7-30-2013

Cellular Simultanous Recurrent Networks for Image Processing

John Keith Anderson

CELLULAR SIMULTANEOUS RECURRENT NETWORKS FOR IMAGE PROCESSING

by

John Keith Anderson

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Engineering

The University of Memphis

August 2013

ACKNOWLEDGEMENTS

I would like to express my deepest thanks to my major advisor, Dr. Khan Iftekharuddin, for his unwavering commitment to this effort. His knowledge of image processing and computer vision set the foundation for this work. He has patiently guided me through funding changes, a change in research topic, and several personal and family illnesses.

My thanks to all my committee members: Dr. Micheal Bartz, Dr. Eddie Jacobs, Dr. Khan Iftekharuddin, Dr. Robert Kozma, and Dr. Mohammed Yeasin. Thanks for your insight and support. A special thanks for your thoughts and prayers on behalf of my daughter, Lauren, throughout her kidney failure and subsequent transplant.

I would also like to thank Dr. Roman Ilin for sharing his maze traversal application and his subsequent help in understanding its code. Thanks also to Drs. Paul and Luda Werbos for sharing their insight into the background and history of cellular simultaneous recurrent networks.

I would like to thank my family for their patience, love and support. Special thanks to my loving wife and best friend, Lisa. Thanks so much for seeing me through all the difficulties involved in an undertaking of this nature. Without you, I never would have made it through.

Finally, I thank God our creator, sustainer, and savior. You laid the foundations of the universe and know all its hidden secrets that we as humans struggle so hard to understand. I am in awe of your creation and thank you so much for helping me to better understand a small part of it.

ABSTRACT

Anderson, John Keith Anderson. PhD. The University of Memphis. August 2013. Cellular Simultaneous Recurrent Networks for Image Processing. Co-major Professors: Khan M. Iftekharuddin and Eddie L. Jacobs.

Artificial neural networks are inspired by the abilities of humans and animals to learn and adapt. Feed-forward networks are both fast and powerful, and are particularly useful for statistical pattern recognition. These networks are inspired by portions of the brain such as the visual cortex. However, feed-forward networks have been shown inadequate for complex applications such as long-term optimization, reinforced learning and image processing. Cellular Neural Networks (CNNs) are a type of recurrent network which have been used extensively for image processing. CNNs have shown limited success solving problems which involve topological relationships. Such problems include geometric transformations such as affine transformation and image registration. The Cellular Simultaneous Recurrent Network (CSRN) has been exploited to solve the 2D maze traversal problem, which is a long-term optimization problem with similar topological relations. From its inception, it has been speculated that the CSRN may have important implications in image processing. However, to date, very little work has been done to study CSRNs for image processing tasks. In this work, we investigate CSRNs for image processing. We propose a novel, generalized architecture for the CSRN suitable for generic image processing tasks. This architecture includes the use of sub-image processing which greatly improves the efficacy of CSRNs for image processing. We demonstrate the application of the CSRN with this generalized architecture across a variety of image processing problems including pixel level transformations, filtering, and geometric transformations. Results are evaluated and compared with standard

MATLAB® functions. To better understand the inner workings of the CSRN we investigate the use of various CSRN cores including: 1) the original Generalized Multi-Layered Perceptron (GMLP) core used by Pang and Werbos to solve the 2D maze traversal problem, 2) the Elman Simultaneous Recurrent Network (ESRN), and 3) a novel ESRN core with multi-layered feedback. We compare the functionality of these cores in image processing applications. Further, we introduce the application of the unscented Kalman filter (UKF) for training of the CSRN. Results are compared with the standard Extended Kalman Filter (EKF) training method. Finally, implications of current findings and proposed research directions are presented.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF ALGORITHMS

1 INTRODUCTION

1.1 Motivation

Artificial neural networks (ANNs) are inspired by the abilities of humans and animals to learn and adapt. Feed-forward networks are both fast and powerful, and are particularly useful for statistical pattern recognition. These networks are inspired by working principles of brain sensory processing areas such as the visual cortex. However, feed-forward networks have been shown inadequate for complex applications such as long-term optimization, reinforced learning and image processing.

The use of Cellular Neural Networks (CNNs) for image processing has been widely accepted, from both application and biologics points of view. A simple search using Google Scholar shows that cellular networks account for approximately 13.4% of all image processing publications [104]. The similarity of the cellular structure of the CNNs to that of the human retina and the neural structure of the visual cortex, make it biologically attractive.

1.2 Literature Review

1.2.1 Image Processing

Image Processing (IP) is a broad and well studied subject. A topic search via Google Scholar for 'image processing' yields over 2 million publications [105]. A complete literature review of IP techniques lies outside the scope of this work. There are many excellent texts that cover general IP methods. Jain [49] is an older text that is still popular today due to its treatment of IP fundamentals. Gonzalez e*t al.* [32]covers digital

image processing techniques and is a used in many undergraduate and graduate level courses.

1.2.2 Cellular Neural Networks

Application of feed-forward neural networks to IP is impractical do to the size of the networks required to handle the data in typical images. As mentioned above, Cellular Neural Networks (CNNs) have been used extensively to perform IP tasks. CNNs were introduced by Chua *et al* [13][14]. Chua and Roska [11] provide a thorough coverage of the fundamentals of the CNN and its application to visual computing. Yang [101] gives in depth coverage of the CNN's use for IP, including detailed templates for performing a multitude of specific IP tasks.

Although CNNs have been referred to as '*universal image processors'* they have shown limited success with geometric transformations such as affine transformations and image registration. CNNs are capable of performing fractional and single pixel translation [17][26]. By extension, the CNN can approximate rotation by first decomposing the rotation into multiple, single-pixel translations and one, fractional-pixel translation, then applying the CNN to the image once for each single pixel or fractional translation. CNNs have not been successfully applied to image registration.

Most CNN IP applications do not determine weight values through a learning process. The weights are typically computed mathematically or experimentally, and are fixed based on the goals of the application. Some recent works have been effective in training of CNNs for specific applications. Takizawa *et al*. [85] utilize back-propagation to train CNNs to perform loss-less image coding. Aein *et al*. [1] use a modified form of back-propagation to train CNNs for modeling of mechanical vibration.

The need still exists for a general image processor capable of learning to performing basic IP tasks, as well as more complicated tasks posed by geometric transformations.

1.2.3 Cellular Simultaneous Recurrent Networks

A type of cellular network that is capable of both long-term optimization and learning is the Cellular Simultaneous Recurrent Network (CSRN).

The CSRN was developed by Pang and Werbos in their efforts to show that neural networks (NNs) could be successfully applied to optimization [68][69]. To demonstrate this ability, they applied NNs to the maze traversal problem. Their solution utilized simultaneous recurrent networks (SRNs) in a cellular structure, effectively combining SRNs with Cellular Neural Networks (CNNs). They referred to the network as a Cellular SRN, which has been re-coined CSRN.

Wuncsh reexamined the generalized maze problem from a mathematical perspective, and showed that a closed form solution exists [98]. He also obtained a worst case convergence time. His results were consistent with Pang *et al.*'s conclusion that an SRN-based critic is necessary for the solution of the maze problem.

To date, the CSRN has seen limited application. This was primarily due to poor training times. With the application of newer training methods, such a parameter estimation via extended Kalman filters [37] and particle swarm optimization (PSO)[53], CSRNs have recently gained traction.

Ilin *et al*. applied parameter estimation via extended Kalman filter (EKF) to the training of CSRNs in the maze traversal application [47]. Their results indicate that the EKF method is superior in both speed and accuracy to the backpropagation through time

(BPTT) method. They also investigated the use of CSRNs to a subset of the connectedness problem [45][46], which was the first application of CSRNs to an IP task.

We demonstrated, for the first time in literature, simple registration of low-complexity, binary and gray scale images subjected to in-plane rotation using CSRN [3][5]. These works are presented in detail in Chapter 6.

Ren *et al*. exploited the CSRNs ability to perform time-series prediction in an application performing pose invariant facial recognition in image sequences [74][75][76].

Grant *et al*. applied CSRNs adapted to the identification and prediction of buss voltage dynamics in practical power systems [34]. That work utilized a form of PSO called Small Population Particle Swarm Optimization (SPPSO)[18] to train CSRNs.

Recently, Rice *et al*. have explored general purpose graphical processing unit (GPGPU) cluster acceleration for the maze traversal application trained via Ilin *et al*.'s EKF method [77] and Ren *et al*.'s pose invariant facial recognition application [78]. These works demonstrate improved training times, allowing the applications to be scaled for practical application. Rice *et al*. also implemented a decoupled EKF training algorithm which resulted in improved training times over Ilin's standard EKF method.

1.3 Goals

From its inception, it has been speculated that the CSRN would have important implications in image processing. However, to date, very little work has been done to adapt CSRNs to IP tasks. In this work, we investigate CSRNs for image processing. The specific goals of this work are discussed below.

1.3.1 Goal 1

Hypothesis 1: *The generalized CSRN architecture, presented in Section 4.3, is capable of performing general image processing.*

Goal 1: *To demonstrate the ability of the generalized CSRN architecture to perform variety of image processing tasks.*

We adapt the CSRN to perform a variety of tasks, taken from a range of basic problem groups encountered in image processing. These include pixel level transformations, filtering and geometric transformations. Specifically, these tasks include grey-scale to binary conversion, low-pass filtering, affine transformation and registration. Once again, we demonstrate the CSRN's ability to perform these tasks on both small binary test images and larger, grey-scale, facial images. Convergence, training times, forward computation times and transformation accuracy are examined.

1.3.2 Goal 2

Hypothesis 2: *The CSRN can learn to perform affine transformations on binary and gray-scale images.*

Goal 2: *To demonstrate the ability of CSRNs to perform affine transformations on images.*

In pursuit of this goal, we investigate the CSRN to perform translation, rotation and scaling transformations. As proof of concept we demonstrate the CSRN's ability to perform these affine transformations on small, 15x15, binary test images. After introduction of a sub-image processing technique, which allows the CSRN to operate on larger more practical images, we examine the CSRN's ability to perform the above mentioned transformations on larger, grey-scale, facial images. Convergence, training times, forward computation times and transformation accuracy are examined.

In addition, we examine several core networks for the CSRN, such as the generalized multi-layered perceptron (GMLP), the Elman simultaneous recurrent network (ESRN), and a novel extension of the ESRN to include multi-layered feedback (ESRN/mlf). Comparisons are made using the affine rotation application described above. Convergence, transformation, training times, forward computation times and transformation accuracy are compared.

1.3.3 Goal 3

Hypothesis 3: *The CSRN can learn to register images under affine transformation without prior knowledge of transformation parameters.*

Goal 3: *To demonstrate the ability of the CSRN to learn to register images under affine transformation without prior knowledge of transformation parameters.*

We adapt the CSRN to perform image registration on images which have been transformed via translation, rotation and scaling transformations. As proof of concept we demonstrate the CSRN's ability to perform registration on small, 15x15, binary test images. We also demonstrate the CSRN's ability to perform registration on larger, grey-scale, facial images. Convergence, training times, forward computation times and registration accuracy are examined.

1.3.4 Goal 4

Hypothesis 4: *The CSRN can be trained via parameter estimation with an unscented Kalman filter (UKF).*

Goal 4: *To demonstrate training of the CSRN via parameter estimation with an UKF and evaluate the efficacy of this training method.*

We implement a training method which uses parameter estimation via UKF. This method is tested using the affine rotation application and comparisons are made with Ilin

et al.'s standard EKF method [47]. Convergence, training times, and transformation accuracy are examined.

1.4 Organization

Chapter 2 contains background on neural computation. Chapter 3 introduces the CSRN and examines its architecture and core network. Learning in CSRN's is discussed, as are training methods and current applications. Chapters 4 thru 8 present the contributions of this work. Chapter 4 discusses image processing with CSRNs. After discussing the adaptation of CSRN to image processing, a sub-image processing technique is present which allows application of CSRN to larger more realistic images. Next we present a generalized architecture suitable for generic image processing tasks. We then demonstrate the CSRN's ability to both learn and perform pixel-level transformations, filtering, and geometric transformations. Chapter 5 investigates the application of CSRNs to affine transformation of images and Chapter 6 the registration of images subjected to the same. In Chapter 7 we explore the internal workings of the CSRN. After investigating the selection of internal CSRN parameters, we examine and compare three core networks for the CSRN, the GMLP, ESRN, and ESRNmlf cores. Chapter 8 examines methods for training the CSRN. Various methods are discussed and a new method based on parameter estimation via UKF is presented. Chapter 9 provides a summary of results, discusses contributions and future work.

2 BACKGROUND

2.1 Artificial Neural Networks

Simon Haykin defines an artificial neural network as follows [38].

A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

While these "*simple processing units*" in Haykin's definition can be linear or non-linear, they have historically been implemented using the McCulloch and Pitts model of a neuron [58], which utilizes a non-linear activation function, thus making ANNs applicable to non-linear computations and further increasing their resemblance to the brain. As in Haykin's text, we will refer to ANNs simply as neural networks (NNs).

2.2 Basic Neuron Model

Figure 2.1 shows a common depiction of a non-linear model of a McCulloch & Pitts neuron. The inputs to the neuron are labeled $x_1 - x_m$. Note that x_0 is a fixed input, referred to as the bias input. It is set to 1 and is used in conjunction with its corresponding weight to implement an external bias to the neuron. Each input is multiplied by a corresponding synaptic weight, and these 'weighted' inputs are summed. The output of the summing junction, v_i, is processed by the activation function, $f( )$, to produce the neuron's output, y_i.

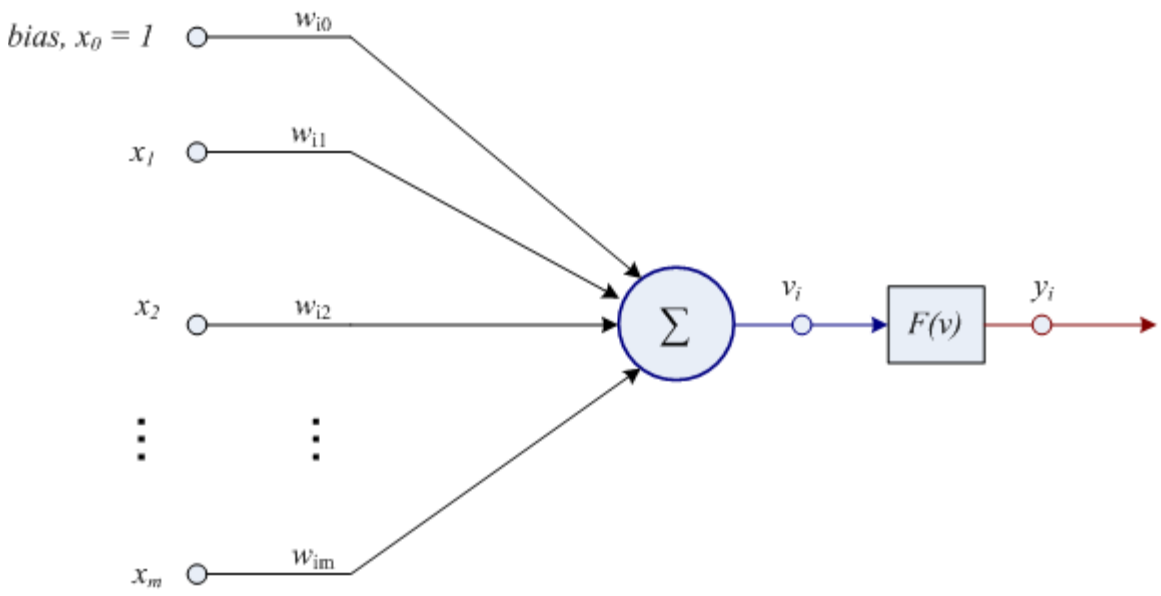


Figure 2.1: Non-linear model of a neuron. X_0 is a bias input. X_1 - X_m are external inputs. $F(v)$ represents the activation function. Y_i is the output. Synaptic weights are represented by w_{im}, where i is the current cell and m is the input number.

Mathematically, the model can be expressed as

$$v_i = \sum_{i=0}^{m} w_{ij} \cdot x_i \tag{2.1}$$

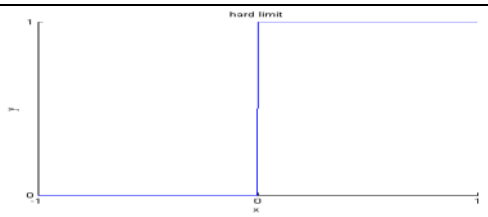
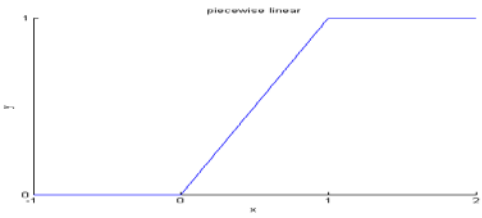
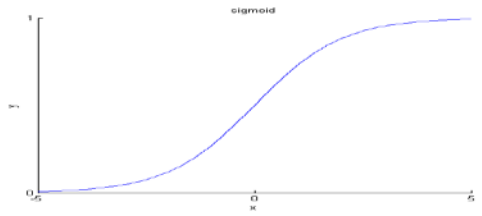
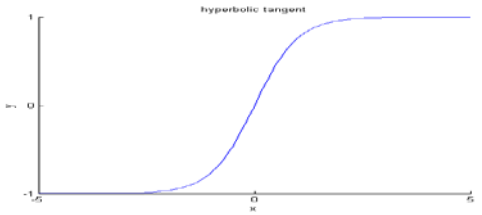
and

$$y_i = f(v_i) \tag{2.2}$$

.

Table 1.1 identifies four commonly used activation functions along with their corresponding mathematical definitions. In this work we use the *tanh* activation function exclusively.

Table 2.1: Commonly used activation functions

Activation Function	Definition
 hard limit	$y = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$
 piecewise linear	$y = \begin{cases} 1, & x \geq \frac{1}{2} \\ x, & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ 0, & x \leq -\frac{1}{2} \end{cases}$
 sigmoid	$y = \frac{1}{1+\exp(-x)}$
 hyperbolic tangent	$y = \tanh(x)$

2.3 Neural Network Architectures

NNs can be divided into two major classes: feed-forward and recurrent. If we view a NN as a signal-flow or directed graph, then recurrent networks have loops or feedback, while feed-forward networks do not. In the discussion below, we present architectures for several types of commonly used NNs.

2.3.1 Feed-Forward Networks

2.3.1.1 Single-layered Feed-forward Networks

Figure 2.2 shows the architecture for a single-layered feed-forward (SLFF) network. The network inputs, [*x1*, …*x3*], are grouped together and are commonly referred to as the input layer. These nodes do not contain active neurons, but simply serve as an entry point for the inputs to the network. The active neurons (shown as circles) are grouped into a single layer, which we refer to as the output layer. Note that the outputs of the neurons do not feed back to the inputs, other neurons, or themselves. All computations occur only in the forward direction, thus the term "feed-forward."

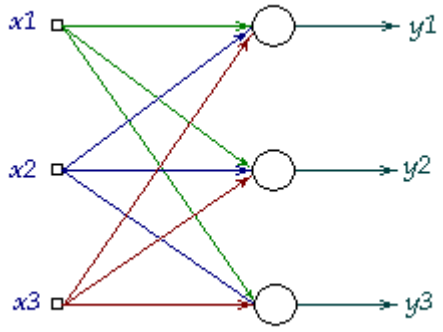


Figure 2.2: Single-layered feed-forward network with three external inputs, *x1-x3*, three active neurons in the output layer and three outputs, *y1-y3*.

SLFF networks include the perceptron, one of the earliest known NNs, introduced in 1957 by Rosenblatt [79]. The perceptron was the first computer to use NNs and to learn by experience. It was capable of classifying its inputs into two classes. The perceptron, and SLFFs in general, are limited to linearly separable problems, a fact that was known as early as McCulloch and Pitts, but was formalized by Minsky and Pappert [61]. For example, SLFFs cannot approximate the XOR function.

2.3.1.2 Multi-layered Feed-forward Networks

Figure 2.3 shows the architecture for a multi-layered feed-forward (MLFF) network. The nodes of the network are once again grouped into layers. In addition to the input and output layers, there is also one or more hidden layers. The nodes/units within this layer are referred to as "hidden" because their outputs are not externally visible. Note, once again, that there is no "feed-back" of the outputs, thus preserving the "feed-forward" nature of the network.

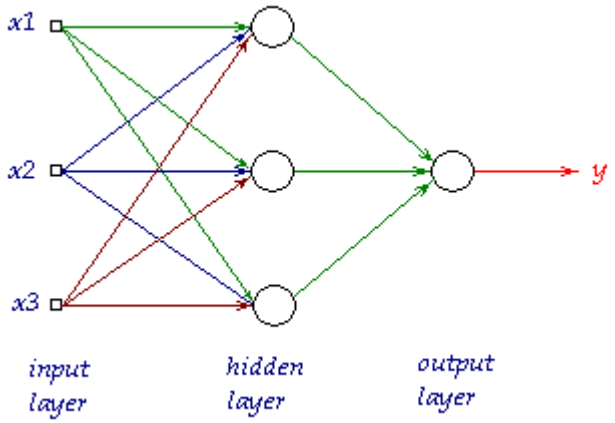


Figure 2.3: Multi-layered feed-forward network with three external inputs, *x1-x3*, three neurons in the hidden layer, one neuron in the output layer, and a single output, *y*.

MLFFs, also known as Multi-layered Perceptrons (MLPs) have been researched extensively [61][62][81] and are among the most common NNs in use today. MLPs have been shown to be universal function approximators for adequately smooth functions [62] and are capable of solving non-linearly separable problems such as the XOR function [58][81].

A major difficulty faced by MLPs was the computation of the required derivatives [61]. These computations were prohibitive and lead to a decreased interest in the MLP.

In 1974 a tractable algorithm for computation of derivatives in complex feed-forward differentiable systems emerged, when Paul Werbos extended his concept of ordered derivatives to describe what he termed 'the rule of backpropagation [95]'. These same concepts were later applied directly to MLPs by Rumelhart *et al* [81].

2.3.1.3 Generalized Multi-layered Perceptron

Until now we have discussed only NNs arranged in layers. In general this need not be the case. Figure 2.4 shows the architecture of a generalized multi-layered perceptron (GMLP) [69]. In this architecture, neurons are arranged in a single layer. The 1^{st} m nodes are input nodes, and are not active neurons. The input nodes are followed by the hidden nodes, which are in turn followed by the output nodes. The difference between this architecture and the SLFF or MLFF network is that the output of all active neurons is fed forward to *all* proceeding neurons. This architecture produces results similar to that of the MLP and the GMLP can be applied in any application where the MLP is used. The parameters for the GMLP are slightly different. The user must specify only the number of active neurons without having to specify the number of hidden layers as a parameter.

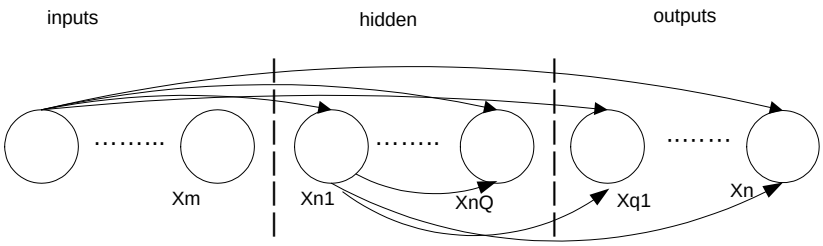


Figure 2.4: Generalized Multi-layered Perceptron with *m* inputs, *Q* hidden neurons and *n* output neurons.

2.3.2 Recurrent Networks

There are several architectures available for recurrent networks. In fact, the only requirement of a recurrent network over a feed-forward network is the addition of at least one feed-back loop. With this in mind, any of the above feed-forward architectures can be used to create a recurrent network, simply by feeding one or more outputs back to the network as inputs. Regardless of the base architecture, the presence of feedback in the network has a profound effect on the learning capability and performance of the network [38]. Indeed, the presence of recurrency in the brain provides a biological basis for the use of recurrent networks [9][57].

2.3.2.1 Time-delayed Recurrent Network

Figure 2.5 shows a time-delayed recurrent network (TLRN). The first two inputs are external inputs. In addition to these, the outputs of each active neuron are feed back, thru a time delay element, as inputs to the network. Note that the first neuron is a hidden node, because its output is only used internally. TLRNs are used extensively for time-series prediction.

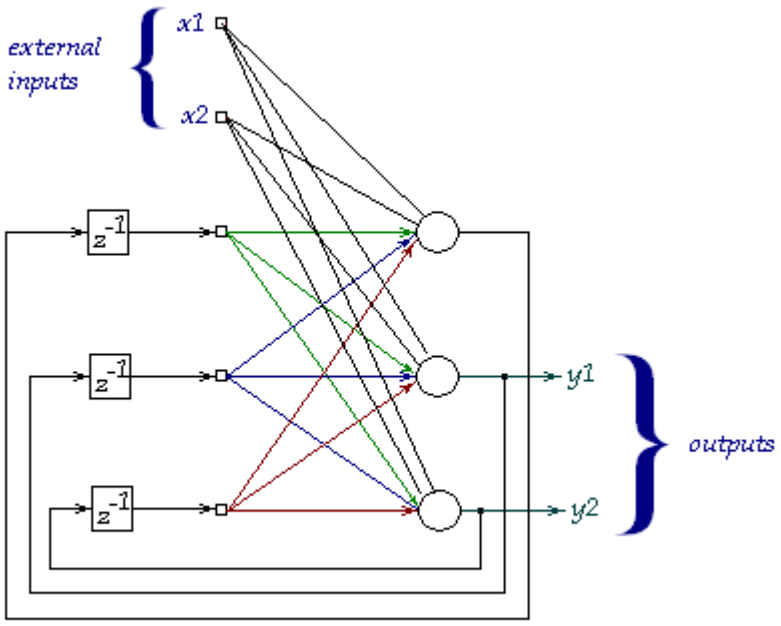


Figure 2.5: Time delayed recurrent network with two external inputs, *x1* and *x2* , three recurrent inputs, one hidden neuron, two output neurons, and two outputs, *y1* and *y2*.

2.3.2.2 Simultaneous Recurrent Networks

Figure 2.6 shows the architecture of a simultaneous recurrent network (SRN). As in other recurrent networks, feedback is present in the SRN. Any feed-forward network can be used as the core of the SRN. Note the lack of any time delay elements. SRNs are characterized by their use of two time scales. The external time scale is slower and is used to present inputs to and read outputs from the network. The internal time scale is much faster and allows the system to iterate the forward computation, allowing the outputs to converge to their final values before they are used externally. From the perspective of the external time scale, the recurrency is almost instantaneous yielding the '*simultaneous*' descriptor for this network. This "settling" response is similar to that of any dynamic system. The recurrency found in SRNs is very similar to that found in parts

of the human brain. Neurobiologists have even proposed the presents of multiple time scales in the learning process [93].

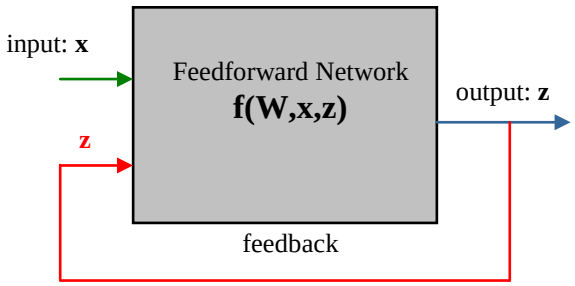


Figure 2.6: Architecture of a simultaneous recurrent network. In this figure, ***f*** is the feed-forward function, ***W*** is the weight matrix, ***x*** is the network input, and ***z*** is the network output.

Previous research has shown SRNs to be more powerful than MLPs [97]. In fact, functions randomly generated by MLPs can always be learned by SRNs, but the opposite is not true [15][16] [97]. SRNs are used successfully in time-series prediction and intelligent control applications.

2.3.2.3 Cellular Neural Networks

Another type of NN is the cellular neural network (CNN) which consists of identical elements, arranged in some sort of geometry. Figure 2.7 shows the cellular structure of the CNN introduced by Chua [14]. Each element or cell of the network can be as simple as a single artificial neuron or more complex, such as a MLP. Chua's original CNN utilized a hardware implementation of a single McCulloch and Pitts neuron, realized via an electrical circuit.

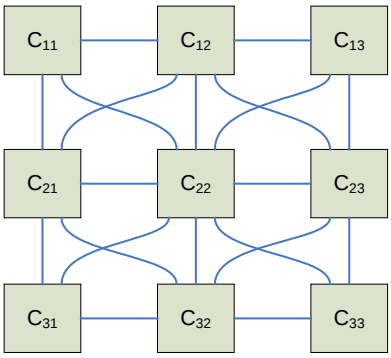


Figure 2.7: A typical 3 x 3 CNN cellular architecture. Interactions between cells are depicted as connections.

Figure 2.8 shows a neural network depiction of Chua's cell circuit. The McCulloch and Pitts neuron model is immediately evident. While Chua's implementation uses the piecewise linear activation function, any of the standard activation functions from Table 2.1 can be used.

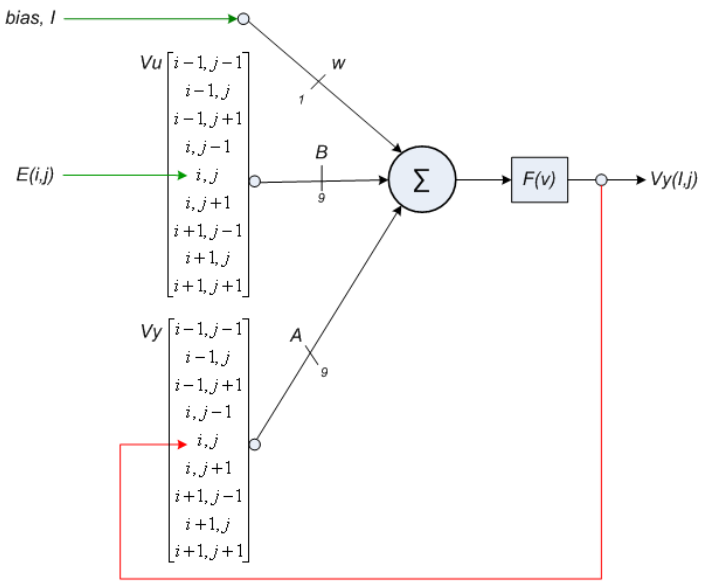


Figure 2.8: A neural network depiction of a CNN cell circuit.

Along with a bias input, each cell has two sets of inputs. Each set of inputs are presented to the network as a matrix, $\boldsymbol{V_u}$ and $\boldsymbol{V_y}$. The $\boldsymbol{V_u}$ matrix contains the current cell's external input along with the inputs of the surrounding eight neighboring cells. The $\boldsymbol{V_y}$ matrix contains the output of the current cell along with the outputs of the surrounding eight neighboring cells.

The cell's weights are stored in two corresponding matrices, which Chua refers to as "*cloning templates"*. The ***A*** matrix is referred to as the "*control template*" while the ***B*** matrix is referred to as the "*feedback template*". A defining feature of the CNN is that the same weights are used for all cells. This "weight sharing" can significantly decrease the number of required weights, which, in turn, can significantly decrease the time needed to train the network. The CNN can be configured as either a feed-forward or recurrent network, depending on how the weight matrix, ***A*** is used. If the ***A*** matrix is set to zero, then the network is strictly a feed forward network, otherwise it is recurrent as shown in Fig. 2.8.

The symmetry of CNNs is useful in solving problems that contain a similar type of inherent geometry. The similarity between the CNN's architecture and a typical image is immediately evident, and indeed, CNNs have been widely used in both image processing and pattern recognition applications [13][101]. Most CNN image processing applications do not determine weight values through a learning process. The weights are typically computed mathematically or experimentally, and are fixed based on the goals of the application. Many of the cloning templates resemble spatial windowing filters seen in standard image processing texts [101].

3 CELLULAR SIMULTANEOUS RECURRENT NETWORKS

3.1 Introduction to CSRNs

In a seminal work Pang *et al*. introduced CSRNs [69]. The authors showed that CSRN could be successfully applied to optimization problems by approximating a solution to the Bellman's equation [97]. To demonstrate this ability, they applied CSRNs to the maze traversal problem as discussed in Section 3.5.1. Furthermore, they showed that MLPs were incapable of solving such maze traversal problem [96]. To solve the maze traversal problem they utilized SRN networks in a cellular structure, effectively combining SRNs with CNNs. They referred to the network as a Cellular SRN, which has been re-coined as CSRN.

3.2 Biological Basis

From a computational standpoint the human brain is a highly complex, nonlinear, parallel processing device. It is composed of billions of neurons and organizes neurons to perform specific functions. The number and organization of neurons to perform a specific function, as well as the synaptic weights between neurons, are learned through experience [79]. Neuroscientists speculate that reinforced learning takes place in the hippocampus of the brain. Populations of neighboring neurons form cell ensembles, which form the basic building block for the entire system [9]. These ensembles are interconnected in a cellular structure. Neurobiologist have long understood that local recurrency plays a critical role in the higher part of the human brain and evidence indicates that the brain is highly recurrent [9][57]. Cellular simultaneous recurrent

networks (CSRNs) mimic both the recurrency and cellular structure of the brain. The reinforced learning in the CSRN is quite similar to that of the human brain.

3.3 Architecture

Figure 3.1 shows the cellular structure of the CSRN. Note the cellular structure of the network matches that of the underlying input pattern. As shown in Fig. 3.1, the outputs of each cell can be brought together to produce an overall network output. Each cell (shown as a grey box) contains an SRN core.

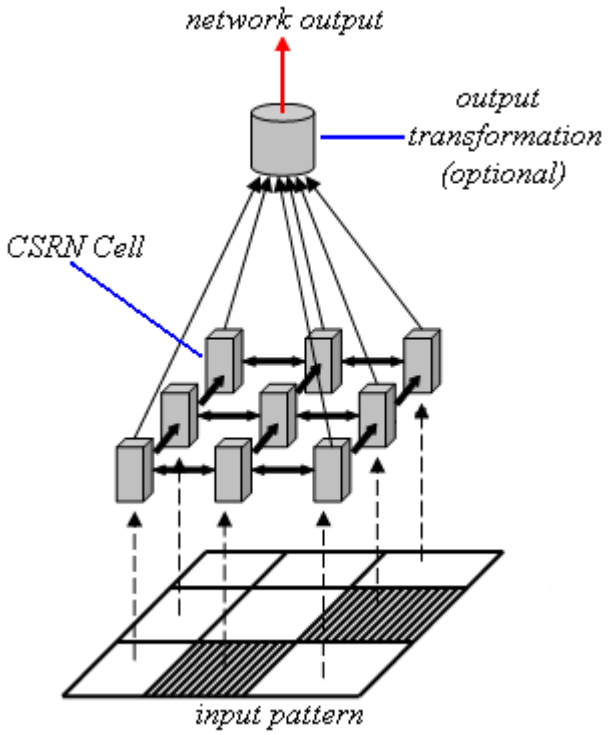


Figure 3.1: Cellular structure of the CSRN.

The core network is shown in Fig. 3.2 and is referred to as a generalized multi-layered perceptron (GMLP) core. The core consists of 17 nodes, five of which are active neurons. Nodes *n1-n12* from the left are input nodes. Node 1 is the bias input. The

network shown here has three external inputs. The four neighbor inputs come from the outputs of node 13 of the neighboring cells. This node is referred to as the *connector node*. The interconnections provided by these inputs form the cellular structure of the network. The final five inputs are self-recurrent inputs consisting of the outputs of the five active neurons fed back as inputs. Note that in the GMLP architecture depicted here, the inputs to each active node consist of the outputs from all previous nodes. For the purposes of clarity, not all interconnects between nodes have been shown. The output of node 17 is multiplied by a scaling weight, W_s, to produce the cell's ultimate output, $\hat{Y}$. CSRNs utilize weight sharing, that is, the weights for each cell are equivalent. This significantly reduces the amount of memory required to store the CSRN's weights.

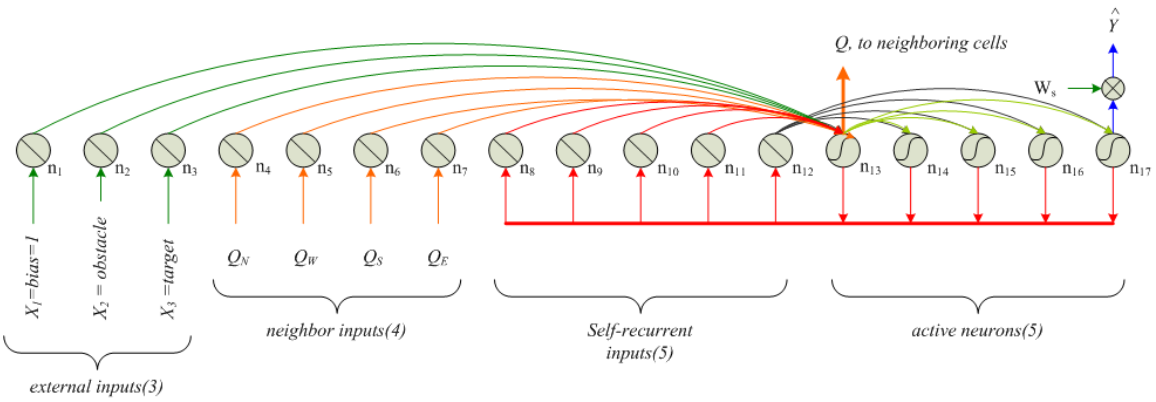


Figure 3.2: Standard GMLP core of the CSRN shown with 3 external inputs, 4 neighbor inputs, 5 recurrent inputs (12 total inputs) and 5 active nodes. Each active node receives an input from all prior nodes. Output is taken from final node, and scaled via a scaling weight. The network has a total of 17 nodes.

3.4 Learning in CSRNs

CSRNs learn by adapting their weights via supervised learning. If the CSRN output is given by, Y_i, and its corresponding target image is T_i, then the error between the two is given by,

$$e_i = T_i - Y_i. \tag{3.1}$$

The total error in the network output is given by,

$$E_i = \frac{1}{2}\sum_{i=1}^{c} e_i^2, \tag{3.2}$$

where *c* is the number of pixels in the image. For a given training set the error is as follows,

$$E = \sum_{n=1}^{N}\left[\frac{1}{2}\sum_{i=1}^{c} E_i^2\right] \tag{3.2}$$

where *N* is the number of training images. Equation (3.2) is the sum-squared error and represents the cost-function of the network. The objective of the learning process is to adjust the network's weights to minimize this cost function. With this simple cost function, the CSRN is capable of approximating much more complicated cost functions [69], including those associated with a variety of image processes tasks.

3.5 Training

In this section, we provide a brief overview of techniques used in training of CSRNs. Chapter 8 presents detailed descriptions of these techniques as well as a comparative study.

3.5.1 Back-Propagation through Time

The original CSRN developed by Pang *et al*. was trained using an extension of basic back-propagation known as back-propagation through time (BPTT)[69][92]. This method of training is fundamental to CSRNs. Prior to BPTT, no tractable method of computing the required derivatives existed. Appendix I contains a detailed study of BPTT including its mathematical derivation, and computational algorithms.

3.5.2 Parameter estimation via EKF

Parameter estimation with Kalman filters has been effectively applied to the training of NN [37]. Ilin *et al*. apply parameter estimation using an extended Kalman filter (EKF) to the training of CSRNs adapted for the solution of the maze traversal problem [47][48]. In this method, the required Jacobian is computed via BPTT, and the weights are adapted via EKF.

3.5.3 Parameter estimation via Decoupled EKF

In a recent work, Rice *et al*. introduce a new training algorithm for CSRNs using parameter estimation with a decoupled EKF [77] and claim improved training times over those encountered with standard EKF.

3.5.4 Small Population Particle Swarm Optimization

Particle Swarm Optimization (PSO) was developed and applied to the training of NNs by Kennedy and Eberhart [53]. PSO is an evolutionary computation technique similar to the Genetic Algorithm (GA). Grant *et al*. utilize a form of PSO called Small Population Particle Swarm Optimization (SPPSO)[18] to train a CSRN adapted to identify and predict bus voltage dynamics [34].

3.5.5 Parameter estimation via Unscented Kalman Filter

In this work, we present a new training method for the CSRN based on parameter estimation with an Unscented Kalman Filter (UKF). This new method is compared against the EKF method described above.

3.6 Applications

3.6.1 2D Maze Traversal

Since the solution of the maze traversal problem using CSRN is fundamental to understanding the CSRN, we discuss the basic maze traversal problem in some detail. The maze traversal problem was first introduced as a robot navigation problem [94]. The maze navigation problem can be represented as a square grid of locations which are considered as path ways, obstacles, or the goal. Figure 3.3 shows a visualization of the problem in which the pathways are represented by blank cells, obstacles are black cells, and the goal is a red circle.

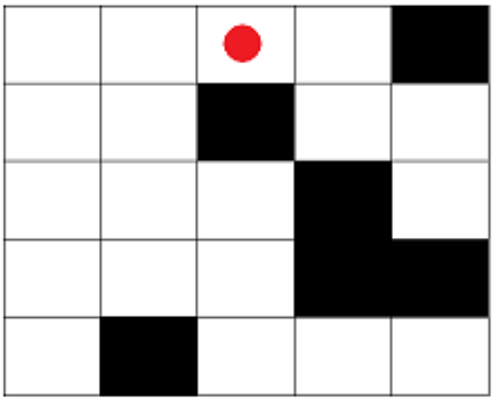


Figure 3.3: Sample maze. Blank cells represent an open pathway, black cells an obstacle, and red cell represents the target.

The objective is to learn the shortest distance to the goal from any cell. This is done by computing the cost-to-go-function for each cell in the grid. This objective is a long-term optimization problem, whose solution is given by the following Bellman equation [103],

$$J^*(i) = \min_{\mu}\left(c(i,\mu(i)) + \gamma \sum_{j=i}^{N} p_{ij}(\mu) \cdot J^*(j) \right) \tag{3.4}$$

where *i* is the initial state. *J*(i)* is the total expected cost from state *i*. The control policy, μ, is the mapping between states and the actions causing state transitions. The immediate cost of moving from state *i* to state *j* is *c(i,j)* and for the 2D maze traversal problem is always 1. The probability of moving from state *i* to *j* is given by p_{ij} and can only take values of 0 or 1. The discount factor is γ and *N* is the total number of cells in the maze.

The maze traversal problem has become a benchmark test for newer types of NNs, in much the same way as the XOR problem was for perceptrons. Figure 3.4 and 3.5 show the cellular structure and core network of the CSRN used by Pang *et al*. to solve the maze traversal problem. The similarity between these figures and the CSRN architecture shown above is immediately obvious.

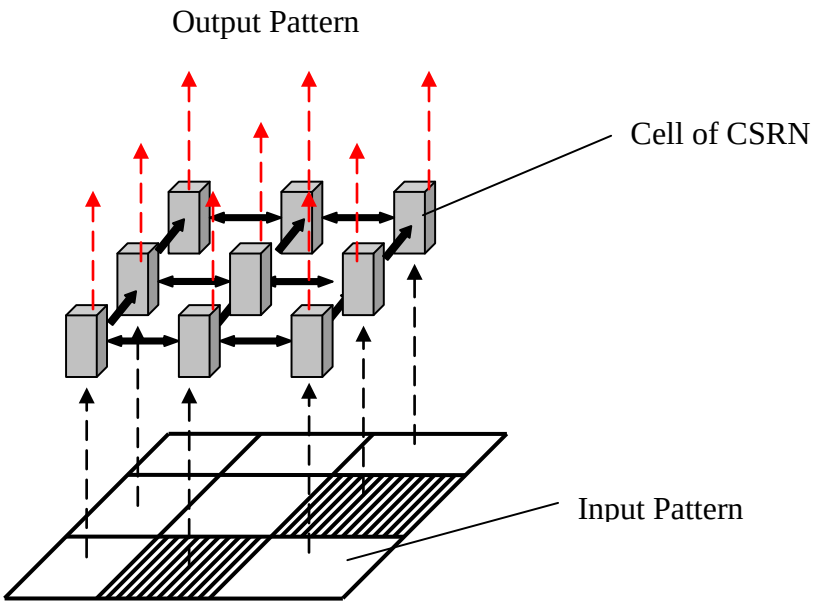


Figure 3.4: Cellular structure of the original CSRN used in solving the maze traversal problem.

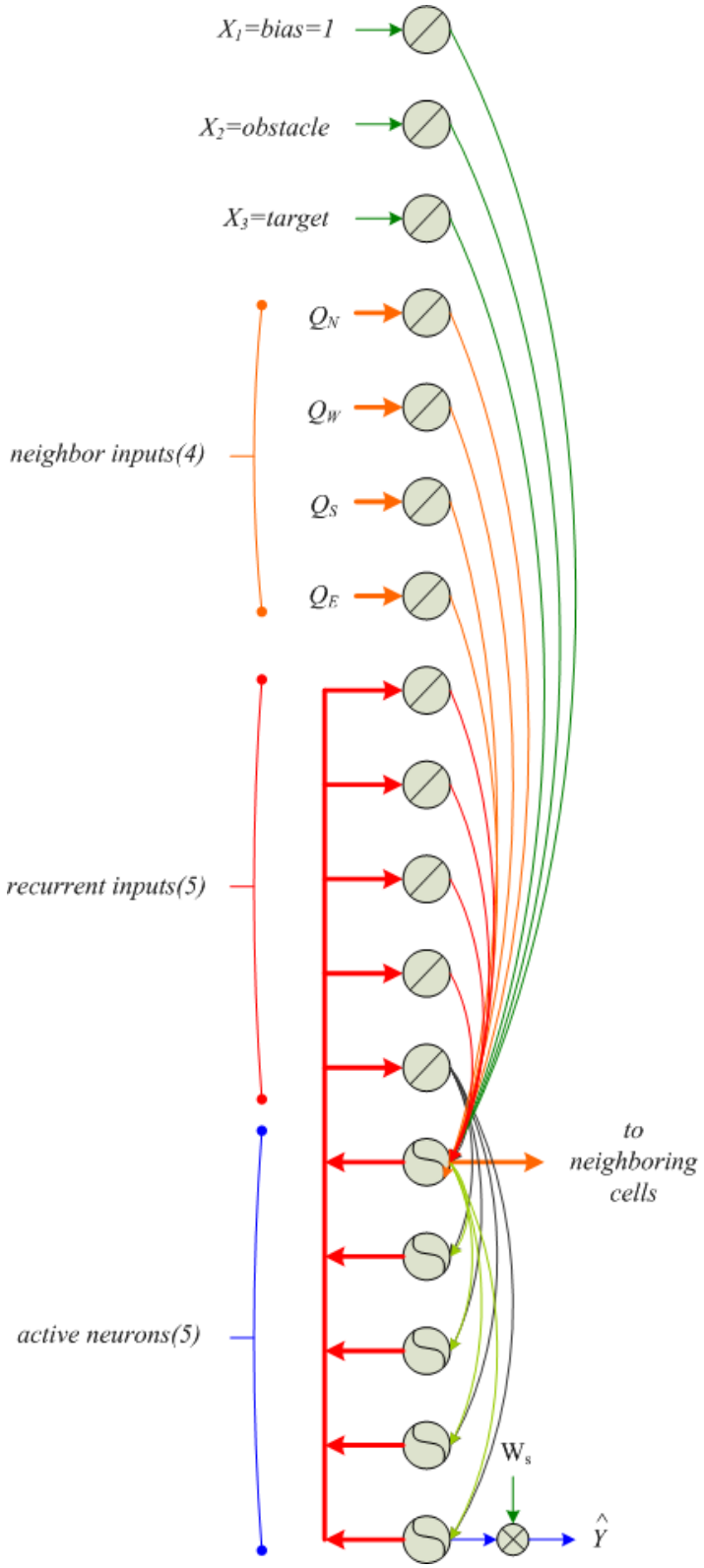


Figure 3.5: Core network of the original CSRN architecture used in solving the maze traversal problem. The core utilized a GMLP network.

The network in Figure 3.5 is a GMLP with 17 nodes; one bias node, two external input nodes, obstacle and goal, four input nodes from the corresponding neighboring cells, five self-recurrent input nodes that are fed back from the active nodes, and five active nodes (neurons). The obstacle input is binary and contains a one if that cell contains an obstacle and a zero if it does not. The goal input is similar. The four neighbor nodes take advantage of the symmetry of the problem and tie the cells together in the desired cellular structure. Note that this matches the geometric constraint of the problem in that the maze may not be navigated by diagonal movement. The five recurrent inputs contain information from the active nodes from the previous iteration. The five active nodes are fully connected within their layer. Due to this connectivity, the calculated value for the cell is taken from the last output node since it contains information from the other four active nodes.

Computing this cost-to-go function is difficult. Problems arise from lack of convergence and difficulties with local minima. Therefore, the solution provided by the CSRN is not always completely correct. The results do generally yield the shortest distance to the goal by moving to the neighboring cell with the lowest value. An example of a target maze and output as computed by the CSRN above are shown in Fig. 3.6.

2	1	0	1	25
3	2	25	2	3
4	3	4	25	4
5	4	5	25	25
6	25	6	7	8

4.5425	2.1212	-0.45413	3.3509	25.2418
4.4913	3.4978	25.1939	4.295	4.2635
4.5854	3.4942	2.9231	25.5665	4.7102
4.7251	3.6697	3.6441	25.0978	25.2649
4.8449	26.1331	4.7872	4.3763	4.0373

Figure 3.6: The left figure shows cost function array for the known solution of the maze shown in Fig. 2.3. The figure on the right shows the solution provided by the output of the CSRN depicted in Figs. 3.4 and 3.5.

3.6.2 Connectedness Problem

The connectedness problem was introduced by Minsky and Papert [62] and is described by the following question: "Is the input pattern connected?" Solution of the connectedness problem is fundamental to brain-like processing of images. It is directly related to image segmentation, *i.e*. segmenting an image into separate objects. Image segmentation is itself the primary preprocessing step in attempting higher level tasks such as object recognition or classification.

Feed-forward networks have been shown incapable of solving the connectedness problem [8]. Indeed, knowledge of how the human eye sequentially traces the boarders of an image during classification, suggest the need for recurrence. Ilin *et al*. demonstrated that the CSRN is capable of solving a subset of the connectedness problem [45][46]. This subset is described as follows: given a square image, are the upper left and lower right corners connected? The authors utilized the CSRN architecture discussed in Section 3.2 and shown specifically in Figs. 3.1 and 3.2. The output transformation used was also

a CSRN core. The input images were 7 x 7 binary images. They further tested the ability of both the MLP and CSRN to classify input images as “connected” or “not connected”. The results showed that the MLP performed slightly better than chance while the CSRN correctly classified the images in the 80% - 90% range. This work is of particular interest because it is the first successful application of CSRNs to image processing.

3.6.3 Image Registration

In previous works, we demonstrated simple registration of low complexity binary and grey-scale images subjected to in-plane rotation [3][5]. Results of those works are discussed in detail in Chapter 6.

3.6.4 Facial Recognition

Ren *et al*. exploited the state transition property of CSRNs to solve large-scale, pose invariant, facial recognition [74][75][76], formulating the recognition problem as a temporal prediction problem on image sequences. The pose angle of the face is varied in a consistent manner throughout each sequence, thus encoding the pose angle as a time index. Recurrent networks have been shown to be very effective for time-series prediction [21] and the added geometric structure of the CSRN makes it particularly attractive. Figure 3.7 shows a typical image sequence. The image dataset used by Ren *et al*. is known as VidTIMIT [82]. The dataset contains multiple pose variation sequences for 43 faces.

Figure 3.7: A typical image sequence use in Ren *et al*.'s pose-invariant facial recognition application. Sequence taken from the VidTIMIT dataset.

Direct application of CSRNs to facial images is computationally expensive due to the size of the images. Therefore, principal component analysis (PCA) was applied to the extracted face sequence to produce a series of vectors representing each face in the eigen-face feature space. The first ten principle components are used. Therefore, the input pattern takes the form of a ten element vector, instead of an [N,N] matrix. As in the case of the maze traversal problem, the geometry of the CSRN mirrors that of the input pattern. The core of the CSRN is the same GMLP core shown in Fig. 3.2 with the following exceptions: 1) only one external input is used, and 2) the cellular structure is reduced to one dimension, and therefore only two neighbors are utilized. Figure 3.8 shows the basic structure of Ren *et al*.'s facial recognition application. To train the CSRN, the vectors representing an image sequence for a specific face in the face database are applied, in sequence, to the input of a CSRN, as shown above. The network is, thereby, trained to classify that particular face. One CSRN is trained to classify each face in the face database.

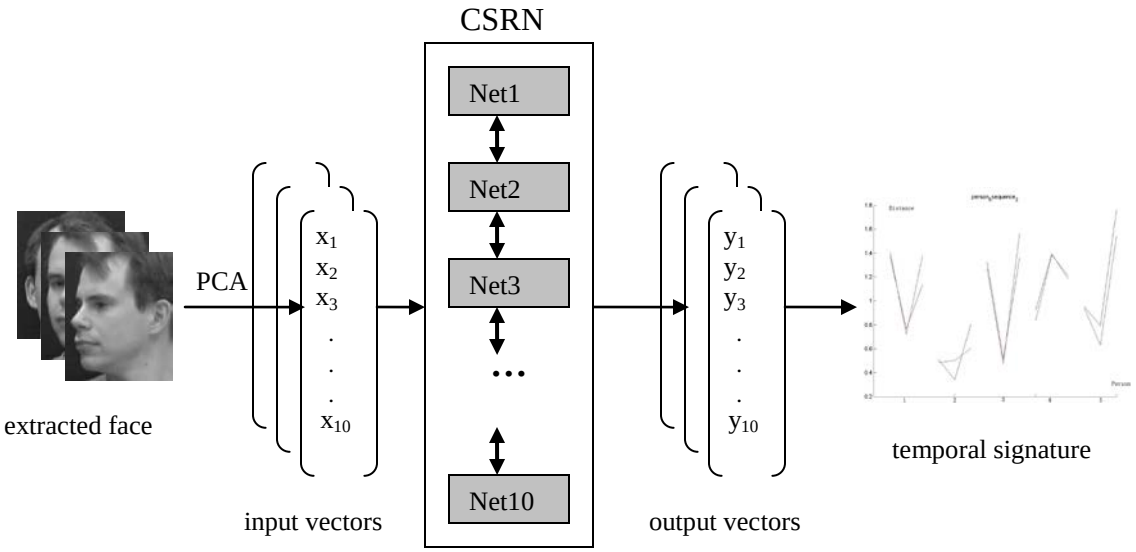


Figure 3.8: The basic structure of Ren *et al.*'s pose-invariant facial recognition system. The one-dimensional architecture of the CSRN is depicted.

Before we discuss the role of the CSRN in this application, let us first discuss Ren *et al.*'s use of the "temporal signature". They describe the temporal signature as the Euclidian distance between two successive pattern vectors in a sequence. This temporal information can be used to classify a face sequence as described below.

When a new testing sequence is encountered, each sequence is transformed into the face-space of every face in the face database, thus producing an input vector sequence. Each input vector sequence is applied to its corresponding CSRN, thereby obtaining a set of output vectors. The temporal signature of both the input sequence and output sequence are obtained, as discussed above. If the face in question corresponds to the face for which a particular CSRN has been trained, then the temporal signature of the input and output sequences will closely match. This temporal information can be plotted and a similarity measure is used to compare the temporal line of the input sequence to that of

the output sequence. If the similarity measure exceeds a given threshold, then the testing sequence is considered to match the face used to train the corresponding CSRN.

Ren *et al*. perform pose-variant experiments using basic PCA, an Elman SRN and the CSRN. He reports recognition rates of 51%, 67% and 80%, respectively. The CSRN significantly out-performs the other methods. A similar left-to-right pose-variant experiment was performed in the 2002 Face Recognition Vendor Test, FRVT 2002 [71]. The best system, one using the Eyematic technique, achieved a recognition rate of only 42%.

3.6.5 Voltage Prediction

Grant *et al*. apply CSRNs to the identification and prediction of buss voltage dynamics in practical power systems [34]. Similar to Ren *et al*.'s application to facial recognition, the geometric architecture of the CSRN is not constrained to a square, but is allowed to reflect the physical layout of the buss system with a single cell representing each individual bus. The connections between the CSRN cells reflect the actual connections between neighboring busses in the system.

Grant utilizes a standard Elman SRN as the core for each CSRN cell. Figure 3.9 depicts this core network. Each network has a single external input, the bus voltage measured at time, *t*; an input for each of its neighboring buses corresponding to the output (predicted voltage at time *t+1*) of the neighboring bus's cell; and a recurrent input from each of its hidden nodes. The number of hidden nodes is fixed at three. Each cell has a single output, the predicted bus voltage at time *t+1*. Since the number of nodes, and therefore the number of weights, varies from cell to cell, weight sharing is not used.

Grant also implements a voltage prediction system using a single Elman SRN with an external input and an output for each buss in the system. Both networks are trained using actual measured data from a small power system (12 busses). Simulations are performed for each network and comparisons between the two systems are made by examining the mean squared error (MSE) between the predicted voltage and the actual measured voltage at each buss. A variety of load disturbances are introduced into the system and in all cases the MSE of the CSRN was significantly lower that that of the single Elman SRN.

Besides being able to more accurately predict buss voltages, the CSRN offers several other advantages in this application. 1) Since the cell structure reflects the exact structure of the power system, the CSRN is easily scaled for larger systems. 2) Since only cells for neighboring busses are interconnected, regardless of how large the overall system becomes, the core networks remains small. 3) The CSRN can provide improved accuracy and speed of training due the small size of the core networks and the fact that the entire network is trained simultaneously.

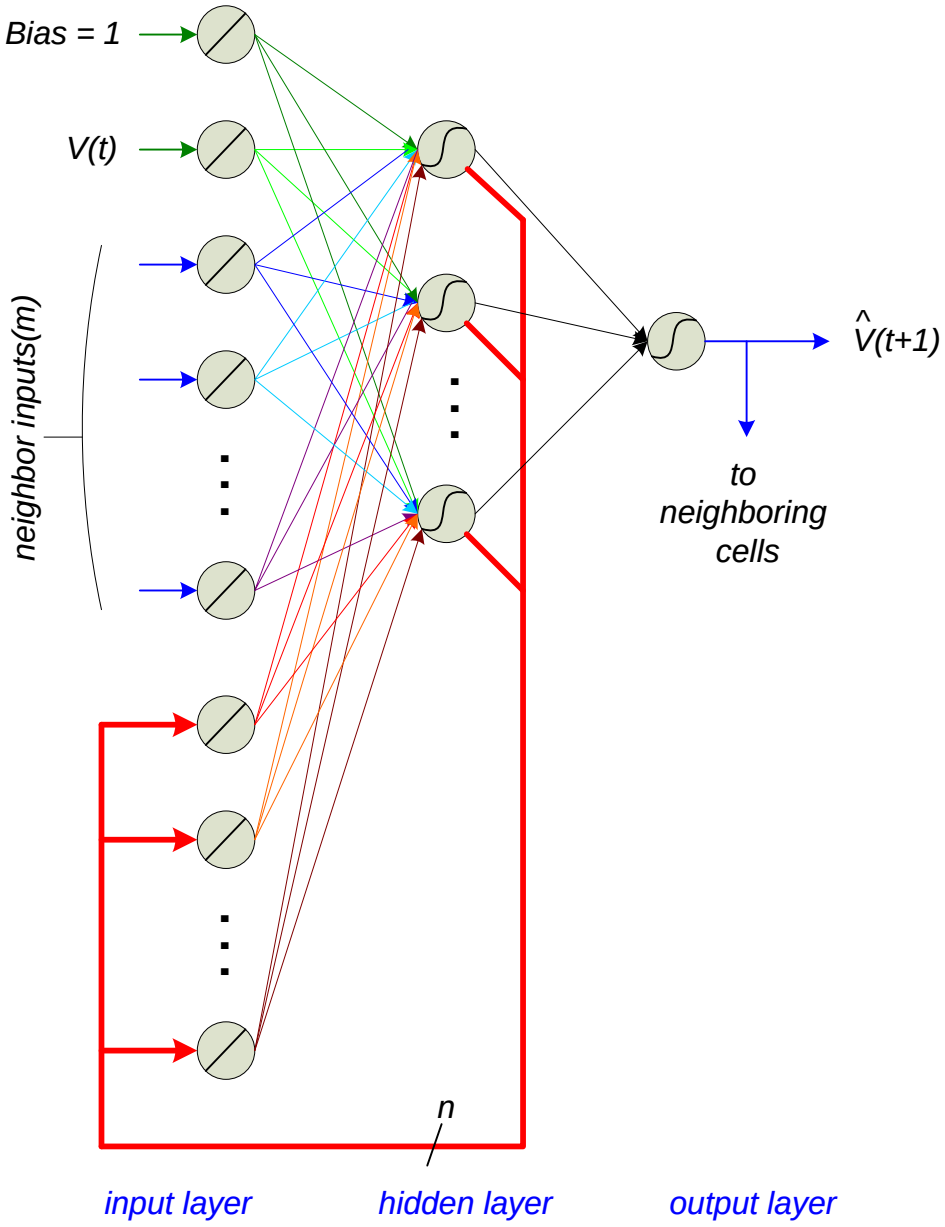


Figure 3.9: Standard Elman SRN core for the CSRN buss voltage prediction application. Core network shown with one bias, one external input, *m* neighbor inputs, *n* recurrent inputs, *n* hidden nodes and one output node.

4 IMAGE PROCESSING WITH CSRN

4.1 Introduction

4.1.1 Chapter Organization

In this chapter we first introduce metrics for the evaluation of the CSRN's performance in image processing (IP) task and discuss the experimental platform utilized for simulations in this work. Next, we discuss adaptation of sub-image processing for use with CSRNs. This concept is fundamental to the practical application of CSRNs to IP. Then, we present a generalized CSRN architecture for use in IP applications. Examples of pixel operations and spatial filtering are demonstrated.

4.1.2 Image Processing Metrics

Before discussing the application of CSRNs to IP tasks, we define several metrics which will allow us to determine the efficacy of the CSRN for IP tasks.

4.1.2.1 Function Approximation Metrics

The following metrics are used to evaluate how well a network learns a given function, J, which is then used to produce a final transformed output image.

- The J_{SSE} of the network is simply the SSE between the target function and the actual network output. This value is plotted as a function of epochs in order to evaluate how well the network learns and its rate of convergences. The magnitude of the SSE varies with image size, therefore, quantitative comparisons between results can only be made for solution with the same image size and number of training images.

- The J_{MSE} of the network is the mean squared error between the target function and the actual network output. This value is computed by normalizing the J_{SSE}, by image size and the number of training images. This eliminates the problem with J_{SSE} discussed above and allows quantitative comparisons

between results for solution with different image sizes and number of training images.

- The function accuracy, J_{ACC}, compares the output of each cell to the known function value (target) for that cell. If the values are equal, we consider that cell a match. The total number of matches normalized by the total number of cells yields the function accuracy in percent.

4.1.2.2 Image Comparison Metrics

The following metrics provide a quantitative method for comparison of a target image with the output image produced by the CSRN.

- The IM_{SSE} of the network is the SSE between the target image and the actual output image. This metric suffers the same limitation as that of the J_{SSE}.

- The IM_{MSE} of the network is the MSE between the target image and the actual output image. This value is computed by normalizing the IM_{SSE}, by the number of pixels in the image, which eliminates the problem with IM_{SSE} and allows quantitative comparisons between images of differing size.

- The image percent accuracy, IM_{ACC}, compares the final output image with the known target image, on a pixel by pixel basis. If the corresponding pixels have equal values, we consider that pixel a match. The total number of matches normalized by the total number of pixels yields the image accuracy in percent. The IM_{ACC} is an adaptation of the percent goodness metric used by Ilin in his evaluation of solutions for the maze traversal problem [48]. While this metric is reported for both binary and grey scale images, it is better suited for binary images due to the limitations placed on pixel values.

- The correlation ratio (*CR*) is a common metric for comparing the similarity of two images [31]. We designate this metric as the IM_{CR}, and compute its value using (4.1) below.

$$IM_{cr} = \frac{\sum_i \left(A(i) - \overline{A}\right)\left(B(i) - \overline{B}\right)}{\sqrt{\sum_i \left(A(i) - \overline{A}\right)^2 \left(B(i) - \overline{B}\right)^2}} \tag{4.1}$$

Here, *A(i)* and *B(i)* represent the individual pixels of image *A* and *B*, respectively and $\overline{A}$ and $\overline{B}$ the mean pixel values for image *A* and *B*, respectively. The closer this ratio is to 1, the more similar the images. This metric is primarily used in analysis of grey-scale images.

4.1.2.3 Time-Based Metrics

In this section we discuss time-based metrics. Unless otherwise, noted, all times are measured using MATLAB's® *tic*/*toc* functions.

- The training time, T_{TR}, is a measure of the time in sec required to train a given network.

- The forward computation time, T_{FC}, is the amount of time in msec required to compute one forward pass of the CSRN. I.e., the amount of time required for the CSRN to compute the function for which it was trained. The T_{FC} is computed by measuring the time required to perform the forward computation 1000 times, then normalizing the result.

- The run time, T_R, is the time required to run a single simulation with the CSRN application. This metric is computing using MATLAB's *etime* function.

- The batch time, T_B, is the total time required to run a batch of simulations with the CSRN application. This metric is computing using MATLAB's *etime* function.

4.1.2.4 Convergence Metrics

The following metrics are used to evaluate the convergence of the CSRN's MSE during the testing process.

- The convergence time is the time in epochs required for the MSE of the network to converge during the testing process. T_C is a computed approximation to the convergence time. It is the point at which the MSE has remained within 2% of its initial value for 20% of the total epochs.

- The settling time is the time, in epochs, required for the MSE to remain within 5% of its final value. T_S, is also used as an approximation to the true convergence time.

- The minimum error, E_{MIN}, is the minimum MSE achieved during the testing process.

- The minimum error location, T_{EM}, is the time, in epochs, at which point the minimum MSE occurs.

4.1.2.5 Summary of Metrics

Table 4.1 summarizes the metrics utilized in this work.

Table 4.1: List of metrics used to evaluate efficacy of the CSRN for IP tasks.

Metric	Description
$\boldsymbol{J_{SSE}}$	SSE between the target function and network output.
$\boldsymbol{J_{MSE}}$	MSE between the target function and network output.
$\boldsymbol{J_{ACC}}$	Percentage of cell outputs which exactly match known target function values.
$\mathbf{IM_{SSE}}$	SSE between the target image and output image.
$\mathbf{IM_{MSE}}$	MSE between the target image and output image.
$\mathbf{IM_{ACC}}$	Percentage of matching pixels between the target image and the output image.
$\mathbf{IM_{CR}}$	Correlation ratio between the target image and the output image.
$\boldsymbol{T_{TR}}$	Training time (secs) – time required to train the CSRN.
$\boldsymbol{T_{FC}}$	Forward computation time (msecs) – time required to compute one forward pass of the CSRN.
$\boldsymbol{T_{R}}$	Run time (min) – time required to run a single simulation of a CSRN application.
$\boldsymbol{T_{B}}$	Batch time (hr) – time required to run a batch of simulation of a CSRN application.
$\boldsymbol{T_{C}}$	Convergence time (epochs) – computed approximation. Point at which MSE remains within 2% of initial value for 20% of total epochs.
$\boldsymbol{T_{S}}$	Settling time (epochs) – computed approximation of time required for MSE to remain within 5% of its final value.
$\boldsymbol{E_{MIN}}$	Minimum error – minimum MSE achieved during testing processes.
$\boldsymbol{T_{EM}}$	Minimum error location (epochs) – time at which minimum error occurs.

4.1.3 Experimental platform

All simulations are performed on a Dell Precision PWS690 workstation with an Intel Xenon® X5355 8-core CPU running @ 2.66 GHz and 3.0 GB of RAM. The systems operating system is Windows© XP SP3. Simulations are run in MATLAB© Version 7, (R14) SP2. Forward and backpropagation computations for the CSRN are performed using custom C++ functions called from MATLAB©.

4.2 Sub-image Processing

As image size grows, so does the complexity of the CSRN. In practical implementations, a tradeoff exists between the number of training images, and the size of the images. Ilin *et al*. limit their work to 7x7 images [45][46][47]. In [5] we extend the use of CSRNs to 15x15 images. We find this image size to be the practical limit for CSRNs using the EKF training method, discussed in Section 8.3, with our hardware/software configuration discussed above. This is due to the size of the matrices required for computation of the Jacobian via BPTT.

To overcome this limitation, we utilize sub-image processing. We divide each training image into smaller, 5x5 sub-images. Next, we train a separate CSRN for each of the sub-images. Once training is complete, to perform a given transformation on an image, the image is divided into sub-images, each of which is processed by its corresponding CSRN. The outputs of each CSRN are then combined to form the final transformed image.

Although we do not discuss the application of CSRNs to image registration until Chapter 6, we utilize some of our early registration results to demonstrate the advantages of sub-IP here. In this experiment we register a simple 15x15 binary image of a cross

subjected to in-plane rotation using a CSRN with a GMLP core trained via the EKF method over the course of 100 epochs. Figure 4.1 below shows the resulting images.

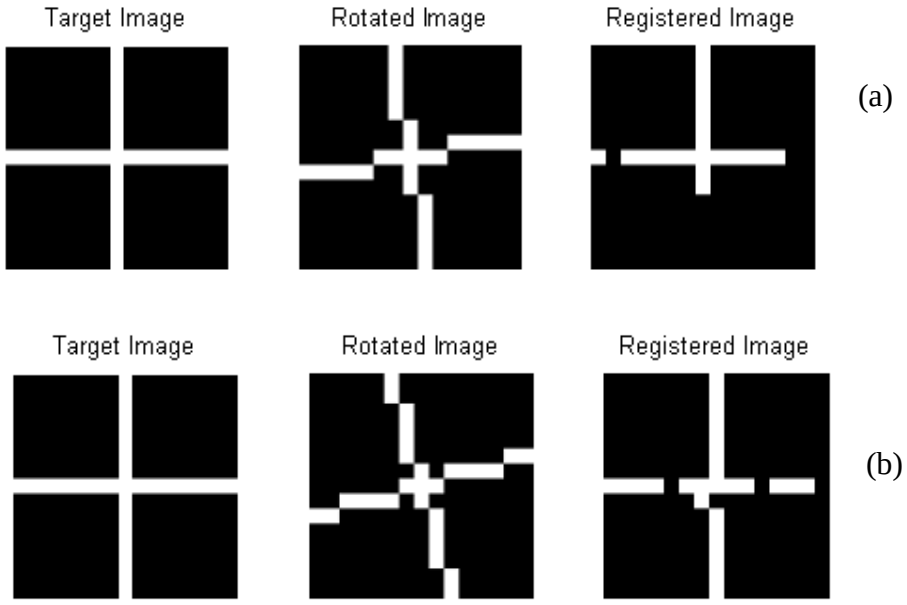


Figure 4.1: Image registration results for 15x15 binary images via a CSRN with a GMLP core. Part a, shows registration without sub-image processing, part b, shows registration with the addition of sub-image processing.

In the first test, the CSRN is trained using 5 training images without the use of sub-image processing. The training images are rotated from 0° to 20° in step of 5°. A single testing image, rotated by and angle of 12°, is used for testing. Figure 4.1a shows the registration results. The upper half of the test image is registered correctly, while the lower half is not. This sort of local registration, with some areas of the image being registered better than others, appears in many early. The best approximation to the registration function yields around 46% accuracy with a final image registration of around 96% accuracy. The total training time for this test is approximately 400secs.

In the second test, sub-IP is utilized and the CSRN is trained using 11 training images rotated from 0° to 20° in step of 2°. A single testing image, rotated by and angle of 16°, is used for testing. Figure 4.1b shows the registration results. In this example, we see a marked improvement in global registration, as indicated by a function accuracy of 64%, and image accuracy of 98.2%. The training time is approximately 150secs.

Table 4.2 tabulates the results.

Table 4.2: Results for the addition of sub-image processing to binary image registration via CSRN.

Test	**Training Images**	J_{ACC} (%)	IM_{ACC} (%)	T_{TR} (sec)
without sub-image processing	5	46	96	400
with sub-image processing	11	64	98.2	150
% improvement	120	18	2.2	-62.5

The addition of sub-IP decreases the training time by 62.5%. This time savings, allows the user to increase the number of training images, for improved network performance, or to increase image size.

In this experiment, the addition of sub-IP allows us to increase the number of training images from 5 to 11, resulting in improvements in both function accuracy and global registration.

4.3 A Generalized CSRN Architecture for Image Processing

The cellular structure of the CSRN shown in Fig. 3.4, is easily adapted for IP as shown in Fig. 4.2. Note the one-to-one correspondence between the input image, the CSRN and the output image.

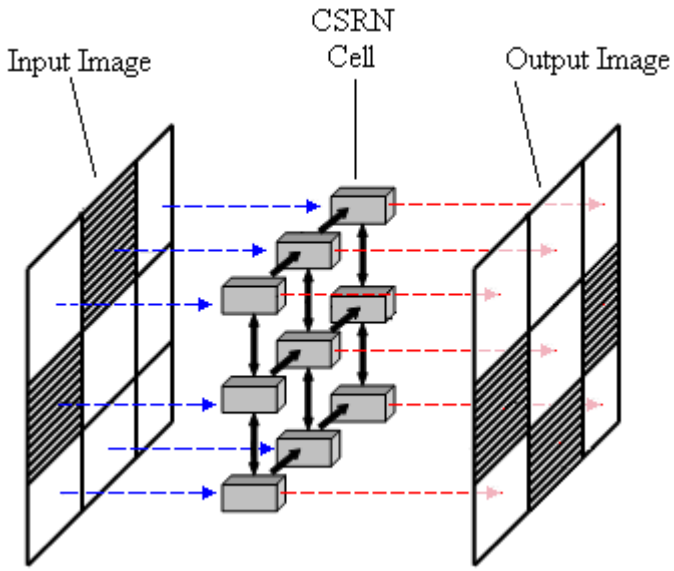


Figure 4.2: The generalized architecture of the CSRN for image processing. Grey boxes represent individual cells of the network.

Early in the course of our efforts to apply CSRNs to IP tasks, we began to suspect that the CSRN architecture might require modification to achieve successful implementation of specific tasks. As we will see later in this chapter, this suspicion holds true. When using training methods which require computation of the Jacobian via BPTT, even slight modifications in the architecture require the equations for BPTT be re-derived and its algorithm recoded. This is not a trivial task! We experienced this first hand when experimenting with the alternative cores discussed in Chapter 7. The process is both tedious and time consuming. As a result, we recognize the need for a generalized

architecture for the CSRN that can easily be scaled to handle the majority of configurations required for IP.

4.3.1 Comparison of CNNs and CSRNs

A literature survey reveals that CNNs have been successfully applied to the translation and rotation of both binary [27] and grey-scaled images [17]. In fact, CNNs are used for a wide variety of IP tasks, from basic filtering to pattern recognition. An examination of the CNN architecture reveals some interesting similarities, as well as differences, between the CNN and CSRN. A detailed comparison follows.

4.3.1.1 Active Neurons

Let us start by examining the active neurons within each type of network. The CNN has only 1 active node per cell, while the CSRN has 5 active nodes. Theoretically, this should allow the CSRN to approximate more complex function.

4.3.1.2 Inputs

Next, we examine the cell inputs. The CSRN has, in addition to the overlying pixel intensity, several external inputs. This gives the CSRN access to problem specific parameters. The CNN has as inputs, not only its underlying pixel intensity, but the intensities of its 8 neighbors as well. This allows the CNN to pre-process a window of inputs, which is very useful when implementing many IP filters. The CSRN has inputs coming from the outputs of its 4-neighbors, while the CNN has inputs from the outputs of its 8-neighbors. This should give the CNN a better resolution in terms of local computations. Finally, both the CNN and CSRN have recurrent inputs. However, since the CSRN has 5 active nodes, it has 4 additional recurrent paths.

The CSRN pixel inputs can be thought of as equivalent to the CNN inputs, with b_{ij} = 1, and all the other elements of $\boldsymbol{B}$ set to zero. The CSRN neighbor inputs can be thought of as the same as the CNN's feedback inputs with the weights for any element which does not have an *i* or *j* subscript set to zero. The recurrency in the CNN and CSRN are similar, with the CNN having only one recurrent path.

4.3.1.3 Weights

In the CNN, the weights are preselected based on the type of application, and no training is required. The weights are stored in two arrays, or cloning templates, the input or control template, $\boldsymbol{B}$, and the feedback template, $\boldsymbol{A}$. In the CSRN, the weights are "learned" during the training process and are store in a single matrix.

4.3.1.4 Activation Functions

The activation function for the active nodes in both circuits is quite similar. The CSRN uses a *tanh* function, while the CNN, uses the piece-wise linear approximation to the *tanh* function. The continuous nature of the *tanh* function has desirable effects in the use of BPTT to compute the derivates required in training the network.

4.3.1.5 Outputs

Each network type has a single output. The CSRN output utilizes a scaling weight, not included in the CNN.

4.3.2 Cell Configuration of the Generalized CSRN

The differences between the CNN and CSRN led us to the development of a generalized core configuration. Table 4.3 summarizes these differences and highlights the features selected in the implementation of the generalized core configuration shown in Fig. 4.3. This configuration utilizes the external inputs of the CSRN, giving the network access to problem specific parameters. The inclusion of the x and y pixel positions as external inputs gives the CSRN an awareness of its location within the cellular structure which is vital to the solution of IP tasks involving topological relationships. The configuration utilizes a CSRN core, which gives the network added computational ability due to the increased number of active neurons and increased recurrency. The CSRN core also allows the weights to be adapted via training. The new configuration utilizes the CNN's 8-neighbor pixel inputs, allowing the CSRN to pre-filter input pixels and the CNN's 8N output connections instead of the CSRN's standard 4N connections, giving it better resolution in terms of local computations. Finally, it uses the scaling weight of the CSRN core to allow scaling of the final output.

Table 4.3: Different attributes of the CNN and CSRN. Attributes selected for use in the generalized architecture are highlighted.

Core Type	Active Neurons	External Inputs	Pixel Inputs	Neighbor Connections	Weights	Recurrent Paths	Outputs
CNN	1	none	9	8	shared fixed	1	1
CSRN	5 variable	13 variable	1	4	shared learned	5	1 variable

Using this configuration, the set of problems which can be solved by the CNN may be considered a subset of those which can be solved by the CSRN. In essence, by proper selection of the CSRN parameters, the CSRN can be easily scaled down to any desired CNN configuration.

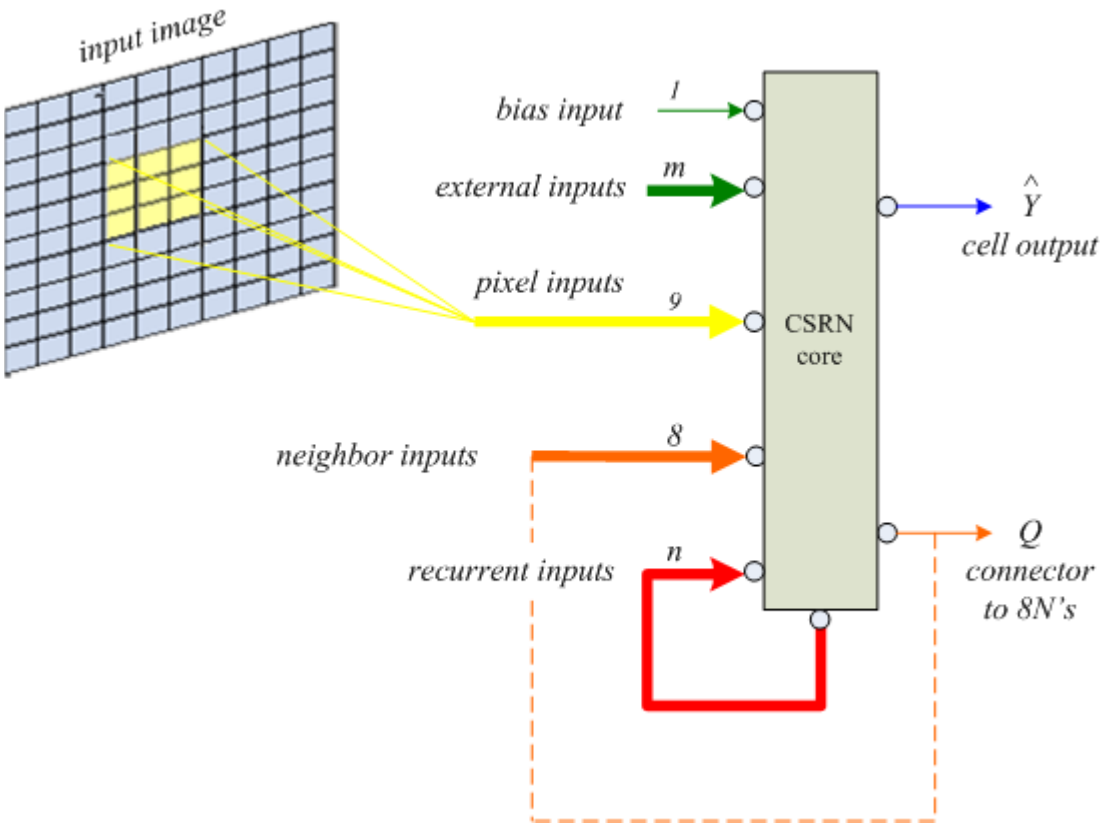


Figure 4.3: Cell configuration for the generalized CSRN.

4.4 Adapting the CSRN for Image Processing

4.4.1 Selection of Network Parameters

In order to adapt the CSRN for a specific IP tasks we must select the following parameters.

- External inputs - the number and type of external inputs. These may be parameters of the function to be approximated and/or location of the cell with the image structure.

- Pixel inputs – the number of pixels from the input image (1, 4 or 8). This can be the cell's corresponding input pixel, in the case of pixel operations or geometric transforms, or a neighboring window of inputs in the case of spatial filters.
- Network outputs – the number and type of outputs for the network.

- Core Network. We implement the GMLP, Elman SRN (ESRN) and the Elman SRN with multi-layered feedback (ESRNmlf) cores.

- Number of core iteration (1 to p) – the number of internal iterations used for making recurrency computations

- Number of neighbor inputs (0, 4 or 8). Selection of no neighbor inputs disables the use of feedback from neighboring cells. Selection of 4 or 8 neighbor inputs allows for connectivity between 4 or 8 neighbors, respectively.

- Number of self recurrent inputs (0 to n). Selection of no self-recurrent inputs disables the use of self-recurrent. Simultaneous selection of no neighbor inputs, no self-recurrent inputs, and 1 core iteration eliminates recurrency from the CSRN.

- Number of active neurons (1 to n).

- Training method – the method used to train the CSRN. We implement EKF and UKF methods.

4.4.2 CSRN Image Processing Algorithm

Figure 4.4 shows a flowchart of the CSRN IP algorithm. This algorithm is generic and can be used to train a CSRN to perform any IP task. Figure 4.5 shows details of the *training*, *testing*, and *results* loops of Fig.4.4.

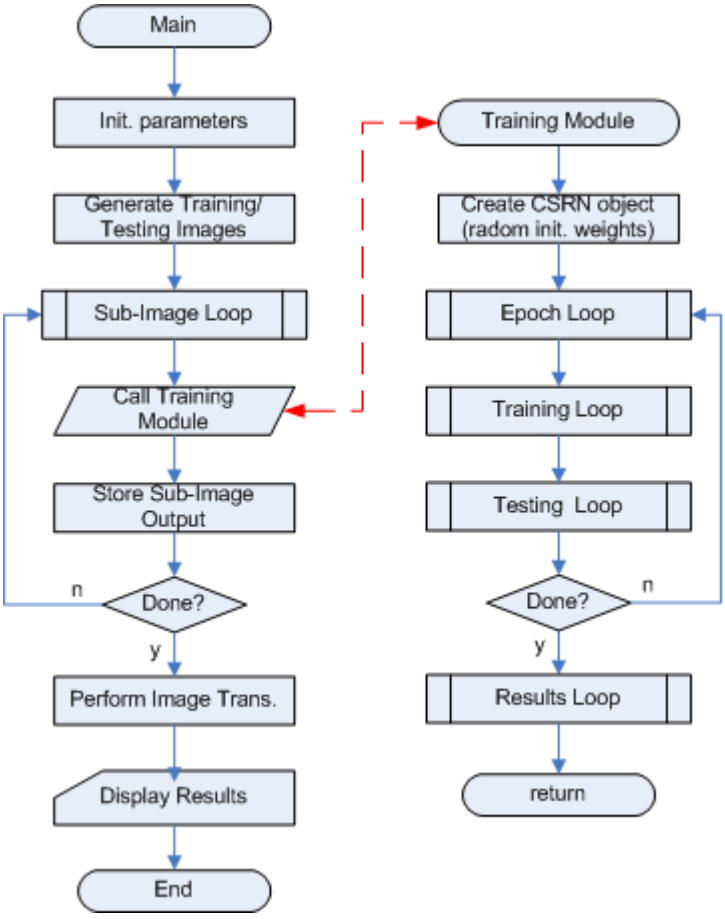


Figure 4.4: Flowchart for the CSRN image processing algorithm.

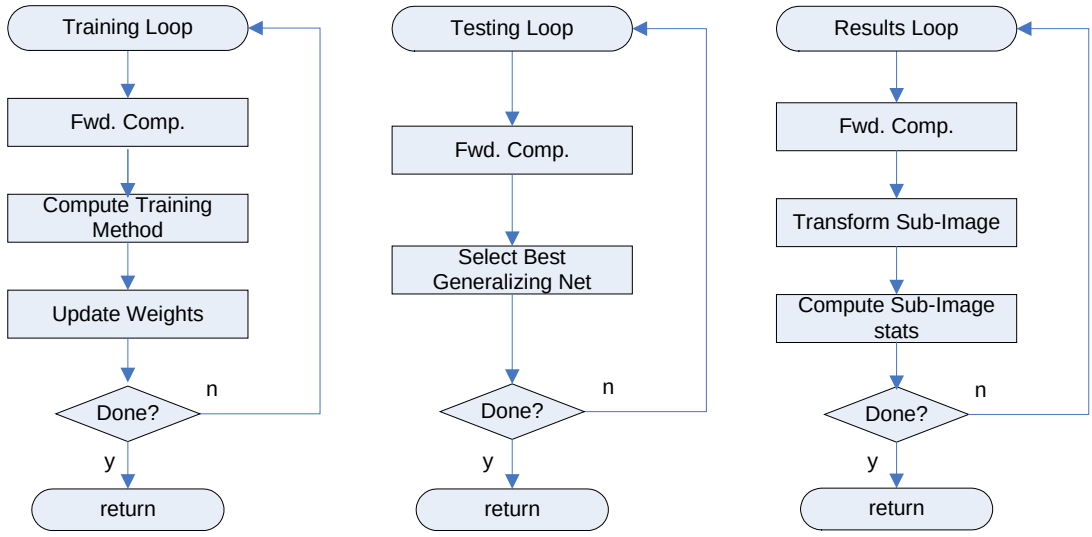


Figure 4.5: Detailed flowchart of the training, testing and results loops of the CSRN image processing algorithm.

Table 4.4 shows the computation load for the IP algorithm. To compute a baseline computational load we select the following parameters:

- a GMLP core with 13 input nodes, 5 active nodes and 10 core iterations
- an image size of 15 x 15 pixels using 5 x 5 sub-images processing
- 11 training images and 11 testing image
- 250 epochs utilizing the EKF training method.

The computational load is given for both the full training algorithm as well processing of a single image once trained.

Table 4.4: Computation load of the CSRN image processing algorithm. Computed for GMLP core with 18 nodes, image size of 15x15, 5x5 sub-image processing, 11 training and testing images, and the EKF training method with 250 epochs.

Computation	Multiplications	Additions
Training Loop		
Fwd. Comp.	2.3650E+05	2.3650E+05
Back. Comp.	4.8675E+05	7.2050E+05
EKF	1.9769E+07	1.9836E+07
Weight Update		8.6000E+01
training loop sub-total	2.0492E+07	1.9836E+07
Testing Loop		
Fwd. Comp.	2.3650E+05	2.3650E+05
single pass sub-total	2.0729E+07	2.1029E+07
epoch subtotal(250 epochs)	5.1822E+09	5.2573E+09
sub-image subtotal(9 sub-ims)	4.6639E+10	4.7316E+10
Full Training Algorithm	**4.6639E+10**	**4.7316E+10**
Single Image Processing	**1.9350E+05**	**1.9350E+05**

4.5 Pixel Operations

For the purposes of this work, we consider pixel operations to be any IP task that operates on a single pixel at a time. Examples include grey-scale to binary conversion (G2BC), sometimes referred to as *threshholding*, contrast adjustment, and histograms. These tasks are easily solved using CNNs. Even standard MLPs can perform these operations, albeit slower due to the fact that they must process each pixel separately then recombine the results into a final image. These tasks require only the corresponding pixel input (and possibly additional function parameters). No connections to surrounding cells and no recurrency are required. Use of a CSRN for these tasks overcomplicates the solution, which can cause longer than necessary training times and convergence

problems. However, the advantage of using the generalized CSRN architecture over the CNN is that the CSRN is capable of learning to perform the pixel operation rather than having to compute the weights a-priori. A simple example is shown below to demonstrate how the generalized structure described above can be scaled down to perform simple pixel operation.

4.5.1 Grey-scale to Binary Conversion using CSRN

In this experiment, we configure and train the CSRN to perform G2BC.

4.5.1.1 Grey-scale to Binary Conversion

When performing G2BC on an image, a threshold value is selected and each pixel is examined. If the pixel value is less that the threshold, then the corresponding output pixel is set to a predefined value, usually zero. If the pixel value is equal to or greater than the threshold value, then the corresponding output pixel is set to a different predefined value, usually one. The resulting output image is a binary or black and white image [31]. This thresh -holding process can be described mathematically as,

$$I_2(x,y) = \begin{bmatrix} 0 \ldots \ if \ \ I_1(x,y) < TH \\ 1 \ldots \ if \ \ I_1(x,y) \geq TH \end{bmatrix}, \quad \begin{matrix} x = 1 \ to \ \ N \\ y = 1 \ to \ \ M \end{matrix}, \tag{4.2}$$

where $I_1(x,y)$ is the NxM image to be filtered and $I_2(x,y)$ is the resulting filtered image. (x,y) represents the current location of the filter mask and TH is the threshold value.

4.5.1.2 CSRN implementation of G2BC

Figure 4.6 shows a scaled down version of the generalized CSRN architecture used to implement the G2BC task. The corresponding network core is shown in Fig. 4.7.

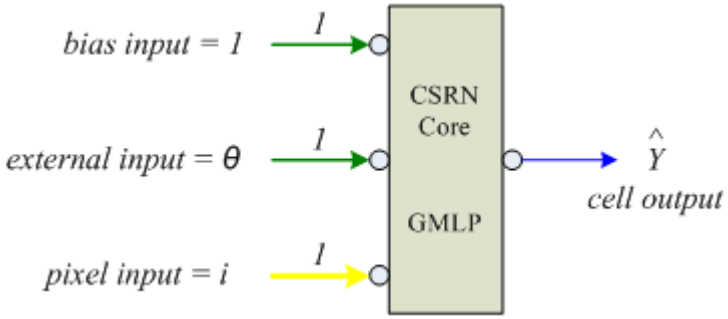


Figure 4.6: Cell configuration diagram for the generalized CSRN scaled down to implement grey-scale to binary conversion.

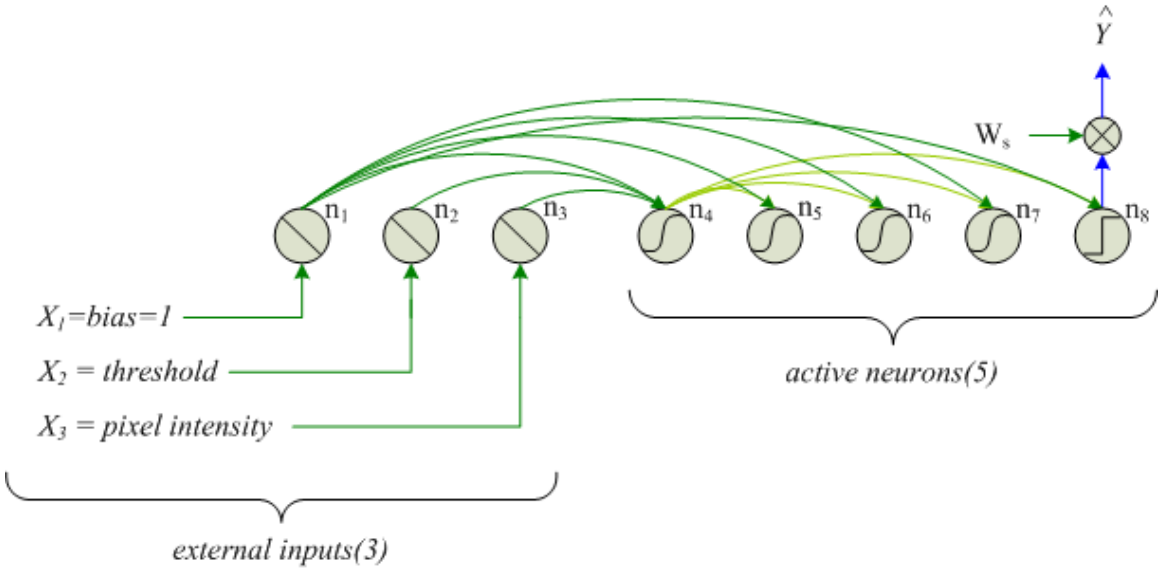


Figure 4.7: The network core used for grey-scale to binary conversion. Core network is a GMLP configure with 5 active neurons. It utilizes 3 inputs which include a bias input, a single external input equal to the threshold parameter, θ, and a single pixel input, that of the cells corresponding pixel value. No neighbor or self-recurrent inputs are used, and the number of core iterations is set to 1. The scaling weight is set to $W_s = 1$. The network uses a total of 8 nodes.

To implement the G2BC function, we utilize a GMLP core network with 5 active neurons and 3 external inputs: bias input, threshold parameter, θ, and a single pixel input. The input image is of type double with intensity values of (0, 1). We set the number of neighbor inputs, as well as, the number of recurrent inputs to zero and set the number of core iterations to one, thereby removing all recurrency. With recurrency remove the core reduced to a feed-forward network. We utilize five active neurons, and a hard-limiter activation function in the last neuron. The scaling weight is set to, Ws = 1, and the cell output becomes a binary signal with values of {0, 1}.

4.5.1.3 G2BC Results

In this section we examine the results of the CSRN implementation of G2BC described above. The network is trained via the unscented Kalman filter method described in Section 8.6. A threshold value of $\theta = 0.4$ is used. Target images are generated using MATLAB'S® G2BC function, *im2bw( )*. In the first experiment, we train the CSRN using 11 facial images taken from the YaleB face dataset [27]. Testing is done using the training set to evaluate how well the CSRN learns. In the second experiment, testing is performed using an independent testing set, also consisting of 11 images from the same dataset, to evaluate how well the CSRN generalizes. In each case, a single testing image is selected as a primary test case (PTC) and is displayed in greater detail.

4.5.1.3.1 Experiment 1: Testing with Training Set

Figure 4.8 shows the image results of *Experiment 1*. Row a) shows the input images. Row b) shows the corresponding CSRN output images. Row c) shows the target images produced by *im2bw( )*. Each column contains the results for a specific input image. The PTC is indicated and shown in detail in Fig. 4.9. Table 4.5 tabulates the results of this experiment.

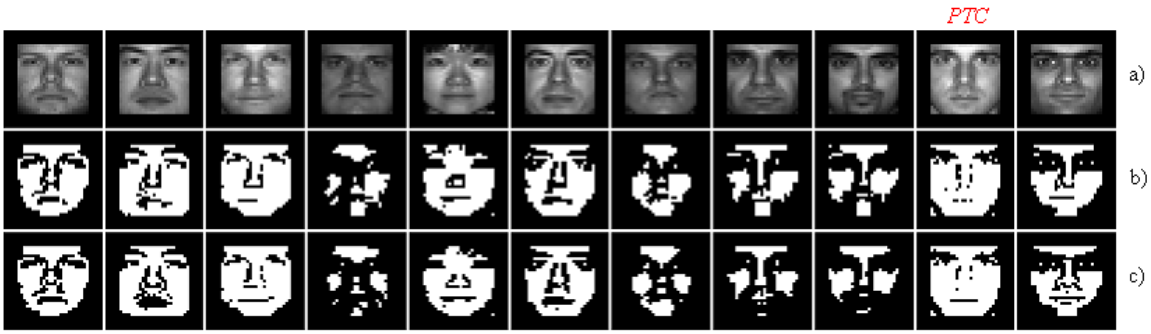


Figure 4.8: Results of CSRN implementation of grey-scale to binary conversion. Network utilizes a GMLP core with 5 active nodes, trained via UKF. Neighbor and self-recurrent inputs have been eliminated and all recurrency turned off. Two external inputs are used, the threshold parameter, θ = 0.4 and the cell's corresponding pixel intensity. Results are shown for testing with the full training set. Row a) shows the input images. Row b) shows the corresponding CSRN output images. Row c) shows the target images produced by *im2bw( )*. The primary test case (PTC) is indicated.

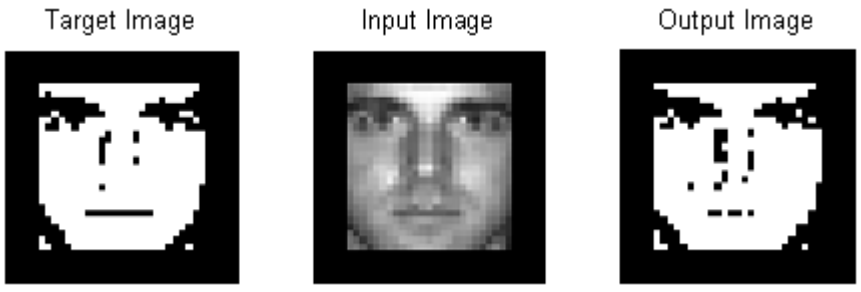


Figure 4.9: Results for primary test case in grey-scale to binary conversion *Experiment 1*, testing with training set. The network achieves a final IM_{ACC} = 98.3% and an IM_{CR} = 96.4%.

Table 4.5: Results of G2BC implementation. Data from testing with full training set shown. PTC is highlighted in red.

	***G2BC Experiment 1* Full Test Results**		
Test Image	**IM_{MSE}**	**IM_{ACC}**	**IM_{CR}**
1	2.12E-02	97.9	0.952
2	3.43E-02	96.6	0.928
3	1.80E-02	98.2	0.963
4	6.37E-02	93.6	0.748
5	3.92E-02	96.1	0.912
6	2.78E-02	97.2	0.937
7	4.16E-02	95.8	0.863
8	6.78E-02	93.2	0.789
9	2.06E+03	94.4	0.830
10	2.88E+03	98.3	0.964
11	3.67E+03	96.3	0.908

An inspection of Figs. 4.8 and 4.9 show that the output images produced by the CSRN are very similar to the baseline images produced by MATLAB'S® *im2bw( )* function. This is supported by the metrics in Table 4.5. The CSRN achieves an average image accuracy, IM_{ACC} = 96.1% over all training images and a best case IM_{ACC} = 98.3%. These results indicate that the CSRN has learned to perform G2BC for the training set.

4.5.1.3.2 Experiment 2: Testing with an Independent Test Set

In this experiment we repeat *Experiment 1* using an independent testing set. Figure 4.10 shows the image results for this experiment. Row a) shows the input images. Row b) shows the corresponding CSRN output images. Row c) shows the target images produced by *im2bw( )*. Each column contains the results for a specific input image. The

PTC is indicated and shown in detail in Fig. 4.11. Table 4.6 tabulates the results of this experiment.

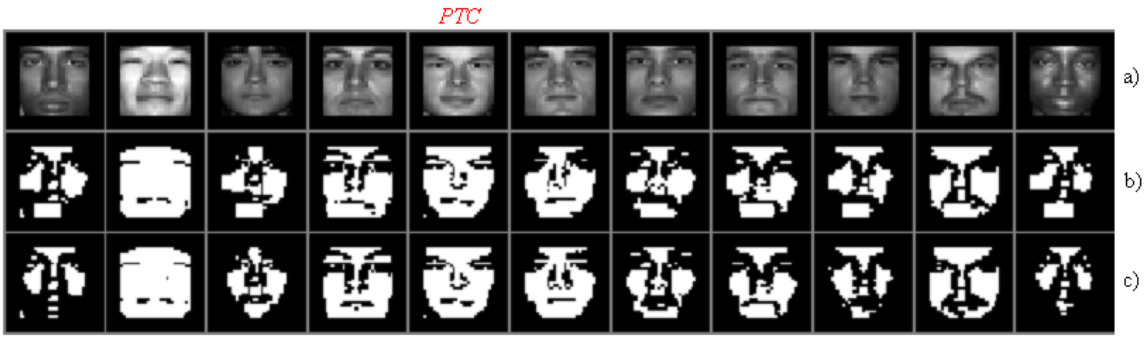


Figure 4.10: Results of CSRN implementation of grey-scale to binary conversion. Network utilizes a GMLP core with 5 active nodes, trained via UKF. Neighbor and self-recurrent inputs have been eliminated and all recurrency turned off. Two external inputs are used, the threshold parameter, $\theta = 0.4$ and the cell's corresponding pixel intensity. Results are shown for use of an independent testing set. Row a) shows the input images. Row b) shows the corresponding CSRN output images. Row c) shows the target images produced by *im2bw( )*. The primary test case is indicated.

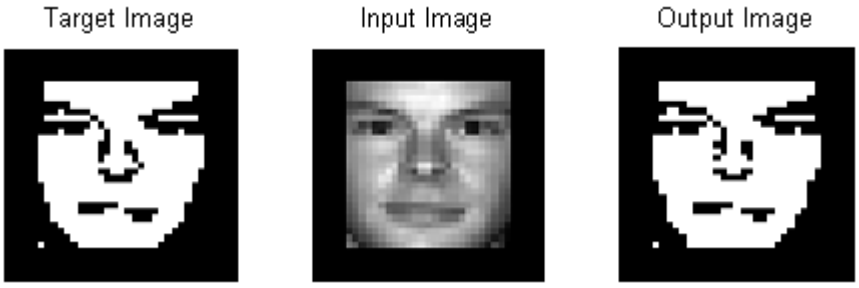


Figure 4.11: Results for primary test case in grey-scale to binary conversion *Experiment 2*, testing with independent test set. The network achieves a final $IM_{ACC} = 98.3\%$ and an $IM_{CR} = 96.4\%$.

Table 4.6: Results of G2BC implementation *Experiment 2*, testing with independent test set. PTC is highlighted in red.

	***G2BC Experiment 2* Full Test Results**		
Test Image	**IM_{MSE}**	**IM_{ACC}**	**IM_{CR}**
1	6.94E-02	93.1	0.786
2	1.06E-02	98.9	0.979
3	5.96E-02	94.0	0.810
4	3.51E-02	96.5	0.922
5	7.35E-03	99.3	0.984
6	3.10E-02	96.9	0.929
7	4.73E-02	95.3	0.868
8	3.67E-02	96.3	0.906
9	6.53E-02	93.5	0.802
10	4.98E-02	95.0	0.882
11	6.69E-02	93.3	0.753

Table 4.7 compares the results of the two experiments. The CSRN performs well with the independent test set, achieving an average image accuracy, IM_{ACC} = 95.6% over all training images and a best case IM_{ACC} = 99.3%. In fact the best case results with the independent test set out-perform the best case results with the training set. However, the average values are more in line with what is expected, with the training results performing slightly better. The key metric here is the average IM_{ACC}. We see that the results with the independent test set measure to within 0.5% of the results with the training set. This indicates that the CSRN generalizes well in our G2BC application.

Table 4.7: Comparison of testing results for the CSRN implementation of G2BC. Results for testing with training set and independent testing set are compared.

G2BC Testing Comparison						
Metric	**Units**		***Training***	***Testing***	***Diff.***	***% Diff.***
$\mathbf{IM_{MSE}}$	-	*Ave*	3.85E-02	4.36E-02	5.10E-03	13.2
		Best	1.71E-02	7.53E-03	-9.57E-03	-56.0
$\mathbf{IM_{ACC}}$	*%*	*Ave*	96.1	95.6	-0.50	-0.5
		Best	98.3	99.3	1.00	1.0
$\mathbf{IM_{CR}}$	-	*Ave*	0.890	0.875	-0.015	-1.7
		Best	0.964	0.984	0.020	2.1

4.5.1.3.3 Conversion, Training and Computation Times

Figure 4.12 shows plots of the MSE as a function of epochs for testing the CSRN with both the training set and the testing set. The results are those generated in the PTC of each experiment.

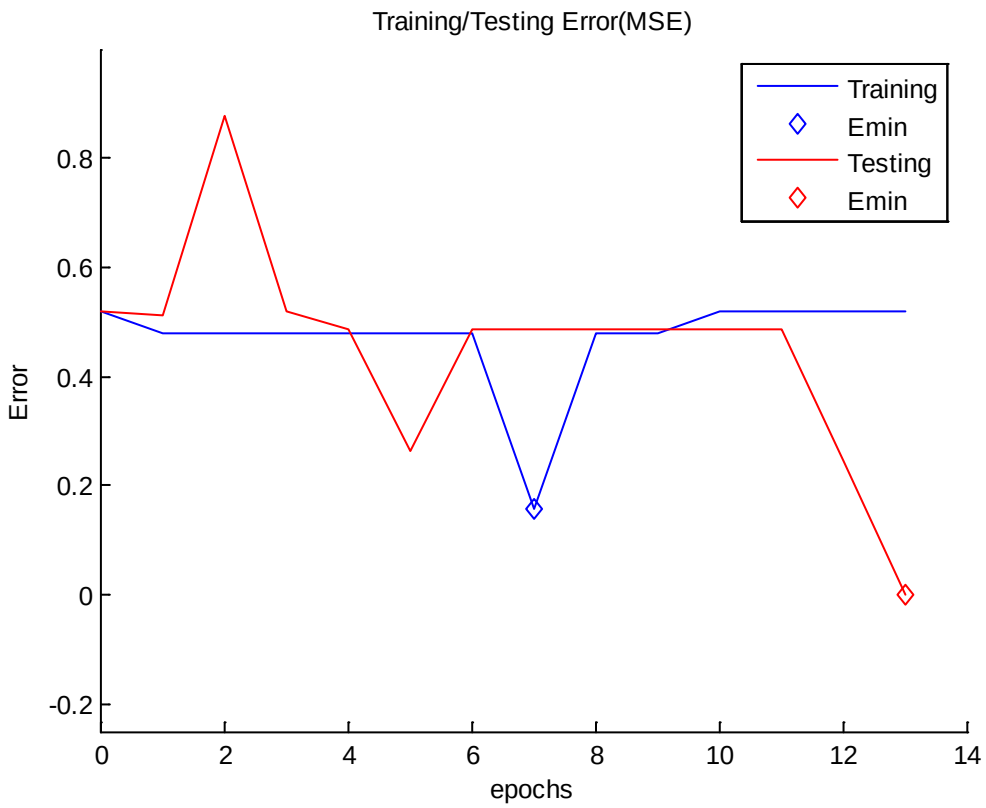


Figure 4.12: Plots of training/testing MSE vs. epochs for the CSRN trained to perform GTB conversion. The blue line is the error for the training set. The red line is that for the testing set. Diamonds indicate the minimum error. Plots represent the PTC for each experiment.

In both cases the learning process is stable and converges in less than 15 epochs. In this particular case, as discussed above, the testing error converges to a lower error than the training error.

Table 4.8 lists the training, computation and run times for the CSRN implementation of G2BC. For the G2BC application with 11, *35x35*, training images the total training time is just over 7.5 min. A normalized training time is also given. For this metric, the training time is normalized for image size and number of training images. Once trained the CSRN performs the G2BC transformation in just 3.69ms. The total run time for a

typical simulation of the G2BC application is 52 min. The total run time may vary depending on how quickly the network converges.

Table 4.8: Computation times for CSRN implementation of G2BC.

		CSRN G2BC Implementation
Metric	**Units**	
T_{TR}**(total)**	*sec*	547
T_{TR}**(norm)**	*msec*	40.6
T_{FC}	*msec*	3.69
T_{R}	*min*	52

4.5.1.3.4 Effects of Recurrency on Pixel Operations

As previously discussed pixel operation require no recurrency. When implementing the G2BC application we initialing tested the application with full recurrency, i.e. both neighbor and self-recurrency. This over-complicates the solution causing very slow convergence and poor function approximation. Figure 4.13 shows the MSE plot for the G2BC application implemented with a GMLP core, utilizing full recurrency, trained via the EKF method. Figure 4.14 shows the corresponding results for the PTC. Note the poor performance of the network in spite of having trained for 500 epochs. This illustrates one advantage of a flexible IP architecture.

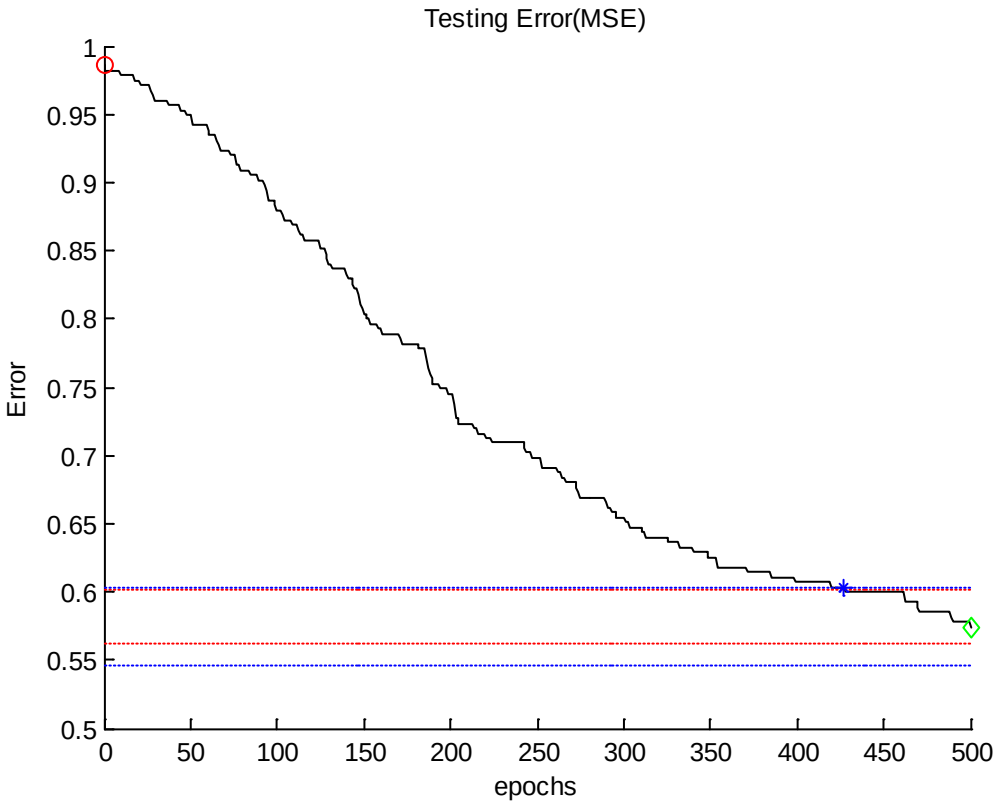


Figure 4.13: Plots of training/testing MSE vs. epochs for the CSRN trained to perform GTB conversion implemented with a GMLP core, utilizing full recurrency, trained with the EKF algorithm set to run for 500 epochs. The green diamond indicates the minimum error. The blue asterisk represents the settling point. Plot is for the PTC.

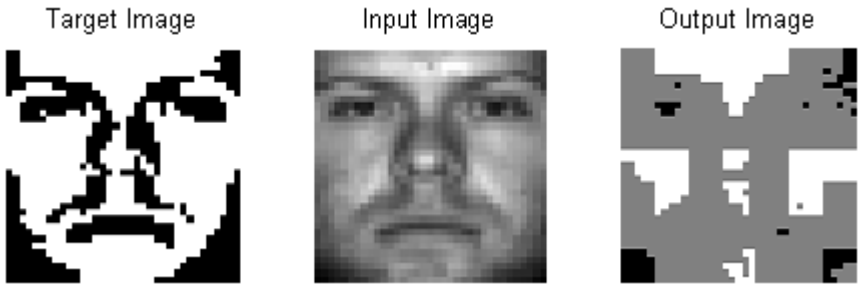


Figure 4.14: Results of the primary test case in grey-scale to binary conversion, implemented with a GMLP core utilizing full recurrency, trained with the EKF algorithm set to run for 500 epochs. The network achieves a final IM_{ACC} = 27.5% and an IM_{CR} = 46%.

4.6 Spatial Filtering

4.6.1 Introduction

Spatial filtering of images refers to filtering techniques that operate directly on the pixels of an image [31]. These operations usually involve the pixels in a *neighborhood* surrounding the pixel of interest. This neighborhood is referred to as the filter *window*. To perform the filtering operation, a filter *mask*, the size of the filter window, is moved across each pixel in the image. The coefficients of the filter mask are selected based on the desired characteristics of filter being implemented. The response of the filter at a given pixel location is the discrete convolution of the filter mask with the underlying sub-image. Figure 4.15 depicts the mechanics involved. Figure 4.15a, depicts the image to be filtered, $I_1(x,y)$. Figure 4.15b, shows the resulting filtered image, $I_2(x,y)$. The grey area in image I_1 represents the filter window at location (x,y). The corresponding response of the filter at location (x,y) in I_2 is also shown in grey. Figure 4.15c depicts the underlying sub-image of I_1, while Fig.4.15d shows the filter mask.

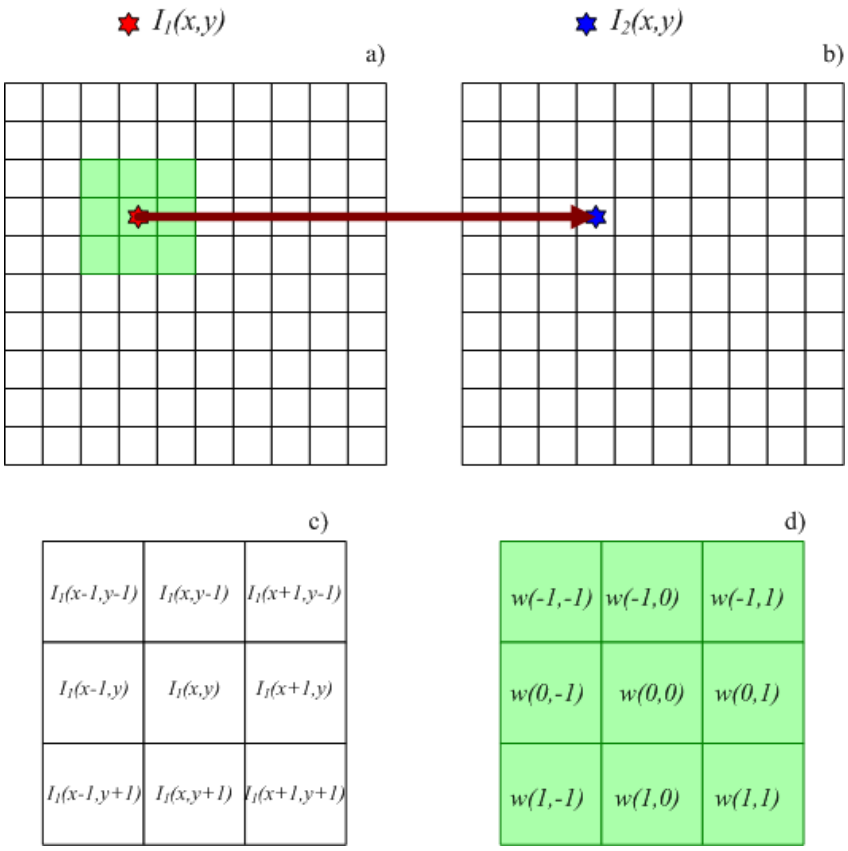


Figure 4.15: Mechanics of spatial filtering of images. a) input image. b) output image. c) input sub-image at location (*x*,*y*). d) filter mask.

This filtering process is a 2-D, discrete convolution of the input image with the filter mask. Mathematically it can be described as,

$$I_2(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} w(i,j) \cdot I_1(x+i, y+j), \qquad (4.3)$$

$$x = 1 \text{ to } N \quad \text{and} \quad y = 1 \text{ to } M,$$

where $a = (m-1)/2$, $b = (n-1)/2$, and the filter mask, $w(i,j)$ is of size *m x n*. $I_1(i,j)$ and $I_2(i,j)$ are the input and filtered images, respectively. Each is of size *NxM*.

In is common in practice to simplify (4.3) to

$$R_{x,y} = \sum_{i=1}^{n \cdot m} w_i \cdot p_i, \tag{4.4}$$

where $R_{x,y}$ is the filter response at location (x,y), w_i is the i^{th} coefficient of the filter mask, and p_i is the intensity value of the corresponding image pixel. The mask is of size m x n, resulting in a total of $m \cdot n$ mask coefficients.

Spatial filtering requires padding to properly handle boarder pixels [33]. The amount of padding depends on the size of the filter's mask and is determined by

$$P_{size}^{X} = \frac{n-1}{2} \quad , \quad P_{size}^{Y} = \frac{m-1}{2} \tag{4.5}$$

where the mask size is [m x n], P_{size}^{X} is the amount of padding in the x dimension and P_{size}^{Y} and is the amount of padding in the y dimension.

4.6.2 Low-Pass Filtering using CSRN

In this experiment, we configure and train the CSRN to implement a simple type of low-pass filter (LPF) known as an *averaging* filter.

4.6.2.1 The Averaging Filter

Figure 4.16 shows the filter mask for an averaging filter.

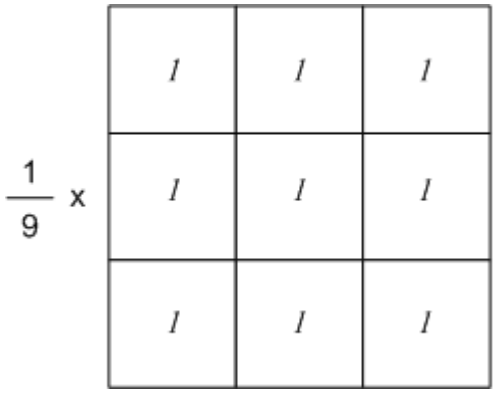


Figure 4.16: Filter mask for a *3x3* averaging filter.

In this case, (4.4) reduces to

$$R_{x,y} = \frac{1}{9}\sum_{i=1}^{9} p_i, \tag{4.6}$$

which is simply the average pixel value of the filter window. This is, of course, how the averaging filter gets its name.

4.6.2.2 CSRN Implementation of the *Averaging* Filter

To implement the averaging filter, we utilize a GMLP core with 5 active nodes. The filter coefficient, $\theta = 1/9$, is passed to the network as the sole external input. The pixel inputs include the cells corresponding pixel value along with the pixel values of its 8-N's. No neighbor inputs or self-recurrent inputs are used, and the number of core iterations is set to 1, eliminating all recurrency in the network. Figures 4.17 and 4.18 show the cell

configuration and core network of the CSRN configured to implement the averaging filter.

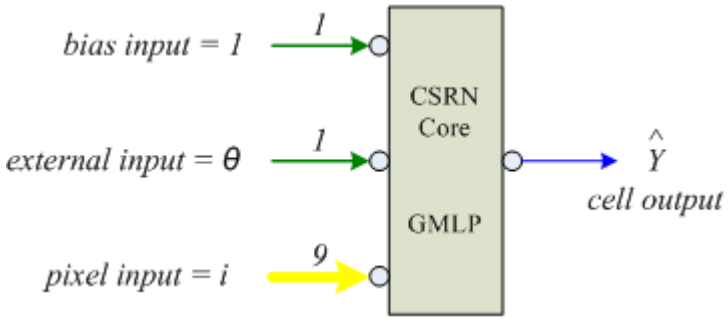


Figure 4.17: Cell configuration diagram for the generalized CSRN scaled down to implement a simple averaging filter.

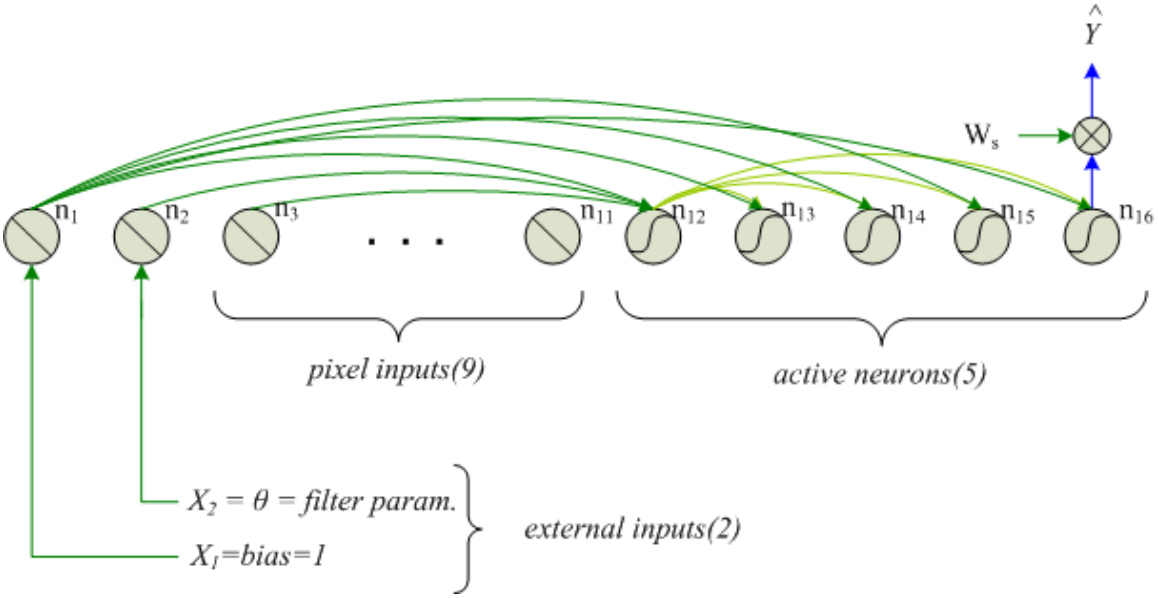


Figure 4.18: The network core used to implement a simple averaging filter. Core network is a GMLP configure with 5 active neurons. It utilizes 10 external inputs which include a bias input, a filter parameter, $\theta = 1/9$. It includes a full set of pixel inputs, the cells pixel value plus those of its 8-N's. No neighbor or self-recurrent inputs are used, and the number of core iterations is set to 1. The scaling weight is set to Ws = 1. The network uses a total of 16 nodes.

4.6.2.3 Low-Pass Filter Results

In this section, we examine the results of the CSRN implementation of the low-pass, averaging, filter described above. The network is trained via the unscented Kalman filter method described in Section 8.6. A fitter parameter of $\theta = 1/9$ is used. Target images are generated using MATLAB'S® image filtering function, *imfilter( ).* In the first experiment, we train the CSRN using 11 facial images taken from the Yale Face Database B [27]. Testing is done using the training set to evaluate how well the CSRN learns. In the second experiment, testing is performed using an independent testing set, also consisting of 11 images from the same dataset, to evaluate how well the CSRN generalizes. In each case, a single testing image is selected as a primary test case (PTC) and is displayed in greater detail.

4.6.2.3.1 Experiment 1: Testing with Training Set

Figure 4.19 shows the image results of *Experiment 1*. Row a) shows the input images. Row b) shows the corresponding CSRN output images. Row c) shows the target images produced by *imfilter( )*. Each column contains the results for a specific input image. The PTC is indicated and shown in detail in Fig. 4.20. Table 4.9 tabulates the results of this experiment.

Figure 4.19: Results of CSRN implementation of a LPF. Network utilizes a GMLP core with 5 active nodes, trained via UKF. Neighbor and self-recurrent inputs have been eliminated and all recurrency turned off. Ten external inputs are used, the threshold parameter, θ = 1/9, the cell's corresponding pixel intensity, and the pixel intensities of its 8-Ns. Results are shown for testing with the full training set. Row a) shows the input images. Row b) shows the corresponding CSRN output images. Row c) shows the target images produced by *imfilter( )*. The primary test case is indicated.

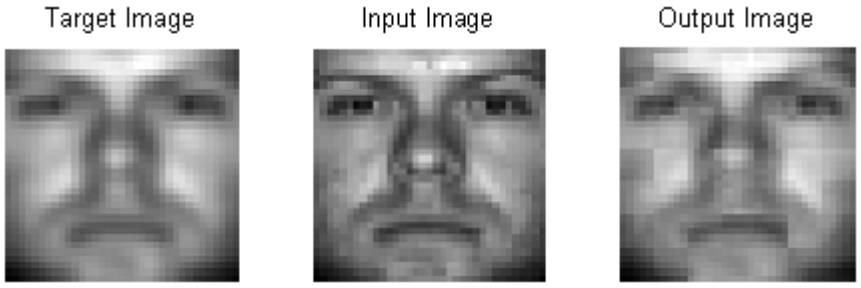


Figure 4.20: Results for primary test case in LPF *Experiment 1*, testing with training set. The network achieves a final IM_{MSE} = 5.0E-04 and an IM_{CR} = 98.2%.

Table 4.9: Results of LPF implementation. Data from testing with full training set shown. PTC is highlighted in red.

	***LPF Experiment 1* Full Test Results**	
Test Image	**IM_{MSE}**	**IM_{CR}**
1	5.00E-04	0.982
2	1.10E-03	0.960
3	1.30E-03	0.974
4	9.00E-04	0.961
5	1.50E-03	0.971
6	9.00E-04	0.972
7	8.00E-04	0.970
8	7.00E-04	0.982
9	8.00E-04	0.982
10	2.20E-03	0.975
11	7.00E-04	0.981

An inspection of Figs. 4.19 and 4.20 show that the output images produced by the CSRN are very similar to the baseline images produced by MATLAB'S® *imfilter( )* function. This is supported by the metrics in Table 4.9. The CSRN achieves an average image accuracy, IM_{CR} = 97.4% over all training images and a best case IM_{CR} = 98.2%. These results indicate that the CSRN has learned to perform LPF for the training set.

4.6.2.3.2 Experiment 2: Testing with an Independent Test Set

In this experiment, we repeat *Experiment 1* using an independent testing set. Figure 4.21 shows the image results for this experiment. Row a) shows the input images. Row b) shows the corresponding CSRN output images. Row c) shows the target images

produced by *imfilter( )*. Each column contains the results for a specific input image. The PTC is indicated and shown in detail in Fig. 4.22. Table 4.10 tabulates the results of this experiment.

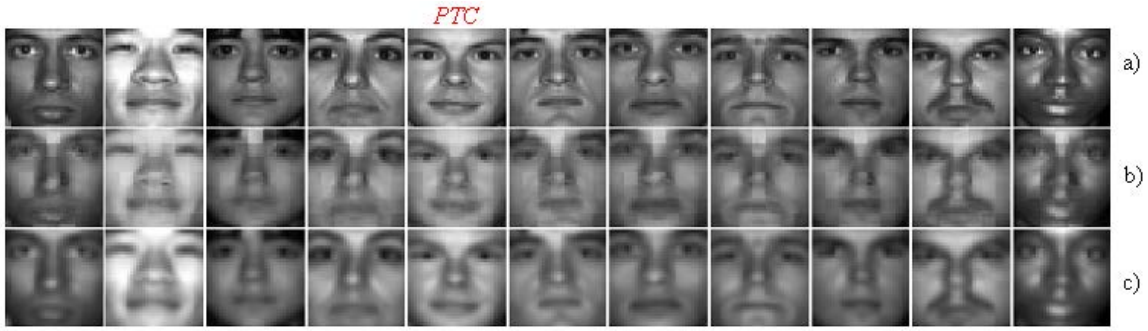


Figure 4.21: Results of CSRN implementation of an averaging, LPF. Network utilizes a GMLP core with 5 active nodes, trained via UKF. Neighbor and self-recurrent inputs have been eliminated and all recurrency turned off. Ten external inputs are used, the threshold parameter, θ = 1/9, the cell's corresponding pixel intensity, and the pixel intensities of its 8-Ns. Results are shown for testing with an independent test set Row a) shows the input images. Row b) shows the corresponding CSRN output images. Row c) shows the target images produced by *imfilter( )*. The primary test case is indicated.

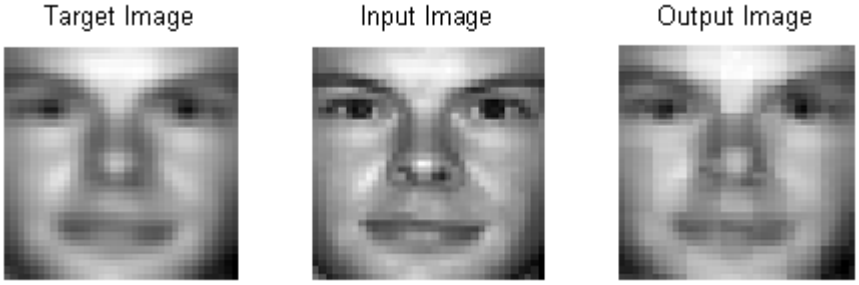


Figure 4.22: Results for primary test case in LPF *Experiment 2*, testing with independent test set. The network achieves a final IM_{MSE} = 7.00E-04 and an IM_{CR} = 98.7%.

Table 4.10: Results of LPF implementation *Experiment 2* testing with independent test set.

	***LPF Experiment 2* Full Test Results**	
Test Image	**IM_{MSE}**	**IM_{CR}**
1	1.80E-03	0.959
2	6.20E-03	0.954
3	2.50E-03	0.956
4	1.30E-03	0.962
5	7.00E-04	0.987
6	1.00E-03	0.968
7	1.50E-03	0.959
8	1.40E-03	0.959
9	1.30E-03	0.977
10	9.00E-04	0.978
11	2.10E-03	0.966

Table 4.11 compares the results of the two experiments. The CSRN performs well with the independent test set, achieving an average correlation ratio, IM_{CR} = 96.6% over all training images and a best case IM_{ACC} = 98.7%. As expected, the CSRN tests better with the training set than with the independent testing set. However, the results with the independent test set measure to within 1.7% of the results with the training set indicating that the CSRN generalizes well in our LPF application.

Table 4.11: Comparison of testing results for the CSRN implementation of LPF. Results for testing with training set and independent testing set are compared.

LPF Testing Comparison						
Metric	**Units**		***Training***	***Testing***	***Diff.***	***% Diff.***
IM_{MSE}	-	*Ave*	1.04E-03	2.07E-02	5.10E-03	13.2
		Best	5.00E-04	7.00E-04	-9.57E-03	-56.0
IM_{CR}	-	*Ave*	0.974	0.966	-0.015	-1.7
		Best	0.982	0.987	0.020	2.1

4.6.2.3.3 Conversion, Training and Computation Times

Figure 4.23, shows plots of the MSE as a function of epochs for testing the CSRN with both the training set and the testing set. The results are those generated in the PTC of each experiment. In both cases the learning processing is stable and converges in less than 15 epochs.

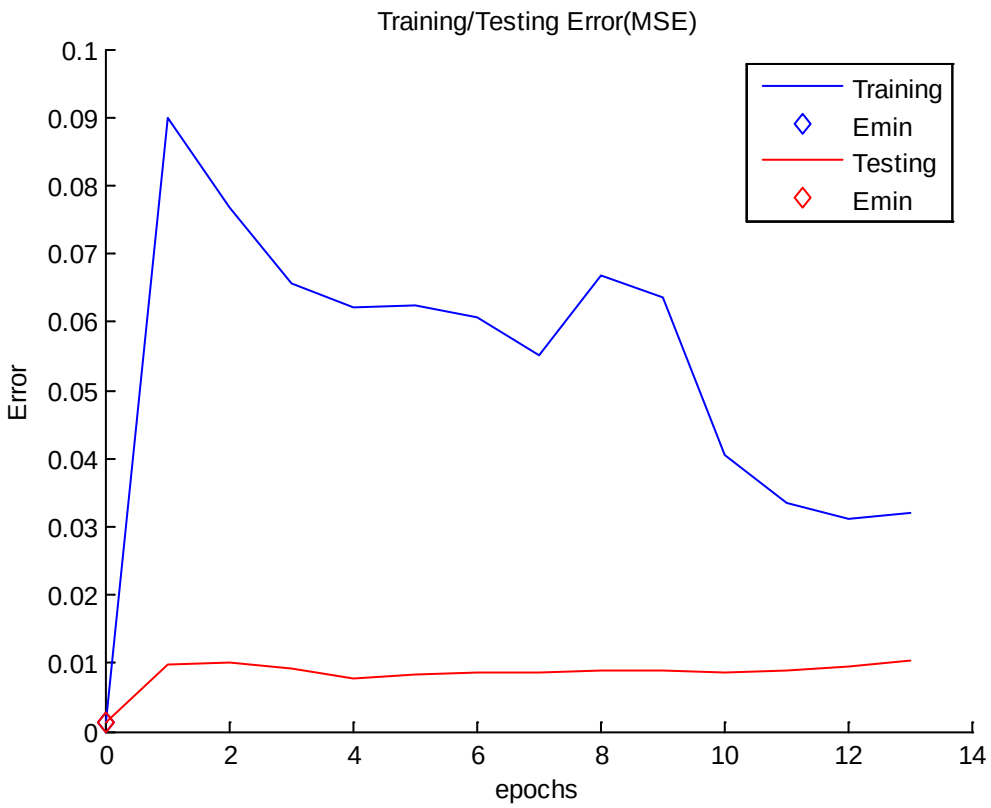


Figure 4.23: Plots of training/testing MSE vs. epochs for the CSRN trained to perform LPF. The blue line is the error for the training set. The red line is that for the testing set. Diamonds indicate the minimum error. Plots represent the PTC for each experiment.

Table 4.12 lists the training, computation and run times for the CSRN implementation of LPF. For the LPF application with 11, *35x35*, training images the total training time is just over 42.5 min. A normalized training time is also given. For this metric, the training time is normalized for image size and number of training images. Once trained the CSRN performs the LPF transformation in just 12.0 msec. The total run time for a typical simulation of the LPF application is approximately 4 hr. The total run time may vary depending on how quickly the network converges.

Table 4.12: Computation times for CSRN implementation of LPF.

		CSRN LPF Implementation
Metric	**Units**	
T_{TR}**(total)**	*sec*	2554
T_{TR}**(norm)**	*msec*	189.5
T_{FC}	*msec*	11.86
T_{R}	*min*	149

4.7 Geometric Transformations

A fundamental challenge to the theory underlying NNs, posed by Rosenblatt in his early work on perceptrons, is the recognition of topological relations [80]. Minsky *et al*. show that perceptrons are incapable of solving this class of problems due to their exponential complexity [61]. They also demonstrate that multi-layered perceptrons (MLPs), in spite of being more powerful than perceptrons, are also unable to solve topological relation problems [62]. In the following two chapters we discuss two geometric transformations, affine transformation (linear) and image registration (non-linear), which fall within this class of IP problems.

4.8 Conclusion

In this chapter, pursuant to *goal 1* of this work, we present a flexible, generalized CSRN architecture for IP and demonstrate its efficacy in performing both pixel level operations and spatial filtering. We also present metrics for measuring such efficacy in IP applications. In addition we adapt standard sub-image processing techniques for use with the CSRN, enabling the CSRN to perform IP tasks on larger, more interesting images.

In the case of pixel operations, we implement G2BC using a CSRN. Tests for this application show that CSRN performs well on the training set, indicating that the CSRN has indeed learned to perform G2BC. In fact the CSRN performs to within 0.5% of the corresponding MATLAB baseline. The CSRN also performs well on the independent testing set, indicating that generalizes well. The learning process is stable, converging in less than 15 epochs.

In the case of spatial filtering, we implement a simple averaging LPF. Tests for this application show that CSRN learns to perform LPF to within 1.7% of the corresponding MATLAB baseline. The CSRN also generalizes well in this application. Similar to the G2BC case, the learning process for LPF is stable and converges in less than 15 epochs.

The application of sub-image processing is fundamental to the practical application of CSRNs to image IP tasks. Prior to this work, application of CSRNs to IP tasks was limited to an image size of 7x7 pixels. Our best efforts, without the use of sub-image processing, extended this size to 15x15 pixels. The introduction of sub-image processing breaks this size barrier making the application of CSRNs to IP tasks tractable. Though we limit image size to 35x35 pixels in this chapter, In Chapter 5, we obtain results for image sizes up 125x125 pixels.

5 AFFINE TRANSFORMATIONS USING CSRNS

5.1 Affine Transformation of Images

Affine transformations are an important class of linear, 2-D, geometric transformation, which map pixel intensity values from their location in an input image to their new location in an output image [24]. These transformations are characterized by their preservation of straight lines within the image.

Affine transformations have been well documented. Virtually any image processing or computer vision text will thoroughly cover the topic. Excellent reviews of affine transformation can be found in Jain [49] and Fisher *et al*. [24]. Affine transformations are primarily used in image editing and play a crucial role in many image registration techniques.

5.1.1 Types of Affine Transformations

Affine transformations in images consist of translation, rotation, scaling, and vertical/horizontal shear. Since vertical and horizontal shear are simple cases of scaling in a single dimension, we focus our attention on the three remaining transformations: translation, rotation and scaling.

The general form of an affine transformation is that of a 2-D, linear equation.

$$X_2 = A \cdot X_1 + B \tag{5.1}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{5.2}$$

.

where X_1 is the current pixel's location and X_2 is it's new location. A and B are the slope and intercept matrices, respectively. In the case of pure transformations, we can further reduce the form to

$$X_2 = T \cdot X_1 \tag{5.3}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \tag{5.4}$$

.

where T is referred to as the transformation matrix.

5.1.1.1 Translation

The equation for translation is given by

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}, \tag{5.5}$$

where x_0 and y_0 are the amount of translation in the x and y directions, respectively [31]. By augmenting B to A in (5.1) we can achieve the form in (5.3).

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_2 \\ 1 \end{bmatrix} \tag{5.6}$$

5.1.1.2 Rotation

The equation for rotation is given by

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} . \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \tag{5.7}$$

where θ is the amount of rotation about the center of the image [31].

5.1.1.3 Scaling

The equation for scaling is given by

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} . \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \tag{5.8}$$

where s_x and s_y are the scale parameters in the x and y directions, respectively [31].

5.1.2 Image Re-sampling

Image re-sampling is an important part of image transformation. Re-sampling includes two related topics: mapping methods and interpolation. Several surveys have been conducted on image re-sampling methods. A thorough comparison of these techniques is done in [67].

5.1.2.1 Mapping Methods

The affine functions above can be implemented by forward or inverse computation. In the forward method, each pixel in the image is transformed to its new location via the

forward equations (given above). This method produces significant error in the new image due to overlaps and/or holes caused by discretization and rounding [102]. In the inverse method, the coordinates of the new image are used along with inverse equations. In this way, one and only one value is calculated for each pixel position in the new image, thus eliminating overlaps and holes.

5.1.2.2 Interpolation

Mapping functions, whether forward or inverse, result in fractional pixel locations. That is, pixels in the first image, do not map directly onto pixels in the second image. To complete the mapping step we must perform some type of interpolation. Nearest neighbor, bilinear, bi-cubic, quadratic splines, cubic B-splines are a few of the more popular methods. Even though higher order methods produce increased accuracy and better visual results, bilinear interpolation offers possibly the best trade-off between accuracy and computational complexity [67].

Since the emphasis here is placed on the efficacy of the CSRN to perform affine transformations, we consistently use the nearest neighbor method for its ease of implementation.

5.1.3 General Steps for Affine Transformation of Images

Affine transformation can be performed using the following steps:

1. identify the type and amount of transformation to perform
2. select mapping and interpolation methods
3. perform desired transformation
4. perform interpolation
5. repeat steps 3 and 4 for all pixels.

5.1.4 Affine Transformation using Neural Networks

A fundamental challenge to the theory underlying NNs, posed by Rosenblatt in his early work on perceptrons, is the recognition of topological relations [80]. Minsky *et al*. show that perceptrons are incapable of solving this class of problems due to their exponential complexity [61]. They also demonstrate that multi-layered perceptrons (MLPs), in spite of being more powerful than perceptrons, are also unable to solve topological relation problems [62]. Affine transformations fall within this class of image processing problems. While translation is a simple one-to-one mapping, rotation and scaling are more complex geometric operations. MLPs and, in general, feed-forward NNs cannot perform these complex geometric transforms on images.

CNNs are capable of performing fractional and single pixel translation [17][26]. By extension, the CNN can perform rotation by first decomposing the rotation into multiple, single-pixel translations and one, fractional-pixel translation, then applying the CNN to the image once for each single pixel or fractional translation. This approach does not lend itself to practical application. In addition, the CNN relies on predetermined weights, referred to as templates [13][14][101], and therefore, does not learn to perform these transformations.

5.2 Affine Transformation via CSRN

5.2.1 Adapting the CSRN for Affine Transformations

Since the CSRN was initially created to solve the maze traversal problem, we must adapt the network to perform image processing tasks, in this case, affine transformation.

5.2.1.1 Inputs/Outputs

An examination of the equations for affine transformations given above, shows that these transformations require the following inputs:

- *Pixel location*: P_x and P_y are the *x* and *y* locations of the pixel within the image.
- *Transformation parameter*: θ, is the amount of transformation to be performed. This may be the number of pixels to translate an image, the angle of rotation or a scaling factor.

The output of the network depends on the chosen mapping method. In the case of forward mapping, the output will be the pixel's (new) location in the transformed image. In the case of inverse mapping, the location of the pixel within the transformed image is known, and the network outputs the pixel's location in the original image.

5.2.1.2 Cost-Function of Affine Transformation

Before adapting the CSRN for any image processing application, we must 1st consider whether the cost-function associated with learning in the CSRN is capable of approximating the cost-function associated with the given application. Let us consider the cost-function for affine transformation of an image. In this work, we treat images as rigid bodies. From (5.1) we see that X_2 represents the output computed by the general equation of an affine transformation. Let X_2^i represent the computed output, i.e. the i^{th} pixel's computed location in the transformed image. Let X_T^i represent the target (true) location of the pixel in the transformed image. The error between the target and calculated locations is given by,

$$e_i = X_T^i - X_2^i. \tag{5.9}$$

The total error in the transformed image is given by,

$$E_i = \frac{1}{2}\sum_{i=1}^{c} e_i^2 \tag{5.10}$$

where c is the number of pixels in the image. If we extend this error to an entire set of images, the error becomes,

$$E = \sum_{n=1}^{N}\left[\frac{1}{2}\sum_{i=1}^{c} E_i^2\right] \tag{5.11}$$

where N is the number of images in the set. Equation (5.11) represents the general cost-function for affine transformation. Minimizing this cost function minimizes the Euclidean distance between the computed the pixels' locations and their target locations. Note this cost-function is equivalent to (3.3), which is the cost-function for learning in CSRNs discussed in Section 3.3. This suggests that the CSRN should be capable of learning affine transformations.

5.2.2 Implementation Issues

5.2.2.1 Training

To train the CSRN to perform a given affine transformation, we generate a set of training images by transforming a test image by various degrees. For example, in the case of rotation, we might use 0°, 5°, 10º, 15º, and 20º to produce a training set consisting of 5 images. For each training images we construct two transformation matrices, which encode the transformation for the x and y dimensions, respectively. These matrices can

use position or movement encoding. Transformation matrices utilizing position encoding contain the new position of the pixel, while those utilizing movement encoding contain the distance the pixel needs to be moved from its current location. In the course of our work, we find that the CSRN is better able to approximate movement encoded functions and utilize this type of encoding for results presented herein. The transformation matrices become the targets by which we train the CSRN.

5.2.2.2 Network Outputs

Because affine transformations in an image are separable in the x and y directions, we may accomplish them in two ways: 1) increase the number of network outputs to two, one each for the *x* and *y* locations, or 2) keep the same network configuration and apply it twice to the image; once for the *x* transformation then again for the *y* transformation. The later method is used in order to simplify the network and eliminate costly recoding.

5.2.2.3 Image Size

As image size grows, so does the complexity of the CSRN. In a practical implementation, a tradeoff exists between the number of training images, and the size of the images. For proof of concept we work with artificial binary images with a size of 15x15 pixels. This size image permits up to 11 training images, which allows for adequate results. When working with grey-scale images, we utilize sub-image processing, discussed in Chapter 4, which allows increased image size.

5.2.2.4 Input Image Padding

There are two issue encountered when performing affine transformations with CSRN's which require image padding.

1) *Geometric Transformations:* When performing any type of geometric transformation, including affine transformation or image registration (Chapters 6), it is necessary to pad the input images to prevent loss of information. For example, if the input image is transformed in such a way that a portion of the image falls outside the designated image size, any transformation method, including CSRNs, will be unable to reconstruct the original image. The amount of padding depends upon the geometry of the transform and the amount being performed.

Consider the case of affine rotation. Figure 5.1 shows a 25x25 face image rotated by an angle of 16°. The figure clearly indicates that the corners of the facial image have been rotated off the image.

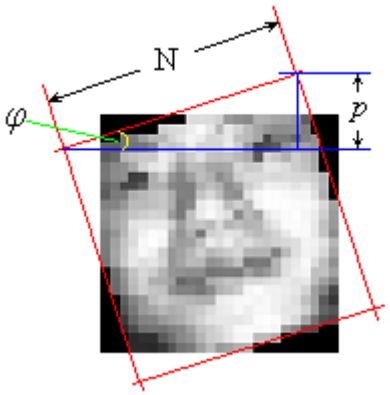


Figure 5.1: Geometry of rotation for a 25x25 facial image rotated by an angle of 16°.

Equation (5.12) can be used to compute the minimum amount of padding required to prevent loss of data.

$$p = N \cdot \sin(\phi), \tag{5.12}$$

where p is the pad size, N is the image size in pixels and ϕ is the angle of rotation in degrees. Figure 5.2a shows an early result of rotation via CSRN without image padding, it is clear that the output image produced by the CSRN suffers from loss of information in the corners of the image. In our rotation application we limit the amount of rotation to 20°. For the 25x25 image shown, (5.12) becomes

$$y = N \cdot \sin(\phi) = 25 \cdot \sin(20^\circ) = 8.55 \tag{5.13}$$

Figure 5.2b shows the results of the CSRN performing the same rotation with zero padding of 5 pixels (on all sides) applied to the input image. This provides a total padding of 10 pixels in both the x and y dims. As can be seen the CSRN's output image is dramatically improved. In both cases the CSRN learns the rotation transformation relatively well, achieving similar function accuracies, 91.4% and 92.1%, respectively. The addition of zero padding, improves the image accuracy from only 79.7% to 94.2%, and improvement of 15%.

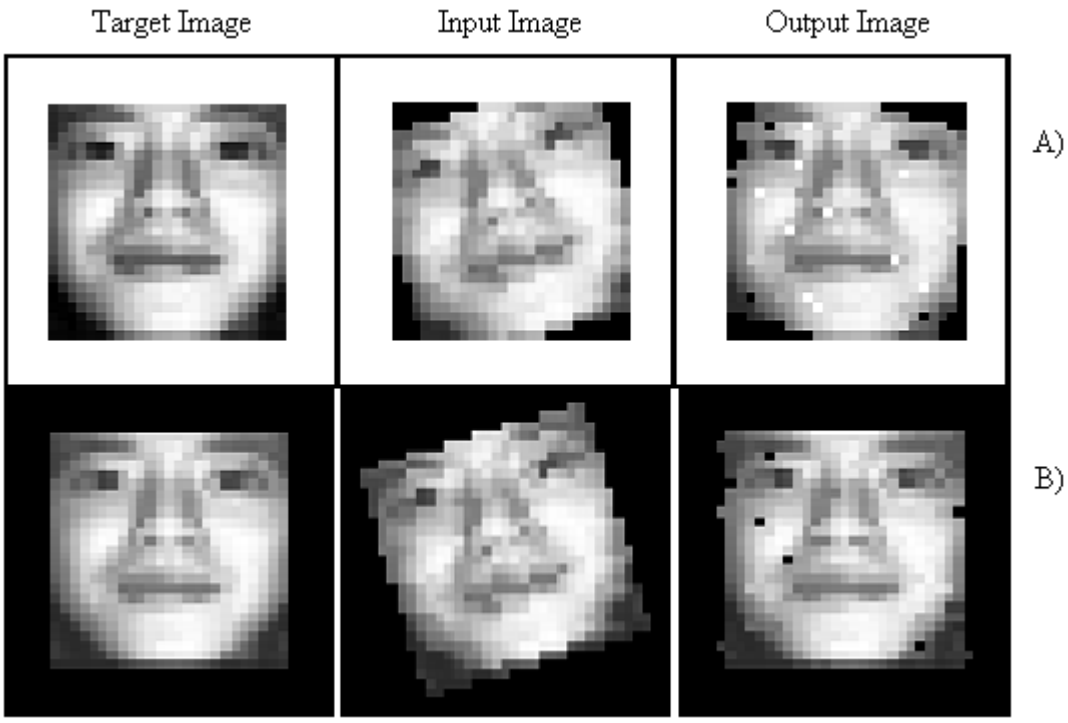


Figure 5.2: A 25x25 facial image rotated by an angle of 16° by the CSRN. A) results without padding. B) The results with a pad size of 5 pixels, increasing the overall image size to 35x35 pixels.

2) *Boundary pixel errors in affine function approximation:* It is insightful to examine the error between the CSRN's output and its training target. Table 5.1 shows this function error, for a typical affine transformation simulation. In this case, the number of training images and the number of epochs are limited in order to introduce additional error. Upon examination, the majority of these errors occur at boundary pixels. Zero padding the input image prevents these function approximation errors from affecting the final output image, thereby increasing its image accuracy. This explains the discrepancy between the function and image accuracies encountered later in this chapter. The pad size need only be a few pixels wide, and is easily accommodated using the pad size required by either of the previously mentioned issues.

Table 5.1: Typical function error for affine transformation (errors highlighted).

-2	-1	-1	-1	0	0	0	0	0	0	0	0	1	1	1
-1	0	0	0	0	0	0	0	0	0	0	0	1	1	1
-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5.2.2.5 Mapping Method

Mapping methods are discussed in Section 5.1.2.1, above. We experiment with both forward and inverse mapping. Figure 5.3 shows the results for two rotation simulations via CSRN. In both experiments, the CSRN performs a rotation of 16° on the same 35x35 face image. The experiment for Fig. 5.3a employs forward mapping, while that of Fig. 5.3b utilized inverse mapping. Note the missing and or misplaced pixels in the forward mapping case. Inverse mapping eliminates these anomalies.

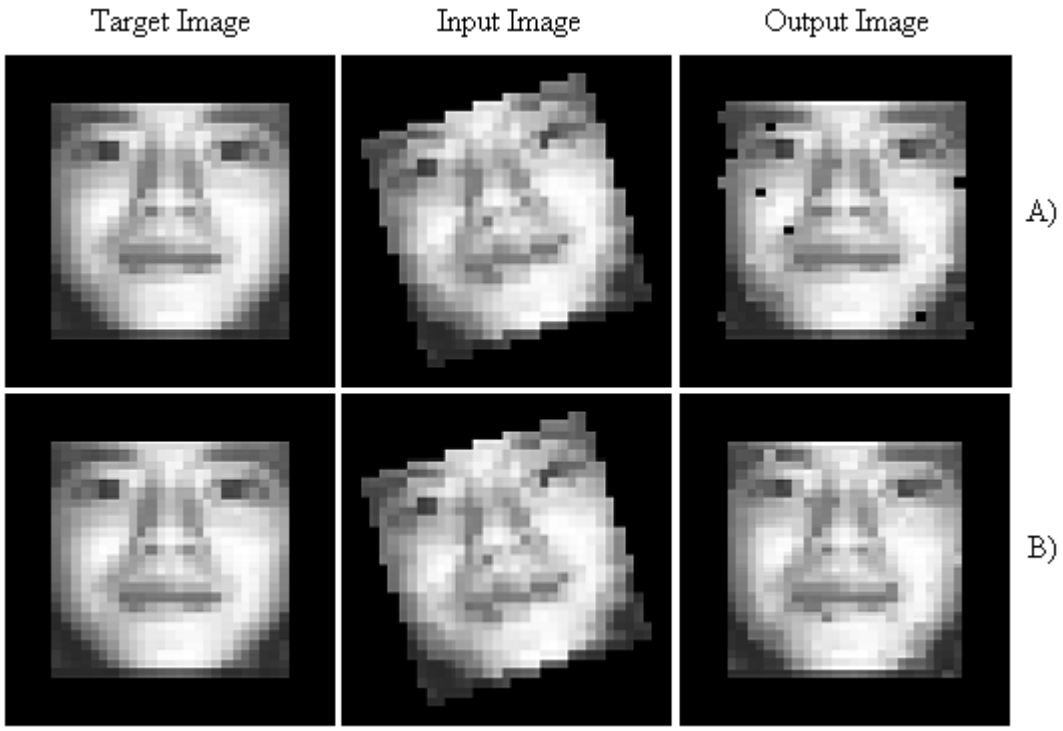


Figure 5.3: A 35x35 facial image rotated by an angle of 16° by the CSRN. A) Results for forward mapping. B) Results for inverse mapping.

As expected, the inverse method produces better results. Therefore, we utilize this mapping method exclusively in this work.

5.2.3 Transformation of Binary Images

In this section we examine the use of CSRNs to perform the following affine transformations on binary images: translation, rotation and scaling.

5.2.3.1 Experiment Configuration

In this series of experiments we utilize a simple binary image of a cross as our test subject. The CSRN is configured with a GMLP core, EKF training, movement encoding and inverse mapping. Two external inputs are used for the core network: the pixel's *x*

location, and the transformation parameter, θ. The network is trained with eleven images which are zero padded as discussed in Section 5.2.2.4. The network is tested with the full set of training images. A single testing image is selected as a primary test case (PTC) and is displayed for greater detail.

5.2.3.2 Translation

In this experiment we train the CSRN to perform affine translation on binary images. Training images utilize zero padding with a width of 5 pixels. The input images are translated through a range of $\theta' = 0$ to 10 pix and the network is trained to translate these images with a transformation parameter of $\theta = -\theta'$. Figure 5.4 shows the resulting images. Row a) shows the input images. Row b) and c) show the corresponding CSRN output images and the results of the raw translation (5.6), respectively. Row d) shows the actual target image. Each column contains the results for a specific input image, and is labeled by its corresponding transformation parameter, θ'. Note the PTC, $\theta' = 10$ pix, is highlighted in red, and is shown in detail in Fig. 5.5.

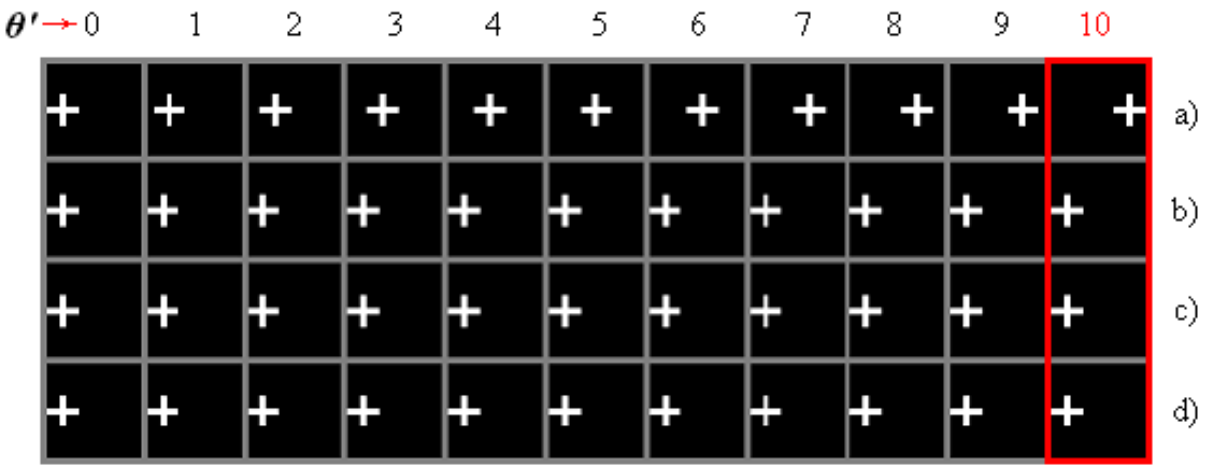


Figure 5.4: Results of affine translation test for binary images. Results are shown for full testing set. Row a): input images translated through a range of $\theta' = 0$ to 10 pix. Row b): corresponding CSRN output images. Row c): results of raw transformation by (5.6). Row d): target image. Columns label with transformation parameter of the input images, θ'. PTC, $\theta' = 10$ pix, is highlighted in red.

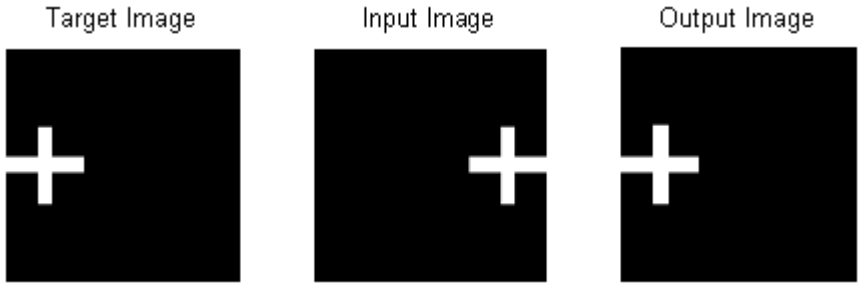


Figure 5.5: Results of binary affine translation test for PTC, $\theta' = 10$ pix. The network achieves a final $J_{ACC} = 100\%$ and an $IM_{ACC} = 100\%$.

Table 5.2 tabulates the results for our translation experiment. Results for the full testing set are shown. Statistics are computed over a batch of 50 simulations. The CSRN achieves a function accuracy, an image accuracy and an image correlation ratio of 100%, indicating that it has learned to perform affine translation perfectly.

Table 5.2: Binary image translation results for full test set.

	Binary Translation Results (full test set)				
$\boldsymbol{\theta'}$	$\boldsymbol{J_{MSE}}$	$\boldsymbol{J_{ACC}}$	$\mathbf{IM_{MSE}}$	$\mathbf{IM_{ACC}}$	$\mathbf{IM_{CR}}$
0	0.00	100.0	0.00	100.0	1.00
1	0.00	100.0	0.00	100.0	1.00
2	0.00	100.0	0.00	100.0	1.00
3	0.00	100.0	0.00	100.0	1.00
4	0.00	100.0	0.00	100.0	1.00
5	0.00	100.0	0.00	100.0	1.00
6	0.00	100.0	0.00	100.0	1.00
7	0.00	100.0	0.00	100.0	1.00
8	0.00	100.0	0.00	100.0	1.00
9	0.00	100.0	0.00	100.0	1.00
10	0.00	100.0	0.00	100.0	1.00

In this experiment, the CSRN correctly translates the cross image, achieving both a function accuracy and an image accuracy of 100%. These results indicate that the CSRN has learned to perform affine translation perfectly. Table 5.3 summarizes the results of the translation experiment for binary images.

Table 5.3: Summary of binary image translation results.

		Binary Translation	
Metric	**Units**	*Ave +/- std dev*	*PTC:* θ' =10 pix
J_{ACC}	*%*	100 +/- 0.0	100
J_{MSE}	-	0.00 +/- 0.00	0.0
IM_{ACC}	*%*	100 +/- 0.0	100
IM_{MSE}	-	0.00 +/- 0.00	0.0
IM_{CR}	-	1.0 +/- 0.0	1.0
T_{TR}**(total)**	*sec*	568 +/- 0.35	567
T_{TR}**(norm)**	*msec*	229 +/- 0.16	229
T_{FC}	*msec*	1.47 +/- 0.01	1.47
T_R	*min*	9.72 +/- 0.01	9.7
T_B	*hrs*	*n/a*	8.12

5.2.3.3 Rotation

Translation is local in nature; however, rotation is global, resulting in a more complex transformation. In this experiment we train the CSRN to perform affine rotation on binary images. Training images utilize zero padding with a width of 5 pixels. The input images are translated through a range of θ' = 0° to 20° in steps of 2° and the network is trained to rotate these images with a transformation parameter of $\theta = -\theta'$. Figure 5.6 shows the resulting images. Row a) shows the input images. Row b) and c) show the corresponding CSRN output images and the results of the raw rotation (5.6), respectively. Row d) shows the actual target image. Each column contains the results for a specific input image, and is labeled by its corresponding transformation parameter, θ'. The PTC, θ' = 16°, case is highlighted in red, and shown in detail in Fig. 5.7.

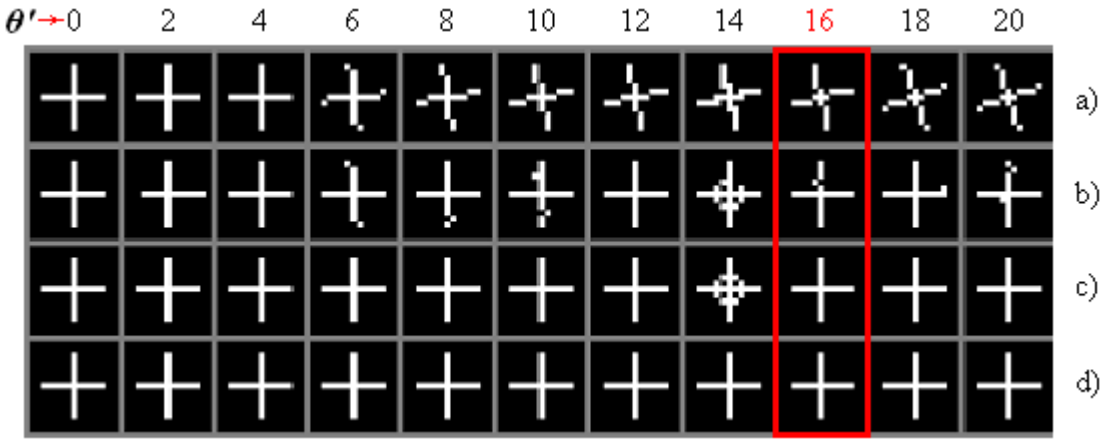


Figure 5.6: Results of affine rotation test for binary images. Results are shown for full testing set. Row a): input images rotated through a range of $\theta' = 0°$ to 20° in steps of 2°. Row b): corresponding CSRN output images. Row c): results of raw transformation by (5.7). Row d): target image. Columns label with transformation parameter, θ'. PTC, $\theta' = 16°$, is highlighted in red.

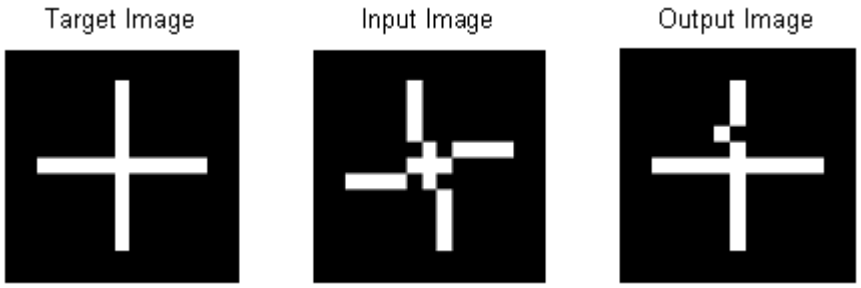


Figure 5.7: Results of affine rotation test for PTC, $\theta' = 16°$. The network achieves a best, $J_{ACC} = 82.6\%$ and an $IM_{ACC} = 94.2\%$.

Table 5.4 tabulates the results for our rotation experiment. Results for the full testing set are shown. Statistics are computed over a batch of 50 simulations. The PTC, $\theta' = 16°$, case is highlighted.

Table 5.4: Binary image rotation results for full test set.

	Binary Rotation Results (full test set)				
θ'	J_{MSE}	J_{ACC}	IM_{MSE}	IM_{ACC}	IM_{CR}
0	7.30E-03	92.0	0.00E+00	100.0	1.0000
2	5.30E-03	94.2	8.90E-03	99.1	0.9475
4	8.90E-03	90.2	0.00E+00	100.0	1.0000
6	9.70E-03	89.3	1.78E-02	98.2	0.8950
8	6.90E-03	92.4	8.90E-03	99.1	0.9475
10	4.40E-03	95.1	1.33E-02	98.7	0.9232
12	4.00E-03	95.6	0.00E+00	100.0	1.0000
14	5.30E-03	94.2	3.11E-02	96.9	0.8510
16	5.30E-03	94.2	8.90E-03	99.1	0.9475
18	6.50E-03	92.9	4.40E-03	99.6	0.9746
20	8.10E-03	91.1	1.33E-02	98.7	0.9232

In this experiment, the CSRN learns to rotate the binary cross image. It achieves a function accuracy of 94.2%, and an image accuracy of 99.1%. Note the lower function accuracy as compared to that for translation. This indicates that the CSRN has more difficulty in learning the rotation transformation. This is due to the global nature of rotation. In spite of this, the CSRN is still able to achieve excellent image accuracy. These results indicate that the CSRN has learned to perform affine rotation. Table 5.5 summarizes the results of the rotation experiment for binary images.

Table 5.5: Summary of binary image rotation results.

		Binary Rotation	
Metric	**Units**	*Ave +/- std dev*	*PTC:* θ' =16°
J_{ACC}	*%*	82.62 +/- 7.18	94.2
J_{MSE}	-	0.02 +/- 0.01	0.01
IM_{ACC}	*%*	99.02 +/- 0.57	99.1
IM_{MSE}	-	0.01 +/- 0.01	0.01
IM_{CR}	-	0.94 +/- 0.03	.95
T_{TR}**(total)**	*sec*	576.21 +/- 0.62	576
T_{TR}**(norm)**	*msec*	232.76 +/- 0.25	232
T_{FC}	*msec*	1.62 +/- 0.01	1.62
T_R	*min*	9.87 +/- 0.01	9.86
T_B	*hrs*	*n/a*	8.22

5.2.3.4 Scaling

In this experiment we train the CSRN to perform affine scaling on binary images. Similar to translation, scaling is a local transformation; however, it presents its own challenges. In the cases of translation and rotation, the final transformed image remains the same size as those of the training images, however, with scaling, the sizes of the final image and each individual training image are different. Our application was not coded to handle individual image sizes. We resolved this issue by zero padding each training image to make it the same size as the target image.

Due to the small image size, so much information is lost when scaling down the target image to produce the required input images, that when these images are up-scaled to produce the corresponding transformation matrices for network training, the raw scaling

function produces only blank images for scales above 2. Therefore, we limit the input images to scale factors of 0.5 to 1.0

In order to obtain a sufficient number of training images, we limit the input image scales, θ', to four scales, 0.6, 0.73, 0.87, and 1.0 and apply these scales to three different images, resulting in a total of 12 input images. These images are the cross, box, and simulated eye patch, shown in Fig. 5.8.

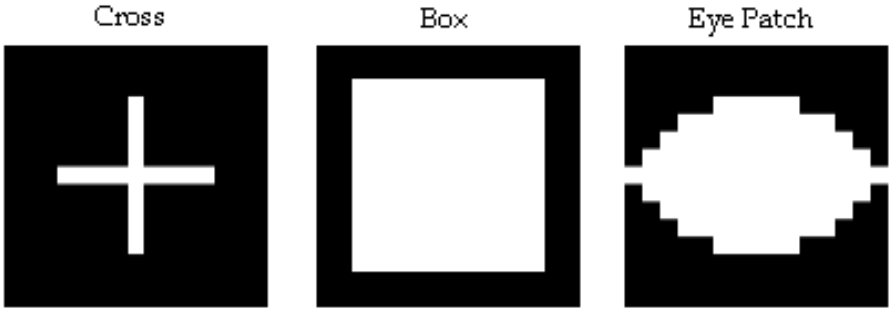


Figure 5.8: Images used for training the CSRN for up-scaling.

In this experiment we utilize two external inputs: the pixel's *x* location, and the transformation parameter, $\theta = 1/\theta'$. θ takes on values of 1.0, 1.15, 1.36, and 1.67. As described above, the network is trained with twelve images and tested with the full set of training images. Figure 5.9 shows the resulting images. Row a) shows the input images. Row b) and c) show the corresponding CSRN output images and the results of the raw rotation (5.6), respectively. Row d) shows the actual target image. Each column contains the results for a specific input image, and is labeled by its corresponding transformation parameter, θ'. The PTC, $\theta' = 0.73$, is highlighted in red, and shown in detail in Fig. 5.10.

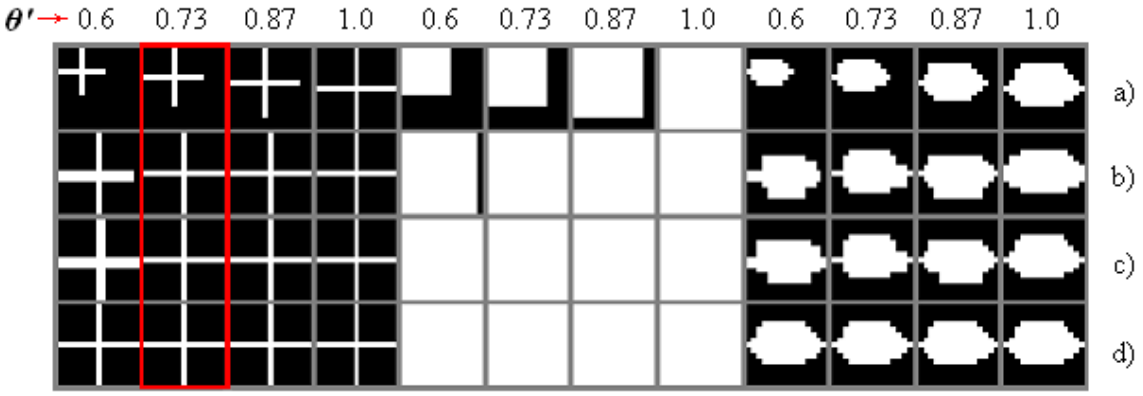


Figure 5.9: Results of affine scaling test for binary images. Results are shown for full testing set. Row a): input images scaled with scaling factor $\theta' = 0.6$ to 1.0 in steps of 0.8. Row b): corresponding CSRN output images. Row c): results of raw transformation by (5.7). Row d): target image. Columns label with transformation parameter, θ'. PTC, $\theta' = 0.73$, is highlighted in red.

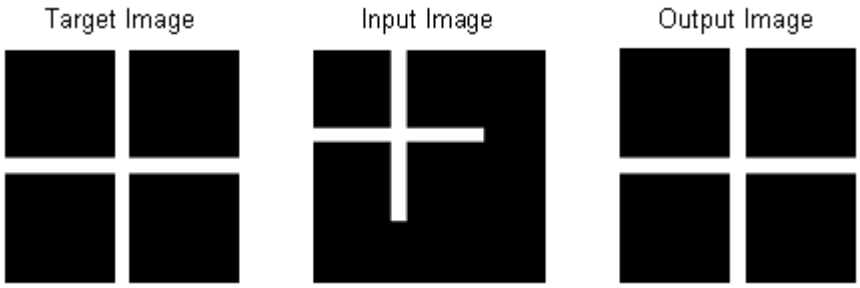


Figure 5.10: Results of affine scaling test for the PTC, $\theta' = 0.73$. The network achieves a best, $J_{ACC} = 81.8\%$ and an $IM_{CR} = 100\%$.

Table 5.6 contains the results for our binary scaling experiment. Results for the full testing set are shown. Statistics are computed over a batch of 50 simulations. The PTC, $\theta' = 0.73$, shown above is highlighted.

Table 5.6: Binary image scaling results for full test set.

	Binary Rotation Results (full test set)				
θ'	J_{MSE}	J_{ACC}	IM_{MSE}	IM_{ACC}	IM_{CR}
0.60	3.89E-02	53.3	6.22E-02	93.8	0.7805
0.73	0.00E+00	100.0	0.00E+00	100.0	1.0000
0.87	1.67E-02	80.0	0.00E+00	100.0	1.0000
1.00	3.33E-02	60.0	0.00E+00	100.0	1.0000
0.60	3.89E-02	53.3	6.67E-02	93.3	0.7805
0.73	0.00E+00	100.0	0.00E+00	100.0	1.0000
0.87	1.67E-02	80.0	0.00E+00	100.0	1.0000
1.00	3.33E-02	60.0	0.00E+00	100.0	1.0000
0.60	3.89E-02	53.3	1.11E-01	88.9	0.7663
0.73	0.00E+00	100.0	1.78E-02	98.2	0.9628
0.87	1.67E-02	80.0	8.00E-02	92.0	0.8335
1.00	3.33E-02	60.0	5.33E-02	94.7	0.8963

In this experiment, the CSRN is trained to perform scaling of binary images. The network achieves a best case result with J_{ACC} = 100%, IM_{ACC} = 100%, and IM_{CR} = 100%. The output images are properly scaled with some slight distortion in the θ = 1.67 case. Note the average function accuracy of 83.2%. This is primarily due to the lack of resolution in training scales, as discussed above. Table 5.7 summarizes the results of our binary scaling experiment.

Table 5.7: Summary of binary image scaling results.

		Binary Translation	
Metric	**Units**	*Ave +/- std dev*	*PTC:* θ' =0.73
J_{ACC}	*%*	83.20 +/- 8.11	100.0
J_{MSE}	-	0.01 +/- 0.01	0.00
IM_{ACC}	*%*	99.57 +/- 1.75	100.0
IM_{MSE}	-	0.00 +/- 0.02	0.00
IM_{CR}	-	0.98 +/- 0.08	1.00
T_{TR}**(total)**	*sec*	688.08 +/- 0.92	688
T_{TR}**(norm)**	*msec*	254.85 +/- 0.34	255
T_{FC}	*msec*	1.49 +/- 0.01	1.49
T_{R}	*min*	11.77 +/- 0.03	11.76
T_{B}	*hrs*	*n/a*	9.81

5.2.4 Transformation of Grey-Scale Images

In the previous section, we demonstrate the CSRN's capability of performing affine transformations in small binary images. What about more realistic images? In this section, we apply CSRNs to the affine transformation of larger, more interesting images, specifically, grey-scale facial images. To achieve this, we use the 25x25, facial, image shown in Fig 5.11. The image is zero-padded with a 5 pixel width, resulting in a 35x35 image. Sub-image processing, discussed in Section 4.2, is employed with a sub-image size of 5x5.

Figure 5.11: 25x25 face image used for testing affine transformations of grey-scale images.

5.2.4.1 Translation

In this experiment we train the CSRN to perform affine translation on grey-scale images. The input images are translated through a range of $\theta' = 0$ to 10 pix and the network is trained to translate these images with a transformation parameter of $\theta = -\theta'$. Figure 5.12 shows the resulting images. Row a) shows the input images. Row b) and c) show the corresponding CSRN output images and the results of the raw translation (5.6), respectively. Row d) shows the actual target image. Each column contains the results for a specific input image, and is labeled by its corresponding transformation parameter, θ'. The PTC, $\theta' = 10$ pixels, is highlighted in red, and shown in detail in Fig. 5.13.

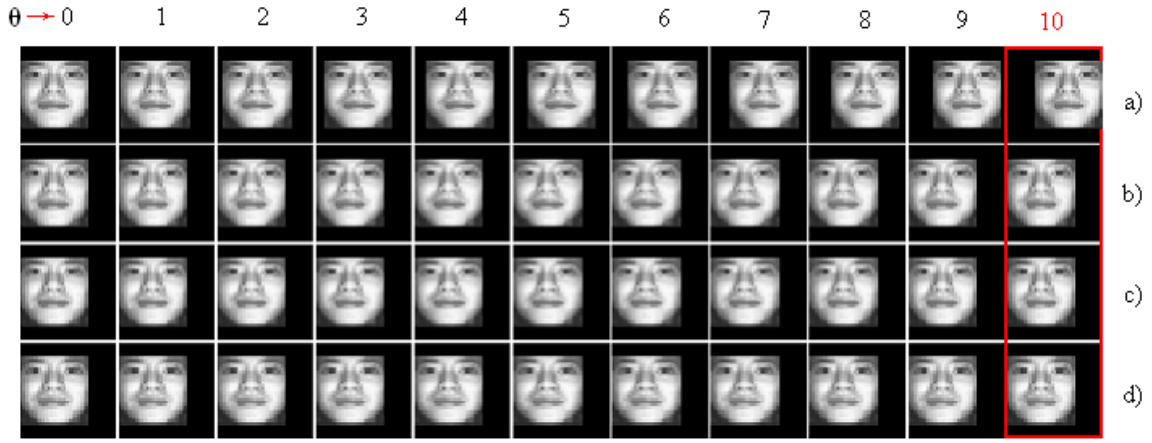


Figure 5.12: Results of affine translation test for grey-scale images. Results are shown for full testing set. Row a): input images translated through a range of $\theta = 0$ to 10 pix. Row b): corresponding CSRN output images. Row c): results of raw transformation by (5.6). Row d): target image. Columns label with transformation parameter, θ'. PTC, $\theta' = 10$ pix, is highlighted in red.

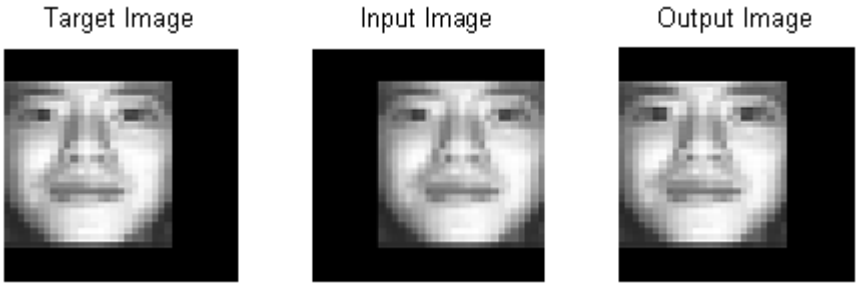


Figure 5.13: Results of affine translation test for the PTC, $\theta = 10$ pix. The network achieves a final $J_{ACC} = 100\%$ and an $IM_{CR} = 100\%$.

Table 5.8 tabulates the results for our translation experiment. Results for the full testing set are shown. The PTC is highlighted in red. The CSRN achieves a function accuracy, an image accuracy and an image correlation ratio all equal to 100%.

Table 5.8: Gray-scale image translation results for full test set.

	Grey-scale Translation Results (full test set)				
$\boldsymbol{\theta'}$	$\boldsymbol{J_{MSE}}$	$\boldsymbol{J_{ACC}}$	$\mathbf{IM_{MSE}}$	$\mathbf{IM_{ACC}}$	$\mathbf{IM_{CR}}$
0	3.71E-04	99.6	0.00	100	1.00
1	0.00	100	0.00	100	1.00
2	0.00	100	0.00	100	1.00
3	0.00	100	0.00	100	1.00
4	0.00	100	0.00	100	1.00
5	0.00	100	0.00	100	1.00
6	0.00	100	0.00	100	1.00
7	0.00	100	0.00	100	1.00
8	0.00	100	0.00	100	1.00
9	0.00	100	0.00	100	1.00
10	0.00	100	0.00	100	1.00

In this experiment, the CSRN correctly translates the face image, achieving a function accuracy of 100%, and an image correlation ratio of 100%. These results indicate that the CSRN has learned to perform affine translation perfectly. Table 5.9 summarizes the results of the translation experiment for grey-scale images.

Table 5.9: Summary of gray-scale image translation results.

		Grey-scale Images Under Translation
Metric	**Units**	*Primary Test Image:* θ' = 5 pixels
J_{ACC}	*%*	100
J_{MSE}	-	0.0
IM_{ACC}	*%*	100
IM_{MSE}	-	0.0
IM_{CR}	-	1.0
T_{TR}**(total)**	*sec*	835
T_{TR}**(norm)**	*msec*	62
T_{FC}	*msec*	
T_{R}	*min*	22

5.2.4.2 Rotation

In this experiment we train the CSRN to perform affine rotation. Three external inputs are used: the pixel's *x* and y locations, and the transformation parameter, θ. We use eleven training images, rotated from θ' = 0 to 20° in steps of 2°. The network is trained to rotate these images with transformation parameter, $\theta = -\theta'$. We test the network using its full set of training images. Figure 5.14 shows the resulting images. Row a) shows the input images. Row b) and c) show the corresponding CSRN output images and the results of the raw rotation (5.7), respectively. Row d) shows the actual target image. Each column contains the results for a specific input image, and is labeled by its corresponding transformation parameter, θ'. The PTC, θ' = 16° is highlighted in red, and shown in detail in Fig. 5.15.

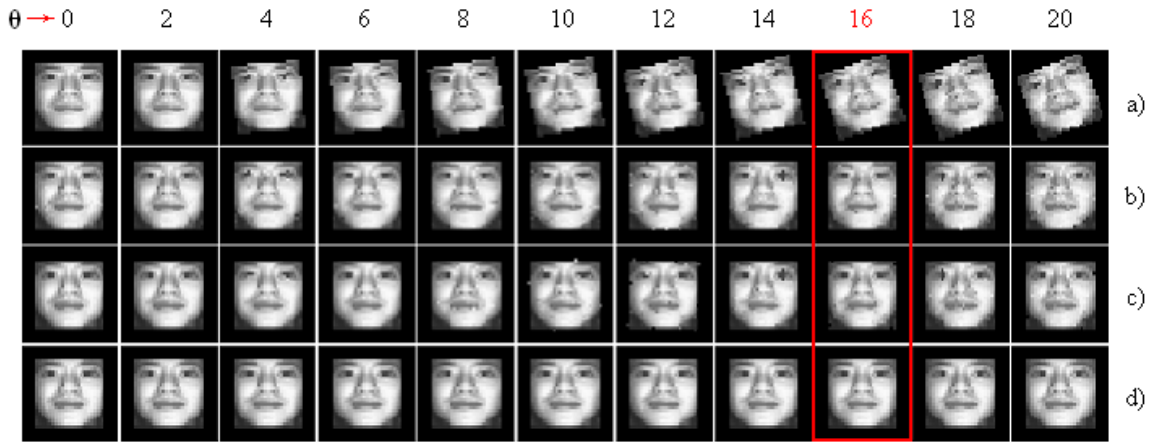


Figure 5.14: Results of affine rotation test. Results are shown for full testing set. Row a): input images rotated through a range of $\theta' = 0°$ to 20° in steps of 2°. Row b): corresponding CSRN output images. Row c): results of raw transformation by (5.7). Row d): target image. Columns label with transformation parameter, θ'. PTC, $\theta' = 16°$, is highlighted in red.

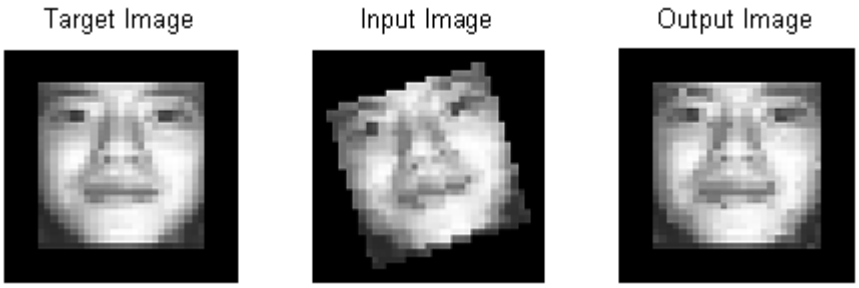


Figure 5.15: Results of affine rotation test for the PTC, $\theta' = 16°$. The network achieves a final, $J_{ACC} = 96.83\%$ and an $IM_{CR} = 100\%$.

The face image has clearly been rotated back to a zero degree angle. Note that the output image appears slightly blurred as compared to the target image. This primarily is due to information loss in the rotated input image. Table 5.10 contains the results of our rotation test for the full testing set. The PTC, $\theta' = 16°$, case is highlighted.

Table 5.10: Gray-scale image rotation results for full test set.

	Grey-scale Rotation Results (full test set)				
$\boldsymbol{\theta'}$	$\boldsymbol{J_{MSE}}$	$\boldsymbol{J_{ACC}}$	$\mathbf{IM_{MSE}}$	$\mathbf{IM_{ACC}}$	$\mathbf{IM_{CR}}$
0	6.46E-03	92.9	22.8	95.8	0.9987
2	1.17E-02	87.1	24.8	95.3	0.9985
4	1.04E-02	88.6	101.3	91.2	0.9941
6	8.53E-03	90.6	17.2	95.8	0.9990
8	1.09E-02	88.0	32.7	93.6	0.9981
10	1.13E-02	87.6	54.0	91.7	0.9968
12	8.24E-03	90.9	88.8	93.1	0.9948
14	8.83E-03	90.3	65.0	92.2	0.9962
16	2.89E-03	96.8	39.6	94.9	0.9977
18	9.35E-03	89.7	78.0	91.6	0.9954
20	1.64E-02	82.0	69.6	89.0	0.9959

In this experiment, the CSRN correctly rotated the face image, achieving a function accuracy of 96.8%, and an image correlation ratio of 99.8%. These results indicate that the CSRN has successfully learned to perform the affine rotation. Table 5.11 summarizes the results of this grey-scale rotation experiment

Table 5.11: Summary of gray-scale image rotation results.

		Grey-scale Rotation
Metric	**Units**	*Primary Test Image:* $\theta' = 16°$
J_{ACC}	*%*	96.8
J_{MSE}	-	2.89E-03
IM_{ACC}	*%*	94.9
IM_{MSE}	-	39.61
IM_{CR}	-	.9979
T_{TR}**(total)**	*sec*	887
T_{TR}**(norm)**	*msec*	66
T_{FC}	*msec*	
T_R	*min*	82

5.2.4.3 Scaling

In this experiment we train the CSRN to perform affine scaling. The network is trained using 7 images down-scaled from $\theta' = 0.52$ to 1.0 in steps of 0.8. The network is trained to up-scale these images with scaling factors of $\theta = 1/\theta'$. Figure 5.16 shows the resulting images. Row a) shows the input images. Row b) and c) show the corresponding CSRN output images and the results of raw scaling (5.8), respectively. Row d) shows the actual target image. Each column contains the results for a specific input image, and is labeled by its corresponding transformation parameter, θ'. The PTC of $\theta' = 0.84$ is highlighted in red, and shown in detail in Fig. 5.17.

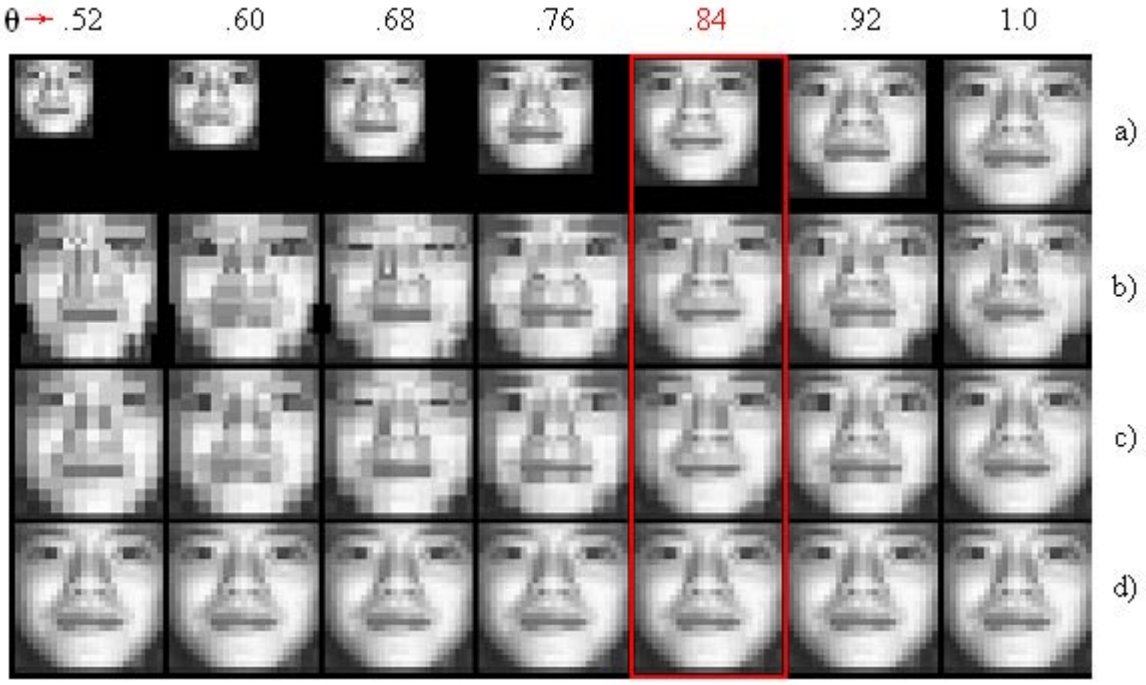


Figure 5.16: Results of affine scaling test. Results are shown for full testing set. Row a): input images scaled with scaling factor $\theta' = 0.52$ to 1.0 in steps of 0.8. Row b): corresponding CSRN output images. Row c): results of raw transformation by (5.7). Row d): target image. Columns label with transformation parameter, θ'. PTC: $\theta' = 0.84$.

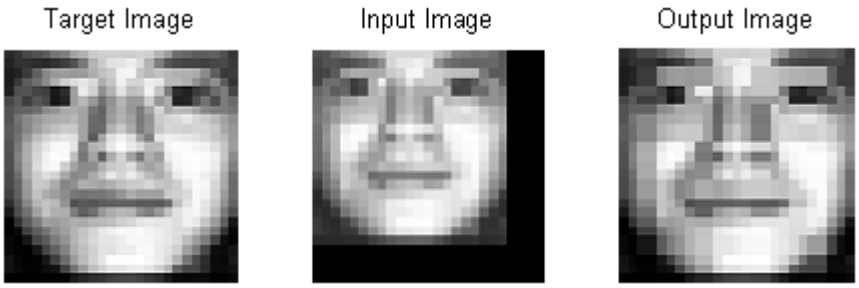


Figure 5.17: Results of affine rotation test for the PTC, $\theta' = 0.84$. The network achieves a final, $J_{ACC} = 68.3\%$ and an $IM_{CR} = 96.8\%$.

The image has been scaled to the target size with some loss of information. This is partially due to information loss in the down-scaled input images. Note that the resulting output images are increasingly distorted as the input image scale is decreased. Note that this is also true for the results of the raw transformation (row C of Fig. 5.16). This loss of

information explains the poor image accuracy which also decreases with input image scale. In spite of this, the CSRN's output images are still quite similar to the target image achieving image correlation ratios from a modest 80% to a very good 97%.

While the output images appear to be properly scaled, the network seems to have more difficulty in learning the scaling transform, as indicate by a mediocre function accuracy, which ranges from just under 50% to 97%. This is in part due to the difficulty in obtaining sufficient training images for such small images. In this case, we train with only seven images vs. the eleven used for translation and rotation. Table 5.12 contains the results of our scaling test for the full testing set. The PTC of $\theta' = 0.84$, shown in Fig. 5.17, is highlighted.

Table 5.12: Gray-scale image rotation results for full test set.

	Grey-scale Rotation Results (full test set)				
θ'	J_{MSE}	J_{ACC}	IM_{MSE}	IM_{ACC}	IM_{CR}
0.52	1.49E-01	48.8	1946.00	15.2	0.8024
0.6	7.43E-02	57.6	756.80	24.5	0.8961
0.68	3.43E-02	76.0	664.86	39.4	0.9053
0.76	2.74E-02	80.8	571.21	48.5	0.9200
0.84	4.57E-03	96.8	184.12	68.3	0.9734
0.92	5.94E-02	63.2	464.69	56.8	0.9358
1.0	1.07E-01	52.8	677.48	53.8	0.9148

In this experiment, the CSRN is trained to perform scaling of grey-scale images. The network achieves a best case result with $J_{ACC} = 96.8\%$, $IM_{ACC} = 68.3\%$, and $IM_{CR} = 97.3\%$. While the output images appear to be properly scaled, there is significant distortion due to information loss in the input images. In addition, the CSRN has

difficulty in learning the scaling transform with the limited training set available for the image size used in this experiment. Table 5.13 summarizes the results of our grey-scale scaling experiment.

Table 5.13: Summary of gray-scale image scaling results.

		Grey-scale Scaling
Metric	**Units**	*Primary Test Image:* $\theta' = 16°$
$\boldsymbol{J_{ACC}}$	*%*	96.8%
$\boldsymbol{J_{MSE}}$	-	2.89E-03
$\mathbf{IM_{ACC}}$	*%*	96.8
$\mathbf{IM_{MSE}}$	-	0.0
$\mathbf{IM_{CR}}$	-	68.3
$\boldsymbol{T_{TR}}$**(total)**	*sec*	4.57E-03
$\boldsymbol{T_{TR}}$**(norm)**	*msec*	.9734
$\boldsymbol{T_{FC}}$	*msec*	1022
$\boldsymbol{T_{R}}$	*min*	233

5.2.5 Baseline Comparison

In order to evaluate any image processing technique, we must establish a "ground truth" as a baseline for comparison. For affine transformation we utilize the raw transformations (RAW) based on (5.6), (5.7) and (5.8). Here we wish to evaluate the basic transformation apart from various methods of interpolation. Therefore, we limit all transformations to the use of nearest-neighbor interpolation.

To perform this experiment, the input images are passed to the desired transformation method (CSRN or RAW), which is required to transform the input image into the target image. By performing the transformation in this manner, all resulting images should be

equivalent to the target image, allowing us to easily perform image comparisons. We compare the function accuracy, by means of J_{MSE} and J_{ACC}, the image accuracy, by means of IM_{MSE}, IM_{ACC} and IM_{CR} and the computation time T_{FC}.

5.2.6 Translation

Figures 5.4 and 5.12 show the results of performing translation via CSRN and RAW for binary and grey-scale images, respectively. The results of these two methods are shown side-by-side in rows b) and c). Both methods can be compared to the actual target image shown in row d).

Table 5.14 shows results for both the CSRN and RAW transformation methods. The date shown is from the binary image experiments for a single test case of θ = -10 pix, where θ is the transformation parameter passed as one of the external inputs to the CSRN. Note that instances where the CSRN performs as well as or better than RAW are highlighted in red.

Table 5.14: Comparison of CSRN and RAW translations.

Translation					
Average					
Metric	**Units**	*CSRN*	*RAW*	*Diff.*	*% Err*
J_{ACC}	*%*	94.7	100.0	-5.3	-5.3
IM_{ACC}	*%*	99.9	100.0	-0.1	-0.1
IM_{CR}	-	0.98	1.00	-0.02	-2.0
T_{FC}	*msec*	1.46	3.83	-2.37	-61.9
Best					
$\boldsymbol{J_{ACC}}$	*%*	100.0	100.0	0	0.0
$\mathbf{IM_{ACC}}$	*%*	100.0	100.0	0	0.0
$\mathbf{IM_{CR}}$	-	1.0	1.0	0	0.0

A visual inspection of the images in Figs. 5.4 and 5.12 indicates the CSRN is capable of performing translation perfectly. In its best case results, the CSRN's performance matches that of the raw transform by which it is trained, indicating that it has fully learned the translation transformation. As indicated by the computation time, T_{FC}, the CSRN, once trained, performs this transformation much faster than the raw transformation.

5.2.7 Rotation

Figures 5.6 and 5.14 show the results of performing rotation via CSRN and RAW for binary and grey-scale images, respectively. The results of these two methods are shown

side-by-side in rows b) and c). Both methods can be compared to the actual target image shown in row d).

Table 5.15 shows results for both methods. The date shown is from the binary image experiments for a single test case of θ = -16 °. Note that instances where the CSRN performs as well as or better than RAW are highlighted in red.

Table 5.15: Comparison of CSRN and RAW rotations.

Rotation					
Average					
Metric	**Units**	*CSRN*	*RAW*	*Diff.*	*% Err*
J_{ACC}	*%*	82.62	100.0	-174	-17.4
IM_{ACC}	*%*	99.0	99.51	-0.51	-0.51
IM_{CR}	-	0.94	0.935	.005	-5.0
T_{FC}	*msec*	1.62	4.74	-3.12	-65.8
Best					
J_{ACC}	*%*	94.2	100.0	-5.8	-5.8
IM_{ACC}	*%*	99.1	100.0	-.09	-.09
IM_{CR}	-	1.0	1.0	0.0	0.0

An inspection of Figs. 5.6 and 5.14, indicates the CSRN is performs rotation well, achieving a best case performs slightly under that of the raw transformation. In spite of its lower function accuracy, 94.2%, the CSRN still achieves an image accuracy of 99.1% and an IM_{CR} of 100%, which is slightly better that that of the raw transform. As in the case of translation, the CSRN performs rotation faster than the raw transformation.

5.2.8 Scaling

Figures 5.9 and 5.16 show the results of performing rotation via CSRN and RAW for binary and grey-scale images, respectively. The results of these two methods are shown side-by-side in rows b) and c). Both methods can be compared to the actual target image shown in row d).

Table 5.16 shows results for both methods. The date shown is from the binary image experiments for a single test case of $\theta = 0.73$. Note that instances where the CSRN performs as well as or better than RAW are highlighted in red.

Table 5.16: Comparison of CSRN and RAW scaling.

Scaling					
Average					
Metric	**Units**	*CSRN*	*RAW*	*Diff.*	*% Err*
J_{ACC}	*%*	83.2	100.0	-16.8	-16.8
IM_{ACC}	*%*	99.57	99.51	0.06	0.06
IM_{CR}	-	0.98	0.935	.0045	4.81
T_{FC}	*msec*	1.49	4.74	-3.25	-68.6
Best					
J_{ACC}	*%*	100.0	100.0	0	0.0
IM_{ACC}	*%*	100.0	99.51	0.49	0.49
IM_{CR}	-	1.0	0.935	.065	6.95

An inspection of Figs. 5.9 and 5.16, indicates the CSRN performs scaling well, achieving a best case performance equal to or better than that of the raw transformation.

As in the other cases of translation and rotation, the CSRN performs scaling faster than the raw transformation.

5.3 Convergence of CSRN for Affine Transformations

Figure 5.18, shows a plot of the CSRN's testing MSE for the three affine transformations discussed in Section 5.2.3 above.

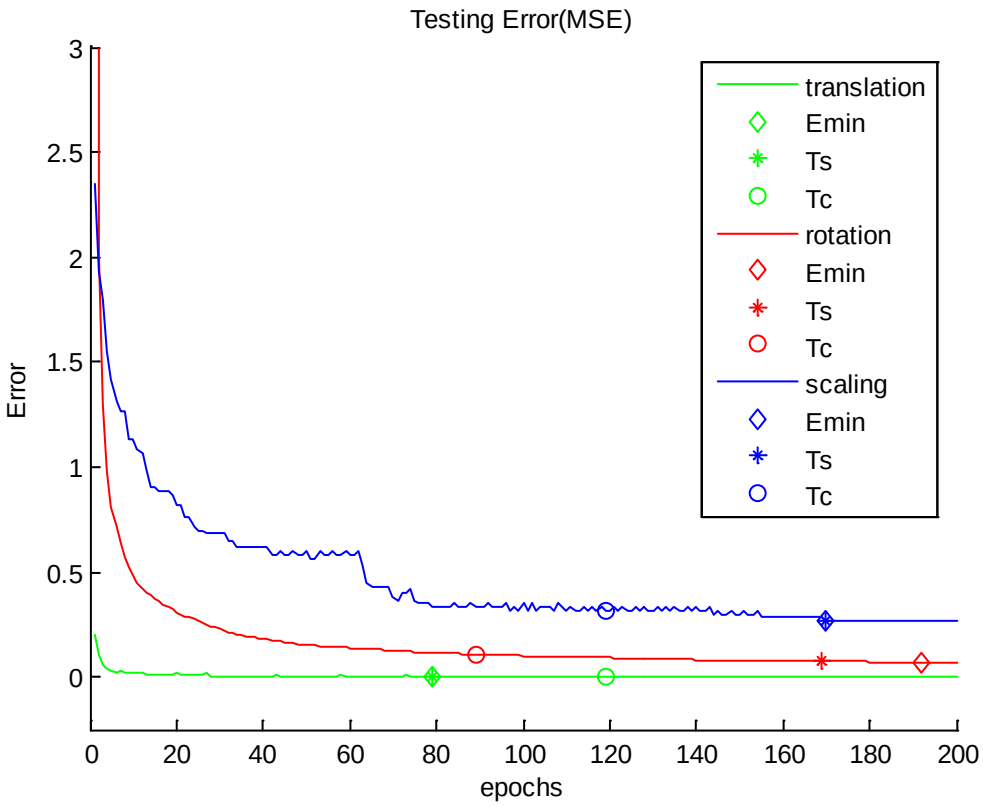


Figure 5.18: Plots of the testing MSE for the CSRN trained to perform affine transformation of binary images. The green line is for translation, the red line is for rotation and the blue line is for scaling. Circles indicate the calculated convergence points, T_C. The asterisks indicate the settling point, T_S and diamonds indicate the location of the minimum error.

Table 5.17 tabulates the convergence metrics of all three experiments. These metrics are computed based on batches of 50 simulations. The data shows that the learning process for the translation, rotation and scaling transforms consistently converge with average setting times of approximately 150, 160, and 180 epochs, respectively. The figure clearly indicates that the CSRN learns in all cases.

Table 5.17: Summary of binary image translation results.

Translation			
Metric	**Units**	*Ave +/- std dev*	*PTC:* θ' = 10 pix
$\boldsymbol{T_C}$	*epochs*	110.78 +/- 41.53	119
$\boldsymbol{T_S}$	*epochs*	156.62 +/- 52.67	79
$\boldsymbol{E_{MIN}}$	-	0.01 +/- 0.01	0.00
$\boldsymbol{T_{ME}}$	*epochs*	138.98 +/- 48.31	79
Rotation			
$\boldsymbol{T_C}$	*epochs*	98.6 +/- 34.4	89
$\boldsymbol{T_S}$	*epochs*	160 +/- 28.9	169
$\boldsymbol{E_{MIN}}$	-	0.20 +/- 0.06	7.15E-2
$\boldsymbol{T_{ME}}$	*epochs*	195 +/- 11.4	192
Scaling			
$\boldsymbol{T_C}$	*epochs*	94.1 +/- 71.6	119
$\boldsymbol{T_S}$	*epochs*	181 +/- 17.9	170
$\boldsymbol{E_{MIN}}$	-	0.48 +/- 0.12	2.67E-1
$\boldsymbol{T_{ME}}$	*epochs*	176 +/- 24.7	170

5.4 Larger Grey-scale Results

Thus far we have limited our test to images of size 35x35 pixels. Our generalized IP architecture utilizing sub-image processing is not limit to image of this size. The only limitation to image size is the processing time required to run simulation for the larger images. As an example we have include affine rotation results result for images of size 75x75 pixels and 125x125pixels. These are shown in Fig. 5.19. Note the quality of the transformation improves as the image size increases.

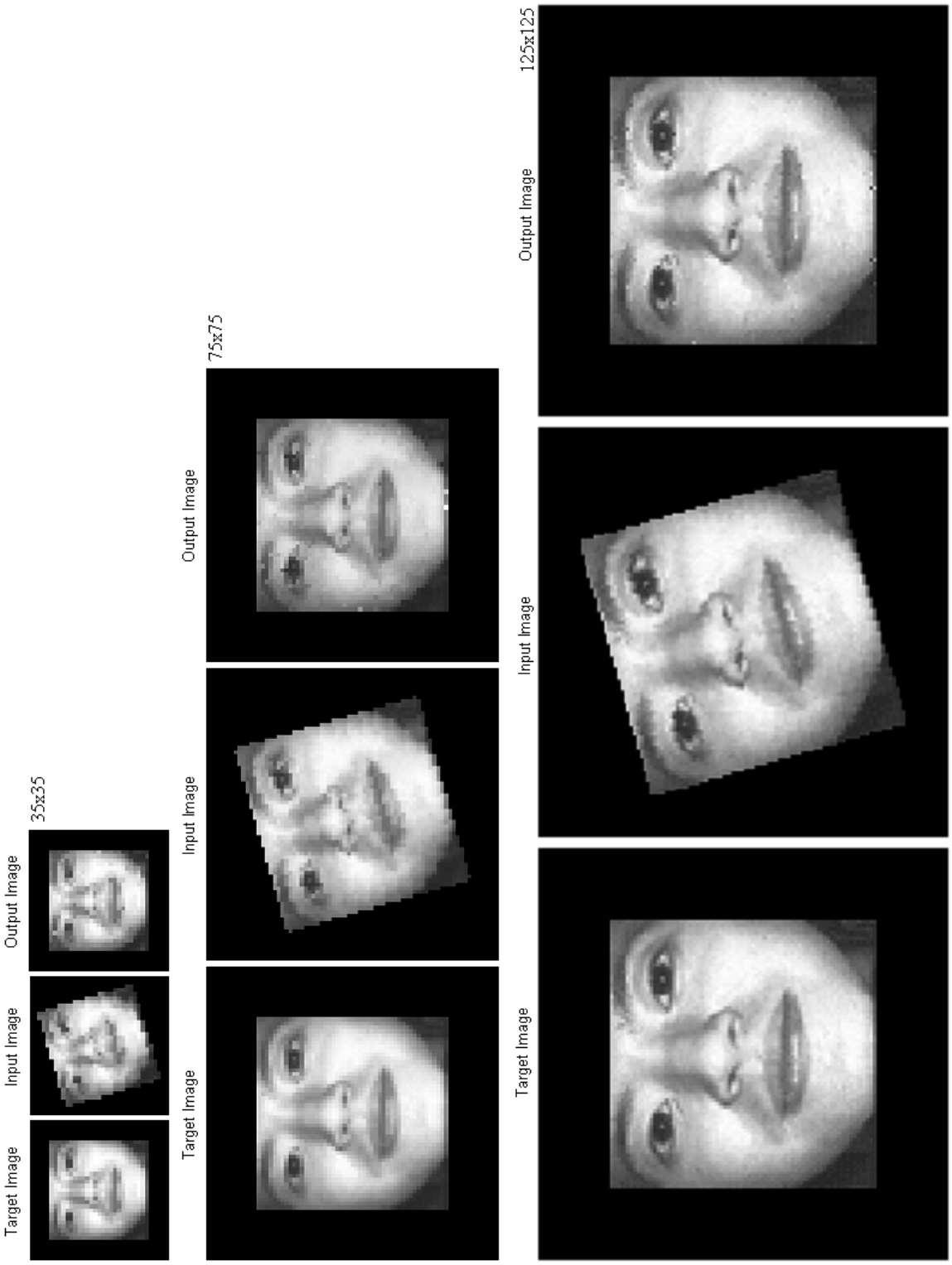


Figure 5.19: Results of affine rotation test for images of 35x35, 55x55 and 125x125 pixels. Results are shown for the PTC of $\theta' = 16°$.

5.5 Conclusions

In this chapter, pursuant to *goal 2* of this work, we demonstrate the CSRN's ability to learn and perform basic affine transformations on binary and grey-scale images. We adapt the CSRN to perform translation, rotation and scaling, discussing solutions to several practical implementation issues. As proof of concept we demonstrate the CSRN's ability to perform these transformations on small, 15x15, binary test images. Utilizing the sub-image processing technique developed in Chapter 4, we demonstrate the CSRN's ability to perform the above mentioned transformations on larger, grey-scale, facial images.

The CSRN achieves average function accuracies, J_{ACC}, of 95%, 83% and 83% for translation, rotation and scaling, respectively. The higher accuracy for translation is due to its local nature and the absence of information loss that occurs in the other two transforms. The CSRN achieves best case function accuracies of 100% for all three transformations. These results indicate that the CSRN is capable of learning the 2-D functions associated with these affine transformations.

The transformed images produced by the CSRN compare favorably to their corresponding target images, as indicated by the image accuracy and image correlation ratio metrics. The CSRN achieves average IM_{ACC} values of 99.9%, 99.0%, and 99.6% and average IM_{CR} values of 98%, 94%, and 98%, respectively. Best case IM_{ACC} results of 100%, 99.1%, 100% and best case IM_{CR} results of 100%, 95% and 100% are achieved. When compared to the results produced by the raw transformation equations, discussed in Section 5.1.1, the CSRN's performance either meets or exceeds the performance of the raw transforms.

The CSRN's learning process is stable, with the mean-squared testing error consistently converging in less than 200 epochs for all three transformations.

The CSRN's forward computation times are 1.46msec, 1.62msec, and 1.49msec, respectively. The CSRN reduces computation times for these transforms by 62%, 66% and 69%, over those achieved by the raw transformations. The faster computation times of the CSRN make it attractive for real-time, embedded applications.

6 IMAGE REGISTRATION USING CSRNS

6.1 Introduction

Image registration (IR) is the process of aligning two or more images. These images encompass the same scene possibly obtained at different times, with different sensors and/or from different viewpoints [102]. By convention, one image (the reference image) is considered stationary, while the second image (the sensed image) is transformed to the coordinates of the reference image. The goal of any IR technique is to find the optimal transformation that best aligns the structures of interest in the two images [65].

6.1.1 Applications

Registration is a fundamental image processing (IP) task. It plays a crucial role in many IP applications, which include: computer vision (facial recognition, target localization/tracking/recognition, robot navigation, automatic quality control), remote sensing (environmental monitoring, land usage, cartography, astrophotography) and medical imaging (fusion of multimodal images such as CT, MRI or PET to gain more complete patient information, tumor detection and growth monitoring, treatment verification, development of and comparisons to anatomical atlases) [102].

6.1.2 Literature Review

Due to its fundament role in so many IP applications, IR has been the topic of much research over the past three decades. Zitova *et al*. [102] report that according to the Institute of Scientific Information's database (now known as the Thomson Reuter's Web of Knowledge) that in the 10 years prior to 2002, over 1000 papers were published on the topic of image registration. A similar search using Google Scholar for the 10 years since,

2002 to 2012, reveals that approximately 664,000 articles on this topic have been published [106].

Considering the enormous volume of research, a complete literature review of IR is a daunting task which lies outside the scope of this work. Fortunately, several in-depth surveys as well as several excellent tutorials exist in literature. A brief overview of these works is presented here, in hopes of providing the reader with a body of information from which to build a foundational understanding of the topic.

There are several surveys that approach IP from a general purpose perspective. Ghaffery [28] presents the historical 'first' paper on this topic in 1983. Brown [7] provides an exhaustive review in 1992, while Zitova *et al.*'s [102], work documents methods established after 1992, and includes those methods prior to 1992 that had become classic and/or introduced key concepts which were still in use as of 2003. Wyawahare *et al.* [100] provide an exhaustive survey in 2009. Perhaps the latest survey to date is present in Deshmukh *et al.* [19] in 2011. Their work provides a very practical overview including results and comparisons using a variety of real world examples.

In additional to those described above, there are multiple surveys that have been done within given application areas. [28][29][30][42][54][56] provide exhaustive surveys for IR in medical imaging. [25][36][63] provide surveys from the area of remote sensing.

Several excellent tutorials for IR exist, including [2][40][66] [84][89][99].

6.1.3 Classification of Image Registration Techniques

With the wealth of IR techniques available, several authors attempt classification of IR techniques. Perhaps the earliest attempt at classification is Elsen, *et al.* [22], followed

by a more comprehensive classification by Maintz [56]. Chmielewski *et al*. [10] summarize several earlier classifications based on the following 10 criteria:

1. *dimensionality*: Dimension of the images, 2D/2D, 2D/3D, 3D/3D.
2. *domain of transformation:* Local or global depending on whether the entire image or a sub-image is used.
3. *type of transformation:* Rigid, affine, projective or non-linear.
4. *tightness of feature coupling:* Exact or approximate transfer of features.
5. *method of parameter determination:* Parameters may be estimated directly or via search methods.
6. *subject of registration:* inter or intra subject registration.
7. *type of data:* Raw data or extracted features.
8. *source of features:* features explicitly present in the data, intrinsic, or those externally introduced, extrinsic.
9. *automation level:* automatic, semiautomatic or manual, depending upon the users level of intervention.
10. *measure of registration quality:* Metrics used to quantify the quality of registration.

6.1.4 General Steps for Image Registration

Regardless of the methods used for implementation, IR can be broken down into the following 4 steps [102]:

1. feature detection
2. feature matching
3. transform model parameter estimation
4. transformation and re-sampling.

6.1.5 Problems with Existing IR methods

Each of the four steps above presents a non-trivial problem in the implementation of any IR technique. If we consider step 4 as the actual IR transformation, then steps 1 – 3 can be viewed as pre-processing steps, each step serving as the input to the proceeding step. Errors in any step are propagated forward and affect the final registration quality. Each of these steps are complex and require significant time to perform and therefore

limit the application of the given IR technique in any real-time or near real-time applications such as target tracking and robot navigation.

Consider the following simple experiment performed early in this work. I ask my daughter (4 years old at the time) to perform the following task. I sat her down at a table, and placed a picture of my wife's face on the table. I rotated the picture about 30 degrees, then showed my daughter how to "straighten" the picture. With this minimal "training" she was able to "straighten" (read that "register") the image from that point forward, regardless of the initial angle at which the image was placed. Further, I could then place pictures of other objects (cars, toys, furniture) under various angles of rotation, and she could register those images as well.

The human brain, unlike existing IR methods, learns to perform these registration steps and once learned performs them quickly and easily without conscience effort on our part. To achieve similar results we must turn to biologically based system, such as NNs.

6.1.6 Neural Networks for Image Registration

In the past decade, NNs have been applied to image registration. Ramirez *et al.* [73] provide a review of existing NN techniques available for IR. However, most applications apply NNs to perform one of the above pre-processing steps. Several examples follow.

Shang *et al*. and He *et al*. use NNs to perform feature extraction (step 1). Shang *et al.* [83] use PCA NNs to compute the 1st principal direction for both the reference and sensed images. This feature is then used to compute the parameters of a rigid body model (rotation and translation). He *et al.* [41] use pulse-coupled NNs (a type of feed-forward network) to extract foveation points that are then matched and used to compute model parameters. Elhanany *et al.* [20], first extract features using the discrete cosine transform

(DCT), then use these features to train an MLP network to compute the parameters of an affine based model (step 3). Mostofa *et al*. [64] use a MLP to perform surface interpolation in fusing standard 2D satellite images with range data for development of 3D geographical models, thus performing the re-sampling portion of step 4.

6.2 Image Registration via CSRN

We limit the scope of our investigation of CSRNs for IR to the use of the rigid body model. The transformations involved will, therefore, be limited to those of translation and rotation.

6.2.1 Adapting the CSRN for Image Registration

As in the case of affine transformations, we must adapt the CSRN to perform IR.

6.2.1.1 Cost-Function of Image Registration

We first consider whether the cost-function associated with learning in the CSRN is capable of approximating the cost-function associated with image registration. Mathematically, IR can be expressed as

$$\min_T D\left[I_R, T\left(I_S\right)\right], \tag{6.1}$$

where, D is the distance measure, T is the transformation, I_R is the reference or stationary image, and I_S is the sensed or moveable image [65]. In the rigid body case, D becomes the Euclidean distance between corresponding pixels, and T the translation and/or rotation transformations. Let X_R^i represent the location of the i^{th} pixel in the reference

image and X_S^i its corresponding pixel in the sensed image. The distance between the reference and sensed locations is given by

$$d_i = \| X_R^i - X_S^i \|. \tag{6.2}$$

We can view this distance as the error in pixel location, e_i.

$$e_i = \| X_R^i - X_S^i \|. \tag{6.3}$$

The total error in the transformed image is given by

$$E_i = \frac{1}{2}\sum_{i=1}^{c} e_i^2, \tag{6.4}$$

where c is the number of pixels in the image. If we extend this error to an entire set of images, the error becomes

$$E = \sum_{n=1}^{N}\left[\frac{1}{2}\sum_{i=1}^{c} E_i^2\right], \tag{6.5}$$

where N is the number of images in the set. Equation (6.5) represents the cost-function for image registration of rigid bodies. Minimizing this cost function minimizes the Euclidean distance between the location of pixels in the reference and sensed images. Note that (6.5) is equivalent to (3.3), which is the cost-functions for learning in CSRNs. This suggests that the CSRN should be capable of learning to perform IR.

6.2.1.2 Inputs/Outputs

In the case of IR, the transformation parameter, θ, is not known a-priori and thus cannot be used as an input to the network. Therefore, we utilize *pixel location* and *intensity* as external inputs for IR.

The output of the network for IR is the same as that for affine transformation, discussed in Section 5.2.1.1.

6.2.2 Implementation Issues

IR involves the same implementation issues as affine transformation. Refer to Section 5.2.2 for a discussion of these issues.

6.2.3 Registration of Binary Images

In this section we examine the use of CSRNs to perform image registration of binary images. In keeping with the rigid body assumption, we restrict transformations to translation and rotation.

6.2.3.1 Experiment Configuration

In this series of experiments we utilize a simple binary image of a cross as our test subject. The CSRN is configured with a GMLP core, EKF training, movement encoding and inverse mapping. Three external inputs are used: the pixel's location (x and y), and intensity. The network is trained with eleven images which are zero padded as discussed in Section 5.2.2.4. The network is tested with the full set of training images. As in the case of affine translations, a primary test case (PTC) is selected for display in detail.

6.2.3.2 Translation

In this experiment we train the CSRN to perform image registration of binary images under affine translation. Training images utilize zero padding with a width of 5 pixels. The input images are translated through a range of $\theta' = 0$ to 10 pixels. Figure 6.1 shows the resulting images. Row a) shows the input images. Row b) shows the corresponding CSRN output images. Row c) shows the results of the baseline registration method, discussed below in Section 6.2.5. Row d) shows the actual target image. Each column contains the results for a specific input image, and is labeled by its corresponding transformation parameter, θ'. Note the results for the PTC, $\theta' = 5$ pixels, are highlighted in red and shown in detail in Fig. 6.2.

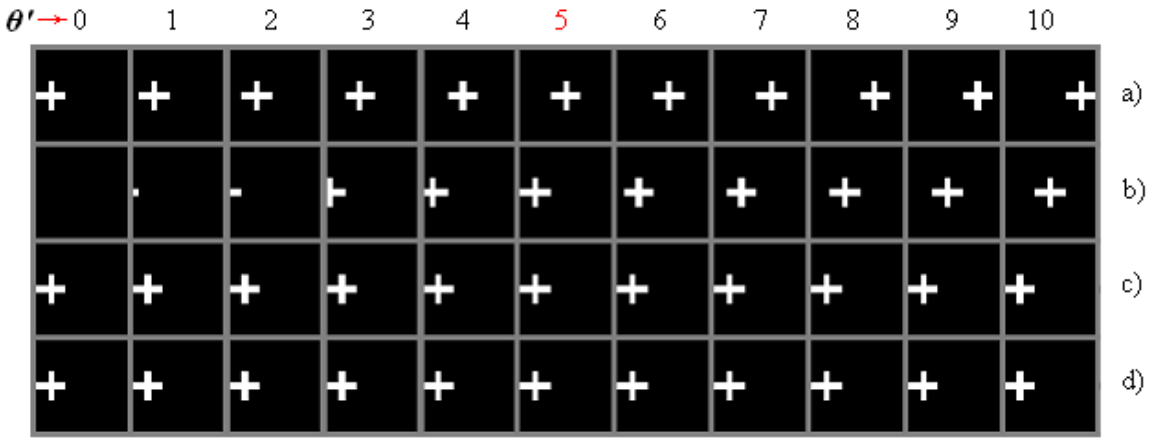


Figure 6.1: Results of registering binary images under affine translation. Results are shown for full testing set. Row a): input images translated through a range of $\theta' = 0$ to 10 pix. Row b): corresponding CSRN output images. Row c): results of baseline registration method. Row d): target image. Columns label with transformation parameter of the input images, θ'. PTC: $\theta' = 5$ pix.

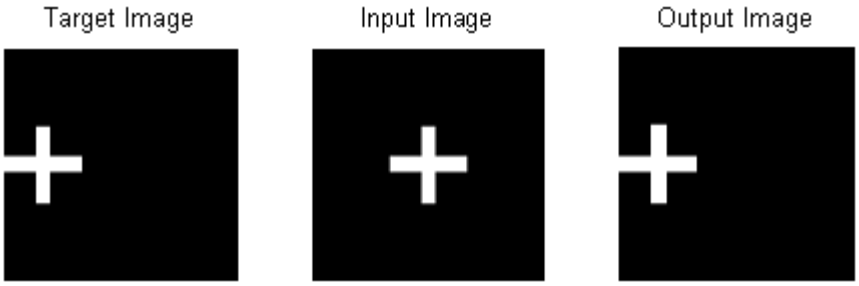


Figure 6.2: Results of IR of binary image under affine translation. PTC: $\theta' = 5$ pix. The network achieves a final $J_{ACC} = 100\%$ and an $IM_{ACC} = 100\%$.

Table 6.1 tabulates the results for this registration experiment. Results for the full testing set are shown. The PTC, $\theta' = 5$ pixels, is highlighted.

Table 6.1: Registration results of binary images under affine translation. Data for full test set shown. The PTC, $\theta' = 5$ pixels, is highlighted.

	Binary Images Under Translation (full test set)				
θ'	J_{MSE}	J_{ACC}	IM_{MSE}	IM_{ACC}	IM_{CR}
0	2.27E+00	0	4.00E-02	96.0	0.0
1	1.45E+00	0	3.56E-02	96.4	0.3273
2	8.18E-01	0	3.11E-02	96.9	0.4639
3	3.64E-01	0	4.44E-02	95.6	0.3553
4	9.09E-02	0	4.00E-02	96.0	0.4507
5	0.00E+00	100	0.00E+00	100.0	1.0000
6	9.09E-02	0	4.44E-02	95.6	0.4213
7	3.64E-01	0	5.33E-02	94.7	0.3056
8	8.18E-01	0	6.22E-02	93.8	0.1898
9	1.45E+00	0	7.11E-02	92.9	0.0741
10	2.27E+0	0.0	8.00E-02	92.0	-0.0417

Inspection of Fig. 6.1 reveals that the CSRN is unable to generalize over all its training images. Without the transformation parameter, available in the affine

transformation application, as an additional input, the CSRN is unable to sufficiently distinguish between input images and produces the same output for all images. Table 6.2 shows this output. Note that this output calls for every image to be translated 5 pixels to the left. While this table may seem trivial at first, much insight can be gained from understanding how the CSRN came to produce this output.

Table 6.2: CSRN's output for registration of 15x15, binary images under affine translation.

5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Consider the IR cost-function given by (6.5), repeated here for convenience,

$$E = \sum_{n=1}^{N}\left[\frac{1}{2}\sum_{i=1}^{c} E_i^2\right]. \tag{6.5}$$

In this case, we train with 11 images translated from 0 to 10 pixels, respectively. The image position which minimizes the error over the entire training set is the mid-point of the translation range, or 5 pixels. Examination of Fig. 6.1 row b), shows that the CSRN has moved each image 5 pixels to the left. As a result, the PTC, θ' = 5 pix, is the only image that is correctly registered.

Table 6.3 summarizes the results of the translation experiment for binary images. Statistics are computed over a batch of 50 simulations.

Table 6.3: Summary of image registration results for binary images under affine translation. PTC: θ' = 5 pixels.

		Binary Images Under Translation	
Metric	**Units**	*Ave +/- std dev*	*PTC:* θ' = 5 pix
J_{ACC}	*%*	0.00 +/- 0.00	100.0
J_{MSE}	-	2.27 +/- 0.01	0.0
IM_{ACC}	*%*	92.00 +/- 0.00	100.0
IM_{MSE}	-	0.08 +/- 0.00	0.0
IM_{CR}	-	-0.04 +/- 0.00	1.0
T_{TR}(total)	*sec*	559.70 +/- 0.41	559.09
T_{TR}(norm)	*msec*	226 +/- 0.17	225.9
T_{FC}	*msec*	1.48 +/- 0.01	1.48
T_R	*min*	9.58 +/- 0.01	9.60
T_B	*hrs*	*n/a*	7.99

6.2.3.3 Rotation

In this experiment we train the CSRN to perform IR for binary images under affine rotation. Training images utilize zero padding with a width of 2 pixels. The input images are translated through a range of θ' = 0° to 20° in steps of 2°. The network is tested with

the full set of training images. Figure 6.3 shows the resulting images. Row a) shows the input images. Row b) and c) show the corresponding CSRN and baseline output images, respectively. Row d) shows the actual target image. Each column contains the results for a specific input image, and is labeled by its corresponding transformation parameter, θ'. Note the results for case $\theta' = 10°$ are highlighted in red and shown in detail in Fig. 6.4.

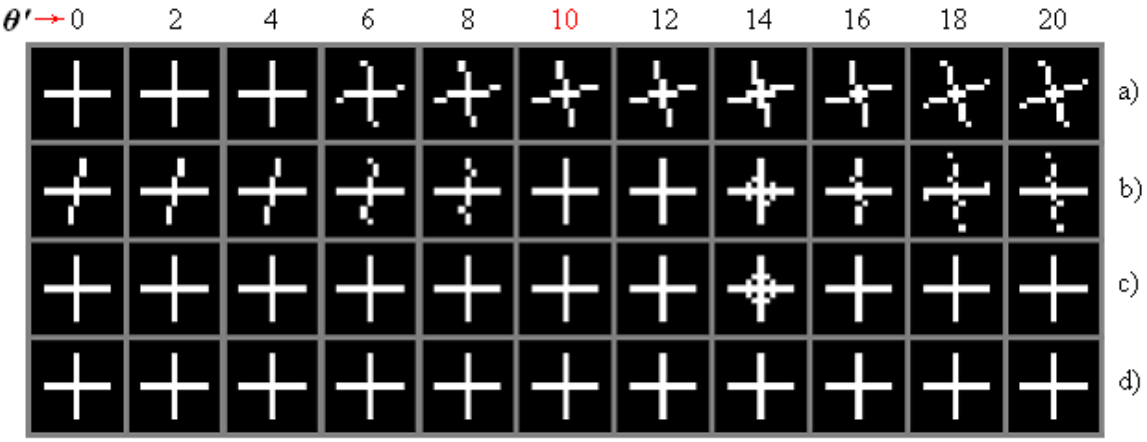


Figure 6.3: Results of registration of binary images under affine rotation. Results are shown for full testing set. Row a): input images rotated through a range of $\theta' = 0°$ to 20° in steps of 2°. Row b): corresponding CSRN output images. Row c): results of baseline registration method. Row d): target image. Columns label with transformation parameter, θ'.

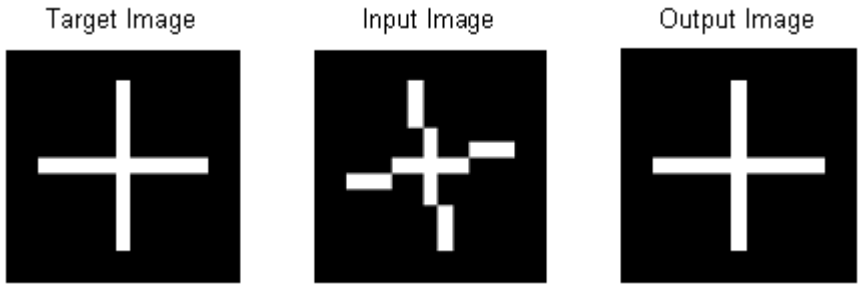


Figure 6.4: Results of image registration for case $\theta' = 10°$. The network achieves a best, $J_{ACC} = 94.7\%$ and an $IM_{ACC} = 100.0\%$.

Table 6.4 tabulates the results of this IR experiment. Results for the full testing set are shown. The PTC, $\theta' = 10°$, is highlighted in red.

Table 6.4: Registration results of binary images under affine rotation. Data for full test set shown.

	Binary Images Under Rotation (full test set)				
θ'	J_{MSE}	J_{ACC}	IM_{MSE}	IM_{ACC}	IM_{CR}
0	5.74E-02	38.2	5.33E-02	94.7	0.685
2	5.74E-02	38.2	5.33E-02	94.7	0.685
4	5.41E-02	40.4	5.33E-02	94.7	0.685
6	2.26E-02	75.1	3.56E-02	96.4	0.790
8	1.05E-02	88.4	1.78E-02	98.2	0.895
10	4.80E-03	94.7	0.00E+00	100.0	1.000
12	9.70E-03	89.3	0.00E+00	100.0	1.000
14	2.59E-02	71.6	2.67E-02	97.3	0.869
16	3.72E-02	59.1	1.78E-02	98.2	0.895
18	4.93E-02	47.1	4.44E-02	95.6	0.749
20	6.79E-02	40.0	3.56E-02	96.4	0.790

These results indicate that the CSRN exhibits the same generalization problem for registering rotated images, as seen when registering translated images. In this case, the CSRN registers all images back to the mid-point of the rotation range, i.e.10° and, once again, the PTC is the only image correctly registered. Table 6.5 summarizes the results of the rotation experiment for binary images. Statistics are computed over a batch of 50 simulations.

Table 6.5: Summary of image registration results for binary images under affine rotation.

		Binary Images Under Rotation	
Metric	**Units**	*Ave +/- std dev*	*PTC:* $\theta' = 10°$
$\boldsymbol{J_{ACC}}$	*%*	55.26 +/- 1.91	94.7
$\boldsymbol{J_{MSE}}$	-	0.04 +/- 0.00	4.85E-3
$\mathbf{IM_{ACC}}$	*%*	99.02 +/- 0.57	100.0
$\mathbf{IM_{MSE}}$	-	0.03 +/- 0.01	0.02
$\mathbf{IM_{CR}}$	-	0.82 +/- 0.04	1.0
$\boldsymbol{T_{TR}}$**(total)**	*sec*	568.72 +/- 0.40	569
$\boldsymbol{T_{TR}}$**(norm)**	*msec*	229.78 +/- 0.16	230
$\boldsymbol{T_{FC}}$	*msec*	1.46+/- 0.01	1.46
$\boldsymbol{T_{R}}$	*min*	9.75 +/- 0.01	9.75
$\boldsymbol{T_{B}}$	*hrs*	*n/a*	8.12

6.2.4 Registration of Grey-scale Images

In this section we examine the use of CSRNs to perform IR of grey-scale images. In keeping with the rigid body assumption, we restrict transformations to translation and rotation.

6.2.4.1 Experiment Configuration

In this series of experiments we utilize the grey-scale facial image, shown in Fig. 5.11, as our test subject. The CSRN is configured with a GMLP core, EKF training, movement encoding and inverse mapping. Three external inputs are used: the pixel's location (x and y), and intensity. The network is trained with eleven images which are zero padded as discussed in Section 5.2.2.4. The network is tested with the full set of training images. As before, we select a PTC for detailed display.

6.2.4.2 Translation

In this experiment we train the CSRN to perform image registration of grey-scale images under affine translation. Training images utilize zero padding with a width of 5 pixels. The input images are translated through a range of $\theta' = 0$ to 10 pixels. Figure 6.5 shows the resulting images. Row a) shows the input images. Row b) shows the corresponding CSRN output images. Row c) shows the results of the baseline registration method. Row d) shows the actual target image. Each column contains the results for a specific input image, and is labeled by its corresponding transformation parameter, θ'. The PTC, $\theta' = 5$ pixels, is highlighted in red and shown in detail in Fig. 6.6.

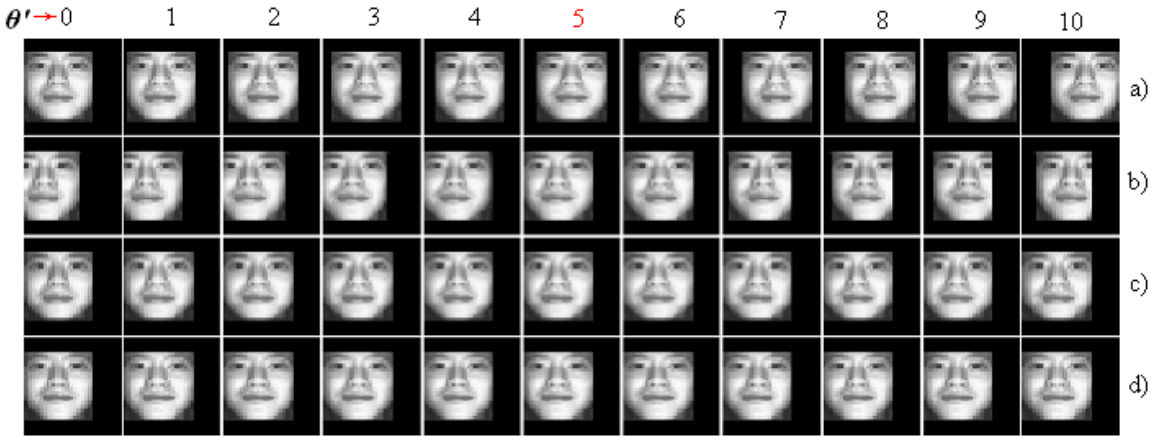


Figure 6.5: Results of registering grey-scale images under affine translation. Results are shown for full testing set. Row a): input images translated through a range of $\theta' = 0$ to 10 pixels. Row b): corresponding CSRN output images. Row c): results of baseline registration method. Row d): target image. Columns label with transformation parameter of the input images, θ'. PTC: $\theta' = 5$ pixels, highlighted in red.

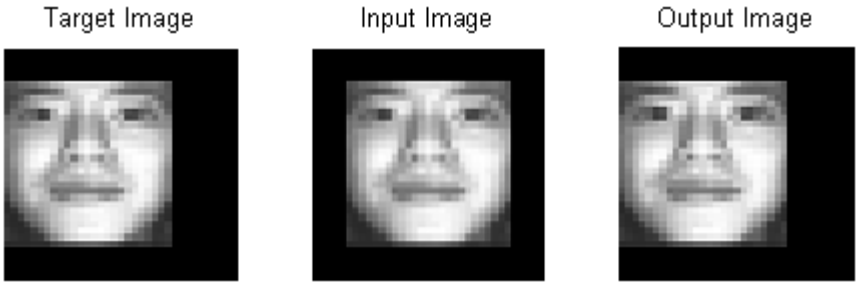


Figure 6.6: Results of IR of grey-scale image under affine translation. PTC: θ' = 5 pixels. The network achieves a final J_{ACC} = 100% and an IM_{CR} = 1.0.

Table 6.6 tabulates the results for this registration experiment. Results for the full testing set are shown. . The PTC, θ' = 5 pixels is highlighted.

Table 6.6: Registration results of grey-scale images under affine translation. Data for full test set shown.

	Grey-scale Images Under Translation (full test set)				
θ'	J_{MSE}	J_{ACC}	IM_{MSE}	IM_{ACC}	IM_{CR}
0	2.27E+00	0.0	4.61E+03	49.3	0.739
1	1.45E+00	0.0	3.62E+03	49.3	0.794
2	8.18E-01	0.0	2.59E+03	49.6	0.852
3	3.64E-01	0.0	1.54E+03	49.6	0.911
4	9.09E-02	0.0	3.69E+02	50.3	0.978
5	0.00E+00	100.0	0.00E+00	100.0	1.000
6	9.09E-02	0.0	3.48E+02	50.3	0.980
7	3.64E-01	0.0	1.23E+03	49.6	0.929
8	8.18E-01	0.0	2.06E+03	49.6	0.883
9	1.45E+00	0.0	2.88E+03	49.3	0.837
10	2.27E+00	0.0	3.67E+03	49.3	0.793

Again, we see the same generalization problem exhibited when registering binary images and the results are similar to those obtained in the binary registration case.

Table 6.7 summarizes the results of the translation experiment for binary images. Statistics are computed over a batch of 50 simulations

Table 6.7: Summary of image registration results for grey-scale images under affine translation.

		Grey-scale Images Under Translation
Metric	**Units**	*PTC:* θ' = 5 pixels
J_{ACC}	*%*	100.0
J_{MSE}	-	0.0
IM_{ACC}	*%*	100
IM_{MSE}	-	0.0
IM_{CR}	-	1.0
T_{TR}(total)	*sec*	752
T_{TR}(norm)	*msec*	56
T_R	*min*	142

6.2.4.3 Rotation

In this experiment we train the CSRN to perform IR for grey-scale images under affine rotation. Training images utilize zero padding with a width of 2 pixels. The input images are translated through a range of θ' = 0° to 20° in steps of 2°. The network is tested with the full set of training images. Figure 6.7 shows the resulting images. Row a) shows the input images. Row b) and c) show the corresponding CSRN output and baseline images, respectively. Row d) shows the actual target image. Each column contains the results for a specific input image, and is labeled by its corresponding transformation parameter, θ'. The PTC is highlighted in red and shown in detail in Fig. 6.8.

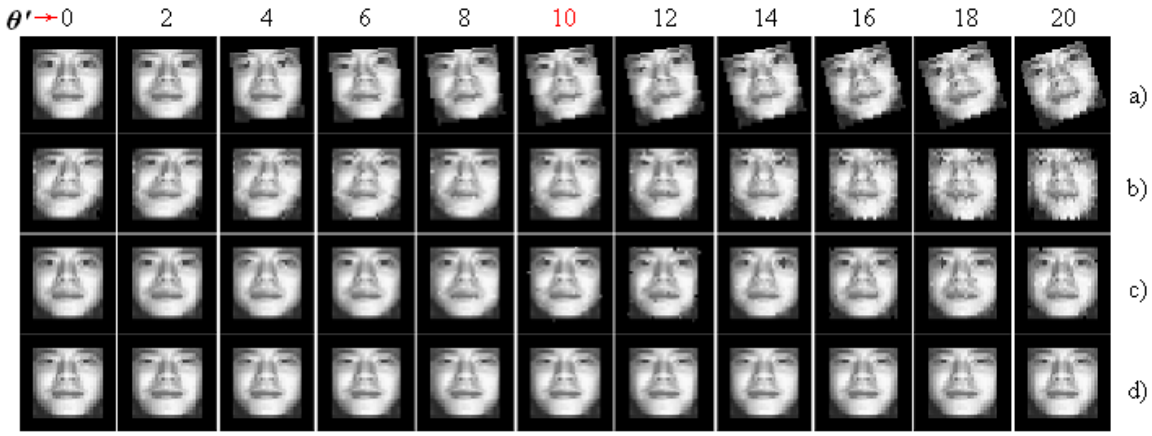


Figure 6.7: Results of registration of grey-scale images under affine rotation. Results are shown for full testing set. Row a): input images rotated through a range of $\theta' = 0°$ to 20° in steps of 2°. Row b): corresponding CSRN output images. Row c): results of baseline registration method. Row d): target image. Columns label with transformation parameter, θ'. PTC: $\theta' = 10°$ highlighted in red.

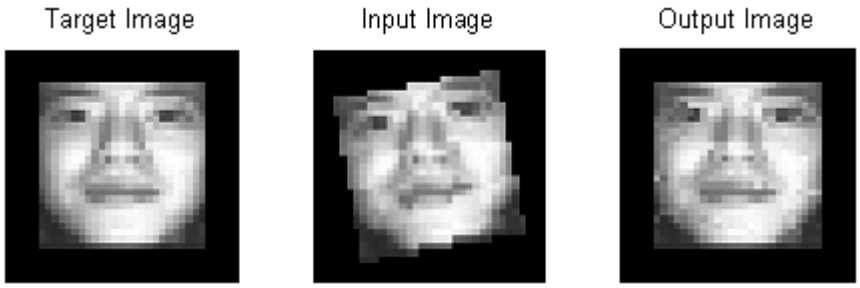


Figure 6.8: Results of image registration for PTC, $\theta' = 10°$. The network achieves a best, $J_{ACC} = 88.7\%$ and an $IM_{CR} = 1.0$.

Table 6.8 tabulates the results of this IR experiment. Results for the full testing set are shown. The PTC, $\theta' = 10°$, is highlighted.

Table 6.8: Registration results of grey-scale images under affine rotation. Data for full test set shown.

	Grey-scale Images Under Rotation (full test set)				
$\boldsymbol{\theta'}$	$\boldsymbol{J_{MSE}}$	$\boldsymbol{J_{ACC}}$	$\mathbf{IM_{MSE}}$	$\mathbf{IM_{ACC}}$	$\mathbf{IM_{CR}}$
0	2.97E-01	16.1	4.14E+02	61.1	0.976
2	2.24E-01	16.1	4.14E+02	61.1	0.976
4	1.19E-01	24.7	2.29E+02	67.1	0.987
6	6.03E-02	40.0	1.88E+02	76.6	0.989
8	3.17E-02	65.1	9.45E+01	86.4	0.995
10	1.03E-02	88.7	3.77E+01	93.2	0.998
12	2.66E-02	70.7	1.13E+02	88.8	0.993
14	5.75E-02	43.1	2.20E+02	76.9	0.987
16	1.15E-01	29.3	2.89E+02	69.6	0.983
18	1.91E-01	21.7	4.62E+02	63.9	0.973
20	2.90E-01	17.7	5.71E+02	61.1	0.967

As in the previous registration cases, the CSRN exhibits the same generalization problem previously discussed. Table 6.9 summarizes the results of registration of grey-scale images under affine rotation. Statistics are computed over a batch of 50 simulations.

Table 6.9: Summary of image registration results for grey-scale images under affine rotation.

		Grey-scale Images Under Rotation
Metric	**Units**	*Primary Test Image:* $\theta' = 10°$
$\boldsymbol{J_{ACC}}$	*%*	88.7
$\boldsymbol{J_{MSE}}$	-	0.0
$\mathbf{IM_{ACC}}$	*%*	93.2
$\mathbf{IM_{MSE}}$	-	1.03E-2
$\mathbf{IM_{CR}}$	-	1.0
$\boldsymbol{T_{TR}}$**(total)**	*sec*	802
$\boldsymbol{T_{TR}}$**(norm)**	*msec*	60
$\boldsymbol{T_R}$	*min*	151

6.2.5 Baseline Comparison

In order to establish a "ground truth" IR technique, we must specify a transformation model and a means for estimating the parameters of the selected model. We have already established the use of the rigid body model, which utilizes the transformations based on (5.6) and (5.7). For our baseline case, we assume that the parameters of the model are perfectly estimated, which will provide the best case registration under the rigid body model. Since we wish to evaluate the CSRN's ability to perform basic registration, apart from re-sampling methods, we limit both the CSRN and baseline methods to use of nearest-neighbor interpolation. Baseline image results are included in row c) of all preceding figures which contain IR results.

Due to the generalization problem, as previously discussed, we limit our comparisons to those of the PTC only.

6.2.6 Registration Under Translation

Figures 6.1 and 6.5 shows the results of IR via CSRN and baseline methods, for binary and grey-scale images, respectively. Table 6.10 shows results for both the CSRN and baseline registration methods. The date shown is that from the PTC, θ' = 5 pixels, of the binary IR experiment. Note that instances where the CSRN performs as well as or better than the baseline method are highlighted in red.

Table 6.10: Comparison of CSRN and baseline registration methods for images transformed by translation.

Translation					
PTC: θ' = 5 pix					
Metric	**Units**	*CSRN*	***BASELINE***	*Diff.*	***% Diff.***
J_{ACC}	*%*	**100.0**	**100.0**	0	0
IM_{ACC}	*%*	**100.0**	**100.0**	0	0
IM_{CR}	**-**	**1.0**	**1.00**	0	0
T_{FC}	*msec*	**1.41**	**3.83**	-2.69	-70.2

A visual inspection of the images in Figs. 6.2 and 6.6, indicate the CSRN is capable registering an image which has undergone affine transformation. This registration is subject to limited generalization as discussed above. In the PTC, the CSRN's performance matches that of the baseline method by which it is trained. As indicated by the computation time, T_{FC}, the CSRN, once trained, performs this registration much faster than the baseline method.

6.2.7 Registration under Rotation

Figures 6.3 and 6.7 show the results of IR via CSRN and baseline methods, for binary and grey-scale images, respectively. Table 6.11 shows results for both the CSRN and baseline registration methods. The data shown is that from the PTC, $\theta' = 10°$, of the binary IR experiment. Note that instances where the CSRN performs as well as or better than the baseline method are highlighted in red.

Table 6.11: Comparison of CSRN and baseline registration methods for images transformed via rotation.

Rotation					
PTC: θ' = 10°					
Metric	**Units**	***CSRN***	***BASELINE***	***Diff.***	***% Err***
J_{ACC}	*%*	94.7	100.0	-5.3	-5.3
IM_{ACC}	*%*	100.0	100.0	0	0.0
IM_{CR}	-	1.0	1.0	0	0.0
T_{FC}	*msec*	1.55	4.74	-3.19	-67.3

A visual inspection of the images in Figs. 6.4 and 6.8, indicate the CSRN is capable registering an image which has undergone affine transformation. Again, this registration is subject to limited generalization as discussed above. In the PTC, the CSRN's registration performance matches that of the baseline method, in spite of achieving a slightly, lower function accuracy. As in the case of registration under translation, the CSRN is much faster than the baseline method.

6.3 Convergence of CSRN for Image Registration

Figure 6.9, shows a plot of the CSRN's testing MSE for registration of images under both translation and rotation.

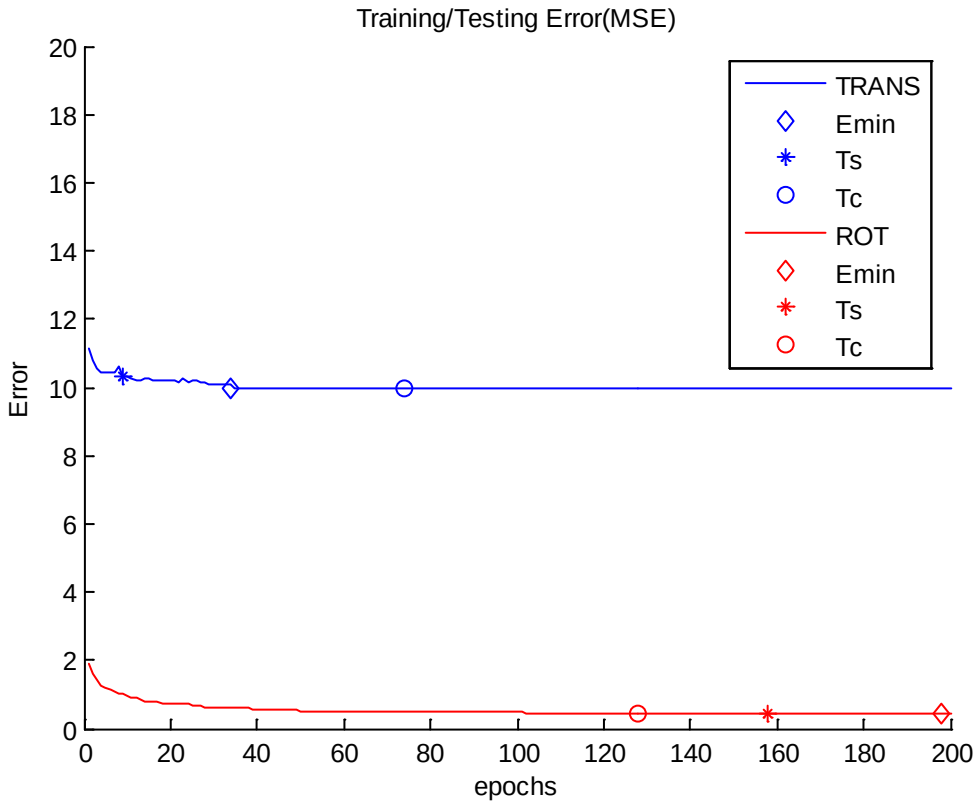


Figure 6.9: Plots of the testing MSE for the CSRN trained to perform IR of binary images. The red line is for translation, the blue line is for rotation. Circles indicate the calculated convergence points, T_C. The asterisks indicate the settling point, T_S and diamonds indicate the location of the minimum error.

Table 6.12 tabulates the convergence metrics for both experiments. These metrics are computed based on a batch of 50 simulations. The data shows that the learning process for both cases consistently converge with average settling times of approximately 8 and 75 epochs, respectively. The figure clearly indicates that the CSRN is learning in both cases.

Table 6.12: Summary of convergence data for image registration of binary images. PTCs for translation and rotation cases are $\theta' = 5$ pixels and $\theta' = 10°$, respectively.

Translation			
Metric	**Units**	*Ave +/- std dev*	*PTC*
T_C	*epochs*	58.02 +/- 13.09	75
T_S	*epochs*	7.72 +/- 4.95	9.0
E_{MIN}	-	10.0 +/- 0.01	10.0
T_{ME}	*epochs*	83.36 +/- 35.12	35
Rotation			
T_C	*epochs*	15.80 +/- 7.84	13.00
T_S	*epochs*	82.98 +/- 36.40	50.00
E_{MIN}	-	0.41 +/- 0.01	0.397
T_{ME}	*epochs*	174.60 +/- 24.11	199

6.4 Conclusion

In this chapter, pursuant to *goal 3* of this work, we demonstrate the CSRN's ability to learn and perform basic image registration on binary and grey-scale images subjected to rigid body transformation. We adapt the CSRN to perform registration of images under both translation and rotation. As proof of concept we demonstrate the CSRN's ability to register small, 15x15, binary test images. Utilizing the sub-image processing technique developed in Chapter 4, we demonstrate the CSRN's ability to register larger, grey-scale, facial images.

The CSRN exhibits an inability to generalize over all its training images. Approximation of the cost-function produces the same output for all images, thereby registering each input image to the mid-point of the transformation range.

The CSRN performs excellent registration for the PTC, which is equal to the mid-point of the transformation range. It achieves function accuracies of 100%, and 94.7%, for the translation and rotation cases, indicating that it is learning the geometric functions associated with registration.

The registered images produced by the CSRN for the primary test case compare favorably with their corresponding target images, as indicated by the image accuracy and image correlation ratio metrics. The CSRN achieves IM_{ACC} and IM_{CR} values of 100%, for both the translation and rotation cases. When compared to the results produced by the baseline registration method, discussed in Section 6.2.5, the CSRN's performance matches that of the baseline method.

The CSRN's learning process is stable, with the mean-squared testing error consistently converging in less than 150 epochs for both transformations.

The CSRN's forward computation times are 1.41msec and 1.55msec for the translation and rotation cases, respectively. The CSRN reduces computation times for these transforms by approximately 70% over those achieved by the baseline registration method. These faster computation times make the CSRN attractive for real-time, embedded applications.

In spite of its inability to generalize over its entire set of training images, the CSRN has demonstrated the ability to learn IR. CSRNs could be used in an on-line learning configuration where a single CSRN is trained to register a single input image, as demonstrated by Ren *et al*. [75] in their facial recognition application. The CSRN could also be used in a two-stage registration technique, where the first stage performs

parameter estimation, and the second stage performs the geometric transformation. Plans for this two-stage method are examined in a discussion of future work in Chapter 9.

7 ALTERNATIVE CORE NETWORKS

In our introduction to the CSRN in Chapter 3, we discuss the one-to-one correspondence of pixels in an image to cells of the network. Figure 3.1 shows the cellular structure of the CSRN and depicts this one-to-one relationship. Each "cell" in the structure contains an instance of the chosen core network. This cell can contain any type of NN, but the use of a SRN as the core, defines the CSRN. In this chapter, we investigate the use of alternative core networks.

Inherent to the application of any NN to the solution of a given problem is the process of "tuning" the network. "Tuning" a network is the process of selecting the internal parameters of the network in order to maximize the efficacy of the network for a selected application. As part of our investigation of various cores for the CSRN, we discuss tuning of these cores for general purpose image processing (GPIP) applications.

7.1 The GMLP Core

We introduce the GMLP core in Section 3.2. Its architecture and core network are shown in Figs. 3.2, and 3.3, respectively. We have discussed several applications for which the GMLP core is used, including the maze traversal application, Section 3.5.1, affine transformation, Section 5.2.1 and image registration, Section 6.8.1.

7.1.1 Tuning the GMLP core

In this section, we discuss selection of the GMLP core's internal parameters. These parameters are listed below:

- number of active nodes
- number of core iterations
- initial scaling weight
- number of neighbor inputs.

7.1.1.1 Number of Active Nodes

Selection of the number of active nodes involves a trade-off between computational power and network complexity. The more complex the network, the longer the forward and backward computation times resulting in longer training/testing times. Ilin *et al*. utilize between 5 and 15 active nodes for the solution of the maze traversal problem [47]. We find 5 active nodes to be adequate for the IP applications investigated herein.

7.1.1.2 Number of Core Iterations

The number of core iterations affects the resolution of the network outputs, particularly those outputs heavily time dependant. Once again there is a trade-off between computational ability and complexity. One must select a high enough number of iterations to allow the final output to settle, while not drastically degrading the speed of the network. We find that 10 iterations are sufficient for our IP applications.

7.1.1.3 Initial Scaling Weight

The scaling weight is used to scale the core's final output. Selection of an initial value for the scaling weight is a function of the geometry of the GPIP task. Ilin *et al*. report the use of 40 as the initial value of the scaling weight for their maze traversal application with 7x7 mazes [47]. Typical output values for mazes of this size range from 1 to 8, however, they assign a value of 25 to represent a cell with an obstacle, which

results in a much higher required scaling weight. In our GPIP applications, we find that the initial scaling weight depends on image size or range of intensity values.

Geometric transformations depend on image size. For these transformations, which include affine transformations and registration, the maximum movement of any given pixel in a given dimension is the size of the image in that dimension. Therefore, the initial scaling weight is set equal to the size of the image. When considering small scale angles of rotation (0° to 30°) with the center of rotation at the image's center, the maximum movement in a given dimension is ½ the image size, and the initial scaling weight is set accordingly.

For those IP applications which directly output a final image, the initial scaling weight is set equal to the upper limit of the image's intensity range. The output of the *tanh* activation function is -1 to 1. We set the intensity values for binary images equal to {-1, 1}, and set the initial value of the scaling weight, $W_s = 1$. For grey-scale images we represent the intensity value with a floating point variable scaled between (-1, 1) and set the initial value of the scaling weight, $W_s = 1$.

7.1.1.4 Number of Neighbor Inputs

The number of neighbor inputs is again affected by the geometry of the problem. In the maze traversal problem, the maze can only be traversed by moving from the current cell to one of its 4-neighbors [47], thus 4 inputs are sufficient. In Grant *et al.*'s voltage prediction application [34], the number and orientation of connections between the CSRN cells reflect the actual connections between neighboring busses in the system. Pixel transformations, discussed in Section 4.5, do not require any neighbor connections, while spatial filtering, discussed in Section 4.6, requires 8-neighbor connections.

7.2 Elman SRN

The Elman SRN (ESRN) core was briefly introduced in Section 3.6.5. Figure 3.9 depicts the ESRN core configured for a voltage prediction application. Figure 7.1 below shows the ESRN core adapted for general purpose IP tasks.

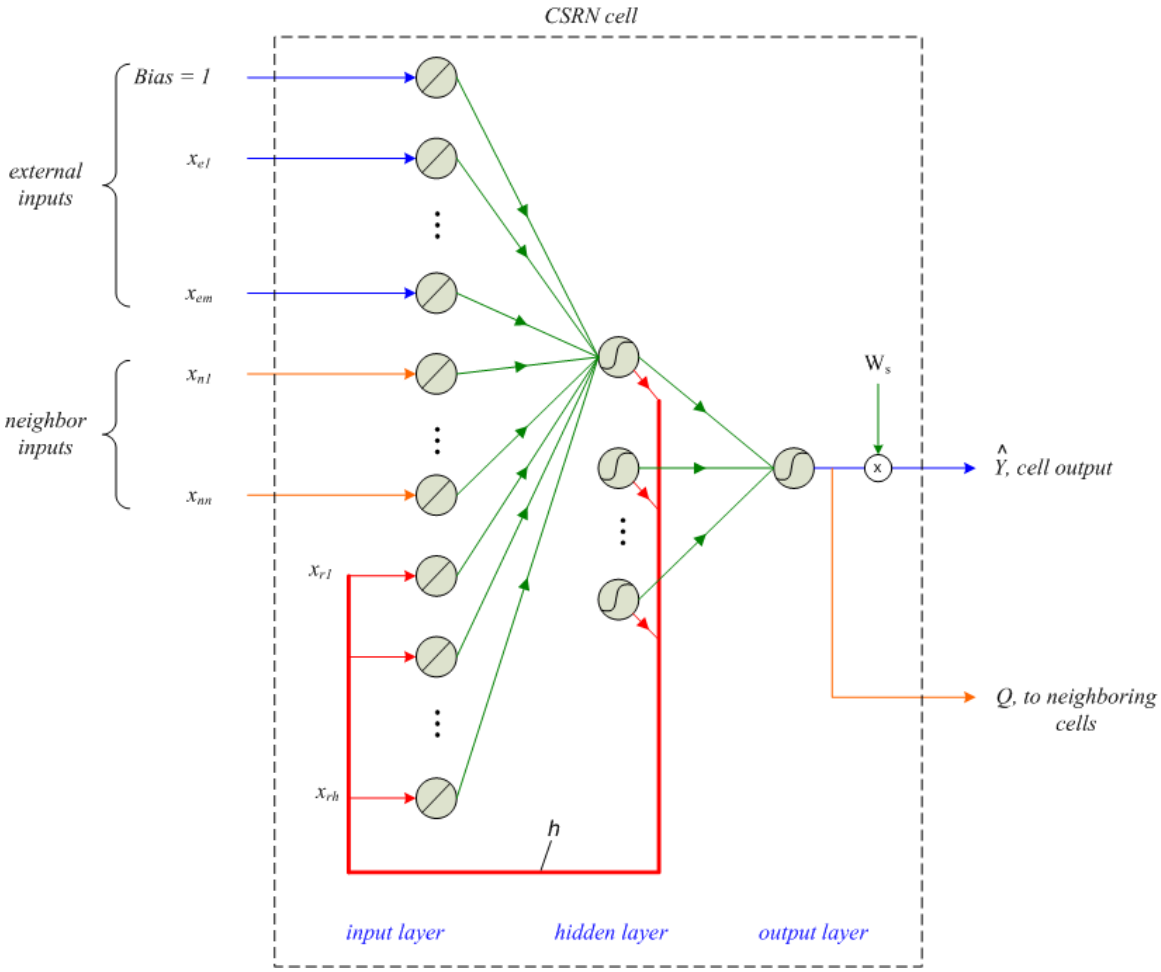


Figure 7.1: ESRN core network configured for image processing. Core shown with bias, 3 external inputs, 4 neighbor inputs, 5 self-recurrent inputs, 5 hidden nodes and one output node, for a total of 6 active nodes.

7.3 Elman SRN with Multi-layer feedback

7.3.1 Motivation

We began our investigation of CSRNs by studying Ilin *et al.*'s maze traversal application [47] which utilizes the GMLP core developed Pang *et al*. [69]. During these early simulations we noted convergence problems with the CSRN. Personal discussions with the author of [69] confirmed that the network error diverges in approximately 40% of trials. With the primary goal of improving the CSRN's performance in geometric transformations and a secondary goal of improving the CSRN's convergence, we set out to modify the CSRN's core network.

7.3.2 The Multi-layered Feedback Core

Due to its similarity to standard MLP networks, we select the ESRN described in the previous section as the basic core. In an attempt to improve the networks convergence, we add feed back from the output node, in addition to the feed back from the hidden nodes. This is done to create an "inner" feedback loop to improve network stability, in much the same way an inner velocity loop is used to stabilize an outer position loop in control theory [52] as shown in Fig. 7.2. Figure 7.3 shows the resulting core, which we refer to as an ESRN with multi-layered feedback (ESRNmlf).

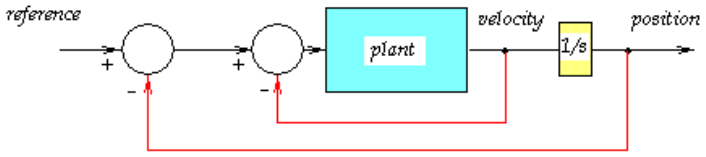


Figure 7.2: Position control loop with inner velocity loop for loop stabilization.

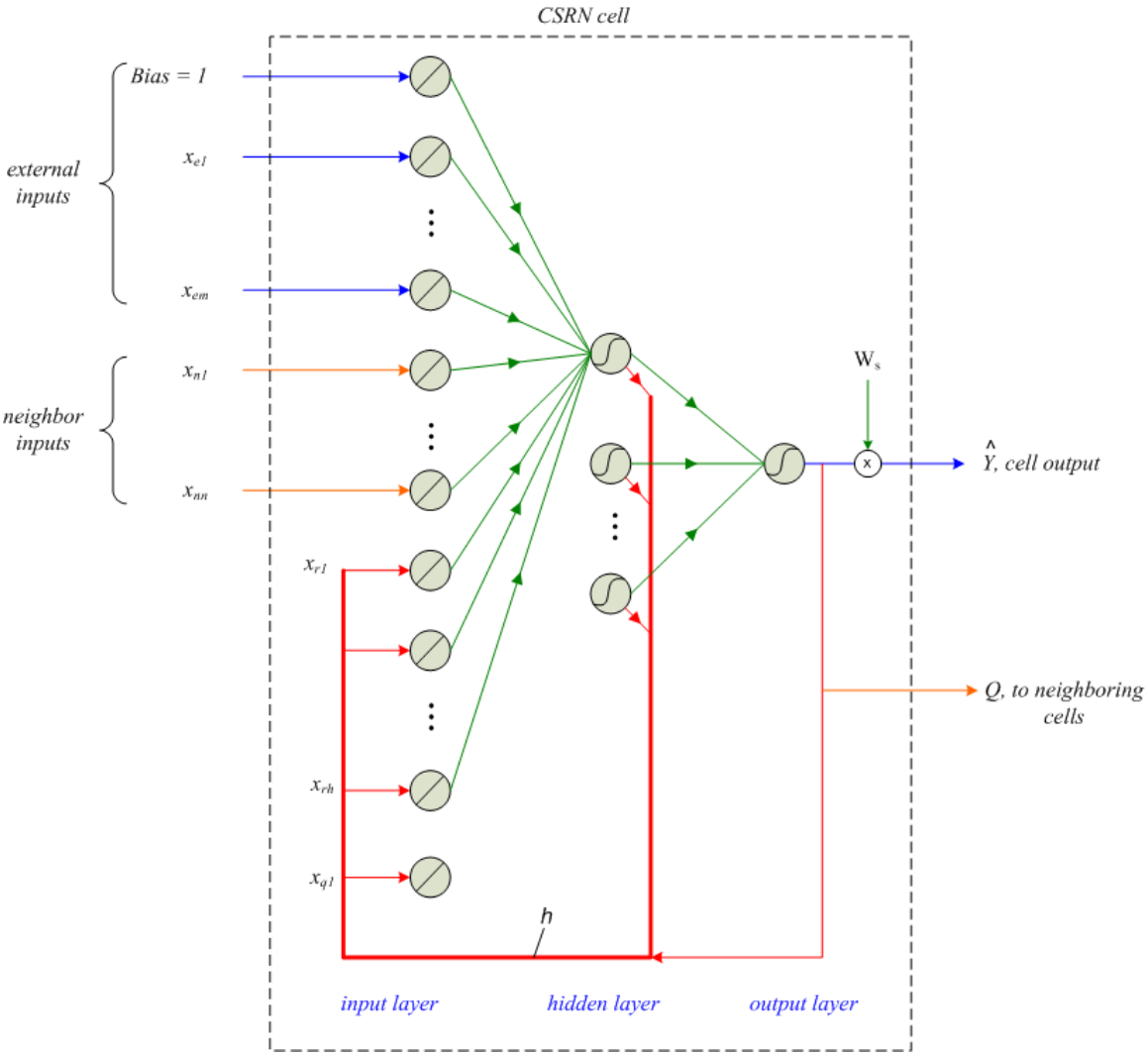


Figure 7.3: Elman SRN architecture with multi-layer feedback used for each cell in the CSRN network. Nodes are fully connected between layers. Network weights shown in blue, label with green text. Feedback paths are shown in red.

7.3.3 Core Comparisons

To compare the performance of these core networks, we examine the results of all three cores in the affine rotation application (Sections 5.2.3 and 5.2.4). Both binary and grey-scale results are presented. Each network utilizes movement encoding with inverse

mapping and is trained via the EKF method. Function and image accuracy, as well as, training and convergence times are compared.

7.3.3.1 Binary Image Results

Figure 7.4 shows the image results for each core trained to perform binary affine rotation. Results for the full testing set are shown. Row a) contains the input images, row b) shows the corresponding output images for the GMLP core, row c) shows the corresponding output images for the ESRN core, and row d) shows the corresponding output images for the ESRNmlf core. Rows e) and f) show the results of the raw transformation equations and the actual target image, respectively. A detailed view of the $\theta' = 16°$ case is shown in Fig. 7.5.

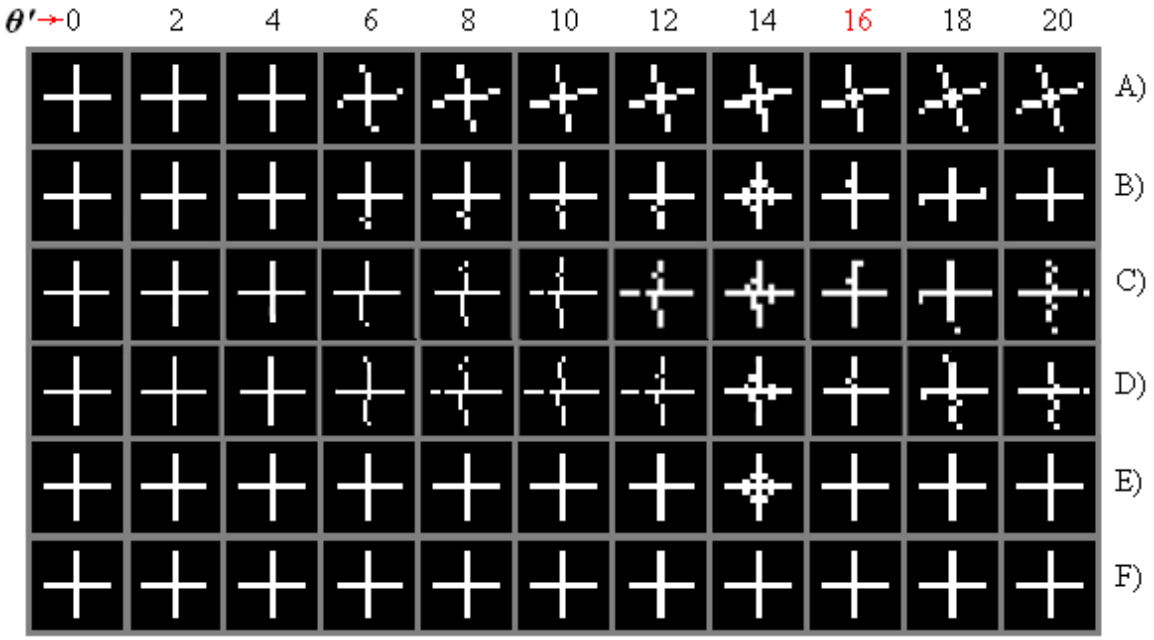


Figure 7.4: Results of rotating an 11x11 cross embedded in a 15x15 binary image. Input images are rotated through a range of $\theta' = 0°$ to 20° in the counter-clockwise direction. Results for GMLP, ESRN and ESRNmlf cores CSRN are shown. Each core utilizes movement encoding, inverse mapping and EKF training. Row A): input images. Row B): GMLP core image results. Row C): ESRN core image results. Row D): ESRNmlf core image results. Row E) results of raw transformation by (5.6). Row F): target image.

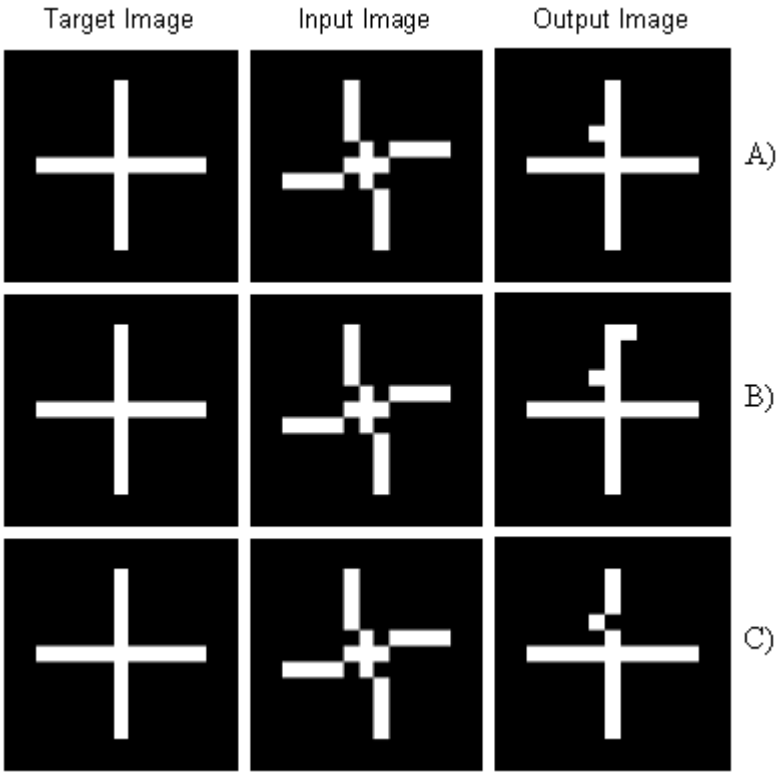


Figure 7.5: Results of binary affine rotation test for case $\theta' = 16°$. Each core utilizes movement encoding, inverse mapping and EKF training. A) Image results for GMLP core. B) Image results for ESRN core. C) Image results for ESRNmlf core.

7.3.3.2 Grey-scale Image Results

Figure 7.6 shows the image results for each core trained to perform grey-scale affine rotation. Results for the full testing set are shown. Row a) contains the input images, row b) shows the corresponding output images for the GMLP core, row c) shows the corresponding output images for the ESRN core, and row d) shows the corresponding output images for the ESRNmlf core. Rows e) and f) show the results of the raw transformation equations and the actual target image, respectively. A detailed view of the $\theta' = 16°$ test case is shown in Fig. 7.6. Table 7.1 compares the results for each core using data from this same test.

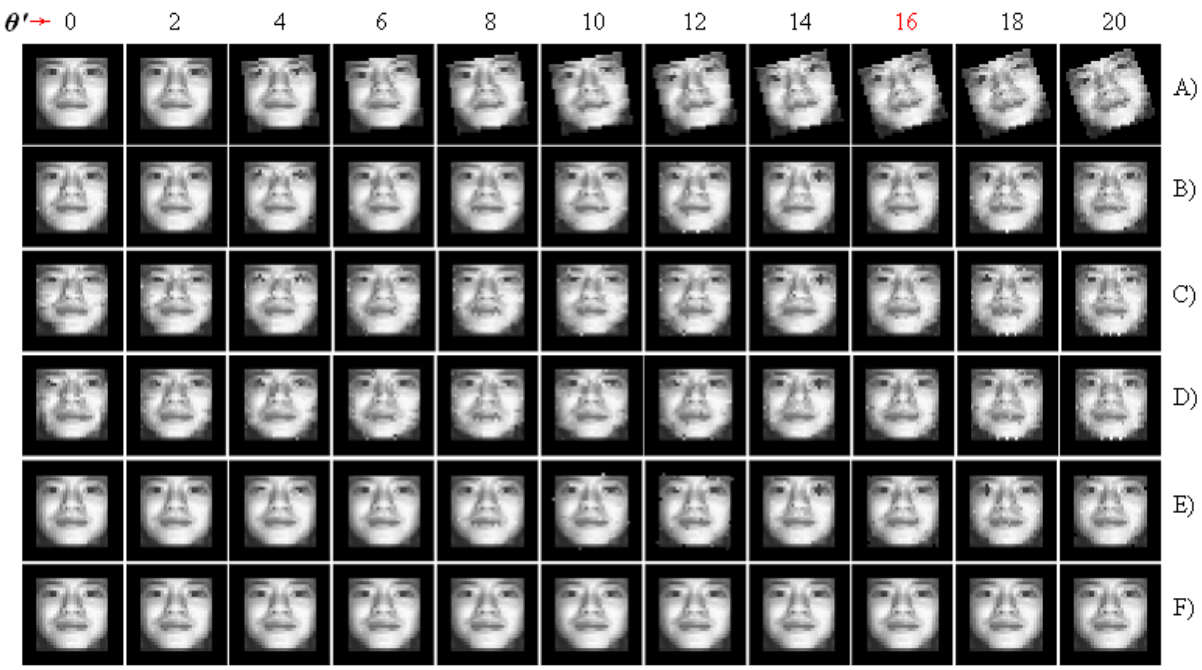


Figure 7.6: Results of rotating an 11x11 cross embedded in a 15x15 grey-scale image. Input images are rotated through a range of θ' = 0° to 20° in the counter-clockwise direction. Results for GMLP, ESRN and ESRNmlf cores CSRN are shown. Each core utilizes movement encoding, inverse mapping and EKF training. Row A): input images. Row B): GMLP core image results. Row C): ESRN core image results. Row D): ESRNmlf core image results. Row E) results of raw transformation by (5.6). Row F): target image.

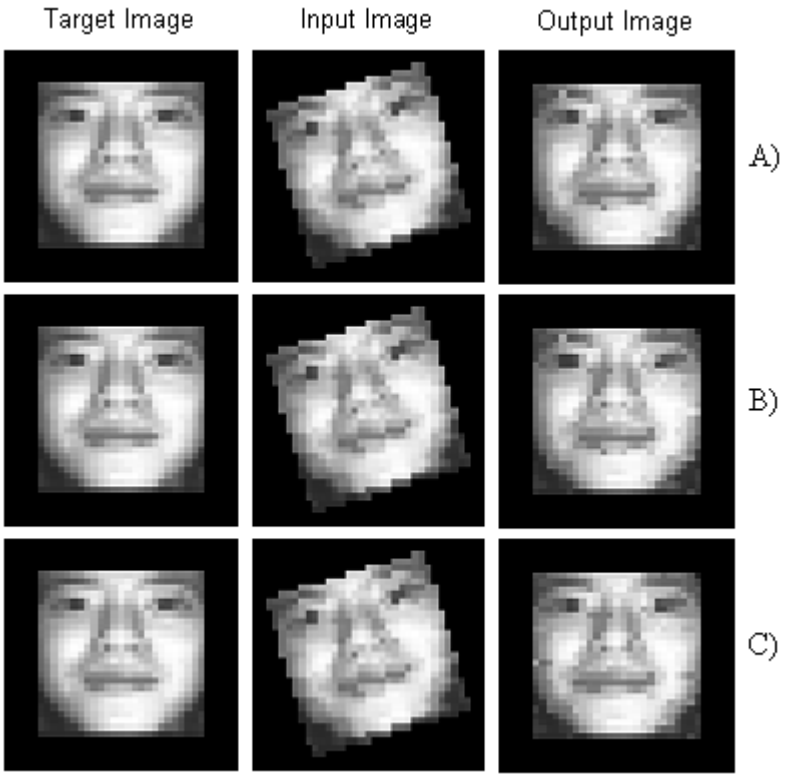


Figure 7.7: Results of grey-scale affine rotation test for case $\theta' = 16°$. Each core utilizes movement encoding, inverse mapping and EKF training. A) Image results for GMLP core. B) Image results for ESRN core. C) Image results for ESRNmlf core.

Table7.1: Comparison of binary rotation results for the CSRN GMLP,ESRN and ESRNmlf core networks. Each core utilizes movement encoding, inverse mapping and EKF training.

Metric	**Units**		***GMLP***	***ESRN***	***ESRNmlf***
$\boldsymbol{J_{ACC}}$	*%*	*Ave*	83.3	57.0	65.5
		Best	95.6	82.7	83.1
$\boldsymbol{J_{MSE}}$	-	*Ave*	2.0E-2	5.0E02	3.0E-2
		Best	3.0E--3	2.0E-2	2.0E-2
$\mathbf{IM_{ACC}}$	%	*Ave*	99.0	96.7	97.7
		Best	99.6	99.1	99.1
$\mathbf{IM_{MSE}}$	-	*Ave*	1.0E-2	3.0E-2	2.0E-2
		Best	0.0E-2	1.0E-2	1.0E-2
$\mathbf{IM_{CR}}$	-	*Ave*	0.94	0.81	0.86
		Best	0.97	0.95	0.95

Visual inspection of Figs. 7.4 through 7.7 reveals that GMLP core has the best overall performance followed by the ESRNmlf and finally the ESRN. The ESRN and ESRNmlf results show a bit more distortion in both the binary and grey-scale images, and a slight blurring in the grey-scale images. This performance trend is supported by the data in Table 7.1 with the GMLP performing best in all metrics.

7.3.3.3 Computation Times

To compare the computation times of these core networks, we examine the following metrics: forward computation time, T_{FC}, training time, T_{TR}, simulation run time, T_R, and total batch time, T_B. Table 7.2 shows these metrics for all three cores. The GMLP core is faster in all metrics, followed this time by the ESRN, then the ESRNmlf. As expected,

the addition of the multi-layered feedback causes the ESRNmlf core to be slightly slower than the standard ESRN.

Table 7.2: Comparison of computation times for the GMLP, ESRN and ESRNmlf core networks in CSRNs applied to binary rotation. Each core utilizes movement encoding, inverse mapping and EKF training. Times include: forward computation time, T_{FC}, training time, T_{TR}, run time, T_R, total batch time, T_B. Total batch time is given for a batch size of 50 simulations.

Metric	**Units**	***GMLP***	***ESRN***	***ESRNmlf***
$\boldsymbol{T_{FC}}$	*msec*	1.56	1.72	1.84
$\boldsymbol{T_{TR}}$	*sec*	576.0	599.2	608.3
$\boldsymbol{T_R}$	*min*	9.86	10.26	10.42
$\boldsymbol{T_B}$	*hour*	8.22	8.55	8.68

7.3.3.4 Core Convergence

Figure 7.8, shows plots of the testing MSE for each of the CSRN cores. These plots indicate that the all three cores exhibit learning. Table 7.3 tabulates the convergence metrics for all three cores. Metrics include: calculated convergence time, T_C, settling time, T_S, minimum error, E_{MIN}, and time of occurrence of minimum error, T_{ME}. Statistics are computed based on batches of 50 simulations. All cores are stable, consistently converging in slightly less than 200 epochs. The ESRN core converges faster than the other two cores, followed by ESRNmlf, and finally the GMLP. However, the GMLP converges to a lower error, which agrees with the results from Table 7.1.

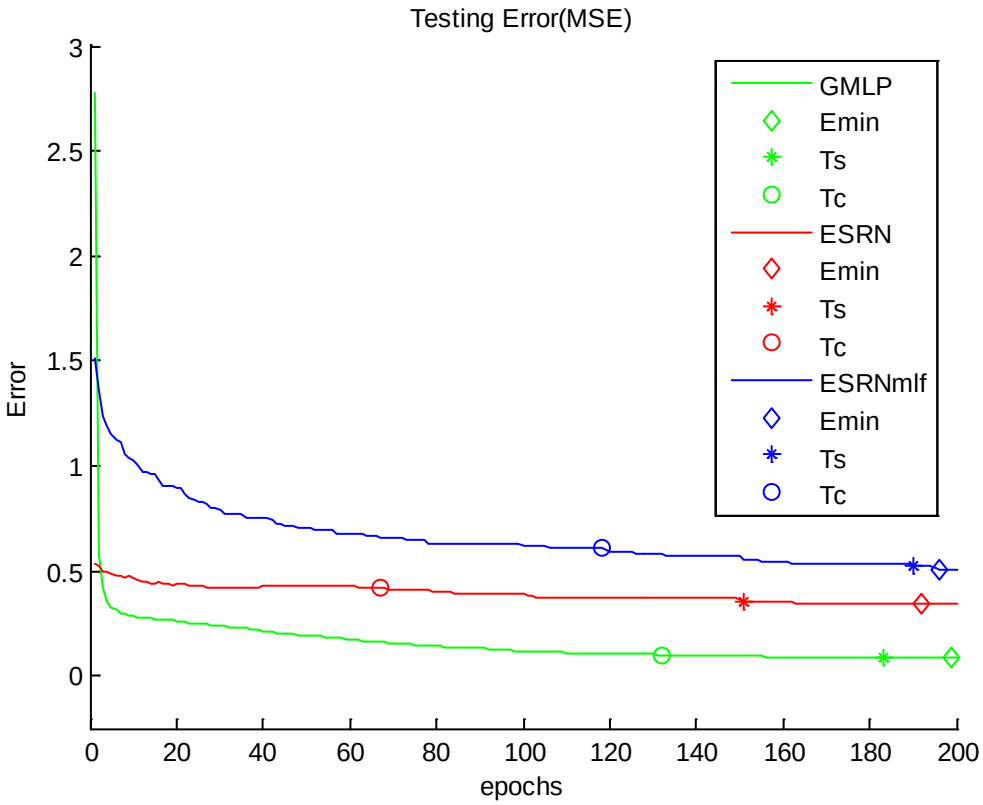


Figure 7.8: Plots of the testing MSE for the GMLP, ESRN and ESRNmlf cores trained to perform affine rotation of binary images. Each core utilizes movement encoding, inverse mapping and EKF training. The green line represents the GMLP core. The red line represents the ESRN core, and the blue line the ESRNmlf core. Circles indicate the calculated convergence points, T_C. Asterisks indicate settling points, T_S and diamonds indicate locations of the minimum error.

Table 7.3: Comparison of convergence of mean square testing error for GMLP, ESRN and ESRNmlf cores trained to perform affine rotation of binary images. Each core utilizes movement encoding, inverse mapping and EKF training. Metrics include: calculated convergence time, T_C, settling time, T_S, minimum error, E_{MIN}, and time of occurrence of minimum error, T_{ME}.

Metric	**Units**		***GMLP***	***ESRN***	***ESRNmlf***
$\boldsymbol{T_C}$	*epochs*	*Ave*	100.4	95.6	99.4
		Best	96	67	118
$\boldsymbol{T_S}$	*epochs*	*Ave*	163.8	124.8	134.9
		Best	181	151	190
$\boldsymbol{E_{MIN}}$	-	*Ave*	0.19	0.82	0.66
		Best	1.07E-1	3.42E-1	3.42E-1
$\boldsymbol{T_{ME}}$	*epochs*	*Ave*	195.8	154.3	169.2
		Best	199	192	196

7.4 Conclusions

Of the three cores tested, the GMLP core has the best overall performance in speed function approximation, and final image result, followed by the ESRNmlf and finally the standard ESRN.

Our initial hope of improving the convergence and/or stability of the CSRN by creating a core with multi-layered feedback turned out to be unnecessary, as all three cores, including the GMLP, exhibit stability in their application to geometric transformations. This stability is more a function of the application rather than the CSRN architecture or its core network. The cost-function associated with the linear, 2D functions of geometric transformations, (see section 5.11), is much less complex and more closely matched to the inherent cost-function for learning in the CSRN, than that of the maze traversal problem, (see section 3.4). However, the experience of creating and

testing a new core for the CSRN provided valuable insight into the inner workings of the CSRN and a thorough understanding of the BPTT learning method that made its initial development possible.

8 TRAINING METHODS FOR CSRN

In this chapter, we examine methods for training the CSRN. Existing methods are discussed and a new method based on parameter estimation via unscented Kalman filtering (UKF) is presented. Finally, this new method is compared against the state-of-the-art, that of parameter estimation via extended Kalman filtering (EKF), developed by Ilin *et al*. [47][48].

8.1 Supervised Learning

To date, all training methods for CSRNs utilize supervised learning. The goal of any training method which uses supervised learning is to minimize some error function over a given training set. Recall from Section 3.3 that the sum-squared error is used in training of CSRNs. For each input in the training set, we apply the inputs, X[k], to the network, calculate the network's output, Y[k], and then obtain the error by subtracting the known target T[k] from the network's output. The sum-squared error over the entire training set is given by,

$$E = \frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{q}(Y_i[k] - T_i[k])^2, \qquad . \qquad (8.1)$$

where q is the number of outputs and N is the number of data points in the training set.

The classic approached to training of NNs via supervised learning is to calculate $\frac{\partial E}{\partial w}$ so that we can adjust the weights via the following learning rule,

$$w_{ij}^{new} = w_{ij}^{old} - \gamma \cdot \frac{\partial E}{\partial w_{ij}} \tag{8.2}$$

where w_{ij} is the weight connected to node i from node j and γ is the learning rate [6].

8.2 Backpropagation through Time

The original CSRN developed by Pang *et al*. is trained using an extension of basic back-propagation known as back-propagation through time (BPTT) [69][92]. This method of training is fundamental to CSRNs. Prior to BPTT, no tractable method of computing the required derivatives existed. A tutorial for the application of BPTT to CSRNs is included as Appendix A. This tutorial includes mathematical derivation and basic algorithms,

8.3 Parameter Estimation via Extended Kalman Filter

Parameter estimation with Kalman filters has been applied to the training of NN in the past [37]. Ilin *et al*. apply parameter estimation using an extended Kalman filter (EKF) to the training of CSRNs [47]. In this method, the required Jacobian is computed via BPTT, and the weights are adapted via EKF. Ilin *et al*. apply this training method to CSRNs adapted for the maze traversal problem, discussed in Section 3.5.1, and compare the results to that of the CSRN trained via BPTT. For training of a single maze, the authors report that the CSRN trained via EKF converges within 10-15 epochs while BPTT requires between 500 to 1000 epochs [47]. This reduction in training time is a breakthrough in the training of CSRNs which make them tractable in complex tasks such as image processing and establishes the EKF method as ‘state-of-the-art’ for training of CSRNs.

8.3.1 Extended Kalman Filter

The EKF can be used for parameter estimation of non-linear systems. This method applies a first-order, Taylor-series expansion to linearize the non-linear system. Once this is done the standard Kalman filter is applied [87]. The state transition and observation equations are given as,

$$W_{t+1} = W_t + \gamma_t \tag{8.3}$$

and,

$$Y_{t+1} = F(W_t, u_t) + \eta_t, \tag{8.4}$$

where, W_t represents the system state (weights) at time t, and γ_t the process noise, Y_{t+1} is the observation at time $t+1$, F is the forward function of the NN, and η_t is the measurement noise. This method computes a new mean and covariance for the state, given the existing state and a new measurement update using the equations below.

$$\Gamma_t = C_t K_t C_t^T + R_t, \tag{8.5}$$

$$G_t = K_t C_t^T \Gamma_t^{-1}, \tag{8.6}$$

$$W_{t+1} = W_t + G_t a_t, \tag{8.7}$$

and

$$K_{t+1} = K_t - G_t C_t K_t + Q_t, \tag{8.8}$$

where W is the mean state matrix, K is the state covariance, R and Q are the covariance of the zero mean process noise and measurement noise, respectively. The innovation, a, is the error between predicted measurement (based on the current state) and the new measurement. C is the system Jacobian and G is the Kalman gain.

8.3.2 EKF Training Method

To apply the EKF method to training of the CSRN, we let the state of the system represent the weights of the network. The process noise is set to 0, and the measurement noise is defined as a constant zero mean Gaussian. Noise annealing may be applied to the measurement noise as well. The network output is used as the measurement. The EKF method can be used in sequential or multi-streaming mode [23][37][47]. In the sequential mode, a weight correction is made for each individual training image. In the multi-streaming mode, a row is added to the Jacobian for each training image, and one update is made for all training images. The multi-streaming mode is computational more efficient and we have used it exclusively with the EKF method.

The algorithm for using the EKF method for training of the CSRN is given below.

Algorithm 8.1: EKF training algorithm

1) Randomly select a set of initial weights, W.
2) Compute an initial covariance matrix, K.
3) For each epoch
4) For each training image
5) Compute the output of the network.
6) Compute the innovation. (The error between the target output and network output.
7) Utilize BPTT to compute the Jacobian.
8) Add a row to the Jacobian.
9) Adapt weights using the EKF equations (8.5-8.8)

8.3.2.1 Computation of Jacobian via BPTT

As can be seen in Appendix A, the BPTT method yields, $\frac{\partial E}{\partial W}$, where E is the error and W the weights. The EKF method described here requires the Jacobian, $\frac{\partial \hat{Y}}{\partial W}$, where $\hat{Y}$ is the network output. Therefore, we must adapt the BPTT method to produce the required Jacobian. Consider the following,

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial \hat{Y}} \cdot \frac{\partial \hat{Y}}{\partial W}. \tag{8.9}$$

Now the error is defined as,

$$E_i \equiv \frac{1}{2}(\hat{Y}_i - T_i) \tag{8.10}$$

and

$$\frac{\partial E_i}{\partial \hat{Y}_i} = F\_\hat{Y}_i = \hat{Y} - T_i. \tag{8.11}$$

Substituting (8.11) into (8.9) yields

$$\frac{\partial E}{\partial W} = F\_\hat{Y} \cdot \frac{\partial \hat{Y}}{\partial W}. \tag{8.12}$$

By forcing $F\_\hat{Y} = 1$, we see that the BPTT computation produces the required Jacobian as follows,

$$\frac{\partial E}{\partial W} = \frac{\partial \hat{Y}}{\partial W}. \tag{8.13}$$

8.4 Parameter Estimation via Decoupled EKF

Puskorius *et al*. apply a decoupled EKF (DEKF) to the training of layered feed-forward networks in [72]. In the DEKF method, the network weights are broken into *g* groups and the weight adjustment is performed for each group, individually. This technique results in the inversion of a smaller Γ_t matrix, which, in turn, improves the

computational efficiency of the algorithm. The equations for the DEKF algorithm are shown below.

$$\Gamma[t] = R[t] + \sum_{i=1}^{g} C_i[t] \cdot K_i[t] \cdot C_i^T[t], \tag{8.14}$$

$$G_i[t] = K_i[t] \cdot C_i^T[t] \cdot \Gamma[t]^{-1}, \tag{8.15}$$

$$W_i[t+1] = W_i[t] + G_i[t] \cdot a_i[t], \tag{8.16}$$

and

$$K_i[t+1] = K_i[t] - G_i[t] \cdot C_i[t] \cdot K_i[t] + Q_i[t], \tag{8.17}$$

where W is the mean state matrix, K is the state covariance, R and Q are the covariance of the zero mean process noise and measurement noise, respectively. The innovation, a, is the error between predicted measurement (based on the current state) and the new measurement. C is the system Jacobian and G is the Kalman gain.

8.4.1 Training the CSRN via DEKF

Rice *et al*. apply the DEKF to the training of CSRNs for the maze traversal problem in [77]. In this technique, they decouple the EKF, not around the weights, but around the inputs to the CSRN. This results in the same reduction in size of the Γ_t matrix and therefore, the same improvement in computation efficiency. The equations for Rice *et al*.'s version of the DEKF are given below,

$$\Gamma_i[t] = C_i[t] \cdot K_i[t] \cdot C_i^T[t] + R[t], \tag{8.18}$$

$$G_i[t] = K_i[t] \cdot C_i^T[t] \cdot \Gamma_i[t]^{-1}, \tag{8.19}$$

$$W[t+1] = W[t] + \sum_{i=1}^{g} G_i[t] \cdot a_i[t], \tag{8.20}$$

and

$$K[t+1] = K[t] - \sum_{i=1}^{g} G_i[t] \cdot C_i[t] \cdot K_i[t] + Q[t], \tag{8.21}$$

where, the variables are defined as in (8.14)-(8.17).

The algorithm for using the DEKF method for training of the CSRN is given below.

Algorithm 8.2: DEKF training algorithm

1) Randomly select a set of initial weights, W.
2) Compute an initial covariance matrix, K.
3) For each epoch
4) For each training image
5) Compute the output of the network.
6) Compute the innovation. (The error between the target output and network output.
7) Utilize BPTT to compute the Jacobian.
8) Compute adjustments to weight and covariance matrices (8.18 - 8.21) only computing 2nd term of (8.20).
9) Adapt weights by completing (8.20).

Rice *et al.* [77] report an average improvement of 12.9 times faster in training times of the DEKF method over that of the standard EKF method. They also report their final implementation, which includes improved methods for inversion of the Γ_t matrix, produces training times 480 times faster than that of the standard EKF method.

8.5 Small Population Particle Swarm Optimization

8.5.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) was developed and applied to the training of NNs in 1995 by Kennedy and Eberhart [53]. PSO is an evolutionary computation technique similar to the Genetic Algorithm (GA). It is a population based optimization tool based on models of the behavior of a flock of birds [18].

Let us consider the application of PSO to the training of a NN. If a network has W weights, then the selection of an optimum set of weights becomes a search through a W-dimensional weight space for a location, i.e. a set of weights, which minimizes some fitness criteria. The fitness criteria are typically the SSE or MSE between known target outputs and actual network outputs for a given set of training inputs. In the PSO algorithm, we create a swarm of agents known as *particles* that "fly" through the weight space, evaluating locations as they move. Each particle is initially given a random position, $X_i(0)$, and velocity, $V_i(0)$ within the weight space. Each particle 'remembers' its best fitness value, $P_{best.}$, denoted as p_i, and its location, X_i^p. The particle has access to the best fitness value within the swarm, $G_{best,}$ and its location. These are referred to as p_g and X_g, respectively. These memories represent the collective memory of the swarm. The essence of the PSO algorithm is the acceleration of each particle toward X_i^p and X_g during

each iteration. Equations (8.22) and (8.23) below form the essential update equations for velocity and position.

$$V_i(k+1) = w \cdot V_i(k) + c_1 \cdot rand_1\left(X_i^p(k) - X_i(k)\right) + c_2 \cdot rand_2\left(X_g(k) - X_i(k)\right) \tag{8.22}$$

and

$$X_i(k+1) = X_i(k) + V_i(k+1) \quad . \tag{8.23}$$

where X_i and V_i are the position and velocity of the current particle . X_i^p is the position of the current particles best fitness value, and X_g is the location of the swarm's best fitness value. W, c_1, c_2 are importance weights for their corresponding terms and affect search characteristics and conversion. $Rand_1$ and $rand_2$ are two uniform functions which utilize a unit interval.

The algorithm for implementation of the PSO training method is given below.

Algorithm 8.3: PSO training algorithm

1) Initialize a swarm of particles with random positions and velocities within the *W*-dim weight space. A particles position is given by X_i and its velocity is denoted as V_i. Note that X_i is a *W-tuplet* and V_i is an *W*-D vector.

2) For each particle, evaluate the fitness function.

3) Compare each particles fitness value, f_i, with that of its best fitness value, p_i. If the current value is better than its best value, then set $p_i = f_i$ and $X_i^p = X_i$, the particles current location within the weight space.

4) Compare each particles best value, p_i with the swarms best value, p_g. If any of the p_i values is better than p_g then set $p_g = p_i$ and capture its position in X_g.

5) Compute the new velocity and position of each particle.

6) Repeat from step 2 until the stopping condition is met. The stopping condition is generally a sufficient fitness measure or a maximum number of iterations.

8.5.2 SPPSO

SPPSO utilized the same algorithm as PSO, with the addition of a regenerative technique. In SPPSO the position and velocity of the particles are re-initialized every *n* iterations. The P_{best} and G_{best} values are maintained. This technique allows for fewer particles than that of the PSO algorithm. Gudise *et al*. show PSO to be a faster training method than standard BPTT [35].

8.6 Parameter Estimation via UKF

In this section, we apply the UKF to the training of CSRNs. Use of the UKF eliminates the need to compute the Jacobian required by the EKF. We first present an overview of the unscented transform. Next, we present the UKF method of parameter

estimation for training of NNs. Following this, we discuss adaption of the UKF method for training of CSRNs and show results of CSRNs trained via the UKF method for the affine rotation application. We compare these results with those of the CSRN trained via the EKF method.

8.6.1 The Unscented Transform

The unscented transform (UT) is a method for computing the statistics of a random variable that has undergone a nonlinear transformation [50][51]. Consider an *n*-dimensional Gaussian random variable, x, with mean, μ_x, and covariance, P_x, which is transformed via an arbitrary non-linear function, g, such that,

$$y = g(x). \tag{8.24}$$

Instead of approximating g with a Taylor-series expansion, as in the EKF, the UT deterministically computes a minimal set of weighted sample points. These samples, referred to as *sigma points*, capture the true mean and covariance of the prior, x, and when passed through the true nonlinear function, g, capture the mean and covariance of the posterior, y, with a minimum accuracy of a 2nd order Taylor-series expansion and a 3rd order accuracy for Gaussian priors [60].

In order to accurately capture the statistics of the prior, we must select $2n+1$ sigma points expressed as, $\mathcal{Z}^{[i]}$. The sigma points are located at the mean and symmetrically along the main axes of covariance, two per dimension. They are computed according to the following:

$$\mathcal{Z}^{[0]} = \mu_x, \tag{8.25}$$

$$\mathcal{Z}^{[i]} = \mu_x + \left(\sqrt{(n+\lambda)\cdot P_x}\right)_i, \qquad for\ i = 1 \ldots n; \tag{8.26}$$

$$\mathcal{Z}^{[i]} = \mu_x - \left(\sqrt{(n+\lambda)\cdot P_x}\right)_{i-n}, \qquad for\ i = n+1 \ldots 2n; \tag{8.27}$$

$$\lambda = \alpha^2 (n+k) - n, \tag{8.28}$$

where n is the number of states, $\boldsymbol{Z}^{[i]}$ is the ith sigma point, μ_x and P_x are the mean and covariance of the current state and α and k are scaling parameters that determine how far the sigma points are spread from the mean.

Each sigma point has two weights associated with it. These are $w_m^{[i]}$ and $w_c^{[i]}$ and are respectively used for recovery of the mean and covariance of the Gaussian RV. These weights are computed as follows,

$$w_m^{[0]} = \frac{\lambda}{n+\lambda}, \tag{8.29}$$

$$w_c^{[0]} = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta), \tag{8.30}$$

$$w_m^{[i]} = w_c^{[i]} = \frac{1}{2(n+\lambda)}, \qquad for\ i = 1 \ldots 2n, \tag{8.31}$$

where n is the number of states, $w_m^{[i]}$ and $w_c^{[i]}$ are the mean and covariance weights of the i^{th} sigma point, and β is a parameter selected to encode prior knowledge about the distribution of the underlying Gaussian. $\beta = 2$ is optimal for an exact Gaussian.

The sigma points are now passed through the true nonlinear function, *g*, in affect probing how *g* changes the shape of the prior, *x*, as follows:

$$y^{[i]} = g(\mathcal{Z}^{[i]}). \tag{8.32}$$

The parameters of the posterior, *y*, are then extracted from the transformed sigma points, $y^{[i]}$, using their associated weights as below,

$$\mu_y = \sum_{i=0}^{2n} w_m^{[i]} \cdot y^{[i]}, \tag{8.33}$$

$$\text{and } P_y = \sum_{i=0}^{2n} w_c^{[i]} \cdot \left(y^{[i]} - \mu_y\right)\left(y^{[i]} - \mu_y\right)^T. \tag{8.34}$$

8.6.2 Training of NNs via UKF Parameter Estimation

The UKF is a straightforward application of the UT. In order to utilize the UKF to train a NN we must couch the problem as one of parameter estimation. In this case we allow the weights to represent the state of the system and utilize the following system equations,

$$w_{t+1} = w_t + \varepsilon_t, \tag{8.35}$$

$$y_t = G(w_t, u_t) + \delta_t, \tag{8.36}$$

where (8.35) and (8.36) are the state transition and measurement equations, respectively. ε_t represents the process noise and δ_t the measurement noise. ε_t and δ_t are both Gaussian additive noise with zero mean and covariance Q_t and R_t, respectively. $\boldsymbol{G}(\cdot)$ represents the forward computation of the NN and y_t, the observation, is the output of the NN.

The UKF implements a Bayesian filter using the UT. The current belief about the system state is represented by its mean, μ_t^w, and covariance, P_t^w. We start with the previous belief, $\left(\mu_{k-1}^w, P_{k-1}^w\right)$, the current input to the NN, u_k, and the known target output, T_k. Recall that we are utilizing supervised learning.

First, we perform the prediction step of a Bayesian filter using (8.35) which yields,

$$\overline{\mu}_k^w = \mu_{k-1}^w, \tag{8.37}$$

and

$$\overline{P}_k^w = P_{k-1}^w + Q_{k-1}. \tag{8.38}$$

Then we extract the sigma points using (8.24)-(8.28) and their associated weights via (8.29)-(8.31) which yields,

$$\mathcal{Z}_k = \left[\overline{\mu}_k^w \quad \overline{\mu}_k^w + \gamma\sqrt{\overline{P}_k^w} \quad \overline{\mu}_k^w - \gamma\sqrt{\overline{P}_k^w}\right] \tag{8.39}$$

where, $\gamma = \sqrt{n+\lambda}$. Now we perform the measurement update step by passing the sigma points through the measurement equation (8.36) as follows,

$$\boldsymbol{Y}_{\boldsymbol{k}}^{[i]} = \boldsymbol{G}(\boldsymbol{Z}_k^{[i]}, u_k), \tag{8.40}$$

and extract the updated belief's statistics using (8.33) and (8.34) of the UT. This yields,

$$\mu_k^y = \sum_{i=0}^{2n} w_m^{[i]} \cdot \boldsymbol{Y}_k^{[i]}, \tag{8.41}$$

$$P_k^y = \sum_{i=0}^{2n} w_c^{[i]} \cdot \left(y_k^{[i]} - \mu_k^y\right)\left(y_k^{[i]} - \mu_k^y\right)^T + R_t, \tag{8.42}$$

and

$$P_k^{w,y} = \sum_{i=0}^{2n} w_c^{[i]} \cdot \left(\boldsymbol{Z}_k^{[i]} - \overline{\mu}_k^w\right)\left(y_k^{[i]} - \mu_k^y\right)^T. \tag{8.43}$$

Now compute the Kalman gain as,

$$K_k = P_k^{w,y} \cdot \left(P_k^y\right)^{-1}. \tag{8.44}$$

Finally, we perform the estimation update,

$$\mu_k^w = \overline{\mu}_k^w + K_k\left(T_k - \mu_k^y\right), \tag{8.45}$$

and

$$P_k^w = \overline{P}_k^w + K_k \cdot P_k^y \cdot K_k^T. \tag{8.46}$$

8.6.3 Training the CSRN via the UKF Method

The algorithm for using the UKF method for training of the CSRN is given below.

Algorithm 8.4: UKF training algorithm

10) Randomly select a set of initial weights, w_0 and set $\mu_{k-1}^w = w_0$.

11) Compute an initial covariance matrix, P_0^w and set $P_{k-1}^w = P_0^w$.

12) For each epoch

13) For each training image

14) Perform the predication step. (8.37) & (8.38)

15) Select sigma points. (8.24)-(8.28)

16) Compute sigma point weights. (8.29)-(8.31)

17) Perform the measurement update, i.e. compute the forward computation of the CSRN. (8.40)

18) Compute the statistics for the update. (8.41)-(8.43).

19) Compute the Kalman gain. (8.44)

20) Perform the estimation update. (8.45) & (8.46)

8.6.4 UKF Results

In order to evaluate the efficacy of our new UKF based training method, we repeat the affine rotation experiments performed in Sections 5.2.3 and 5.2.4. The network used for these experiments is the CSRN using a GMLP core with movement encoding and inverse mapping. We compare these results with those of the same network trained via the EKF method. Function and image accuracy, as well as, training and convergence times are compared.

8.6.3.1 Binary Results

Figure 8.1 shows the image results for the CSRN trained to perform affine rotation of binary images using both the UKF and EKF methods. Results for the full testing set are shown. Row a) contains the input images, row b) shows the corresponding CSRN output images trained via the UKF method, row c) shows the corresponding CSRN output images trained via the EKF method, row d) shows the results of the raw transformation equations, and row e) shows the actual target image. A detailed view of the $\theta' = 16°$ case is shown in Fig. 8.2.

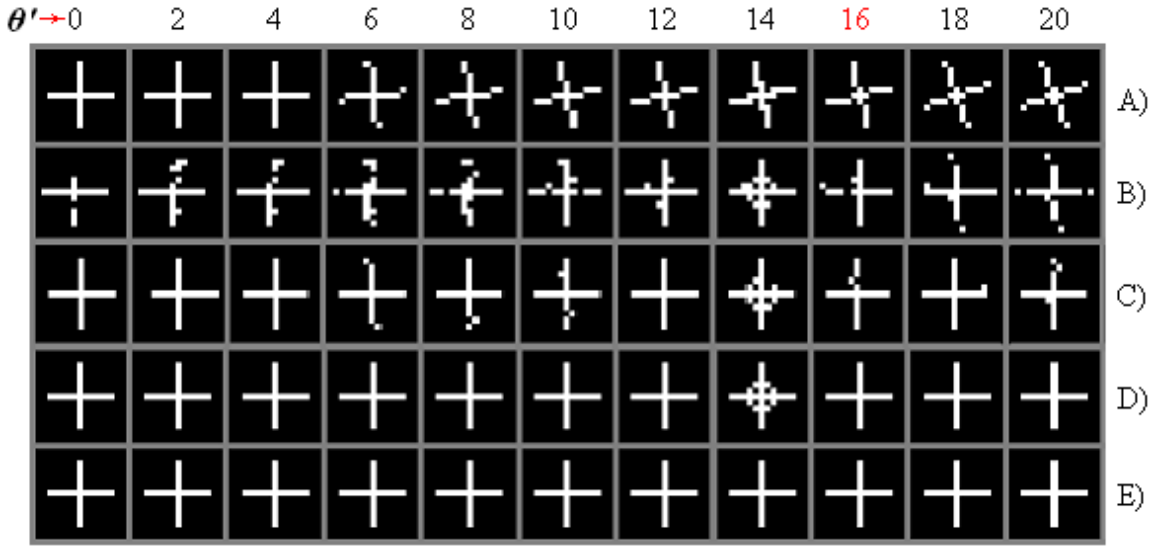


Figure 8.1: Results of rotating an 11x11 cross embedded in a 15x15 binary image. Input images are rotated through a range of $\theta' = 0°$ to 20° in the counter-clockwise direction. CSRN utilizes GMLP core, movement encoding and inverse mapping. Row a): input images. Row b): CSRN output images trained via UKF method. Row c): CSRN output images trained via EKF method. Row d): results of raw transformation by (5.6). Row e): target image.

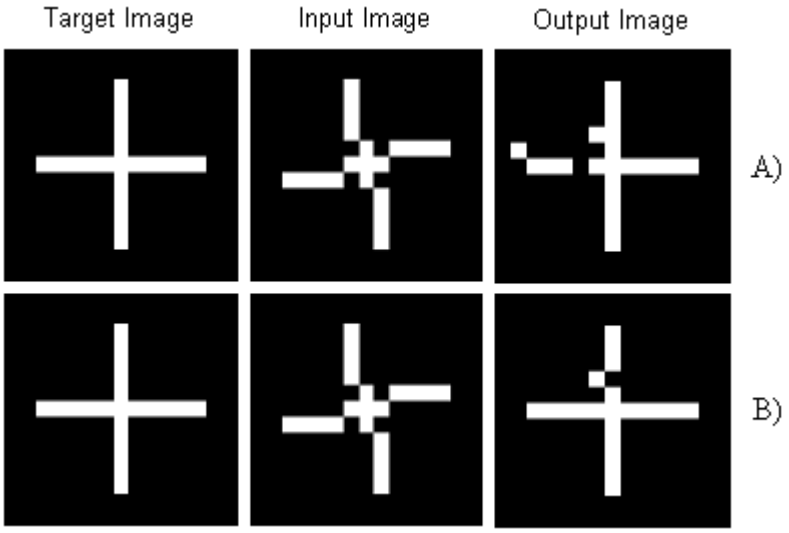


Figure 8.2: Results of binary affine rotation test for case $\theta' = 16°$. A) Image results for CSRN trained via UKF. B) Image results for CSRN trained via EKF. CSRN utilizes GMLP core, movement encoding and inverse mapping.

Table 8.1 tabulates the results of the binary image rotation experiment for a CSRN trained using the UKF training method. Results for the full testing set are shown. The θ' = 16° case is highlighted. In this case the network achieves a best case function accuracy of J_{ACC} = 63.6%, an image accuracy of IM_{ACC} = 98.7%. Table 8.2 summarizes the results of this rotation experiment for binary images.

Table 8.1: Binary image rotation results for CSRN trained via UKF. CSRN utilizes GMLP core, movement encoding and inverse mapping. Data for full testing set.

	Binary Rotation Results (full test set)				
θ'	J_{MSE}	J_{ACC}	IM_{MSE}	IM_{ACC}	IM_{CR}
0	7.03E-02	22.7	8.00E-02	92.0	0.4864
2	6.46E-02	28.9	7.11E-02	92.9	0.5978
4	6.26E-02	31.1	6.22E-02	93.8	0.6482
6	3.72E-02	59.1	5.33E-02	94.7	0.6987
8	2.46E-02	72.9	4.89E-02	95.1	0.7305
10	1.86E-02	79.6	2.67E-02	97.3	0.8500
12	1.90E-02	79.1	1.78E-02	98.2	0.9075
14	2.75E-02	69.8	3.56E-02	96.4	0.8341
16	**3.31E-02**	**63.6**	**1.33E-02**	**98.7**	**0.9232**
18	4.00E-02	58.7	3.11E-02	96.9	0.8295
20	5.78E-02	52.4	4.44E-02	95.6	0.7374

Table 8.2: Summary of binary image rotation for CSRN trained via UKF. CSRN utilizes GMLP core, movement encoding and inverse mapping. Statistics computed from 50 simulations.

Binary Rotation		**UKF Training**	
Metric	**Units**	*Ave +/- std dev*	*Best*
$\boldsymbol{J_{ACC}}$	*%*	51.42 +/- 4.90	63.6
$\boldsymbol{J_{MSE}}$	-	0.05 +/- 0.01	0.03
$\mathbf{IM_{ACC}}$	*%*	96.02 +/- 1.24	98.7
$\mathbf{IM_{MSE}}$	-	0.04 +/- 0.01	0.01
$\mathbf{IM_{CR}}$	-	0.76 +/- 0.07	0.92
$\boldsymbol{T_{TR}}$**(total)**	*sec*	726.88 +/- 7.35	733
$\boldsymbol{T_{TR}}$**(norm)**	*msec*	296.08 +/- 2.97	296
$\boldsymbol{T_{R}}$	*min*	12.21 +/- 0.12	9.86
$\boldsymbol{T_{B}}$	*hrs*	*n/a*	10.18

8.6.3.2 Grey-scale Results

Figure 8.3 shows the image results for the CSRN trained to perform affine rotation of grey-scale images using both the UKF and EKF methods. Results for the full testing set are shown. Different rows in Fig. 8.3 show the following: a) the input images, row b) the corresponding CSRN output images trained via the UKF method, row c) the corresponding CSRN output images trained via the EKF method, row d) the results of the raw transformation equations, and row e) the actual target image. A detailed view of the θ' = 16° case is shown in Fig. 8.4.

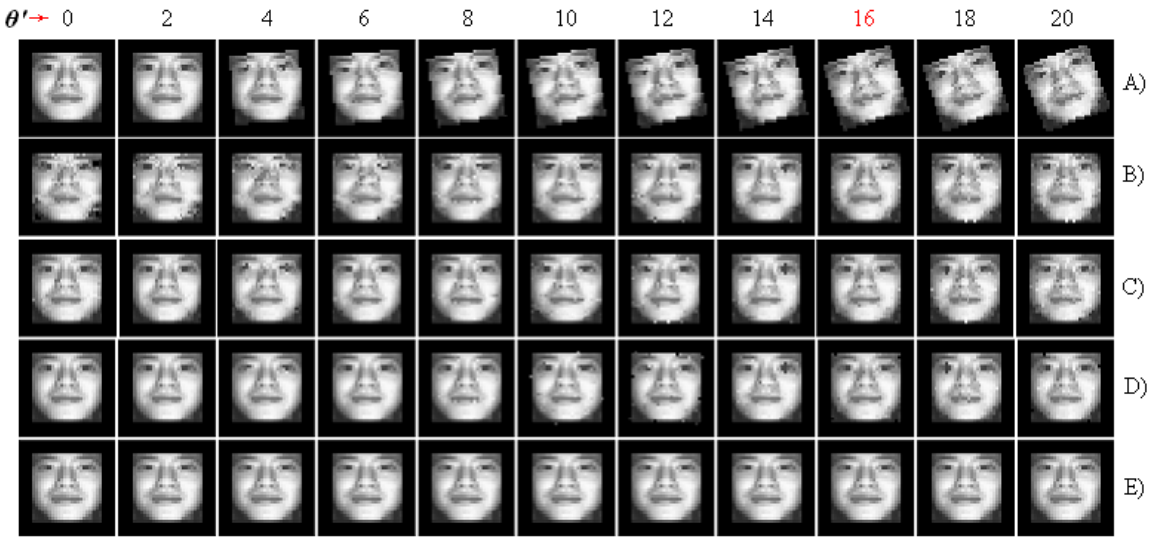


Figure 8.3: Results of rotating a 25x25 face embedded in a 35x35 grey-scale image. Input images are rotated through a range of $\theta' = 0°$ to 20° in the counter-clockwise direction. CSRN utilizes GMLP core, movement encoding and inverse mapping. Row a): input images. Row b): CSRN output images trained via UKF method. Row c): CSRN output images trained via EKF method. Row d): results of raw transformation by (5.6). Row e): target image.

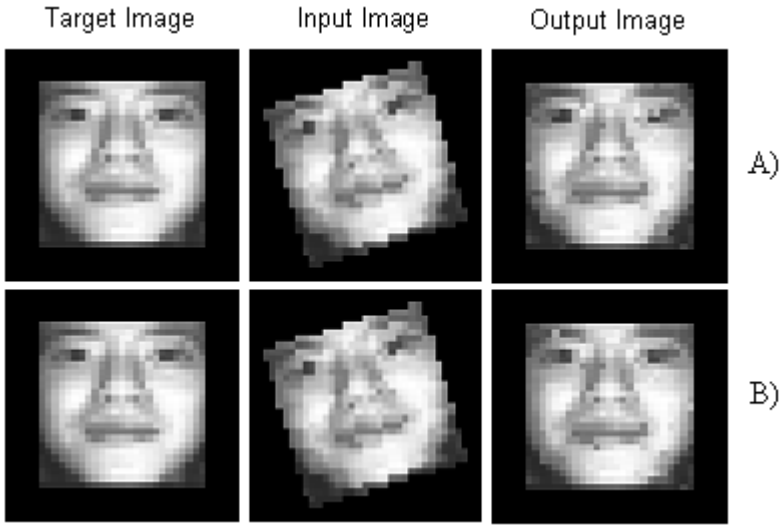


Figure 8.4: Results of grey-scale affine rotation test for case $\theta' = 16°$. A) Image results for CSRN trained via UKF. B) Image results for CSRN trained via EKF. CSRN utilizes GMLP core, movement encoding and inverse mapping.

Table 8.3 tabulates the results of the grey-scale image rotation experiment for a CSRN trained using the UKF training method. Results for the full testing set are shown. The $\theta' = 16°$ case is highlighted. In this case the network achieves a best case function accuracy of $J_{ACC} = 48.9\%$, an image correlation ration of $IM_{CR} = 98.7\%$. Table 8.4 summarizes the results of this rotation experiment for grey-scale images.

Table 8.3: Grey-scale image rotation results for CSRN trained via UKF. CSRN utilizes GMLP core, movement encoding and inverse mapping. Data for full testing set.

	Grey-scale Rotation Results (full test set)				
θ'	J_{MSE}	J_{ACC}	IM_{MSE}	IM_{ACC}	IM_{CR}
0	1.49E-01	48.9	267.1	77.8	0.984
2	1.53E-01	48.8	233.2	78.6	0.987
4	1.08E-01	52.7	248.6	76.9	0.986
6	7.65E-02	58.1	210.8	81.7	0.988
8	5.25E-02	64.4	115.7	85.2	0.994
10	2.97E-02	75.0	73.9	88.8	0.993
12	1.80E-02	82.1	124.7	90.0	0.989
14	9.80E-03	89.2	69.7	91.5	0.997
16	6.38E-03	93.0	60.8	93.4	0.997
18	1.80E-02	80.2	125.4	88.5	0.994
20	3.39E-02	66.4	149.7	83.1	0.990

Table 8.4: Summary of grey-scale image rotation results for CSRN trained via UKF. CSRN utilizes GMLP core, movement encoding and inverse mapping.

Rotation		**UKF Training**
Metric	**Units**	*CSRN w/ GMLP core*
$\boldsymbol{J_{ACC}}$	*%*	93.0
$\boldsymbol{J_{MSE}}$	-	0.01
$\mathbf{IM_{ACC}}$	*%*	93.4
$\mathbf{IM_{MSE}}$	-	60.81
$\mathbf{IM_{CR}}$	-	1.00
$\boldsymbol{T_{TR}}$**(tot)**	*sec*	3400 sec
$\boldsymbol{T_{TR}}$**(norm)**	*msec*	252 msec
$\boldsymbol{T_{R}}$	*min*	668 min

8.6.4 Comparison of UKF and EKF Training Methods

8.6.4.1 Image Results

Figures 8.1 shows the image results of performing binary rotation with a CSRN trained with both the UKF and EKF training methods. Figure 8.3 shows the same results for the grey-scale case. In these Figures, the results for the CSRN trained via these two methods are shown side-by-side in rows b) and c). Both methods can be compared to the raw transformation results in row d) or the actual target image in row e).

Table 8.5 compares the results for both methods. The date shown is that from the binary image experiment for a single test case of θ = -16 °.

Table 8.5: Comparison of binary image results for CSRN trained via UKF and EKF. CSRN utilizes GMLP core, movement encoding and inverse mapping.

Binary Image Rotation						
CSRN with GMLP core						
Metric	**Units**		*UKF*	*EKF*	*Diff.*	*% Diff.*
$\boldsymbol{J_{ACC}}$	*%*	*Ave*	51.4	82.6	-31.20	-37.8
		Best	63.6	94.2	-30.60	-32.5
$\mathbf{IM_{ACC}}$	*%*	*Ave*	96.02	99.0	-2.98	-3.0
		Best	98.7	99.1	-0.40	-0.4
$\mathbf{IM_{CR}}$	-	*Ave*	0.76	0.94	-0.18	-19.1
		Best	0.92	0.95	-0.03	-3.2

Visual inspection of Figs. 8.1 through 8.4 reveals that the image results produced by the CSRN trained via the UKF method are inferior to those produced by that trained via the EKF method. The UKF images show more distortion, particularly in the lower ranges of rotation. This is true of both binary and grey-scale results and is supported by the lower IM_{ACC} and IM_{CR} values of Table 8.5. This distortion directly correlates with the function accuracies achieved with the UKF method, which are significantly lower of the EKF method. This indicates that the CSRN trained via the UKF method does not learn the rotation transformation as effectively as that trained via the EKF method.

8.6.4.2 Computation Times

To compare the speed of the UKF training method with that of the EKF, we examine the following metrics: training time, T_{TR}, simulation run time, T_R, and total batch time,

T_B. Table 8.6 shows these metrics for both the UKF and EKF training methods. The UKF method's training time of 727 sec is 26.2% longer than that of the EKF method. The increased training time, causes corresponding increases in both the simulation time and total batch time.

Table 8.6: Comparison of timed results for CSRN trained utilizing UKF and EKF methods for binary affine rotation. CSRN utilizes GMLP core, movement encoding and inverse mapping. Times include: training time, T_{TR}, run time, T_R, total batch time, T_B. Total batch time is given for a batch size of 50 simulations.

Binary Image Rotation					
CSRN with GMLP core					
Metric	**Units**	***UKF***	***EKF***	***Diff.***	***% Diff.***
$\boldsymbol{T_{TR}}$	*sec*	726.9	576.2	150.7	26.2
$\boldsymbol{T_R}$	*min*	12.21	9.87	2.34	23.7
$\boldsymbol{T_B}$	*hour*	10.18	8.22	1.96	23.8

8.6.4.3 Convergence

Figure 8.5, shows plots of the testing MSE for the CSRN trained via the UKF and EKF methods. These plots indicate that the CSRN learns via both methods. Table 8.7 tabulates the convergence metrics for both cases. Metrics include: calculated convergence time, T_C, settling time, T_S, minimum error, E_{MIN}, and time of occurrence of minimum error, T_{ME}. Statistics are computed based on batches of 50 simulations. Both methods are stable, consistently converging with average setting times of approximately 25 and 160 epochs, respectively. The UKF method converges roughly 80% faster than

the EKF method. However, the EKF method converges to a much lower final error, resulting in the higher function accuracies discussed above.

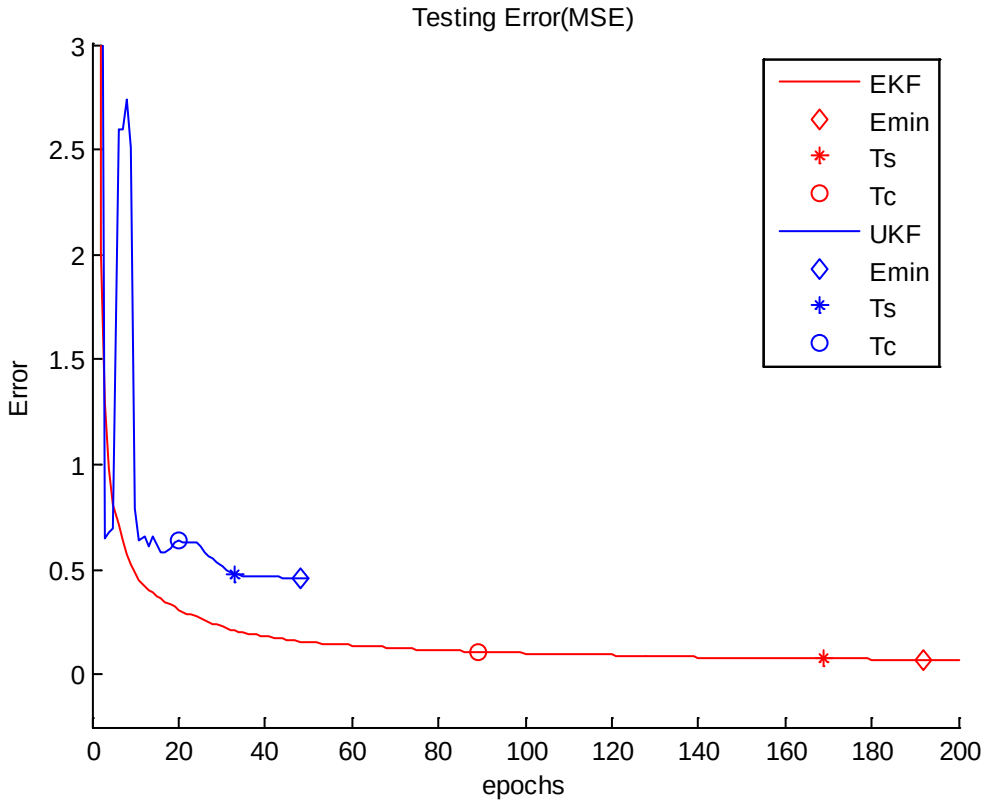


Figure 8.5: Plots of the testing MSE for the CSRN trained to perform affine rotation of binary images. CSRN utilizes GMLP core, movement encoding and inverse mapping. The blue line represents the UKF training method. The red line represents the EKF training method. Circles indicate the calculated convergence points, T_C. Asterisks indicate settling points, T_S and diamonds indicate locations of the minimum error.

Table 8.7: Comparison of convergence of mean square testing error for CSRN trained using UKF and EKF training methods. Metrics include: calculated convergence time, T_C, settling time, T_S, minimum error, E_{MIN}, and time of occurrence of minimum error, T_{ME}. Network is a CSRN with GMLP core, using movement encoding and inverse mapping.

Binary Image Rotation						
CSRN with GMLP core						
Metric	**Units**		*UKF*	*EKF*	*Diff.*	*% Diff.*
T_C	*epochs*	*Ave*	23.8	97.68	-73.88	-75.6
		Best	20.0	89	-69.00	-77.5
T_S	*epochs*	*Ave*	25.3	160.06	-134.76	-84.2
		Best	33.0	169	-136.00	-80.5
E_{MIN}	-	*Ave*	0.59	0.20	0.39	195.0
		Best	4.5E-1	7.15E-2	0.38	529.4
T_{ME}	*epochs*	*Ave*	35.6	195.2	-159.60	-81.8
		Best	48	192	-144.00	-75.0

8.6.5 Conclusions

The CSRN clearly learns via the UKF training method. The UKF method converges much more rapidly than the EKF method. However, computation of the UKF is less efficient than that of the EKF resulting in a training time 26% longer than that of the EKF method. The EKF method converges to a lower final error that the UKF method, resulting in improved function accuracy, and ultimately better approximation of the rotation transformation.

There are two distinct advantages of the UKF training method, arising from the fact that it is a derivative free optimization method. As such, it does not require computation of the Jacobian, as in the EKF method. Use of the UKF method eliminates the restriction on image size imposed by the multi-streaming EKF's need for extremely large matrices used in computation of the Jacobian, as discussed in Section 5.2.2.2. Elimination of the Jacobian eliminates the need for the BPTT computation. As discussed in Chapter 7, even minor changes to the CSRN's core network may require the equations for BPTT to be re-derived. Derivation of these equations is a non-trivial. When using the UKF method, one need only compute the equations for forward computation of the CSRN core, making modifications to and or development of new core networks much easier.

9 CONCLUSION

9.1 Summary

From its inception, it has been speculated that the CSRN would have important implications in image processing (IP). However, to date, very little work has been done to adapt CSRNs to IP tasks. In this work, we present a flexible, generalized architecture for the CSRN which is capable of performing a wide variety of IP tasks, including pixel level transformations, spatial filtering, and complex geometric transformations. These novel contributions are crucial for obtaining a generalized image processor using recurrent neural networks (NNs) such as the CSRN.

9.2 Brief Overview of Goals

The overall goal of this dissertation is to innovate the CSRN to perform IP. The specific goals of this work are repeated here:

- ***Goal 1:*** To develop and demonstrate the ability of a generalized CSRN architecture to perform a variety of image processing tasks.
- ***Goal 2:*** To demonstrate the ability of CSRNs to perform affine transformations on images.
- ***Goal 3:*** To demonstrate the ability of the CSRN to learn to register images under rigid-body transformation without prior knowledge of transformation parameters.
- ***Goal 4:*** To demonstrate training of CSRNs via parameter estimation with an unscented Kalman filter (UKF) and to evaluate the efficacy of this training method.

9.2.1 *Goal 1:* A Generalized CSRN Architecture for IP

In Chapter 4, we present a generalized CSRN architecture adapted for image processing. Also in Chapter 4, we demonstrate its efficacy in performing both pixel level

operations and spatial filtering, by implementing a grey-scale to binary transformation and a low-pass (averaging filter) respectively.

Experimental results indicate that the CSRN is able to learn and perform grey-scale to binary conversion and low-pass filtering. In these applications, the CSRN generalizes well, and performs, very similar to MATLAB's® *im2bw( )* and *imfilter( )* functions, respectively. The ability of the CSRN to perform complex geometric transformations is demonstrated by its ability to perform affine transformations and rigid-body registration in Chapters 5 and 6, as discussed above.

9.2.2 *Goal 2:* Affine Transformation of Images

In Chapter 5, we demonstrate the CSRN's ability to learn and perform basic affine transformation of both binary and grey-scale images. We utilize the standard transformation equations, (5.5), (5.6) and (5.7), to implement a baseline method for comparison. Experimental results indicate that the CSRN is cable of learning the 2-D linear functions associated with translation, rotation and scaling, and is capable of performing these transformations nearly as well as the baseline method. In addition, the CSRN performs these transformations, on average, 65% faster than the baseline method. The faster computation time for performing affine transforms makes the CSRN attractive for real-time, embedded applications.

9.2.3 *Goal 3:* Image Registration under Rigid Body Transformation

In Chapter 6, we demonstrate the CSRN's ability to learn and perform basic image registration on binary and grey-scale images subjected to rigid-body transformation. We utilize the standard rigid-body equations, (5.5), (5.6), and assumes perfect estimation of

model parameters to implement a baseline registration method for comparison. Experimental results indicate that the CSRN is capable of learning rigid body registration. However, CSRN has difficulty in generalizing over all images in the training set. Best case results for the primary test case show that the CSRN performs as well as the baseline registration method. In addition, the CSRN reduces computation time of rigid-body registration, on average, by 70% over that achieved by the baseline method.

9.2.4 *Goal 4:* Training the CSRN via UKF

In Chapter 8, we present a method of training the CSRN via parameter estimation using an UKF. This UKF method converges much more rapidly than the EKF method. However, it is computationally less efficient than the EKF method, resulting in longer training times. The UKF converges to a higher final error than the EKF method offering slightly reduced accuracies in function approximation.

9.3 Discussions of Contributions

In this section we summarize the novel contributions of this work.

9.3.1 Development of a Generalized CSRN Architecture for IP

In Chapter 4 we propose a generalized architecture for the CSRN which is capable of performing a wide range of IP tasks. In Chapters 4, 5 and 6, we demonstrate the ability of the CSRN to perform pixel operations, filtering and geometric transformations utilizing this new architecture. These novel contributions are crucial for obtaining a generalized image processor using recurrent NNs such as the CSRN. To date, this contribution is unpublished. However, we plan to submit this to an appropriate journal for publication very soon.

9.3.2 Implementation of sub-image processing for CSRNs

In Chapter 4, we adapt standard sub-image processing techniques for use with the CSRN. The application of sub-image processing is fundamental to the practical application of CSRNs to generalized IP. Prior to this work, application of CSRNs to IP tasks was limited to an image size of 7x7 pixels. Our best efforts, without the use of sub-image processing, extended this size to 15x15 pixels. The introduction of sub-image processing breaks this size barrier making the application of CSRNs to IP tasks tractable. While we have shown examples of image sizes of 125x125 pixels in this work, image size is limited only by the amount of run-time available for simulation and not by the CSRN. Therefore, high performance computation can facilitate processing of larger images. In addition, the sub-image structure utilized in this method is directly transferable to hardware implementations. Making real-time or near-real-time, embedded application a real possibility. This contribution has been published in a few major conferences [3][43][75], and will ultimately be submitted to an appropriate journal as part of a larger work.

9.3.3 Application of CSRNs to Geometric Transformations

A fundamental challenge to the theory underlying NNs, posed by Rosenblatt in his early work on perceptrons [80], is the recognition of topological relations. Feed-forward networks are incapable of solving this class of problems due to their exponential complexity. Geometric transformations, which include affine transformation and image registration (IR), fall within this class of IP problems.

In Chapter 5, we demonstrate the ability of the CSRN to perform affine transformations, specifically those of translation, rotation and scaling. The CNN is

capable of performing single-pixel and fractional translation, and through repeated application, can perform translation and rotation. However to date, no other NN has been shown capable of performing all three of these affine transformations. The results of Chapter 5 have been accepted for publication to the International Joint Conference on Neural Networks (IJCNN)[2].

In general, IR involves the following four steps: 1) feature detection, 2) feature matching, 3) transform model parameter estimation, and 4) transformation and re-sampling. NNs have been applied to IR. However, most of these applications perform only one of the four IR steps, and none have been applied directly to the transformation step for IR. In Chapter 6, we demonstrate the CSRN's ability to perform rigid body registration, and it's potential for performing all four steps of the image registration process. Preliminary results of Chapter 6 have been published in a few major conferences [3][5][43][75] and a technical report to the National Science Foundation [44]. Ultimately this contribution will be submitted to an appropriate journal as part of a larger work.

9.3.4 Development of a Multi-layered Elman SRN

In Chapter 7, we present an extension of an Elman simultaneous recurrent network (ESRN) as a new type of network core for the CSRN. This extension adds feedback from the ESRN's output layer, providing feedback, for recurrency, from both the hidden and output layers of the network, thus the moniker ESRN with multi-layered feedback (ESRNmlf). This network was developed to provide improved stability and convergence over that of the GMLP, demonstrated in the maze traversal application. Since these convergence issues are not evident in the IP applications tested herein, the ESRNmlf's

ability to provide improved convergence remains untested to date. The ESRNmlf network and its preliminary application to image registration has been published in a few major conferences [3][5].

9.3.5 Implementation of a UKF Training Method for CSRNs

In Chapter 8, we present a method of training the CSRN via parameter estimation using an UKF. This represents the first time that the UKF has been applied to the training of CSRNs. This method is a derivative free training method and, as such, does not require computation of the Jacobian, as is required in the EKF method. This eliminates the restriction on image size imposed by the multi-streaming EKF's need for extremely large matrices used in computation of the Jacobian. Likewise, elimination of the Jacobian eliminates the BPTT computation, allowing users to compute only the equations for the forward computation of the network. This simplification makes modification and/or development of new core networks for the CSRN much easier and more tractable. To date, this contribution is unpublished. However, we plan to submit this to an appropriate conference for publication very soon and ultimately to an appropriate journal as part of a larger work.

9.3.6 Development of a Gentle Engineering Tutorial for BPTT

In Appendix A, we develop a gentle engineering tutorial for the application of BPTT to the training of complex NNs such as the CSRN. This tutorial includes mathematical derivation of the BPTT method and basic algorithms. To date, this contribution is unpublished. However, we plan to submit a review article for journal publication soon.

9.4 Future Works

In this section, we discuss a few interesting directions in which this work may proceed in the future.

9.4.1 Pattern Recognition Using the CSRN

Now that we have developed a generalized CSRN architecture for implementing generic IP functions, a wealth of complex IP tasks can be implemented using CSRNs. Of particular interest is feature extraction and object detection/recognition. Topics of interest may include histograms, edge detection, and Fourier transformation. More complicated topics may include principle component analysis (PCA) and scale invariant feature transforms (SIFT). Object detection and recognition tasks of interest may include direct detection of facial features, and object detection for navigation of autonomous robots.

9.4.2 Improved IR Using the CSRN

We propose that the limitations encountered in the application of CSRNs to rigid-body IR, can be overcome by the two-stage registration method, depicted in Fig. 9.1

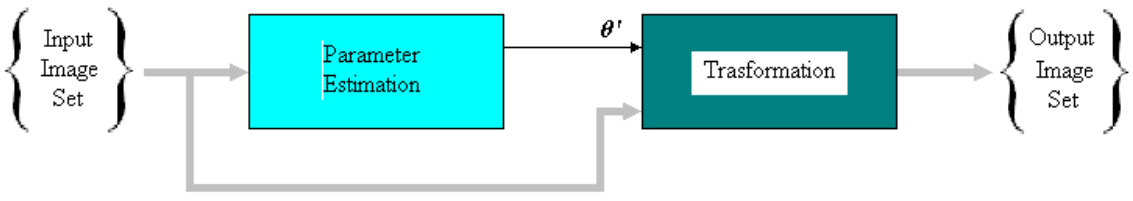


Figure 9.1: A two-staged registration method for improved rigid body image registration. θ' represents the parameters of the rigid body model.

Both stages of this method would be implemented using CSRNs. The second, or transformation stage, has already been implemented in Chapter 5. The architecture for the application of the CSRN to the parameter estimation stage is shown in Fig. 9.2.

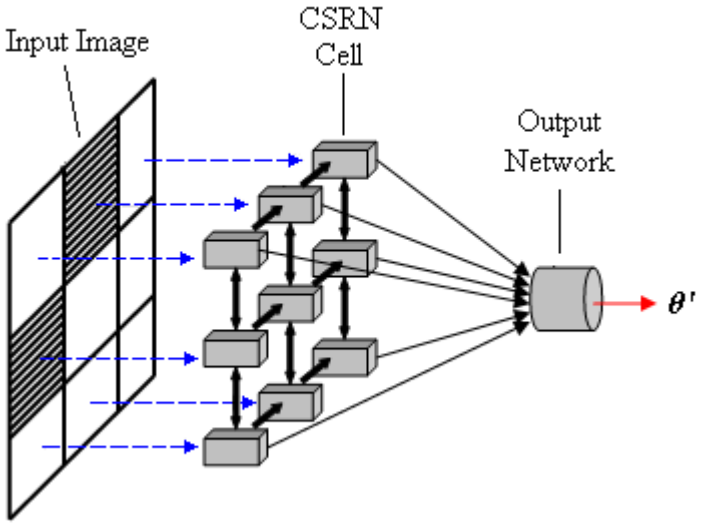


Figure 9.2: CSRN architecture for parameter estimation step of rigid body IR. θ' represents the parameters of the rigid body model.

9.4.3 Application of ESRNmlf to Maze Traversal Problem

The ESRNmlf was developed to provide improved stability and convergence over that of the GMLP, demonstrated in the maze traversal application. Since these convergence issues are not evident in the IP applications tested herein, the ESRNmlf's ability to provide improved convergence remains untested to. In order to test this hypothesis, we propose implementation of the maze traversal application with a CSRN utilizing the ESRNmlf core network.

9.4.4 Distributed Control of Multi-joint Robots

As in any robotic application, actuation of one joint causes disturbances in the control of the remaining joints. This problem is compounded by robots with a high degree of freedom due to a large number of joints, such as a serpentine robot. Coordination and control of these types of robots can be extremely difficult. SRNs have been successfully applied to control systems [97]. The CSRN has the added advantage that its cellular structure can be mapped to the geometric structure of the robot, utilizing one CSRN to control each robot joint for autonomous navigation.

REFERENCES

[1] M. Aein, H. Talebi. "Introducing a training methodology for cellular neural networks solving partial differential equations." Neural Networks, 2009, IEEE International Joint Conference on, 2009.

[2] K. Anderson, K. Iftekharuddin, “Learning topological image transforms using cellular simultaneous recurrent networks,” 2013 IEEE International Joint Conference on Neural Networks, accepted for publication.

[3] K. Anderson, K. Iftekharuddin, P. Kim, “Gray Scale Image Registration using Cellular Simultaneous Recurrent Networks,” Proc. of 54th SPIE Annual Meeting, Vol. 7442, pp: 74420 C1-C12, San Diego, CA. 2009

[4] K. Anderson, K. Iftekharuddin, B. Montgomery, “Single Camera Based Object Detection and Tracking for Mobile Robots, “Proceedings of the SPIE, Volume 7072, pp. 70720T-70720T-12. 2008.

[5] K. Anderson, K. Iftekharuddin, E. White, P. Kim, “Binary Image Registration Using Cellular Simultaneous Networks”. Conference proceedings *IEEE CIMSVP 2009*.

[6] C.M. Bishop, *Neural Networks for Pattern Recognition*, Cambridge, UK: Oxford University Press, 1995.

[7] L. Brown. "A survey of Image Registration Techniques", ACM Computing Surveys, Vol. 24, Issue 4, pp.325 - 376, 1992.

[8] G. Burdet, P. Combe, and O. Parodi, Eds., “Algorithms for the detection of connectedness and their neural implementation,” in Neuronal Information Processing: From Biological Data to Modelling and Applications, Series in Mathematical Biology and Medicine, P. R. Roelfsema, S. Bohte, and H. Spekreijse, Eds. Singapore: World Scientific,1999, vol. 7.

[9] H. Chang, W. Freeman, “Parameter optimization in models of the olfactory neural system,” *Neural Networks*, Vol.9, No.1, pp 1-14, 1996.

[10] L. Chmielewski, D. Kozinska. "Image registration." Proceedings of the 3rd Polish Conference on Computer Pattern Recognition Systems, MiBków (Poland), 2003.

[11] L. Chua, T. Roska, *Cellular Neural Networks and Visual Computing: Foundations and Application*, Cambridge University Press, 2004.

[12] L. Chua, T. Roska, “The CNN Paradigm,” IEEE Transactions on Circuits and Systems, Vol.40, No.3. 1993.

[13] L. Chua, L. Yang, “Cellular Neural Networks: Applications,” IEEE Transactions on Circuits and Systems, Vol.35, No.10. 1988.

[14] L. Chua, L. Yang, “Cellular Neural Networks: Theory,” IEEE Transactions on Circuits and Systems, Vol.35, No.10. 1988.

[15] R. Cleaver, G. Venayagamoorthy, “Learning functions generated by randomly initialized MLPs and SRNs”, IEEE Symposium on Computational Intelligence in Control and Automation, 2009. CICA 2009.

[16] R. Cleaver, G. Venayagamoorthy, “Learning nonlinear functions with MLPs and SRNs”, IEEE International Joint Conference on Neural Networks 2009.

[17] G. Costantini, D. Casali, R. Perfetti, “Cellular neural network template for rotation of grey-scale images,” Electron. Letters, Vol. 39, Issue 25, pp.1803–1805, 2003.

[18] T. Das, G. Venayagamoorthy, “Bio-inspired algorithms for the design of multiple optimal power system stabilizers: SPPSO and BFA,” IEEE Industrial Applications Conference, vol. 2, Oct. 2006, pp. 635-641.
[19] M. Deshmukh, U. Bhosle, “A survey of image registration,” International Journal of Image Processing, Vol. 5, Iss. 3, 2011.
[20] I. Elhanany, *et al*. "Robust image registration based on feed-forward neural networks," IEEE International Conference on Systems, Man, and Cybernetics. Vol. 2. IEEE, 2000.
[21] J. Elman, “Finding structure in time”. Cognitive Science, Vol.14, No.2, pp.179-211, 1990.
[22] P. Elsen, E. Pol, M. Viergever. “Medical image matching – a review with classification,” IEEE Engr. In Medicine and. Biology, 12(1):26–39. 1993.
[23] L. Feldkamp, D. Prokhorov, C. Eagen, and F. Yuan, Nonlinear Modeling: Advanced Black-Box Techniques, Ch. Enhanced multi-stream Kalman filter training for recurrent networks, pp.29–53, 1998.
[24] R. Fisher, S. Perkins, A. Walker and E. Wolfart. “Affine Transformation.” Hypermedia Image Processing Reference 2. http://homepages.inf.ed.ac.uk/rbf/HIPR2/affine.htm. 2003.
[25] L. Fonseca, B. Manjunath, “Registration techniques for multisensor remotely sensed imagery,” Photogrammetric Engineering and Remote Sensing, Vol. 62, pp. 1049–1056, 1996.
[26] Q. Gao, P. Messmer, G. Moschytz, “Binary Image Rotation Using Cellular Neural Networks,” IEEE International Symposium on Circuits and Systems, Vol.3, pp 113-116, 2002.
[27] S. Georghiades, P. Belhumeur, and D. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," Pattern Analysis and Machine Intelligence, IEEE Transactions on 23.6, pp. 643-660, 2001.
[28] B.K. Ghaffary, A.A. Sawchuk, A survey of new techniques for image registration and mapping, Proceedings of the SPIE: Applications of Digital Image Processing 432 pp. 222–239, 1983.
[29] A. Gholipour, N. Kehtarnavas, R. Briggs, M. Devous, and K. Gopinath, “Brain Functional Localization: A Survey of Image Registration Techniques,” IEEE Transactions on Medical Imaging, Vol.26, No. 4, 2007.
[30] B. Glocker, *et al*., “Deformable Medical Image Registration: Setting the State of the Art with Discrete Methods,” Annual Review of Biomedical Engr. Vol. 13, pp. 219-244. Annual Reviews. 2011.
[31] R. Gonzalez, R. Woods, *Digital Image Processing*, Reading, Massachusetts, Addison-Wesley, 1992.
[32] R. Gonzalez, R. Woods, *Digital Image Processing*, *3rd ed*., Pearson/Prentice Hall, 2008.
[33] R. Gonzalez, R. Woods, S. Eddins, *Digital Image Processing Using MATLAB®*. Upper Saddle River, New Jersey, Pearson Prentice Hall, 2004.
[34] L. Grant, G. Venayagamoorthy, “Voltage Prediction Using a Cellular Network”, IEEE PES General Meeting, 2010.

[35] V. Gudise, G. Venayagamoorthy, “Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks,” IEEE Swarm Intelligence Symposium, April 2003, pp. 110-117.
[36] E. Gulch, “Results of test on image matching of ISPRS WG, ISPRS”, Journal of Photogrammetry and Remote Sensing. Vol. 46, pp. 1-18, 1991.
[37] S. Haykin, *Kalman Filtering and Neural Networks*, Wiley, 2001.
[38] S. Haykin, *Neural Networks: A Comprehensive Foundation, 2nd ed.*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1999.
[39] M.H. Hedjazi, P. Abolmaesumi. "Point-Based Rigid-Body Registration Using an Unscented Kalman Filter," IEEE Transactions on Medical Imaging, Vol.26, Issue 12, 2007, pp. 1708 - 1728.
[40] D. L. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, “Medical image registration,” Phys. Med. Biol., vol. 46, pp. R1–R45, 2001.
[41] C. He, F. Lang, H. Li. "Medical image registration using cascaded pulse coupled neural networks," Information Technology Journal 10.9, pp.1733-1739, 2011.
[42] M. Holden, “A Review of Geometric Transformations for Nonrigid Body Registration,” IEEE Transactions on Medical Imaging, Vol.27, No. 1, 2008.
[43] K. Iftekharuddin, K. Anderson, “Image Registration Under Affine Transformation Using Cellular Simultaneous Recurrent Networks,” SPIE Optics and Photonics for Information Processing IV. 2010 Proceedings.
[44] K.Iftekharuddin, K.Anderson, Y.Ren. E.White, National Science Foundation SGER Year 2 Report, Grant No. ECCS-0738519 and ECCS-0715116, 2009.
[45] R. Ilin, ‘Learning and Parameterization of Recurrent Neural Network Arrays for Brain Models and Practical Applications’, PhD dissertation, Dept. of Computer Science, University of Memphis, August 2008.
[46] R. Ilin, R. Kozma, P. Werbos, “Beyond Feed-forward Models Trained by Backpropagation: A Practical Training Tool for a More Efficient Universal Approximator”, IEEE Transactions on Neural Networks, Vol.19, No.6. 2008.
[47] R. Ilin, R. Kozma, P. Werbos, “Cellular SRN Trained by Extended Kalman Filter Shows Promise for ADP,” 2006 International Joint Conference on Neural Networks. Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada. July, 2006.
[48] R. Ilin, R. Kozma, and P. Werbos. “Efficient Learning in Cellular Simultaneous Recurrent Neural Networks – The Case of Maze Navigation Problem.” Proceedings of IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning, ADPRL 2007.
[49] A. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, pp.321. 1986.
[50] S. Julier, J. Uhlmann, “A new extension of the Kalman filter to nonlinear systems,” in Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defence Sensing, Simulation and Controls, 1997.
[51] S. Julier, J. Uhlmann, H. Durrant-Whyte, “A new approach for filtering nonlinear systems,” in Proceedings of the American Control Conference, 1995, pp. 1628–1632.
[52] B. Kuo, *Automatic Control Systems*, *5th ed.*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc. pp-161, 1997.
[53] J. Kennedy, R. Eberhart, “Particle swarm optimization,” in Proc. IEEE Intl’ Conference on Neural Networks, vol. 4, December 1995, pp. 1942-1948.

[54] A. Klein, et. al, “Evaluation of 14 nonlinear deformation algorithms applied to human brain MRI registration,” NeuroImage, Vol.46, Elsevier, 2009.
[55] R. Luo, M. Kay, "A tutorial on multisensor integration and fusion," Industrial Electronics Society, 1990, IECON'90, 16th Annual Conference of IEEE, 1990.
[56] J. Maintz, M. Viergever, “A survey of medical image registration,” Medical Image Analysis, Vol.2, Num.1, pp. 1-36. Oxford University Press. 1998.
[57] C. von der Malsburg, W. Schneider. “A neural cocktail-party processor,” Biological *Cybernetics*, Vol.54, pp 29-40, 1986.
[58] W.S. McCulloch, W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” Mathematical Biophysics, vol.5, no.115-133, pp.18–27, 1943.
[59] R. van der Merwe, *ReBEL: Recursive Bayesian Estimation Library, A MATLAB toolkit*. Copyright 2006, Oregon Health and Science University.
[60] R. van der Merwe, E. Wan, S. Julier, "Sigma-Point Kalman Filters for Nonlinear Estimation and Sensor-Fusion: Applications to Integrated Navigation," Proceedings of the AIAA Guidance, Navigation & Control Conference, 2004, pp. 16-19.
[61] M. Minsky and S. Papert, *Perceptrons*, Cambridge, MA: MIT Press, 1969.
[62] M. Minsky, S. Papert, *Perceptrons - Expanded Edition: An Introduction to Computational Geometry*, MIT Pres., 1987.
[63] J. le Moigne, First evaluation of automatic image registration methods, Proceedings of the International Geoscience and Remote Sensing Symposium, pp. 315–317, 1998.
[64] M.G. Mostofa, A. Farag, E. Essock, “Multimodality Image Registration and Fusion Using Neural Network,” Proceedings of the IEEE International Conference on Information Fusion. 2000
[65] F. Oliveira, J. Tavares, “Medical image registration: a review,” Computer Methods in Biomechanics and Biomedical Engineering, pp. 1-21, Taylor and Francis. 2012.
[66] G. Pajares, M. Jesus, "A wavelet-based image fusion tutorial," Pattern Recognition, 37.9, pp.1855-1872, 2004.
[67] J.A. Parker, R.V. Kenyon, D.E. Troxel, Comparison of interpolating methods for image re-sampling, IEEE Transactions on Medical Imaging 2, pp. 31–39, 1983.
[68] X. Pang, P. Werbos, ‘Generalized Maze Navigation SRN Critics Solve What Feedforward or Hebbian Nets Cannot’, IEEE Conference on Systems, Man, and Cybernetics, Vol.3, 1996.
[69] X. Pang, P. Werbos, “Neural Network Design for J Function Approximation in Dynamic Programming,” arXiv:adap-org/9806001v1, June. 1998.
[70] X. Pang, P. Werbos, “New type of neural network learns to navigate any maze,” IEEE conference proceedings of Systems, Man and Cybernetics, Beijing, 1996.
[71] P. Phillips and P. Grother. “Face Recognition Vendor Test 2002: Evaluation Report,” www.frvt.org/DLs/FRVT_2002_Evaluation_Report.pdf.
[72] G. V. Puskorius and L. A. Feldkamp, “Decoupled extended Kalman filter training of feed-forward layered networks,” in Proc. Int. Joint Conf. on Neural Networks, pp. 771-777, 1991.
[73] L. Ramirez, N. Durdle, V. Raso, “Medical image registration in computational intelligence framework: a review,” Electrical and Computer Engineering, 2003. IEEE Canadian Conference on. 2003.

[74] Y. Ren, 'Pose Invariant Face Recognition Using Cellular Simultaneous Recurrent Networks' MS thesis. Dept. of Electrical and Computer Engr. University of Memphis, July, 2009.
[75] Y. Ren, K. Anderson, K. Iftekharuddin, "Pose Invariant Face Recognition Using Cellular Simultaneous Recurrent Networks," IEEE IJCNN 2009.
[76] Y. Ren, K. Iftekharuddin, and W. White, "Large-scale pose-invariant face recognition using cellular simultaneous recurrent network", Applied Optics, vol. 49, no. 10, pp. B92-B103, 2010.
[77] K. Rice, T. Taha, K. Iftekharuddin, K. Anderson, and T. Salan, "GPGPU acceleration of cellular simultaneous recurrent networks adapted for maze traversals," Int. Joint Conf. on Neural Networks, 2011.
[78] K. Rice, T. Taha, K. Iftekharuddin, K. Anderson, T. Salan, R. Linderman, "GPGPU Cluster Acceleration of Cellular Simultaneous Recurrent Network Based Face Recognition," SC11, November 12–18, 2011, Seattle, WA, USA.
[79] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," Psychological Review, vol.65, no.6, pp.386–408, 1958.
[80] F. Rosenblatt, *Principles of Neural Dynamics*, New York, Spartan, 1962.
[81] D. Rumelhart, G. Hinton, and R. Williams, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, in *Learning Internal Representations by Error Propagation*, pp.318–362, Cambridge, MA: The MIT Press, 1986.
[82] C. Sanderson, B.C. Lovell. Multi-Region Probabilistic Histograms for Robust and Scalable Identity Inference. Lecture Notes in Computer Science (LNCS), Vol. 5558, pp. 199-208, 2009.
[83] L. Shang, J. Cheng Lv, Z. Yi, "Rigid medical image registration using PCA neural network", Neurocomputing 69, no.13, 1717–1722, 2006.
[84] R. Szeliski, "Image alignment and stitching: A tutorial," Foundations and Trends® in Computer Graphics and Vision 2.1, pp. 1-104, 2006.
[85] K. Takizawa *et al*. "Lossless image coding by cellular neural networks with backward error propagation learning." Neural Networks (IJCNN), The 2012 International Joint Conference on. IEEE, 2012.
[86] G.A. Terejanu, "Unscented Kalman Filter Tutorial," Workshop on Large-Scale Quantification of Uncertainty, Sandia National Laboratories. 2009.
[87] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics,* The MIT Press. 2005.
[88] E. Trucco, A. Verri, *Introductory Techniques for 3-D Computer Vision.* Upper Saddle River, NJ: Prentice Hall, 1998.
[89] P. Varshney, B. Kumar, M. Xu, A. Drozd, I. Kasperovich, Image Registration: A Tutorial, in Advances and Challenges in Multisensor data and Information Processing, E. Lefebvre, Ed., IOS Press, 2007.
[90] E. Wan, R. van der Merwe, "The Unscented Kalman Filter," in S.Haykin, ed, *Kalman Filtering and Neural Networks*, New York, NY, John Wiley & Sons, Inc., pp. 221-280, 2001.
[91] E. Wan and R. van der Merwe, "The unscented Kalman filter for nonlinear estimation," in Proceedings of Symposium 2000 on Adaptive Systems for Signal Processing, Communication and Control (AS-SPCC), IEEE, Lake Louise, Alberta, Canada, October 2000.

[92] P. Werbos, “Backpropagation Through Time: What It Does and How to Do It,” Proceedings of the IEEE, vol. 78, no 10, Oct. 1990.
[93] P. Werbos, “The Brain as a Neurocontroller: New Hypotheses and New Experimental Possibilities,” In K. Pribram, ed, *Origins: Brain and Self-Organization*, Hillsdale NJ: Erlbaum, 1994, p.680-706.
[94] P. Werbos, “Learning in the brain: An Engineering Interpretation,” In K. Pribram, ed., *Learning as Self-Organization*, Erlbaum, 1996.
[95] P. Werbos, The roots of backpropagation: from ordered derivatives to neural networks and political forecasting, Adaptive and Learning Systems for Signal Processing, Communications, and Control, John Wiley and Son, Inc., 1994. Contains author’s 1974 PhD dissertation.
[96] P. Werbos, “Supervised learning: can it escape its local minimum,” WCNN93 Proceedings, Erlbaum, 1993. Reprinted in V. Roychowdhury *et al.*(eds.), *Theoretical Advances in Neural Computation and Learning*, Kluwer, 1994.
[97] D. White, D. Sofge(eds), *Handbook of Intelligent Control: Neural, Adaptive and Fuzzy Approaches*, Van Nostrand., 1992.
[98] D. Wuncsh, ‘The Cellular Simultaneous Recurrent Network Adaptive Critic Design for the Generalized Maze Problem Has a Simple Closed-Form Solution’, IJCNN 2000.
[99] R. Woods, "Spatial tranformations models," in *Handbook of Medical Imaging: Processing and Analysis*, I. Bankman, Ed. New York Academic, 2000, pp. 465-497.
[100] M. Wyawahare, P. Patil, H. Abhyankar, “Image Registration Techniques: An Overview,” International Journal of Signal Processing, Image Processing and Pattern Recognition, Vol. 2, No. 3, 2009.
[101] T. Yang, *Handbook of CNN Image Processing: All You Need to Know about Cellular Neural Networks*, Yang’s Scientific Research Institute, LLC. YangSky.com Monographs in Information Sciences, 2002.
[102] B. Zitova, J. Flusser, “Image registration methods: a survey,” Image and Vision Computing, 21, pp. 977-1000. Elsevier. 2003.
[103] R. Bellman, *Dynamic Programming*, Princeton University Press,Princeton, New Jersey, 1957.
[104] Google Scholar search, “image processing cellular neural networks”. Search performed May, 2013.
[105] Google Scholar search, “image processing”. Search performed May, 2013.
[106] Google Scholar search, “image registration”. Search performed December, 2013.

APPENDIX A: BACK-PROPOGATION THROUGH TIME: A TUTORIAL FOR IT'S APPLICATION TO SUPERVISED LEARNING IN CELLULAR SIMULTANEOUS RECURRENT NETWORKS

A.1 Introduction

The original Cellular Simultaneous Recurrent Network (CSRN) developed by Pang *et al.* [69] was trained using an extension of basic back-propagation known as back-propagation through time (BPTT) [92]. This method of training is fundamental to CSRNs. Prior to BPTT, no tractable method of computing the required derivatives existed.

In this tutorial we present a gentle approach to the application of BPTT to supervised learning in CSRNs. After reviewing the concept of supervised learning, we introduced the basic mathematical concepts of the ordered system, the ordered derivative, and the chain rule for ordered derivatives. Next, we derive the basic back-propagation (BP) equations for a simple feed-forward network known as a generalized multi-layered perceptron (GMLP) and provide a basic algorithm. Finally, we extend the basic BP concept to handle networks with recurrency by demonstrating the use of BPTT for a CSRN using a GMLP core.

A.1.1 Supervised Learning

The goal of any training method that utilizes supervised learning is to minimize some error function over a given training set. The sum-squared error has historically been used in training of CSRNs.

For each set of training data, we apply the inputs, X[k], to the network, calculate the network's output, Y[k], then obtain the error by subtracting the known target T[k] from the network's output, Y[k]. The sum-squared error over the entire training set is given by

$$E = \frac{1}{2}\sum_{k=1}^{N}\sum_{i=1}^{q}(Y_i[k] - T_i[k])^2, \qquad (1)$$

where q is the number of outputs and N is the number of data points in the training set.

The classic approached to training of NNs via supervised learning is to calculate $\frac{\partial E}{\partial w}$ so that we can adjust the weights via the following learning rule

$$w_{ij}^{new} = w_{ij}^{old} - \gamma \cdot \frac{\partial E}{\partial w_{ij}}, \qquad (2)$$

where γ is the learning rate [6].

A.1.2 The Ordered Derivative

To introduce the concept of the ordered derivative, consider the simple ordered calculation shown in Fig. A.1. By "ordered" we mean that each variable is calculated in order of its occurrence, that is, z_1, z_2 then z_3.

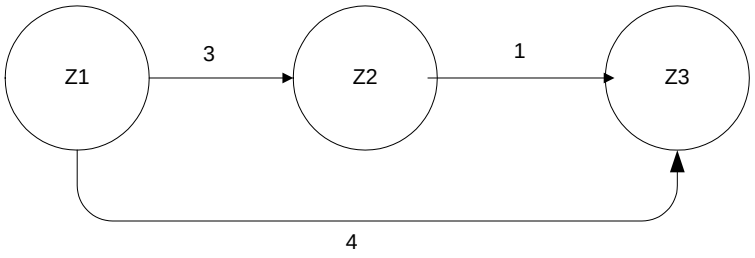


Figure A.1: Simple Ordered System where each node represents a computation. Calculations proceed in sequential order.

We can easily calculate equations for z_2 and z_3 as

$$z_2 = 3z_1 \tag{3}$$

and

$$z_3 = z_2 + 4z_1. \tag{4}$$

Now if we take the direct partial derivatives of z_3 we get

$$\frac{\partial z_3}{\partial z_1} = 4, \frac{\partial z_3}{\partial z_2} = 1. \tag{5}$$

However, since we are making ordered calculations, z_1 and z_2 are known prior to computation of z_3. If we first substitute (3) into (4), then take the partial derivative, we get

$$\frac{\partial z_3}{\partial z_1} = \frac{\partial}{\partial z_1}\left(3z_1 + 4z_1\right) = 7, \tag{6}$$

which differs from the result obtained using the direct partial.

In his PhD. Dissertation [95], Werbos proves that in an ordered system the ordered derivative is the true derivative.

A1.3 Chain Rule for Ordered Derivatives

Since the chain rule is used in the derivation of standard backpropagation equations for neural networks (NNs), let us examine the chain rule for ordered derivatives given by (7).

$$\frac{\partial^+ z_n}{\partial z_i} = \sum_{j=i+1}^{n} \frac{\partial^+ z_n}{\partial z_j} \cdot \frac{\partial z_j}{\partial z_i} \tag{7}$$

We now apply this equation to our ordered system from Fig. A.1, which yields

$$\frac{\partial^+ z_3}{\partial z_1} = \sum_{j=2}^{3} \frac{\partial^+ z_n}{\partial z_j} \cdot \frac{\partial z_j}{\partial z_i}, \tag{8}$$

$$\frac{\partial^+ z_3}{\partial z_1} = \frac{\partial^+ z_3}{\partial z_2} \cdot \frac{\partial z_2}{\partial z_1} + \frac{\partial^+ z_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial z_1},$$

$$\frac{\partial^+ z_3}{\partial z_1} = 1 \cdot 3 + 1 \cdot 4 = 3 + 4 = 7.$$

As can be seen, this result agrees with that obtain in (6).

A.2 Basic Backpropagation using the Chain Rule for Ordered Derivatives

In basic BP, the system error is back-propagated through the system to compute the partial derivative of the error with respect to the weights. The equations for

backpropagation are directly tied to the network architecture. Any change in architecture, for example, adding inputs/outputs, changing an activation function, or adding a layer, results in a change to the backpropagation equations. As a result, we must first specify the network architecture before we can demonstrate how to compute the backpropagation equations.

A.2.1 Backpropagation for the GMLP

In his body of work on CSRNs, Werbos has consistently used the generalized multi-layered perceptron (GMLP) as the CSRN's core network. I questioned Dr. Werbos on this selection. His answer was multi-fold. First, he reminded me that the standard MLP's, with which so many of us begin our study of NN's, weren't around when he developed backpropagation and the GMLP. Second, since the GMLP has only 1 layer, he did not have to specify the number of layers. Third, the one-layered architecture had a distinct coding advantage in that the backpropagation calculation could be done in a single backwards loop.

Since the GMLP was developed in conjunction with the ordered derivative, and since it has been used, almost exclusively, to implement CSRN's we will begin by examining basic BP for the GMLP.

A.2.1.1 The GMLP

Figure A.2 depicts a basic GMLP network. Note that in this figure, the network is configured as a feed-forward network.

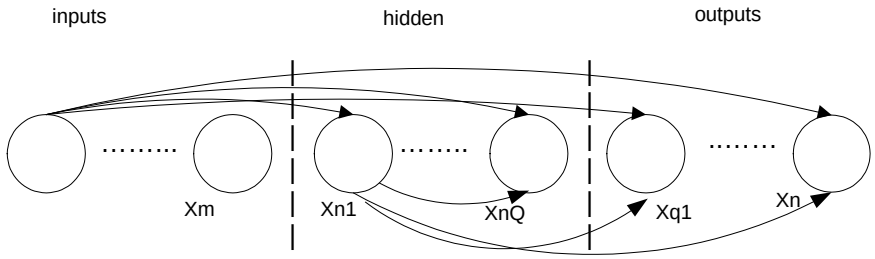


Figure A.2: The Generalized Multi-Layered Perceptron

The neurons in the network, also referred to as nodes, are numbered sequentially. In this case, there are m, input nodes, h, hidden nodes and q, output nodes. The total # of nodes is N = m + h + q. The output of each node is given by x_i. The nodes are arranged in 3 groups, input, hidden, and output nodes. Note that only the hidden and output nodes contain active neurons. The input vector, X, is copied directly to the corresponding input nodes, x_1-x_m. The outputs of the output nodes, x_{m+h+1}-$x_{N,}$ are copied directly to the output vector Y.

To simplify coding, it is convenient to specify two additional variables: h1 = m+1, which indicates the index to the first hidden node and q1 = m + h + 1 which indicates the index to the first output node.

The most significant detail about the GMLP is that the input to any active node is given by the sum of the outputs from all the preceding nodes.

The weights of the network are specified by w_{ij}, which represents the weight connecting the j^{th} node to the i^{th} node.

An algorithm for the forward computation of the GMLP network is shown below.

Algorithm A.1: Forward Computation for the GMLP network

1) Copy the inputs from the input vector X to input nodes x_1-x_m. (*Input nodes are not active neurons and their inputs are passed thru to their outputs.*)

2) Loop thru active nodes computing the summation of the inputs to each node and the nodes output.

3) Copy the output of the output nodes to the output vector Y.

A.2.1.2 Derivation of Backpropagation Equations for the GMLP

Recall that our goal is to calculate $\frac{\partial E}{\partial w}$, so that we can adjust the weights via the learning rule given in (2). Recall that *E* is the sum-squared error over the entire training set given by (1).

Let us begin with the chain rule for ordered derivatives, repeated below

$$F\_Z_i = \frac{\partial^+ Z_n}{\partial Z_i} = \sum_{j-i+1}^{n} \frac{\partial^+ Z_n}{\partial Z_j} \cdot \frac{\partial^+ Z_j}{\partial Z_i} \tag{9}$$

Notice that we have introduced a simpler notation developed by Pang *et al*. in [69]. If the ultimate target of the ordered derivative is Z_n, then we can consider F_Z_i as the *feedback* of the partial of Z_n with respect to Z_i.

Now if our ultimate target, Z_n represents *E*, then the chain-rule for ordered derivatives becomes

$$F\_Z_i = \frac{\partial^+ Z_n}{\partial Z_i} = \frac{\partial^+ E}{\partial Z_i} = \sum_{j>i}^{n} \frac{\partial^+ E}{\partial Z_j} \cdot \frac{\partial Z_j}{\partial Z_i}$$

$$= \frac{\partial^+ E}{\partial Z_n} \cdot \frac{\partial Z_n}{\partial Z_i} + \sum_{j>i}^{n-1} \frac{\partial^+ E}{\partial Z_j} \cdot \frac{\partial Z_j}{\partial Z_i}$$

$$F\_Z_i = \frac{\partial E}{\partial Z_i} + \sum_{j>i}^{n-1} F\_Z_j \cdot \frac{\partial Z_j}{\partial Z_i} \tag{10}$$

We can now use eq. 10 to work backward to find the feedback to each point in the GMLP network. Let's start with the output, Y.

$$F\_Y_i = \frac{\partial E}{\partial Y} + \sum_{j>i} F\_Z_j \cdot \frac{\partial Z_j}{\partial Z_i} \tag{11}$$

Since there are no neuron's after the output, Y_i, the summation above becomes zero and

$$F\_Y_i = \frac{\partial E}{\partial Y_i} = \frac{\partial}{\partial Y_i}\left[\frac{1}{2}(Y_i - T_i)^2\right] = Y_i - T_i, \tag{12}$$

which is simply the error between the output and the target.

Continuing to move backward through the network, we must now calculate the feedback to the output of the active neurons, given by

$$F\_x_i = \frac{\partial E}{\partial x_i} + \sum_{j-i+1}^{N} F\_Z_j \cdot \frac{\partial Z_j}{\partial Z_i}. \tag{13}$$

If $x_i = Y_i$ is an output node, then

.

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial Y_i} = F\_Y_i.$$

Figure A.3 shows an exploded view of neuron *j*. As can be seen, the output from node x_i is directly connected to the summation node, net_j. In this case $Z_j = net_j$ and $F\_Z_j = F\_net_j$ and eq. 13 becomes

$$F\_x_i = F\_Y_i + \sum_{j-i+1}^{N} F\_net_j . \frac{\partial net_j}{\partial x_i} \tag{14}$$

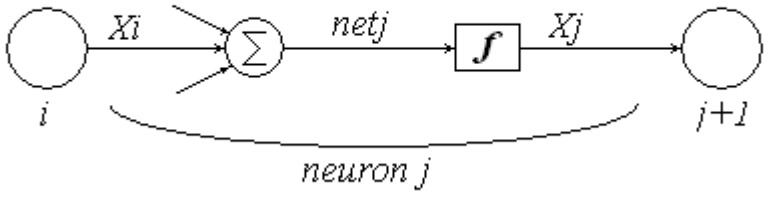


Figure A.3: Exploded view of neuron *j* in the GMLP network.

Now,

$$net_j \equiv \sum_{i=1}^{j-1} x_i \cdot w_{ji} \tag{15}$$

and

$$\frac{\partial net_j}{\partial x_i} = \frac{\partial}{\partial x_i}\left[\sum_{i=1}^{j-1} x_i \cdot w_{ji}\right] = w_{ji}. \tag{16}$$

Substituting (16) into (14) yields

$$F\_x_i = F\_Y_i + \sum_{j-i+1}^{N} F\_net_j . w_{ji}. \tag{17}$$

Note that this is a general equation covering both output neurons as well as hidden neurons. In the case of hidden neurons, there is no direct connection to an output, so the F_Y_i term is zero. In the case of the final output neuron, there are no subsequent neurons and the summation term is zero. In the case of the remaining output neurons, they receive feedback from both an output and proceeding neurons, therefore both terms remain active.

As we continue backward through the network, we must next compute F_net_i. Once again we begin with the equation for the ordered derivative.

$$F\_net_i = \frac{\partial E}{\partial net_i} + \sum_{j \geq i} F\_x_j \cdot \frac{\partial x_j}{\partial net_i} \tag{18}$$

Since there is no direct connection to an output, where E is computed, the 1st term becomes zero. Also, since net_i only has a connection to x_i, there is only 1 term in the sum and (18) reduces to

$$F\_net_i = F\_x_i \cdot \frac{\partial x_i}{\partial net_i} \tag{19}$$

As can be seen from Fig.A.3,

$$x_i = f(net_i),$$

where f is the activation function of the neuron, therefore

$$\frac{\partial x_i}{\partial net_i} = \frac{\partial}{\partial net_i}\left[f(net_i)\right] = f'(net_i) \tag{20}$$

and

$$F\_net_i = F\_x_i . f'(net_i). \tag{21}$$

Continuing to move backward from net_i, we encounter the weights connected to net_i. The final term we need to compute is F_w_{ij} given by

$$F\_w_{ij} = \frac{\partial E}{\partial w_{ij}} + \sum_{j>i} F\_net_i . \frac{\partial net_j}{\partial w_{ij}}, \tag{22}$$

Once again, since there is no direct connection between w_{ij} and an output, the 1st term is zero. Now,

$$net_i \equiv \sum_{j<i} x_j . w_{ij} \tag{23}$$

and

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left[\sum_{j>i} x_j . w_{ij} \right] = x_j, \tag{24}$$

such that (22) becomes

$$F\_w_{ij} = F\_net_i . x_j \tag{25}$$

Of course we need to sum the feedback to the weights over all training sets, such that

$$F\_w_{ij} = \sum_{k=1}^{TS} F\_net_i [k] . x_j . \tag{26}$$

A.2.1.3 BP Algorithm for the GMLP

The basic steps for the backward computation (backpropagation) through the GMLP are listed in Algorithm A.2 below.

Algorithm A.2: Backward Computation for the GMLP network

1) Initialize the feedback vector, F_x, to 0 for the input and hidden nodes.

2) Compute the feedback from outputs to the output of the active nodes, F_x. (1st term of eq. 17).

3) Loop backwards thru the active nodes performing steps 4,5 & 6 for each node.

4) Finish computation of the feedback to the node outputs, F_x. (2nd term of eq. 17)

5) Compute the feedback to the summer of the node, F_net. (21)

6) Compute the feedback to the input weights coming into the node, F_w. (25)

A.3 Backpropagation through Time

BPTT is an extension of the basic backpropagation technique which allows computation of ordered derivatives for recurrent systems. Once again in order to compute the equations for BBTT we must specify a specific network. In this section we will examine the CSRN using the GMLP core.

A.3.1 CSRN with GMLP Core

Figure A.4 shows the cellular structure of the CSRN. Each cell of the CSRN consists of a GMLP core shown in Fig. A.5.

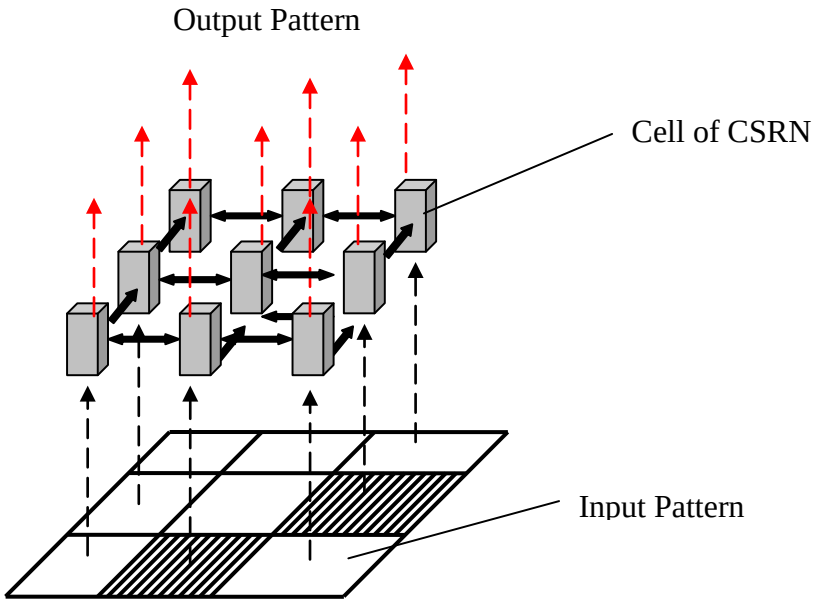


Figure A.4. Cellular structure of a CSRN.

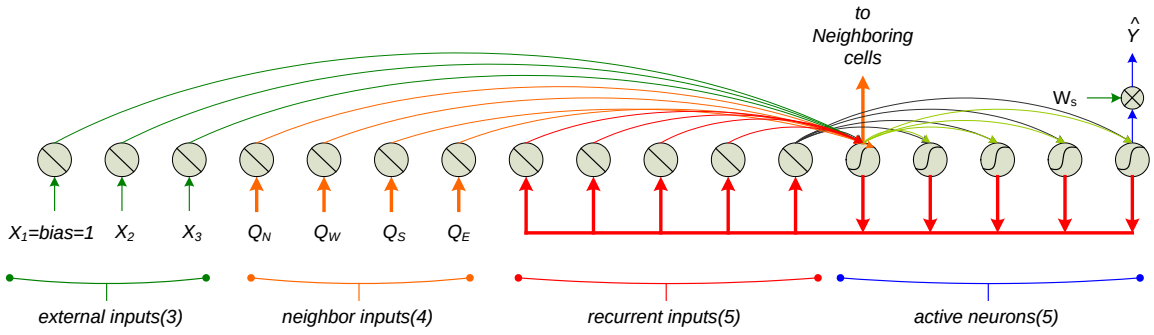


Figure A.5. GMLP core of the CSRN shown with 3 external inputs, 4 neighbor inputs, 5 recurrent inputs (12 total inputs) and 5 active nodes. Each active node receives an input from all prior nodes. Output is taken from final node, and scaled via a scaling weight. The network has a total of 17 nodes.

Recurrency in the CSRN's core is two-fold. First, the outputs from the connector nodes of the 4 neighboring cells are feed back as inputs to the current cell. These inputs are referred to as the "neighbor" inputs and give the CSRN its cellular structure. Second, the outputs of the active nodes in the current cell are feed back as inputs to the current cell. We refer to these inputs as self-recurrent inputs. This accounts for the term '*Simultaneous Recurrent'* in the CSRN name.

One identifying feature of CSRN's is their use of two computation rates. An external or input/output rate and an internal or core rate. Inputs are presented to the network, and outputs read from the network at the external rate. The faster internal rate is used to compute the core computations, and to update the recurrent inputs. This faster internal rate allows the network outputs to settle prior to being output on the external cycle. When applying BP to the CSRN not only must we back-propagate the error thru the network, we must also back-propagate it through time for each of the internal core iterations, thus, the term back-propagation through time.

Algorithms A.3a and A.3b represent the forward computation of a CSRN using a GMLP core.

Algorithm A.3a: Forward Computation for the CSRN.

1) For each core iteration perform steps 2 -7.
2) For each cell perform steps 3 & 4.
3) Compute fwd calc for cell (see alg. 3b)
4) Store outputs of cell nodes in storage array(used in BPTT algorithm)
5) For each cell perform steps 6 & & to update recurrent inputs for next iteration.
6) Copy output of each active node to its corresponding input(self-recurrent inputs)
7) Copy connector node output to inputs of 4 neighboring cells

Algorithm A.3b: Forward Computation for the GMLP core.

1) For each input node, copy input to output.
2) For each active node, initialize output to zero.
3) For each active node, perform steps 4 & 5.
4) Sum inputs to node
5) Pass input sum thru activation function

A.3.1.1 Derivation of BPTT Equations for the CSRN with a GMLP Core

As before, we will begin our analysis at the network's output. In this case there is only one output for each cell. The feedback to the output, $F\_\hat{Y}$, can be computed using (12).

$$F\_\hat{Y} = \frac{\partial E}{\partial Y} = \frac{\partial}{\partial Y}\left[\frac{1}{2}\left(\hat{Y} - T\right)^2\right] = \hat{Y} - T \tag{27}$$

The network output is taken from the final active neuron's (node 17's) output passed through a scaling weight, w_s. Next, compute the feedback to this scaling weight. We can modify (25) to achieve this goal.

$$F\_w_s = F\_w_s + F\_\hat{Y} \cdot x_{17} \tag{28}$$

Note that we maintain a running sum for F_w_s to satisfy the requirements of (26).

Continuing back through the network, we must next compute the feedback to the output of the last active node (node 17)

$$F\_x_{17} = F\_x_{17} + F\_\hat{Y} \cdot w_s . \tag{29}$$

Now we must compute the feedback to the summer of node 17. We begin by applying (21)

$$F\_net_{17} = F\_x_{17} . f'(net_{17}). \tag{30}$$

In this case the activation function is $f(net) = \tanh(net)$, for all active nodes.

In order to used (30), we must compute the derivative of the *tanh( )* activation function. Let

$$g(v) = \tanh(v) = \frac{1 - e^{-v}}{1 + e^{-v}}. \tag{31}$$

Taking the derivative via the quotient rule yields

$$g'(v) = \frac{\frac{d}{dv}\left(1 - e^{-v}\right) \cdot \left(1 + e^{-v}\right) - \frac{d}{dv}\left(1 + e^{-v}\right) \cdot \left(1 - e^{-v}\right)}{\left(1 + e^{-v}\right)^2} \tag{31}$$

$$= \frac{e^{-v} \cdot \left(1 + e^{-v}\right) + e^{-v} \cdot \left(1 - e^{-v}\right)}{\left(1 + e^{-v}\right)^2} \tag{32}$$

$$g'(v) = \frac{2e^{-v}}{\left(1 + e^{-v}\right)^2}. \tag{33}$$

Next we factor out a ½ and separate the numerator into two terms

$$g'(v) = \frac{1}{2} \cdot \frac{4e^{-v}}{\left(1+e^{-v}\right)^2} \tag{34}$$

$$= \frac{1}{2} \cdot \frac{(2e^{-v}) - (-2e^{-v})}{\left(1+e^{-v}\right)^2}. \tag{35}$$

Completing the square on the two terms in the numerator and reducing yields

$$= \frac{1}{2} \cdot \frac{(1+2e^{-v}+e^{-2v}) - (1-2e^{-v}+e^{-2v})}{\left(1+e^{-v}\right)^2} \tag{36}$$

$$= \frac{1}{2} \cdot \frac{\left(1+e^{-v}\right)^2 - \left(1-e^{-v}\right)^2}{\left(1+e^{-v}\right)^2} \tag{37}$$

$$= \frac{1}{2} \cdot \left[\frac{\left(1+e^{-v}\right)^2}{\left(1+e^{-v}\right)^2} - \frac{\left(1-e^{-v}\right)^2}{\left(1+e^{-v}\right)^2} \right] \tag{38}$$

$$= \frac{1}{2} \cdot \left[1 - \left(\frac{1-e^{-v}}{1+e^{-v}} \right)^2 \right] \tag{39}$$

$$g'(v) = \frac{1}{2} \cdot \left[1 - g^2(v)\right] \tag{40}$$

Now, substituting (40) into (30) we obtain

$$F\_net_{17} = F\_x_{17}.\frac{1}{2}(1 - x_{17}^2). \tag{41}$$

Now we must feedback these results to all weights connected to node 17, using (25).

$$F\_w_{17\,j} = F\_net_{17}.x_j \tag{42}$$

for all $j < 17$, i.e. the weights connected to node 17 from all previous nodes.

We note here that since the bias is included as an input node, the bias weights are computed as part of the network weights and we do not need to make a separate calculation for them here.

Now that we have finished the required calculations for the last active node, we loop backward through the remaining active nodes (16 thru 13) computing F_x_i, F_net_i and F_w_{ij} as a set using (17), (21), and (25), respectively. Note that the 1st term in (17) is zero, and the derivative in (21) will be the same as that found in (41). These modifications are shown below.

$$F\_x_i = F\_x_i + \sum_{j-i+1}^{N} F\_net_j.w_{ji}. \tag{43}$$

$$F\_net_i = F\_x_i.\frac{1}{2}(1 - x_i^2). \tag{44}$$

$$F\_w_{ij} = F\_net_i.x_j \tag{45}$$

To this point, the back-propagation computation has mirrored (with minor modifications) the standard backpropagation algorithm. However, because of the recurrency in the network, as the network performs the internal core iterations for its forward computation, the current outputs become inputs in the next iteration in time. Now in the BPTT calculation, the recurrent inputs (both neighbor inputs and self-recurrent inputs) are connected through time to their corresponding output nodes in the previous iteration. We must therefore compute the feedback to these input nodes and add them to their respective output nodes in the previous iteration. We do this by applying (43) for all the input nodes. Once these values have been computed, they are stored in a vector, *F_Y* for use in the previous core iteration.

Since in the forward computation the connector node (1st active node) is connected to an input for each of its four neighbors, then the feedback to the connector node, F_Y_c, consists of feedback to those four input nodes.

$$F\_Y_c = F\_Y_c + F\_x_U + F\_x_D + F\_x_L + F\_x_R \tag{46}$$

where the *U,D,L,R* subscripts represent direction of the corresponding neighbor.

Likewise, the feedback to the self-recurrent inputs must be held over for their corresponding outputs in the previous iteration.

$$F\_Y_j = F\_Y_j + F\_x_{j+m} \tag{47}$$

where $j = 1$ to n, n is the number of active nodes, and m is the number of inputs.

Note that these computations occur as we move backward through time for each core iteration.

A.3.1.2 BPTT Algorithm for the CSRN with a GMLP Core

The basic steps for BPTT for a CSRN utilizing a GMLP core are listed in A.4a and A.4b below.

Algorithm A.4a: BPTT Computation for the CSRN.

1) Loop backward thru core iterations performing steps 2 – 7.
2) For each cell
3) Compute backward calc. for cell (see Alg. 4b)
4) Initialize the hold-over vector, *F_Y*, to zero.
5) For each cell perform steps 5 & 6 to update feedback to recurrent inputs, and store for prev. iteration)
6) Compute feedback to connector node from 4 neighbor inputs (46)
7) Compute feedback to each active node from its corresponding self-recurrent input (47)

Algorithm A.4b: BPTT Computation for a single cell in the GMLP core.

1) Create temp. array for F_net.
2) Initialize F_x(input nodes set to zero, active notes set to F_Y)
3) Compute feedback to scaling weight, F_W_s. (28).
4) Compute feedback to last active node F_x_{17}. (29).
5) Compute feedback to summer of last node, F_net_{17}. (41)
6) Compute F_W_{17j} for all weights connected to last node. (42)
7) Loop backward thru remaining active nodes computing steps 8-10 as a set.
8) F_x_i. (43)
9) F_net_i. (44)
10) F_W_{ij}. (45)
11) Loop backward thru input nodes
12) Loop thru all nodes after current input node
13) Compute feedback to input node, F_x_i. (43)

A.4 Summary

In Section A.1, we review supervised learning, and present the ordered derivative and the chain-rule for ordered derivatives. In section A.2, we present a feed-forward GMLP network and derive its basic backpropagation equations. In section A.3, we describe the architecture of the CSRN, and present the GMLP core network, and then derive the BPTT equations for the network. Derivations for all computations are provided along with algorithms for practical implementation.