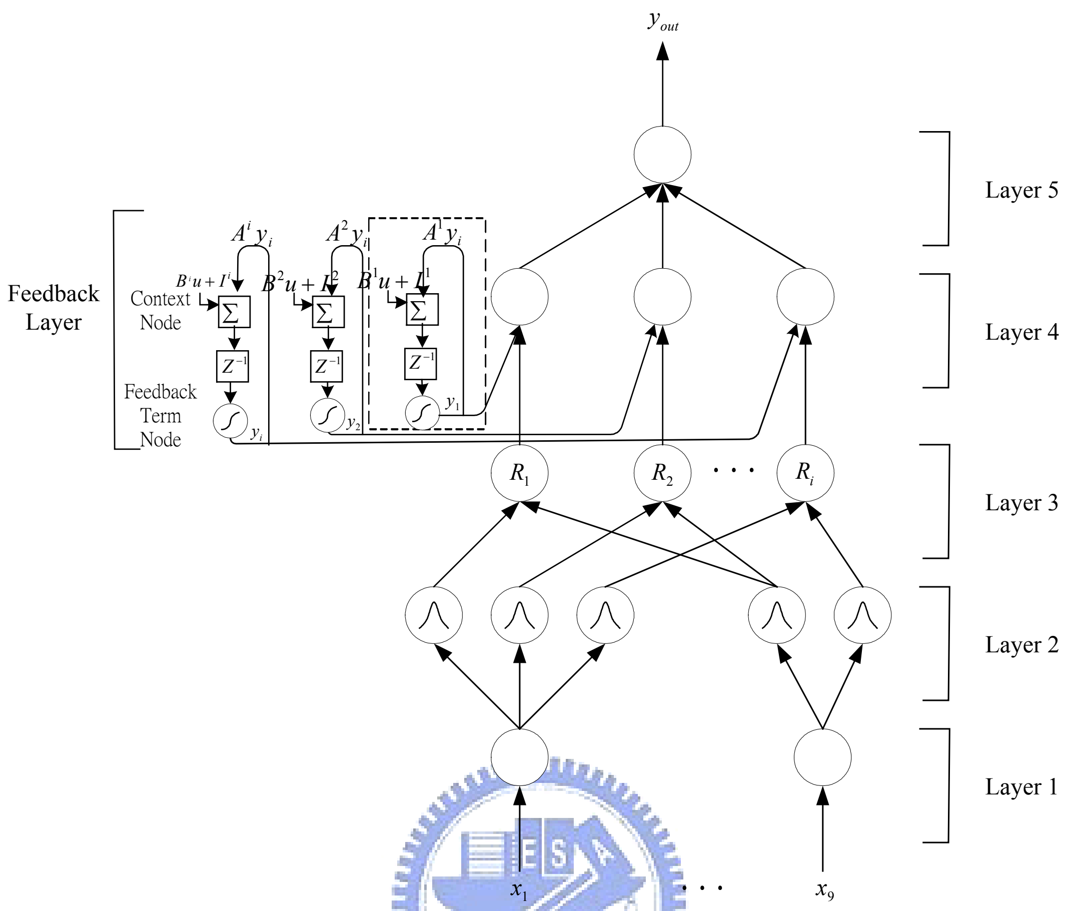


Figure 3.1 Structure of the proposed RFCCNN.

rest of this section.

After the network structure is adjusted according to the current training pattern, the network then enters the parameter identification phase to adjust the parameters of the network optimally based on the same training pattern. Notice that the following parameter learning is performed on the whole network after structure learning; no matter whether the nodes (links) are newly added or are existent originally. Since the RFCCNN is a dynamic system with feedback connections, the backpropagation learning algorithm cannot be applied to it directly. Also, due to the on-line learning property of the RFCCNN, the off-line learning algorithms for the recurrent neural networks, like backpropagation through time and time-dependent recurrent backpropagation [12], cannot be applied here. Instead, the ordered derivative [14], which is a partial derivative whose constant and varying terms are defined using an ordered set of equations, is used to derive our learning algorithm. The ordered set of equations, described in Section 2 in each layer, is summarized in (3.5)-(3.10). Our goal is to minimize the error function

$$E(t+1) = \frac{1}{2}[y_{out}(t+1) - y_{out}^{d}(t+1)]^2 = \frac{1}{2}\varepsilon(t+1)^2, \tag{3.4}$$

where $y_{out}^{d}(t+1)$ is the desired output, $y_{out}(t+1)$ is the current output, and $\varepsilon(t+1)$ is $(y_{out}(t+1) - y_{out}^{d}(t+1))$. For each training data set, starting at the input nodes, a forward pass is used to compute the activity levels of all the nodes in the network to obtain the current output $y_{out}(t+1)$. In the followings, dependency on time will be omitted unless emphasis on temporal relationships is required.

Summarizing the node functions defined in Section 2, the function performed by the network is

$$y_{out}(t+1) = \sum_{i} u_i^{(5)} \tag{3.5}$$

$$u_i^{(5)} = o^{(4)} = o^{(6)} \cdot h^i, \tag{3.6}$$

where

$$h^i = f[u_i^{(3)}] = \prod_{i} u_i^{(3)} \tag{3.7}$$

$$o^{(6)} = \frac{2}{1+e^{-2x^i}} - 1 \tag{3.8}$$

$$x^i(t+1) = A_{\mathrm{j}}^{\mathrm{i}} y_i(t) + B_{\mathrm{j}}^{\mathrm{i}} u(t) + z^i(t), \tag{3.9}$$

and (3.1) is redefined as the following equation for clarity:

$$A_{\mathrm{j}}^{\mathrm{i}} = [a_1^i, a_2^i, a_3^i; a_4^i, a_5^i, a_6^i; a_7^i, a_8^i, a_9^i],\ B_{\mathrm{j}}^{\mathrm{i}} = [b_1^i, b_2^i, b_3^i; b_4^i, b_5^i, b_6^i; b_7^i, b_8^i, b_9^i]. \tag{3.10}$$

With the above formula and the error function defined in (3.4), we can derive the update rules for the free parameters in the RFCCNN as follows.

Update rule of a_j^i (the parameter of feedback template of the *i*th CNN) is

$$a_j^i(t+1) = a_j^i(t) - \eta \frac{\partial^+ E(t+1)}{\partial a_j^i} \tag{3.11}$$

$$\begin{aligned} \frac{\partial^+ E(t+1)}{\partial a_j^i} &= \frac{\partial E\ (t+1)}{\partial a_j^i} + \sum_k \frac{\partial E\ (t+1)}{\partial y_{out,k}(t+1)} \frac{\partial^+ y_{out,k}(t+1)}{\partial a_j^i} \\ &= \frac{\partial E\ (t+1)}{\partial y_{out}(t+1)} \frac{\partial^+ y_{out}(t+1)}{\partial a_j^i} \end{aligned} \tag{3.12}$$

where

$$\frac{\partial E\ (t+1)}{\partial y_{out}(t+1)} = \varepsilon(t+1), \tag{3.13}$$

and

$$\frac{\partial^+ y_{out}(t+1)}{\partial a_j^i} = \sum_k \frac{\partial y_{out}\ (t+1)}{\partial o_k^{(4)}} \frac{\partial^+ o_k^{(4)}}{\partial a_j^i} = \frac{\partial y_{out}(t+1)}{\partial o_i^{(4)}} \frac{\partial^+ o_i^{(4)}}{\partial a_j^i}. \tag{3.14}$$

where

$$\frac{\partial y_{out}(t+1)}{\partial o_i^{(4)}} = \frac{\partial}{\partial o_i^{(4)}} \sum_k o_k^{(4)}(t+1) = 1, \tag{3.15}$$

and

$$\begin{aligned} \frac{\partial^+ o_i^{(4)}}{\partial a_j^i} &= \frac{\partial}{\partial a_j^i}[o_i^{(6)}(A_{\mathrm{j}}^{\mathrm{i}} y_i(t) + B_{\mathrm{j}}^{\mathrm{i}} u(t) + z^i(t))]h^i \\ &= h^i(1+o_i^{(6)})(1-o_i^{(6)})[y_i(t) + a_j^i \frac{\partial y_i(t)}{\partial a_j^i}] \end{aligned} . \tag{3.16}$$

Hence, the parameter a^i is updated by

$$a_j^i(t+1) = a_j^i(t) - \eta\varepsilon(t+1)\{h^i(1+o_i^{(6)})(1-o_i^{(6)})[y_i(t) + a_j^i \frac{\partial y_i(t)}{\partial a_j^i}]\}. \tag{3.17}$$

Similarly, the parameter b_j^i (the parameters of control template of the *i*th CNN) is updated by

$$b_j^i(t+1) = b_j^i(t) - \eta\varepsilon(t+1)[h^i(1+o_i^{(6)})(1-o_i^{(6)})x_j(t)] \tag{3.18}$$

and the parameter z_i (the bias of the *i*th CNN) is updated by

$$z^{i}(t+1)=z^{i}(t)-\eta\varepsilon(t+1)[h^{i}(1+o_{i}^{(6)})(1-o_{i}^{(6)})]. \qquad (3.19)$$

As shown in (3.14) to (3.16), the update rules are in recursive form. The value $\partial^{+}y/\partial a_{j}^{i}$ is equal to zero initially. For the rest free parameters in the RFCCNN, they are obtained during the structure-learning phase by the on-line ICA mixture model algorithm proposed in Section 3.1 of Chapter 2.

3.4. Experimental Results and Discussions

For comparison to the RFCNN described in Chapter 2, the capability of the proposed RFCCNN is demonstrated on the same defect inspection problems for color filter. As mentioned earlier, the difficulties in the defect inspection of color filter are its complex texture and demand for high-speed processing. For the reasons of high-speed processing, and that different kinds of defects in color filter need different CNN templates and some complex defects cannot be detected by a single CNN, the proposed RFCCNN is a good alternative to detect defect of color filter images. The way to train the RFCCNN is the same as the RFCNN. The training image and corresponding desired output are shown in Figs. 2.8(a) and 2.8(b). As mentioned in Section 3, there are no rules (and no CNNs) in the RFCCNN initially. They are created dynamically as learning proceeds upon receiving on-line incoming training data by performing the learning processes. When the learning processes are done, six clusters (six fuzzy rules and CNN templates) were obtained. For the example of color filter, it takes about 90 seconds to learn the structure (interconnection set) and the parameters with Pentium IV 2.0G PC. However, the training can be done off-line, so it is not a problem for the on-line processing of CNN. For simulation in this chapter, it causes about 9 seconds. After the proposed RFCCNN is implemented by analog circuit in the future, we believe that the processing times will be much faster than those of simulation.

Figure 3.2 shows the output of RFCCNN and the outputs of Layer 3, 4, and Feedback Layer for the training image. Figures 3.2(a) shows the output of the RFCCNN. Figures 3.2(b) to 3.2(g) show the outputs of the six Layer-4 nodes, respectively, i.e., the outputs of the six CNNs in the Feedback Layer multiplied by the outputs of the six Layer-3 nodes (i.e., firing strength of each rule), respectively. Figures 3.2(h) to 3.2(m) show the outputs of the six CNNs in the Feedback Layer, respectively. Figures 3.2(o) to 3.2(t) show the outputs of the six Layer-3 nodes, respectively (firing strength of each rule). The sum of the outputs of the six Layer-4 nodes (i.e., Figs. 3.2(b) to 3.2(g)) forms the RFCCNN final output (Fig. 3.2(a)). From Figs. 3.2(b) to 3.2(g), we can see that CNNs 4 and 5 take care of the defect texture on the right side of the training image, CNNs 1 and 6 mainly take care of the defect textures on the left side of the training image, and the other CNNs balance the output of the RFCCNN. The template of each learned CNN is given in (3.20).

Based on the learned structure and parameters of the RFCCNN, we test several images and some of those images as shown in Fig. 3.3. Figs. 3.3(a), 3.3(d), and 3.3(g) are input testing images. Figs. 3.3(b), 3.3(e), and 3.3(h) are corresponding detection results of the RFCCNN. Figs. 3.3(c), 3.3(f), and 3.3(i) are corresponding detection results of the uncoupled RFCNN. From Figs. 3.3(b), 3.3(e), and 3.3(h), we can see that the learned structure and CNN templates of the RFCCNN are well suited to detect the defects of color filer images. It has also been confirmed that detection results are still good if the images are shifted or rotated. One of simulation results is shown in Fig. 3.4.

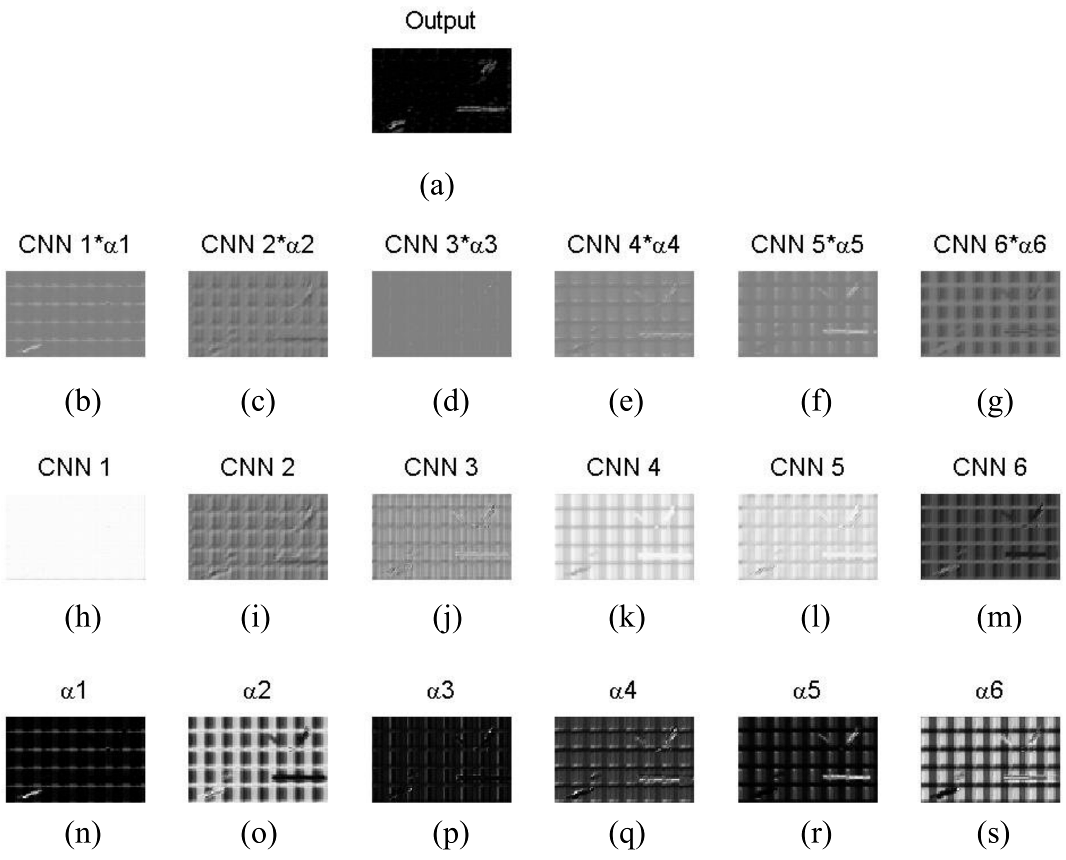


Figure 3.2 The outputs of Layer 3, 4, and Feedback Layer for the training image. (a) The output of the RFCCNN. (b)~(g) The outputs of the six Layer-4 nodes, respectively. (h)~(m) The outputs of the six CNNs in the Feedback Layer, respectively. (n)~(s) The outputs of the six Layer-3 nodes, respectively (firing strength of each rule).

$$\begin{aligned}
A^1 &= \begin{bmatrix} -0.34 & -0.24 & 0 \\ -0.20 & 0 & 0.39 \\ 0.60 & 0.43 & 0.19 \end{bmatrix}, B^1 = \begin{bmatrix} 0.67 & 0.15 & -0.03 \\ -0.31 & -0.54 & 0.09 \\ 0.10 & -0.08 & 0.28 \end{bmatrix}, z^1 = 0.90. \\
A^2 &= \begin{bmatrix} -0.30 & -0.23 & 0 \\ -0.42 & -0.39 & -0.14 \\ 0 & -0.12 & 0 \end{bmatrix}, B^2 = \begin{bmatrix} -0.50 & -0.22 & -0.01 \\ -0.08 & 0.11 & 0.19 \\ -0.06 & 0.25 & 0.07 \end{bmatrix}, z^2 = 0.10. \\
A^3 &= \begin{bmatrix} -0.25 & -0.20 & 0.1 \\ -0.46 & -0.30 & -0.06 \\ -0.05 & -0.08 & -0.09 \end{bmatrix}, B^3 = \begin{bmatrix} -0.03 & 0.13 & -0.18 \\ -0.39 & 0.49 & -0.18 \\ 0.26 & 0.22 & 0.51 \end{bmatrix}, z^3 = -0.05. \\
A^4 &= \begin{bmatrix} -0.11 & 0.01 & 0.28 \\ -0.25 & -0.26 & -0.02 \\ -0.06 & -0.06 & 0.11 \end{bmatrix}, B^4 = \begin{bmatrix} 0.11 & 0.40 & 0.25 \\ -0.08 & 0.61 & 0.39 \\ 0. & 0.01 & 0.07 \end{bmatrix}, z^4 = 0.33. \\
A^5 &= \begin{bmatrix} -0.30 & -0.34 & -0.03 \\ -0.27 & -0.16 & -0.01 \\ 0.16 & -0.10 & 0.18 \end{bmatrix}, B^5 = \begin{bmatrix} 0.64 & -0.10 & 0.27 \\ 0.12 & 0.90 & 0.09 \\ 0.08 & 0.54 & -0.15 \end{bmatrix}, z^5 = -0.14. \\
A^6 &= \begin{bmatrix} -0.14 & -0.09 & 0.06 \\ -0.37 & -0.27 & -0.14 \\ 0.03 & 0 & 0.12 \end{bmatrix}, B^6 = \begin{bmatrix} 0.35 & -0.17 & -0.02 \\ -0.57 & -0.28 & -0.03 \\ -0.64 & -0.12 & -0.20 \end{bmatrix}, z^6 = -0.16.
\end{aligned} \tag{3.20}$$

From Fig. 3.3, we can see some differences between RFCCNN (coupled RFCNN) and uncoupled RFCNN for defect inspection of color filter. First, as shown in Figs. 3.3(g) to 3.3(i), the results of the coupled RFCNN is better than those of uncoupled RFCNN for detecting the large defects on the top-left of test image shown in Fig. 3.3(g). Second, as shown in all test images and corresponding detection results, the results of the coupled RFCNN are better than those of uncoupled RFCNN for detecting the black defects. The results of the coupled RFCNN are better for detecting the large defects in that the coupled RFCNN, like coupled CNN, has fully outputs feedback and will take care of further neighboring pixels. Some quantity comparisons are shown in Table 3.1. Here detection rate is defined as the ratio of detected defect pixel number to real defect pixel number. As we can see from Table 3.1, the detection rates of the RFCCNN are better than those of the RFCNN, no matter for the all defects, large defects, or black defects. More testing images and corresponding detection results of the RFCCNN are shown in Fig. 3.5 to 3.8.

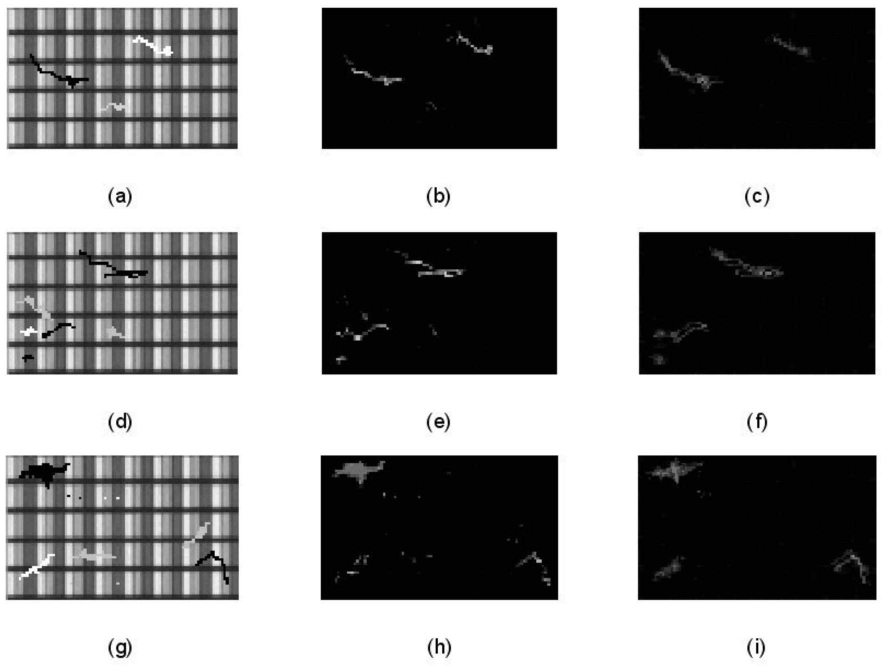


Figure 3.3 Simulation (Testing) results of the learned RFCCNN and RFCNN. (a), (d), and (g) are input testing images. (b), (e), and (h) are corresponding detection results of RFCCNN. (c), (f), and (i) are corresponding detection results of uncoupled RFCNN.

Table 3.1 Comparison of detection rate.

	Detection rate of the RFCCNN	Detection rate of the RFCNN
All defects	56%	43%
Large defects	76%	52%
Black defects	72%	51%

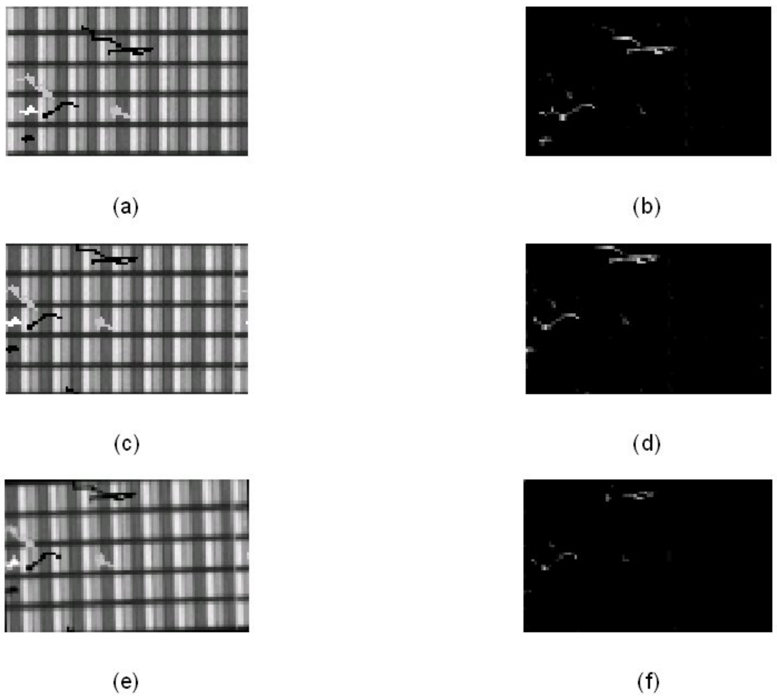


Figure 3.4 Simulation results of shifted and rotated images. (a), (c), and (e) are input testing images of original, shifted, and rotated ones, respectively. (b), (d), and (e) are corresponding detection results of RFCCNN.

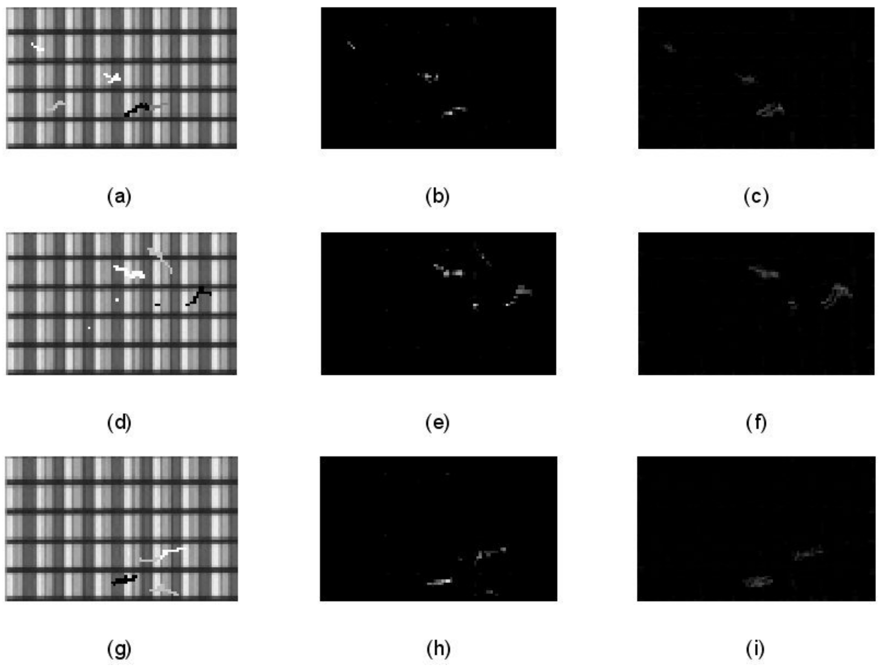


Figure 3.5 Other testing results of the learned RFCCNN and RFCNN, part 1. (a), (d), and (g) are input testing images. (b), (e), and (h) are corresponding detection results of RFCCNN. (c), (f), and (i) are corresponding detection results of uncoupled RFCNN.

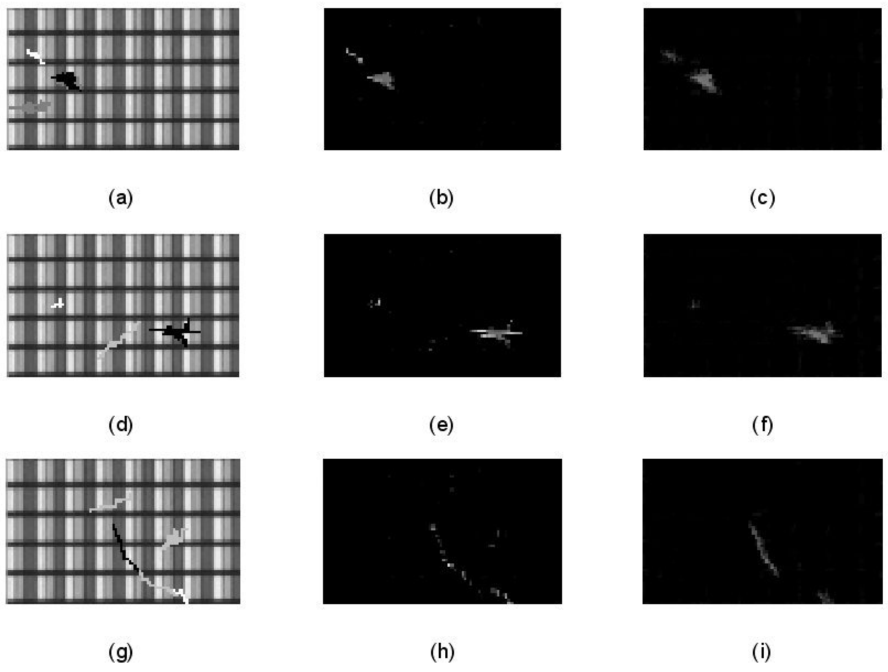


Figure 3.6 Other testing results of the learned RFCCNN and RFCNN, part 2. (a), (d), and (g) are input testing images. (b), (e), and (h) are corresponding detection results of RFCCNN. (c), (f), and (i) are corresponding detection results of uncoupled RFCNN.

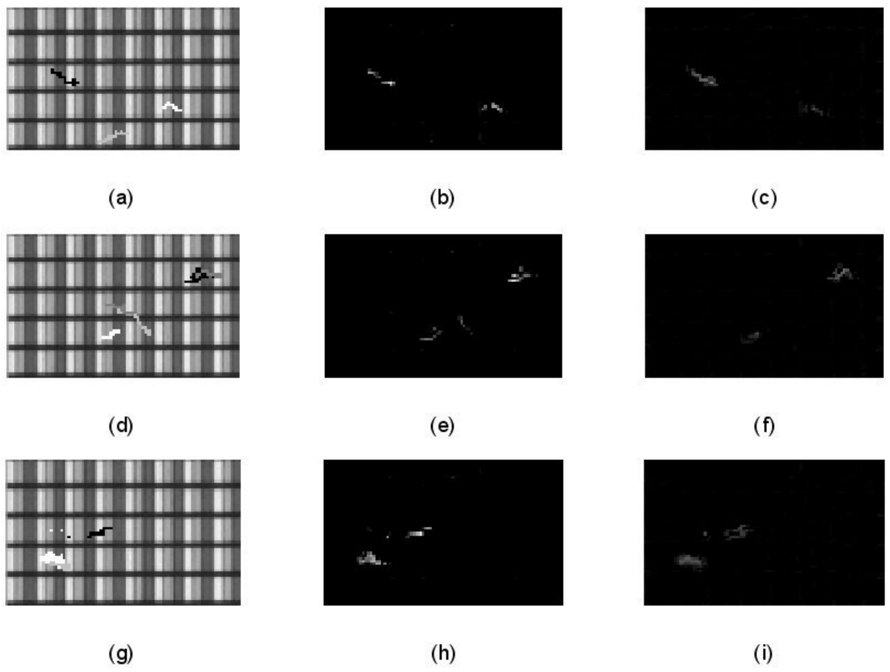


Figure 3.7 Other testing results of the learned RFCCNN and RFCNN, part 3. (a), (d), and (g) are input testing images. (b), (e), and (h) are corresponding detection results of RFCCNN. (c), (f), and (i) are corresponding detection results of uncoupled RFCNN.

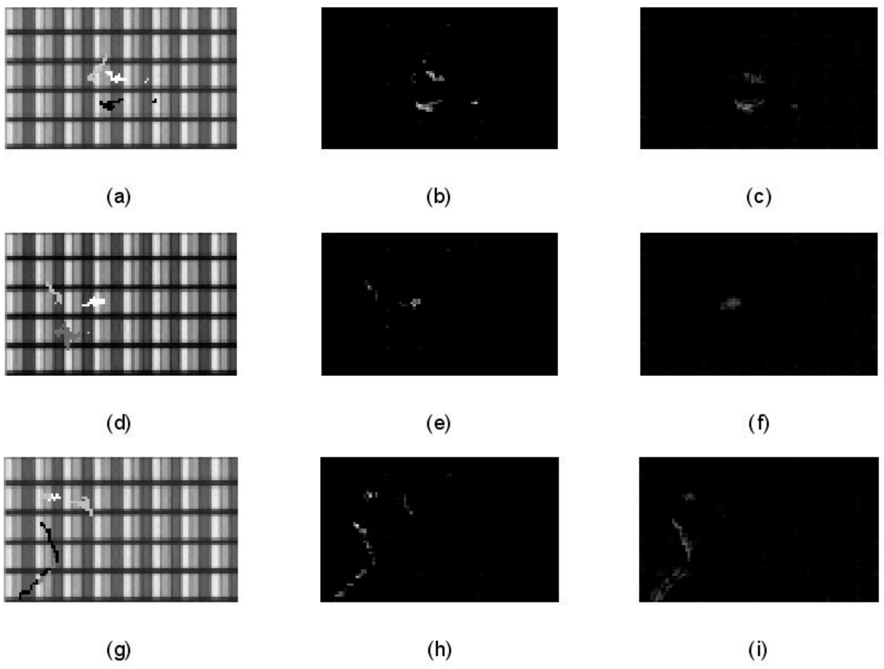


Figure 3.8 Other testing results of the learned RFCCNN and RFCNN, part 4. (a), (d), and (g) are input testing images. (b), (e), and (h) are corresponding detection results of RFCCNN. (c), (f), and (i) are corresponding detection results of uncoupled RFCNN.

The main idea of the proposed RFCCNN is an integrated system of FIS and CNNs, which can construct fuzzy rules and CNN templates automatically. There are two general approaches to realizing the RFCNN model by real-time hardware. First, it can be "coded" and run in the CNN universal machine (CNNUM). The CNN-UM can handle analog and digital signals with built-in converters and memories by instructions. It is generally considered as a general-purpose image-processing computer. In fact, a fuzzy-rule-based image-processing algorithm [7] has been successfully implemented on the CNN-UM. Another approach is to design application-specific RFCNN circuits for particular applications with prelearned fuzzy rules and CNN templates. To achieve this, the circuit design technique of multilayer CNN [43, 44] called MLCNN can be applied to implement the multilayer structure of the proposed RFCNN model in Figure 3. A CNN-based Gaussian function circuit as designed in [45] can realize the Gaussian membership function required in Layer 2.

The fuzzy logic operations in Layers 3 and 4 can be realized by analog CNN circuits as studied in [7, 11]. Therefore, it is very promising and feasible to implement the RFCCNN using high-speed analogic circuits.

3.5. Concluding Remarks

In this chapter, we extended our previous work, called RFCNN, from considering uncoupled CNNs to coupled CNNs, called RFCCNN, for automatically constructing a multiple-CNN integrated neural system. This CNN-based fuzzy neural network can automatically learn its proper network structure and parameters simultaneously. In order to verify the capability of the RFCCNN, a defect inspection problem has been demonstrated and compared to the RFCNN. The simulation results show that the performance of the proposed RFCCNN is better than that of RFCNN. In addition, a scheme for hardware realization of the RFCCNN has been proposed. We believe such an integrated CNN system, the RFCCNN, has potential to solve more complex intelligent problems.

4. Conclusions and Perspectives

In this thesis we propose two novel frameworks, called RFCNN and RFCCNN, for automatically constructing a multiple-CNN integrated neural system to solve more complex intelligent problems. This CNN-based fuzzy neural network can automatically learn its proper network structure and parameters simultaneously. That is, the fuzzy rules and the corresponding templates of CNNs can be learned and obtained automatically, instead of assigning the corresponding templates of CNNs in advance and manually getting the fuzzy rules by domain experts. In the RFCNN/RFCCNN, each learned fuzzy rule corresponds to a CNN. Hence, each CNN takes care of a fuzzily separated problem region, and the functions of all CNNs are integrated through the fuzzy inference mechanism. A new on-line adaptive ICA mixture-model technique is proposed for structure learning of the RFCNN/RFCCNN, which can be applied to not only the proposed CNN integrated systems (i.e., the RFCNN and the RFCCNN), but also generic neural fuzzy network, such as ICA-mixture-model-based self-constructing FNN [46]. The proposed RFCNN/RFCCNN provides a solution to the current dilemma on the decision of templates and/or fuzzy rules in the existing integrated (fuzzy) CNN systems. In order to verify the capability of the RFCNN and the RFCCNN, a real-world defect inspection problem has been demonstrated and compared. The experimental results show that the proposed schemes are effective and promising. Besides the defect inspection of color filter, the proposed RFCNN/RFCCNN can be also applied to those images with regular pattern, such as texture webs.

In addition, a scheme for hardware realization of the RFCCNN has been proposed. We believe such an integrated CNN system, the RFCCNN, has potential to solve more complex intelligent problems and those that one single CNN cannot solve. The future works include finding more real-world applications, extending the framework, and designs specific-application analogic CNN circuits.

Appendix

A.1 GA-based Template Learning for Defect Inspection

In this appendix, we propose a CNN-based defect inspection algorithm to detect defect images with regular pattern such as color filter, where the templates of CNN are obtained by means of GA. The CNN is a good alternative to achieve the high speed processing of defect inspection and genetic algorithm is applied to learn templates of CNN, which can perform defect inspection for color filter and web material images. Experimental results show that the proposed algorithm is a promising method to detect the defects of color filter images.

A1.1 Introduction

In [38]-[41], a CNN (cellular neural network) based defect inspection algorithm was proposed to detect defect images. However, their test images belonged to uniform webs such as metal laminates [38], [39] and cotton [41]. In this appendix, we propose a CNN-based defect inspection algorithm to detect defect images with regular pattern such as color filter and texture webs images, where the templates of CNN are obtained by means of GA (genetic algorithm). The main advantages of using CNN are capable of operating at a very high speed. Many industrial inspections such as visual quality control of color filter and web materials mentioned above, require real-time processing. Therefore analog CNN chips are well suited to this kind of visual inspection applications.

Several prototype CNN architectures have been implemented to show the capability of very high speed compared with traditional digital image processor. However, even in the simplest case, there is no methodology that allows us to get the template associated to a given image-processing task. Nossek [12] made an overview of CNN template design and learning. The template coefficients of a CNN can be found by design [12], [13] or by learning [12], [38], [39]. "By design" or "designing fixed points" bears the following drawbacks: it is only suitable to binary images and it is not guaranteed that all its initial states will converge to the prescribed global stable points. "By learning" or "global learning" [12], [13], [38], [39], which has to learn by a set of input images (training patterns) and the corresponding desired output images, is a robust tool to obtain template coefficients of a CNN. In this appendix, we choose GA to determine CNN template coefficients for defect inspection, since GA is a global learning tool and it is very suitable to gray level images. The GA-based template learning for defect inspection will be described in the following section.

A1.2 The Proposed GA-based Template Learning

In the section, we shall introduce a CNN-based defect inspection scheme with the feature of automatic template generation learned from input/output training examples using GA (genetic algorithm). The number of parameters in a complete CNN template set (neighborhood within a radius =1) is 19, which is shown in (A.1); where 9 parameters for each matrix (*A* and *B* 3x3 matrices) and one for the bias current *I*. In the case that the image processing task has a symmetric behavior (e.g., averaging, border detection, etc.), the chromosome length can be reduced to 7 representative parameters (3 parameters for each matrix and another one for the bias current).

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}, Z = I \qquad \text{(A.1)}$$

The speed of convergence in the GA depends on the selected options. Population size determines the number of chromosomes in population. If there are too many chromosomes, GA converges slowly. On the other hand, if there are too few chromosomes, only a small part of search space is explored. Crossover selects genes from parent chromosomes and creates a new offspring. In this algorithm, a one-crossover point is used. Mutation follows crossover. It is to prevent falling all solutions in population into a local optimum by changing randomly the new offspring.

This algorithm searches the template to minimize a performance index proportional to the difference between pixels from the current settled output image and the desired one. This performance index is as (A.2):

$$f(p) = \beta \sum_{i=1}^{k} | y_i^d - y_i(p) | \qquad \text{(A.2)}$$

where p is the template, k is the number of the CNN cells, y_i^d is the value of ith pixel of desired output, $y_i(p)$ is the value of corresponding pixel of settled output, β = 5 if the output image is completely white or black, and β = 1 otherwise. The overall flow of automatic template generation using GA is shown in Fig. A.1.

A1.3 Experimental Results

In this section, the genetic algorithm has been used to learn some templates, which can perform defect inspection for color filter images. Our template format is shown in (A.3), which resulted in good performance in our experimental results.

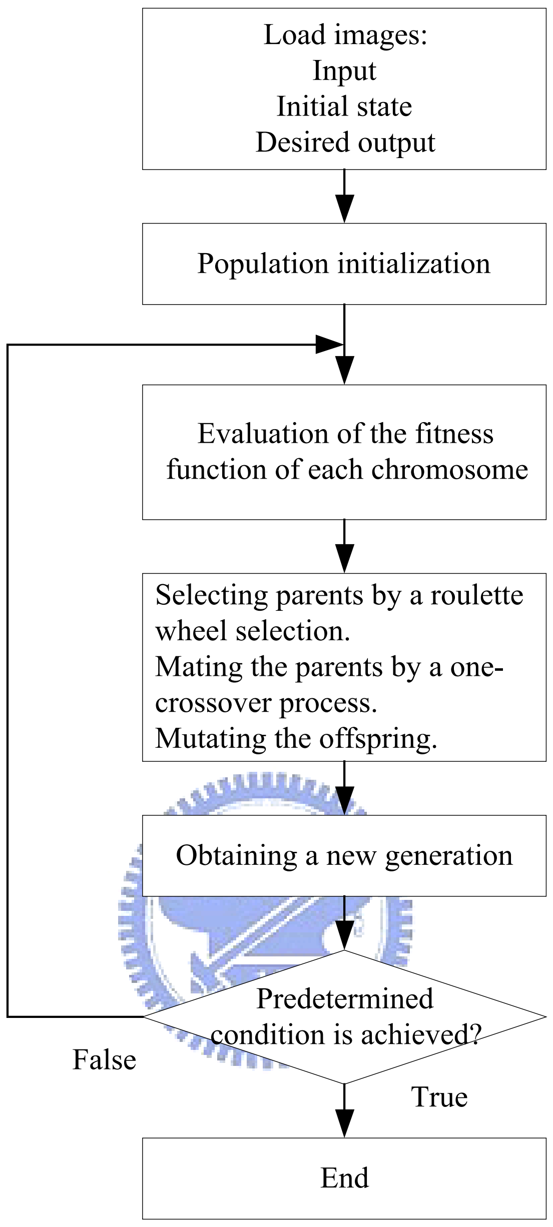

Load images:
Input
Initial state
Desired output
Population initialization
Evaluation of the fitness function of each chromosome
Selecting parents by a roulette wheel selection.
Mating the parents by a one-crossover process.
Mutating the offspring.
Obtaining a new generation
Predetermined condition is achieved?
False
True
End

Fig. A.1 The overall flow of automatic template generation using GA.

$$A=\begin{bmatrix} 0 & a_{12} & 0 \\ a_{12} & a_{22} & a_{12} \\ 0 & a_{12} & 0 \end{bmatrix}, B=\begin{bmatrix} 0 & b_{12} & 0 \\ b_{12} & b_{22} & b_{12} \\ 0 & b_{12} & 0 \end{bmatrix}, Z=I \qquad \text{(A.3)}$$

We utilize 5 bits for encoding each template parameter, then the utilized length of the chromosome for encoding the template parameters is 5 (parameters) x 5 (bits/parameters) = 25 bits. The CNN simulator parameters used in GA are as follows: population size = 60, crossover probability = 80%, mutation rate = 5%, the time step of the simulation = 0.2, the number of iteration = 25, (which correspond approximately to 5 τ when $R = 1$ and $C = 1$). To improve GA performance, we use the elitist strategy that copies the best member of each generation into the succeeding generation.

Figure A.2 shows the trained images. Fig. A.2(a) shows the input image and Fig. A.2(b) shows the desired image. The templates, learned by using GA according to (A.2), are shown in (A.4). Fig. A.3 shows the test images with templates learned by GA. Figs. A.3(a), A.3(c), and A.3(e) are the input images and Figs. A.3(b), A.3(d), and A.3(f) are the corresponding results of defect inspection with the templates in (A.4). From Figs. A.3(a) to A.3(f), we can see that the learned templates are well suited to detect the defect of color filer images.

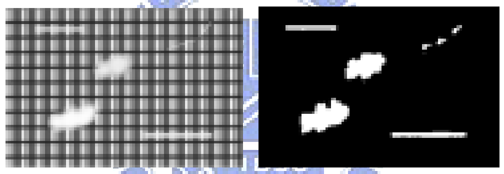

(a) (b)

Fig. A.2 Trained images. (a) Input image. (b) Desired output.

$$A = \begin{bmatrix} 0 & 3.6 & 0 \\ 3.6 & -7.4 & 3.6 \\ 0 & 3.6 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 8.7 & 0 \\ 8.7 & -1.0 & 8.7 \\ 0 & 8.7 & 0 \end{bmatrix}, Z = -10.0 \qquad (A.4)$$

As mentioned earlier, we focus on defect images with regular defect patterns, which belong to texture webs and cannot be detected by simple algorithms such as thresholding. Fig. A.4 shows the compared results. Fig. A.4(a) shows the defect detection result by GA, where the original image is from Fig. A.3(e). Figure A.4(b) shows defect detection result by thresholding. Obviously, simple algorithms such as thresholding do not work for this kind of defect images.

To test the robustness of templates obtained by the genetic algorithm, we change template values by 10 % variation randomly. The modified templates are shown in (A.5) and (A.6). The input image is shown in Fig. A.3(e) and the corresponding test results are shown in Fig. A.5(a) and A.5(b). Figure A.5(a) shows good result due to parameter variation. However, Fig. A.5(b) shows poor result because some false defects are detected.

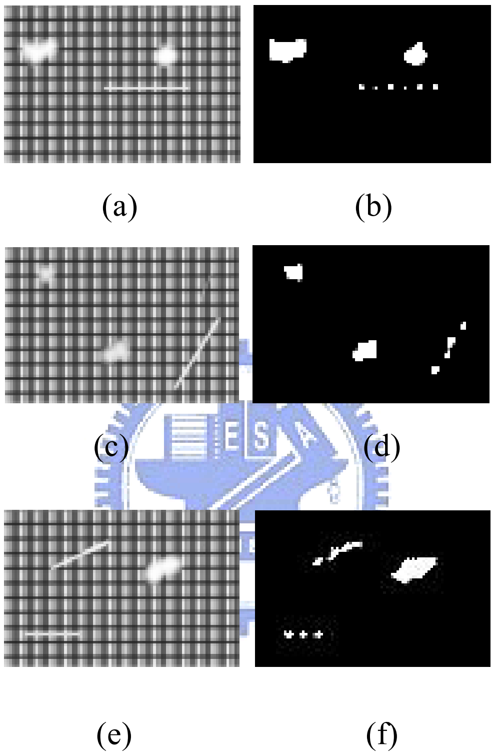


Fig. A.3 Test images using templates learned by GA.

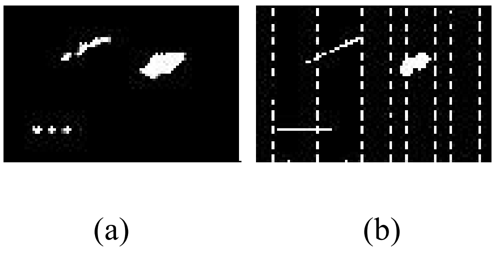
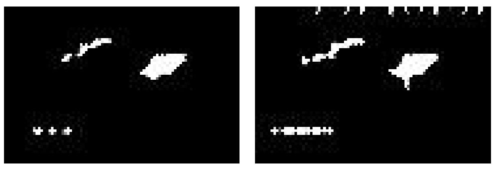


Fig. A.4 Test results obtained by (a) GA. (b) thresholding.

$$A=\begin{bmatrix}0 & 3.3 & 0\\ 3.3 & -6.7 & 3.3\\ 0 & 3.3 & 0\end{bmatrix}, B=\begin{bmatrix}0 & 7.9 & 0\\ 7.9 & -0.9 & 7.9\\ 0 & 7.9 & 0\end{bmatrix}, Z=-9.0 \tag{A.5}$$

$$A=\begin{bmatrix}0 & 3.3 & 0\\ 3.3 & -6.7 & 3.3\\ 0 & 3.3 & 0\end{bmatrix}, B=\begin{bmatrix}0 & 9.1 & 0\\ 9.1 & -0.9 & 9.1\\ 0 & 9.1 & 0\end{bmatrix}, Z=-9.0 \tag{A.6}$$



(a) (b)

Fig. A.5 Test results using templates from (A.5) and (A.6).

A1.4 Conclusion

This chapter proposes a CNN-based defect inspection algorithm to detect defect images with regular pattern such as color filter or web materials images. Genetic algorithm is used to learn templates of CNN, which can perform defect inspection for color filter images. Experimental results show that the proposed algorithm is a promising method to detect the defects of color filter images.

References

[1] L. O. Chua and L. Yang. "Cellular neural networks: theory," *IEEE Trans. on Circuits and Systems*, vol. 35, no. 10, pp. 1257-1272, 1988.

[2] L. O. Chua and L. Yang. "Cellular neural networks: applications," *IEEE Trans. on Circuits and Systems*, vol. 35, no. 10, pp. 1273-1290, 1988.

[3] J. Hámori and T. Roska, "Receptive field atlas of the retinotopic visual pathway and some other sensory organs using dynamic cellular network models," *Analogical and neural computing laboratory*, MTA-SZTAKI, DNS-8-2000, Budapest.

[4] Szatmári, D. Bálya, G. Tímár, C. Rekeczky, and T. Roska, "Multi-channel spatio-temporal topographic processing for visual search and navigation," *Microtechnologies for the New Millenium, SPIE,* paper ID: 5119-38, Gran Canaria, Spain, May, 2003.

[5] T. Yang, L. B. Yang, C. W. Wu, L. O. Chua, "Fuzzy cellular neural networks: applications", *Proc. of CNNA-96*, pp. 225-230.

[6] T. Yang, L. B. Yang, "Fuzzy cellular neural network: a new paradigm for image processing", *International Journal of Circuit Theory and Applications*, vol. 25(6), pp. 469-481, 1997.

[7] M. Balsi and F. Voci, "Implementation of fuzzy rule based image processing on the CNN universal machine," *European Conference on Circuits Theory and Design (ECCTD)*, pp. 1167-1170, 1999.

[8] M. Balsi and F. Voci, "Fuzzy reasoning for the design of CNN-based image processing systems," *IEEE symposium on Circuits and System (ISCAS 2000)*, pp. 405-408, 2000.

[9] F. Colodro and A. Torralba, "Cellular neuro-fuzzy networks (CNFNs) a new class of cellular networks", *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems*, vol. 1, pp. 8-11, 1996.

[10] Cs. Rekeczky, T. Roska, and A. Ushida, "CNN-based difference-controlled adaptive nonlinear image filters," *International Journal of Circuit Theory and Applications*, vol. 26, pp. 375-423, July-August, 1998.

[11] Cs. Rekeczky, Á.Tahy, Z.Végh, and T. Roska, "CNN based spatio-temporal nonlinear filtering and endocardial boundary detection in echocardiography," *International Journal of Circuit Theory and Applications*, vol. 27, pp. 171-207, January-February, 1999.

[12] J. A. Nossek, "Design and learning with cellular neural networks," *International Journal of Circuit Theory and Applications*, no.24, pp. 15-24, 1996.

[13] Zarandy, "The art of CNN template design," *International Journal of Circuit Theory and Applications*, no. 27, pp. 5-23, 1999.

[14] T. Kozek, T. Roska, and L. O. Chua, "Genetic algorithm for CNN template learning," *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 40, no. 6, pp. 392-402, 1993.

[15] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[16] C. T. Lin, *Neural Fuzzy Control Systems with Structure and Parameter Learning*, New York: World Scientific, 1994.

[17] C. T. Lin and C. S. G. Lee, *Neural Fuzzy Systems: A Neural-Fuzzy Synergism to Intelligent Systems*. Englewood Cliffs, NJ: Prentice-Hell, 1996.

[18] R. Jang, C.T. Sun and E. Mizutani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, NJ: Prentice-Hall, 1997.

[19] D. Nauck, F. Klawonn and R. Kruse, *Foundations of Neuro-Fuzzy Systems*, John Wiley, 1997.

[20] L. Wang and R. Langari, "Building Sugeno-type models using fuzzy discretization and orthogonal parameter estimation techniques," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 454-458, Nov., 1995.

[21] E. H. Ruspini, "Recent development in fuzzy clustering," *Fuzzy Set and Possibility Theory*, pp. 113-147, New York: North Holland, 1982.

[22] C. T. Sun and J. S. Jang, "A neuro-fuzzy classifier and its applications," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, San Francisco, CA, vol. I, pp. 94-98, Mar. 1993.

[23] C. F. Juang and C. T. Lin, "An on-line self constructing neural fuzzy inference network and its applications," *IEEE Trans. Fuzzy System*, vol. 6, no.1, pp. 12-32, 1998.

[24] Hyvärinen, J. Karhunen, and E. Oja, *Independent Component Analysis*, John Wiley & Sons, Inc., 2001.

[25] Jutten and J. Herault, "Independent components analysis (ICA) versus principal components analysis," *Signal Processing IV: Theories and Applications, Proc. EUSIPCO-88*, J. Lacoume et al, Eds. Amsterdam, The Netherlands: Elsevier, pp. 643–646, 1988.

[26] J. C. Bezdek, J. Keller, R. Krisnapuram, and N. R. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Boston, MA: Kluwer, 1999.

[27] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York: Wiley, 2001.

[28] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler, *Fuzzy Cluster Analysis*. New York: Wiley, 1999.

[29] G. J. McLachlan and T. Krishnan, *The EM Algorithms and Extensions*. New York: Wiley, 1997.

[30] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[31] L. X. Wang, *Adaptive Fuzzy Systems and Control*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[32] T. W. Lee, M. S. Lewicki, and T. J. Sejnowski, " ICA mixture models for unsupervised classification of non-Gaussian classes and automatic context switching in blind signal separation," *IEEE Trans. on pattern analysis and machine intelligence*, vol. 22, no. 10, October, 2000.

[33] T. W. Lee, M. S. Lewicki, and T.J. Sejnowski, " ICA mixture models for unsupervised and automatic context switching," *Proc. Int'l Workshop ICA*, pp. 209-214, 1999.

[34] P. Werbos, "Beyond regression: new tools for prediction and analysis in the behavior sciences," Ph.D. dissertation, Harvard Univ., Cambridge, MA, Aug. 1974.

[35] R. J. Williams and D. Zipser, "A learning algorithm for continually running recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, 1989.

[36] B. A. Pearlmutter, “Gradient calculations for dynamic recurrent neural networks: a survey,” *IEEE Trans. Neural Networks*, vol. 6, pp. 1212–1228, 1995.

[37] S. W. Pich´e, “Steepest descent algorithms for neural-network controllers and filters,” *IEEE Trans. Neural Networks*, vol. 5, pp. 198–212, 1994.

[38] V. Preciado, D. Guinea, R. Montufar, and J. Vicente, “Real-time inspection of metal laminates by means of CNN’s,” *Proc. of SPIE*, vol. 4301, no. 39, 2001.

[39] D. Guinea, A. Gordaliza, J. Vicente, and M. C. García-Alegre, “CNN based visual processing for industrial inspection,” *Proc. of SPIE*, vol. 3966, no. 45, 2000.

[40] C. L. Chang and C. T. Lin, “CNN-based defect inspection in images with regular pattern,” *The 16th European Conference on Circuit Theory and Design (ECCTD)*, pp. I221-I224, 2003.

[41] R. Perfetti and L. Terzoli, “Analogic CNN algorithms for textile applications,” *International Journal of Circuit Theory and Applications*, no. 28, pp. 77-85, 2000.

[42] C. T. Lin and Y. C. Lu, “A neural fuzzy system with fuzzy supervised learning,” *IEEE Transactions on Systems, Man, and Cybernetics --- Part B: Cybernetics*, vol. 26, no. 5, pp. 744-763, 1996.

[43] L. Raffo, S.P. Sabatini, and G.M. Bisio, “A reconfigurable architecture mapping multilayer CNN paradigms,” *Proceedings of IEEE Int. Workshop on Cellular Neural Networks and Their Applications (CNNA’94)*, pp. 393–398, Rome, 1994.

[44] G. Liu and S. Oe, “Texture image segmentation method based on multilayer CNN,” *12th IEEE Int. Conference on Tools with Artificial Intelligence (ICTAI’00)* , pp. 147–150, Canada, 2000.

[45] S.A. Chen, J.F. Chung, S.F. Liang, and C.T. Lin, “Hybrid-order texture boundary detection based on CNN circuit models,” *Proceedings of IEEE Int. Workshop on BioMedical Circuits and Systems (BioCAS’04)*, Singapore, 2004.

[46] Chin-Teng Lin, Wen-Chang Cheng, and S. F. Liang, “An On-line ICA-Mixture-Model-Based Self-Constructing Neural Fuzzy Network,” *IEEE Transactions on Circuits and Systems Part I: Regular paper,* vol. 52, no. 1, pp. 207-221, 2005.

List of Publication

著作目錄

姓名: 張俊隆(Chun-Lung Chang)

已刊登或被接受之期刊論文：

[1] Chin-Teng Lin, Chun-Lung Chang, and Jen-Feng Chung, "New horizon for CNN with fuzzy paradigms for Multimedia," *IEEE Circuits and Systems Magazine,* Vol. 5, No. 2, pp. 20-35, 2nd Quarter, 2005.

[2] Chin-Teng Lin, Chun-Lung Chang, and Wen-Chang Cheng, "A Recurrent Fuzzy Cellular Neural Network System with Automatic Structure and Template Learning," *IEEE Transactions on Circuits and Systems Part I: Regular Papers,* Vol. 51, No. 5, pp. 1024-1035, May 2004.

[3] Chin-Teng Lin and Chun-Lung Chang, "A Recurrent Fuzzy Coupled Cellular Neural Network System with Automatic Structure and Template Learning," Accepted to *IEEE Transactions on Circuits and Systems Part II: Express Briefs.*

研討會論文：

[1] Chin-Teng Lin, and Chun-Lung Chang, "A fuzzy-logic-based cellular neural network integrated system," *The 8th IEEE International Biannual Workshop on Cellular Neural Networks and their Applications (CNNA),* Hungary, 2004.

[2] Chin-Teng Lin, and Chun-Lung Chang, "A neural-fuzzy approach for defect inspection in images with regular pattern," *Conference of Chinese Automatic Control*, 2004.

[3] Chun-Lung Chang, and Chin-Teng Lin. "CNN-based defect inspection in images with regular pattern," *The 16th European Conference on Circuit Theory and Design (ECCTD,* pp. I221-4, 2003.

Vita

博士候選人學經歷資料

姓名: 張俊隆

性別: 男

生日: 中華民國 56 年 3 月 4 日

籍貫: 台灣省台北縣

論文題目: 中文: 模糊細胞神經網路整合系統

英文: A Fuzzy Cellular Neural Network Integrated System

學歷:

1. 民國 79 年 6 月 逢甲大學自動控制工程系畢業。
2. 民國 81 年 6 月 國立清華大學動力機械工程研究所畢業。
3. 民國 88 年 9 月 國立交通大學電機及控制工程研究所博士班。

經歷:

1. 民國 81 年至民國 88 年 工業技術研究院機械工業研究所副研究員
2. 民國 88 年起至今 工業技術研究院機械工業研究所研究員