Circuits Systems Signal Process
Vol. 12, No. 2, 1993, Pp. 177–210

Morphology Neural Networks: An Introduction with Applications*

Jennifer L. Davidson[1] *and Frank Hummer*[2]

Abstract. The area of artificial neural networks has recently seen an explosion of theoretical and practical results. In this paper, we present an artificial neural network that is algebraically distinct from the classical artificial neural networks, and several applications which are different from the typical ones. In fact, this new class of networks, called *morphology neural networks*, is a special case of a general theory of artificial neural nets, which includes the classical neural nets. The main difference between a classical neural net and a morphology neural net lies in the way each node algebraically combines the numerical information. Each node in a classical neural net combines information by multiplying output values and corresponding weights and summing, while in a morphology neural net, the combining operation consists of adding values and corresponding weights, and taking the maximum value. We lay a theoretical foundation for morphology neural nets, describe their roots, and give several applications in image processing. In addition, theoretical results on the convergence issues for two networks are presented.

1. Introduction

Morphology neural nets arose out of an investigation of image algebra and its relationship to artificial neural networks [10]. Image algebra, developed by the United States Air Force, can be viewed as a high-level image processing language, which has applications in algorithm development, optimization, comparison, and efficient software and parallel hardware design [13]. Formally, the image algebra is a heterogeneous algebra in the sense of Birkhoff [1]. Image algebra is capable of describing any image-to-image transform that can be defined in terms of finite algorithmic procedures [12], as well as those transformations for which the image has a finite number of gray values [11]. In essence, any

* Received February 2,1992; Accepted May 15, 1992. This research was supported in part by National Science Foundation, Contract No. ECS-9010403.

[1] Department of Electrical Engineering, Iowa State University, Ames, IA 50011.

[2] Department of Mathematics, Iowa State University, Ames, IA 50011.

algorithm which can be implemented on a digital computer can be expressed in image algebra; this includes most of the typical neural net algorithms. Image algebra allows a generalized product between an image and a template, which can be used to describe a generalized neural net node computation. Two specific cases of this general product give the classical neural computation and the morphology neural computation. In Section 2 we describe the image algebra product, and how a generalized neural net architecture and node computation can be expressed in image algebra. We will assume that the reader is already familiar with classical neural nets; for a general introduction, see [7]. As will be seen by the results presented in this paper, morphology neural nets provide a novel method of solution to a class of pattern recognition problems of a nonlinear type, specifically those following a form as expressed in Equation (1.2) below. We apply this basic technique to emulate several image processing transformations (Examples 1, 2 and 3), and provide several networks which have learning rules to three image processing problems.

One main difference between classical nets and morphology nets is the algebraic computation at each node. In classical neural nets, each node gathers output values $\mathbf{a} = (a_1, \ldots, a_N)$ from the previous layer nodes, and multiplies by the corresponding weights $\{\mathrm{w}_{ji}\}$; these pairs are then summed, and passed through the present node's activation function f_j:

$$\mathrm{b}_j = f_j\left(\sum_{i=1}^{N} \mathrm{a}_i \mathrm{w}_{ji}\right). \tag{1.1}$$

In a morphology neural network, the nodal computation is

$$\mathrm{b}_j = f_j\left(\bigvee_{i=1}^{N} \mathrm{a}_i + \mathrm{w}_{ji}\right). \tag{1.2}$$

This algebraically different computation enables a morphology neural net to perform distinctly different tasks from classical neural nets. The main reason is the underlying algebraic structure of the values used in the computation. Classical neural nets use the real ($\mathbb{R}$) or complex ($\mathbb{C}$) numbers to perform the numerical combination in the network, with the corresponding field $(\mathbb{R}, +, *, 0, 1)$ or $(\mathbb{C}, +, *, 0, 1)$, respectively, providing algebraic structure to the methods of solutions. Here, 0 is the identity for the addition (+) operation, and 1 is the identity for the multiplication ($*$) operation. In morphology neural nets, the underlying algebraic structure is the *lattice* of extended real numbers, $\mathbb{R}_{\pm\infty} = \mathbb{R} \cup \{-\infty, \infty\}$. The operation $\bigvee$, maximum or least upper bound, replaces the + operation above, and the operation +, real extended addition, replaces the $*$ operation above. The value $-\infty$ plays the same role as the value 0 above, and the value 0 replaces the value 1 above. Thus, the number system is

$$\left(\mathbb{R}_{\pm\infty}, \bigvee, +, -\infty, 0\right). \tag{1.3}$$

It is possible to describe a transform which has the spirit of a linear transformation by mimicking a linear transformation, substituting the operations in the linear transform with the corresponding ones from the lattice. The resulting transform is called a *lattice transform*, from the study of minimax matrix algebra [2]. While we will not explicitly be using results from minimax algebra, it is worthwhile to note that many theoretical aspects of this matrix transformation theory parallel those established in linear algebra.

The operation inside the brackets of (1.1) is a linear operation. Understanding linear systems and linear functionals has led to a greater understanding of complex dynamical systems. As it is possible to view a classical neural net as a complex dynamical system, the analysis of classical neural nets has benefited from the research in the linear domain. It is highly probable that morphology neural networks will continue to benefit from the applications of minimax algebra results in a similar way.

In Section 2, we present a more detailed description of how the lattice $\mathbb{R}_{\pm\infty}$ plays a role in the morphology net. In Section 3, the basic underlying theory is given. In Section 4, we give several examples of morphology neural nets, with and without learning rules, and for two of them, prove convergence theorems. The paper concludes with a discussion of future research.

2. Image algebra and its relation to neural nets

This section contains an explanation of the image algebra model which describes a general neural net [10], and how this gave rise to morphology neural nets.

The image algebra was originally developed for image processing purposes, although the theory is in the abstract and can be used to express manipulation of data sets more general than typical one- or two-dimensional signals. The current morphology nets have applications in image processing, and we will interpret image algebra in this context. We next describe some operands and operations in the image algebra which are useful to neural net description.

An *image* is a function from a subset $\mathbf{X}$ of N-dimensional Euclidean space $\mathbb{R}^N$ to a set of values, denoted by $\mathbb{F}$. Thus, $\mathbf{a} \in \mathbb{F}^{\mathbf{X}}$ has form $\{(\mathbf{x}, \mathbf{a}(\mathbf{x})): \mathbf{x} \in \mathbf{X}, \mathbf{a}(\mathbf{x}) \in \mathbb{F}\}$, or, if $\mathbf{X}$ is finite with q elements, $\{(\mathrm{i}, \mathbf{a}(\mathrm{i})): \mathrm{i} = 1, \ldots, \mathrm{q}, \mathbf{a}(\mathrm{i}) \in \mathbb{F}\}$. A *template* is an element of $(\mathbb{F}^{\mathbf{X}})^{\mathbf{Y}}$, where $\mathbf{X} \subset \mathbb{R}^N$, $\mathbf{Y} \subset \mathbb{R}^M$. Equivalently, a template $\mathbf{t}$ is a function, $\mathbf{t}: \mathbf{Y} \to \mathbb{F}^{\mathbf{X}}$, having form

$$\{(\mathrm{y}, \{\mathrm{x}, \mathbf{t}_{\mathrm{y}}(\mathrm{x})\}): \mathrm{y} \in \mathbf{Y}, \{\mathrm{x}, \mathbf{t}_{\mathrm{y}}(\mathrm{x})\} \in \mathbb{F}^{\mathbf{X}}\},$$

where we write $\mathbf{t}_{\mathbf{y}}$ instead of $\mathbf{t}(\mathbf{y})$ for notational convenience. Thus, a template is a function such that for every point $\mathbf{y}$ in its domain, it is assigned some image $\mathbf{t}_{\mathbf{y}} \in \mathbb{F}^{\mathbf{X}}$.

An invariant template that is used with linear convolution is often depicted by drawing the pixel locations and corresponding values where the template has nonzero values; this is called the *support* of the template. The pixel locations

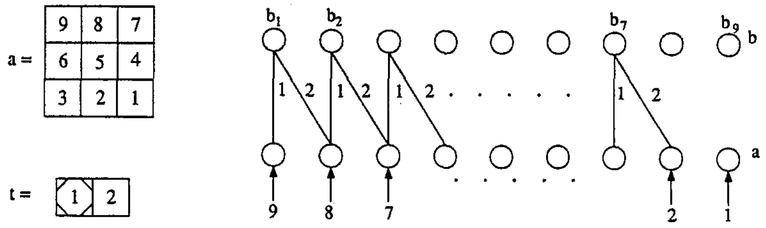


Figure 1. Mapping an image **a** and a template **t** to a classical neural net architecture.

where the template has zero values do not contribute to the sum and are essentially ignored. For example, the template **t** in Figure 1 is such a template. Typically, a two-dimensional template $\mathbf{t} \in (\mathbb{R}^N_{\pm\infty})^M$ has as its domain a finite subset of $\mathbb{Z} \times \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers. If it is an invariant template, then it can be represented pictorially by drawing its support and the finite-valued numbers in its support. The template in Figure 1 "looks the same" no matter where it is shifted. Variant templates typically do not have that property, and must be explicitly defined or have their values expressed as equations.

Images are transformed through use of a template and a generalized operation $\textcircled{\gamma}$ defined between an image and a template. For the remainder of the paper we will assume that **X** and **Y** are finite subsets of $\mathbb{R}^N$ and $\mathbb{R}^M$, respectively. For $\mathbf{a} \in \mathbb{F}^{\mathbf{X}} = \mathbb{F}^N$, $(\mathbb{F}^{\mathbf{X}})^{\mathbf{Y}} = (\mathbb{F}^N)^M$, we have

$$\begin{aligned}\mathbf{a} \textcircled{\gamma} \mathbf{t} &= \mathbf{b} \in \mathbb{F}^{\mathbf{Y}}, \mathbf{b} \equiv \{(\mathbf{y}, \mathbf{b}(\mathbf{y})) \colon \mathbf{b}(\mathbf{y}) \\ &= (\mathbf{a}(\mathbf{x}_1 o \mathbf{t}_{\mathbf{y}}(\mathbf{x}_1))\gamma(\mathbf{a}(\mathbf{x}_2) o \mathbf{t}_{\mathbf{y}}(\mathbf{x}_2))\gamma \cdots \gamma(\mathbf{a}(\mathbf{x}_N) o \mathbf{t}_{\mathbf{y}}(\mathbf{x}_N))\} \\ &= \{(\mathbf{y}, \mathbf{b}(\mathbf{y})) \colon \mathbf{b}(\mathbf{y}) = \Gamma\{(\mathbf{a}(\mathbf{x}_i) o \mathbf{t}_{\mathbf{y}}(\mathbf{x}_i)) \colon i = 1, \ldots, N\}\}. \end{aligned} \tag{2.1}$$

Here, o is a binary operation on $\mathbb{F}$, and γ is a binary and commutative operation on $\mathbb{F}$. (For a more general description of image algebra, see [13].) It is this general operation which allows a variety of different combining operations at each node in a neural network, giving rise to different types of neural networks.

For example, if we set $\mathbb{F} = \mathbb{R}$, $\gamma = +$ (so $\Gamma = \sum$), $o = *$, then we have a *generalized convolution* $\oplus$:

$$\mathbf{a} \oplus \mathbf{t} = \mathbf{b} \in \mathbb{R}^{\mathbf{Y}}, \ \mathbf{b} \equiv \left\{ (\mathbf{y}, \mathbf{b}(\mathbf{y})) \colon \mathbf{b}(\mathbf{y}) = \sum_{i=1}^{N} \mathbf{a}(\mathbf{x}_i)\mathbf{t}_{\mathbf{y}}(\mathbf{x}_i), \mathbf{y} \in \mathbf{Y} \right\}. \tag{2.2}$$

The $\oplus$ operation can express any linear operation, and of course be used to represent Equation (1.1) in the obvious way. For example, a typical approach to using neural nets for image processing is to represent an image with N pixels in its domain as an N-vector, scanning the image from the top-left corner to the bottom right corner, $\mathbf{a} = (\mathrm{a}_1, \ldots, \mathrm{a}_N)$; these represent the N node values in one layer of the neural net. If the image-to-image transformation results in an image having the same domain as the input image, then the neural net architecture will have N pixels at each level. The template that maps one image to another is represented as the weights between successive layers of nodes in the neural net.

We denote the weight from lower layer node i to upper layer node j by $\mathbf{w}_{ji}$. The weight $\mathbf{w}_{ji}$ gets set to template value $\mathbf{t}_j(i)$. See Figure 1 for an example of a linear neural network, which has all activation functions $f_j(x) = x$.

For this particular example, the template $\mathbf{t}$ is invariant, its region of support has two pixels, and (for simplicity) it wraps around the boundary. Also, zero weights are not displayed. In order not to clutter up the figure, not all nonzero weights are shown. In the general case for a classical neural network, if each node j has arbitrary activation function f_j, then the output of the upper layer can be written in image algebra as

$$f(\mathbf{a} \oplus \mathbf{t}) = (f_1(\mathbf{a} \oplus \mathbf{t}), \ldots, f_N(\mathbf{a} \oplus \mathbf{t})). \tag{2.3}$$

We may view a network as *fully connected*, that is, all nodes in one layer are connected to all nodes in the layer above it, or, if the network has zero weights, then we may view those particular weights as not existing. We say this latter type of network has *limited connections*. For various reasons we may wish to view a fully connected network as a limited connection network, particularly if many of the weights are not used in the computation of the nodal value.

Now taking different values for the value set and operations, we set $\mathbb{F} = \mathbb{R}_{\pm\infty}$, $\gamma = \vee$ (so $\Gamma = \bigvee$), where $\bigvee$ is the maximum operation,$o = +$ (real extended addition). Then we define the *additive maximum operation* $\boxed{\vee}$:

$$\mathbf{a} \boxed{\vee} \mathbf{t} = \mathbf{b} \in \mathbb{R}^{\mathbf{Y}}_{\pm\infty}, \ \mathbf{b} \equiv \left\{ (\mathbf{y}, \mathbf{b}(\mathbf{y})) \colon \mathbf{b}(\mathbf{y}) = \bigvee_{i=1}^{N} \mathbf{a}(\mathbf{x}_i) + \mathbf{t}_\mathbf{y}(\mathbf{x}_i), \mathbf{y} \in \mathbf{Y} \right\}. \tag{2.4}$$

This operation is the one used to describe the combining operation at each node in a morphology neural net. While for almost all applications in image processing, the input image $\mathbf{a}$ has finite values (that is,it does not have either value $-\infty$ or ∞), for consistency it is necessary to define the addition operation between all elements of $\mathbb{R}_{\pm\infty}$. The operation + in $\mathbb{R}_{\pm\infty}$ has been defined in a consistent manner [2] to be:

$$\begin{aligned} a + (-\infty) = (-\infty) + a = -\infty \quad & a \in \mathbb{R}_{-\infty} \\ a + \infty = \infty + a = \infty \quad & a \in \mathbb{R}_{-\infty} \\ (-\infty) + \infty = \infty + (-\infty) = -\infty \quad & \end{aligned}$$

Here $\mathbb{R}_{-\infty} = \mathbb{R} \cup \{-\infty\}$. However, for the rest of this paper we will make the reasonable assumption that all input images $\mathbf{a} \in \mathbb{R}^{\mathbf{X}}_{\pm\infty}$ will have only finite values, that is, $\mathbf{a} \in \mathbb{R}^{\mathbf{X}}$. If $\mathbf{a}$ is a *boolean* image, then $\mathbf{a}$ must have only values of 0 and 1. If $\mathbf{t}$ is a *boolean* template, however, the only values $\mathbf{t}$ may assume are 0 and $-\infty$.

As mentioned earlier, the major distinction from the classical nets is the different value set. Morphology neural nets use the values in $\mathbb{R}_{\pm\infty}$. In image processing applications, the template values outside its support are ignored. To accomplish this in context of the value set $\mathbb{R}_{\pm\infty}$, those values must be set to $-\infty$. Thus, we could construct a simple morphology neural net, with the activation functions being the identity function, using the image **a** and template **t** as in Figure 1, but using Equation (1.2) for the calculation. See Figure 2. In performing the additive maximum at each node, we force the weights corresponding to template values outside the region of support to have value $-\infty$. For example, the seventh node value at the upper layer of the network in Figure 2 can be found by calculating

$$\begin{aligned} \mathbf{b}_7 &= \bigvee_{i=1}^{9} \mathbf{a}_i + \mathbf{w}_{7i} = \bigvee_{i=1}^{9} \{\mathbf{a}_1 + \mathbf{w}_{71}, \dots, \mathbf{a}_9 + \mathbf{w}_{79}\} \\ &= \bigvee \{9 + (-\infty), \dots, 4 + (-\infty), 3 + 1, 2 + 2, 1 + (-\infty)\} \\ &= \bigvee \{-\infty, \dots, -\infty, 4, 4, -\infty\} = \bigvee \{4, 4\} = 4. \end{aligned}$$

In general, the basic calculation in a morphology neural net expressed in image algebra is

$$f(\mathbf{a} \boxtimes \mathbf{t}) = (f_1(\mathbf{a} \boxtimes \mathbf{t}), \dots, f_N(\mathbf{a} \boxtimes \mathbf{t})). \tag{2.5}$$

We thus define an *artificial morphology neural network* to be an artificial neural network which uses Equation (2.5) as its basic nodal calculation.

When combining an image and a template, the computation needs to take place only over the support of the template. Formally, the *(infinite) support* $S_{-\infty}(\mathbf{t_y})$ for a template $\mathbf{t} \in (\mathbb{R}^N_{\pm\infty})^M$ is the set of pixels in $\mathbf{X}$ where $\mathbf{t_y}(\mathbf{x}) \neq -\infty$. Thus,

$$S_{-\infty}(\mathbf{t_y}) = \{\mathbf{x} \in \mathbf{X} \colon \mathbf{t_y}(\mathbf{x}) \neq -\infty\}.$$

Some typical supports include the von Neumann or 4-neighborhood of the pixel **y**; and the Moore or 8-neighborhood. See Figure 3 for the description of the two neighborhoods.

We will use the image algebra to express the neural net algorithms, as it is quite efficient in succinctly achieving this. To this end, we next describe several more necessary operations. Given two images $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{\mathbf{X}}_{\pm\infty}$, we define three basic pointwise operations between them:

$$\begin{aligned} \mathbf{a} + \mathbf{b} = \mathbf{c} &\equiv \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) \colon \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) + \mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\} \\ \mathbf{a} * \mathbf{b} = \mathbf{c} &\equiv \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) \colon \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) * \mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\} \\ \mathbf{a} \vee \mathbf{b} = \mathbf{c} &\equiv \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) \colon \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) \vee \mathbf{b}(\mathbf{x}), \mathbf{x} \in \mathbf{X}\}. \end{aligned}$$

The operations of image subtraction, division, and minimum can be defined from these operations. We can also define pointwise operations for templates. Let $\mathbf{s}, \mathbf{t} \in (\mathbb{R}^{\mathbf{X}}_{\pm\infty})^{\mathbf{Y}}$. Then we have

$$\begin{aligned} \mathbf{s} + \mathbf{t} &= \mathbf{r}, \mathbf{r_y}(\mathbf{x}) = \mathbf{s_y}(\mathbf{x}) + \mathbf{t_y}(\mathbf{x}) \\ \mathbf{s} * \mathbf{t} &= \mathbf{r}, \mathbf{r_y}(\mathbf{x}) = \mathbf{s_y}(\mathbf{x}) * \mathbf{t_y}(\mathbf{x}) \\ \mathbf{s} \vee \mathbf{t} &= \mathbf{r}, \mathbf{r_y}(\mathbf{x}) = \mathbf{s_y}(\mathbf{x}) \vee \mathbf{t_y}(\mathbf{x}). \end{aligned}$$

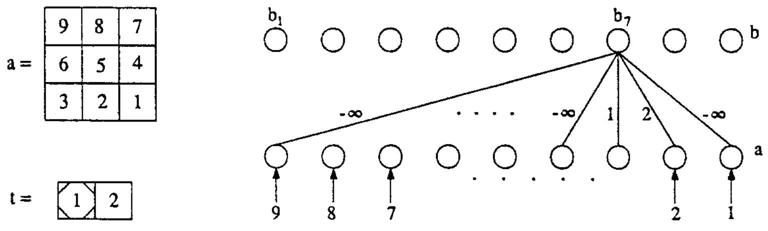


Figure 2. Mapping an image **a** and a template **t** to a morphological neural net architecture.

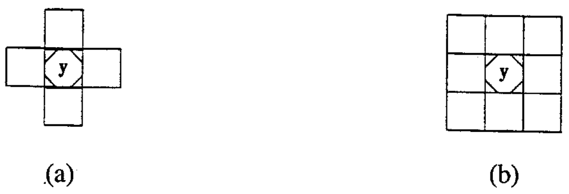


Figure 3. (a) The von Neumann or 4-neighborhood of **y**; (b) the Moore or 8-neighborhood of **y**.

Another useful function is the *characteristic function*. While the typical characteristic function χ acts on an image (or template) by producing a 0/1 value output:

$$\chi_{>0}(\mathbf{a}) = \mathbf{b},\ \ \mathbf{b}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{a}(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases},$$

we will need one that produces a 0 or $-\infty$ valued output:

$$\chi_{>0}^{-\infty}(\mathbf{a}) = \mathbf{b},\ \ \mathbf{b}(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{a}(\mathbf{x}) > 0 \\ -\infty & \text{otherwise.} \end{cases}$$

In a similar manner we can apply χ to a template. Also, the comparison $>$ can be replaced with any other, such as $=$, $\leq$, etc.

We conclude this section with a short example. We give the algorithm for the synchronous Hopfield network in image algebra [10]. Let N be the number of nodes in the Hopfield net, and P be the number of exemplar patterns. Let $\mathbf{X} = \{1, 2, \ldots, N\} \subset \mathbb{Z}$. The weights are determined by setting

$$\mathbf{w}_{ji} = \mathbf{t}_j(i) = \begin{cases} \sum_{k=1}^{P} x_j^k x_i^k & i \neq j \\ 0 & i = j \end{cases},$$

where x_j^k is the jth element of the exemplar for the pattern class k. Let $\mathbf{c} \in \mathbb{R}^{\mathbf{X}}$ be a bipolared valued image as input, having values 1 or -1. Then the Hopfield algorithm is:

```
repeat
        $\mathbf{a} := \mathbf{c}$
        $\mathbf{b} := \mathbf{a} \oplus \mathbf{w}$
        $\mathbf{c} := \chi_{>0}(b) - \chi_{<0}(\mathbf{b}) + \mathbf{a} * \chi_0(\mathbf{b})$
until
        $\mathbf{c} = \mathbf{a}$
```

This is a good example of how concise the image algebra can be in expressing neural net algorithms.

Currently, the investigation of special cases of Equation (2.1) has only covered classical neural nets and morphology neural nets, described by Equations (1.1) and (1.2),respectively. However,it can be easily shown that the value set $\mathbb{F} = \mathbb{R}_{\infty}^{\geq 0} = \{r \in \mathbb{R}\colon r \geq 0\} \cup \{\infty\}$, the set of nonnegative real numbers union $\{\infty\}$, together with the operation $\gamma = \vee$ and $o = *$, is a lattice which is isomorphic to the lattice of extended real numbers in Equation (1.3). Thus, a neural net which has the operation

$$f_j\left(\bigvee_{i=1}^{N} \mathbf{a}_i * \mathbf{w}_{ji}\right)$$

at each node can be shown to mimic the appropriate morphology neural net.

3. The theory of morphology neural networks

Shift-invariant image processing transforms that can be described by Equation (2.4), where $\mathbf{X} = \mathbf{Y}$, are called *morphological transforms*, and are part of the area of image processing known as mathematical morphology. Derived from Minkowski and Hadwiger set theoretic operations, [8] and [6], respectively, and applied to boolean image processing, the two morphological operations of dilation and erosion were used to perform *shape* analysis on images [14]: hence the name *morphology*. It has also been shown [3] that the image algebra operation $\boxtimes$ operation is a generalization of the morphology dilation operation. Since morphology neural nets use the general $\boxtimes$ operation and not the strictly invariant mathematical morphology dilation, we use the image algebra $\boxtimes$ operation to describe our morphology net operations. Although the image algebra Equation (2.4) describes a far more general image transformation than invariant ones, we chose the name *morphology* to describe our networks to maintain the spirit of the basic operation of the network.

The erosion operation can be defined in terms of the dilation, and essentially uses the minimum operation in place of the maximum. For $\mathbf{a} \in \mathbb{R}_{\pm\infty}^{N}$ and $\mathbf{t} \in (\mathbb{R}_{\pm\infty}^{N})^{M}$, the image algebra *additive minimum* operation,which generalizes the mathematical morphology erosion, is defined to be

$$\mathbf{a} \boxed{\wedge} \mathbf{t} = \mathbf{b} \in \mathbb{R}_{\pm\infty}^{\mathbf{Y}}, \quad \mathbf{b} \equiv \left\{ (\mathbf{y}, \mathbf{b}(\mathbf{y})) \colon \mathbf{b}(\mathbf{y}) = \bigwedge_{i=1}^{N} \mathbf{a}(\mathbf{x}_i) + \mathbf{t}_{\mathbf{y}}(\mathbf{x}_i), \mathbf{y} \in \mathbf{Y} \right\}. \tag{3.1}$$

For a more detailed discussion on boolean and grayscale morphology operations, see [15]. We remark that an entirely different approach to a morphology network was presented in [16]. However, that particular model uses the classical neural net to simulate the nonlinear morphology operations, and so is entirely different from the morphology neural net in this paper.

The additive maximum and additive minimum operations are dual operations, and the additive minimum can be expressed in terms of the additive maximum operation. For $\mathbf{a} \in \mathbb{R}_{\pm\infty}^{N}$, $\mathbf{t} \in (\mathbb{R}_{\pm\infty}^{N})^{M}$, we have

$$\mathbf{a} \boxed{\wedge} \mathbf{t} = -(-\mathbf{a} \boxed{\vee} -\mathbf{t}). \tag{3.2}$$

Specifically, the mathematical operation of erosion is expressed in image algebra where $\mathbf{X} = \mathbf{Y}$, and $\mathbf{t}$ is an invariant template, $\mathbf{t} \in (\mathbb{R}_{\pm\infty}^{N})^{N}$, where $-\mathbf{t}'$ replaces $\mathbf{t}$ in Equation (3.2):

$$\mathbf{a} \boxed{\wedge} (-\mathbf{t}') = -(-\mathbf{a} \boxed{\vee} \mathbf{t}'). \tag{3.3}$$

Here, the *transpose* $\mathbf{t}'$ of a template $\mathbf{t}$ is defined by $\mathbf{t}'_{\mathbf{y}}(\mathbf{x}) = \mathbf{t}_{\mathbf{x}}(\mathbf{y})$. All mathematical morphology operations can be expressed as cascades of dilations and erosion, so the $\boxed{\vee}$ operation is sufficient to describe all mathematical morphology transformations.

Just as a classical neural net can emulate a linear transform with the appropriate choice of architecture and activation function, a morphology neural net can emulate a transform of type Equation (2.4). If $\mathbf{a} \in \mathbb{R}_{\pm\infty}^{N}$ and $\mathbf{t} \in (\mathbb{R}_{\pm\infty}^{N})^{M}$, then a morphology neural net which realizes Equation (2.4) has N input nodes with values $\mathbf{a} = (a_1, \ldots, a_N)$, M output nodes with values $\mathbf{b} = (b_1, \ldots, b_M)$, has weights $\mathbf{w}_{ji} = \mathbf{t}_j(i)$, $j = 1, \ldots, M$, $i = 1, \ldots, N$, and has activation functions $f_j(x) = x$, $j = 1, \ldots, M$. This net is fully interconnected, or, as discussed above, can be viewed as a limited connection network if for each weight satisfying $\mathbf{w}_{ji} = -\infty$, the connection between upper node j and lower node i is considered not connected. Hence, given any transform of form Equation (2.4) or (3.1), which includes the boolean and grayscale dilations and erosions, there exists a morphological neural network which can perform the same calculations, and it will use only the maximum operation, that is, it will have nodal computation of form Equation (1.2). However, just as the linear neural nets give us nothing new, neither do the morphology nets which implement the straight additive maximum transform. The more interesting and complex results arise from deviating from the well known. In this vein, in Section 4 we show two morphology neural nets that have learning rules.

4. Examples of applications

In this section we describe six examples of specific morphology neural nets. The first two examples are straightforward realizations of the $\boxed{\vee}$ and $\boxed{\wedge}$ operations, while the third is the realization of the generalized opening transform in morphology, which is a cascade of the $\boxed{\wedge}$ and $\boxed{\vee}$ operations. These three networks have no learning rules associated with them, and all three are valid for both boolean or grayscale data. The remaining examples are networks with learning rules. Example 4 describes a boolean dilation network with learning for an invariant template, and Example 5 a grayscale dilation with learning for an invariant template. Example 6 is a network which learns the grayscale additive maximum for the general case of a variant template. Formal proof is provided for convergence of the weights for Examples 4, 5, and 6. In all three cases, the algorithm is able to converge to weights which perfectly recall all the training data. Implementation of the algorithm on image data is also given as a demonstration.

Example 1. Additive maximum or generalized dilation network (without learning). This network, as the name implies, implements the additive maximum operation as given in Equation (2.4). This is the network that was described at the end of the previous section. Figure 2 portrays a specific example.

Example 2. Additive Minimum or Generalized Erosion Network (without learning). This network implements the additive minimum operation as given in Equation (3.1). As mentioned previously, the additive minimum can be described in terms of the additive maximum operation. For purposes of this paper, we will be most interested in Equation (3.3), which happens to be a true expression regardless of whether $\mathbf{t}$ is invariant or not. Equation (3.3), $\mathbf{a} \boxed{\wedge} (-\mathbf{t}') = -(-\mathbf{a} \boxed{\vee} \mathbf{t}')$, suggests how to construct our morphology neural net. We note that we will have an input layer and an output layer of neurons, and one set of weights interconnecting them. We will use the accepted convention that the input layer nodes perform no calculations, while nodes at all other layers do, unless otherwise noted. We wish to use the additive maximum operation, so the values $-\mathbf{a}$ will be the input, and the weight from input node i to output node j will have value $\mathbf{w}_{ji} = \mathbf{t}'_j(i) = \mathbf{t}_i(j)$, where $j \in \mathbf{Y} = \{1, \ldots, M\}$, and $i \in \mathbf{X} = \{1, \ldots, N\}$. We see upon calculating the values in the network we designed thus far that the expression $-\mathbf{a} \boxed{\vee} \mathbf{t}'$ is the current state of values at the output nodes. Thus, we need to negate each of these values so that Equation (3.3) will be calculated. Hence, we choose our activation function to be

$$f_j(x) = -x, \qquad j = 1, \ldots, M.$$

The output of the network will then be

$$f(-\mathbf{a} \boxed{\vee} \mathbf{t}') = -(-\mathbf{a} \boxed{\vee} \mathbf{t}'),$$

as desired.

Example 3. Generalized Opening Network (without learning). In mathematical morphology, the sequence of operations

$$\left(\mathbf{a} \boxed{\wedge} (-\mathbf{t}')\right) \boxed{\vee} \mathbf{t},$$

called the *opening* of **a** by **t**, is a well-used technique, and can be applied to filter out speckle-type noise. Using Equation (3.3) we can also write

$$\left(\mathbf{a} \boxed{\wedge} (-\mathbf{t}')\right) \boxed{\vee} \mathbf{t} = \left[-(-\mathbf{a} \boxed{\vee} \mathbf{t}')\right] \boxed{\vee} \mathbf{t}.$$

To design a morphology neural net that implements the opening, we first note that we will need an input layer, a hidden layer **h**, and an output layer **b**. Let $\mathbf{w}^1$ denote the weights between the input and hidden layers, and $\mathbf{w}^2$ denote the weights between the hidden and output layers. Denote the two activation functions by f^1 and f^2 for the hidden layer and output layer neurons, respectively. Using the results from Example 2, if we input the values $-\mathbf{a}$ and set $f_j^1(x) = -x$, then the hidden layer values are $\mathbf{h} = -(-\mathbf{a} \boxed{\vee} \mathbf{w}^1)$, where $\mathbf{w}_{ji}^1 = \mathbf{t}_j'(i)$. If the upper layer weights are set to the values $\mathbf{w}_{ji}^2 = \mathbf{t}_j(i)$, and each activation function at the output nodes is set to $f^2(x) = x$, then the output neurons will have values $\mathbf{b} = f(\mathbf{h} \boxed{\vee} \mathbf{w}^2) = \mathbf{h} \boxed{\vee} \mathbf{w}^2 = [-(-\mathbf{a} \boxed{\vee} \mathbf{w}^1)] \boxed{\vee} \mathbf{w}^2$, which is the desired calculation. See Figure 4 for a pictorial representation of this network.

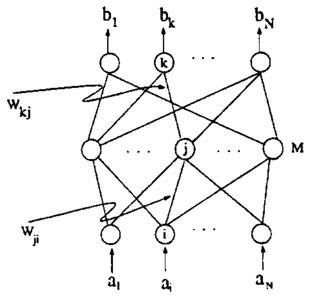


Figure 4. The opening morphology neural network for Example 3.

Note that no assumption was made about whether **t** was invariant or not, nor that $M = N$. Thus, this is a *generalized* opening network.

Another commonly used mathematical morphology transform is the *closing*. The general form is described in image algebra by the equation

$$(\mathbf{a} \boxed{\vee} \mathbf{t}) \boxed{\wedge} (-\mathbf{t}').$$

The construction of a morphology neural net to calculate this is straightforward, and we leave this to the reader as an exercise.

Up to this point we have had no need to make a distinction between boolean and grayscale nets. Our next example is a net with a learning rule, that is, a rule for updating weights. This morphology neural net is the dilation neural net but with a training rule. It is constructed so that after it is trained on a set of training data consisting of inputs plus their respective dilation, the weights will closely approximate the template used to originally dilate the inputs. This algorithm will work only for boolean images and templates. The grayscale case, given immediately after the boolean network, has a distinctly different learning rule.

Example 4. The Boolean Dilation Net with Learning. [9] This is a network which has a training rule, and, once properly trained, will produce at the output nodes an approximation of the dilated version of the input which is applied to the input nodes. As its title suggests, this network is applicable only to boolean data and templates. The network has two layers of nodes, an input and an output layer. We will give an example for the network where **t** is invariant, although the algorithm is general enough to use with any variant template, including cases where $\mathbf{X} \neq \mathbf{Y}$.

The set of training data for this network consists of **P** pairs of images, $(\mathbf{a}^k, \mathbf{d}^k)$, $k = 0, \ldots, \mathbf{P} - 1$, where $\mathbf{a}^k$ represents the input image and $\mathbf{d}^k = \mathbf{a}^k \boxtimes \mathbf{t}$ is the input image dilated with an ideal (invariant) template **t**, which is the same for all images. In practical cases, the template **t** is assumed to be unknown. The weights are initially set to some values, which may or may not be the template values, as will be discussed momentarily. Then, the image **a** is applied to the input layer, and the output calculated at the output layer using the morphology neural net calculation as in Equation (2.5). The activation functions for this net are all

$$f_j(x) = x, \qquad j = 1, \ldots, N.$$

The main goal of this net is to determine a set of weights for the network which will correctly produce the corresponding output $\mathbf{d}^k$ when $\mathbf{a}^k$ is applied to the input nodes. This is done by changing the weights using a training rule.

For practical considerations, we will assume that the network is a limited-connection network, and in fact, assume the connections occur over only a rectangular or square support $\mathcal{N}(j)$ in the image domain **X** which includes the assumed support $S_{-\infty}(\mathbf{t}_j)$ of the template, $j = 1, \ldots, N$. For example, if we assume the template **t** had a 4-neighborhood support, then we might choose the 8-neighborhood to be $\mathcal{N}(j)$. The other weights into node j are considered to be disconnected, or, as mentioned above, have value $-\infty$. This has the effect of reducing the number of training pairs needed to get a good representation of the ideal template **t**.

Data pairs such as these may represent a morphological process taking place with an unknown ideal template **t**. Hence, a network such as this and the ones in Examples 5 and 6 allow for the recovery of a transformation or template from a set of data where the transform is assumed to be of the form of an additive maximum. Thus, the weights need to be adjusted via a training rule from their original values so that they exactly match those from **t**, or are very close, and produce correct results on the training data. Since this is a boolean network, it is necessary to recall that $\mathbf{a} \in \{0, 1\}^N$, and $\mathbf{t} \in (\{0, -\infty\}^N)^N$. The following table, given in Figure 5, represents the training rule, or the change of weights we would like to make. We make use of the fact that if a template has finite nonnegative values on its support, that is, $\mathbf{t}_{\mathbf{y}}(\mathbf{x}) \geq 0$ for $\mathbf{x} \in S_{-\infty}(\mathbf{t}_{\mathbf{y}})$, we can easily show $\mathbf{a} \leq \mathbf{a} \boxtimes \mathbf{t}$. Initially, all values for all weights in the neighborhood $\mathcal{N}(j)$ are set to 0. Thus, the template corresponding to these weights has perhaps larger support than the unknown template $\mathbf{t}$. The values in the N-vector $\mathbf{c}$ are the ones calculated after the input is passed through the network is initialized.

case	a_i^k	d_j^k	c_j^k	new value for w_{ji} is
1	0	0	1	current value
2	0	1	0	current value
3	1 or 0	1	1	current value
4	1 or 0	0	0	current value
5	1	0	1	$-\infty$
6	1	1	0	0

Figure 5. Table of change-of-weight rules for boolean dilation network.

As an example of how these rules were derived, consider cases 3 and 4. If the desired output and calculated output at location j have the same value,then no matter what the input image value $\mathbf{a}_i^k$ is in $\mathcal{N}(j)$, we wish the weight $\mathbf{w}_{ji}$ to remain the same, as currently it is producing the correct output. However, consider case 5: the calculated value is 1, which is greater than a desired value of 0, and $\mathbf{a}_i^k$ has value 1. We know that

$$\mathbf{d}_j^k = 0 = \bigvee_{i \in S_{-\infty}(\mathbf{t}_j)} \mathbf{a}_i^k + \mathbf{t}_{ji} \Rightarrow \mathbf{a}_i^k = 0$$

for all $i \in S_{-\infty}(\mathbf{t}_j)$. But

$$\mathbf{c}_j^k = 1 = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{w}_{ji}$$

implies that there exists at least one $i \in \mathcal{N}(j)$ such that $\mathbf{a}_i^k = 1$ (as $\mathbf{w}_{ji} = 0$, or $\mathbf{w}_{ji} = -\infty$). So for each value $i \in \mathcal{N}(j)$, where $\mathbf{a}_i^k = 1$, we set the corresponding weight $\mathbf{w}_{ji}$ that contributes to calculating $\mathbf{c}_j^k$ to $-\infty$, so that the sum $\mathbf{a}_i^k + \mathbf{w}_{ji}$ does not contribute to calculating the finite value for $\mathbf{c}_j^k$. The remaining rules were derived similarly. The convergence properties of the grayscale case for variant templates, Example 6, show that this network will converge to a set of weights that will always work perfectly on the training data. Also, using these results, we can say that if the weights are initialized on $\mathcal{N}(j)$ to have values that are greater than or equal to a fixed threshold $\mathbf{T}$, then those values can only get changed to $-\infty$ and never will a value of $-\infty$ get changed back to 0. Hence, case number 6 in Figure 5 will never occur. However, it was included for completeness in the table.

The algorithm for this network is

```
INITIALIZE WEIGHTS TO 0 IN $\mathcal{N}(j)$
REPEAT
      do $k = 0,\ P - 1$
   C     calculate output node values for input $\mathbf{a}^k$
           $\mathbf{c}_j^k = \bigvee_{i=1}^{N} \mathbf{a}_i^k + \mathbf{w}_{ji},\ j = 1, \ldots, M$
   C     adjust weights by learning rule, only changing weights
           $\mathbf{w}_{ji}$ in the neighborhood $\mathcal{N}(j)$ according to Figure 5.
      end do
UNTIL net gives correct response for each training pair
$(\mathbf{a}^k, \mathbf{d}^k),\ k = 0, \ldots, P - 1.$
```

After it has been trained, then, just as in the classical case, the network can be used to produce a dilated version of some image not in the training set, with the hopes that the output produced is close to what the true version would be if the template were known. Of course, if all possible binary images on the domain **X** and their corresponding dilated versions were used to train the net, then, at the end of the training process, the weights would be exactly equal to the template values. As in the case with classical nets, the general idea in using a morphology net to find the weights is that if the training data used is a small but "good" representation of all possible data, then the final weights will be a "good" approximation to the template **t**.

The change-of-weight rule can be expressed algebraically as follows. Let the value for the weight at iteration $k + 1$ be denoted by $\mathbf{w}_{ji}^{k+1}$. Then

$$\begin{aligned} \mathbf{w}_{ji}^{k+1} = &\left\{\chi_{>0}^{-\infty}\left[\mathbf{a}_i^k(\mathbf{d}_j^k - \mathbf{c}_j^k)\right]\right\} \\ &\bigvee \left\{\left(\chi_0^{-\infty}\left[\mathbf{a}_i^k(\mathbf{d}_j^k - \mathbf{c}_j^k)\right]\right) + \mathbf{w}_{ji}^k\right\}. \end{aligned} \tag{4.1}$$

Note the similarity of Equation (4.1) to the generalized delta learning rule:

$$\mathbf{w}_{ji}^{k+1} = \gamma \mathbf{a}_i^k \left(\mathbf{d}_j^k - \mathbf{c}_j^k\right) + \mathbf{w}_{ji}^k.$$

It is understood in Equation (4.1) that the superscripts on the input and dilated training pairs is done modulo **P**, in the case that $k > M$.

Expressed in image algebra, the algorithm is:

```
REPEAT
      do $k = 0,\ P - 1$
   C     calculate output node values
           $\mathbf{c}^k = \mathbf{a}^k \boxed{\vee}\, \mathbf{w}^k$
   C     adjust weights by learning rule
           $\mathbf{w}^{k+1} = \{\chi_{>0}^{-\infty}(\mathbf{s}^k\} \bigvee \{[\chi_0^{-\infty}(\mathbf{s}^k)] + \mathbf{w}^k\}$
        end do
    UNTIL net gives correct response for each training pair
    $(\mathbf{a}^k, \mathbf{d}^k)$, $k = 0, \ldots, P-1$.
```

Here, the template $\mathbf{s}$ is defined by

$$\mathbf{s}_{ji}^k = \begin{cases} \mathbf{a}_i^k(\mathbf{d}_j^k - \mathbf{c}_j^k) & \text{if } i \in \mathcal{N}(j), \\ -\infty & \text{otherwise} \end{cases}, \quad k = 0, \ldots, P-1.$$

This network was run on one set of test images. This was not meant to be a thorough analysis, but a simple demonstration of how the network can perform. The training data used were 20 randomly generated boolean images, where the probability of assigning a value of 1 to a pixel location was slightly less than the probability of assigning a value of 0. This was done to prevent large contiguous areas of black pixels (value 1) which might not allow the weights to change to their correct values. The image size was 64×64 pixels, and the boolean template used was the von Neumann with values of 0 on the support. The neighborhood $\mathcal{N}(j)$ used was the Moore neighborhood. The first 10 test images plus their dilated versions were applied and the learning rule, Equation (4.1), used to update the weight values in the network. After the first ten test images, we applied four images of natural scenes to the network, without training, to determine how well the network produced their correctly dilated versions. The network was then trained on the remaining ten test images, after which the four natural scenes were again applied. There were two measures used to determine the effectiveness of this algorithm on our test data. The first was a measure of the difference between the ideal template values and the weights determined by the network. The number of values in each neighborhood $\mathcal{N}(j)$, that differed from the ideal template values were counted and summed up, for all $j = 1, \ldots, 64$. Due to the result of Theorem 3 in Example 5, since the neighborhood values were initialized to 0 and can only change to $-\infty$, we need only to count then number of zeros in the four corners of $\mathcal{N}(j)$. This gives a rough measure of how closely the weights matched the ideal template values. The second measure, which is more data dependent, used the natural scenes. Their perfectly dilated versions,dilated with the ideal template $\mathbf{t}$, were compared with the output to the network after the original scene was input. Thus, this is a measure of how well the net dilated the natural scene with the weight $\mathbf{w}$, versus its perfect dilation with $\mathbf{t}$. The number of pixel values that differed between the calculated dilation from the network and the dilation produced with $\mathbf{t}$ were counted. In Figure 6 is a selection of some of the image data used. The pixels that are black in Figure 6 have value 0, while white is value 1. The results of the experiment are shown in Figure 7. In Figure 7b, column 1 gives a short description of the black and white image used; column 2 gives the number of test images used to train the network before the natural scene data was applied; column 3 gives the number of pixels differing between $\mathbf{g}_i \boxed{\vee} \mathbf{t}$ and $\mathbf{g}_i \boxed{\vee} \mathbf{w}$, where $\mathbf{g}_i$, $i = 1, \ldots, 4$, are the four natural scene data; and column 4 gives the numbers in column 3 calculated

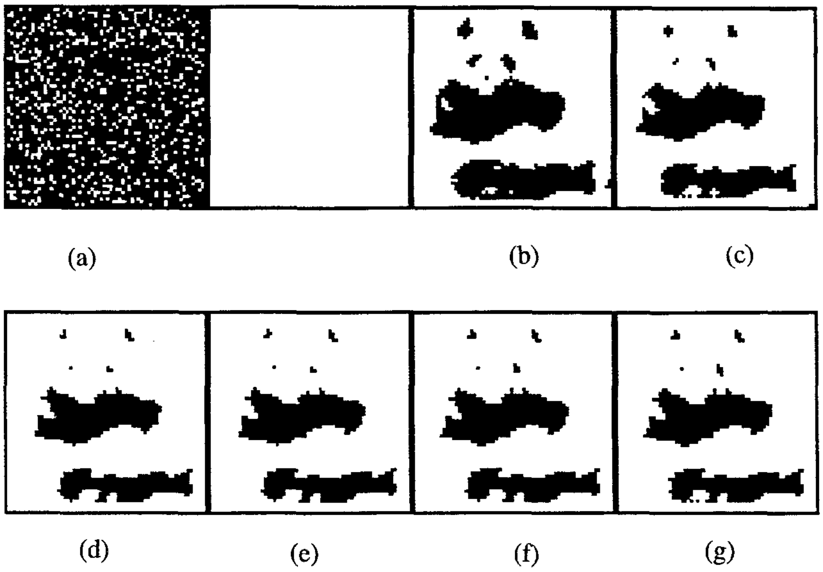


Figure 6. (a) Sample input training data; (b) sample natural scene (panda); (c) image in (b) dilated with von Neumann template; (d) output of net after X = 5 training images applied, for input of (b); (e) output of net after X = 10, for input of (b); (f) output of net after X = 15, for input of (b); (g) output of net after X = 20, for input of (b).

as a percentage of the total possible number of pixels that could differ, which is $64 \times 64 = 4096$.

By the nature of the training rules, once a test image $\mathbf{a}^k$ is applied and the weights are changed to correct for the error at the net's output, the network will always subsequently produce the perfectly dilated version, $\mathbf{d}^k$ at the output nodes when $\mathbf{a}^k$ is applied again. Hence, there is no new information that the network can gain from applying the training pair $(\mathbf{a}^k, \mathbf{d}^k)$ more than once. This network can therefore "learn" the training data perfectly.

Example 5. Grayscale Dilation with Learning for Invariant Templates. This network trains on training data pairs to learn a grayscale dilation, assuming the unknown template is invariant. It is a grayscale version of the boolean network with training as described in the previous example. We present two algorithms: the first uses simply one training pair, while the second uses a set of P training data. Both use the typical approach, initializing the weights, then applying the training data, and finally adjusting the weights according to the training rule. For the first algorithm where only one training pair is used, $(\mathbf{a}, \mathbf{d})$, where $\mathbf{a}$ is the input and $\mathbf{d}$ is the corresponding dilated input, the weights converge after the first iteration. As usual, the learning rule for this net produces an approximation of $\mathbf{t}$ from the input-output pair. The proof of convergence to weights that give perfect reproduction of $\mathbf{d}$ is given below. This result leads in

after X number of training images applied	total number of different values between weights and template values	column #2 calculated as a percentage
X = 5	11331	31%
X = 10	9012	24%
X = 15	7418	20%
X = 20	5842	16%

(a)

description of natural scene data	after X number of training data was applied	# of pixels differing between perfect dilation and net output	column #3 calculated as a percentage (out of 4096)
town	X = 10	113	2.8%
ship		36	.9%
panda		64	1.6%
jet		53	1.3%
town	X = 20	74	1.8%
ship		25	.6%
panda		46	1.1%
jet		29	.7%

(b)

Figure 7. (a) Measure of difference between weight values calculated by net and ideal weight values. (b) Difference between net output and perfectly dilated natural scenes.

a natural way to the second algorithm, which uses a set of P training data to adjust the weights. Convergence for this network is also guaranteed, and the weights to which the network converges allow perfect recall of all the training data. In other words, the network learns the training data perfectly. The second algorithm needs only two passes through the training data set to produce perfect recall of the training data, where one pass is one application of the entire data set $\{(\mathbf{a}^0, \mathbf{d}^0), (\mathbf{a}^1, \mathbf{d}^1), \ldots, (\mathbf{a}^{P-1}, \mathbf{d}^{P-1})\}$.

For the first algorithm, we let **a** be the input image and **d** be the corresponding dilated version, so $\mathbf{d} = \mathbf{a} \boxtimes \mathbf{t}$. Let $\mathcal{N}(j)$ be a neighborhood of j such that $\mathcal{N}(j) \supset S_{-\infty}(\mathbf{t}_j)$, $j = 1, \ldots, N$. We assume a limited-connection network where connections are allowed only in the neighborhood $\mathcal{N}(j)$. We denote the weight value from input node i to output node j at iteration k by $\mathbf{w}_{ji}(k)$. We will assume that for all $i \in S_{-\infty}(\mathbf{t}_j)$ and for all j, $\mathbf{t}_j(i) = \mathbf{t}_{ji} \geq T \geq 0$ for some nonnegative real constant T.

The weights are initially set to the following values:

$$\mathbf{w}_{ji}(0) = \begin{cases} \bigwedge\limits_{\{z: i+z \in \mathcal{N}(j+z)\}} \mathbf{d}_{j+z} - \mathbf{a}_{i+z} & \text{if } i \in \mathcal{N}(j) \\ -\infty & \text{otherwise} \end{cases} \tag{4.2}$$

For $k \geq 1$, we use the following two step rule:

$$\text{Step 1:} \quad \hat{\mathbf{w}}_{ji}(k) = \begin{cases} \mathbf{w}_{ji}(k-1) & \text{if } \mathbf{w}_{ji}(k-1) \geq T \\ -\infty & \text{if } \mathbf{w}_{ji}(k-1) < T \end{cases}$$

$$\text{Step 2:} \quad \mathbf{w}_{ji}(k) = \bigwedge_{\{z: i+z \in \mathbf{X}, j+z \in \mathbf{X}\}} \hat{\mathbf{w}}_{j+z,i+z}(k)$$

Step 2 allows the computation of the minimum value over all corresponding pixels in the neighborhood $\mathcal{N}(j)$; this forces the weights to represent an invariant template after Step 2. Note that this learning rule doesn't use the computed output, $\mathbf{c} = \mathbf{a} \boxed{\vee} \mathbf{w}(0)$, to calculate the new weight value. To show that the weights converge to values $\mathbf{w}$ such that $\mathbf{a} \boxed{\vee} \mathbf{w} = \mathbf{d}$, the following theorem will be useful. For the remainder of this example, we will assume that the invariant template $\mathbf{t}$ is the one that gives the dilated version of $\mathbf{a}$, $\mathbf{a} \boxed{\vee} \mathbf{t} = \mathbf{d}$, and that $\mathbf{a}$ is finite-valued. We will also assume that $j \in \mathcal{N}(j)$ for all j. We will also use the fact that if $\mathbf{a}$ is finite valued, and $\mathbf{t}$ is a template which has no values of ∞ and has nonempty infinite support at each pixel location j, then the image $\mathbf{d}$ given by $\mathbf{d} = \mathbf{a} \boxed{\vee} \mathbf{t}$ is also finite-valued.

Theorem 1. *Let* $\mathbf{a} \boxed{\vee} \mathbf{t} = \mathbf{d}$. *Let* $\mathbf{w}$ *be a template such that*

$$\mathbf{t}_{ji} \leq \mathbf{w}_{ji} \leq \mathbf{d}_j - \mathbf{a}_i \quad \textit{for all } i, j = 1, \ldots, N.$$

Then $\mathbf{a} \boxed{\vee} \mathbf{w} = \mathbf{d}$.

Proof. Let $\mathbf{a} \boxed{\vee} \mathbf{w} = \mathbf{c}$. Then $\mathbf{c}_j = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i + \mathbf{w}_{ji}$ for all j. Now,

$$\mathbf{d}_j = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i + \mathbf{t}_{ji} \leq \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i + \mathbf{w}_{ji} \leq \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i + \mathbf{d}_j - \mathbf{a}_i = \mathbf{d}_j,$$

so $\mathbf{c}_j = \mathbf{d}_j$ for all j, and $\mathbf{c} = \mathbf{d}$. □

In fact, if we let $\mathbf{w}_{ji} = \mathbf{d}_j - \mathbf{a}_j$ for all i in $\mathcal{N}(j)$ and for all j, then by Theorem 1, $\mathbf{a} \boxed{\vee} \mathbf{w} = \mathbf{d}$. Of course $\mathbf{w}$ defined in this way is not guaranteed to be invariant, but the above observation provides the motivation for our initializing $\mathbf{w}(0)$ as above; also, it is clear from Equation (4.2) that $\mathbf{w}(0)$ is invariant. Not surprisingly, $\mathbf{w}(0)$ turns out to be a very good initial guess.

Theorem 2. $\mathbf{a} \boxed{\vee} \mathbf{w}(0) = \mathbf{d}$.

In order to prove Theorem 2, we will need the following lemma.

Lemma 1. *If* $\mathbf{d} = \mathbf{a} \boxed{\vee} \mathbf{t}$, *then* $\mathbf{t}_{ji} \leq \mathbf{dj} - \mathbf{a}_i$ *for all* i *and* j.

Proof of Lemma 1.

$$\mathbf{d}_j = \bigvee_{m \in \mathcal{N}(j)} \mathbf{a}_m + \mathbf{t}_{jm} \geq \mathbf{a}_i + \mathbf{t}_{ji},$$

for all $i \in \mathcal{N}(j)$, and for all $j = 1, \ldots, N$. If $i \notin \mathcal{N}(j)$, then since $\mathbf{t}_{ji} = -\infty$ and since $\mathbf{d}_j$ and $\mathbf{a}_i$ are each finite-valued, their difference $\mathbf{d}_j - \mathbf{a}_i$ is finite-valued and hence greater than $-\infty$. □

Proof of Theorem 2. By Lemma 1, $\mathbf{t}_{j+z,i+z} \leq \mathbf{d}_{j+z} - \mathbf{a}_{i+z}$ for each $i + z$ and $j + z \in \mathbf{X}$. Since the template $\mathbf{t}$ is invariant, we have $\mathbf{t}_{j+z,i+z} = \mathbf{t}_{ji}$ for all z such that $i+z \in \mathcal{N}(j+z)$. Thus, $\mathbf{t}_{ji} \leq \mathbf{d}_{j+z} - \mathbf{a}_{i+z}$ for all z such that $i+z \in \mathcal{N}(j+z)$. Therefore,

$$\mathbf{t}_{ji} \leq \bigwedge_{\{z: i+z \in \mathcal{N}(j+z)\}} \mathbf{d}_{j+z} - \mathbf{a}_{i+z}.$$

Because $0 \in \{z : i + z \in \mathcal{N}(j + z)\}$, it follows that

$$\bigwedge_{\{z: i+z \in \mathcal{N}(j+z)\}} \mathbf{d}_{j+z} - \mathbf{a}_{i+z} \leq \mathbf{d}_{j+0} - \mathbf{a}_{i+0} = \mathbf{d}_j - \mathbf{a}_i.$$

We now have that, if $i \in \mathcal{N}(j)$,

$$\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(0) \leq \mathbf{d}_j - \mathbf{a}_i.$$

If $i \notin \mathcal{N}(j)$, then $\mathbf{w}_{ji} = -\infty$, but because $S_{-\infty}(\mathbf{t}_j) \subset \mathcal{N}(j)$, we know that $\mathbf{t}_{ji} = -\infty$, and again we have

$$\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(0) \leq \mathbf{d}_j - \mathbf{a}_i.$$

Thus we have

$$\mathbf{d}_j = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i + \mathbf{w}_{ji}(0) \leq \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i + \mathbf{d}_j - \mathbf{a}_i = \mathbf{d}_j.$$

Thus, $\mathbf{d} = \mathbf{a} \boxtimes \mathbf{w}(0)$. □

Any further iteration of $\mathbf{w}$ beyond initializing $\mathbf{w}(0)$ essentially makes use only of information about the threshold T as demonstrated in the proof of the following theorem.

Theorem 3.
i. $\mathbf{w}(k) = \mathbf{w}(1)$ *for all* $k \geq 1$.
ii. $\mathbf{w}_{ji}(1) \neq \mathbf{w}_{ji}(0)$ *only if* $i \in \mathcal{N}(j)$ *and* $\mathbf{t}_{ji} = -\infty$.

Proof. Proof of i: It suffices to show that $\mathbf{w}_{ji}(k) = \mathbf{w}_{ji}(k - 1)$ for $k \geq 2$. If $i \notin \mathcal{N}(j)$, then $\mathbf{w}_{ji}(0) = -\infty$, and inspection of the learning rule shows that $\mathbf{w}_{ji}(k) = -\infty$ for all $k \geq 2$ (in fact, for $k \geq 0$) because $-\infty \lneq T$. So the claim is true for the case of $i \notin \mathcal{N}(j)$. So let $i \in \mathcal{N}(j)$. If $\mathbf{w}_{ji}(k - 1) \lneq T$, then $\hat{\mathbf{w}}_{ji}(k) = -\infty$, and clearly $\mathbf{w}_{ji}(k) = -\infty \leq \mathbf{w}_{ji}(k - 1)$. Therefore, only for

$k = 1$ is it possible that $-\infty \lesssim \mathbf{w}_{ji}(k-1) \leq T$, and so $\mathbf{w}_{ji}(k) = -\infty$ for $k \geq 1$. If $\mathbf{w}_{ji}(k-1) \geq T$ (and $i \in \mathcal{N}(j)$), then because $0 \in \{z : i+z \in \mathcal{N}(j+z)\}$,

$$\mathbf{w}_{ji}(k) = \bigwedge_{\{z:i+z\in\mathbf{X},j+z\in\mathbf{X}\}} \hat{\mathbf{w}}_{j+z,i+z}(k) = \bigwedge_{\{z:i+z\in\mathcal{N}(j+z)\}} \mathbf{w}_{j+z,i+z}(k-1) = \mathbf{w}_{ji}(k-1).$$

The latter equality holds because $\mathbf{w}_{ji}(k-1)$ is invariant for every $k \geq 1$. Recall that after Step 2, the weights $\mathbf{w}_{ji}(k)$ correspond to an invariant template.

Proof of ii. First we show that $i \notin \mathcal{N}(j) \Rightarrow \mathbf{w}_{ji}(1) = \mathbf{w}_{ji}(0)$. We have $i \notin \mathcal{N}(j)$ implies $\mathbf{w}_{ji}(0) = -\infty$ (by the learning rule). Because $-\infty \lesssim T$, $\hat{\mathbf{w}}_{ji}(1) = -\infty$. Because $0 \in \{z : i+z \in \mathcal{N}(j+z)\}$, $\mathbf{w}_{ji}(1) = -\infty$, hence $\mathbf{w}_{ji}(1) = \mathbf{w}_{ji}(0)$. Using the contrapositive statement of what we have just proved, we can say that $\mathbf{w}_{ji}(1) \neq \mathbf{w}_{ji}(0)$ implies that $i \in \mathcal{N}(j)$. It remains to show that $\mathbf{t}_{ji} \neq -\infty$ implies $\mathbf{w}_{ji}(1) = \mathbf{w}_{ji}(0)$. To this end, we note that $\mathbf{t}_{ji} \neq -\infty \Rightarrow i \in S_{-\infty}(\mathbf{t}_j) \subset \mathcal{N}(j)$, by definition and hypothesis, respectively. Hence, the learning rules give us

$$\mathbf{w}_{ji}(0) = \bigwedge_{\{z:i+z\in\mathcal{N}(j+z)\}} \mathbf{d}_{j+z} - \mathbf{a}_{i+z}.$$

By Lemma 1, $\mathbf{t}_{j+z,i+z} \leq \mathbf{d}_{j+z} - \mathbf{a}_{i+z}$ for all i and j, where z is such that $i+z \in \mathcal{N}(j+z)$. But $\mathbf{t}$ is invariant, so $\mathbf{t}_{ji} = \mathbf{t}_{j+z,i+z}$ for each of these z values. Therefore,

$$\mathbf{t}_{ji} \leq \bigwedge_{\{z:i+z\in\mathcal{N}(j+z)\}} \mathbf{d}_{j+z} - \mathbf{a}_{i+z} = \mathbf{w}(0).$$

Also, $\mathbf{t}_{ji} \neq -\infty$ implies $T \leq \mathbf{t}_{ji}$, so $T \leq \mathbf{w}_{ji}(0)$. According to the learning rule,

$$\mathbf{w}_{ji}(1) = \bigwedge_{\{z:i+z\in\mathbf{X},j+z\in\mathbf{X}\}} \hat{\mathbf{w}}_{j+z,i+z}(1).$$

Because $\mathbf{w}(0)$ is invariant, $T \leq \mathbf{w}_{j+z,i+z}(0)$ for all z such that $i+z \in \mathbf{X}$, and $j+z \in \mathbf{X}$. Therefore, for all such z, $\hat{\mathbf{w}}_{j+z,i+z}(1) = \mathbf{w}_{j+z,i+z}(0)$, and we have

$$\mathbf{w}_{ji}(1) = \bigwedge_{\{z:i+z\in\mathbf{X},j+z\in\mathbf{X}\}} \mathbf{w}_{j+z,i+z}(0) = \mathbf{w}_{ji}(0).$$

□

We can see from the proof of part (i) of Theorem 3 and by inspecting the learning rule that $\mathbf{w}_{ji}(1) \neq \mathbf{w}_{ji}(0)$ only if $\mathbf{w}_{ji}(1) = -\infty$ and $\mathbf{t}_{ji} = -\infty$. This implies that $\mathbf{w}_{ji}(0) \geq \mathbf{w}_{ji}(1) = -\infty = \mathbf{t}_{ji}$, for all i and j. Therefore $\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(1) \leq \mathbf{d}_j - \mathbf{a}_i$ for all i and j, and Theorem 1 tells us that $\mathbf{a} \boxtimes \mathbf{w}(1) = \mathbf{d}$. Thus, the weights reached by this network at iteration $k = 1$ will allow the network to reproduce $\mathbf{d}$ when $\mathbf{a}$ is input, and no further change of weights will occur past this iteration. Now, note that the weights $\mathbf{w}(0)$ will also reproduce $\mathbf{d}$ when $\mathbf{a}$ is input, by Theorem 2. However, it is not necessarily true that the initial weights $\mathbf{w}(0)$ don't change after another iteration is performed. This is evidenced by Theorem 3, which gives conditions under which $\mathbf{w}(1)$ is not equal to $\mathbf{w}(0)$. Theorem 3 says that all weights $\mathbf{w}_{ji}(0)$ will be the same as $\mathbf{w}_{ji}(1)$ except for those values of i where $i \in \mathcal{N}(j)$ and the true template value $\mathbf{t}_{ji}$ has value $-\infty$. Furthermore, $\mathbf{w}(1)$ is invariant and satisfies the property that $\mathbf{w}_{ji}(1) \geq T$ for $i \in S_{-\infty}(\mathbf{w}_j(1))$.

Often we have a larger data set than one pair of training images. Without too much extra work, we can use the results established by the previous algorithm to show convergence to a set of weights that recall the training data for a set of P training pairs. Let $\{\mathbf{a}^k\}_{k=0}^{P-1}$ be the ordered input data, and $\{\mathbf{d}^k\}_{k=0}^{P-1}$ be the corresponding ordered output, where $\mathbf{d}^k = \mathbf{a}^k \boxed{\vee} \mathbf{t}$, $k = 0, \ldots, P-1$, and $\mathbf{t}$ is an invariant template. The following algorithm arrives at weights $\mathbf{w}$, representing an invariant template, where $\mathbf{d}^k = \mathbf{a}^k \boxed{\vee} \mathbf{w}$ for $k = 0, \ldots, P-1$.

In this algorithm, we apply the training pair one at a time. The pair $(\mathbf{a}^0, \mathbf{d}^0)$ is used to initialize the weights as per Equation (4.2), resulting in weights $\mathbf{w}^0_{ji}(0)$. Then steps 1 and 2 are applied to these weights as in the first algorithm, resulting in weights $\mathbf{w}^0_{ji}(1)$. Then the next training pair $(\mathbf{a}^1, \mathbf{d}^1)$, is applied (with a slight modification in the initial part), followed by steps 1 and 2, producing weights $\mathbf{w}^1_{ji}(1)$. Then the pair $(\mathbf{a}^2, \mathbf{d}^2)$ is used to go through the three-step procedure, etc.

Specifically, for $k = 0$ we initialize the weights to

$$\mathbf{w}^0_{ji}(0) = \begin{cases} \displaystyle\bigwedge_{\{z: i+z \in \mathcal{N}(j+z)\}} \mathbf{d}^0_{j+z} - \mathbf{a}^0_{i+z} & \text{if } i \in \mathcal{N}(j) \\ -\infty & \text{if } i \notin \mathcal{N}(j) \end{cases},$$

$$\hat{\mathbf{w}}^0_{ji}(1) = \begin{cases} \mathbf{w}^0_{ji}(0) & \text{if } \mathbf{w}^0_{ji}(0) \geq T \\ -\infty & \text{if } \mathbf{w}^0_{ji}(0) \not\geq T \end{cases},$$

and

$$\mathbf{w}^0_{ji}(1) = \bigwedge_{\{z: i+z, j+z \in \mathbf{X}\}} \hat{\mathbf{w}}^0_{j+z, i+z}(1).$$

For $k \geq 1$, we have

$$\mathbf{w}^k_{ji}(0) = \begin{cases} \left(\displaystyle\bigwedge_{\{z: i+z \in \mathcal{N}(j+z)\}} \mathbf{d}^k_{j+z} - \mathbf{a}^k_{i+z} \right) \bigwedge \mathbf{w}^{k-1}_{ji}(1) & \text{if } i \in \mathcal{N}(j) \\ -\infty & \text{if } i \notin \mathcal{N}(j) \end{cases} \tag{4.3}$$

$$\hat{w}^k_{ji}(1) = \begin{cases} w^k_{ji}(0) & \text{if } w^k_{ji}(0) \geq T \\ -\infty & \text{if } w^k_{ji}(0) \not\geq T \end{cases},$$

and

$$\mathbf{w}^k_{ji}(1) = \bigwedge_{\{z: i+z, j+z \in \mathbf{X}\}} \hat{\mathbf{w}}^k_{j+z, i+z}.$$

It is Equation (4.3) that is a modification of Equation (4.2), which allows information from previous data sets to accumulate. Also, note that $\mathbf{w}^0_{ji}(1)$ is identical to $\mathbf{w}_{ji}(1)$ in the previous net for the single data set $(\mathbf{a}^0, \mathbf{d}^0) = (\mathbf{a}, \mathbf{d})$. It is clear that $\mathbf{w}_{ji}^k(1) \leq \mathbf{w}_{ji}^m(1)$ for $m \leq k$, and that $\mathbf{t}_{ji} \leq \mathbf{w}_{ji}^k(1)$ for all $k = 1, \ldots, P-1$. Also, $\mathbf{w}_{ji}^k(1) \leq \mathbf{d}_j^k - \mathbf{a}_i^k$ for all $k = 1, \ldots, P-1$. Combining these results, and applying Theorem 1, we have $\mathbf{a}^m \boxed{\vee} \mathbf{w}^k(1) = \mathbf{d}^m$ for $m \leq k$, and, in particular, $\mathbf{a}^k \boxed{\vee} \mathbf{w}^{P-1}(1) = \mathbf{d}^k$ for all $k = 1, \ldots, P-1$.

Example 6. Grayscale Dilation with Learning for Variant Templates. This network is a generalization of nets in Examples 4 and 5, and is a more general form of the network in [5]. The additive maximum transform that this net attempts to learn can be any arbitrary grayscale one, including a variant transform. For a given set of P training data pairs, the learning rule needs about three passes through the data set to guarantee perfect recall of the P training images.

As before, let $\{\mathbf{a}^k\}_{k=0}^{P-1}$ and $\{\mathbf{d}^k\}_{k=0}^{P-1}$, be ordered sets of P input images and output images, respectively, where $\mathbf{d}^k = \mathbf{a}^k \boxed{\vee} \mathbf{t}$ for all $k = 0, \ldots, P-1$, and where $\mathbf{t}$ is a template that is not presumed to be invariant. As in Examples 4 and 5, the domains of $\mathbf{a}$ and $\mathbf{d}$ need not be identical. The weights $\mathbf{w}_{ji}$ are trained in an iterative fashion where $\mathbf{w}_{ji}(k+1)$ is the estimate of $\mathbf{t}_{ji}$ that results from training on images $\mathbf{a}^0, \mathbf{a}^1, \ldots$, and $\mathbf{a}^k$, in that order.

Let $\mathbf{c}^k = \mathbf{a}^k \boxed{\vee} \mathbf{w}(k)$. For this learning rule, we will reuse training pairs, but always so as to maintain their original order. We will assume then that $\mathbf{a}^k = \mathbf{a}^{k \bmod P}$ and $\mathbf{d}^k = \mathbf{d}^{k \bmod P}$ for all $k = 0, 1, \ldots$. Note, however, that it is not generally true that $\mathbf{c}^k = \mathbf{c}^{k \bmod P}$. This net, as the ones in Examples 4 and 5, has a limited connection scheme; it is also assumed that $S_{-\infty}(\mathbf{t}_j) \subset \mathcal{N}(j)$ for each $j = 1, \ldots, M$. Initially the weights $\mathbf{w}(0)$ are randomly selected values from the set $\{h : h \geq T\}$, where T is a finite number satisfying $\mathbf{t}_{ji} \geq T$, $i \in S_{-\infty}(\mathbf{t}_j)$. Thus, $\mathbf{w}_{ji}(0) \geq T$ for $i \in \mathcal{N}(j)$, and $\mathbf{w}_{ji}(0) = -\infty$ otherwise. The table in Figure 8 summarizes the change of weight rules.

Rule #	Condition Satisfied	New Value
1	$\mathrm{w_{ji}(k)} \lesssim \mathrm{T}$	$\mathrm{w_{ji}(k+1)} = -\infty$
2	$\left(\mathrm{a_i^k} + \mathrm{w_{ji}(k)} - \mathrm{d_j^k}\right)\left(\mathrm{c_j^k} - \mathrm{d_j^k}\right) \gtrsim 0$	$\mathrm{w_{ji}(k+1)} = \mathrm{d_j^k} - \mathrm{a_i^k}$
3	$\left(\mathrm{a_i^k} + \mathrm{w_{ji}(k)} - \mathrm{d_j^k}\right)\left(\mathrm{c_j^k} - \mathrm{d_j^k}\right) \leqslant 0$	$\mathrm{w_{ji}(k+1)} = \mathrm{w_{ji}(k)}$

Figure 8. Table of rules for network in Example 6.

This learning rule is reminiscent of (and motivated by) learning rules for neural nets which utilize standard linear algebraic operations. The following lemmas will allow us to give these rules a different expression that is easier to work with.

Lemma 2. $\mathbf{w}_{ji}(k) \gtrsim \mathbf{d}_j^k - \mathbf{a}_i^k \Rightarrow \mathbf{c}_j^k \gtrsim \mathbf{d}_j^k$.

Proof. Fix $i \in \mathcal{N}(j)$.

$$\mathbf{c}_j^k = \bigvee_{m \in \mathcal{N}(j)} \mathbf{a}_m^k + \mathbf{w}_{jm}(\mathbf{k}) \geq \mathbf{a}_i^k + \mathbf{w}_{ji}(\mathbf{k}) \geq \mathbf{a}_i^k + \mathbf{d}_j^k - \mathbf{a}_i^k = \mathbf{d}_j^k.$$

□

Lemma 3. $\mathbf{w}_{ji}(k) = \mathbf{d}_j^k - \mathbf{a}_i^k \Rightarrow \mathbf{c}_j^k \geq \mathbf{d}_j^k$.

The proof of Lemma 3 is similar to the proof of Lemma 2.

If we assume that rule 1 is not used in computing $\mathbf{w}_{ji}(k+1)$, then we can use these lemmas and the learning rules to consider nine cases:

Conditions Satisfied		Action Taken
$c_j^k \geq d_j^k$	$w_{ji}(k) \geq d_j^k - a_i^k$	apply rule 2
$c_j^k \geq d_j^k$	$w_{ji}(k) = d_j^k - a_i^k$	apply rule 3
$c_j^k \geq d_j^k$	$w_{ji}(k) \leq d_j^k - a_i^k$	apply rule 3
$c_j^k = d_j^k$	$w_{ji}(k) \geq d_j^k - a_i^k$	impossible (lemma 2)
$c_j^k = d_j^k$	$w_{ji}(k) = d_j^k - a_i^k$	apply rule 3
$c_j^k = d_j^k$	$w_{ji}(k) \leq d_j^k - a_i^k$	apply rule 3
$c_j^k \leq d_j^k$	$w_{ji}(k) \geq d_j^k - a_i^k$	impossible (lemma 2)
$c_j^k \leq d_j^k$	$w_{ji}(k) = d_j^k - a_i^k$	impossible (lemma 3)
$c_j^k \leq d_j^k$	$w_{ji}(k) \leq d_j^k - a_i^k$	apply rule 2

From this table we can see that only the following three cases are possible, assuming that rule 1 is not to be used:

Case	Conditions Satisfied	Action Taken
1	$c_j^k - d_j^k \geqslant 0$ and $w_{ji}(k) \leqslant d_j^k - a_i^k$	apply Rule #3
2	$c_j^k - d_j^k \leq 0$; then $w_{ji}(k) \leq d_j^k - a_i^k$ must be true	apply Rule #2
3	$c_j^k - d_j^k \geq 0$ and $w_{ji}(k) \geq d_j^k - a_i^k$	apply Rule #2

With these results we can derive the following equivalent learning rules, which we use to prove the convergence results in the rest of this example:

Rule #	Conditions Satisfied	New Value
1	$w_{ji}(k) \leq T$	$w_{ji}(k+1) = -\infty$
2	if $c_j^k - d_j^k \leq 0$ or $w_{ji}(k) \geq d_j^k - a_i^k$	$w_{ji}(k+1) = d_j^k - a_i^k$
3	otherwise	$w_{ji}(k+1) = w_{ji}(k)$

Figure 9. Learning rules used for network in Example 6.

We will show that number of iterations of the learning rule necessary to guarantee convergence to a set of weights which gives the correct dilated output on all of the input images is $3P$-2. That is, given any sets $\{\mathbf{a}^k\}_{k=0}^{P-1}$ and $\{\mathbf{d}^k\}_{k=0}^{P-1}$ as described above, then

$$\mathbf{d}^k = \mathbf{a}^k \boxed{\vee}\, \mathbf{w}(3P-2) \quad \text{for all } k = 0, \ldots, P-1.$$

In fact, this result is sharp in that there exist sets of input and dilated output images $\{\mathbf{a}^k\}_{k=0}^{P-1}$ and $\{\mathbf{d}^k\}_{k=0}^{P-1}$ such that, for some $k \in \{0, \ldots, P-1\}$,

$$\mathbf{d}^k \neq \mathbf{a}^k \boxed{\vee}\, \mathbf{w}(3P-3).$$

The proof of this result will be made easier by first proving a series of lemmas.

Lemma 4. $\mathbf{c}_j^k \not\leq \mathbf{d}_j^k \Rightarrow \mathbf{w}_{ji}(k) \not\leq \mathbf{t}_{ji}$ *for some* $i \in \mathcal{N}(j)$.

Proof. If $\mathbf{w}_{ji}(k) \geq \mathbf{t}_{ji}$ for all $i \in \mathcal{N}(j)$, then

$$\mathbf{c}_j^k = \bigvee_{m \in \mathcal{N}(j)} \mathbf{a}_m^k + \mathbf{w}_{jm}(k) \geq \bigvee_{m \in \mathcal{N}(j)} \mathbf{a}_m^k + \mathbf{t}_{jm} = \mathbf{d}_j^k.$$

□

Lemma 5. $\mathbf{w}_{ji}(k) \geq \mathbf{d}_j^k - \mathbf{a}_i^k$ *for some* $i \in \mathcal{N}(j) \Leftrightarrow \mathbf{c}_j^k \geq \mathbf{d}_j^k$.

Proof. ($\Rightarrow$) This is immediate from Lemmas 2 and 3.

($\Leftarrow$) $\mathbf{c}_j^k = \bigvee_{m \in \mathcal{N}(j)} \mathbf{a}_m^k + \mathbf{w}_{jm}(\mathbf{k})$. Let $i \in \mathcal{N}(j)$ be such that $\mathbf{a}_i^k + \mathbf{w}_{ji}^k = \mathbf{c}_j^k$. Then $\mathbf{a}_i^k + \mathbf{w}_{ji}^k \geq \mathbf{d}_j^k$ and $\mathbf{w}_{ji}^k \geq \mathbf{d}_j^k - \mathbf{a}_i^k$. □

Lemma 6. $\mathbf{w}_{ji}(k) \not\geq \mathbf{d}_j^k - \mathbf{a}_i^k$ *for some* $i \in \mathcal{N}(j) \Leftrightarrow \mathbf{c}_j^k \not\geq \mathbf{d}_j^k$.

Proof. ($\Rightarrow$) Follows from Lemma 2.

($\Leftarrow$) Proof is similar to that of Lemma 5. □

Lemma 7. $\mathbf{w}_{ji}(k+1) \not\geq \mathbf{w}_{ji}(k) \Rightarrow \left(\mathbf{c}_j^k \not\leq \mathbf{d}_j^k \text{ and } \mathbf{w}_{ji}(k) < \mathbf{d}_j^k - \mathbf{a}_i^k\right)$.

Proof. According to learning rules 1, 2, and 3 (in Figure 9), and under our hypothesis, rule 2 is applied in computing $\mathbf{w}_{ji}(k+1)$. Given that $\mathbf{w}_{ji}(k+1) \not\geq \mathbf{w}_{ji}(k)$, it can't occur that rule 2 was applied due to the condition $\mathbf{w}_{ji}(k) \not\geq \mathbf{d}_j^k - \mathbf{a}_i^k$. So if rule 2 was applied it must be that $\mathbf{c}_j^k \not\leq \mathbf{d}_j^k$, in which case, by Lemma 5, we must have $\mathbf{w}_{ji}(k) \not\leq \mathbf{d}_j^k - \mathbf{a}_i^k$. □

Lemma 8. *For all* $k \geq 0$, *for all* i *and* j, $\mathbf{w}_{ji}(k+1) \neq \mathbf{w}_{ji}(k) \Rightarrow \mathbf{t}_{ji} \leq \mathbf{w}_{ji}(k+1)$.

Proof. $\mathbf{w}_{ji}(k+1) \neq \mathbf{w}_{ji}(k)$ implies that $\mathbf{w}_{ji}(k+1)$ results from either rule 1 or rule 2. If it is computed by rule 2, then $\mathbf{w}_{ji}(k+1) = \mathbf{d}_j^k - \mathbf{a}_i^k$, and Lemma 1 shows that $\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(k+1)$.

If $\mathbf{w}_{ji}(k+1)$ is computed by rule 1, then $\mathbf{w}_{ji}(k) \not\geq T$. If $i \notin \mathcal{N}(j)$ then $\mathbf{t}_{ji} = -\infty$, and so $\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(k+1)$. If $i \in \mathcal{N}(j)$, then $\mathbf{w}_{ji}(0) \geq T$. Let k' be the first iteration (strictly) greater than 0 for which $\mathbf{w}_{ji}(k') \not\geq T$. Inspection of the learning rules clearly shows that $\mathbf{w}_{ji}(k') = \mathbf{d}_j^{k'-1} - \mathbf{a}_i^{k'-1}$. But $\mathbf{d}_j^{k'-1} - \mathbf{a}_i^{k'-1} \not\geq T$, together with Lemma 1, implies that $\mathbf{t}_{ji} = -\infty$, and again we have $\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(k+1)$. □

Lemma 9. *For a given j, there is at most one integer k_0 such that $\mathbf{c}_j^{k_0} \not\geq \mathbf{d}_j^{k_0}$.*

Proof. Suppose that for some $k_0 \geq 0$, $\mathbf{c}_j^{k_0} \not\geq \mathbf{d}_j^{k_0}$. Then by the learning rules, for all $i \in \mathcal{N}(j)$ such that $\mathbf{w}_{ji}(k_0) \geq T$, $\mathbf{w}_{ji}(k_0+1) = \mathbf{d}_j^{k_0} - \mathbf{a}_i^{k_0} \geq \mathbf{t}_{ji}$ (because rule 2 and not rule 1 will be used for these i values). If $\{i : \mathbf{w}_{ji}(k_0) \geq T\}$ is not empty, then $\mathbf{c}_j^{k_0+1} \geq \mathbf{d}_j^{k_0+1}$ by Lemma 5 and the fact, just mentioned, that $\mathbf{w}_{ji}(k_0+1) = \mathbf{d}_j^{k_0} - \mathbf{a}_i^{k_0} \geq \mathbf{t}_{ji}$ for some $i \in \mathcal{N}(j)$. In fact, under the assumption that $\mathbf{c}_j^{k_0} \not\geq \mathbf{d}_j^{k_0}$ for some $k_0 \geq 0$, $\{i : \mathbf{w}_{ji}(k_0) \geq T\}$ is not empty. To see this, use Lemma 8 together with the initialization rule to show that for all j, $\{i : \mathbf{w}_{ji}(k_0) \geq T\}$ is empty only if $\mathbf{t}_{ji} = -\infty$ for all $i \in \mathcal{N}(j)$, in which case $\mathbf{d}_j^{k_0} = -\infty$, contradicting our assumption that $\mathbf{c}_j^{k_0} \not\geq \mathbf{d}_j^{k_0}$.

Let k_1 be the smallest integer (strictly) greater than k_0+1 such that $\mathbf{c}_j^{k_1} \not\geq \mathbf{d}_j^{k_1}$. Then by Lemma 4, $\mathbf{w}_{ji}(k_1) \not\geq \mathbf{t}_{ji}$ for some $i \in \mathcal{N}(j)$. By assumption, $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ for $k = k_0+1, \ldots, k_1 - 1$, so by Lemma 7, $\mathbf{w}_{ji}(k+1) \leq \mathbf{w}_{ji}(k)$ for $k = k_0+1, \ldots, k_1-1$ for all $i \in \mathcal{N}(j)$. By repeatedly applying Lemma 8 to the cases $k = k_0+1, \ldots, k_1-1$, we see that, for all $i \in \mathcal{N}(j)$, $\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(k_1) \leq \mathbf{w}_{ji}(k_0+1)$, contradicting our earlier claim that $\mathbf{w}_{ji}(k_1) \not\geq \mathbf{t}_{ji}$ for some $i \in \mathcal{N}(j)$. Therefore, k_1, as described, does not exist. □

It is useful to consider what happens to $\mathbf{w}_{ji}$ when $\mathbf{c}_j^{k_0} \not\geq \mathbf{d}_j^{k_0}$ for some j, for some k_0. In this case, $\mathbf{w}_{ji}(k_0+1) \geq \mathbf{t}_{ji}$ for all i (by rules 1 and 2, and Lemma 1), and inspection of the learning rules, together with Lemma 9, shows that if $\mathbf{w}_{ji}$ ever changes after iteration k_0+1 it can only happen by applying rule 2 $\left(\mathbf{w}_{ji}(k+1) = \mathbf{d}_j^k - \mathbf{a}_i^k\right)$ in the case $\mathbf{w}_{ji}(k) \not\leq \mathbf{d}_j^k - \mathbf{a}_i^k$. This gives us the following lemma:

Lemma 10. *If $\mathbf{c}_j^{k_0} \not\geq \mathbf{d}_j^{k_0}$ for some $k_0 \geq 0$, then for all $k \geq k_0+1$, and for all i, $\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(k+1) \leq \mathbf{w}_{ji}(k)$.*

Lemma 11. *If $\mathbf{c}_j^{k_0} \not\geq \mathbf{d}_j^{k_0}$ for some $k_0 \geq 0$, then for all m such that $k_0 \not\geq m$,*

$$\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(m) \leq \bigwedge_{k=k_0}^{m-1} (\mathbf{d}_j^k - \mathbf{a}_i^k) \quad \textit{for} \quad i \in \mathcal{N}(j)$$

$$\mathbf{t}_{ji} = \mathbf{w}_{ji}(m) = -\infty \quad \textit{for} \quad i \notin \mathcal{N}(j).$$

Proof. The first part of the conclusion follows from Lemma 10 and rules 1 and 2. The second part of the conclusion is clear from rule 1. □

Lemma 12. *If* $\mathbf{c}_j^{k_0} \not\leq \mathbf{d}_j^{k_0}$ *for some* $k_0 \geq 0$, *then for all* m *such that* $k_0 \not\leq m$, $\mathbf{d}_j^k = \bigvee_{i\in\mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{w}_{ji}(m)$ *for all* k *such that* $k_0 \leq k \leq m-1$.

Proof. Using Lemma 11 we get

$$\begin{aligned}\mathbf{d}_j^k = \bigvee_{i\in\mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{t}_{ji} &\leq \bigvee_{i\in\mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{w}_{ji}(m) = \bigvee_{i\in\mathcal{N}(j)} \mathbf{a}_i^k + \left(\bigwedge_{r=k_0}^{m-1} \mathbf{d}_j^r - \mathbf{a}_i^r\right)\\ &\leq \bigvee_{i\in\mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{d}_j^k - \mathbf{a}_i^k = \mathbf{d}_j^k.\end{aligned}$$

□

By applying Lemma 12 to the special case $m = k_0+P$, we get that for any j, $\mathbf{d}_j^k = \bigvee_{i\in\mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{w}_{ji}(k_0+P)$ for $k = k_0, \ldots, k_0+P-1$, but because $\mathbf{a}^k = \mathbf{a}^{k+P}$ and $\mathbf{d}^k = \mathbf{d}^{k+P}$ for all $k \geq 0$, this is to say that $\mathbf{w}_{ji}(k_0+P)$ computes the correct dilated output on all of the input images. Remember that k_0 is just the iteration such that, for a given j, $\mathbf{c}_j^{k_0} \not\leq \mathbf{d}_j^{k_0}$; k_0 might not exist, but, intuitively, we would like to show that for any j, k_0 must be less than some fixed value if k_0 exists, and that we only need to iterate P times beyond that value in order to assure convergence to a template which correctly dilates all P of the input images. Toward that end, we will now consider situations in which $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ for values of k over various ranges.

Lemma 13. *Let* $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ *for* $0 \leq k \leq P-1$. *Then for any* i:

(a) If $\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(0)$ *then* $\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(m) \leq \bigwedge_{k=0}^{m-1} \mathbf{d}_j^k - \mathbf{a}_i^k$ *for* $1 \leq m \leq P$.

(b) If $\mathbf{w}_{ji}(0) \not\leq \mathbf{t}_{ji}$, *then* $\mathbf{w}_{ji}(m) = \mathbf{w}_{ji}(0) \leq \bigwedge_{k=0}^{P-1} \mathbf{d}_j^k - \mathbf{a}_i^k$ *for* $0 \leq m \leq P$.

Proof. (a) Lemma 8 gives us the left inequality. To prove the right inequality, let m_0 be the smallest integer such that $\mathbf{w}_{ji}(m_0) \not\leq \mathbf{d}_j^k - \mathbf{a}_i^k$ for some k, $0 \leq k \not\leq m_0$. It is important to note that actually $0 \leq k \not\leq m_0 - 1$. To show this we will suppose that $k = m_0 - 1$ and arrive at a contradiction. $\mathbf{w}_{ji}(m_0) \not\leq \mathbf{d}_j^{m_0-1} - \mathbf{a}_i^{m_0-1}$ implies, by Lemma 1, that $\mathbf{w}_{ji}(m_0)$ was not computed by rule 1. Also, under our hypothesis that $\mathbf{c}_j^{m_0-1} \geq \mathbf{d}_j^{m_0-1}$, the fact that $\mathbf{w}_{ji}(m_0) \not\leq \mathbf{d}_j^{m_0-1} - \mathbf{a}_i^{m_0-1}$ implies rule 2 was not used to compute $\mathbf{w}_{ji}(m_0)$. But if rule 3 was used to compute $\mathbf{w}_{ji}(m_0)$, then $\mathbf{w}_{ji}(m_0) = \mathbf{w}_{ji}(m_0 - 1) \not\leq \mathbf{d}_j^{m_0-1} - \mathbf{a}_i^{m_0-1}$, implying that rule 2 was used to compute $\mathbf{w}_{ji}(m_0)$, and we have our contradiction. So the value of k such that $\mathbf{w}_{ji}(m_0) \gtrsim \mathbf{d}_j^k - \mathbf{a}_i^k$ is an element of $\{0, \ldots, m-2\}$. By the minimality of m_0, we have

$$\mathbf{w}_{ji}(m_0 - 1) \leq \bigwedge_{r=0}^{m_0-2} \mathbf{d}_j^r - \mathbf{a}_i^r \leq \mathbf{d}_j^k - \mathbf{a}_i^k \lesssim \mathbf{w}_{ji}(m_0).$$

But then $\mathbf{w}_{ji}(m_0) \gtrsim \mathbf{w}_{ji}(m_0 - 1)$, and Lemma 7 then implies that $\mathbf{c}_j^{m_0-1} \lesssim \mathbf{d}_j^{m_0-1}$, contrary to our hypothesis. Therefore no such m_0 exists.

(b) First we'll verify the equality. Suppose there exists some $m \in \{0, 1, \ldots, P\}$ such that $\mathbf{w}_{ji}(m) \neq \mathbf{w}_{ji}(0)$ and that m is minimal in this respect ($m \geq 1$). By Lemma 8, $\mathbf{t}_{ji} \leq \mathbf{w}_{ji}(m)$. Because $\mathbf{w}_{ji}(m-1) = \mathbf{w}_{ji}(0) \lesssim \mathbf{t}_{ji}$, we know that rule 2 was not used to compute $\mathbf{w}_{ji}(m)$, therefore rule 1 was used, but this implies $\mathbf{w}_{ji}(0) \lesssim T$, contradicting our initialization rule (because $i \in \mathcal{N}(j)$ implies $\mathbf{w}_{ji}(0) \geq T$). Therefore, m, as described, does not exist, and we get the equality.

The inequality in part (b) follows easily from Lemma 1 and the initialization rule. □

Lemma 14. *Let* $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ *for* $k = 0, \ldots, 2P-2$. *Then* $\mathbf{w}_{ji}(P+m) = \mathbf{w}_{ji}(P+1)$ *for* $m = 1, \ldots, P-1$ *for all* i, *and* $\mathbf{w}_{ji}(P+1) \neq \mathbf{w}_{ji}(P)$ *only if* $\mathbf{w}_{ji}(P+1)$ *results from applying rule 1.*

Proof. Let m' be the smallest element in $\{1, \ldots, P-1\}$ such that $\mathbf{w}_{ji}(P+m') \neq \mathbf{w}_{ji}(P)$. Rule 2 was not used to compute $\mathbf{w}_{ji}(P+m')$: to see this, note that Lemma 13 implies $\mathbf{w}_{ji}(P) \leq \bigwedge_{k=0}^{P-1} \mathbf{d}_j^k - \mathbf{a}_i^k$, and this fact, together with our hypothesis that $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ for $k = 0, \ldots, 2P-2$, shows that the conditions needed for applying rule 2 to compute $\mathbf{w}_{ji}(P+m')$ does not hold. Therefore rule 1 was used to compute $\mathbf{w}_{ji}(P+m')$ in the case that $-\infty \lesssim \mathbf{w}_{ji}(P+m'-1) \lesssim T$. But this implies that $m' = 1$. To prove this, note that by the definition of m', $\mathbf{w}_{ji}(P+m'-1) = \mathbf{w}_{ji}(P)$. But it is clear from rule 1 that $\mathbf{w}_{ji}(P+m'-1) \neq \mathbf{w}_{ji}(P+m'-2)$, so $\mathbf{w}_{ji}(P+m'-2) \neq \mathbf{w}_{ji}(P)$. Now by the definition of m', and because $m'-2 \lesssim m'$, we must have $1 \leq m' \leq 2$. But $m' = 2$ is impossible because $\mathbf{w}_{ji}(P+m'-2) \neq \mathbf{w}_{ji}(P)$, so $m' = 1$.

Let m'' be the smallest integer in $\{2, \ldots, P-1\}$ such that $\mathbf{w}_{ji}(P+m'') \neq \mathbf{w}_{ji}(P+1)$ for any i. The previous argument shows that m'' exists only if $\mathbf{w}_{ji}(P+1)$ actually differs from $\mathbf{w}_{ji}(P)$ due to the application of rule 1. This implies that $\mathbf{w}_{ji}(P+m'')$ is computed by rule 2 because $\mathbf{w}_{jh}(P+1) \geq T$ for $h \in S_{-\infty}(\mathbf{w}_j(P+1))$. But the fact that

$$\mathbf{w}_{ji}(P+m''-1) = \mathbf{w}_{ji}(P+1) \leq \bigwedge_{k=0}^{P-1} \mathbf{d}_j^k - \mathbf{a}_i^k$$

together with our hypothesis implies that rule 2 is not used to compute $\mathbf{w}_{ji}(P+m'')$, and we have a contradiction, so m'' does not exist. □

Lemma 15. *If* $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ *for* $k = 0, \ldots, 2P-2$, *then* $\mathbf{c}_j^m = \mathbf{d}_j^m$ *for* $m = P, \ldots, 2P-1$.

Proof. Suppose $\mathbf{c}_j^m \gneq \mathbf{d}_j^m$ for some $m \in \{P, \ldots, 2P-1\}$. Then by Lemma 6, we have $\mathbf{w}_{ji}(m) \gneq \mathbf{d}_j^m - \mathbf{a}_i^m$ for this m, for some i, which is false by Lemmas 13 and 14, as shown by the following equation:

$$\underbrace{\mathbf{w}_{ji}(2P-1) = \cdots = \mathbf{w}_{ji}(P+1) \leq \mathbf{w}_{ji}(P)}_{\text{Lemma14}} \leq \underbrace{\bigwedge_{r=0}^{P-1} \mathbf{d}_j^r - \mathbf{a}_i^r = \bigwedge_{r=P}^{2P-1} \mathbf{d}_j^r - \mathbf{a}_i^r}_{\text{Lemma13}}.$$

□

Lemma 16. *If* $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ *for* $k = 0, \ldots, 2P-2$, *then* $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ *for all* $k \geq 0$. *In particular,* $\mathbf{c}_j^k = \mathbf{d}_j^k$ *for* $k \geq 2P-1$.

Proof. Lemma 15 says that, under the hypothesis, $\mathbf{w}_j(2P-1)$ (which equals $\mathbf{w}_j(P+1)$) will produce the correct value $\mathbf{d}_j^k$ for $k = P, \ldots, 2P-1$. But this is to say it works for all of the images, so $\mathbf{w}_{ji}(k)$ will never change for $k \geq 2P-1$ as a result of rule 2. Also, because $\mathbf{w}_j(2P-1) \geq T$ on its support, $\mathbf{w}_{ji}(k)$ will never change as a result of rule 1. □

Lemma 17. *If* $\mathbf{c}_j^k \lneq \mathbf{d}_j^k$ *for some* $k \geq 0$ *then* $k \in \{0, \ldots, 2P-2\}$.

Proof. Suppose $\mathbf{c}_j^k \lneq \mathbf{d}_j^k$ for some $k \geq 0$. By Lemma 9, k is unique. If $k \notin \{0, \ldots, 2P-2\}$ then by Lemma 16, $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ for all $k \geq 0$, contradicting our supposition. □

Lemma 18. *If* $\mathbf{c}_j^{k_0} \lneq \mathbf{d}_j^{k_0}$ *for some* $k_0 \geq 0$ *then* $\mathbf{d}_j^k = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{w}_{ji}(3P-2)$ *for all* $k = 0, \ldots, P-1$.

Proof. By Lemma 17, $0 \leq k_0 \leq 2P-2$. By Lemma 12, $\mathbf{d}_j^k = \bigvee \mathbf{a}_i^k + \mathbf{w}_{ji}(3P-2)$ for all k such that $k_0 \leq k \leq 3P-3$. But $\{\mathbf{a}^k\}_{k_0}^{3P-3} = \{\mathbf{a}^k\}_{2P-2}^{3P-3} = \{\mathbf{a}^k\}_0^{P-1}$ and, similarly, $\{\mathbf{d}^k\}_{k_0}^{3P-3} = \{\mathbf{d}^k\}_{2P-2}^{3P-3} = \{\mathbf{d}^k\}_0^{P-1}$. □

Lemma 19. *If* $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ *for all* $k \geq 0$, *then* $\mathbf{d}_j^k = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{w}_{ji}(3P-2)$ *for all* $k = 0, \ldots, P-1$.

Proof. By the proof of Lemma 16, $\mathbf{w}_{ji}(3P-2) = \mathbf{w}_{ji}(k)$ for $k \geq 2P-1$. Also Lemma 16 says that $\mathbf{d}_j^k = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{w}_{ji}(k)$ $(= \mathbf{c}_j^k)$ for $k \geq 2P-1$. Combining results we get $\mathbf{d}_j^k = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{w}_{ji}(3P-2)$ for $2P-1 \leq k \leq 3P-2$, but this is to say that the same holds true for $0 \leq k \leq P-1$. □

Theorem 4. $\mathbf{d}^k = \mathbf{a}^k \boxtimes \mathbf{w}(3P-2)$ *for all* $k = 0, \ldots, P-1$.

Proof. For any j, Lemmas 18 and 19 together show that (whether or not $\mathbf{c}_j^k \lesssim \mathbf{d}_j^k$ for all $k \geq 0$), $\mathbf{d}_j^k = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{w}_{ji}(3P-2)$ for all $k = 0, \ldots, P-1$. This holds for all j, so the conclusion follows. □

In fact, the proof of Lemma 19 shows that if $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ for all $k \geq 0$ for all j, then $\mathbf{d}^k = \mathbf{a}^k \boxtimes \mathbf{w}(P)$ because $\mathbf{w}_{ji}(P) = \mathbf{w}_{ji}(3P-2)$ in this case. However, in actual practice we don't know that $\mathbf{c}_j^k \geq \mathbf{d}_j^k$ for all $k \geq 0$ for all j until we have computed $\mathbf{w}(2P-2)$ and $\mathbf{c}^{2P-2}$; Lemma 16 tells us this. So for a given j, we can perform the training of $\mathbf{w}_{ji}$ for all i for only P iterations beyond the time when $\mathbf{c}_j^k \lesssim \mathbf{d}_j^k$, if there is such a time k such that $k \leq 2P-2$ (see Lemmas 12 and 15). Otherwise, only update $\mathbf{w}_{ji}$ for $2P-2$ iterations. Of course this general strategy can result in our needing to compute $\mathbf{w}_{ji}(3P-2)$ in some cases.

Our result that $\mathbf{d}^k = \mathbf{a}^k \boxtimes \mathbf{w}(3P-2)$ for $k = 0, \ldots, P-1$ is sharp in the sense that $\mathbf{d}^k \neq \mathbf{a}^k \boxtimes \mathbf{w}(3P-3)$ for some k is possible, as shown by the following example:

In this example we let $P = 3$ and $j = \mathbf{y}_3$, and $\mathbf{X} = \mathbf{Y}$. Recall that each template $\mathbf{t} \in (\mathbb{R}_{\pm\infty}^{\mathbf{X}})^{\mathbf{Y}}$ consists of a collection of images on $\mathbf{X}$: $\mathbf{t}_{\mathbf{y}_1}, \mathbf{t}_{\mathbf{y}_2}, \mathbf{t}_{\mathbf{y}_3}, \mathbf{t}_{\mathbf{y}_4}$. Thus we will be looking at the (possibly variant) template $\mathbf{t}$ at location $\mathbf{y}_3$, where $\mathbf{X}$ is given below. For notational purposes we denote $\mathbf{t}_{\mathbf{y}_3}$ by $\mathbf{t}_3$, $\mathbf{w}_{\mathbf{y}_3}$ by $\mathbf{w}_3$, $\mathbf{d}_{\mathbf{y}_3}$ by $\mathbf{d}_3$, and $\mathbf{c}_{\mathbf{y}_3}$ by $\mathbf{c}_3$. The neighborhood for each $j = \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4$, is $\mathcal{N}(j) = \mathbf{X}$. Each input image $\mathbf{a}^0, \mathbf{a}^1$, and $\mathbf{a}^2$, and the corresponding output images $\mathbf{d}^0$, $\mathbf{d}^1$, and $\mathbf{d}^2$ at location $\mathbf{y}_3$, are shown below. The $*$ values in the $\mathbf{d}$ images represent values not needed to perform the calculations. The weights are randomly initialized to values as shown in Figure 10, where the threshold value T is not relevant as it is not used in these calculations. After $3P - 3 = 3{*}3 - 3 = 6$ iterations, the weights used to calculate the output location $\mathbf{y}_3$ do not recall the correct value for $\mathbf{d}_3^6 = \mathbf{d}_3^0$. However, at the next iteration, $3P - 2 = 7$, the weights $\mathbf{w}_{3i}(7)$ correctly calculate $\mathbf{d}_3^k$ for $k = 0, 1$, and 2. Hence, calculations show that $3 = \mathbf{d}_3^6 \neq \mathbf{c}_3^6 = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i^6 + \mathbf{w}_{3i}(6) = 4$ but that $\mathbf{d}_3^k = \bigvee_{i \in \mathcal{N}(j)} \mathbf{a}_i^k + \mathbf{w}_{3i}(7)$ for $k = 0, 1, 2$.

We should note that although $\mathbf{w}(3P-2)$ computes correct dilated outputs for all of the input images, it may not satisfy the condition that $\mathbf{w}(3P-2) \geq T$ for $i \in S_{-\infty}(\mathbf{w}_j(3P-2))$. It may be necessary to apply rule 1 to get weights $\mathbf{w}(3P-1)$, which will satisfy the condition (and which also produces correct dilations).

Intuitively, the proof of the convergence results can be viewed as follows. We are looking for the worst case which gives us the maximum number of iterations needed to provide convergence. What occurs in the worst case is that during the first pass through the data, those values of $\mathbf{w}_{ji}$ that are clearly too large ($> \mathbf{d}_j - \mathbf{a}_i$), and which are therefore concealing other values $\mathbf{w}_{ji}$ which are too small ($< \mathbf{t}_{ji}$), are reduced to the smallest values possible (by rule 2). The results of the theorems show that if any j in $\mathbf{X}$ and for iteration k, if $\mathbf{c}_j^k \lesssim \mathbf{d}_j^k$, then we need only apply the next $P-1$ training images once more to arrive at weights which recall all training images correctly. Since if any weights change

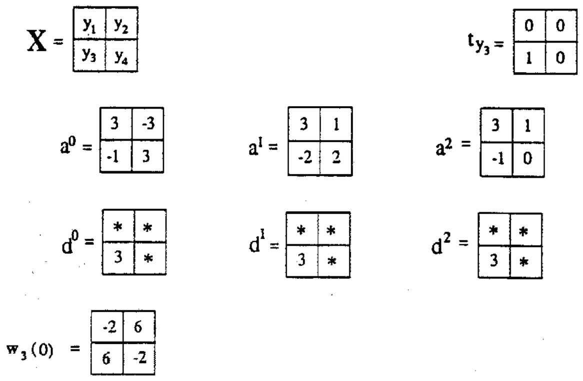
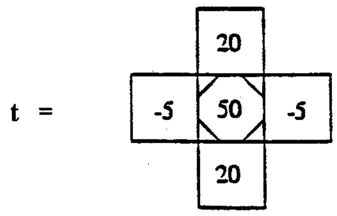


Figure 10. Example showing that $3P - 2$ is sharp.

during the first pass, then those weights satisfy $\mathbf{w}_{ji}(m) \leq \mathbf{w}_{ji}(k)$, for $k \leq m$, we then have the possibility that $\mathbf{c}_j \not\lesssim \mathbf{d}_j$ will occur during the second pass through the data (even though it didn't on the first pass). Thus, the situation $\mathbf{c}_j \not\lesssim \mathbf{d}_j$ could occur for any of the P images during the second pass except the *last* image $\mathbf{a}^{P-1}$. This is because (if $\mathbf{c}_j \not\lesssim \mathbf{d}_j$ hasn't occurred by that time) the last time that weights were changed (or not changed), they were changed (or not changed) so that they would "work" on this very image $\mathbf{a}^{P-1}$. But $P - 1$ images after $\mathbf{a}^{P-1}$ takes us up to image $\mathbf{a}^{2P-1}$. So $\mathbf{c}_j \not\lesssim \mathbf{d}_j$ can't happen after iteration $2P - 2$. And if $\mathbf{c}_j \not\lesssim \mathbf{d}_j$ does happen as late as iteration $2P - 2$, then this image data, $\mathbf{a}^{2P-2}$, is itself used (rule 2) to "kick up" the weights. Going through the rest of the next $P - 1$ data pairs, for what is essentially the third pass, actually only requires us to use data up to $3P - 3$ (and not, say, $3P - 1$). After training on the data $\mathbf{a}^{3P-3}$, we have weights $\mathbf{w}(3P - 2)$, which correctly recall all training data.

We next present an implementation on data for the network given in Example 6. This network was run on a set of 13 training pairs consisting of 13 natural grayscale images (with a range of pixel values of 0 to 255) and their dilations computed by the template **t** shown below.



Although the template **t** happens to be invariant, this fact is not used during the training of the net. The resulting dilation template **w** was then used to dilate a test image. Comparisons were made between the templates **t** and **w** and also between dilations of the test image by **t** and **w**. The test image was also a natural grayscale image and was not used for training. The weights **w**(0) were initialized as follows: for each j and for each i in the 3 by 3 neighborhood centered at j, $\mathbf{w}_{ji}(0)$ was randomly assigned a value between -5 and 80. The results of this implementation are in Figure 11 and displayed in Figure 12; their presentation is similar to the one given in Example 4.

number of training images applied	number of pixels out of the 36100 in which t and w differ	number of pixels out of the 20224 pixels in the support of t in which t and w differ	column 2 calculated as a percentage of the 36100 possible	ave. absolute difference between t and w on the support of t
0	35798	19922	99%	29.8
13	24554	11402	68%	14.3
26	23320	10420	65%	12.2
37 = 3P-2	23299	10399	65%	12.1

(a)

number of training images applied	number of pixels out of the 4096 in which the net dilation and the ideal dilation differ	ave. absolute difference between the net dilation and the ideal dilation	column 3 as a percentage of the gray value range of 255
0	4079	27.9	10.9%
13	1728	7.7	3.0%
26	1430	6.2	2.4%
37 = 3P-2	1413	6.1	2.4%

(b)

Figure 11. Results of applying the grayscale dilation learning algorithm. (a) Measures of the difference between weight values calculated by the net and ideal weight values. (b) Measures of the difference between dilations of the test image by the net and ideal weight values.

We should note that, although the template **w**(37) does seem to differ considerably from the template **t**, it does correctly dilate the 13 training images. Also, the absolute difference between the dilations of the test image by **w**(37)

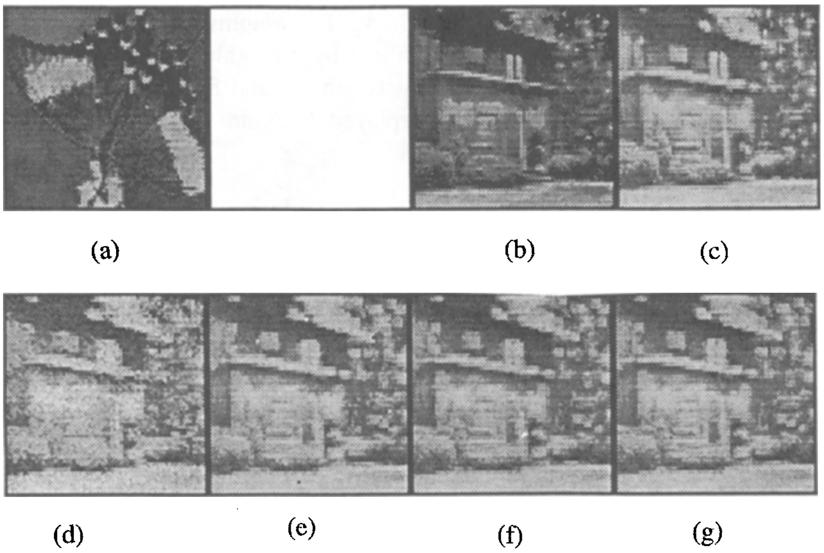


Figure 12. (a) One of the 13 training input images. (b) Test image. (c) Dilation of the test image by **t**. (d) Dilation of the test image by **w**(0). (e) Dilation of the test image by **w**(13). (f) Dilation of the test image by **w**(26). (g) Dilation of the test image by **w**(37) = **w**(3P − 2).

and **t** is smaller than the difference between the templates. This is not surprising considering the mathematical properties of the dilation operation.

Some general remarks.

The first three examples represent a straightforward implementation of the additive maximum rule and as such provide nothing new. The last three Examples, 4, 5 and 6, each have a learning rule. It is the neighborhood $\mathcal{N}(j)$ from which the approximation of the template and its weights is made. Due to the nature of the learning rules and the fact that $\mathcal{N}(j)$ always at least contains the support of the template **t**, the weights determined by the network may be greater than or equal to the values that the ideal template **t** has. In particular, on the pixels lying outside the support of **t** but still in $\mathcal{N}(j)$, the weights $\mathbf{w}_{ji}(k)$ may be finite while the template values are $-\infty$. Nonetheless, the weights to which these networks converge will still allow perfect recall of the training data. In this sense, the learning rules presented here allow a very good approximation of the template **t** to be recovered. But just as in the case for classical neural networks, these morphology nets cannot generalize to include recovery of information which is not present in the data. Different data sets may converge to different sets of weights, especially for the network of Example 6 which recovers a variant template. Also, the convergence results produce weights based on a particular ordering of the data. A different ordering would most likely produce convergence at a different iteration number, and perhaps slightly different weight values. Thus this network, like many classical ones, will most likely produce data-dependent results.

Practical applications of these networks include the modeling of a template or structuring element which produced input-output pairs having a nonlinear underlying transformation similar to the additive maximum. A demonstration of this type of application to image data was shown in Example 4. Other networks remain to be developed which use the additive maximum but perhaps a different activation function, and a cascade of the additive maximum and minimum. This would allow the modeling of a more complex nonlinear process. The mathematical morphology opening operation is an example of a cascade of operations; see [5] for a learning rule for that particular network. Convergence for that network has not been proven, although all experimental results on data have produced a set of converged weights.

5. Conclusions

The research presented in this paper has laid the fundamental theoretical foundations of the theory of artificial morphology neural networks. The origins of morphology neural nets, which lie in image algebra, have intrinsic ties to the image processing tool called mathematical morphology, and, more generally, to the algebra of minimax matrix theory. Several applications in image processing have been presented, including three networks (Examples 4, 5, and 6) which have learning rules. Convergence of these learning rules to a set of weights has been proven mathematically, and the weight values allow perfect recall of any of the training data. These results may be very useful for solving additional image processing problems; investigation into this area of research is currently being pursued.

References

[1] G. Birkhoff and J. Lipson, Heterogeneous algebras, *J. Combinatorial Theory*, vol. 8, pp. 115–133, 1970.

[2] R. Cuninghame-Green, *Minimax Algebra: Lecture Notes in Economics and Mathematical Systems* **166**, New York: Springer-Verlag, 1979.

[3] J. Davidson, *Lattices Structures in the Image Algebra and Applications to Image Processing*, Ph.D. thesis, Department of Mathematics, University of Florida, Gainesville, FL, August 1989.

[4] J. Davidson and K. Sun, Template learning in morphological neural nets, *SPIE - Proc. Soc. Photo-Optical Instr. Eng.*, vol. 1568, pp. 176–187, July 1991.

[5] J. Davidson and K. Sun, Opening template learning in morphological neural nets, Heuristics, *The Journal of Knowledge Engineering*, vol. 5 No. 2, pp. 28–36, Summer 1992.
[6] H. Hadwiger, Minkowskische addition und subtraktion beliebiger punktmengen und die theoreme von Erhard Schmidt, *Mathematische Zeitschrift*, vol. 53, pp. 210–218, 1950.
[7] R. P. Lippmann, An introduction to computing with neural nets, *IEEE Magazine on Acoust. Speech Signal Proc.*, vol. ASSP-4, pp. 4–22, 1987.
[8] H. Minkowski, Volumen und oberfläche, *Mathematische Annalen*, vol. 57, pp. 447–495, 1903.
[9] G. Ritter and J. Davidson, Recursion and feedback in image algebra, *SPIE 19th AIPR Wkshp Image Understanding in the 90's*, vol. 1406, pp. 74–86, October 1990.
[10] G. Ritter, D. Li, and J. Wilson, Image algebra and its relationship to neural networks, *Proc. 1989 SPIE Tech. Symp. Optics, Elec.-Opt. Sensors*, March 1989.
[11] G. Ritter, M. Shrader-Frechette, and J. Wilson, Image algebra: A rigorous and translucent way of expressing all image processing operations, *Proc. 1987 SPIE Tech. Symp. Optics, Elec.-Opt. Sensors*, pp. 116–121, May 1987.
[12] G. Ritter and J. Wilson, Image algebra: A unified approach to image processing, *Proc. SPIE Med. Imag. Conf.*, vol. 767, pp. 338–345, February 1987.
[13] G. Ritter, J. Wilson, and J. Davidson, Image algebra: An overview, *Comp. Vis., Graphics, Image Proc.*, vol. 49, pp. 297–331, March 1990.
[14] J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, London, 1982.
[15] S. Sternberg, Grayscale morphology, *Comp. Vis., Graph, Image Proc.*, vol. 35, pp. 333–355, 1986.
[16] S. Wilson, Morphological networks, *Proc. 1989 SPIE Vis. Comm. Image Proc.*, IV, November 1989.