DIGIT RECOGNITION USING KOHONEN NEURAL NETWORK

A Special Project Presented
In Partial Fulfillment of the
Requirements for the Course
ECE 178

Submitted to:

Engr. Cristina P. Dadula

Submitted by:

Karen Joyce A. Ganacias
Lhea D. Mahinay
Cheryl L. Miquiabas
Kathleen Joy S. Silagan

March 2012

ABSTRACT

Pattern recognition application has shown to be useful in accurately detecting and recognizing words. Typically, pattern recognition is an area of research which deals with neural approach. Optical character recognition is one of the pattern recognition applications that make use of neural networks. One example of a neural network is the Kohonen neural network. It is a neural network that does not use any sort of activation function and does not use any sort of a bias weight. When a pattern is presented to a Kohonen network one of the output neurons is selected as a "winner". This "winning" neuron is the output from the Kohonen network. This paper presents a simple learning rule for recognition of mouse dragged digit on the computer screen using artificial neural network. A Kohonen self organization map for pattern classification was used which employs unsupervised learning algorithm. Digits are accepted by allowing the user to draw into a high resolution box. Unfortunately, this resolution is too high to directly present to a neural network. To alleviate this problem, cropping and down sampling technique was used. By using these two techniques, the image can be transformed in to a second image that is of much lower resolution. Once the image has been entered, it must be cropped. Once the image has been cropped, it must be downsampled. The resulting downsampled image will then be fed to either the training or recognition process of the neural network. The Kohonen neural network used in this project has a number of input neurons that is equal to the number of pixels in the downsampled image and has a number of output neurons equal to the number of digits that the application is to recognize. Neural networks provide an efficient way of performing certain operations that would otherwise be very difficult. Any form of data that can have patterns is a candidate for a neural network solution.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Background of the Study

The word of recognition plays an important role in our lives. It is a basic property of all human beings. When a person sees an object, he or she first gathers all information about the object and compares its properties and behaviors with the existing knowledge stored in the mind. If we find a proper match, we recognize it. This recognition concept is simple and familiar to everybody in the real world environment, but in the world of artificial intelligence the recognition is done through machine that is called pattern recognition.

Pattern recognition is the research area that studies operation and design of system to recognize patterns whether it is in text, numeric or image. For several years, various applications have been developed to assist and solve certain problem such as face recognition and character recognition. Character recognition application has shown to be useful in accurately detecting and recognizing words. Typically, pattern recognition is an area research which deals with neural approach. The neural approach applies biological concepts to machines to recognize patterns. The outcome of this effort is invention of Artificial Neural Network (ANN). The advantages of neural networks are their adaptive-learning, self-organizing and fault tolerance capabilities. For these outstanding capabilities, neural networks are suitable used for pattern recognition.

One example of a neural network is the Kohonen neural network. It is a neural network that does not use any sort of activation function and does not use any sort of a bias weight. Output from the Kohonen neural network does not consist of the output of several neurons.

When a pattern is presented to a Kohonen network one of the output neurons is selected as a "winner". This "winning" neuron is the output from the Kohonen network.

Optical character recognition is one of the pattern recognition applications that make use of neural networks. OCR programs are capable of reading printed text. This could be text that was scanned in from a document, or hand written text that was drawn to a hand-held device, such as a Personal Digital Assistant (PDA). They are used widely in many industries.

This paper discusses optical digit recognition application using Kohonen neural network in recognizing handwritten digits from 0 to 9.

1.2 Objectives of the Study

This project aims to develop an optical digit recognition program in Java using Kohonen neural network to recognize handwritten digits from 0 to 9.

1.3 Significance of the Study

The project will show the use of artificial neural network that will simplify the development of an optical character recognition application, while achieving highest quality of recognition and good performance. Such a program could be useful in demonstrating how a Kohonen neural network is used in recognizing handwritten digits. This project will also provide a reference for future studies on developing character recognition applications recognizing different patterns or using other neural networks that employs unsupervised learning.

1.4 Scope and Limitation

The project will be focusing in developing an optical digit recognizer using the neural network - the Kohonen Network. The character to be recognized will be the decimal digits from 0 to 9. The project will be able to create an easy way in which the user can draw characters on the screen and then let the program interpret or recognize the drawn by the user. One limitation of the application is that the network is limited to recognizing digits one at a time. Thus the user can only draw one digit in the drawing area for every recognition process. Another limitation of the program is that only one drawing can be defined for every digit during training. This means that only one sample input pattern can be assigned and trained to the neural network for every digit. The sample input patterns also depends on the handwriting of the user. Thus, the more clear and accurate the handwriting of the user is, the more accurate the training and recognition process becomes.

1.5 Definition of Key Terms

Artificial Neural Network - refers to a network or circuit of interconnecting artificial neurons or programming constructs that mimic the properties of biological neurons

Downsampling - the process of reducing the sampling rate of a signal, usually done to reduce the data rate or the size of the data

Image - an artifact, for example a two-dimensional picture that has a similar appearance to some subject - usually a physical object or a person

Input neuron - performs no action on the values other than distributing them to the neurons in the hidden and output layers

Kohonen network - a neural network that does not use any sort of activation function and does not use any sort of a bias weight

Neuron - an electrically excitable cell that processes and transmits information by electrical and chemical signaling and are connected to each other to form neural networks

Optical character recognition - the mechanical or electronic translation of scanned images of handwritten, typewritten or printed text into machine-encoded text

Output neuron - receives values from all of the input neurons (including the bias) and all of the hidden layer neurons

1.6 Theoretical Framework

Optical Character Recognition

Optical character recognition (OCR) is the mechanical or electronic translation of scanned images of handwritten, typewritten or printed text into machine-encoded text. It is widely used to convert books and documents into electronic files, to computerize a record-keeping system in an office, or to publish the text on a website. OCR makes it possible to edit the text, search for a word or phrase, store it more compactly, display or print a copy free of scanning artifacts, and apply techniques such as machine translation, text-to-speech and text mining to it. OCR systems require calibration to read a specific font; early versions needed to be programmed with images of each character, and worked on one font at a time [1].

Kohonen Neural Network

The Kohonen Self-Organizing Map (SOM) designed by Tuevo Kohonen is a variation of the traditional Artificial Neural Network. It is a third generation neural network, meaning that

many of its functional characteristics are thought to mirror those found in biological fact. An SOM consists of a collection of nodes of neurons that are each connected to every other node and each node has associated with it a set of input weights w. The SOM also has associated with it a metric for determining which nodes are in the neighborhood N of a given node.

When the network is presented with a vector x_i at its input, it computes the neural response s_j of the node j using the formula:

$$S_j = w_j * x_i$$

Normalize both w_j and x_i before computing the dot product, s_j, and refer to the node that produces the largest value of s as node k. Since the dot product of the normalized w_k and x_i vectors is the cosine of the angle between them, we can conclude that the winning node is the one with the weight vector closest to the input vector in its spatial orientation. We can then say that node k giving the largest s is closest to recognizing the input vector. We allow the nodes to learn by applying a Δw to their weights using the formula:

$$\Delta w_k = \alpha (x_i - w_k)$$

Where α is a constant in the range [0,1] called the learning constant. The learning process is applied to the maximum response neuron and neurons in its defined neighborhood. This training process can be described by the following algorithm:

1. A cycle: for every input vector x_i

    [a] Apply vector input to the network and evaluate the dot products of the normalized weights on each node and a normalized input vector. Call these dot products s.

    [b] Find the node k with the maximal response s_k.

    [c] Train node k, and all the nodes in some neighborhood of k, according to the learning equation above.

[d] Calculate a running average of the angular distance between the values of w_k and their associated input vectors.

[e] Decrease the learning rate.

2. After every M cycles, called the period, decrease the size of the neighborhood N.

3. Repeat steps 1-2 for some finite period of time or until the average angular distance. One advantage to this scheme is that the system is quite tolerant to changing conditions and inputs.

The Kohonen network has two layers, an input layer and a Kohonen out layer. The input layer is a size determined by the user and much match the size of each row (pattern) in the input data file. A Kohonen feature map may be used by it self or as a layer of another neural network. A Kohonen layer is composed of neurons that compete with each other. The Kohonen SOM use winner-take-all strategy. Inputs are feed into each of the neurons in the Kohonen layer (from the input layer). Each neuron determines its output according to a weighted sum formula:

$$\text{Output} = \Sigma\, w_{ij}\, x_j$$

The weights and the inputs are usually normalized which mean that the magnitude of the weight and input vectors are set equal to one. The neuron with the largest output is winner. The neuron has a final output of 1. All other neurons in the layer have an output of zero. Different input patterns end up with firing different wining neurons. Similar or identical input patterns classify to the same output neuron. Only winning neurons and their neighbor's par learns for a given input pattern [2].

CHAPTER 2

REVIEW OF RELATED LITERATURE

A study conducted by Marina Yusoff, Shuzlina Abdul Rahman, Sofianita Mutalib, Azlinah Mohamed entitled Kohonen Neural Network Performance in License Plate Number Identification aimed to develop a character recognition application for vehicle identification. Character recognition is valuable when it comes to the needs of identifying vehicle due to its potential to uniquely distinguish the correct vehicle that belongs to a particular person. Although there has been a number of commercialized software in the market, the identification of characters in plate number in Malaysia illustrates differences compared to Europian region. Malaysia uses different description for each type of character and different fonts. Thus, the research and the development of license plate recognition in Malaysia are necessary because of the lack of conventional ways to identify the vehicles. The application employed Kohonen Self Organising MAP (SOM) algorithm to recognize license plate number. Several stages have been performed in the development process. Preprocessing stages include image normalization, image segmentation, image enhancement, and binary segmentation are accomplished using MATLAB tool. The next stage is the unsupervised neural network architecture determination and finally the development of interface and evaluation [3].

Another study was conducted by Adnan Md. Shoeb Shatil entitled Bangla Optical Character Recognition Using Kohonen Network. This project was developed to recognize Bangla (National Language of Bangladesh) characters. The whole idea is converting text images into editable texts. In this study, the raw data are offline printed characters. These characters are collected from computer images and also scanned documents. Since no skew conversion and correction is

considered in processing or preprocessing stages, the documents are scanned with great care. After collecting the raw data, they are pre-processed with gray scale conversion and then black and white conversion. When the black and white images are obtained, only two types of data on that image are present. So it was considered as a binary file. The character area is represented by 1 and the rest of image area is represented with 0. In the next step, the whole image is grabbed; the binary data is taken and placed on a 25 x 25 pixel by pixel mapping procedures. Then a 625-bit long vector for each word or character is collected. This vector is then trained with Kohonen neural network considered as the classification stage. Thus, an optical character is recognized with Kohonen Network [2].

Another license plate recognition application was developed by Norfaeza binti Mat Noor in his study entitled License Plate Recognition Using Kohonen Neural Network Algorithm. This paper focuses on the development of character recognition for license plate numbers. License plate recognition is one of important techniques that can be used for the identification vehicles. It is useful in many applications such as entrance admission, security, parking control, traffic enforcement, and toll gate automation. In the development of this system, several stages were executed. The preprocessing is implemented by using MATLAB tool to preprocess the images to become an input to the network in binary form. For the recognition of the license plate number, the Kohonen self-organizing map is used to recognize the license plate number using Euclidean distance to determine best-matching unit and employed two dimensional Kohonen layer map. It is easily trained and has attractive properties such as topological ordering and good generalization. Experiments were performed to determine the network parameter aside from measuring the performance of Kohonen neural network. The test result of the prototype was shown with 78.57% accuracy [4].

A program was also developed by Jeff Heaton that recognizes the handwritten characters of the 26 letters of the Latin alphabet [5]. It also employed downsampling of the image where Adnan Md. Shoeb Shatil's study was based from.

The different studies and project conducted by different researchers are related to this study in that they also aimed to develop an application that recognizes characters using Kohonen Neural Network. Most of these related studies were performed and developed in MatLab. This study however aimed to develop a program that recognizes digits from 0-9 and was developed using Java as the programming language in Netbeans IDE 7.0.

CHAPTER 3

METHODOLOGY

3.1 Gathering Data

The team gathered the data necessary for the development of the application. Most of the related studies that the team acquired recognize characters in different alphabets. Thus, the team acquired their program and modified them to accomplish the aim of developing a program that will recognize digits from 0 to 9. The sample input patterns of the previous studies which contain the sample data of letters in the alphabet were modified to contain sample patterns for the digits 0 to 9. This sample data were used as the input patterns for the training of the neural network in the application. Also, the team acquired the user requirements based on what the team ponders the user will need in interacting with the application software in recognizing the digits he/she will draw. These covered the features of the user interface of the program.

3.2 Domain Analysis and Design

In order for the team to further understand how the application will behave, use cases were developed based on the requirements realized. Use case is a construct that helps analysts work with the users to determine system usage. It is an excellent tool for stimulating the users to talk about the system from their own viewpoints and enables design of tests for the application [6]. To visualize the use cases, a use case diagram showing the use cases were designed. Visualization allows the analysts to show the use cases and the actors of the use cases. Additionally, the team also formulated the class diagrams of the program to provide

representations of the system that the team will work from. The class diagrams show the classes used in the software as well as their methods.

The team also modified the graphical user interfaces designed by the previous studies in order to conform to the general objective of this study – that is, to recognize handwritten digits. Graphical user interfaces help the user interact or control the application software without having to memorize many complicated commands. They also help in conveying proper information to the user in an uncomplicated, straightforward, and intuitive visual context [7].

3.3 Software Programming and Development

With the use case and class diagrams, the code for the application was modified. The code is the work-product from this action. This involved developing a Java program that takes the sample input patterns for every digit to be recognized. These sample input patterns were inputted to the neural network during training. The actual input which is the digit written manually by the user in the user interface were inputted to the neural network during recognition process. The code for the system was modified in Netbeans IDE 7.0.

3.3.1 Setting Network Specifications

In a Kohonen neural network, the number of output neurons should match the number of unique digit samples that are provided. Thus, the number of output neurons was set to 10 – each output neuron representing digit from 0 to 9. From the previous section, the downsampled image forms a 5x7 array. Thus, the number of input patterns that will be inputted to the neural network is 35. The neural network has only one input layer and one output layer. The learning rate is a constant that will be used by the learning algorithm and was chosen to be 0.3. The error is the

percent number that gives an idea of how well the Kohonen Network is in classifying the input into the output groups and this was chosen to be 10%.

3.3.4 Learning and Recognition

In this, the 5x7 array input will be fed in the form of binary pixels of 1 for white and zero for black pixel. As a result, the program will feed it the value of 0.5 for a black pixel and -0.5 for a white pixel. This array of 35 values will be fed to the input neurons. This will be done by passing the input array to the Kohonen neural network. This will return which of the 35 neurons won and will be stored as the "best" integer. Calculating winning node *c* based on the maximum activation among all *p* neurons participating in a competition, $C = \max \Sigma\ w_{ij}\ x_i$. So the neuron with the largest activation will be the winner. The neuron has the final output of 1 or the firing neuron. All other neurons in the layer will have an output of zero.

3.4 Application Testing

Application testing was conducted to verify if the application meets the functional and technical requirements or to assess whether or not the developed program performs as it is supposed to – that is, it does what the use cases specify. In this part, it was first ensured that the program was build and compiled with no errors. The team tested the application by inputting the sample input pattern and handwriting the digits from 0 to 9 into the application.

CHAPTER 4

RESULTS, DISCUSSION, AND INTERPRETATION OF DATA

4.1 Data Gathering Results

In gathering the user requirements, the researchers have evaluated and considered the following general necessities in the user interface that the user needs in recognizing handwritten digits:

1. inputting of sample input patterns for each digit from 0 to 9
2. training the neural network with the sample input patterns provided
3. handwriting area for the actual digit to be recognized
4. recognizing the digit handwrote by the user

These requirements covered the features of the application.

4.2 Domain Analysis and Design Results

The use case diagram showing the developed use cases is shown in Figure 4.1. Moreover, the class diagrams of the application are shown in Figure 4.2.

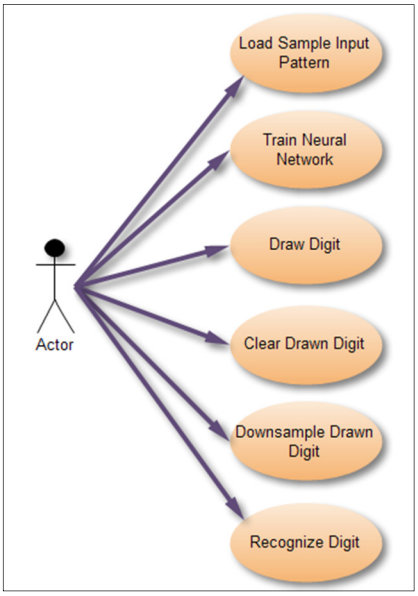

Figure 4.1. Use case diagram

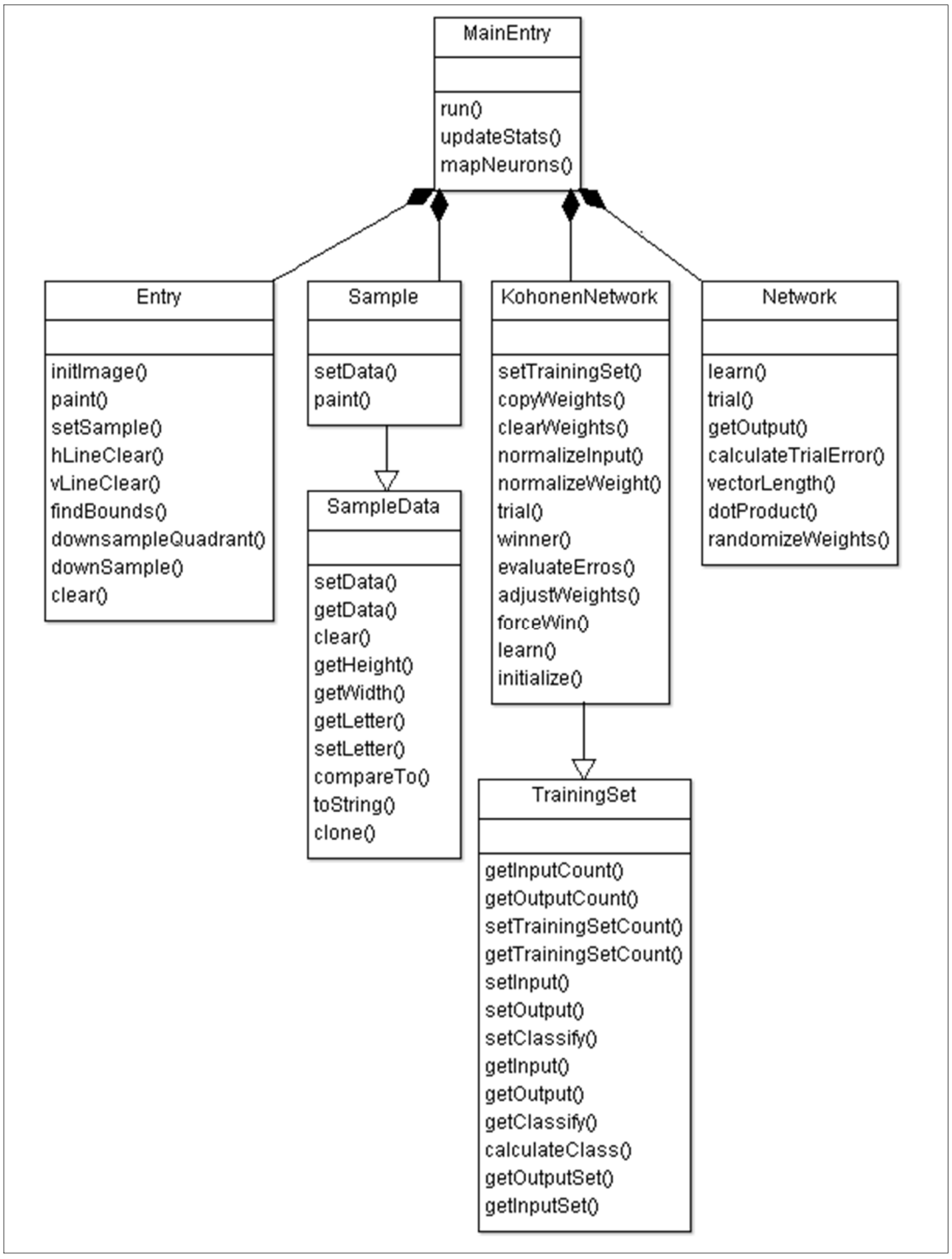

Figure 4.2. Class diagrams and their associations

4.3 Software Programming and Development Results

4.3.1 Downsampling the Image

All images are downsampled before being used, which prevents the neural network from being confused by size and position. The drawing area is large enough so the user could draw a letter in several different sizes. By downsampling the image to a consistent size, it won't matter how large the user draws the letter, as the downsampled image will always remain a consistent size. This section shows how this is done.

When the user draws an image, the program first draws a box around the boundary of the digit. This allows the program to eliminate all the white space around your letter. This process is done inside the "downsample" method of the Entry.java class. As the user draws a digit, this digit is also drawn onto the "entryImage" instance variable of the Entry object. To crop this image and eventually downsample it, the bit pattern of the image must be grabbed. This is done using a PixelGrabber class:

```
int w = entryImage.getWidth(this);
int h = entryImage.getHeight(this);

PixelGrabber grabber =
new PixelGrabber(entryImage,0,0,w,h,true);
grabber.grabPixels();
pixelMap = (int[]) grabber.getPixels();
```

After this code completes, the pixelMap variable, which is an array of int data types, now contains the bit pattern of the image. The next step is to crop the image and remove any white space. Cropping is implemented by dragging four imaginary lines from the top, left, bottom, and right sides of the image. These lines will stop as soon as they cross an actual pixel. By doing this,

these lines snap to the outer edges of the image. The hLineClear and vLineClear methods both accept a parameter that indicates the line to scan, and returns true if that line is clear. The program works by calling hLineClear and vLineClear until they cross the outer edges of the image. The horizontal line method (hLineClear) is shown here.

```
protected boolean hLineClear (int y)
{
int w = entryImage.getWidth(this);
for ( int i=0; i<w; i++ ) {
if ( pixelMap [(y*w) + i] !=-1 )
return false;
}
return true;
}
```

As seen, the horizontal line method accepts a *y* coordinate that specifies the horizontal line to check. The program then loops through each *x* coordinate on that row, checking for any pixel values. The value of -1 indicates white, so it is ignored. The "findBounds" method uses "hLineClear" and "vLineClear" to calculate the four edges. The beginning of this method is shown here:

```
protected void findBounds(int w,inth)
{ // top line
for ( int y=0;y<h;y++ ) {
if ( !hLineClear(y) ) {
downSampleTop=y; break;
}

}// bottom line
for ( int y=h-1;y>=0;y-- ) {
if ( !hLineClear(y) ) {
downSampleBottom=y; break;
}
}
```

The program shows how to calculate the top and bottom lines of the cropping rectangle. To calculate the top line, the program starts at 0 and continues to the bottom of the image. As soon as the first nonclear line is found, the program establishes this as the top of the clipping rectangle. The same process, only in reverse, is carried out to determine the bottom of the image. The processes to determine the left and right boundaries are carried out in the same way.

Now that the image has been cropped, it must be downsampled. This involves taking the image from a larger resolution to a 5x7 resolution. To reduce an image to 5x7, think of an imaginary grid being drawn over the high-resolution image. This divides the image into rectangular sections, five across and seven down. If any pixel in a section is filled, the corresponding pixel in the 5x7 downsampled image is also filled. Most of the work done by this process is accomplished inside the "downSampleQuadrant" method shown here.

```
protected boolean downSampleQuadrant (int x,int y)
{
int w = entryImage.getWidth(this);
int startX = (int) (downSampleLeft + (x*ratioX));
int startY = (int) (downSampleTop + (y*ratioY));
int endX = (int) (startX + ratioX);
int endY = (int) (startY + ratioY);

for ( int yy=startY; yy<=endY; yy++ ) {
for ( int xx=startX;
xx<=endX; xx++ ) {
int loc = xx + (yy*w);

if ( pixelMap[ loc ]!= -1 )
return true;
}
}

return false;
}
```

The "downSampleQuadrant" method accepts the section number that should be calculated. First the starting and ending *x* and *y* coordinates must be calculated. To calculate the first *x* coordinate for the specified section, first the "downSampleLeft" is used; this is the left side of the cropping rectangle. Then *x* is multiplied by "ratioX", the ratio of how many pixels make up each section. This determines where to place "startX". The starting *y* position, "startY", is calculated by similar means. Next the program loops through every *x* and *y* covered by the specified section. If even one pixel is determined to be filled, the method returns true, which indicates that this section should be considered filled.

The "downSampleQuadrant" method is called in succession for each section in the image. These results of the sample image are stored in the "SampleData" class, a wrapper class that contains a 5x7 array of Boolean values. It is this structure that forms the input to both training and digit recognition.

4.3.2 Neural Network Recognition

The recognition process begins when the user draws a digit and then clicks the "Recognize" button. First the digit is downsampled to a 5x7 image. This image must be copied from its two-dimensional array to an array of doubles that will be fed to the input neurons.

```
entry.downSample();

double input[] = new double[5*7];
int idx=0;
SampleData ds = sample.getData();
for ( int y=0;y<ds.getHeight();y++ )
{
for ( int x=0;x<ds.getWidth();x++
) {
input[idx++] = ds.getData(x,y)?.5:-.5;
}
}
```

This code does the conversion. Neurons require floating point input. As a result, the program feeds it the value of 0.5 for a white pixel and -0.5 for a black pixel. This array of 35 values is fed to the input neurons by passing the input array to the Kohonen's "winner" method. This returns which of the 35 neurons won and is stored in the "best" integer.

```
int best = net.winner ( input , normfac , synth ) ;
char map[] = mapNeurons();

JOptionPane.showMessageDialog(this, " " + map[best]
+ " (Neuron #" + best + " fired)",
"That Letter Is",
JOptionPane.PLAIN_MESSAGE);
```

Knowing the winning neuron is not too helpful because it does not show the user which digit was recognized. To line up the neurons with their recognized digits, each digit image the network was trained from must be fed into the network and the winning neuron is determined. For example, if the user were to feed the training image for "3" into the neural network and the winning neuron were neuron #4, the user would know that it is the one that had learned to recognize 3's pattern. This is done by calling the "mapNeurons" method, which returns an array of digits. The index of each array element corresponds to the neuron number that recognizes that digit.

Most of the actual work performed by the neural network is done in the winner method. The first thing the winner method does is normalize the inputs and calculate the output values of each output neuron. The output neuron with the largest output value is considered the winner. First the "biggest" variable is set to a very small number to indicate there's no winner yet.

```
biggest = -1.E30;
for ( i=0 ; i<outputNeuronCount; i++ ) {
```

```
optr = outputWeights[i];
output[i] = dotProduct (input , optr ) * normfac[0] + synth[0] * optr[inputNeuronCount] ;
// Remap to bipolar(-1,1 to 0,1)
output[i] = 0.5 * (output[i] + 1.0) ;
if ( output[i] > biggest ) {
biggest = output[i] ;
win = i ;
}
```

Each output neuron's weight is calculated by taking the dot product of each output neuron's weights to the input neurons. The dot product is calculated by multiplying each of the input neuron's input values against the weights between that input neuron and the output neuron. These weights were determined during training, which is discussed in the next section. The output is kept, and if it is the largest output so far, it is set as the "winning" neuron.

4.3.3 Neural Network Learning

Learning is the process of selecting a neuron weight matrix that will correctly recognize input patterns. A Kohonen neural network learns by constantly evaluating and optimizing a weight matrix. To do this, a starting weight matrix must be determined. This matrix is chosen by selecting random numbers. Once the initial random weight matrix is created, the training can begin. First the weight matrix is evaluated to determine what its current error level is. This error is determined by how well the training input (the digits that the user created created) maps to the output neurons. The error is calculated by the "evaluateErrors" method of the KohonenNetwork class. If the error level is low, say below 10%, the process is complete.

When the user clicks the "Begin Training" button, the training process begins with the following code:

```
int inputNeuron = MainEntry.DOWNSAMPLE_HEIGHT*
MainEntry.DOWNSAMPLE_WIDTH;
int outputNeuron = letter ListModel.size();
```

This calculates the number of input and output neurons. First, the number of input neurons is determined from the size of the downsampled image. Since the height is 7 and the width is 5, the number of input neurons will be 35. The number of output neurons matches the number of characters the program has been given.

Now that the size of the neural network has been determined, the training set and neural network must be constructed. The training set is constructed to hold the correct number of "samples." These will be the 10 digits provided.

```
TrainingSet set = new TrainingSet(inputNeuron,outputNeuron);
set.setTrainingSetCount(letterListModel.size());
```

Next, the downsampled input images are copied to the training set; this is repeated for all 26 input patterns.

```
for ( intt=0; t<letterListModel.size(); t++ ) {
int idx=0; SampleData ds = (SampleData) letterListModel.getElementAt(t);
for ( int y=0; y<ds.getHeight(); y++ ) {
for ( int x=0; x<ds.getWidth(); x++ ) {
set.setInput (t, idx++, ds.getData(x,y), 0.5:-0.5);
}
}
}
```

Finally the neural network is constructed and the training set is assigned, so the "learn" method can be called. This will adjust the weight matrix until the network is trained.

```
net = newKohonenNetwork(inputNeuron,outputNeuron,this);
net.setTrainingSet(set);
net.learn();
```

The learn method will loop up to an unspecified number of iterations. Because this program only has one sample per output neuron, it is unlikely that it will take more than one iteration. When the number of training samples matches the output neuron count, training occurs very quickly.

```
n_retry = 0 ;
for ( iter=0 ; ; iter++ ) {
```

A method, "evaluateErrors", is called to evaluate how well the current weights are working. This is determined by looking at how well the training data spreads across the output neurons. If many output neurons are activated for the same training pattern, then the weight set is not a good one. An error rate is calculated, based on how well the training sets are spreading across the output neurons.

```
evaluateErrors ( rate, learnMethod, won, bigerr, correc, work ) ;
```

Once the error is determined, it must be checked if it is below the best error so far. If it is, this error is copied to the best error, and the neuron weights are also preserved.

```
totalError = bigerr[0] ;

if ( totalError < best_err ) {
best_err = totalError ;
copyWeights ( bestnet , this ) ;
}
```

The total number of winning neurons is then calculated, allowing the user to determine if no output neurons were activated. In addition, if the error is below the accepted quit error (10%), the training stops.

```
winners = 0 ;
for ( i=0; i<won.length; i++ )
if ( won[i]!=0 )
winners++;
if ( bigerr[0] < quitError )
break ;
```

If there is no acceptable number of winners, one neuron is forced to win.

```
if ( (winners <outputNeuronCount) && (winners <train.getTrainingSetCount())) {
forceWin ( won ) ;
continue ;
}
```

Now that the first weight matrix has been evaluated, it is adjusted based on its error. The adjustment is slight, based on the correction that was calculated when the error was determined. This two-step process of adjusting the error calculation and adjusting the weight matrix is continued until the error falls below 10%.

```
adjustWeights ( rate, learnMethod, won, bigcorr, correc ) ;
```

This is the process by which a neural network is trained. The method for adjusting the weights and calculating the error is shown in the KohonenNetwork.java file.

4.4 Application Testing Results

All the digits from 0 to 9 were manually handwritten into the application to test whether the program recognizes the digits correctly. Unfortunately, one digit was incorrectly recognized by the program – the digit 5. This digit was recognized by the application as 6. This is because the sample pattern for the digit 5 is very close to the sample pattern of the digit 6. However, if care is severely implemented in handwriting the digit 5, the program recognizes it correctly as 5. This is due to one of the limitations of the application – that is, the sample input patterns depend on the accuracy of the handwriting of the user.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

Selecting random numbers for the weight matrices is a terrible choice for a weight matrix, but it gives a starting point to optimize from. Getting the results from a neural network is a quick process. It is actually the determination of the weights of the neurons that is the complex portion of the process.

Neural networks provide an efficient way of performing certain operations that would otherwise be very difficult. Consider how a character recognition program would work without neural networks. One would likely find himself writing complex routines that traced outlines, analyzed angles, and did other graphical analysis. Neural networks should be considered anytime complex patterns must be recognized. These patterns do not need to be graphical in nature. Any form of data that can have patterns is a candidate for a neural network solution.

5.2 Recommendations

Based on the results of the study and the conclusions presented, the team would like to recommend the following:

1. Further improve the program by allowing the user to define more than one drawing per digit.

2. Further improve the program by providing more accurate sample input pattern so that training and recognition becomes more accurate.

3. Further improve the program by recognizing many digits or characters at a time.

REFERENCES

[1] Wikipedia (n.d.). "Optical Character Recognition." Internet: http://en.wikipedia.org/wiki/Optical_character_recognition, [February 2012]

[2] A.S. Shatil, "Bangla Optical Character Recognition Using Kohonen Network." B.D. thesis, BRAC University, Bangladesh, May 2006.

[3] M. Yusoff, S.A. Rahman, S. Mutalib, A. Mohamed, "Kohonen Neural Network Performance in License Plate Number Identification." Universiti Teknologi MARA, Malaysia, June 2007.

[4] N.B. Mat Noor. "License Plate Recognition Using Kohonen Neural Network Algorithm." B.D. thesis, Universiti Teknologi MARA, Malaysia, November 2006.

[5] J. Heaton. (2002). "Introduction to Neural Network in Java." HR Publication.

[6] J. Schmuller. (2004). *SAMS Teach Yourself UML in 24 Hours*. (3rd Edition) [Online]. Available: http://ims.xebe.be/systeemontwikkeling/e-books%20UML/Teach.Yourself.Uml.In.24.Hours_Joseph.Schmuller_SAMS.pdf [December 2011]

[7] Knowledge Base. (2011, February 3). "What is a GUI?" Internet: http://kb.iu.edu/data/afhv.html [March 2012]

[8] Ben Krose, Patrick van der Smagt. *An Introduction to Neural Networks Eight Edition*. November 1996