Pakistan Sign Language Recognition System (Using LBP and SVM)

Image Processing

Developed By

Ch. Muhammad Awais	**2695-FBAS/BSCS/F-13**
Nouman Dilshad	**2709-FBAS/BSCS/F-13**

Supervised By

Syed Muhammad Saqlain Shah

Assistant Professor

Department of Computer Science and Software Engineering

Faculty of Basic and Applied Sciences International Islamic University, Islamabad

2017

بسم الله الرحمن الرحيم

Final Approval

This is to certify that Ch. Muhammad Awais, Registration No. 2695-FBAS/BSCS/F13 and Nouman Dilshad 2739-FBAS/BSCS/F13 has successfully completed the final project naming **Pakistan Sign Language Recognition System** (Using LBP and SVM) to fulfill the partial requirements of the degree of Bachelors in Computer Science.

COMMITTEE

External Examiner

FBAS, IIUI ____________________

Internal Examiner

FBAS, IIUI ____________________

Supervisor

Syed Muhammad Saqlain Shah

Assistant Professor

DCS&SE,

FBAS, IIUI ____________________

Dissertation

A dissertation is submitted to the

Department of Computer Science & Software Engineering

International Islamic University Islamabad

as a partial fulfillment of the requirement

for the award of the degree of **BS In Computer Science**.

Dedicated to

With the blessings of Allah (S.W.T) who is the biggest source of Knowledge.
To His Messenger and the Teacher for the whole world, The Prophet Muhammad (P.B.U.H),
To our beloved parents and all respected teachers.

Declaration

We, hereby declare that this software, neither as a whole nor as a part has been copied out from any source. It is further declared that we have developed this software and accompanied report entirely on the basis of our personal efforts. If any part of this project is proved to be copied out from any source or found to be reproduction of some other. We will stand by the consequences. No portion of the work presented has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Ch. Muhammad Awais **(2695-FBAS-BSCS-F13)**

Nouman Dilshad **(2709-FBAS-BSCS-F13)**

Acknowledgement

We would like to express our special thanks to our supervisor **Syed Muhammad Saqlain Shah** who gave us the golden opportunity to do this wonderful project on the topic **“Pakistan Sign Language Recognition System Using LPB and SVM”** which also helped us in doing a lot of research and we came to know about so many new things.

We would like to pay our sincere thanks to all of our teachers **Dr. Hussain Naqvi , Mr. Asim Munir, Mr. Muhammad Nadeem, Mr. Imran Saeed, Dr. Adeel , Mr. Imran Khan, Mr. Waseem, Dr Qasir Javaid, Dr Rashid Bin Mohammad, Imran Khan, Professor Mohammad Sher, Mr. Sher Afghan.** This Is all because of them as they taught us very informative and interesting courses that proved worthy to improve our skills. We are also thankful to the staff of Department of Computer Science of International Islamic University Islamabad. We are very grateful to all our friends and our class fellows, thank you for your understanding and encouragement in our many moments of crisis.

This thesis is a First Step of our journey to success.
Thanks to google, stack overflow, Plural sight, youtube and the entire site we use to learn about computer science.

Ch. Muhammad Awais (2695-FBAS-BSCS-F13)

Nouman Dilshad (2709-FBAS-BSCS-F13)

Project in Brief

Project Title:	Pakistan Sign Language Recognition System Using LPB and SVM **(**Image Processing)
Objectives:	Pakistan has its own sign language for special people our aim is to build a software which can recognize the signs through image and tell the user which sign is it.
Undertaken By:	Ch. Muhammad Awais Nouman Dilshad
Supervised By:	Syed Muhammad Saqlain Shah
Date Started:	Feb,2017
Date Completed:	July, 2017
Tools:	Anaconda IDE, Python IDLE
Technologies:	Python 2.7 Open CV
And Languages Used:	Python
Platform used:	Windows
System used:	Core i5 2.30 GHz, Ram 8 GB

Abstract

Wherever communities of deaf people exist, sign languages have developed, and are at the cores of local deaf cultures. Although signing is used primarily by the deaf, it is also used by others, such as people who can hear but cannot physically speak, or have trouble with spoken language due to some other disability (augmentative and alternative communication.

The motivation of this project comes from those children's unable to hear or speak. The technology is rapidly changing day by day but we are still on that same old fashion technique for understanding the signs language. This project is an initial step toward the goal of achieving the same modern technique of sign language recognition for Pakistan.

Our objective is to make a system which can read the image and outputs the result. For example some input an Image making the sign of any alphabet the system will read it and process it and output the actual alphabet.

Table of Contents

Acronyms and Abbreviations

Term	**Definition**
PSL	Pakistan Sign Language
LBP	Local Binary Pattern
SVM	Support Vector Machine

Chapter 1

Introduction

1.1 Image Processing

Image processing is processing of images using mathematical operations by using any form of signal processing for which the input is an image, a series of images or a video, such as a photograph, video frame, the output of image processing may be either an image or a set of characteristics or parameters related to the image.

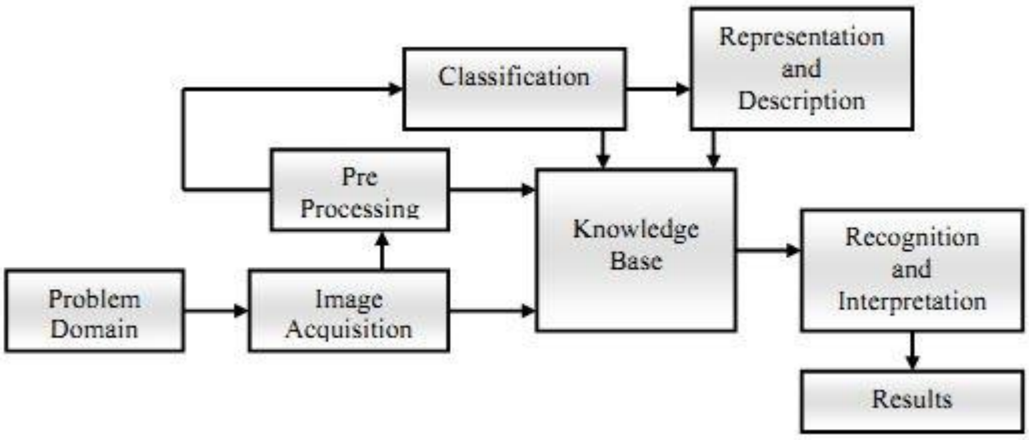
1.1.1 Brief Detail

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps:

- Importing the image via image acquisition tools;
- Analyzing and manipulating the image;
- Output in which result can be altered image or report that is based on image analysis.

There are two types of methods used for image processing namely, analogue and digital image processing. Analogue image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are pre-processing, enhancement, and display, information extraction.



1.1.2 Purpose of Image Processing

The purpose of image processing is divided into 5 groups. They are:

- Visualization - Observe the objects that are not visible.
- Image sharpening and restoration - To create a better image.
- Image retrieval - Seek for the image of interest.
- Measurement of pattern – Measures various objects in an image.
- Image Recognition – Distinguish the objects in an image.

Types

The two types of **methods used for Image Processing** are **Analog and Digital** Image Processing. Analog or visual techniques of image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. The image processing is not just confined to area that has to be studied but on knowledge of analyst. Association is another important tool in image processing through visual techniques. So analysts apply a combination of personal knowledge and collateral data to image processing.

i. **Analog Processing** is any image processing task conducted on two-dimensional analog signals by analog means (as opposed to digital image processing). Basically any data can be represented in two types named as 1.Analog 2.Digital if the pictorial representation of the data represented in analog wave formats that can be named as analog image. E.g television broadcasting in older days through the dish antenna systems.. Whereas the digital representation or storing the data in digital form is termed as a digital image processing e.g. image data stored in digital logic gates.

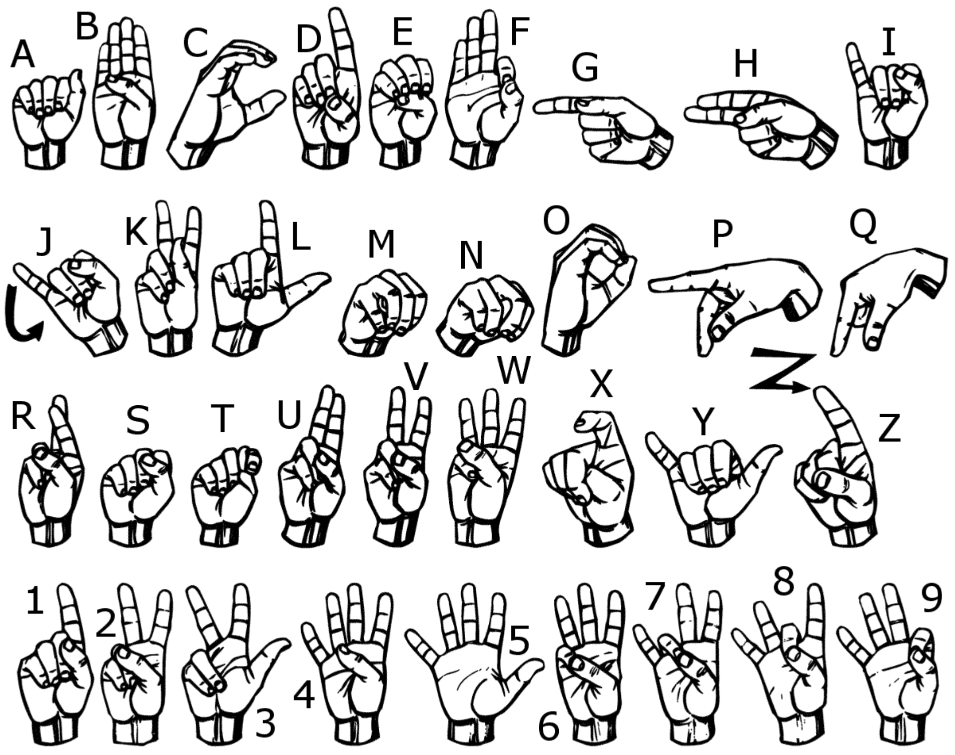
ii. **Digital Processing** techniques help in manipulation of the digital images by using computers. As raw data from imaging sensors from satellite platform contains deficiencies. To get over such flaws and to get originality of information, it has to undergo various phases of processing. The three general phases that all types of data have to undergo while using digital technique are Pre- processing, enhancement and display, information extraction.

1.2 Sign language

Sign language (also **signed language**) is a language which chiefly uses manual communication to convey meaning, as opposed to spoken language. This can involve simultaneously combining hand shapes, orientation and movement of the hands, arms or body, and facial expressions to express a speaker's thoughts. Sign languages share many similarities with spoken languages (sometimes called "oral languages"), which depend primarily on sound, and linguists consider both to be types of natural language. Although there are some significant differences between signed and spoken languages, such as how they use space grammatically, sign languages show the same linguistic properties and use the same language faculty as do spoken languages. They should not be confused with body language, which is a kind of non-linguistic communication.

Wherever communities of deaf people exist, sign languages have developed, and are at the cores of local deaf cultures. Although signing is used primarily by the deaf, it is also used by others, such as people who can hear but cannot physically speak, or have trouble with spoken language due to some other disability (augmentative and alternative communication).

It is not clear how many sign languages there are. A common misconception is that all sign languages are the same worldwide or that sign language is international. Aside from the pidgin International Sign, **each country generally has its own**, native sign language, and some have more than one (although there are also substantial similarities among all sign languages). The 2013 edition of Ethnologue lists 137 sign languages. Some sign languages have obtained some form of legal recognition, while others have no status at all.



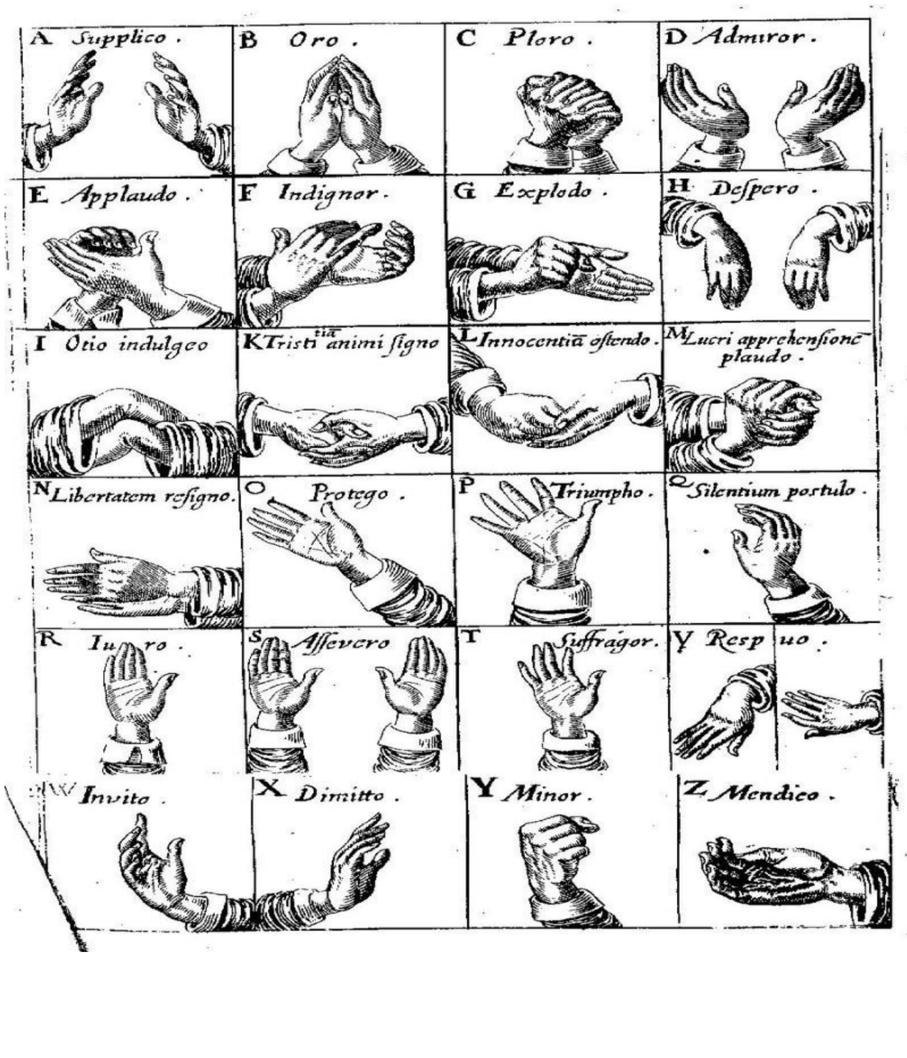
1.2.1 History

Groups of deaf people have used sign languages throughout history. One of the earliest written records of a sign language is from the fifth century BC, in Plato's *Cratylus*, where Socrates says: "If we hadn't a voice or a tongue, and wanted to express things to one another, wouldn't we try to make signs by moving our hands, head, and the rest of our body, just as dumb people do at present?

Until the 19th century, most of what we know about historical sign languages is limited to the manual alphabets (fingerspelling systems) that were invented to facilitate transfer of words from a spoken language to a sign language, rather than documentation of the language itself. Pedro Ponce de León (**1520–1584**) is said to have developed the first manual alphabet.

In **1620**, **Juan Pablo Bonet** published *Reducción de las letras y arte para enseñar a hablar a los mudos* (‘Reduction of letters and art for teaching mute people to speak’) in Madrid. It is considered the first modern treatise of sign language phonetics, setting out a method of oral education for deaf people and a manual alphabet.

In Britain, manual alphabets were also in use for a number of purposes, such as secret communication, public speaking, or communication by deaf people. In **1648**, **John Bulwer** described "Master Babington", a deaf man proficient in the use of a manual alphabet, "contryved on the joynts of his fingers", whose wife could converse with him easily, even in the dark through the use of tactile signing.



In **1680**, **George Dalgarno** published ***Didascalocophus****, or, The deaf and dumb man's tutor*, in which he presented his own method of deaf education, including an "arthrological" alphabet, where letters are indicated by pointing to different joints of the fingers and palm of the left hand. Arthrological systems had been in use by hearing people for some time, some have speculated that they can be traced to early Ogham manual alphabets.

The vowels of this alphabet have survived in the contemporary alphabets used in British Sign Language, Auslan and New Zealand Sign Language. The earliest known printed pictures of consonants of the modern two-handed alphabet appeared in **1698** with ***Digiti Lingua*** (Latin for *Language* [or *Tongue*] *of the Finger*), a pamphlet by an anonymous author who was himself unable to speak. He suggested that the manual alphabet could also be used by mutes, for silence and secrecy, or purely for entertainment. Nine of its letters can be traced to earlier alphabets, and 17 letters of the modern two-handed alphabet can be found among the two sets of **26 handshapes** depicted.

Charles de La Fin published a book in **1692** describing an alphabetic system where pointing to a body part represented the first letter of the part (e.g. Brow=B), and vowels were located on the fingertips as with the other British systems. He described codes for both English and Latin.

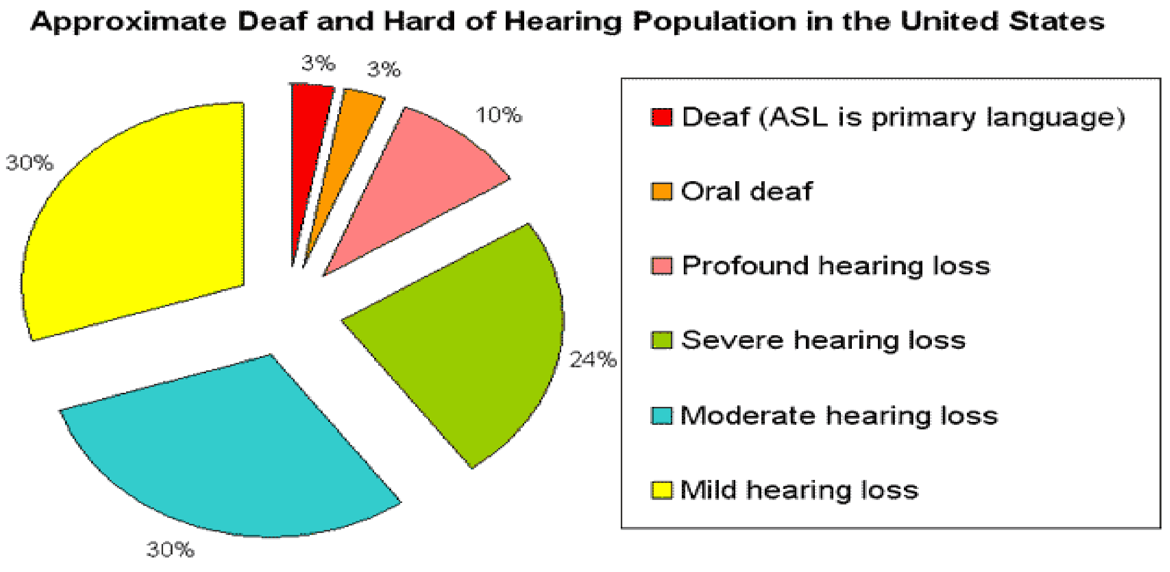
By **1720**, the British manual alphabet had found more or less its present form. Descendants of this alphabet have been used by deaf communities (or at least in classrooms) in former British colonies India, Australia, New Zealand, Uganda and South Africa, as well as the republics and provinces of the former Yugoslavia, Grand Cayman Island in the Caribbean, Indonesia, Norway, Germany and the USA.

Frenchman Charles-Michel de l'Épée published his manual alphabet in the 18th century, which has survived basically unchanged in France and North America until the present time. In 1755, Abbé de l'Épée founded the first school for deaf children in Paris; Laurent Clerc was arguably its most famous graduate. Clerc went to the United States with Thomas Hopkins Gallaudet to found the American School for the Deaf in Hartford, Connecticut, in 1817. Gallaudet's son, Edward Miner Gallaudet founded a school for the deaf in 1857 in Washington, D.C., which in 1864 became the National Deaf-Mute College. Now called Gallaudet University, it is still the only liberal arts university for deaf people in the world.

1.2.2 Statistics about Sign language

Almost ten percent of people are either completely deaf or hard of hearing. Hearing loss may come at any age and can cause frustration and isolation, as well as other more difficult emotions. Learning facts and statistics on deafness and sign language can be beneficial in dealing with such tricky emotions. This can be highly encouraging as people will start breaking down barriers that were created due to their hearing problem. After learning about these facts and statistics, you will realize that you are not alone.
The National Institute on Deafness and Other Communication Disorders maintains records on most disabilities and disorders. Here are some surprising statistics related to deafness and loss of hearing, or in layman terms, deafness facts in regards to the U.S. population.

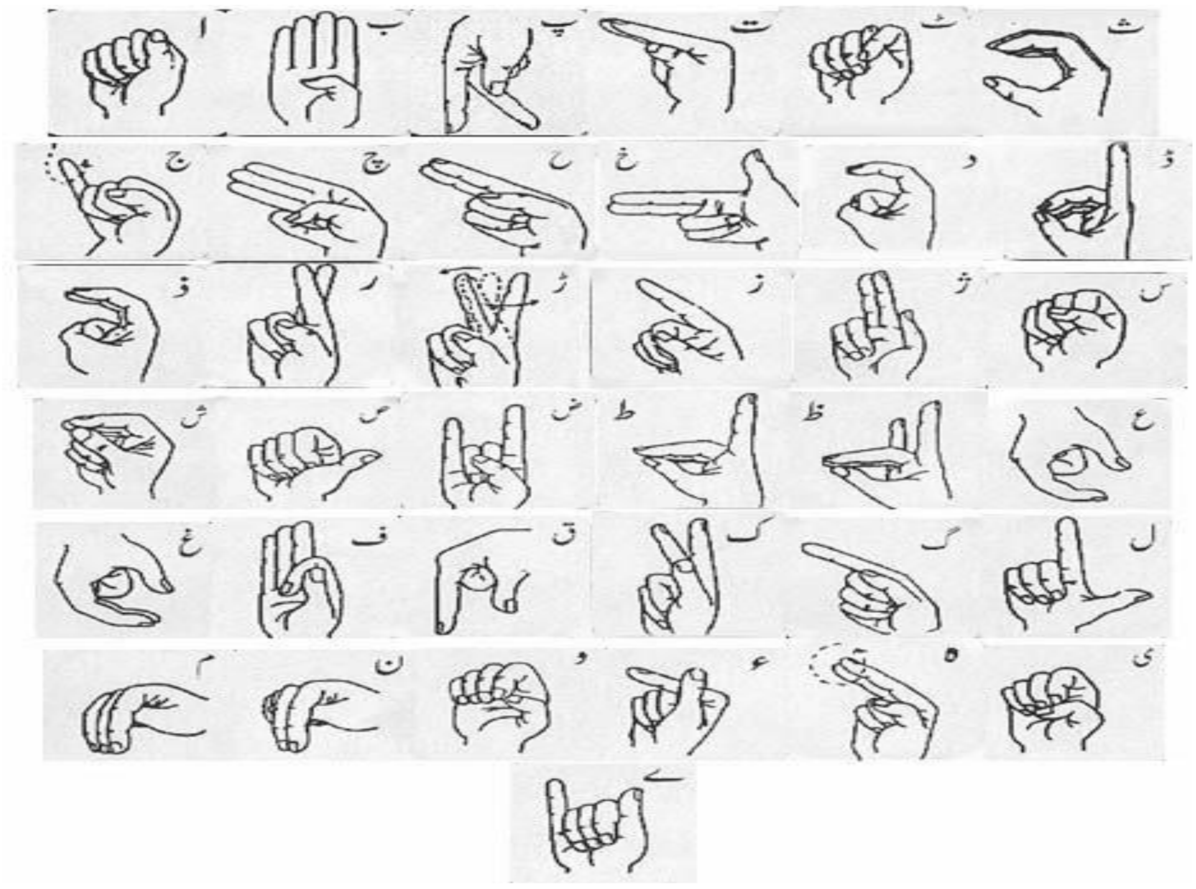
- Hearing loss is more common in men than in women.
- Approximately seventeen percent (thirty six million) of American adults are dealing with some degree of hearing loss.
- Age and reported hearing loss has a strong relationship: 47 percent of adults 75 years old or older, 30 percent of adults 65-74 years old, and 18 percent of adults 45-64 years old have a hearing problem.
- About 12.3 percent of men and 14 percent of women, aged 65 or older, are affected by tinnitus.
- More than 15 percent of adults have high frequency of hearing loss just because they are frequently exposed to loud noises either at work or while performing leisure activities.
- Many children (approximately ¾) experience ear infections by the time they are three years old.
- Approximately 4,000 people are diagnosed with sudden deafness each year in the U.S., and only 10 or 15 percent of them know what caused the sudden deafness.
- Sign language is commonly used among deaf people for expressing their thoughts and emotions, but most of them use a different form of sign language, for example, ASL, BSL, LSQ and so on.



1.3 Pakistan Sign Language

Pakistan has a deaf population of 0.24 million, which is approximately 7.4% of the overall disabled population in the country.

There are many native sign languages. Pakistan has its own sign language known as Pakistan Sign Language. There are 36 alphabets in PSL and 33 among them are static and 3 continuous. All alphabets in Pakistan sign language are right handed signs. Currently the people who know the language can understand can communicate with deaf people there is no working solution to this in the form of software



Alphabets of Pakistan Sign Language

1.4 Concept

Signs can be described at the sub-unit level using phonemes. These encode different elements of a sign. Unlike speech they do not have to occur sequentially, but can be combined in parallel to describe a sign. Studies of American Sign Language (ASL) by Liddell and Johnson model sign language on the movement-hold system. Signs are broken into sections where an aspect is changing and sections where a state is held steady. This is in contrast to the work of Stokoe where different components of the sign are described in different channels; the motion made by the hands, the place at which the sign is performed, the hand shapes, the relative arrangement of the hands and finally the orientation of both the hands and fingers to explain the plane in which the hands sit. Both of these models are valid in their own right and yet they encode different aspects of sign. Within SLR both the movement hold, sequential information from Liddell and Johnson and the parallel forms of Stokoe are desirable annotations.

Below are described a small subset of the constructs of sign language. There is not room here to fully detail the entire structure of the language; instead the focus is on those that pose significant challenges to the field of SLR:

a. Adverbs modifying verbs; signers would not use two signs for ‘run quickly’ they would modify the sign for run by speeding it up.
b. Non-manual features (NMFs); facial expressions and body posture are key in determining the meaning of sentences, e.g. eyebrow position can determine the question type. Some signs are distinguishable only by lip shape, as they share a common manual sign.
c. Placement; pronouns like ’he’, ’she’ or ’it’ do not have their own sign; instead the referent is described and allocated a position in the signing space. Future references point to the position, and relationships can be described by pointing at more than one referent.
d. Classifiers; these are hand shapes which are used to represent classes of objects; they are used when previously described items interact. E.g. to distinguish between a person chasing a dog and vice versa.
e. Directional verbs; these happen between the signer and referent(s), the direction of motion indicates the direction of the verb. Good examples of directional verbs are ‘give’ and ‘phone’. The direction of the verb implicitly conveys which nouns are the subject and object.
f. Positional Signs; where a sign acts on the part of the body descriptively. e.g. ‘bruise’ or ‘tattoo’.
g. Body Shift; represented by twisting the shoulders and gaze, often used to indicate role-shifting when relating a dialogue.
h. Iconicity; when a sign imitates the thing it represents, it can be altered to give an appropriate representation. E.g. the sign for getting out of bed can be altered between leaping out of bed with energy to a recumbent that is reluctant to rise.
i. Finger spelling; where a sign is not known, either by the signer or the recipient, the local spoken word for the sign can be spelt explicitly by finger spelling.

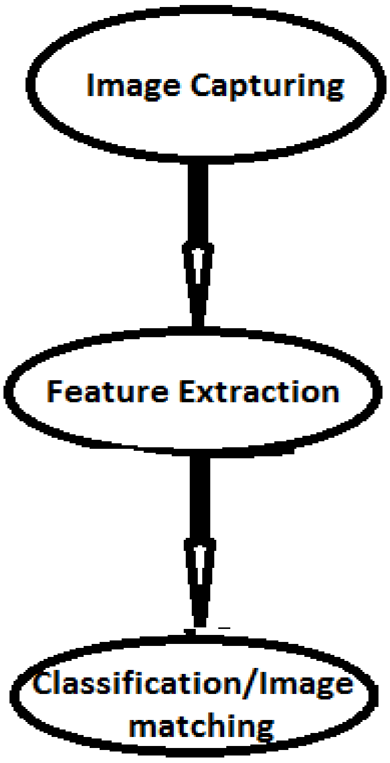
1.5 Existing Systems

The recognition of sign language appeared almost 20 years behind speech recognition. Also, sign language processing is not yet completely explored. Understanding sign language requires better linguistic knowledge, but until now there are no general rules that define the signing from a linguistic point of view. The first scientific publications in the field of sign language recognition emerged in the beginning of the 90s. Most applications presented in previous works do not operate in real-time and need up to 20 seconds after the sign production to complete the processing. There is a scarcity of published work which provides details on camera hardware and resolution. Generally, most proposed approaches suggest the use of professional hardware, optimal camera placement, low noise and high resolution. Magnetic or optical markers on hands and face facilitate the determination of manual configuration and facial expression. However, this method is restrictive and unnatural for the user. Furthermore, data gloves, which measure the flexion of the finger joints, are undesirable for practical systems because of their high cost. Some existing systems process continuous production of signs but their vocabulary is not large. To improve the recognition rate, the exploitation of grammar and context is necessary.

1.6 Problem Statement

Problem statement is to provide software for recognition of Pakistan sign language statics alphabets.

1.7 Proposed System



1.7.1 Image/Video Capturing

- Via camera
- Via file

1.7.2 Feature Extraction

We will extract the local binary patterns as salient features for classifying Pakistan Sign Language alphabets. Local binary patterns (LBP) are a type of visual .

Descriptor used for classification in computer vision. The Local Binary Patterns algorithm has its roots in 2D texture analysis. The basic idea is to summarize the local structure in an image by comparing each pixel with its neighborhood. Take a pixel as center and threshold its neighbors against. If the intensity of the center pixel is greater-equal its neighbor, then denote it with 1 and 0 if not. You'll end up with a binary number for each pixel, just like 11001111. With 8 surrounding pixels you'll end up with 2^8 possible combinations, which are called Local Binary Patterns or sometimes abbreviated as LBP codes. The LBP feature vector, in its simplest form, is created in the following manner:

- Divide the examined window into cells (e.g. 16x16 pixels for each cell).
- For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
- Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).
- Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.
- Optionally normalize the histogram.
- Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.

1.7.3 Classification Feature Matching

Euclidean Distance

The Euclidean distances between any points can be used to compute a histogram. The Shannon entropy of this histogram is defined as distributive entropy of Euclidean distance (DEED). The value of DEED provides the uncertainty information concerning feature vectors. For a mutual classification problem, the distributive entropy of Euclidean distance between any sample and the sample mean of the same class is defined as within-class DEED (WCDEED). The distributive entropy of Euclidean distance between any sample in one class and the sample mean of other class is defined as between-class DEED (BCDEED). Theoretically, reparability of features in training samples will be enhanced when the mapped points in Euclidean space are more consistently convergence to the center point. However, if the distribution of all mapped points in Euclidean space is diffusive, it should be difficult to achieve a good classification performance by using such training samples.

Hausdorff Distance

The Hausdorff distance measures the extent to which each point of a model set lies near some point of an image set and vice versa. Thus, this distance can be used to determine the degree of resemblance between two objects that are superimposed on one another. Efficient algorithms for computing the Hausdorff distance between all possible relative positions of a binary image and a model are presented. The focus is primarily on the case in which the model is only allowed to translate with respect to the image. The techniques are extended to rigid motion. The Hausdorff distance computation differs from many other shape comparison methods in that no correspondence between the model and the image is derived. The method is quite tolerant of small position errors such as those that occur with edge detectors and other feature extraction methods. It is shown that the method extends naturally to the problem of comparing a portion of a model against an image.

1.8 Project Goal

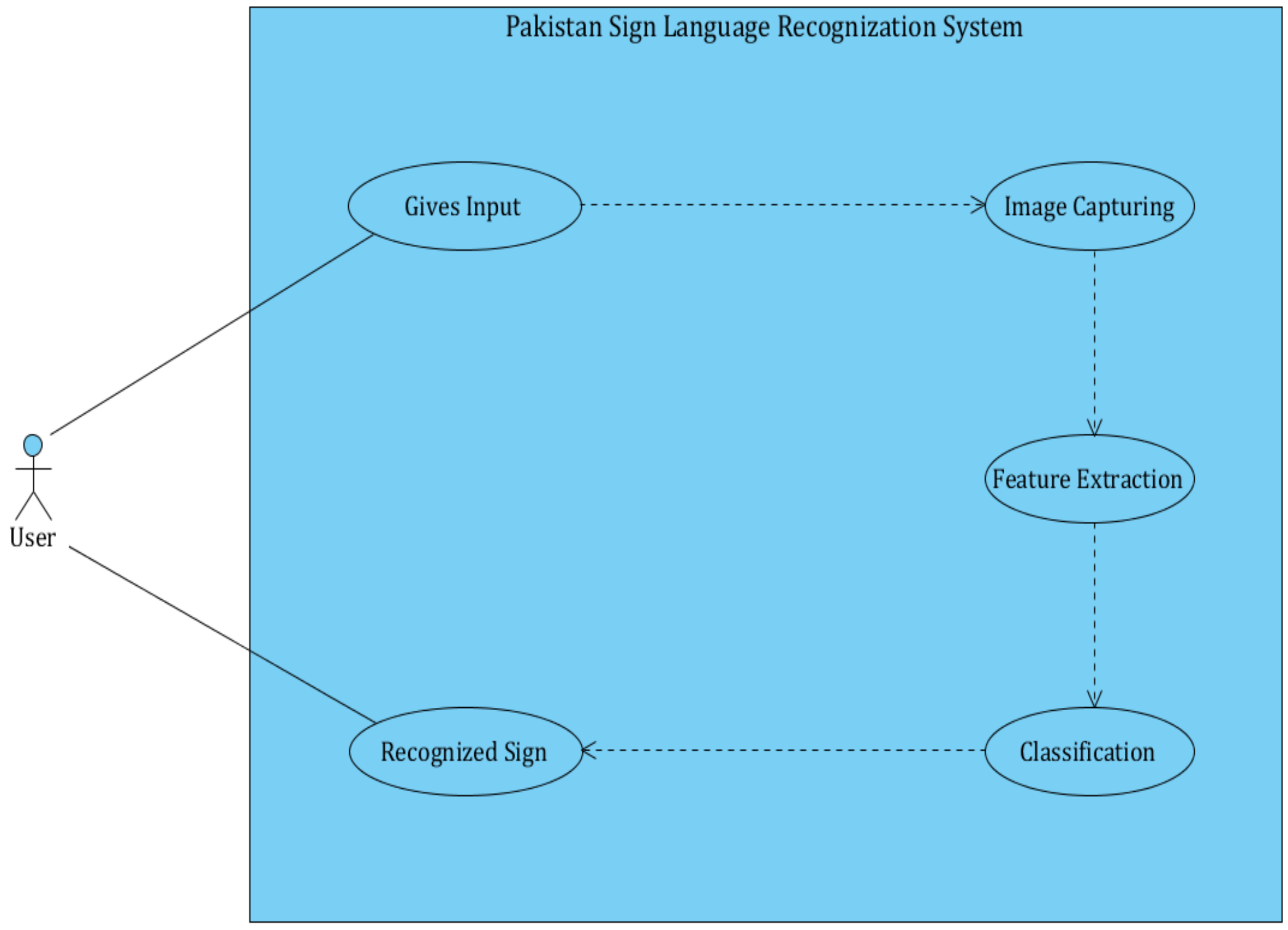
The goals of the project are:

- To make an environment for deaf people to understand and to be understood.
- To promote PSL.
- To make desktop application
- To make an environment of communication between deaf people and other people.

Chapter 2
System Analysis

2.1 Use Case Model

2.1.1 Use Case Diagram



2.1.2 Use Case Descriptions in Brief Format

The application will ask the user to give input in the form of video or image after then it will process the image capture the hand and extract the features of it then classify it and after that it will show the recognized sign.

2.1.3 Use Case Descriptions in Detailed Expanded Format

2.1.3.1 Input Image

The input can be in form of video or image and can be uploaded in the application via live camera or already recorded image/video.

2.1.3.2 Hand Recognition

The application will process the uploaded file process it and capture the hand from the image after then it will send the snaps of hand to the application for further processing.

2.1.3.3 Feature Extraction:

Feature Extraction is the process extracting the features of the image using an algorithm know as local binary pattern.

2.1.3.4 Feature Classification and Feature Matching

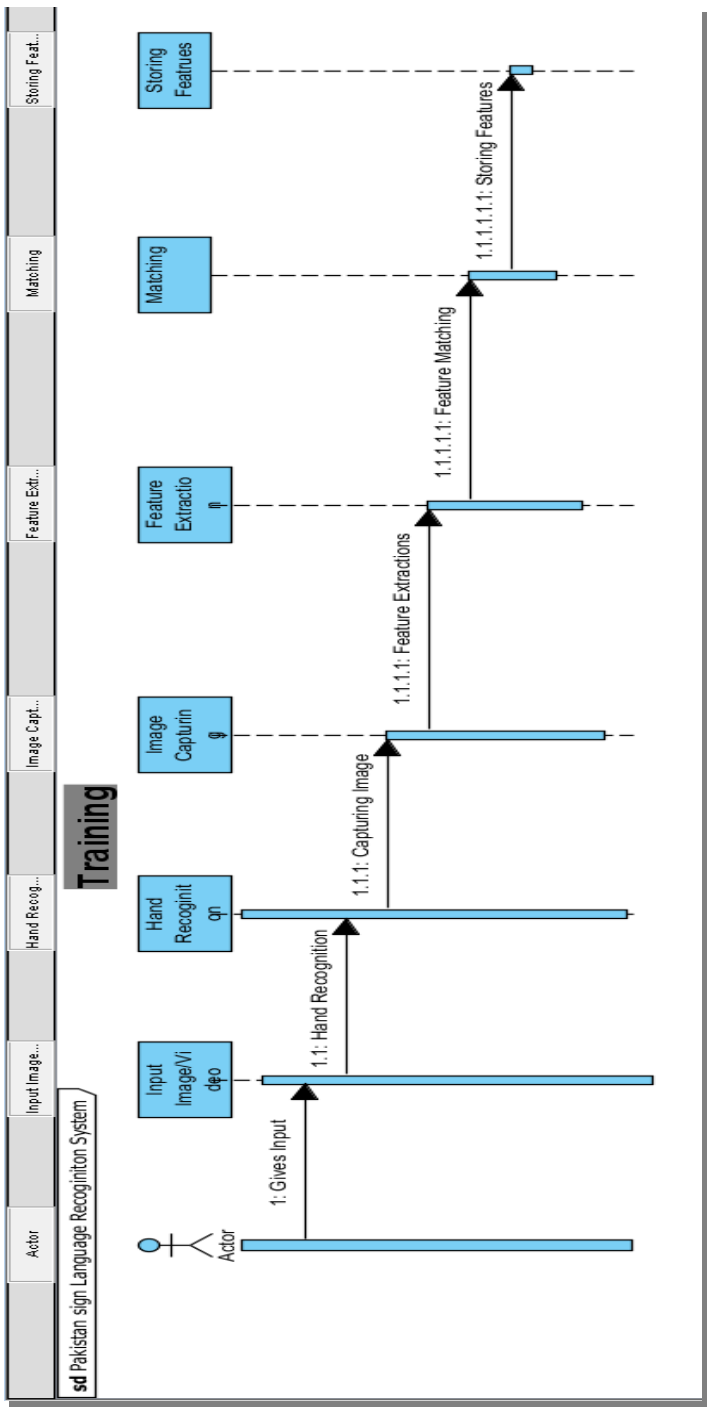
The application will match the features with the previously tested data in case of testing or live application but while training it will calculate the features classify them using supervised learning and unsupervised learning and store them for later purposes. We used Support Vector Machine for this purpose..

2.1.3.5 Output:

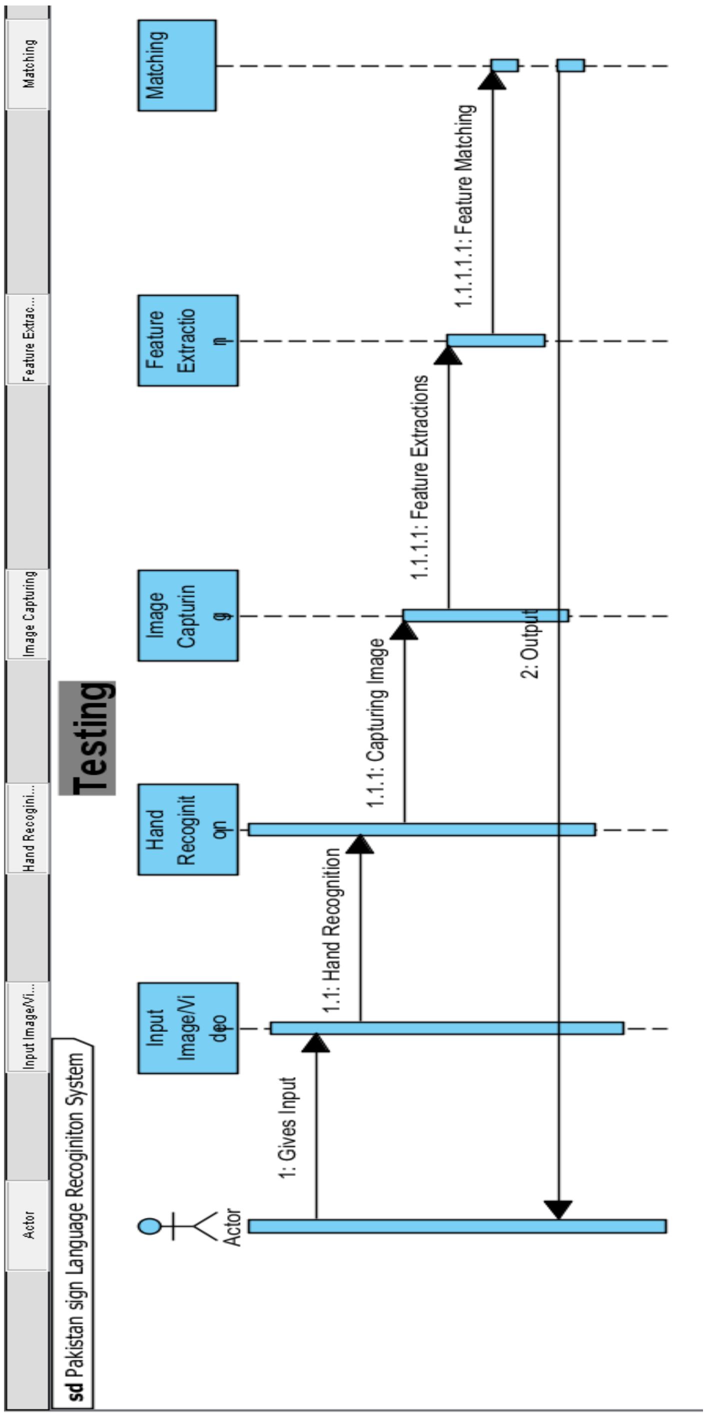
The output will be in the form of the recognized sign the user will get to know the alphabet of the sign.

2.1.4 System Sequence Diagram

2.1.4.1 Training Sequence Diagram



2.1.4.2 Testing/Live Application Sequence Diagram



Chapter 3

System Design

The Software design that show interaction of the user to the system. It is the necessary part for system. After carefully analyzing the requirements and the functionality of the system, had to analyze the system design. System design is related to how the system works. The working of the system is to visualize for the sake of analyzing the flow of system. System design is used to determine the relationship between components and identify the component dependencies. It is the process of solving problems related to the system and planning for a software solution.

The goal of system design is to establish a design approach that provides the functions that are described in the system requirements. System design will establish a discipline and integrated engineering plan for the proposed design, understand the technical risks, and determine estimates for performance and cost to completion. In the system design, a design subject is represented in the form of conceptual entities and their relationships.

Some diagrams are included here to describe the flow of the system:

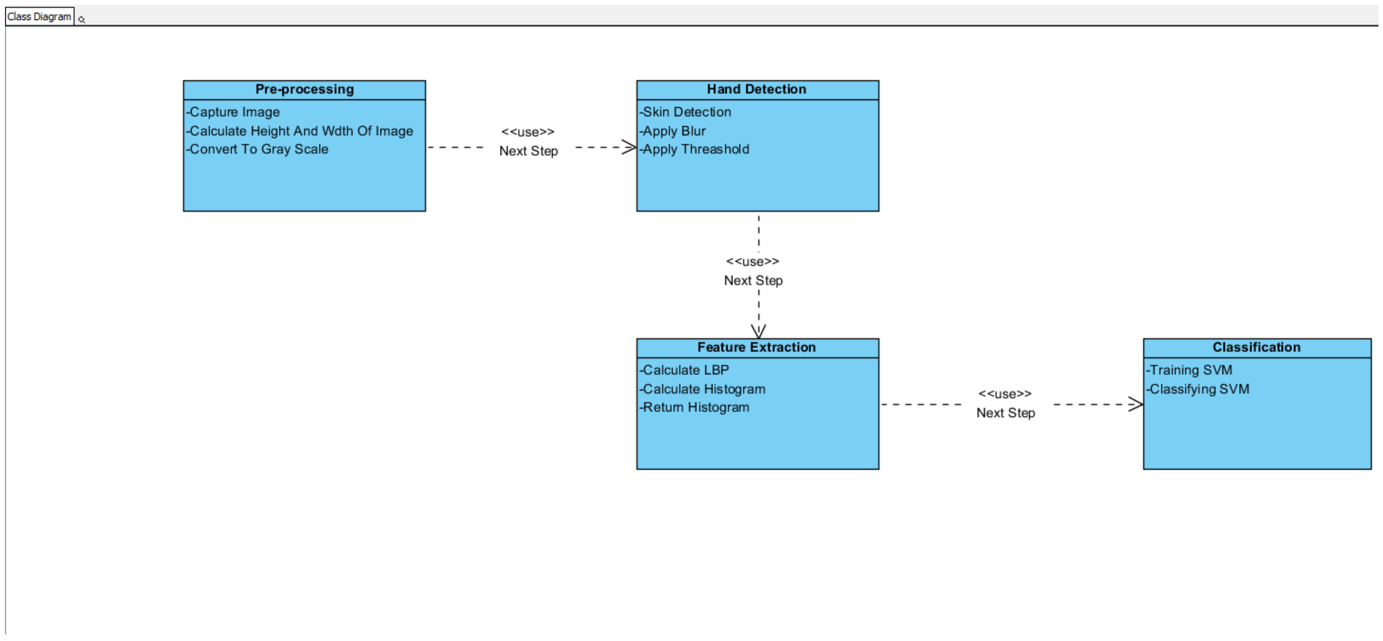
3.1 Interaction Diagrams (Sequence or Communication)

A sequence diagram is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message sequence chart. A sequence diagram shows object interactions arranged in time sequence. Sequence diagrams are sometimes called event diagrams or event scenarios.

3.2 Design Class Diagram

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematics of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are represented with boxes which contain three parts.

- The upper part holds the name of the class.
- The middle part contains the attributes of the class.
- The bottom part gives the methods or operations the class can take or undertake.



Chapter 4
Implementation

4.1 Local Binary Pattern

Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. Due to its discriminative power and computational simplicity, LBP texture operator has become a popular approach in various applications. It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused, for example, by illumination variations. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings.

Local binary patterns are a type of visual descriptor used for classification in computer vision. LBP is the particular case of the Texture Spectrum model proposed in 1990. LBP was first described in 1994. It has since been found to be a powerful feature for texture classification; it has further been determined that when LBP is combined with the Histogram of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets. A comparison of several improvements of the original LBP in the field of background subtraction was made in 2015 by Silva et al. A full survey of the different versions of LBP can be found in Bouwmans et al.

Local Binary Patterns, or LBPs for short, are a texture descriptor made popular by the work of Ojala et al. in their 2002 paper, *Multiresolution Grayscale and Rotation Invariant Texture Classification with Local Binary Patterns* (although the concept of LBPs were introduced as early as 1993).Unlike Haralick texture features that compute a *global representation* of texture based on the Gray Level Co-occurrence Matrix, LBPs instead compute a *local representation* of texture. This local representation is constructed by comparing each pixel with its surrounding neighborhood of pixels.

The first step in constructing the LBP texture descriptor is to convert the image to grayscale. For each pixel in the grayscale image, we select a neighborhood of size *r* surrounding the center pixel. A LBP value is then calculated for this center pixel and stored in the output 2D array with the same width and height as the input image.

For example, let's take a look at the original LBP descriptor which operates on a fixed *3 x 3*neighborhood of pixels just like this:

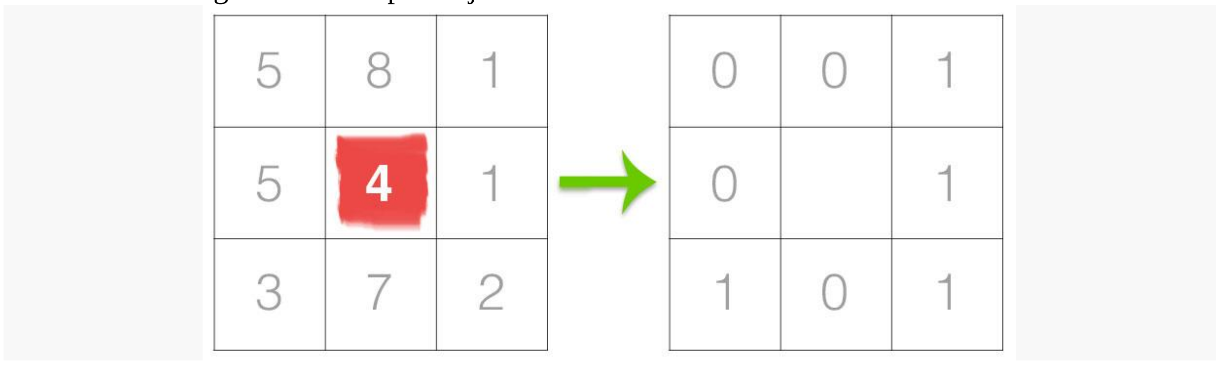


Figure 1: The first step in constructing a LBP is to take the 8 pixel neighborhood surrounding a center pixel and threshold it to construct a set of 8 binary digits.

In the above figure we take the center pixel (highlighted in red) and threshold it against its neighborhood of 8 pixels. If the intensity of the center pixel is greater-than-or-equal to its neighbor, then we set the value to *1*; otherwise, we set it to *0*. With 8 surrounding pixels, we have a total of *2 ^ 8 = 256* possible combinations of LBP codes.

From there, we need to calculate the LBP value for the center pixel. We can start from any neighboring pixel and work our way clockwise or counter-clockwise, but our ordering must be kept *consistent* for all pixels in our image and all images in our dataset. Given a *3 x 3*neighborhood, we thus have 8 neighbors that we must perform a binary test on. The results of this binary test are stored in an 8-bit array, which we then convert to decimal, like this:

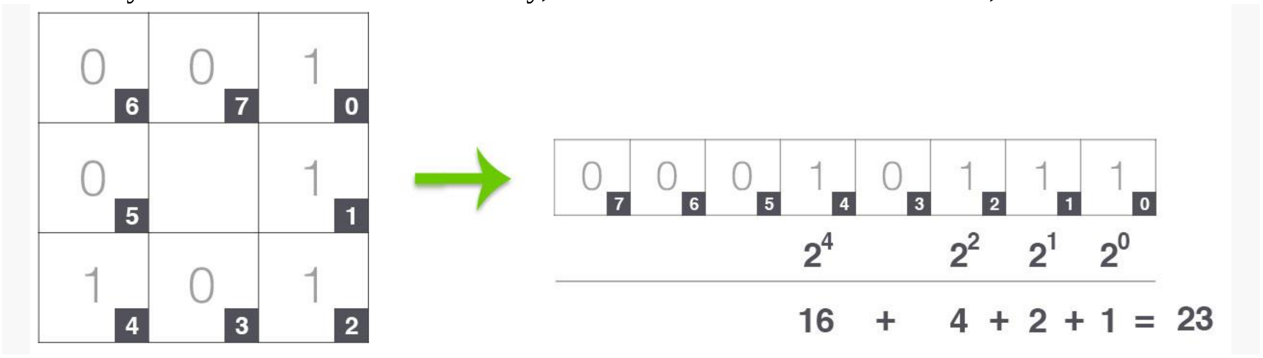


Figure 2: Taking the 8-bit binary neighborhood of the center pixel and converting it into a decimal representation. (Thanks to Bikramjot of Hanzra Tech for the inspiration on this visualization!)

In this example we start at the top-right point and work our way ***clockwise*** accumulating the binary string as we go along. We can then convert this binary string to decimal, yielding a value of 23.

This value is stored in the output LBP 2D array, which we can then visualize below:

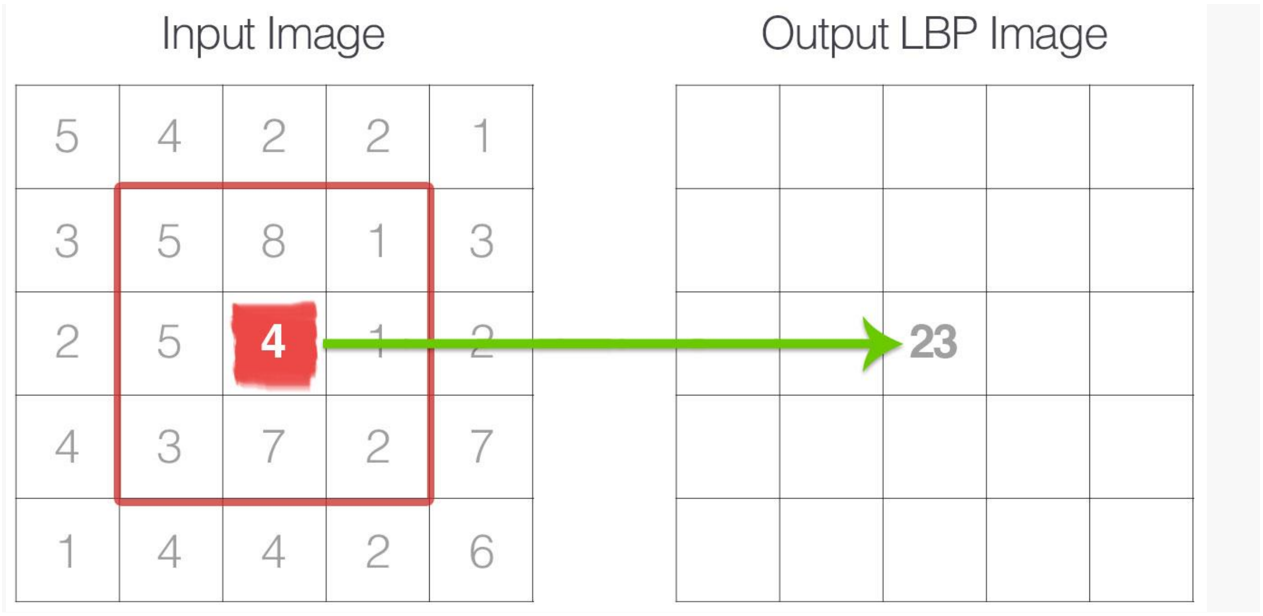


Figure 3: The calculated LBP value is then stored in an output array with the same width and height as the original image.

This process of thresholding, accumulating binary strings, and storing the output decimal value in the LBP array is then repeated for each pixel in the input image.

Here is an example of computing and visualizing a full LBP 2D array:

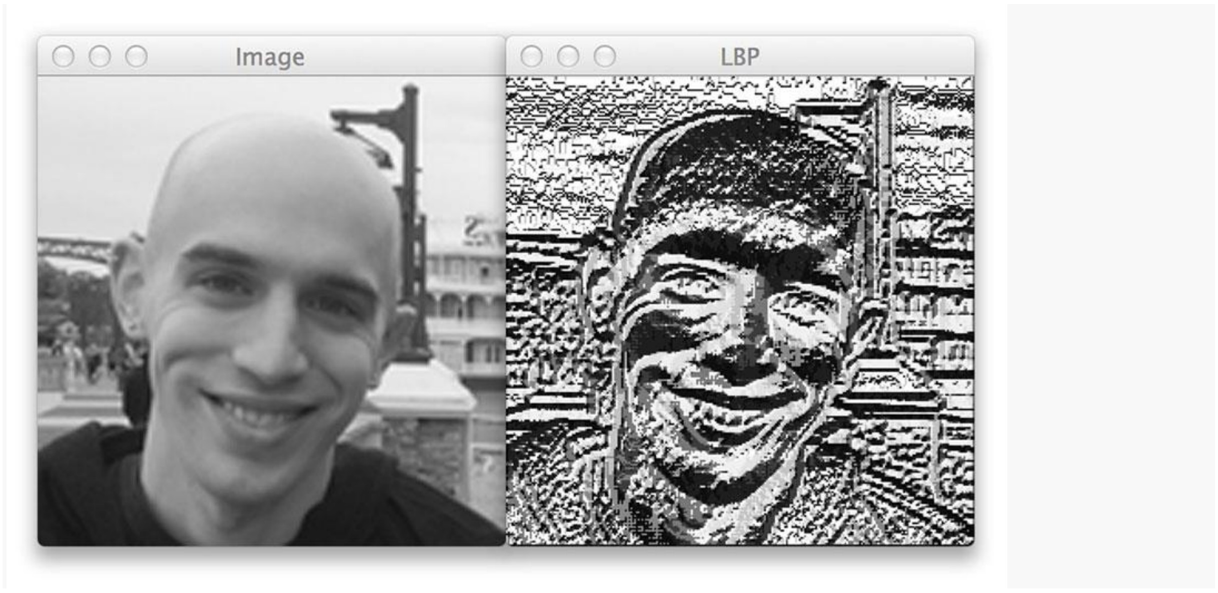


Figure 4: An example of computing the LBP representation *(right)* from the original input image *(left)*.

The last step is to compute a histogram over the output LBP array. Since a *3 x 3* neighborhood has *2 ^ 8 = 256* possible patterns, our LBP 2D array thus has a *minimum value of 0* and a *maximum value of 255*, allowing us to construct a 256-bin histogram of LBP codes as our final feature vector:

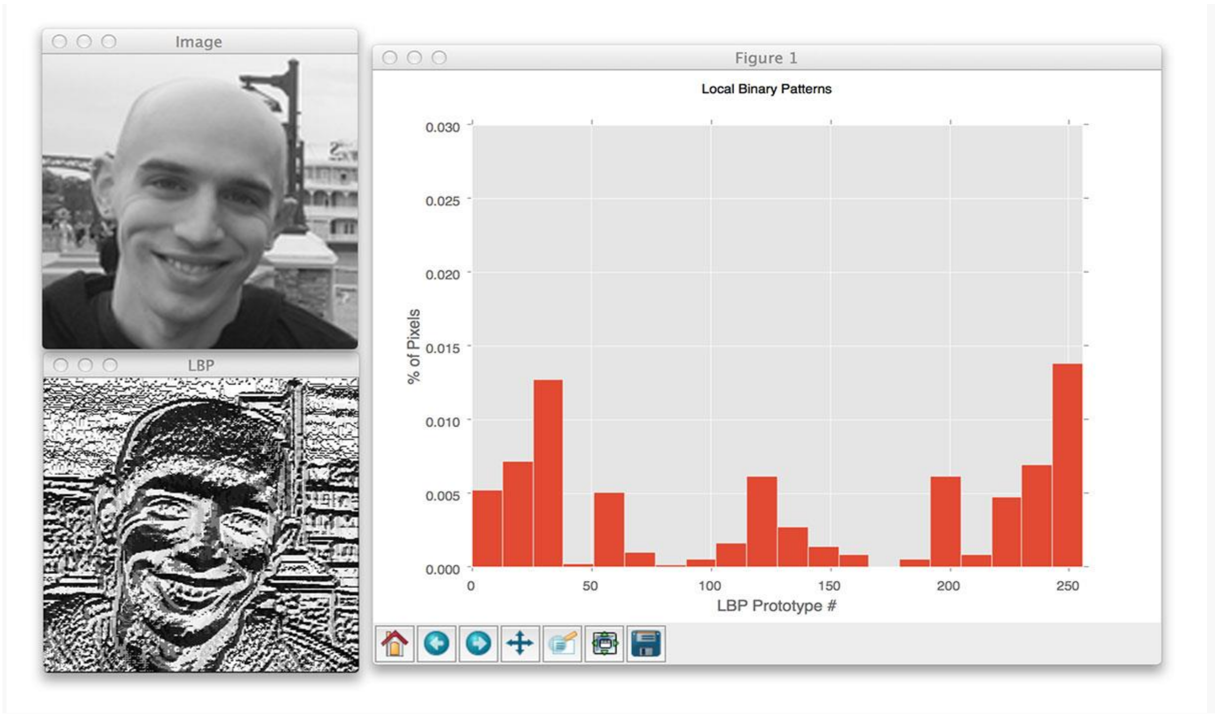


Figure 5: Finally, we can compute a histogram that tabulates the number of times each LBP pattern occurs. We can treat this histogram as our feature vector.

A primary benefit of this original LBP implementation is that we can capture extremely fine-grained details in the image. However, being able to capture details at such a small scale is also the *biggest drawback to the algorithm* — we cannot capture details at varying scales, only the fixed *3 x 3* scale!

To handle this, an extension to the original LBP implementation was proposed by Ojala et al. to handle variable neighborhood sizes. To account for variable neighborhood sizes, two parameters were introduced:

1. The number of points *p* in a circularly symmetric neighborhood to consider (thus removing relying on a square neighborhood).
2. The radius of the circle *r*, which allows us to account for different scales.

Below follows a visualization of these parameters:

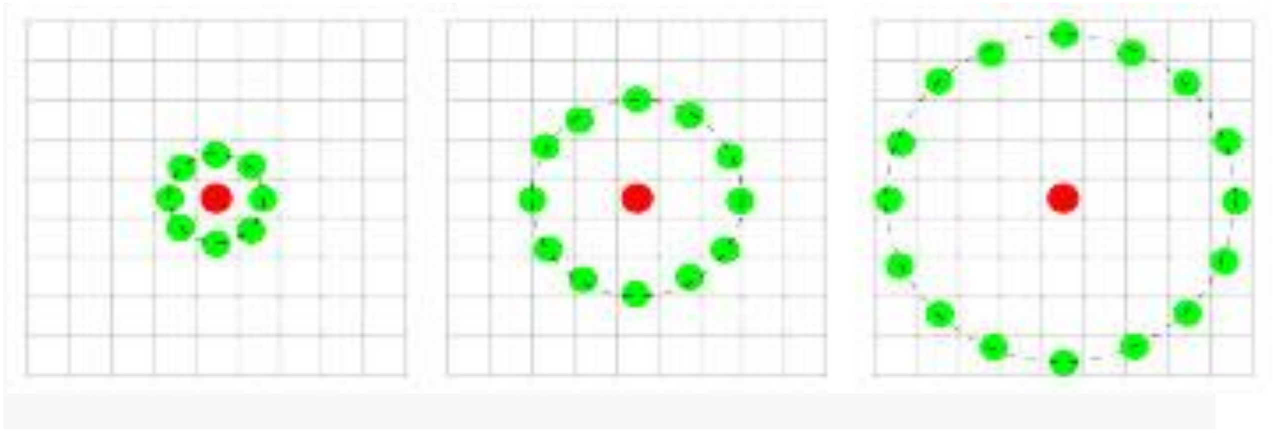


Figure 6: Three neighborhood examples with varying *p* and *r* used to construct Local Binary Patterns.

Lastly, it's important that we consider the concept of LBP ***uniformity***. A LBP is considered to be uniform if it has ***at most*** two *0-1* or *1-0* transitions. For example, the pattern 00001000 (2 transitions) and 10000000 (1 transition) are both considered to be **uniform patterns** since they contain at most two *0-1* and *1-0* transitions. The pattern 01010010) on the other hand is *not* considered a uniform pattern since it has six *0-1* or *1-0* transitions.

The number of uniform prototypes in a Local Binary Pattern is completely dependent on the number of points *p*. As the value of *p* increases, so will the dimensionality of your resulting histogram. Please refer to the original Ojala et al. paper for the full explanation on deriving the number of patterns and uniform patterns based on this value. However, for the time being simply keep in mind that given the number of points *p* in the LBP there are *p + 1* **uniform patterns**. The final dimensionality of the histogram is thus *p + 2*, where the added entry tabulates all patterns that are ***not uniform***.

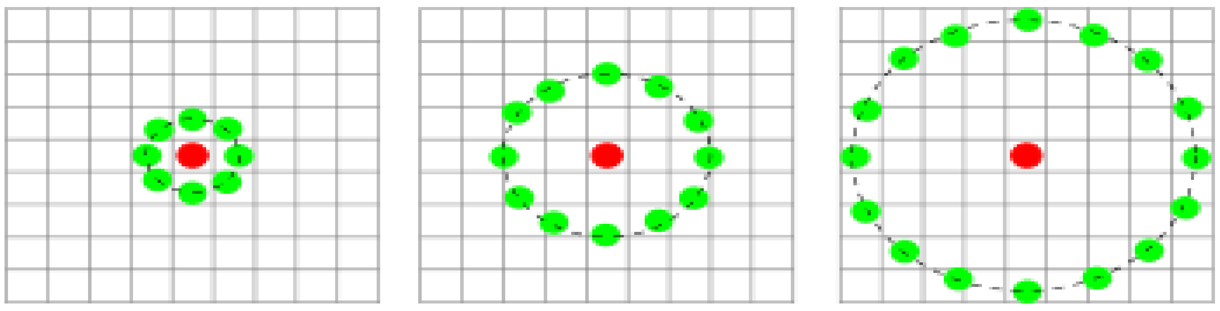
So why are uniform LBP patterns so interesting? Simply put: they add an extra level of *rotation and grayscale invariance*, hence they are commonly used when extracting LBP feature vectors from images.

4.1.1 Concept

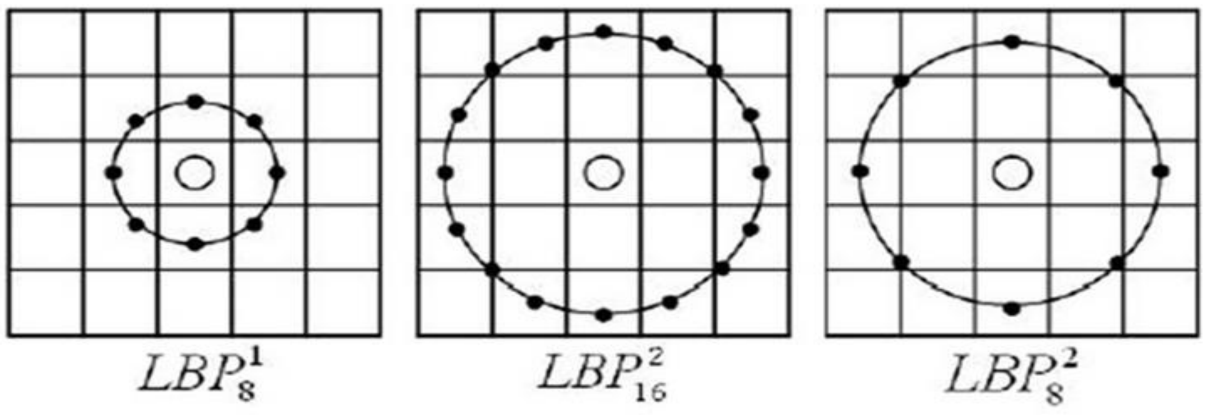
The LBP feature vector, in its simplest form, is created in the following manner:

- Divide the examined window into cells (e.g. 16x16 pixels for each cell).
- For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
- Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).
- Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.
- Optionally normalize the histogram.
- Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.

Some Figures:



Three neighborhood examples used to define a texture and calculate a local binary pattern (LBP)



Local binary pattern operations

4.1.2 Extensions

- Over-Complete Local Binary Patterns (OCLBP). OCLBP is a variant of LBP that has been shown to improve the overall performance on face verification. Unlike LBP, OCLBP adopts overlapping to adjacent blocks. Formally, the configuration of OCLBP is denoted as S : (a, b, v, h, p, r): an image is divided into a×b blocks with vertical overlap of v and horizontal overlap of h, and then uniform patterns LBP(u2,p,r) are extracted from all the blocks. Moreover, OCLBP is composed of several different configurations. For example, in their original paper, the authors used three configurations: S : (10,10,12,12,8,1),(14,14,12,12,8,2),(18,18,12,12,8,3). The three configurations consider three block sizes: 10×10, 14×14, 18×18, and half overlap rates along the vertical and horizontal directions. These configurations are concatenated to form a 40877 dimensional feature vector for an image of size 150x80.
- Transition Local Binary Patterns (tLBP). Binary value of transition coded LBP is composed of neighbor pixel comparisons clockwise direction for all pixels except the central.
- Direction coded Local Binary Patterns(dLBP): the dLBP encodes the intensity variation along the four basic directions through the central pixel in two bits.
- Modified Local Binary Patterns (mLBP): the mLBP compares the values of neighboring pixels to the average of the intensity values in the 3x3 window.
- Multi-block LBP: the image is divided into many blocks, a LBP histogram is calculated for every block and concatenated as the final histogram.
- Volume Local Binary Pattern(VLBP). VLBP looks at dynamic texture as a set of volumes in the (X,Y,T) space where X and Y denote the spatial coordinates and T denotes the frame index. The neighborhood of a pixel is thus defined in three dimensional space, and volume textons can be extracted into histograms.
- RGB-LBP: This operator is obtained by computing LBP over all three channels of the RGB color space independently, and then concatenating the results together.

4.2 Histogram

A **histogram** is an accurate graphical representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable (quantitative variable) and was first introduced by Karl Pearson. It is a kind of bar graph. To construct a histogram, the first step is to "bin" the range of values that is, divide the entire range of values into a series of intervals and then count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable. The bins (intervals) must be adjacent, and are often (but are not required to be) of equal size.

If the bins are of equal size, a rectangle is erected over the bin with height proportional to the frequency the number of cases in each bin. A histogram may also be normalized to display "relative" frequencies. It then shows the proportion of cases that fall into each of several categories, with the sum of the heights equaling 1.

However, bins need not be of equal width; in that case, the erected rectangle is defined to have its *area* proportional to the frequency of cases in the bin. The vertical axis is then not the frequency but *frequency density*.

The number of cases per unit of the variable on the horizontal axis. Examples of variable bin width are displayed on Census bureau data below.

As the adjacent bins leave no gaps, the rectangles of a histogram touch each other to indicate that the original variable is continuous.

Histograms give a rough sense of the density of the underlying distribution of the data, and often for density estimation: estimating the probability density function of the underlying variable. The total area of a histogram used for probability density is always normalized to 1. If the length of the intervals on the *x*-axis is all 1, then a histogram is identical to a relative frequency plot.

A histogram can be thought of as a simplistic kernel density estimation, which uses a kernel to smooth frequencies over the bins. This yields a smoother probability density function, which will in general more accurately reflect distribution of the underlying variable. The density estimate could be plotted as an alternative to the histogram, and is usually drawn as a curve rather than a set of boxes.

Another alternative is the average shifted histogram, which is fast to compute and gives a smooth curve estimate of the density without using kernels.

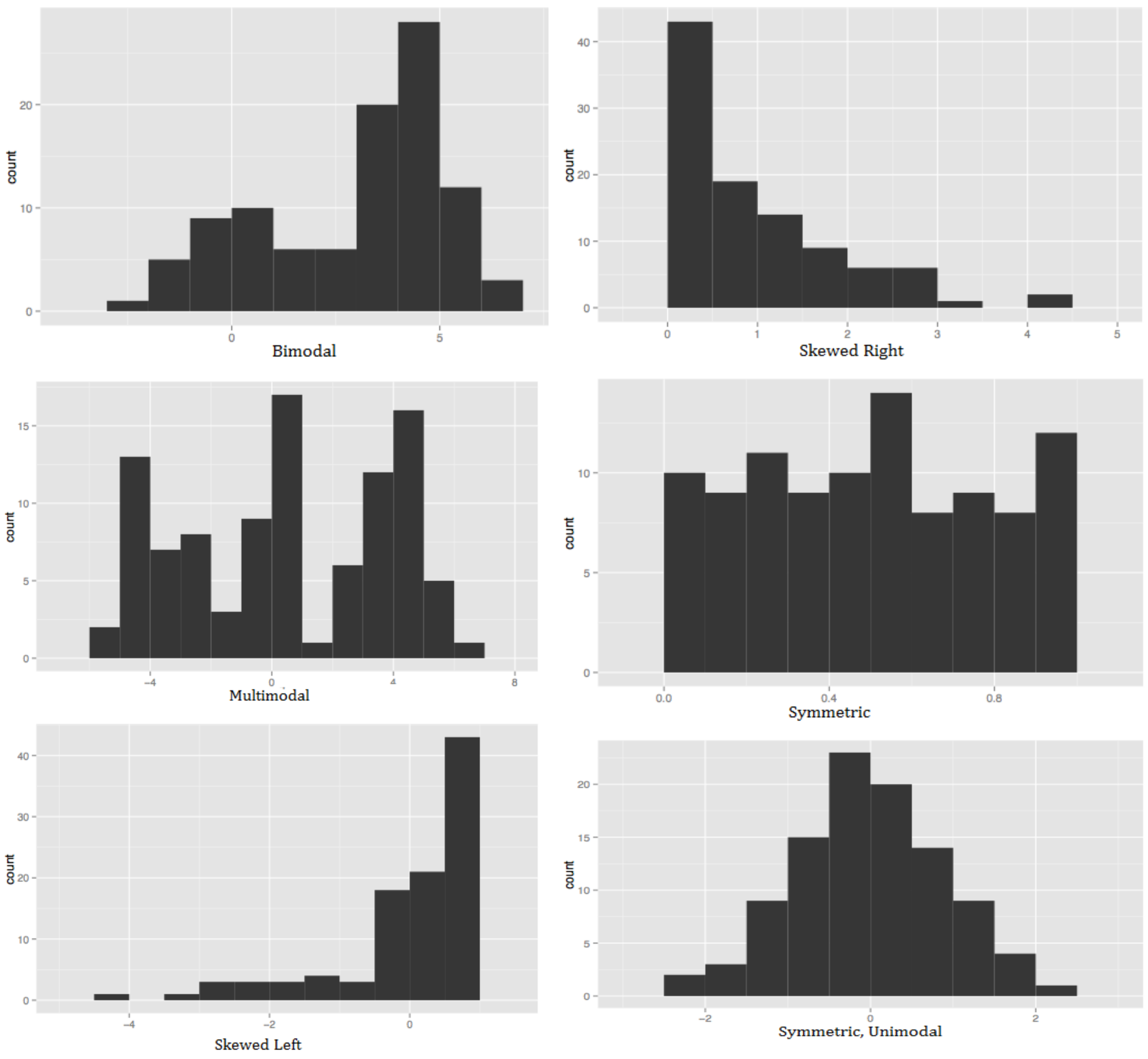
The histogram is one of the seven basic tools of quality control.

Histograms are sometimes confused with bar charts. A histogram is used for continuous data, where the bins represent ranges of data, while a bar chart is a plot of categorical variables. Some authors recommend that bar charts have gaps between the rectangles to clarify the distinction.

4.2.1 Examples

This is the data for the histogram to the right, using 509 items:

Bin	Count
−3.5 to -2.51	9
−2.5 to -1.51	32
−1.5 to -0.51	109
−0.5 to 0.49	180
0.5 to 1.49	132
1.5 to 2.49	34
2.5 to 3.49	4



The words used to describe the patterns in a histogram are: "symmetric", "skewed left" or "right", "unimodal", "bimodal" or "multimodal".

4.2.2 Mathematical Explanation

In a more general mathematical sense, a histogram is a function m_i that counts the number of observations that fall into each of the disjoint categories (known as *bins*), whereas the graph of a histogram is merely one way to represent a histogram. Thus, if we let n be the total number of observations and k be the total number of bins, the histogram m_i meets the following conditions:

$$n = \sum_{i=1}^{k} m_i .$$

Cumulative Histogram

A cumulative histogram is a mapping that counts the cumulative number of observations in all of the bins up to the specified bin. That is, the cumulative histogram M_i of a histogram m_j is defined as:

$$M_i = \sum_{j=1}^{i} m_j .$$

Number of Bins and Width

There is no "best" number of bins, and different bin sizes can reveal different features of the data. Grouping data is at least as old as Graunt's work in the 17th century, but no systematic guidelines were given until Sturges's work in 1926.

Using wider bins where the density of the underlying data points is low reduces noise due to sampling randomness; using narrower bins where the density is high (so the signal drowns the noise) gives greater precision to the density estimation. Thus varying the bin-width within a histogram can be beneficial. Nonetheless, equal-width bins are widely used.

Some theoreticians have attempted to determine an optimal number of bins, but these methods generally make strong assumptions about the shape of the distribution. Depending on the actual data distribution and the goals of the analysis, different bin widths may be appropriate, so experimentation is usually needed to determine an appropriate width. There are, however, various useful guidelines and rules of thumb.

The number of bins k can be assigned directly or can be calculated from a suggested bin width h as:

$$k = \left\lceil \frac{\max x - \min x}{h} \right\rceil .$$

The braces indicate the ceiling function.

Square-root Choice

$$k = \sqrt{n},$$

which takes the square root of the number of data points in the sample (used by Excel histograms and many others).

Sturges' Formula

Sturges' formula is derived from a binomial distribution and implicitly assumes an approximately normal distribution.

$$k = \lceil \log_2 n \rceil + 1,$$

It implicitly bases the bin sizes on the range of the data and can perform poorly if $n < 30$, because the number of bins will be small—less than seven—and unlikely to show trends in the data well. It may also perform poorly if the data are not normally distributed.

Rice Rule:

$$k = \lceil 2n^{1/3} \rceil,$$

The Rice Rule is presented as a simple alternative to Sturges's rule.

Doane's Formula

Doane's formula is a modification of Sturges' formula which attempts to improve its performance with non-normal data.

$$k = 1 + \log_2(n) + \log_2\left(1 + \frac{|g_1|}{\sigma_{g_1}}\right)$$

where g_1 is the estimated 3rd-moment-skewness of the distribution and

$$\sigma_{g_1} = \sqrt{\frac{6(n-2)}{(n+1)(n+3)}}$$

Scott's Normal Reference Rule:

$$h = \frac{3.5\hat{\sigma}}{n^{1/3}},$$

where $\hat{\sigma}$ is the sample standard deviation. Scott's normal reference rule is optimal for random samples of normally distributed data, in the sense that it minimizes the integrated mean squared error of the density estimate.

Freedman–Diaconis Choice:

The Freedman–Diaconis rule is:

$$h = 2\frac{\mathrm{IQR}(x)}{n^{1/3}},$$

which is based on the interquartile range, denoted by IQR. It replaces 3.5σ of Scott's rule with 2 IQR, which is less sensitive than the standard deviation to outliers in data.

Minimizing Cross-validation Estimated Squared Error:

This approach of minimizing integrated mean squared error from Scott's rule can be generalized beyond Normal distributions, by using leave-one out cross validation.

$$\underset{h}{\arg\min}\hat{J}(h) = \underset{h}{\arg\min}\left(\frac{2}{(n-1)h} - \frac{n+1}{n^2(n-1)h}\sum_k N_k^2\right)$$

Here, N_k is the number of datapoints in the *k*th bin, and choosing the value of *h* that minimizes *J* will minimize integrated mean squared error.

Choice based on minimization of an estimated *L*$^{2.}$

$$\underset{h}{\arg\min}\frac{2\bar{m} - v}{h^2}$$

where $\bar{m}$ and v are mean and biased variance of a histogram with bin-width

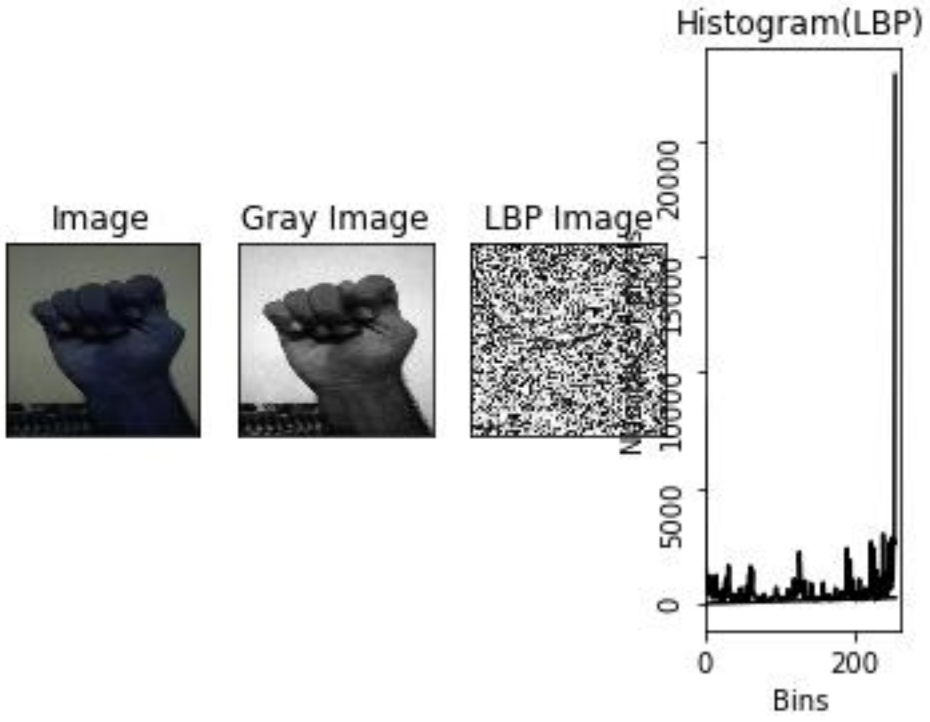
$$h, \bar{m} = \frac{1}{k}\sum_{i=1}^{k} m_i \text{ and } v = \frac{1}{k}\sum_{i=1}^{k}(m_i - \bar{m})^2.$$

4.3 Source Code

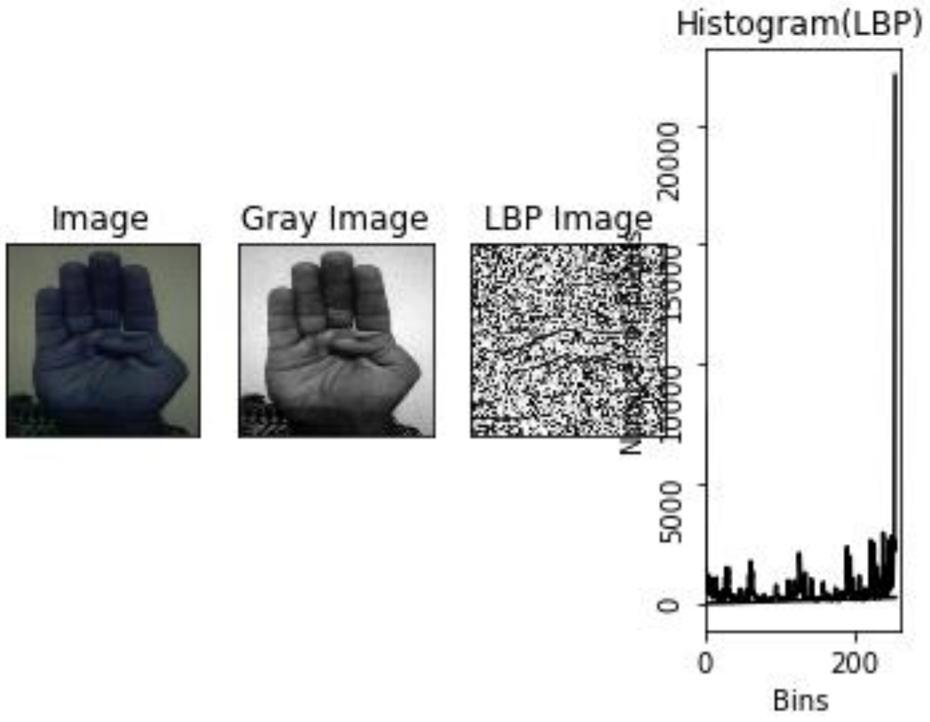
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

def get_pixel(img, center, x, y):
    new_value = 0
    try:
        if img[x][y] >= center:
            new_value = 1
    except:
        pass
    return new_value

def lbp_calculated_pixel(img, x, y):

    '''

     64 | 128 |   1
    ----------------
     32 |   0 |   2
    ----------------
     16 |   8 |   4

    '''
    center = img[x][y]
    val_ar = []
    val_ar.append(get_pixel(img, center, x-1, y+1))     # top_right
    val_ar.append(get_pixel(img, center, x, y+1))       # right
    val_ar.append(get_pixel(img, center, x+1, y+1))     # bottom_right
    val_ar.append(get_pixel(img, center, x+1, y))       # bottom
    val_ar.append(get_pixel(img, center, x+1, y-1))     # bottom_left
    val_ar.append(get_pixel(img, center, x, y-1))       # left
    val_ar.append(get_pixel(img, center, x-1, y-1))     # top_left
    val_ar.append(get_pixel(img, center, x-1, y))       # top
    power_val = [1, 2, 4, 8, 16, 32, 64, 128]
    val = 0
    for i in range(len(val_ar)):
        val += val_ar[i] * power_val[i]
    #print(val)
    return val

def show_output(output_list):
    output_list_len = len(output_list)
    figure = plt.figure()
    for i in range(output_list_len):
        current_dict = output_list[i]
        current_img = current_dict["img"]
        current_xlabel = current_dict["xlabel"]
        current_ylabel = current_dict["ylabel"]
        current_xtick = current_dict["xtick"]
        current_ytick = current_dict["ytick"]
        current_title = current_dict["title"]
        current_type = current_dict["type"]
        current_plot = figure.add_subplot(1, output_list_len, i+1)
        if current_type == "gray":
            current_plot.imshow(current_img, cmap =
plt.get_cmap('gray'))
            current_plot.set_title(current_title)
            current_plot.set_xticks(current_xtick)
            current_plot.set_yticks(current_ytick)
            current_plot.set_xlabel(current_xlabel)
            current_plot.set_ylabel(current_ylabel)
        elif current_type == "histogram":
            current_plot.plot(current_img, color = "black")
            current_plot.set_xlim([0,260])
            current_plot.set_title(current_title)
            current_plot.set_xlabel(current_xlabel)
            current_plot.set_ylabel(current_ylabel)
            ytick_list = [int(i) for i in current_plot.get_yticks()]
            current_plot.set_yticklabels(ytick_list,rotation = 90)

    plt.show()

def main():
    image_file = 'GP/TTay.jpg'
    img_load = cv2.imread(image_file)
    img_bgr = cv2.resize(img_load, (400, 400))
    cv2.imshow('imgbgr',img_bgr)
    #for getting the height widht and channel of the image
    height, width, channel = img_bgr.shape
    #Gray image
    img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
    #print(height)
    #print(width)
    # print(channel)
    #cv2.imshow('imggray',img_gray)
    #getting lbp of image
    img_lbp = np.zeros((height, width), np.uint8)
    #cv2.imshow('imglbp',img_lbp)
    #print(img_lbp)
    imgoned = np.zeros((16, 16), np.uint8)
    inputnumpy = np.zeros((height, width), np.uint8)
    trc=0
    tcc=0
    for i in range(0, height):
        for j in range(0, width):
            img_lbp[i, j] = lbp_calculated_pixel(img_gray, i, j)
    print(len(img_lbp))
    for rc in range(0,height,16):
        for cc in range(0,width,16):
            trc=rc
            for i in range(15):
                tcc=cc
                for j in range(15):
                    #print(trc,tcc,img_lbp[trc,tcc])
                    imgoned[i][j]=img_lbp[trc][tcc]
                    tcc=tcc+1
                trc=trc+1
            hist_lbp1=cv2.calcHist([imgoned], [0], None, [256], [0, 256])
            with open("_1test.txt", "a") as myfile:
                np.savetxt(myfile, np.array(hist_lbp1), fmt='%.2f')
            #np.savetxt('_file.txt', np.array(hist_lbp1), fmt='%.2f')
            #imgoned.empty()
    print(img_lbp[0,0])
    print(img_lbp[0,1])
    print(img_lbp[1,2])
    print(img_lbp[0][0])

    hist_lbp = cv2.calcHist([img_lbp], [0], None, [256], [0, 256])
    print(hist_lbp[0])
    print(len(hist_lbp))
    np.savetxt('GH/TTay.txt', np.array(hist_lbp), fmt='%.2f')
```

```
     output_list = []
    output_list.append({
        "img": img_bgr,
        "xlabel": "",
        "ylabel": "",
        "xtick": [],
        "ytick": [],
        "title": "Image",
        "type": "gray"
    })
    output_list.append({
        "img": img_gray,
        "xlabel": "",
        "ylabel": "",
        "xtick": [],
        "ytick": [],
        "title": "Gray Image",
        "type": "gray"
    })
    output_list.append({
        "img": img_lbp,
        "xlabel": "",
        "ylabel": "",
        "xtick": [],
        "ytick": [],
        "title": "LBP Image",
        "type": "gray"
    })
    output_list.append({
        "img": hist_lbp,
        "xlabel": "Bins",
        "ylabel": "Number of pixels",
        "xtick": None,
        "ytick": None,
        "title": "Histogram(LBP)",
        "type": "histogram"
    })

    show_output(output_list)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
    print("LBP Program is finished")

if __name__ == '__main__':
    main()
```

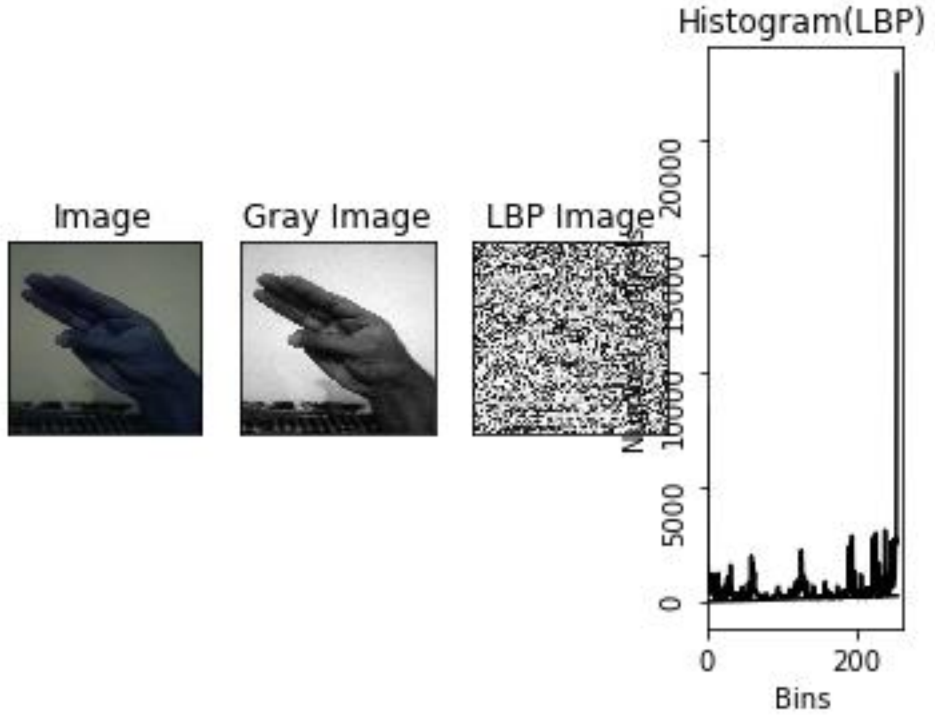
4.3.1 Outputs:



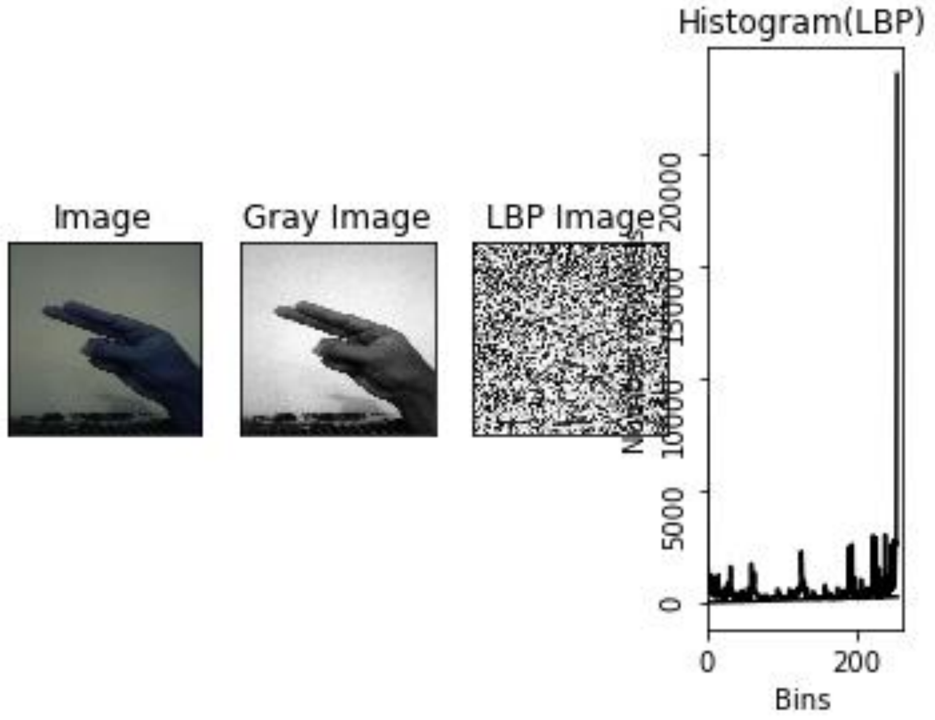
Result of LBP for Alif



Result of LBP for Bay



Result of LBP for Chay



Result of LBP for Hay

4.4 Support Vector Machine

In machine learning, **support vector machines** (**SVMs**, also **support vector networks**[]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data are not labeled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The clustering algorithm which provides an improvement to the support vector machines is called **support vector clustering**[2] and is often used in industrial applications either when data are not labeled or when only some data are labeled as a preprocessing for a classification pass.

4.4.1 Source Code (Training)

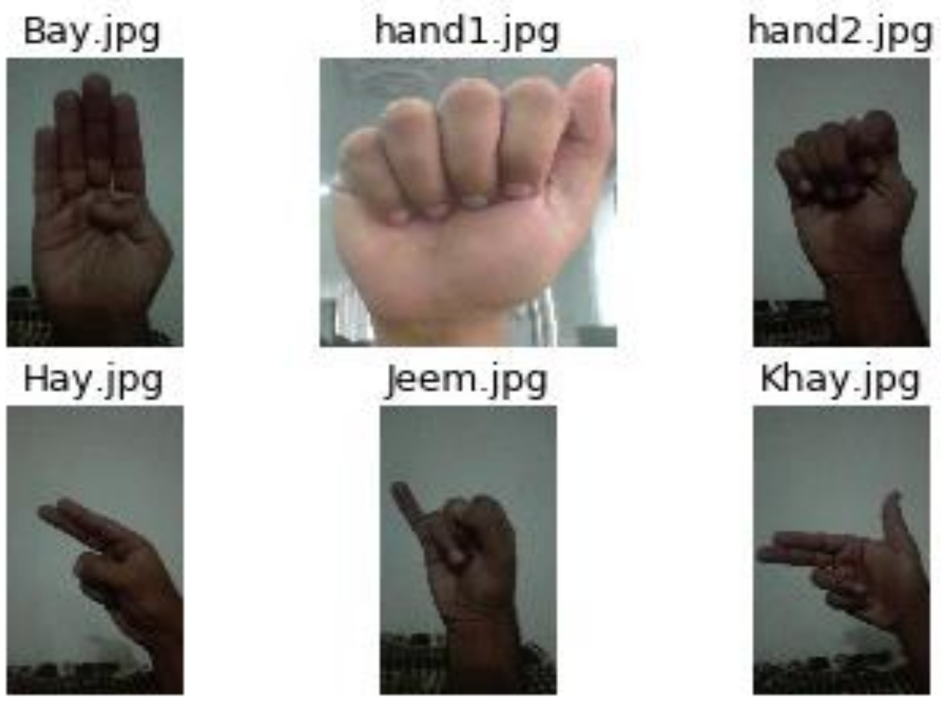
```
# OpenCV bindings
import cv2
# To performing path manipulations
import os
# Local Binary Pattern function
from skimage.feature import local_binary_pattern
# To calculate a normalized histogram
from scipy.stats import itemfreq
from sklearn.preprocessing import normalize
# To read class from file
import csv
# For plotting
import matplotlib.pyplot as plt
# For array manipulations
import numpy as np
# For saving histogram values
from sklearn.externals import joblib
# For command line input
import argparse as ap
# Utility Package
import cvutils
from sklearn.svm import LinearSVC

# Get the path of the training set
# Store the path of training images in train_images
train_images = cvutils.imlist("data/lbp/train/")
# Dictionary containing image paths as keys and corresponding label as value
#train_dic = {}
#with open('data/lbp/class_train.txt', 'rb') as csvfile:
#    reader = csv.reader(csvfile, delimiter=' ')
#    for row in reader:
```

```
#       train_dic[row[0]] = int(row[1])

# List for storing the LBP Histograms, address of images and the corresponding label
X_test = []
X_name = []
y_test = []

# For each image in the training set calculate the LBP histogram
# and update X_test, X_name and y_test
im = cv2.imread(train_images[0])
    # Convert to grayscale as LBP works on grayscale image

for i in range(len(train_images)):
    print(train_images[i])
    # Read the image
    im = cv2.imread(train_images[i])
    # Convert to grayscale as LBP works on grayscale image
    im_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(im_gray,(5,5),0)
    ret,thresh1 = cv2.threshold(blur,70,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    radius = 3
    # Number of points to be considered as neighbourers
    no_points = 8 * radius

    # Uniform LBP is used
    lbp = local_binary_pattern(thresh1, no_points, radius, method='uniform')
    # Calculate the histogram
    x = itemfreq(lbp.ravel())
    # Normalize the histogram
    hist = x[:, 1]/sum(x[:, 1])
    # Append image path in X_name
    X_name.append(train_images[i])
    Xname='X_name'+train_images[i]
    print(Xname)
    # Append histogram to X_name
    X_test.append(hist)
    # Append class label in y_test
    y_test.append(train_images[i])
    yname='y_test'+train_images[i]
    print(yname)

print(X_name)
print(X_test)
print(y_test)
joblib.dump((X_name, X_test, y_test), "lbp.pkl", compress=3)
# Display the training images
#nrows = 2
#ncols = 3
#fig, axes = plt.subplots(nrows,ncols)
#for row in range(nrows):
#    for col in range(ncols):
#       axes[row][col].imshow(cv2.cvtColor(cv2.imread(X_name[row*ncols+col]), cv2.COLOR_BGR2RGB))
#       axes[row][col].axis('off')
#       axes[row][col].set_title("{}".format(os.path.split(X_name[row*ncols+col])[1]))
#
## Convert to numpy and display the image
#fig.canvas.draw()
#im_ts = np.fromstring(fig.canvas.tostring_rgb(), dtype=np.uint8, sep='')
#im_ts = im_ts.reshape(fig.canvas.get_width_height()[::-1] + (3,))
#cv2.imshow("Training Set", im_ts)
#cv2.waitKey()
```

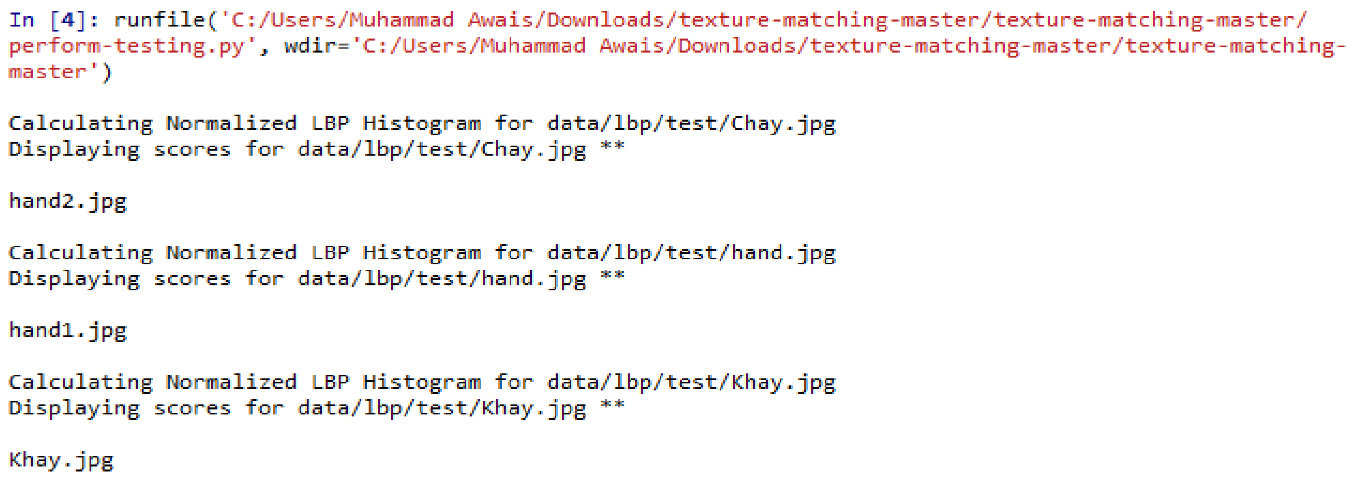
4.4.2 Source Code (Testing)

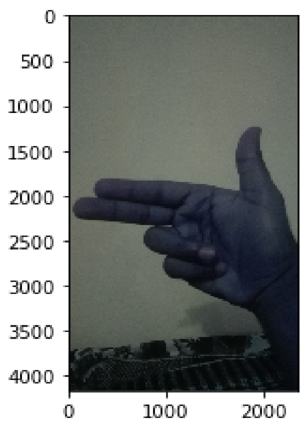
```
# OpenCV bindings
import cv2
# To performing path manipulations
import os
# Local Binary Pattern function
from skimage.feature import local_binary_pattern
# To calculate a normalized histogram
from scipy.stats import itemfreq
from sklearn.preprocessing import normalize
# To read class from file
import csv
# For plotting
import matplotlib.pyplot as plt
# For array manipulations
import numpy as np
# For saving histogram values
from sklearn.externals import joblib
# For command line input
import argparse as ap
# Utility Package
import cvutils

# Get the path of the training set

# Load the List for storing the LBP Histograms, address of images and the corresponding label
X_name, X_test, y_test = joblib.load("lbp.pkl")

#opening camera and uploading images
#cap=cv2.VideoCapture(0)
#while(1):
#    ret, frame = cap.read()
#    if 0xFF == ord('r'):
#        break;
#cap.release()

# Store the path of testing images in test_images
test_images = cvutils.imlist("data/lbp/test/")
# Dictionary containing image paths as keys and corresponding label as value
test_dic = {}
with open('data/lbp/class_test.txt', 'rb') as csvfile:
    reader = csv.reader(csvfile, delimiter=' ')
    for row in reader:
        test_dic[row[0]] = int(row[1])

# Dict containing scores
results_all = {}

for test_image in test_images:
    print "\nCalculating Normalized LBP Histogram for {}".format(test_image)
    # Read the image
    im = cv2.imread(test_image)
    # Convert to grayscale as LBP works on grayscale image
    im_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(im_gray,(5,5),0)
    ret,thresh1 = cv2.threshold(blur,70,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    radius = 3
    # Number of points to be considered as neighbourers
    no_points = 8 * radius
    # Uniform LBP is used
    lbp = local_binary_pattern(thresh1, no_points, radius, method='uniform')
    # Calculate the histogram
    x = itemfreq(lbp.ravel())
    # Normalize the histogram
    hist = x[:, 1]/sum(x[:, 1])
    # Display the query image
    results = []
    # For each image in the training dataset
    # Calculate the chi-squared distance and the sort the values
    for index, x in enumerate(X_test):
        score = cv2.compareHist(np.array(x, dtype=np.float32), np.array(hist, dtype=np.float32),
cv2.cv.CV_COMP_CHISQR)
        results.append((X_name[index], round(score, 3)))
    results = min(results, key=lambda score: score[1])
    results_all[test_image] = results
    print "Displaying scores for {} ** \n".format(test_image)
    name=results[0][15:]
    print(name)
    #for image, score in results:
        #print "{} has score {}".format(image, score)
#print(results)
#for test_image, results in results_all.items():
#    # Read the image
#    im = cv2.imread(test_image)
#    # Display the results
#    nrows = 2
#    ncols = 3
#    fig, axes = plt.subplots(nrows,ncols)
#    fig.suptitle("** Scores for -> {}**".format(test_image))
#    for row in range(nrows):
#        for col in range(ncols):
#            axes[row][col].imshow(cv2.imread(results[row*ncols+col][0]))
#            axes[row][col].axis('off')
#            axes[row][col].set_title("Score {}".format(results[row*ncols+col][1]))
#    fig.canvas.draw()
#    im_ts = np.fromstring(fig.canvas.tostring_rgb(), dtype=np.uint8, sep='')
#    im_ts = im_ts.reshape(fig.canvas.get_width_height()[::-1] + (3,))
#    cv2.imshow("** Query Image -> {}**".format(test_image), im)
#    cv2.imshow("** Scores for -> {}**".format(test_image), im_ts)
#    cv2.waitKey()
#    cv2.destroyAllWindows()
```

4.4.3 Outputs (Training)



4.4.4 Output (Testing)


```
In [4]: runfile('C:/Users/Muhammad Awais/Downloads/texture-matching-master/texture-matching-master/
perform-testing.py', wdir='C:/Users/Muhammad Awais/Downloads/texture-matching-master/texture-matching-
master')

Calculating Normalized LBP Histogram for data/lbp/test/Chay.jpg
Displaying scores for data/lbp/test/Chay.jpg **

hand2.jpg

Calculating Normalized LBP Histogram for data/lbp/test/hand.jpg
Displaying scores for data/lbp/test/hand.jpg **

hand1.jpg

Calculating Normalized LBP Histogram for data/lbp/test/Khay.jpg
Displaying scores for data/lbp/test/Khay.jpg **

Khay.jpg
```



Chapter 5

Testing

5.1 Testing/ Evaluation

After implementation of software, it is important to validate software by testing. Testing provides a neutral view of software. It is just like error detection. It specifies whether software fulfill user requirements or not. It helps in distinguishing contrasts between given input and expected output.

5.1.1 Why is software testing necessary?

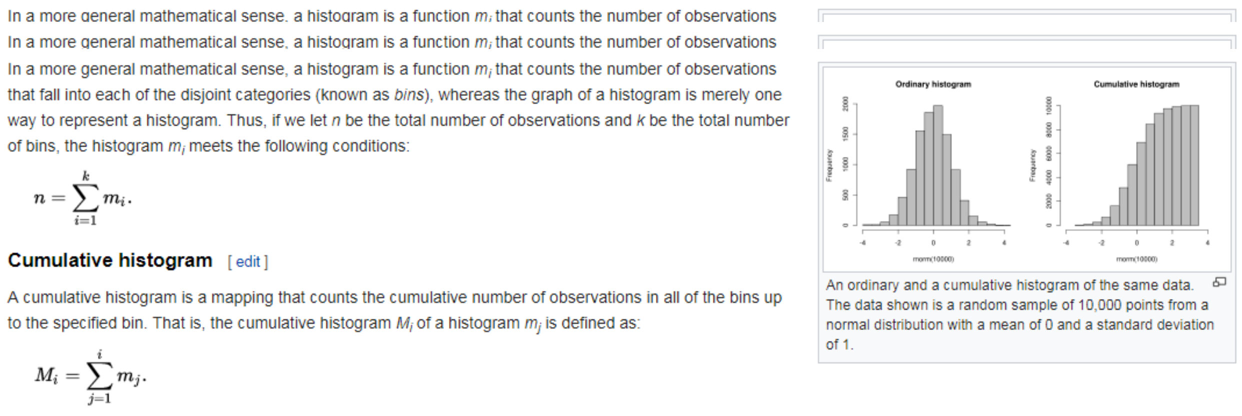
Software Testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong.

There are several reasons which clearly tell us as why software testing is important and what are the major things that we should consider while testing of any product or application.

Software testing is very important because of the following reasons:

1. Software testing is really required to point out the defects and errors that were made during the development phases

2. It's essential since it makes sure of the Customer's reliability and their satisfaction in



In a more general mathematical sense, a histogram is a function m_i that counts the number of observations

In a more general mathematical sense, a histogram is a function m_i that counts the number of observations

In a more general mathematical sense, a histogram is a function m_i that counts the number of observations that fall into each of the disjoint categories (known as *bins*), whereas the graph of a histogram is merely one way to represent a histogram. Thus, if we let *n* be the total number of observations and *k* be the total number of bins, the histogram m_i meets the following conditions:

$$n = \sum_{i=1}^{k} m_i.$$

Cumulative histogram [edit]

A cumulative histogram is a mapping that counts the cumulative number of observations in all of the bins up to the specified bin. That is, the cumulative histogram M_i of a histogram m_j is defined as:

$$M_i = \sum_{j=1}^{i} m_j.$$

An ordinary and a cumulative histogram of the same data. The data shown is a random sample of 10,000 points from a normal distribution with a mean of 0 and a standard deviation of 1.

5.2 Test Cases

The tests cases were performed during the project development which was verified successfully as follows:

5.2.1 Test-1

Test Case ID	Test-1
Test Engineer (s)	Nouman Dilshad Ch. Muhammad Awais
Name	Read Image
Actor	User
Goal	To recognize a sign.
Precondition	Application should be started.
Main Success Scenario	• User will select a valid image (image should have hand in it). • System will detect the hand. • Image will be recognized.
Alternate Scenario	• User selects invalid image (image without hand). • System will check the validity of image (hand detection). • Image will not be selected. • Appropriate error message will be displayed showing the reason.
Test Result	Passed

Chapter 6
Conclusion

6.1 Conclusion

With this project we have learned a lot. This project was fun from the start and only got more interesting as we went on developing it. We became more interested in the subject the more we researched it.

In the proposed work, we implemented double layer security by combining the cryptography and steganography. In recognition we have used LBP which is considered as indestructible. In matching/classification/testing we have used a SVM algorithm.

Appendix

References