

## Acknowledgement

---

We consider it as our privilege to express through the pages of this report a few word of gratitude and respect to all those personality who inspired us in the completion of this project. At the outset, it is our duty to acknowledge with gratitude the generous help that we have received from our lecturer Manish Aryal, Krishna Kandel, Santosh Prajapati and all other teachers who have helped us in this project. We are grateful to all the personnel's for giving us chance to do this project to explore our talent in programming skills, making us capable of utilizing the object oriented concept, and beside these their co-operation in collection and collation of information and material required for preparation of this project report is also appreciable. We would also like to express our gratitude to our college Kathmandu Engineering College and the Department of Electronics and Computer Engineering with due respect to our Lecturer, Sudip Lama, who not only taught us the topics related to Java and object oriented programming but also motivated us for completing this project report. Finally, it would not be impartial to acknowledge our friends who extended their helping hands to our project completion.

Rajeev Thapaliya

Saubhagya Joshi

Sushil Dahal

## Abstract

---

The project entitled “Live! Character Recognition” is comparable to optical character recognition software where input is fed through a camera attached to the computer and the processing is done at the instant of taking input. The goal is to recognize only the text component from the image stream which may have text-characters buried in the colored background, complex patterns and text like structures. The camera attached to the computer takes the live moving video or the image streams as an input then the text part of the image is only recognized through the use of a method, Multiscale Edge-Based Text Extraction. Finally, the use of Kohonen neural network helps to identify the text. This project is live i.e. all the processing is done to the image streams captured by the camera at that particular instant of capturing, hence no offline storage or buffering of large video is required. The programming platform that is used to develop this project is Java with the support from ImageJ for image processing and vlcj for capturing image streams.

# Table of Contents

---

Acknowledgement .....	iii
Abstract .....	iv
Table of Figures .....	vii
List of Abbreviations .....	viii
<b>CHAPTER ONE: INTRODUCTION .....</b>	<b>1</b>
1.1 Background .....	2
1.2 Objectives .....	3
1.3 Application.....	4
<b>CHAPTER TWO: FEASIBILITY STUDY .....</b>	<b>5</b>
2.1 Feasibility Basis .....	6
2.2 Operational Feasibility .....	7
2.3 Technical Feasibility .....	7
2.4 Economic Feasibility .....	8
<b>CHAPTER THREE: LITERATURE SURVEY.....</b>	<b>9</b>
3.1 Introduction.....	10
3.2 Optical Character Recognition: Limits of the Current Technology .....	10
3.3 Character Recognition from natural images .....	11
3.4 Real Time Image Processing .....	13
<b>CHAPTER FOUR: CASE STUDY .....</b>	<b>14</b>
4.1 Case study on: LookTel Recognizer .....	15
4.2 Case study on: ConTEXTract .....	16
<b>CHAPTER FIVE: WORKING .....</b>	<b>17</b>
5.1 Design Principle.....	18
5.2 Working Principle.....	20
5.2.1 Candidate Text Region Detection .....	21

5.2.2	Feature map generation.....	22
5.2.3	Text Region Localization.....	23
5.2.4	Character Extraction .....	23
5.2.5	Kohonen Neural Network .....	24
<b>CHAPTER SIX: PROBLEMS.....</b>		<b>34</b>
6.1	Problems in Analysis Phase .....	35
6.2	Problems in Design Phase.....	35
6.3	Problems in Implementation Phase.....	36
<b>CHAPTER SEVEN: GANTT CHART.....</b>		<b>37</b>
7.1	Project Gantt chart .....	38
<b>CHAPTER EIGHT: TESTING AND VERIFICATION .....</b>		<b>39</b>
8.1	Testing .....	40
<b>CHAPTER NINE: RECOMMENDATIONS AND CONCLUSIONS.....</b>		<b>43</b>
9.1	Recommendations.....	44
9.2	Conclusion .....	45
<b>BIBLIOGRAPHY .....</b>		<b>46</b>
<b>APPENDIX.....</b>		<b>48</b>

## Table of Figures

---

Figure 5.2.1: Block Diagram of Live! Character Recognition .....	20
Figure 5.2.2: A sample input image.....	20
Figure 5.2.3: Compass Operators.....	21
Figure 5.2.4: Featuremap of sample image.....	22
Figure 5.2.5: Text Region localization of sample image .....	23
Figure 5.2.6: Character Extraction of sample image .....	24
Figure 5.2.7: A Kohonen Neural Network.....	26
Figure 5.2.8: Training the Kohonen neural network.....	30
Figure 7.1.1: Descriptive Gantt Chart.....	38
Figure 7.1.2: Project Gantt Chart.....	38
Table 5.2.1: Sample Inputs to a Kohonen Neural Network.....	26
Table 5.2.2: Connection weights in the sample Kohonen neural network .....	27

## List of Abbreviations

---

API	-	Application Programming Interface
ACL	-	Association for Computational Linguistics
AWT	-	Abstract Window Toolkit
BMP	-	Bitmap
DICOM	-	Digital Imaging and Communication in Medicine
FITS	-	Flexible Image Transport System
GHz	-	Giga Hertz
GIF	-	Graphic Interchange Format
GUI	-	Graphical User Interface
IDE	-	Integrated Development Environment
iOS	-	iPhone Operating System
IPTV	-	Internet Protocol Television
ISRI	-	Information Science Research Institute
JPEG	-	Joint Photographic Experts Group
MATLAB	-	Matrix Laboratory
OCR	-	Optical Character Recognition
PDA	-	Personal Digital Assistant
PNG	-	Portable Network Graphics

RAM	-	Random Access Memory
SRI	-	Stanford Research Institute
TIFF	-	Tagged Image File Format
WORA	-	Write Once Run Anywhere

# **CHAPTER ONE: INTRODUCTION**



## 1.1 Background

Over the past few years, text extraction has become one of the prime concern for many people, be it for data entry or character recognition. Data entry brings a whole lot of manually entered data to the digital world. Similarly, character recognition is a whole new aspect where the characters from scanned documents are recognized. Understanding the context of the text in an image automatically can open up avenue for a lot of value-added applications. Now, what do we mean by understanding the context? Text context is usually associated with orientation, language, styles etc. that are embedded within the image – these are the information that helps us understand the content of the image. Text extraction can provide a lot of contextual information on the live video clips. In general, there are two types of texts embedded inside video namely, scene texts and artificial texts. Scene texts appear naturally in scenes shot by the cameras. Since artificial text is purposefully added, it is usually more structured and closely related to context than a scene text. The text data in video frames contain useful information for automatic annotation, indexing and summarization of the visual information.

To accomplish the task of developing Live! Character Recognition software we the three students, namely Rajeev Thapaliya, Saubhagya Joshi and Sushil Dahal, formed a group and started to develop it. We had divided the whole software project into small different tasks so that each one of us can independently work on it. The tasks were: GUI designing, documentation and processing. The GUI designing task was assigned to Rajeev Thapaliya. Saubhagya Joshi's task was to document each & every aspect of the project throughout its lifecycle and Sushil Dahal's task was to handle all the processing aspect of the project. Finally when every one's task completed and all components were combined, the Live! Character Recognition software took its final shape.

Throughout the software development lifecycle we devoted enough time and labor to finish the task that we were assigned in the allotted time.

## 1.2 Objectives

The objectives of our project are:

- To distinguish text from non-text in video imagery.
- To extract text from complex, patterned, and/or colored backgrounds.
- To process the input in real time.
- To make a complete user friendly environment to meet the requirement.
- To extract text from image as accurate as possible.
- To be able to use the software by physically handicapped person, like a blind person.

## 1.3 Application

Many developers can take advantage of the principles used in Live! Character Recognition. Currently, the largest use of Character Recognition among consumers is from Adobe Acrobat and Microsoft OneNote. This allows for easy character recognition and conversion to digital form. The software incorporates simple drop down menus, and scans the live video in very little time. These features are useful when scanning textbooks, as search functions becomes available. One may also export novels to text for easier storage after scanning.

Another application of Live! Character Recognition technology can be at the post office. Addresses and zip codes are often handwritten on the envelope. Live! Character Recognition allows the post office to automatically read the address of a piece of mail and sort it to the appropriate bin for delivery.

By adding extra functionality of converting text to speech, Live! Character Recognition can help the blind people to recognize the objects, matters and many other useful things that they come across by just initiating the program which can then spell the text that is recognized.

Live! Character Recognition greatly helps in locating texts in images. It can be used in following areas:

1. It helps in archiving the printed documents. Old documents which are of great importance to future generation can be stored in digital formats without the use of costly scanners and without causing any harm to the documents by simply skimming the camera over the documents.
2. This can be used in banks, hospitals, hotels and colleges to keep the information of users by scanning their documents with a simple camera.
3. It can be used in place of costly Bar-code readers, by providing a Bar-code to an item, scanning it by a camera and by extracting the Bar-code; the item can be easily identified.
4. This can be further enhanced to machine learning, where machines are able to process text in real objects and can be used in Artificial Intelligence.

# **CHAPTER TWO: FEASIBILITY STUDY**

## 2.1 Feasibility Basis

Feasibility studies aim to objectively and rationally uncover the strengths and weaknesses of an existing business or proposed venture, opportunities and threats as presented by the environment, the resources required to carry through, and ultimately the prospects for success.

Looking at original printed resources to assess whether character recognition will be an effective process requires a combination of factors to be compared. The key issue will be the likely accuracy of the character recognition engine in producing text output appropriate to the project requirements. If the accuracy is not in an acceptable range for the desired purpose then to improve this will require human intervention – particularly neural networks training, correction of text and proofreading. Once human intervention of any significance is introduced then the time, effort and thus project cost increases very quickly. The factors to compare are:

- Scanning – the standard of the page image
- The original – its nature and condition
- Nature of the printing
- Content
- Output requirements

Once these are considered in relation to the intellectual value desired from the character recognition text then decisions regarding project budgets and the effort required may be made. It is worth remembering that there are always a number of ways to deliver text to the user and that Live! Character Recognition is just one of the tools along the path to achieve this and so if it is not deemed the best method there will be other pathways to the project goals.

## 2.2 Operational Feasibility

The image processing task in itself is a very intuitive task. There are no predefined laws or algorithms to achieve an ideal outcome from it. Although high accuracy will be maintained, a perfect result cannot be expected. The outcome of the software also depends on the quality of image given as an input; the high pixel rate helps in good outcome and besides that making the pixel very large will eventually take larger time to process. So by giving a standard quality image, output delivered will not be unsatisfactory.

Further, the use of neural networks makes this software's output to divert away from the true result. As the neural networks are dependent on the training data's and the learning rate also greatly affects the outcome. The more trained the neural network is the better will be the output. For this, the neural networks are fairly trained while keeping the learning rate value in between 0.3-0.5, as a result the outcome will be closer to expectation.

## 2.3 Technical Feasibility

The programming language required for the development of Live! Character Recognition is Java. Programming in Java requires the knowledge of object oriented analysis and design as it is strictly an object oriented programming language. Having previous knowledge of object oriented programming language C++, learning Java was not a complicated task and beside that for everything that we want to do, there is a library in Java. Java is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. This feature of Java is very helpful as writing codes for multiple platforms would require a lot of time. Java is licensed under GNU General Public License so it is freely available to use and the IDE's like NetBean and Eclipse greatly helps in reducing the effort of designing and managing the project.

To capture images from a camera attached to the computer requires vlcj. vlcj is an Open Source project that provides Java bindings for the excellent vlc media player from VideoLAN. The vlcj provides Java bindings to allow an instance of a native vlc media player to be embedded in a Java AWT Window or Swing JFrame. We get more

than just simple bindings; we also get a higher level API that hides a lot of the complexities of working with libvlc.

For the processing of image requires ImageJ. ImageJ is a public domain Java image processing program. ImageJ and its Java source code are freely available and in the public domain. No license is required. We can even extend ImageJ by developing plugins using ImageJ's built in text editor and Java compiler. It can filter a 2048x2048 image in 0.1 seconds. Almost all image file formats are supported by ImageJ and to analyze and edit images, large amount of pre-defined tools are available.

Another aspect of technical feasibility study is the hardware resources. All the software tools required for the development of Live! Character Recognition would run in the environment with minimum resources. Further the speed and power greatly depends on the hardware available. Live! Character recognition even requires a camera to be present. Any camera that is detected by the libvlc is supported. Better the pixel quality of the camera more accurate will be the output.

## **2.4 Economic Feasibility**

The development of this software requires a good knowledge about image processing and neural networks. Time is also one of the key players in the successful development of this software. Through the use of open source technology in its development, the cost of resource is minimal but the effort required is above normal standard. The broader implementation of this software can replace the scanners, bar-code readers and can also provide a way to communicate for blind persons with the world. Even products like this can be easily found today, Live! Character Recognition places it on top of others by giving users the freedom from the restriction of hardware resources, as a result the profit that can be gained by the successful implementation of Live! Character recognition will obviously be high and up to the mark.

# **CHAPTER THREE: LITERATURE SURVEY**



### **3.1 Introduction**

Our project on Live! Character Recognition is basically in the field of research. To gain better knowledge, techniques and solutions regarding the procedures that we want to follow, we studied the various research papers on existing online character recognition systems. These papers are much more relevant with our project. All these study helped us with clarifying our target goals.

It is important to recognize that processing an image with complex elements in real time is a unique problem and standards in OCR technology and that directly impact the techniques we use to address the processing problem. Having said that, it is important for us to understand that both the research areas of character recognition and real time processing play a part in helping us build a solution to this problem. For this reason there are two main areas of research that this literature review covers, character recognition from complex images and real time image processing.

### **3.2 Optical Character Recognition: Limits of the Current Technology**

OCR equipment has been steadily built since the 1950s, and despite its early emergence, OCR has yet to achieve document read rates that are 100% perfect. More importantly, OCR accuracy depends highly upon the documents that are being processed; and as is the case for the ACL Anthology, document standards have differed over the years of proceedings and conferences. A detailed analysis of how the characteristics of a document may affect OCR accuracy is discussed by Nagy et al., where they explain how the typography, image defects, similar symbols and punctuation of documents all make OCR much harder than it seems. The specifications of the OCR device also affect the outcome of the text file; for example a scanner with a scanning resolution of 600dpi (dots per inch), 16-level grayscale, will ultimately have better chance of producing a better text result than a 300dpi, 8-level grayscale scanner. A study conducted by the Information Science Research Institute (ISRI) compares the OCR accuracy of seven companies and vendors, including companies such as Xerox, Recognita and Maxsoft-Ocron Recore. They report that during that years tests there is no clear 'winner', indicating relatively close accuracy rates from all tested OCR devices, despite the fact that ISRIs OCR device (the ISRI

Voting Machine) entry came first in most tests with an average accuracy percentage of 98.244% over five document types. Interestingly, ISRI report that little or no benefit may be obtained when increasing the resolution from an OCR device from 300dpi to 400dpi, suggesting that today's technology of 600dpi resolution will impact little on the improvement of OCR accuracy rates. Later research into algorithms such as expert voting for OCR, suggest algorithms themselves significantly impact OCR accuracy rates, complementing ISRI's Voting Machine, which already demonstrated the high recognition accuracy of their voting algorithm. In [Breithaupt] reports he obtained OCR accuracy rates of 98.15%, stating that his results measure characters that have been recognized correctly as opposed to commercial vendors that report accuracy rates based on recognized characters that can be either recognized correctly or incorrectly. Essentially, the accuracy rate of being able to recognize correct or incorrect characters mean that the accuracy rate is a measure of non-failure to recognize there is some symbol on the document to be interpreted. This is opposed to a measure used for correctly recognizing a character as it appears in the original document. ISRI staff also report on the DOE Document Conversion System, where they quote to have "selected and installed the best available OCR technology". They report conveniently on non-stop word recognition accuracy as well as character recognition accuracy, giving 98.15% for word accuracy and 99.44% for character accuracy on 17 first generation (photocopied once) documents. A clearer picture of the extent to which these percentages presented is relevant is that provided the OCR process is 99% accurate, this translates to 30 errors on a standard page of 3000 characters. Furthermore, assuming an average word-length of five characters, this equates to roughly one error per 20 words; suggesting that there is stills a lot of work to be accomplished in the optical character recognition field.

### **3.3 Character Recognition from natural images**

Text can be detected by exploiting the discriminate properties of text characters such as the vertical edge density, the texture or the edge orientation variance. One early approach for localizing text in covers of Journals or CDs assumed that text characters were contained in regions of high horizontal variance satisfying certain spatial properties that could be exploited in a connected component analysis process. [Smith et al.] localized text by first detecting vertical edges with a predefined template, then grouping vertical edges into text regions using a smoothing process. These two

methods are fast but also produce many false alarms because many background regions may also have strong horizontal contrast. The method of [Wu et al.] for text localization is based on texture segmentation. Texture features are computed at each pixel from the derivatives of the image at different scales. Using a K-means algorithm, pixels are classified into three classes in the feature space. The class with highest energy in this space indicates text while the two others indicate non-text and uncertainty. However, the segmentation quality is very sensitive to background noise and image content and the feature extraction is computationally expensive. More recently, [Garcia et al.] proposed a new feature referred to as variance of edge orientation. This relies on the fact that text strings contain edges in many orientations. Variation of edge orientation was computed in local area from image gradient and combined with edge features for locating text blocks. The method, however, may exhibit some problems for characters with strong parallel edges characteristics such as “i” or “l”. Besides the properties of individual characters, [Sobottka et al.] suggested that baseline detection could be used for text string localization. More precisely, printed text strings are characterized by specific top and bottom baselines, which can be detected in order to assess the presence of a text string in an image block. The above manually designed heuristic features usually perform fast detection but are not very robust when the background texture is very complex. As an alternative, a few systems considered machine learning tools to perform the text detection. These systems extracted wavelet or derivative features from fixed-size blocks of pixels and classified the feature vectors into text or non-text using artificial neural networks. However, since the neural network based classification was applied to all the possible positions of the whole image, the detection system was not efficient in terms of computation cost and produced unsatisfactory false alarm and rejection rates.

A focus of attention based system for text region localization has been proposed by [Liu and Samarabandu]. The intensity profiles and spatial variance is used to detect text regions in images. A Gaussian pyramid is created with the original image at different resolutions or scales. The text regions are detected in the highest resolution image and then in each successive lower resolution image in the pyramid. The overall average computation time for 75 test images (with  $480 \times 640$  resolution) using un-optimized MATLAB codes on a personal laptop with Intel Pentium(R) 1.8GHZ

processor and 1.0 GB RAM is 14.5 seconds (std dev. = 0.156), which includes entire run time including image reading, computation as well as image display. This method also showed the precision rate of 91.8% tested for four different types of images.

### **3.4 Real Time Image Processing**

Real Time Image Processing involves many aspects of hardware and software in order to achieve high resolution input, low latency capture, high performance processing and efficient display. Very careful choice of hardware and software is required for the development of high performance real time image processing applications.

Problems concerning any practical applications of image processing, such as in road traffic management system requires the images to be processed in real time. Various algorithms are mainly based on background techniques have been developed for this purpose but the background techniques are very sensitive to ambient lighting conditions, they do not yield the expected outcome. So, a real-time image tracking approach have been developed which takes the image sequences as input and then the processing is done in each input image at the very instant of taking the input.

## **CHAPTER FOUR: CASE STUDY**

Only a handful of Live! Character Recognition like software is available in the market. Those available are also not up to the mark, very costly, uses are limited, requires large hardware resources and very few users have access to it. Some of them are listed:

## **4.1 Case study on: LookTel Recognizer**

LookTel Recognizer allows users with visual impairments or blindness to instantly recognize everyday objects such as packaged goods in the pantry, identity cards, soda cans at the grocery store, or CDs in a music collection. Once a library has been built, users can simply point the iPhone's camera at an object and the phone will recognize and describe the item instantly.

The LookTel Recognizer App is intended for visually impaired users who can benefit from technology to assist in identifying objects. Recognizer permits users to store images of objects in a database, and have the iOS device quickly "recognize" these items later, by scanning the items with the camera. A barcode scanner is also included to provide additional identification and labeling help.

Recognizer allows users to backup and export their created databases via E-Mail, enabling them to restore a database from lost or replaced devices, or share it with other members of the community to use.

But there are drawbacks. For LookTel to work, users have to first save images into the app's library, and similar apps need an Internet connection to work.

## 4.2 Case study on: ConTEXTract

SRI International (SRI) has developed ConTEXTract, a text recognition technology that can find and read text (such as street signs, name tags, and billboards) in real scenes. This optical character recognition (OCR) for text within imagery and video requires a more specialized approach than is provided by off-the-shelf OCR software, which is designed primarily for recognizing text within documents. ConTEXTract distinguishes lines of text from other contents in the imagery, processes the lines, and then sends them to an OCR sub module, which recognizes the text. Any OCR engine can be integrated into ConTEXTract with minor modifications.

ConTEXTract is a configurable, real-time, end-to-end solution that provides a scalable architectural framework adaptable to a wide variety of imagery and processing environments. Unique capabilities developed by SRI for ConTEXTract include the following:

- The ability to distinguish text from non-text in video imagery
- The ability to extract text from complex, patterned, and/or colored backgrounds
- Character recognition despite oblique viewing angles, low resolution, and other distortion
- High accuracy by combining text information that appears over multiple video frames

## **CHAPTER FIVE: WORKING**



## 5.1 Design Principle

Live! Character recognition does not rely on the images already stored in the storage media rather it depends on the online images through camera. The project itself involves the development of different modules which can act independently. Taking live images as an input through camera is one part of the project, distinguishing text from non-text and extracting text from complex, patterned, and/or colored backgrounds is the other and the conversion of text is completely a new aspect of the project.

Taking live images from camera is done using vlcj. The vlcj project is an Open Source project that provides Java bindings for the excellent vlc media player from VideoLAN.

The bindings can be used to build media player client and server software using Java - everything from simply playing local media files to a full-blown video-on-demand streaming server is possible.

vlcj is being used in diverse applications, helping to provide video capabilities to software in use on oceanographic research vessels and bespoke IPTV and home cinema solutions.

The images are extracted and processing is done using ImageJ. ImageJ is a public domain, Java-based image processing program developed at the National Institutes of Health. ImageJ was designed with an open architecture that provides extensibility via Java plugins and recordable macros. Custom acquisition, analysis and processing plugins can be developed using ImageJ's built-in editor and a Java compiler. User-written plugins make it possible to solve many image processing and analysis problems, from three-dimensional live-cell imaging, to radiological image processing, multiple imaging system data comparisons to automated hematology systems. ImageJ's plugin architecture and built in development environment has made it a popular platform for teaching image processing.

ImageJ can be run as an online applet, a downloadable application, or on any computer with a Java 5 or later virtual machine. Downloadable distributions are

available for Microsoft Windows, Mac OS, Mac OS X, Linux, and the Sharp Zaurus PDA. The source code for ImageJ is freely available.

ImageJ can display, edit, analyze, process, save, and print 8-bit color and grayscale, 16-bit integer and 32-bit floating point images. It can read many image formats including TIFF, PNG, GIF, JPEG, BMP, DICOM, FITS, as well as raw formats. ImageJ supports image *stacks*, a series of images that share a single window, and it is multithreaded, so time-consuming operations can be performed in parallel on multi-CPU hardware. ImageJ can calculate area and pixel value statistics of user-defined selections and intensity thresholded objects. It can measure distances and angles. It can create density histograms and line profile plots. It supports standard image processing functions such as logical and arithmetical operations between images, contrast manipulation, convolution, Fourier analysis, sharpening, smoothing, edge detection and median filtering. It does geometric transformations such as scaling, rotation and flips. The program supports any number of images simultaneously, limited only by available memory.

## 5.2 Working Principle

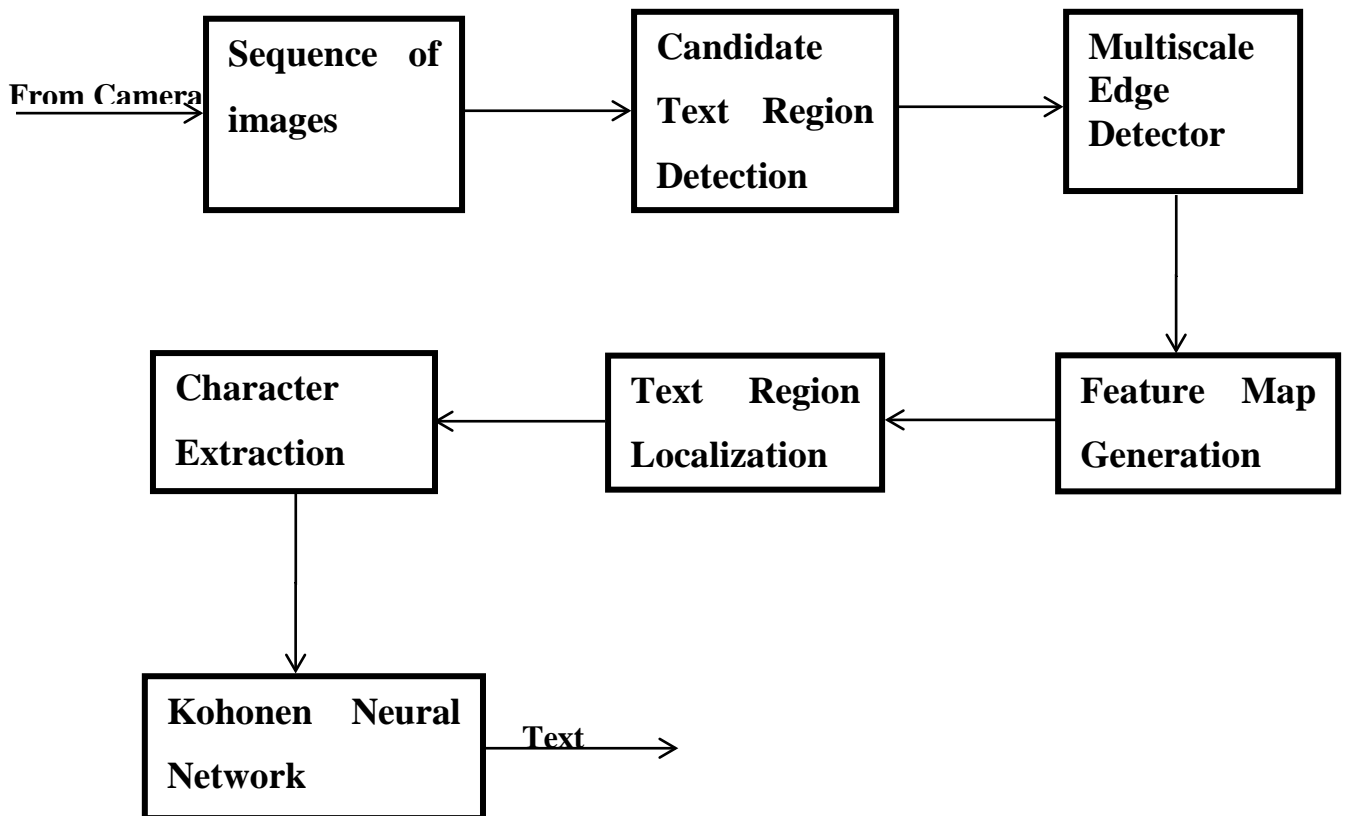


Figure 5.2.1: Block Diagram of Live! Character Recognition

This project is based on the character recognition from the input image. So the first step is to feed the image to the processor. The images are given to the system through the webcam or some digital camera or the static images that are saved on the hard disk. But, the camera descriptions and other related information's are manually given by the user.



Figure 5.2.2: A sample input image

After the extraction of sequence of images from camera. The extraction is done using a method of Multiscale Edge-Based Text Extraction.

The above mentioned method is based on the fact that images are a reliable feature of text regardless of color/intensity, layout, orientations, etc. Edge strength, density and the orientation variance are three distinguishing characteristics of text embedded in images, which can be used as main features for detecting text. The method consists of three stages: candidate text region detection, text region localization and character extraction.

### 5.2.1 Candidate Text Region Detection

This stage aims to build a feature map by using three important properties of edges: edge strength, density and variance of orientations. The feature map is a gray-scale image with the same size of the input image, where the pixel intensity represents the possibility of text.

#### Multi-scale edge detector

In this method, we use magnitude of the second derivative of intensity as a measurement of edge strength as this allows better detection of intensity peaks that normally characterize text in images. The edge density is calculated based on the average edge strength within a window. Considering effectiveness and efficiency, four orientations ( $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ , and  $135^\circ$ ) are used to evaluate the variance of orientations, where  $0^\circ$  denotes horizontal direction,  $90^\circ$  denote vertical direction, and  $45^\circ$  and  $135^\circ$  are the two diagonal directions, respectively. A convolution operation with a compass operator (as shown in Fig. 1) results in four oriented edge intensity images  $E(\theta)$ , ( $\theta \in \{0, 45, 90, 135\}$ ), which contain all the properties of edges required in this proposed method. Edge detector is carried out by using a multiscale strategy,

1	-1	-1
2	2	2
-1	-1	-1
$0^\circ$ Kernel		
-1	-1	2
-1	2	-1
2	-1	-1
$45^\circ$ Kernel		
-1	2	-1
-1	2	-1
-1	2	-1
$90^\circ$ Kernel		
2	-1	-1
-1	2	-1
-1	-1	2
$135^\circ$ Kernel		

Figure 5.2.3: Compass Operators

where the multiscale images are produced by Gaussian pyramids which successively low-pass filter and down-sample the original image reducing image in both vertical and horizontal directions. In our proposed method, those obtained multiscale images are simultaneously processed by the compass operator as individual inputs.

### 5.2.2 Feature map generation

As we mentioned before, regions with text in them will have significantly higher values of average edge density, strength and variance of orientations than those of non-text regions. We exploit these three characteristics to generate a feature map which suppresses the false regions and enhances true candidate text regions. This procedure is described in Eq.1.

$$fmap(i, j) = \bigvee_{s=0}^{\theta} \sum_{\theta} N \left\{ \sum_{x=-c}^{x=c} \sum_{y=-c}^{y=c} E(s, \theta, i+x, j+y) * W(i, j) \right\} \dots \dots \dots (1)$$

In the above equation, *fmap* is the output feature map;  $\bigvee$  is an across-scale addition operation, which employs the scale fusion.  $n$  is the highest level of scale, which is determined by the resolution (size) of the input image. Generally speaking, the higher the resolution is, the more scales can be used. In our implementation, we use two scales for images with resolution of  $640 \times 480$ .  $\theta \in \{0, 45, 90, 135\}$  are different orientation and  $N$  is a normalization operation.  $(i, j)$  are coordinates of an image pixel.  $W(i, j)$  is the weight for pixel  $(i, j)$ , whose value is determined by the number of edge orientations within a window. The window size is determined by a constant  $c$ . namely, the more orientations a window has, the larger weight the center pixel has. By employing this nonlinear weight mapping, the proposed method distinguishes text regions from texture-like regions, such as window frames, wall patterns, etc.

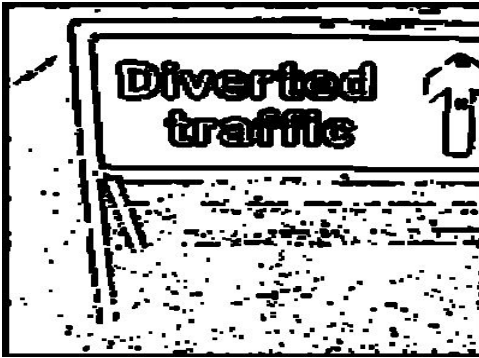


Figure 5.2.4: Featuremap of sample image

### 5.2.3 Text Region Localization

Normally, text embedded in an image appears in clusters, i.e., it is arranged compactly. Thus, characteristics of clustering can be used to localize text regions. Since the intensity of the feature map represents the possibility of text, a simple global thresh holding can be employed to highlight those with high text possibility regions resulting in a binary image. A morphological dilation operator can easily connect the very close regions together while leaving those whose positions are far away to each other isolated. In our proposed method, we use morphological dilation operator with a  $7 \times 7$  square structuring element to the previous obtained binary image to get joint areas referred to as text blobs.

Two constraints are used to filter out those blobs which do not contain text, where the first constraint is used to filter out all the very small isolated blobs whereas the second constraint filters out those blobs whose widths are much smaller than corresponding heights. The retaining blobs are enclosed in boundary boxes. Four pairs of coordinates of the boundary boxes are determined by the maximum and minimum coordinates of the top, bottom, left and right points of the corresponding blobs. In order to avoid missing those character pixels which lie near or outside of the initial boundary, width and height of the boundary box are padded by small amounts.



Figure 5.2.5: Text Region localization of sample image

### 5.2.4 Character Extraction

Existing OCR (Optical Character Recognition) engines can only deal with printed characters against clean backgrounds and cannot handle characters embedded in shaded, textured or complex backgrounds. The purpose of this stage is to extract accurate binary characters from the localized text regions so that we can use the

existing OCR directly for recognition. In this proposed method, use of uniform white character pixels in a pure black ground is done by using Eq.2.

$$T = \bigcup_{i=1 \dots} |\overline{SUB_i}|_z \dots \dots \dots (2)$$

In the above equation,  $T$  is the text extracted binary output image.  $\bigcup$  is a union operation.  $SUB_i$  are sub-images of the original image, where  $i$  indicate the number of sub-images. Sub-images are extracted according to the obtained boundary boxes in stage two.  $|\cdot|_z$  is a thresh holding algorithm which segments the text regions into white characters in a pure black background.



Figure 5.2.6: Character Extraction of sample image

### 5.2.5 Kohonen Neural Network

After the character extraction, for further processing the data is fed to the neural network system. The neural network used is a Kohonen neural network. Kohonen neural network provides a topological mapping between two layers of input. It places a fixed number of input patterns from the input layer into a higher- dimensional output or Kohonen layer. The major role of this network is to make the text extraction more clear and vivid to understand.

The Kohonen neural network differs considerably from the feed forward back propagation neural network. The Kohonen neural network differs both in how it is trained and how it recalls a pattern. The Kohonen neural network does not use any sort of activation function. Further, the Kohonen neural network 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. Often these "winning" neurons represent groups in the data that is presented to the Kohonen network.

The most significant difference between the Kohonen neural network and the feed forward back propagation neural network is that the Kohonen network trained in an unsupervised mode. This means that the Kohonen network is presented with data, but the correct output that corresponds to that data is not specified. Using the Kohonen network this data can be classified into groups.

It is also important to understand the limitations of the Kohonen neural network. We know that neural networks with only two layers can only be applied to linearly separable problems. This is the case with the Kohonen neural network. Kohonen neural networks are used because they are a relatively simple network to construct that can be trained very rapidly.

### The Structure of the Kohonen Neural Network

The Kohonen neural network contains only an input and output layer of neurons. There is no hidden layer in a Kohonen neural network. First we will examine the input and output to a Kohonen neural network.

The input to a Kohonen neural network is given to the neural network using the input neurons. These input neurons are each given the floating point numbers that make up the input pattern to the network. A Kohonen neural network requires that these inputs be normalized to the range between -1 and 1. Presenting an input pattern to the network will cause a reaction from the output neurons.

The output of a Kohonen neural network is very different from the output of a feed forward neural network. We also know that if we had a neural network with five output neurons we would be given an output that consisted of five values. This is not the case with the Kohonen neural network. In a Kohonen neural network only one of the output neurons actually produces a value. Additionally, this single value is either true or false. When the pattern is presented to the Kohonen neural network, one single output neuron is chosen as the output neuron. Therefore, the output from the Kohonen neural network is usually the index of the neuron (i.e. Neuron #5) that fired. The structure of a typical Kohonen neural network is shown in Figure 5.2.3.



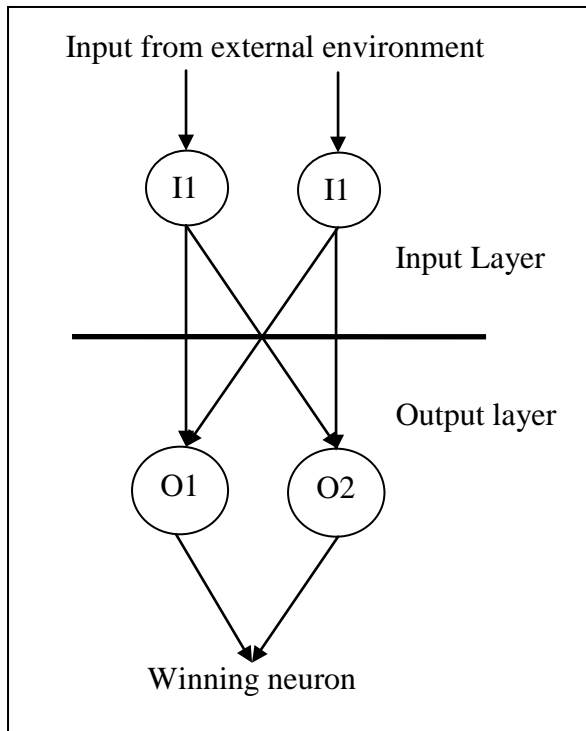


Figure 5.2.7: A Kohonen Neural Network

To examine how the network process information we will step through the calculation process. For this example we will consider a very simple Kohonen neural network. This network will have only two input and two output neurons. The input given to the two input neurons is shown in Table 5.2.1.

Input Neuron 1 (I1)	0.5
Input Neuron 2 (I2)	0.75

Table 5.2.1: Sample Inputs to a Kohonen Neural Network

We must also know the connection weights between the neurons. These connection weights are given in Table 5.2.2.

I1-> O1	0.1
I2->O1	0.2
I1->O2	0.3
I2->O2	0.4

Table 5.2.2: Connection weights in the sample Kohonen neural network

Using these values we will now examine which neuron would win and produce output. We will begin by normalizing the input.

### Normalizing the Input

The Kohonen neural network requires that its input be normalized. Because of this some texts refer to the normalization as a third layer. For the purposes of this book the Kohonen neural network is considered a two layer network because there are only two actual neuron layers at work in the Kohonen neural network.

The requirements that the Kohonen neural network places on its input data is one of the most severe limitations of the Kohonen neural network. Input to the Kohonen neural network should be between the values -1 and 1. In addition, each of the inputs should fully use the range. If one, or more, of the input neurons were to use only the numbers between 0 and 1, the performance of the neural network would suffer.

To normalize the input we must first calculate the "vector length" of the input data, or vector. This is done by summing the squares of the input vector. In this case it would be.

$$(0.5 * 0.5) + (0.75 * 0.75)$$

This would result in a "vector length" of 0.8125. If the length becomes too small, say less than the length is set to that same arbitrarily small value. In this case the "vector length" is a sufficiently large number. Using this length we can now determine the normalization factor. The normalization factor is the reciprocal of the square root of the length. For our value the normalization factor is calculated as follows.

$$\frac{1}{\sqrt{0.8125}}$$

This results in a normalization factor of 1.1094. This normalization process will be used in the next step where the output layer is calculated.

### Calculating Each Neuron's Output

To calculate the output the input vector and neuron connection weights must both be considered. First the "dot product" of the input neurons and their connection weights must be calculated. To calculate the dot product between two vectors you must multiply each of the elements in the two vectors. We will now examine how this is done.

The Kohonen algorithm specifies that we must take the dot product of the input vector and the weights between the input neurons and the output neurons. The result of this is as follows.

$$|0.5 \ 0.75| \text{ (dot)} |0.1 \ 0.2| = (0.5 * 0.75) + (0.1 * 0.2)$$

As we can see from the above calculation the dot product would be 0.395. This calculation will be performed for the first output neuron. This calculation will have to be done for each of the output neurons. Through this example we will only examine the calculations for the first output neuron. The calculations necessary for the second output neuron are calculated in the same way.

This output must now be normalized by multiplying it by the normalization factor that was determined in the previous step. We must now multiply the dot product of 0.395 by the normalization factor of 1.1094. This results in an output of 0.438213. Now that the output has been calculated and normalized it must be mapped to a bipolar number.

### Mapping to Bipolar

As you know a bipolar number is an alternate way of representing binary numbers. In the bipolar system the binary zero maps to -1 and the binary remains a 1. Because the input to the neural network normalized to this range we must perform a similar normalization to the output of the neurons. To make this mapping we add one and divide the result in half. For the output of 0.438213 this would result in a final output of 0.7191065.

The value 0.7191065 is the output of the first neuron. This value will be compared with the outputs of the other neuron. By comparing these values we can determine a "winning" neuron.

### Choosing the Winner

We have seen how to calculate the value for the first output neuron. If we are to determine a winning output neuron we must also calculate the value for the second output neuron. We will now quickly review the process to calculate the second neuron.

The second output neuron will use exactly the same normalization factor as was used to calculate the first output neuron. As we recall from the previous section the normalization factor is 1.1094. If we apply the dot product for the weights of the second output neuron and the input vector we get a value of 0.45. This value is multiplied by the normalization factor of 1.1094 to give the value of 0.0465948. We can now calculate the final output for neuron 2 by converting the output of 0.0465948 to bipolar yields 0.49923.

As we can see we now have an output value for each of the neurons. The first neuron has an output value of 0.7191065 and the second neuron has an output value of 0.49923. To choose the winning neuron we choose the output that has the largest output value. In this case the winning neuron is the first output neuron with an output of 0.7191065, which beats neuron two's output of 0.49923.

We have now seen how the output of the Kohonen neural network was derived. As we can see the weights between the input and output neurons determine this output. We can even adjust these weights to produce output that is more suitable for the desired task. The training process is what modified these weights.

### How Kohonen Network Learns

There are several steps involved in the training process. Overall the process for training a Kohonen neural network involves stepping through several epochs until the error of the Kohonen neural network is below acceptable level.

The training process for the Kohonen neural network is competitive. For each training set one neuron will "win". This winning neuron will have its weight adjusted so that it

will react even more strongly to the input the next time. As different neurons win for different patterns, their ability to recognize that particular pattern will be increased.

We will first examine the overall process involving training the Kohonen neural network. These individual steps are summarized in figure 5.2.4.

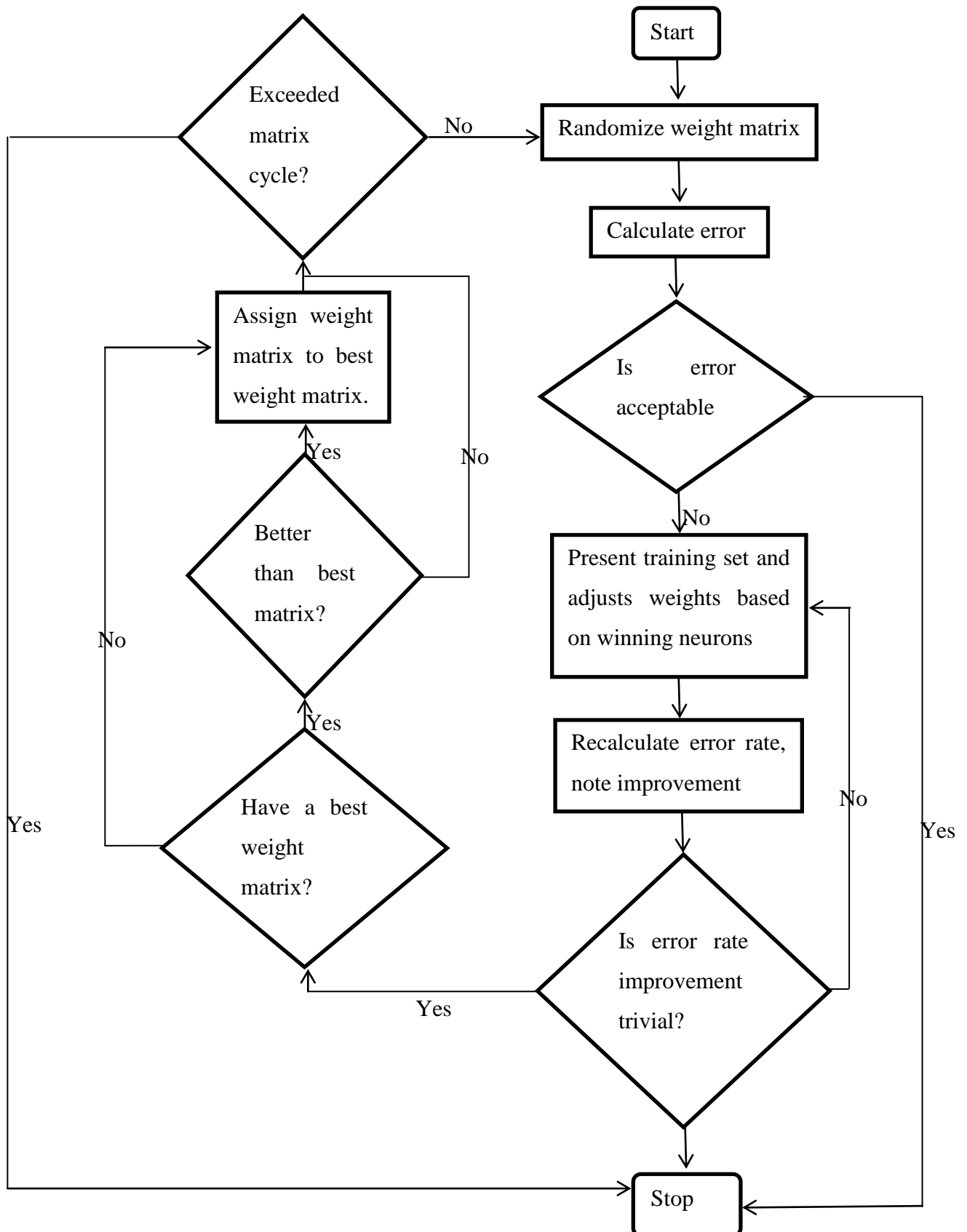


Figure 5.2.8: Training the Kohonen neural network

As we can see from the above diagram that Kohonen neural network is trained by repeating epochs until one of two things happens. If the calculated error is below acceptable level business at block will complete the training process. On the other hand, if the error rate has all only changed by a very marginal amount this individual cycle will be aborted with tile any additional epochs taking place. If it is determined that the cycle is to be aborted the weights will be initialized random values and a new training cycle began. This training cycle will continue the previous training cycle and that it will analyze epochs on to solve get the two is either abandoned or produces a set of weights that produces an acceptable error level.

The most important part in the network's training cycles is the individual epochs. We will now examine what happens turning each of these epochs. We will begin by examining how the weights are adjusted for each epoch.

### Learning Rate

The learning rate is a constant that will be used by the learning algorithm. The learning rate must be a positive number less than 1. Typically the learning rate is a number such as .4 or .5. In the following section the learning rate will be specified by the symbol alpha.

Generally setting the learning rate to a larger value will cause the training to progress faster. Though setting the learning rate to too large a number could cause the network to never converge. This is because the oscillations of the weight vectors will be too great for the classification patterns to ever emerge. Another technique is to start with a relatively high learning rate and decrease this rate as training progresses. This allows initial rapid training of the neural network that will be "fine tuned" as training progresses.

The learning rate is just a variable that is used as part of the algorithm used to adjust the weights of the neurons. In the next section we will see how these weights are adjusted using the learning rate.

### Adjusting Weights

The entire memory of the Kohonen neural network is stored inside of the weighted connections between the input and output layer. The weights are adjusted in each epoch. An epoch occurs when training data is presented to the Kohonen neural network and the weights are adjusted based on the results of this item of training data. The adjustments to the weights should produce a network that will yield more favorable results the next time the same training data is presented. Epochs continue as more and more data is presented to the network and the weights are adjusted.

Eventually the return on these weight adjustments will diminish to the point that it is no longer valuable to continue with this particular set of weights. When this happens the entire weight matrix is reset to new random values. This forms a new cycle. The final weight matrix that will be used will be the best weight matrix determined from each of the cycles. We will now examine how these weights are transformed.

The original method for calculating the changes to weights, which was proposed by Kohonen, is often called the additive method. This method uses the following equation.

$$w^{y+1} = \frac{w^y + \alpha x}{||w^y + \alpha x||}$$

The variable  $x$  is the training vector that was presented to the network. The variable  $w^y$  is the weight of the winning neuron, and the variable  $w^{y+1}$  is the new weight. The double vertical bars represent the vector length. This method will be implemented in the Kohonen example shown later in this chapter.

The additive method generally works well for Kohonen neural networks. Though in cases where the additive method shows excessive instability, and fails to converge, an alternate method can be used. This method is called the subtractive method. The subtractive method uses the following equations.

$$e = x - w^y$$

$$w^{y+1} = w^y + 1$$

These two equations show the basic transformation that will occur on the weights of the network.

### Calculating the Error

Before we can understand how to calculate the error for Kohonen neural network we must first understand what the error means. The Kohonen neural network is trained in an unsupervised fashion so the definition of the error is somewhat different involving the normally think of as an error.

In unsupervised training calculating an error means the difference between the anticipated output of the neural network and the actual output of the neural network. And in unsupervised training there is no anticipated output. Because of this we may be wondering exactly how we can calculate an error. The answer is that the error where calculating is not be true error, or at least not an error in the normal sense of the word.

The purpose of the Kohonen neural network is to classify the input into several sets. The error for the Kohonen neural network, therefore, must be able to measure how well the network is classifying these items. There is no official way to calculate the error for a Kohonen neural network. The error is just a percent number that gives an idea of how well the Kohonen network is classifying the input into the output groups. The error itself is not used to modify the weights, as is done in the back propagation algorithm.



## **CHAPTER SIX: PROBLEMS**

## 6.1 Problems in Analysis Phase

During the analysis phase, we have encountered many problems such as, initially we were in trouble of choosing programming platform for our project, and different programming platforms like visual basic, Java, etc are available. Since we had studied about java, it is platform independent too and there are large resources available for java we made a decision to choose java for our project. The next problem faced was whether to let only a single integrated camera to be interfaced or multiple cameras are to be allowed. User may want input from any camera attached to computer or a new high quality camera need to be used instead of the existing low quality integrated one, so we decided on designing the software which enables interfacing of multiple cameras for the effective use of the software. The problem of deciding to make text recognized to be available in editable form or not was decided following a test in which the outcome was, if we provide output in editable form we may have the option to save the result or modify according to our need but the output can eventually be edited which can raise the false alarm of the output being incorrect, so we had to choose not to make output in editable form. Using live image form camera only may not be sufficient so we had to provide an option to take input from camera as well as saved image. We also analyzed on converting the final output text to speech in our project but considering the available time span this feature was omitted.

## 6.2 Problems in Design Phase

During the design phase, different java bindings and plugins related to our project were analyzed. The vlcj binding to interface camera and ImageJ binding to process the image was found to be suitable to be used in developing the software. But, using vlcj did not allow us to access hardware directly, so we had to do it manually. We also had to set image folder to display saved image from the disk for and input can be given by selecting the displayed images, this feature allowed user to load the image from the same folder by just selecting the image.

There are different types of neural networks available but the Kohonen neural networks was used because they are relatively simple networks than othes to construct and it can also be trained very rapidly.

## 6.3 Problems in Implementation Phase

During implementation phase it was found that large images cannot be opened by the method that we have incorporated. Also the neural networks cannot give perfect results and so all the characters of the images may not be recognized. It is a very difficult task to train an image with a neural network and it is also time consuming as images suffer from various noise and lighting conditions. Other problems faced during implementation were that the text in the images should be of relatively uniform size, the text with much variance in size may not be recognized. In order to minimize this problem we took the images which contained text of almost the same size. Also the image processing is a heuristic task; and so the characters extracted from the processing method may contain some non-characters.

# **CHAPTER SEVEN: GANTT CHART**

## 7.1 Project Gantt chart

### Live! Character Recognition









		Name	Duration	Start	Finish	Predecessors
1		System Analysis	7d	06/03/2012	06/10/2012	
2		System Design	7d	06/11/2012	06/18/2012	1
3		Coding	30d	06/19/2012	07/19/2012	2
4		Testing	5d	07/20/2012	07/25/2012	3
5		Debugging	7d	07/26/2012	08/02/2012	4
6		Efficiency and performance testing	3d	08/03/2012	08/06/2012	5
7		Documentation	69d	06/03/2012	08/11/2012	

Figure 7.1.1: Descriptive Gantt Chart

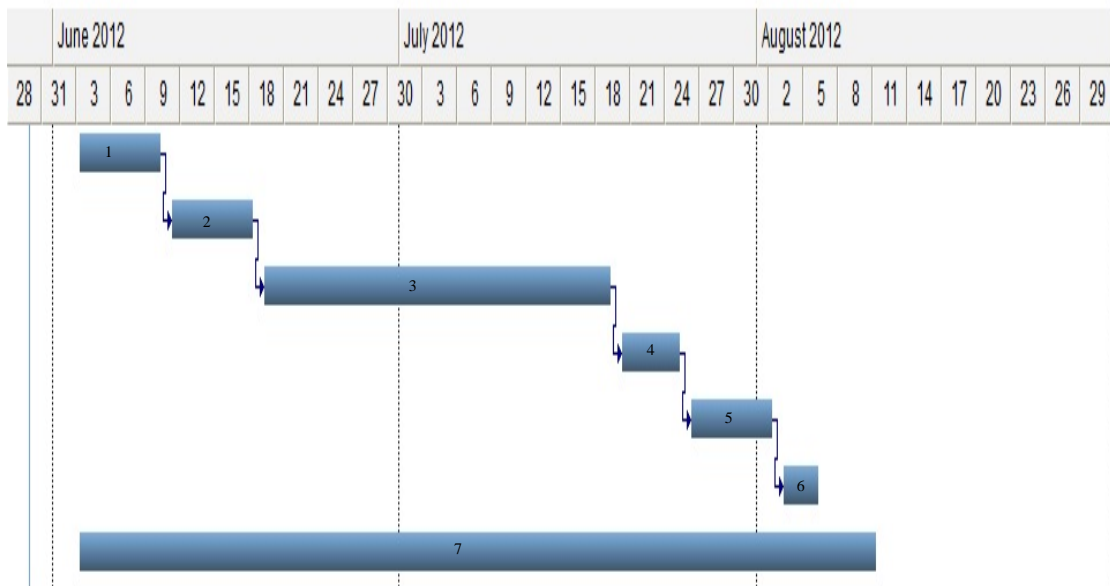


Figure 7.1.2: Project Gantt Chart

# **CHAPTER EIGHT: TESTING AND VERIFICATION**

## 8.1 Testing

Software testing is the critical element of software quality assurance and represents the ultimate review of specification, design and coding. Testing enables us to know whether the system we have proposed to meet the requirements or not. So in this phase, the system is used experimentally to ensure that the software does not fail or the software will run accordingly to its specifications and in the way users expect.

As testing can be done should take various forms to increase the chance of catching and verify correctness, here the testing includes internal reviews, black box testing, white box testing and alpha beta testing in the context of entire testing. Another purpose of testing is to avoid the complex responses produced by the interaction of hardware, environmental inputs and software.

### Internal Reviews

More than fifty percent of the discrepancies can be found by code inspection and /or code audit. So the internal reviews are done by colleagues and examined the correctness of the software in order to ferret out mistakes and errors in logic.

### White Box Testing

The White box testing uses the control structure of the procedural design to drive the test cases. Using this method one can design the test case that: guarantee that the independent paths within a module have been exercised at least once: execute all loops at their boundaries and within their operational bound: exercise internal data structures to assure their validity: and exercise all logical decisions and functional paths within a software module.

### Black Box Testing

Black box testing uses typical scenarios to ensure that input and output interfaces are functioning correctly. Hence, to ensure the integrity of the information flow into and out of a module without concern for what happens within the software, we performed this testing.

The Black Box Testing attempts to find the errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures
- Performance errors and
- Initialization and termination errors.

### Alpha and Beta Testing

The nearly complete software is exercised software is exercised in the actual operating environments. Different personals, both native and expert users are allowed to generate inputs and results for testing. So alpha test is done to collaborate with and observe people using the software and necessary components of beta testing is done by recording and reporting the results consistently.

In this testing phase the prepared final application is tested using a sample data extracted from the final database. The same sample of data is processed manually too and compared the result with that got from the run. Hence, the parallel processing is done to ensure the reliability of the system for future use.

Though the above mentioned tests are the industry standard tests. The test we followed is described below:

In order to evaluate the performance of the processing method, we use test images with different font sizes, perspective and alignment under different lighting conditions. From the results obtained, we saw that our proposed method can extract text with different font sizes, perspective, alignment, any number of characters in a text string under different lighting conditions.

Although there is no generally accepted method that can be used to evaluate the performance of the text localization and extraction method, we use several metrics used by others to facilitate a meaningful comparison. In our proposed method, we compute the accuracy of the algorithm output by manually counting the number of correctly located characters, which are regarded as ground-truth. Precision rate and recall rate are quantified to evaluate the performance given by (3) and (4).

$$Precision = \frac{CorrectlyLocated}{CorrectlyLocated + FalsePositive} \times 100\% \dots \dots \dots (3)$$



$$Recall = \frac{CorrectlyLocated}{CorrectlyLocated + FalseNegative} \times 100\% \dots \dots \dots (4)$$

Although (5) is not a standard definition, it intuitively interprets the false positive rate.

$$FalsePosotive = \frac{FalsePositive}{CorrectlyLocated + FalseNegative} \times 100\% \dots \dots \dots (5)$$

The proposed method has been implemented in Java with the IamgeJ bindings. The overall average computation time for 25 test images (with 480x 640 resolution) on a PC with a 1.5 GHz Pentium IV CPU and 256MB memory is 2.497s (std dev.=0.697), which includes entire run time including image reading, computation as well as image display. Although around 3 seconds of average processing time is very small, we have not optimized the proposed method, where the edge labeling and length labeling operations take the most time about 1.891s (std dev.=0.694). Our goal is to optimize it less than 1 second, which is based on the initial guess of the real time requirement.

# **CHAPTER NINE: RECOMMENDATIONS AND CONCLUSIONS**

## 9.1 Recommendations

Although our project accomplishes most of the task set by requirements, further more innovative improvements can be achieved.

1. This software can't recognize the handwritten text hence this improvement can be incorporated in the future developments.
2. Large attention and care has been taken to make this project OS independent, but the problem still lies while interfacing the camera in the different platform. This problem can be removed by using a new vlcj binding i.e. of version 2.2.0 which can even display the list of video device attached.
3. This software is language dependent as no other language than English is accepted. This problem can be removed by training the neural network with the different languages to be recognized.
4. This software doesn't give the accurate result unless trained for all the possible test cases, which is impossible to achieve. But then effort can be made to minimize the error by giving the experienced user the chance to train the neural network.
5. The final outcome can be transformed into audio or the output can be made available to be saved for future use.

## 9.2 Conclusion

Character Recognition is the process of taking text image or handwritten text as an input and converting it to digital form. This is done by a “training” and “Recognition” process in software. The software uses existing algorithms to try to decipher characters. Once a match has been found, it is manually checked for accuracy. If there are any discrepancies, it will be corrected and the software will save that character for future matching. The more characters in the pool, the more accurate it becomes.

Live! Character Recognition is a very helpful tool, with lots of implementation possibilities. What you otherwise take a lot of time to do manually, is a relatively quick process. Many digital copies come from larger, more funded sources. Live! Character Recognition allows for any user to convert a document accurately.

This software was very accurate for extracting text from video images that were created using the standards formats and better resolutions. When other formats of images or the images containing different language were used, however, the accuracy of the software declined significantly.

# BIBLIOGRAPHY

- [1] M. Breithaupt, "Improving ocr and icr accuracy through expert voting," *Oce Document Technologies*, 2001.
- [2] G. Nagy, T. A. Narktker and S. V. Rice, "Optical character recognition: An illustrated guide to the frontier," vol. 3967, pp. 58-69, 1999.
- [3] S. V. Rice, F. R. Jenkins and T. A. Nartker, "The fifth annual test of ocr," *Information Science Research Institute*, 1996.
- [4] L. Xu, "Kanji Character Detection from Complex Real Scene Images based on Character Properties," in *IEEE conference*, 2008.
- [5] X. Liu and J. Samarabandu, "Multiscale Edge Based Text Extraction From Complex Images," in *IEEE conference*, 2006.
- [6] X. Liu and J. Samarabandu, "An edge based text recognition and extraction algorithm for indoor mobile robot navigation," in *IEEE conference*, 2005.
- [7] A. C. Bovik, "Introduction to Digital Image and Video Processing," in *Handbook of Image and Video Processing*, Ed. Elsevier Academic Press, 2005.
- [8] S. Kumari, "SlideShare," March 2010. [Online]. Available: <http://www.slideshare.net/jassics/real-time-image-processing-8587749>. [Accessed 25 September 2012].
- [9] AIM Inc., "AIM," 2000. [Online]. Available: <http://www.aimglobal.org/technologies/othertechnologies/ocr.pdf>. [Accessed 25 September 2012].
- [10] Y. Zong, K. Karu and A. K. Jain, " Locating text in complex color images," *Pattern Recognition*, vol. 10, pp. 1523-1536, 1995.
- [11] M. A. Smith and T. Kanade, "Video skimming for quick browsing based on

audio and image characterization," Carnegie Mellon University, 1995.

- [12] V. K. Y. Wu, R. Manmatha and E. M. Riseman, "Finding text in images," in *ACM International Conference on Digital Libraries*, 1997.
- [13] C. Garcia and X. Apostolidis, "Text detection and segmentation in complex color images," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2000.
- [14] K. Sobottka, H. Bunke and H. Kronenberg, "Identification of text on colored book and journal covers," in *International Conference on Document Analysis and Recognition*, 1999.
- [15] H. Li and D. Doermann, "Text enhancement in digital video using multiple frame integration," in *ACM Multimedia*, 1999.
- [16] R. Lienhart and A. Wernicke, "Localizing and segmenting text in images and videos," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, pp. 256-268, 2002.
- [17] J. Heaton, "Heaton Research," 23 12 2007. [Online]. Available: <http://www.heatonresearch.com/articles/6/page2.html>. [Accessed 25 September 2012].
- [18] Caprica Software Limited, "vlcj," [Online]. Available: <http://www.capricasoftware.co.uk/vlcj/tutorial1.php>. [Accessed 25 September 2012].
- [19] Wayne Rasband, "ImageJ," [Online]. Available: <http://rsbweb.nih.gov/ij/docs/examples/index.html>. [Accessed 25 September 2012].

# APPENDIX

## User Manual

