Al-Baath University
Faculty of Information Technology
Department of Software Engineering
and Information Systems

SIGN LANGUAGE RECOGNITION

"Based on digital image processing and machine learning techniques"

Prepared by
Majd AKLEH
Hussain HUSSAIN
Marsil ZAKOUR

Supervised by
Yosser ATASSI Ph.D
Mouhammad Rabee SHAHEEN Ph.D
Eng. Ossama NASSER

Academic year 2015-2016

Abstract

Sign language recognition using computer systems is strongly related to computer vision and machine learning fields. Such systems require accurate methods in order to be able to provide good results.

In this project, we present a simple system for sign language recognition. This study is done on a subset of the ASL (American Sign language), consisting of eight signs. The resulting system takes as input a still image from a normal mobile camera, then it uses morphological operations and face detection algorithms to extract skin and therefore detect hands. After that, features are extracted from a binary image; these features include geometric properties, invariant moments, principal components and active fingers attributes. The extracted features are fed into two different classifiers, Artificial Neural Network and Support Vector Machine.

The experimental results in the study show a classification accuracy of 99.83%, also hand extraction is done with average time of 0.2377 seconds, feature extraction stage produce 27 features within average time of 0.0426 seconds, and finally classification done with average of 0.0119 seconds.

Final result of our study shows that mobiles and computers with average quality cameras are capable of providing a real time and accurate classification for ASL. signs.

ملخص المشروع

أنظمة التعرف على الإشارات هي أنظمة مرتبطة بشكل وثيق بكل من مجالي الذكاء الاصطناعي والرؤية الحاسوبية. إن مثل هذه الأنظمة تتطلب طرائق دقيقة من أجل الحصول على أفضل النتائج .في هذا المشروع نقدم نظام بسيط للتعرف على لغة الإشارة، حيث أن هذه الدراسة مخصصة للتعامل مع ثمانية حركات من لغة الإشارة الأمريكية أوASL. إن النظام الناتج سوف يتعامل مع معلومات دخل تمثل صورة ملتقطة من جهاز محمول عادي، ثم يتم تطبيق عمليات لتحسين الصورة وخوارزميات تحديد مكان الوجه، بحيث يتم استخدامه في عملية تحديد البشرة ومنه اليدين. بعدها يتم جمع معلومات خصائص عن اليدين من الصورة الثنائية الناتجة عن العمليات السابقة، مثل هذه الخصائص تتضمن قيم هندسية تعطي واصفات عن الشكل الثنائي الموجود بالصورة، مكونات PCA ومعلومات عن الاصابع المرقوعة. هذه الخصائص يتم استخدامها في نوعين من المصنفات، وهم الشبكات العصبونية ANN وSVM. النتائج التجريبية في هذه الدراسة تظهر دقة تصنيف عالية 99.83%، أزمنة تحديد اليد في الصورة كانت بمعدل 0.2377 ثا، كما أن ازمنة استخراج 27 قيمة من المعلومات تمت بمعدل 0.0426 ثا، وأخيرا زمن التصنيف كان بمعدل 0.0119 ثا .النتائج النهائية لدراستنا تظهر أن أجهزة المحمول والحاسب التي تملك كاميرات ذو نوعية عادية يمكنها اعطاء نتائج في الزمن الحقيقي ودقة تصنيف جيدة للإشارات.

Contents

List of Figures

1 Introduction

1.1 Problem overview

People with hearing disabilities have a difficulty in communicating with other people which creates a gap between the two. In order to cover this gap, they use some known gestures to express their thoughts, the meanings of these gestures are formalized and defined as the "Sign Language" or SL.

The SL becomes the means of the people with hearing disabilities to deal with the world. However, it differs from one region to another, so each impaired person has to learn the SL of their region; people who regularly deal with them also should learn it.

Automated sign language recognition is the process of translating a certain SL to spoken words. Many works have been presented in this field. In this work, we study some methods and techniques used in sign language recognition, starting from the very first step in the process and ending at the recognition of the sign; and compare between the considered methods.

1.2 Project objectives

The aim of this project is to apply machine learning techniques in sign language recognition process and enhance recognition rate by

1. Studying hand detection in still pictures
2. Show the specification of different features and their contribution to the recognition process

3. Designing of two different machine learning models in order to recognize the signs and training them using data collected by different people

4. Conducting experiments on the proposed models to test out their accuracy and performance

5. Comparing the results with similar previous works

We end up with recommending certain classification models with a subset of features to solve the problem

1.3 Project limitations

- Performing classification on 8 signs with minimum apparent similarity, which restricts our system to be formalized on the entire alphabet

- Poor skin detection accuracy in extreme illumination conditions, decreasing our system flexibility

1.4 Methodology

After studying and testing several related works, the process has been broken into 3 main phases: hand localization, feature extraction and classification. Each phase has different approaches to be mentioned and compared.

- In chapter 2, we review our study of previous related works in the field of sign language recognition and related fields, and after that we start to discuss the recognition process.

- Chapter 3 discusses the methods of hand localization in a still image and proposes two different approaches.

- Chapter 4 proposes 4 different sets of features that can be extracted from the hand binary shape, discusses the reasons for choosing each feature and shows how the proposed features separate the studied signs.
- Chapter 5 shows the two considered machine learning models to differentiate between signs.
- After implementing the recognition system as discussed above, different tests have been conducted on the proposed models. The accuracy of the combination of one model with a certain subset of features is shown and also compared to the results of different works. This whole testing process was reviewed in chapter 6.
- In chapter 7, we conclude the results and challenges; and view the expected future work and possible applications.

1.5 Tools and Methods

- MATLAB R2015a and R2014a were mainly used to do the whole work of implementing the system, visualizing results
- Peter Kovesi's `hysthresh` used to apply hysteresis thresholding (http://www.peterkovesi.com/matlabfns/Spatial/hysthresh.m)
- `libsvm` was used to build and train SVM model
- Tex-maker was used to make this report
- Flow Chart Maker & Online Diagram Software (http://draw.io) was used to draw some charts and diagrams
- Using IPWebcam mobile application for capturing the images.

2 Related works

Sign language recognition motivated many researchers for years, trying to develop large number of artificial models in order to overcome this challenge. Thousands of proposed solutions and published papers have been made in this field, presenting many views to be considered later in the future studies.

2.1 Sign language

Deafness is defined as a person's is inability to understand speech [2]. Sign languages differ from a country to another regardless of the spoken language, for example, the American Sign Language (ASL) is not mutually intelligible with the British one, even though the two countries have the same spoken language [3]. Every country has a deaf community, while most of these individuals are born with hearing loss, others lose their sense of hearing while they are children, or even as adults [4].

This research aims to reduce the gap of communications between deaf and hearing people by focusing on the recognition of letters and words, which are the bases of a sign language.

2.2 Hand detection and localization

Hands play the most important part in most of the signs, so one of the most vital steps for any sign language recognition system is to get information about the hand shape in order to classify the presented gesture.

Some works depend on advanced technologies and powerful hardware, e.g., data gloves, such as CyberGlove [5, 2], and depth sensors, such as Microsoft's Kinect sensor [6, 7, 8, 9]. Unlike optical sensors, e.g., cameras, such sensors are usually more reliable and are not affected by lighting conditions or cluttered backgrounds [6].

Although data glove based SL recognition achieves good performance even for large vocabularies, the device is too expensive to popularize [7]. However, depth sensors are also expensive comparing to regular cameras.

Other works extract the hand information by capturing an image of the signer and trying to find the signer's hands using image processing and machine learning techniques [10, 11, 12, 6]. Regarding the image capturing choice for this step, we've seen some works depending on skin filtering to localize the hands [10, 11]. Papers published in the field of skin filtering [13, 14, 15] mostly depend on HSV, YCbCr color spaces to separate between skin from non-skin pixels.

In [12], Mittal *et al.* used a compromise between 3 detectors: shape, context, and skin color. Each method delivers a number of hypotheses for a bounding box for the hand, and then each of the hypotheses is given a score in order to be classified as a hand or not.

2.3 Features Extraction

Sign languages comprise many hand-shapes representing words and letters of the language, and in order to differentiate between all of them, a set of features and characteristics should be produced to make it possible for classifiers to recognize each sign correctly, depending on the method in which the information is acquired.

Features could be gathered from spatial or frequency domain, for instance geometric properties, DCT Coefficients (as in [11]), or any related kind of meaningful parameters can be used to uniquely identify each sign.

In fact, a lot of works and studies has considered this field as a huge

concern due to its significant importance, for example [16] proposed in their paper some features to be the state of the hand gestures given by the attributes called Point of Interest (POI) of the hands. they consider two POIs in order to represent the "shape" and "direction of movement".

Table 2.1 explains the feature vector.

Feature type	Elements	Notes
5-finger tip elements	5	$\{A, B, C, D, E\}$: $(1 \leftrightarrow present, 0 \leftrightarrow sent)$
Motion vector elements	4	-
MV sequence	6	-
Wavelet transform of the Fourier transformed image of a gesture	40	-
Total	55	-

Table 2.1: Feature vector used in [16]

Fingers attributes such as localization, presence, distances, etc. also can specify the gesture. For example, [10] extract active fingers attributes for digit recognition and [6] also uses fingers attributes along with time-series curve in order to classify different signs.

[17] makes use of hierarchical hand model in which the hand is represented as (i) the palm as a coarse scale blob, (ii) the five fingers as ridges at finer scales and (iii) finger tips as even finer scale blobs.

2.4 Classification

The main purpose of classification techniques is to differentiate between signs (i.e., classify a specific hand-shape as being a certain word or letter in the subject sign language).

A variety of classification models has been introduced to solve hand-shape recognition problem, including Artificial Neural Networks

(ANN), Support Vector Machine (SVM), and Decision trees.

In the following method, [5] did a large proportion of neural networks research and indicated that the use of a single network would not be appropriate. Therefore SLARTI systems consist of several networks, and the connection together of these form the final system.

In contrast, [18] accomplished good results in classifying signs using Support Vector Machines (SVM). SVM is a linear classifier which is based on the principle of maximization of margin between two sets of data. It is a powerful machine learning technique for classification and regression. Figure 2.1 shows the SVM in the proposed system by [18].

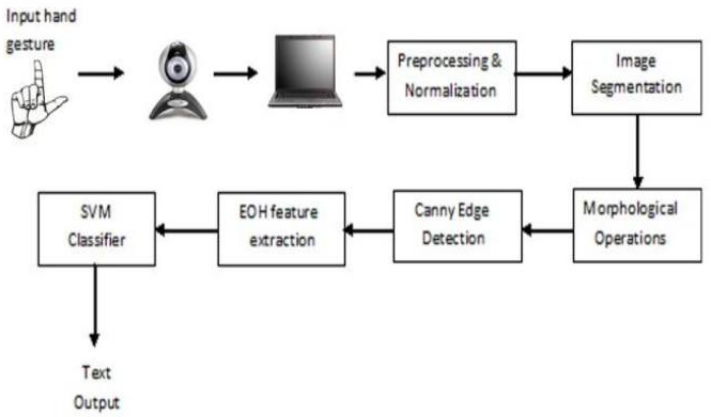


Figure 2.1: Using SVM in a gesture recognition system

Other studies proved that the use of Fuzzy Logic could be also promising, Benjamin C. *et al.* achieved in [19] the recognition of some hand gestures with data obtained from a data glove, after calculating how much each finger is bent, since any sign shape can be represented as fingers shapes. the proposed fuzzificator system accepts 3 points for each finger and other 4 points represent the septation between two fingers, and depending on the different states for these angles, a finger is determined as straight, curved or bent.

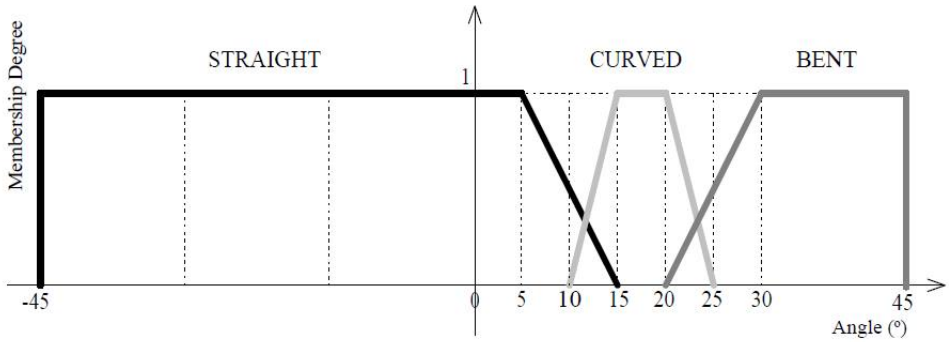


Figure 2.2: Fuzzification of the linguistic variable of the thumb finger

2.5 Preface of our work

Depending on the previous discussion in this chapter, we decide to work with an image of the signer with their palms present, to recognize the sign among several signs representing letters of the ASL (ASL alphabet is shown in figure 2.3).

It is necessary to mention that a subset of any sign language other than ASL can be used in this work in the same manner knowing that the chosen signs must follow the specifications mentioned in each of chapters 3, 4 and 5, but the ASL is specifically chosen because it is one of the most widespread sign languages, it maps some signs to Latin letters and there is much work and research being conducted on the recognition of its sign, so it gives us (for now or in the future) the ability to compare our work with a large number of works.

This work have nothing to do with signs which need motion, only static signs which can be recognized seeing the palm of the signer are considered. The selected set of signs also have to be separable seeing only the shade of the palm (i.e., the binary mask). The chosen subset will be explained in details in section 6.1.

This will be done according the three steps: hand localization, feature extraction and classification (figure 2.4) using different approaches for each step and then comparing the proposed approaches

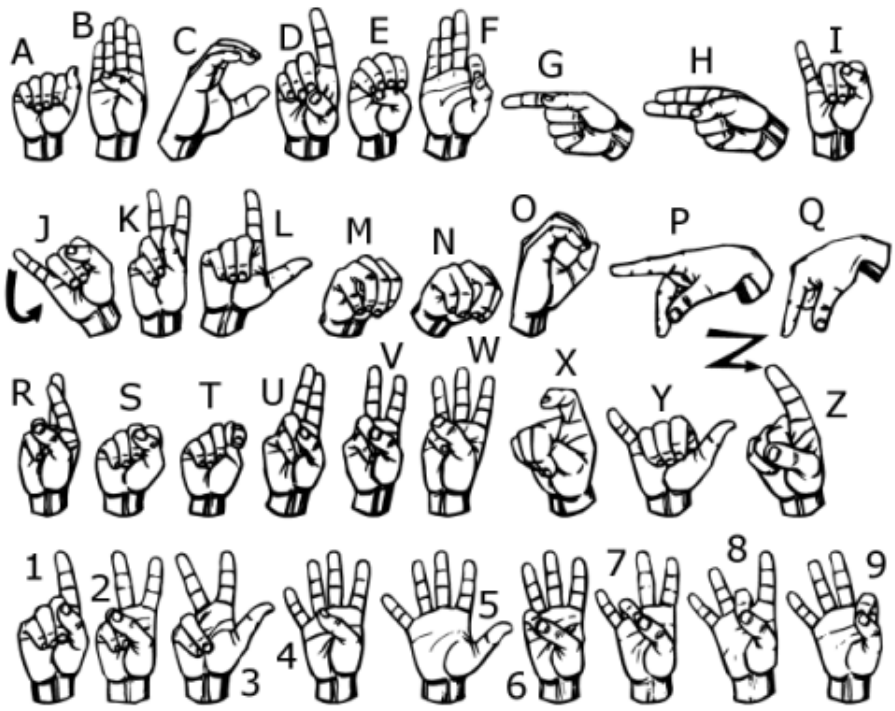


Figure 2.3: American sign language alphabet

and also comparing them to other related works.

No depth sensors, special data gloves or any high-tech devices will be used in image acquisition process, only regular cameras.

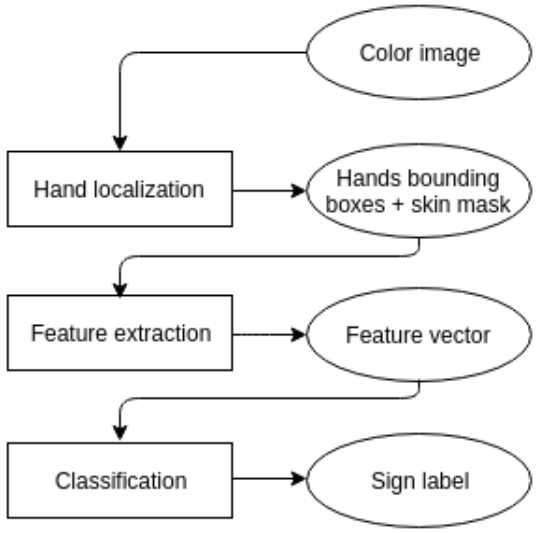


Figure 2.4: Project pipeline and outputs

3 Image Processing and hand localization

In this chapter, we discuss the proposed methods to localize hands in the picture and compare them.

This phase considers an input of a RGB image containing the face and hands of the signer with a background of colors which are mostly different from the signer's skin color. The hands and face have to be relatively big enough in the picture.

The output of this phase is the bounding boxes of the two hands and two mask images representing the hands pixels in the corresponding bounding boxes.

The proposed methods mainly depend on the skin-color filters; some of them use static skin filters with hard-coded known ranges of values in different color spaces (HSV, YCbCr), and some of them use dynamic skin-color filters which select the thresholds depending on the given image. Then we locate the hands using a naive method which depends on the sizes of the skin regions in the image.

Basically, we propose two different approaches, each of which has many options to be implemented. The first approach filters the skin using some general skin detectors and then localizes the hands. The second one depends on the signer's face to learn about their skin and then makes use of the face position to localize the hands more precisely.

3.1 First approach: Naive skin filtering and hand localization

In this approach, the skin is filtered using one of the proposed filters and then the hands are located in the resulting mask. Let's first view the proposed skin filters and then discuss the hands localization method.

3.1.1 Skin-color filters

Three different skin filtering methods are proposed in this approach; the first two of them use static skin-color filtering, while the last one uses a dynamic skin-color filtering.

The static skin filters proposed depend on converting the input RGB image into a different color space and then selecting skin pixels which are of a certain hard-coded range of values for each color component.

The main drawback of the RGB color space is that the luminance and chrominance are not separated in different color components, so R,G and B contain both luminance and chrominance information; this makes thresholding sensitive to lighting changes and the variance of darkness for different skin colors.

Dynamic skin-color filters have no hard-coded ranges but some threshold finding technique which is used to classify each pixel as skin or non-skin.

The output of this sub-phase is a mask of the picture representing possible skin pixels.

3.1.1.1 HSV static skin detector

The HSV color space isolates the intensity (value) of the image in the V component. The Hue, Saturation and Value of the skin fall mostly in fixed ranges of values. According to [13, 20] the values of the H, S and V of the skin color fall within these ranges

Figure 3.1: Skin filtering using HSV ranges: (Left column) original RGB images, (Right column) Images after skin filtering

$$V \geq 40$$

$$0.2 < S < 0.6$$

$$0 < H < 25 \, or \, 335 < H < 360$$

Experimentally, this detector gave unstable results in accordance to different lighting conditions; sometimes it would detect the skin regions and sometimes it would not.

3.1.1.2 YCbCr static skin detector

The YCbCr color space is preferred above RGB since it isolates the luminance in a single component Y and the chrominance in Cb and Cr. This provides the ability of filtering the skin color almost regardless of the skin color variations. According to [14], the chrominance values of the skin generally fall within this range

Figure 3.2: Skin filtering using YCbCr ranges: (Left column) original RGB images, (Right column) Images after skin filtering

$$77 \leq Cb \leq 127$$

$$133 \leq Cr \leq 173$$

However, this method for filtering did not really work smoothly in our experiments, especially in different lighting conditions. We refer this problem to the difference in the hardware used although these values were adopted by several previous works such as [13, 10]. The main drawback is that the values close to this range but lie out of it are considered non-skin pixels, and this results in many false negatives. This obstacle could be skipped if we smoothed the filtering process (which we are going to do in the second approach), but the problem of this static range can be studied later since it is not the aim of our work.

Besides, we tested different ranges from other works [15, 21], but the performance in different lighting conditions kept being unstable.

3.1.1.3 Dynamic Intensity filtering

A very simple, yet very fast method is to select a threshold of the intensity image using Otsu's threshold selection method [22] and then threshold the intensity image using the resulting threshold. This method requires a dark background and a white skin and thus, it is difficult to filter the skin when the user's skin is dark. This is the main drawback of this method as well as the background restriction. An example of this filtering is shown in figure 3.3.

Figure 3.3: Skin filtering using intensity filtering: (Left) original RGB image, (Right) Images after skin filtering

3.1.2 Hand Localization

For this sub-phase, we use a pretty naive method. Given that the face and the hands are present in the picture and composing a relatively large portion of the image and that the hands appear larger than the face, after getting the mask of the skin, we can simply deduce that the largest two components are the hands of the user.

This method for hands localization fails when the face region is connected to one of the hands regions and when the face comprises a larger portion than one of the hands does.

3.2 Second approach: extracting the skin depending on face color

This approach consists of three main steps (face detection, skin filtering and hand localization). First of all, we extract the face of the user to learn about their skin color in the target image. Secondly, we filter out the pixels of the image which are very different from the user's face color in CbCr space, and so we get the possible skin pixels. Finally, we localize the hands in two bounding boxes.

In this section we discuss the three steps in detail. Generally, this approach considers an input of a RGB image containing the face and hands of the user with good lighting and with background colors relatively different from the skin color of the user.

3.2.1 Face detection

The most vital step of this approach is to detect the face. Viola & Jones cascade object detector [23] is used to detect possible faces in the image. This results in multiple bounding boxes denoting possible faces. Since it is required that just one face is present in the image, and the background colors are different from user's skin color, the required bounding box is expected to be the box with colors close to skin color, so the box with closest colors is selected as a face bounding box.

The detection of the face color eases the filtering of the skin; it gives information about skin color specifically in the target image. This approach is adaptive with lighting changes and variance of skin colors [12] which may differ from one image to another.

In Figure 3.4, possible faces on the left image are the output of the Viola & Jones detector, while the right one denotes the actual face of the image (i.e., the closest to human skin color).

Figure 3.4: (Left) Faces proposed by Viola & Jones detector, (Right) The box closest to human skin color

3.2.2 Skin filtering

The aim of the face detection step is to give information about user's skin color in the target image. The face image is converted into $CbCr$ space (i.e., blue chrominance and red chrominance). Figure 3.5 shows a picture with the face detected as explained in Section 3.2.1; this is the case we are studying to show the steps of this approach. Before continuing, the face bounding box is shrunk to contain only parts from the face without anything outside the face.

Figure 3.5: Face detected in an input image (First step of in the previous section)

To extract the skin color pixels, we need a measure which tells us the probability for each pixel of being skin. To get this measure, we first calculate the median of the face region in both Cb and Cr components (i.e., (M_{Cb}, M_{Cr})). The distribution of Cb,Cr values in both the whole image and the face image as well as the median (M_{Cb}, M_{Cr}) are all shown in Figure 3.6. Second, we calculate for each pixel, $P(i,j)$, in the image the **euclidean** distance of its Cb, Cr components (i.e. $(P(i,j)_{Cb}, P(i,j)_{Cr})$) from the median (M_{Cb}, M_{Cr}) in the $CbCr$ space (the mean is not feasible in this case because of its sensitivity to outliers, e.g. eyes, facial hair, nose holes, lips, etc.). The result is an image, D which we call the *distance image* and which is calculated as follows for each pixel $P(i,j)$

$$D(i,j) = \sqrt{(P(i,j)_{Cb} - M_{Cb})^2 + (P(i,j)_{Cr} - M_{Cr})^2}$$

The lower $D(i,j)$ value is, the closer is $P(i,j)$ to skin color, so dark pixels in distance image have higher probability of being skin.

3.2.3 Distance image thresholding

The resulting distance image should now be thresholded to classify each pixel as skin/non-skin. Two methods were used for thresholding.

3.2.3.1 Otsu's threshold selection

This gray threshold selection method [22] minimizes weighted within-class variability, which means it would be misleading for this approach if the background contains regions that are much closer to skin color than other regions (they might be counted as skin) or if the skin comprises a small portion of the image.

To get the threshold, MATLAB's `graythresh` which is an implementation of this method is used . The image is then thresholded (and then negated) so that it is split into two classes (i.e., skin as white pixels and non-skin as black pixels).

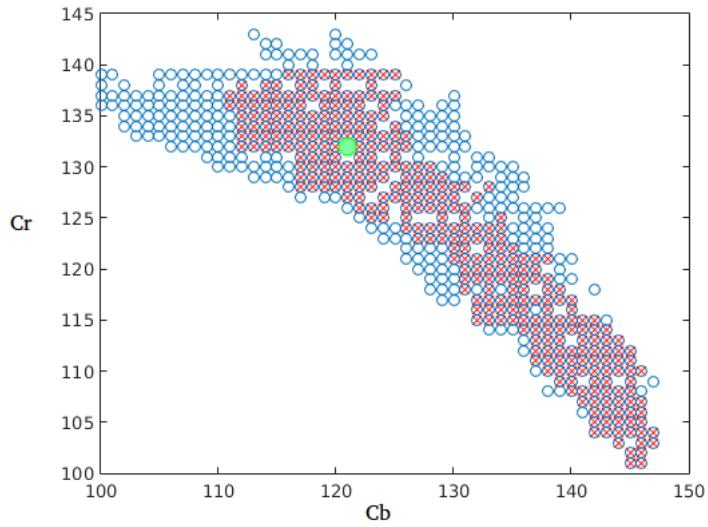


Figure 3.6: Distribution of Cb,Cr values in the input image: blue region denotes all Cb,Cr values, red region denotes the face Cb,Cr values and the green disk denotes (M_{Cb}, M_{Cr})

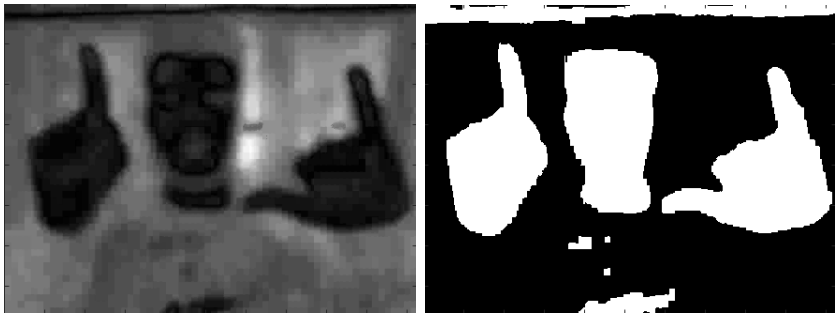


Figure 3.7: (Left) Distance image, (Right) Skin regions

The distance image as well as the skin-color regions is shown in Figure 3.7.

Otsu's method works perfectly when the values of the input image (i.e., distance image here) contain two separated clusters; the case in

which the distance image histogram contains two separated hills just as shown in figure 3.8 which is the histogram of the distance image in figure 3.7. However, this threshold selection method sometimes gives unwanted results specifically when the values of the distance image contain more than two clusters as shown in figure 3.9. Another proposal method to overcome this problem is shown in section 3.2.3.2.

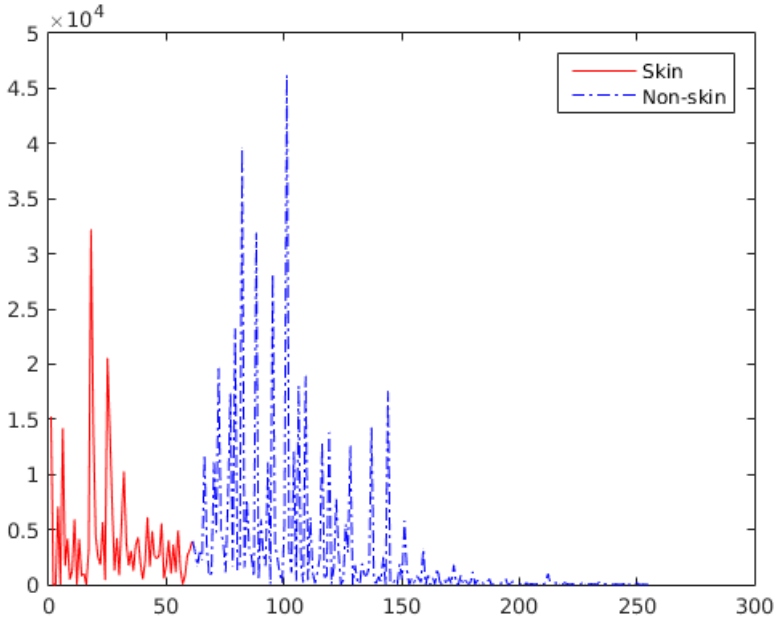
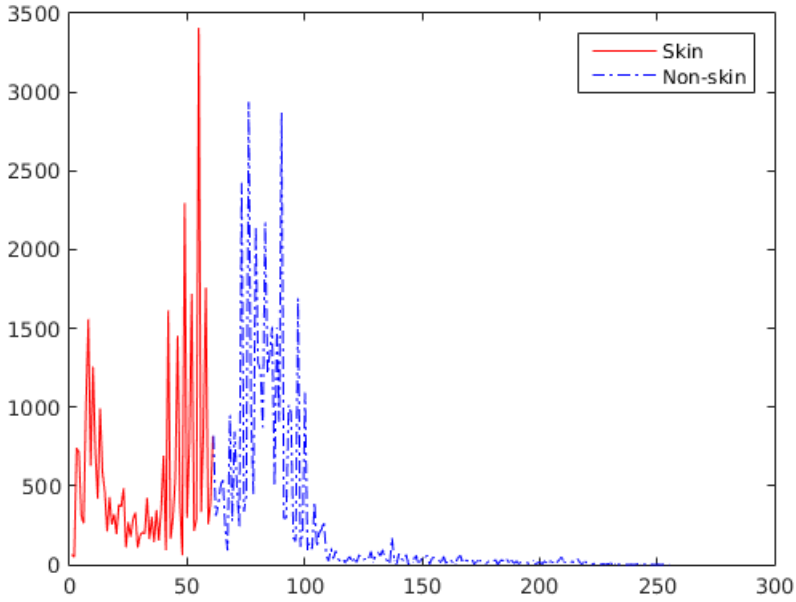


Figure 3.8: The histogram of distance image in figure 3.7 separated

3.2.3.2 Thresholding with hysteresis

Thresholding with hysteresis, first defined in Canny's edge detector [24], is a widely accepted technique for picking up features that are not very strong by themselves but whose significance gets boosted by its spatial proximity [25]. In our work, we invert the distance image and perform hysteresis thresholding.

(a) RGB image (b) distance image (c) skin regions



(d) histogram of distance image separated by Otsu's method

Figure 3.9: Otsu's threshold selection

(a) RGB image (b) distance image (c) skin regions

Figure 3.10: Thresholding with hysteresis

Figure 3.10 shows how the problem mentioned in 3.2.3.1 (see figure 3.9) was solved using hysteresis thresholding.

3.2.4 Hand localization

After getting the skin mask, the hands should be wrapped in two bounding boxes. The mask image consists of several white components; one of them is the face and two others are the hands plus some other components that are close to skin color. Since we know the position of the face, we can simply eliminate the face component from the image as shown in Figure 3.11.

We assume that the largest two components in the result image represent the hands, and any other component is considered noise, so their bounding boxes are the required result of this step as shown in Figure 3.12.

It is noticed that the hands of the user should not cover any part of the face, or else they will be eliminated in this step and no sufficient information about the hands might be retrieved. They also cannot cover each other or touch each other, or else they will be considered a single component.

Having extracted the hands from the picture, we can now move to the next phase in which we are going recognize the gesture of each hand.

Figure 3.11: Skin regions with face eliminated

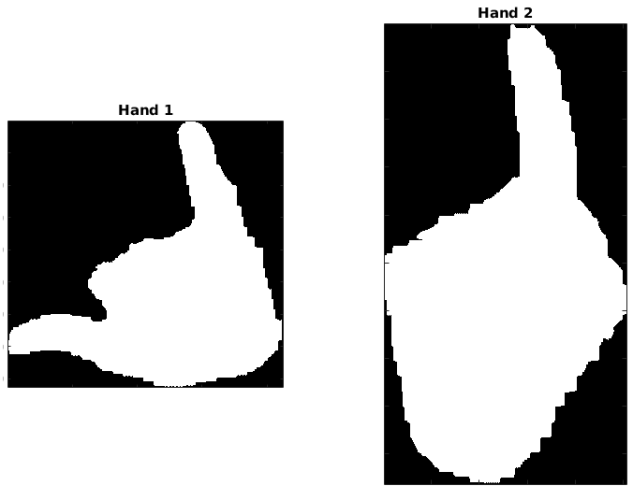


Figure 3.12: Extracted hands

The proposed method in this section gives the ability to robustly detect the hands in a good lighting environment with a non-deceiving background. This flowchart (Figure 3.13) summarizes the proposed

method.

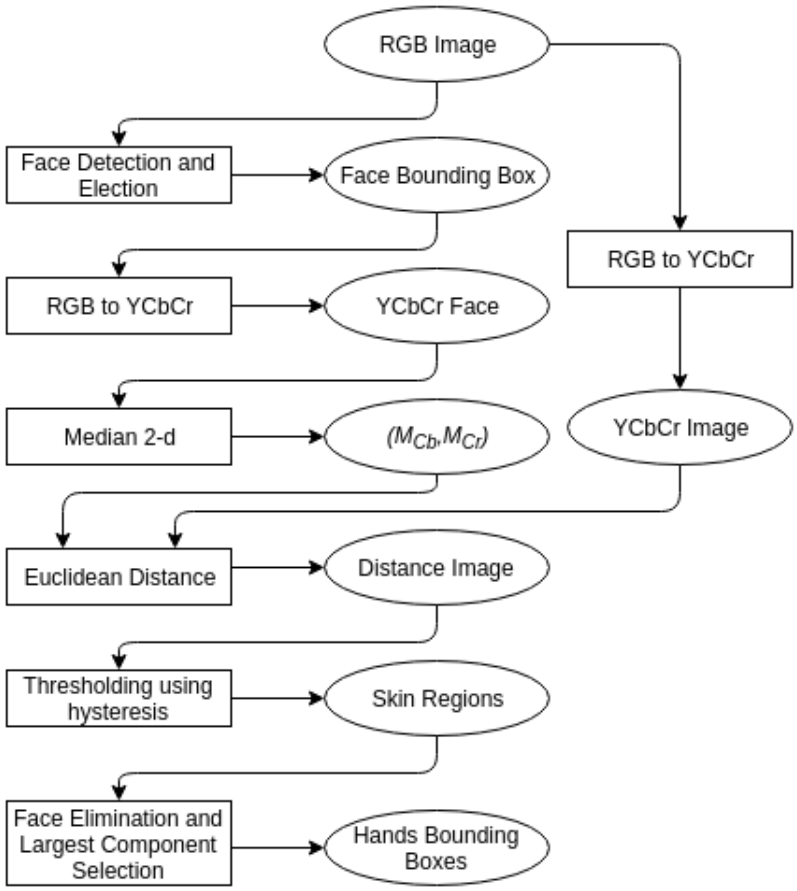


Figure 3.13: Face dependent approach flowchart

4 Feature Extraction and Data Acquisition

The next step in hand recognition process is to extract features, properties, information, or characteristics from the resulted binary image, however this step is one of the most important steps, because these acquired data will be later the only information that a classifier model would depend on to recognize different gestures. Many systems use gloves or sensors to acquire specific features of the hands. The measurements (orientation, velocity) were measured directly using a sensor.

4.1 Geometric shape descriptors

As we are extracting our features from binary images, we can capture information about the white region in the image, which represent the hand-shape itself.

The following features are extracted

4.1.1 Eccentricity

A scalar that specifies the eccentricity of the ellipse that has the same second-moments as the region.

The human hands have an elliptic shape [10], if we could find the ellipse that fits the hand, then the eccentricity is the ratio of the distance between the foci of the ellipse and its major axis length, and its value is between zero and one.

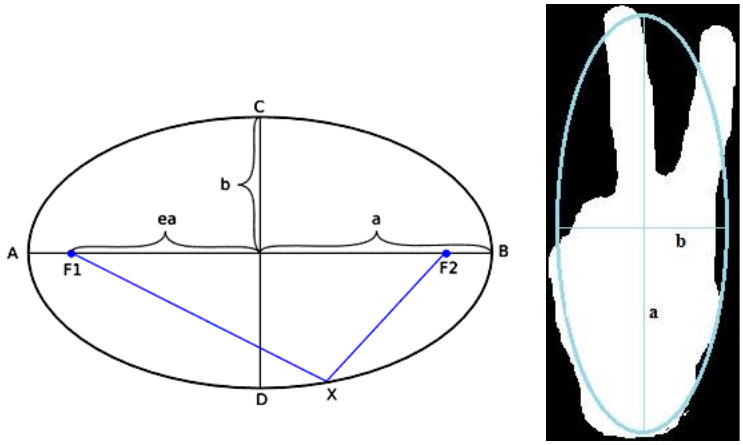


Figure 4.1: (Left) The ellipse which fits the hand, (right) ellipse Eccentricity Equation

$$E = \sqrt{1 - \frac{b^2}{a^2}}$$

where $E = eccentricity$, $a = majoraxis$ and $b = minoraxis$

4.1.2 Extent

A scalar that specifies the ratio of pixels in the region to pixels in the total bounding box. Computed as the area divided by the area of the entire bounding box as shown in the equation (also in figure 4.2).

$$Extent = \frac{a}{b}$$

where
a = area of hand component

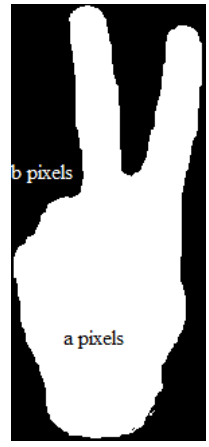


Figure 4.2: Binary Shape Extent

b = area of bounding box

4.1.3 Solidity

A scalar specifying the proportion of the pixels in the convex hull that are also in the region. Computed as follows (also shown in figure 4.3 in which the convex hull is represented using a blue line bounding the hand component).

$$Extent = \frac{a}{c}$$

where
a = area of hand component
c = area of convex hull

Note: *convex hull boundary of an image may be thought of as the outline of a "rubber band" stretched around the image.*

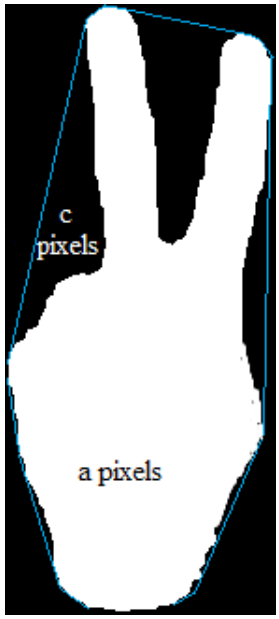


Figure 4.3: Binary Shape Solidity

4.1.4 Roundness

Roundness as a formula is defined as

$$roundness = \frac{4 \times area}{\pi \times (majoraxis)^2}$$

- *area* is number of white pixels in the image
- *majoraxis*, as defined in MATLAB documentation, is a scalar specifying the length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region.

Actually, if the shape is already a circle, it will satisfy $area = \pi \times r^2, majoraxis = 2 \times r$

So its roundness will be $\frac{4\pi r^2}{\pi(2r)^2} = 1$

While when the shape is almost a straight line (or an ellipse with minor axis seeks to zero), then it will satisfy $area \simeq majoraxis = numberofwhitepixels$ and the roundness here will be $\frac{majoraxis}{\pi \times minoraxis^2} = \frac{4}{2 \times \pi \times r}$ where r is length of radius of circle C such that its radii equals the major axis of this shape, and in this case roundness will be $\frac{4}{perimeter\,of\,C}$

So the roundness seeks to one if the shape is similar to a circle and to zero (small number actually) if the shape is a straight-line segment. To map this to our signs set, signs like two and one will have a small roundness value while signs like A (a closed fist) will have a high roundness value.

4.1.5 Circularity

A similar measure is circularity, which defined as

$$circularity = \frac{4 \times \pi \times area}{perimeter^2}$$

Actually, this measure will be equal to one for a circle ($\frac{4 \times \pi^2 \times r^2}{(2 \times \pi \times r)^2} = 1$) but this measure depends on perimeter instead of major axis this provide higher circularity value to shapes with long perimeter even if they have the same major axes, for example ('L' against 'One') 'L' has longer perimeter than 'One' due to thumb perimeter length (figure 4.4).

Figure 4.4: (Left) 'L' sign, (right) 'One' sign

4.2 Invariant Moments

Moment invariants have become classical measures for object recognition recently. No doubt, they are one of the most important and most frequently used shape descriptors. Hu was the first setting out the mathematical foundation for two-dimensional moment invariants and demonstrated their applications to shape recognition; he defines seven of these shape descriptor values computed from central moments through order three that are independent to object **translation**, **scale** and **orientation** [26].

Traditionally, moment invariants are computed based on the information provided by both the shape boundary and its interior region. The moments used to construct the moment invariants are defined in a continuous space but for practical implementation they are computed in a discrete form [27]. Given a function $f(x, y)$, these regular moments are defined by

$$M_{pq} = \int \int x^p y^q f(x, y)\, dx\, dy \tag{4.1}$$

M_{pq} is the two-dimensional moment of the function $f(x, y)$ the order of the moment is $(p+q)$ where p and q are both natural numbers.

For implementation in digital from this becomes

$$M_{pq} = \sum_X \sum_Y x^p y^q f(x, y) \tag{4.2}$$

To normalize for translation in the image plane, the image centroids are used to define the central moments. The co-ordinates of the center of gravity of the image are calculated using equation 4.2 and are given by

$$\bar{X} = \frac{M_{10}}{M_{00}} \quad \bar{Y} = \frac{M_{01}}{M_{00}} \tag{4.3}$$

The central moments can then be defined in their discrete representation as

$$M_{pq} = \sum_X \sum_Y (x - \bar{x})^p (y - \bar{y})^q f(x, y) \tag{4.4}$$

The moments are further normalized for the effects of change of scale using the following formula

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{4.5}$$

Where the normalization factor: $\gamma = (p + q)/2 + 1$. From the normalized central moments, a set of seven values can be calculated and are defined by

$$\varphi_1 = \eta_{20} + \eta_{02}$$
$$\varphi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$\varphi_3 = (\eta_{30} - 3\eta_{12})^2 + (\eta_{03} - 3\eta_{21})^2$$
$$\varphi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{03} + \eta_{21})^2$$

$$\varphi_5 = (3\eta_{30} - 3\eta_{12})(\eta_{30} + \eta 12)[(\eta_{30} + \eta_{12})^2 - 3(\eta_{03} + \eta_{21})^2]$$
$$+(3\eta_{21} - \eta_{03})(\eta_{03} + \eta_{21}) \times [3(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2]$$

$$\varphi_6 = (\eta_{20} + \eta_{02}) \times [(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{03} + \eta_{21})$$

$$\varphi_7 = (3\eta_{21} + \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{03} + \eta_{21})^2] + (3\eta_{12} + \eta_{30})(\eta_{03} + \eta_{21}) \times [3(\eta_{30} + \eta_{12})^2 - (\eta_{30} + \eta_{21})^2]$$

These seven invariant moments, φ_I, $1 \leq I \leq 7$, set out by Hu [26], were additionally shown to be independent of rotation. However, they are computed over the shape boundary and its interior region.

4.3 Active fingers attributes

Information about the distribution of the fingers in the image gives a good separation between the different classes. Namely, we use the number of finger components, mean and standard deviation of finger components areas over the area of the hand component and the mean and standard deviation of finger components centroids squared euclidean distance from the center of the hand over the area of the hand. Here, if two or more fingers are joined or attached to each others, they are considered a single finger component which means the number of finger components actually represents the number of joined groups of fingers. This will not affect the feature extraction process because the area and distance properties will also help us separating the classes.

Before we explain the features, let's show the method used to get the finger components out of a binary image of the hand component. This method mainly depend on the approach shown in [6].

First of all, we apply distance transform on the binary image to get an image DT where each pixel $DT(i, j)$ is the distance transform of the pixel $P(i, j)$ in the image P.

$DT = distance_transtorm(\bar{P})$ where $\bar{P}$ is the inverse of image P.

The distance transform defines, for each pixel, the euclidean distance from the nearest non-zero pixel, which means the pixels with high DT value (brighter color in middle figure of 4.5) have more chance to be the center of an object. Secondly, we choose the center which is the pixel with the highest value of DT.

$C = \{(i, j) : DT(i, j) = max(DT)\}$

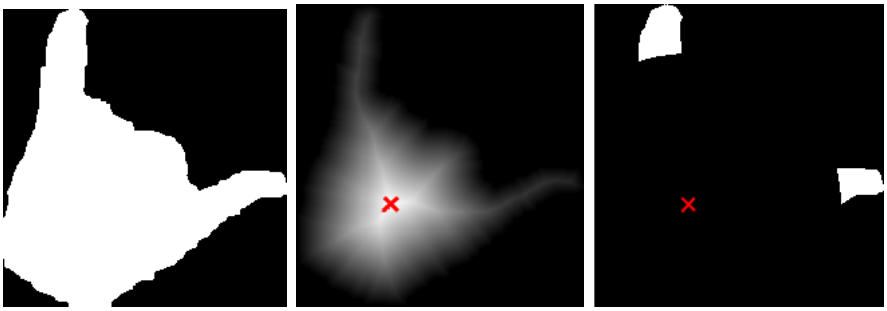


Figure 4.5: (Left) Raw hand binary image, (middle) distance transform with the center shown, (right) finger components

Finally, we eliminate the pixels $P(i, j)$ which are close to C and thus we get the fingers isolated. The eliminated pixels are the pixels whose euclidean distance from hand component center is less than $DT(C) \times \alpha$, where $\alpha = 1.9209$ is an experimental constant which we got from testing several values and choosing the one with least error for our data set. In short, all pixels within a circle of center C and radius $DT(C) \times \alpha$ are eliminated.

Figure 4.5 shows an example of the process, the picture to the left shows a sample from our data set, the picture in the middle shows the distance transform of this sample with a red mark denoting the center of the hand component and the picture to the right shows the hand after eliminating the pixels inside the mentioned circle. The chart in figure 4.6 summarizes the process of finger isolation.

Here, we explain each type of the proposed features.

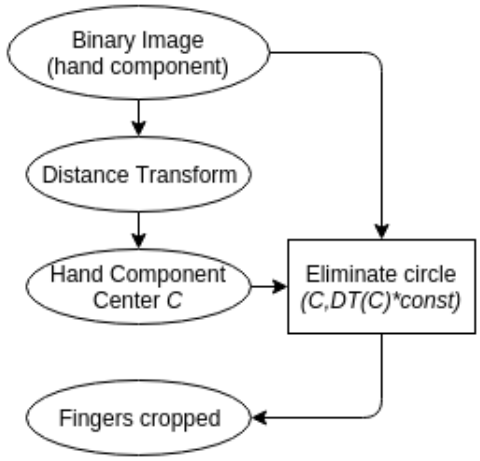


Figure 4.6: Illustration of fingers separation method

4.3.1 The number of finger components

Intuitively, the number of fingers presented in the image separates different classes from each others and gives much information about the class to be predicted. For example, in the chosen 8 signs, there are some signs that have 2 fingers presented (e.g. signs V, L and Y) while others have one finger presented (e.g. signs B and one).

4.3.2 Areas of finger components

The area of each finger component is computed and divided by the area of the whole hand to regularize the measurements and to give details about how large is the finger component relatively to the hand component. Area fractions are the features considered in this step.

As shown in figure 4.7, green color denotes the finger components while the red color denotes the other parts of the hand. For each component (in green), we calculate its area and divide it by the total area (purple + green). This division is vital to regularize the feature value and thus, make it insensitive to the changes in the size of the image.

The image to the left (letter Y) in figure 4.7 has an average area rate of 0.0161 and a standard deviation of 0.0061 (finger components are small and relatively of the same size) while the image to the right (letter C) has an average of 0.0635 and a standard deviation of 0.0410 (the two finger components are relatively bigger than the previous one and somehow not very similar in size)

Figure 4.7: Areas of finger components

Note: A feature of the sum of finger components areas divided by the total area of hand component is also added.

4.3.3 Distance of the finger components from the hand center

After isolating the finger components and getting the center of the hand component, we calculate, for each finger component, the distance of its centroid (center of mass of the hand component) from the center of the hand component.

The mean and standard deviation of these values are also added as features. In figure 4.8, each finger component has its center represented by a "+" sign; the squared euclidean distance of this sign is

calculated and then divided by the area of the hand component. The squaring and division are also important in this step for the same reason.

The image to the left in figure 4.8 has an average distance value of 0.1787 while the image to the right has an average of 0.5633 even though they might seem to have very similar distances, and this is thanks to the regularization process we mentioned above.

Figure 4.8: Distance of center for each finger component

4.4 Using principle component analysis (PCA)

PCA is a transformation used to reduce the size of some input and summarize its data by mapping it into a lower number of dimensions.

The approach we are using to get components is based on SVD (singular value decomposition). A singular value decomposition provides a convenient way for breaking a matrix, which perhaps contains

some data we are interested in, into simpler, meaningful pieces [28]. For more information about SVD theory you can read [29]

In this section, we will show how we used PCA to visualize and train the machine learning structures.

Since we are dealing with binary images, we can define binary image as $W \times H$ matrix P so that $P_{i,j} \in \{0, 1\}, i \in [1, W], j \in [1, H]$ where W is the width of the binary image and H is the height. During this process, all images will have the same size.

In fact, any binary image is representable as a point in $W \times H$ dimension space. Of course, this will cause limitations on visualizing the whole data set we have (on $W \times H$ dimension plane, so we may use PCA to get an approximate view of our data set in order to figure out which signs are easy to separate (linearly separable) and which are not.

Visualization In order to be able to extract features that efficiently separate the eight classes we have, we will use PCA to check whether first two or three components are enough for separating our data.

First, we will use first two principal components $coeff1$ and $coeff2$ to visualize some pairs of different signs we have. This will help us to find pairs that are hard to classify using only first few principal components and need more features (like geometric and shape dependent features).

Note: all pairs visualization in 2-D and 3-D are discussed in appendix **??**.

Some pairs of signs are easily separable even using a linear classifier [30] such as SVM with linear kernel or linear regression (for some pairs). Some of such pairs are (A,L), (B,Y), (B,C), (B,V), (L,one). Next, we are discussing two different cases in which the signs are linearly separable.

First, we will view the distribution of 'B' and 'V' data samples in $(coeff1, coeff2)$ 2-D space. Figure 4.9 shows one sample from each of 'B' and 'V', and the difference between the two samples. Figure 4.10 shows the distribution of 'B' and 'V' samples throughout the

mentioned 2-D space.

For pairs like 'B' an 'V' decision boundary is clear and we can replace SVM with simpler Machine Learning structure like linear regression, see figure 4.11.

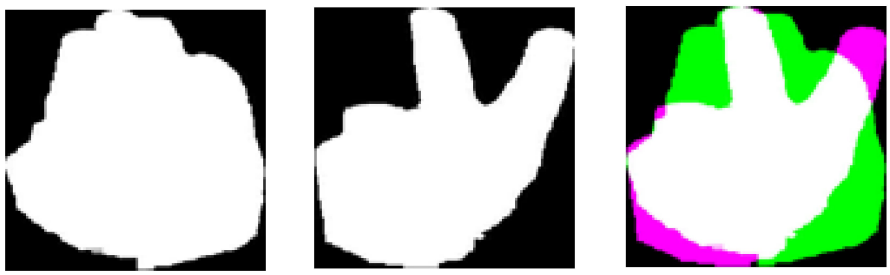


Figure 4.9: (Left) 'B' binary image, (middle) 'V' binary image, (right) difference between 'B' and 'V'

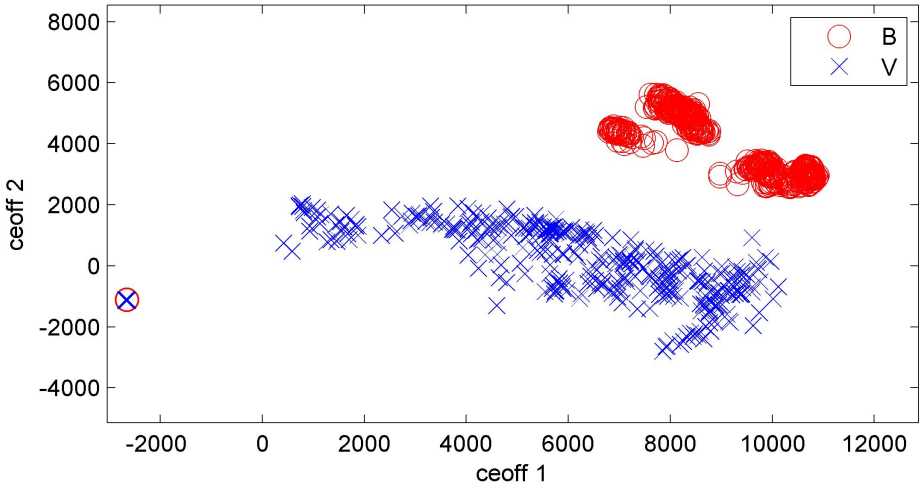


Figure 4.10: Distribution of 'B' and 'V' samples

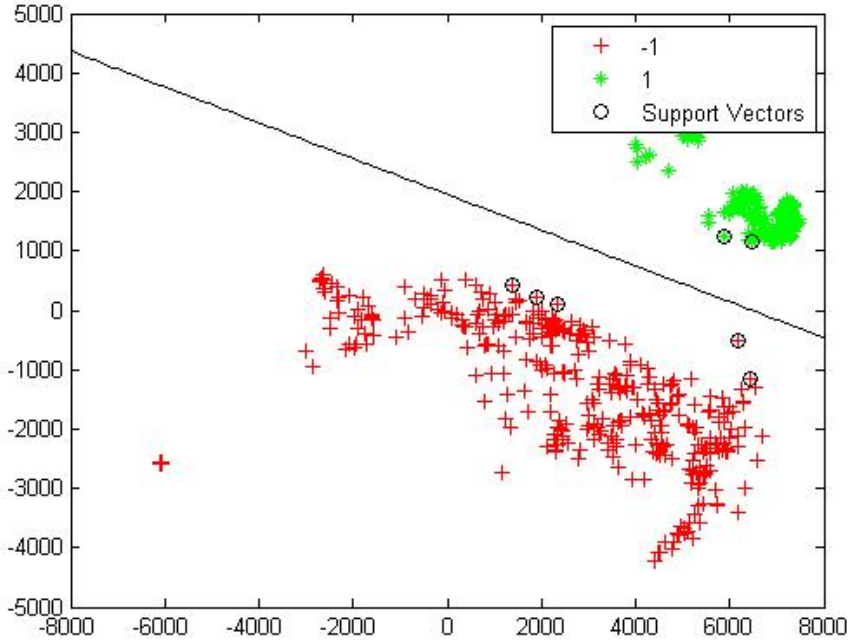


Figure 4.11: 'B' v.s. 'V' binary classification using SVM with linear kernel

Second, we will separate between 'L' and 'One' data samples with linear kernel SVM. Note that we will not separate the data here into training/validation/testing subsets because the objective here is only to show weather data is separable or not. We also show two samples and their difference, plus the distribution of the data samples across the 2-D space in figures 4.12, 4.13.

Note *Classification using SVM will be discussed in chapter 5 in details.*

Figure 4.14 shows the separation between 'L' and 'One' data samples with linear kernel SVM. Data is somewhat separable using linear classifier, but for SVM, this linear boundary costs a lot of support

Figure 4.12: (Left) 'L' binary image, (middle) 'One' binary image, (right) difference between 'L' and 'One'

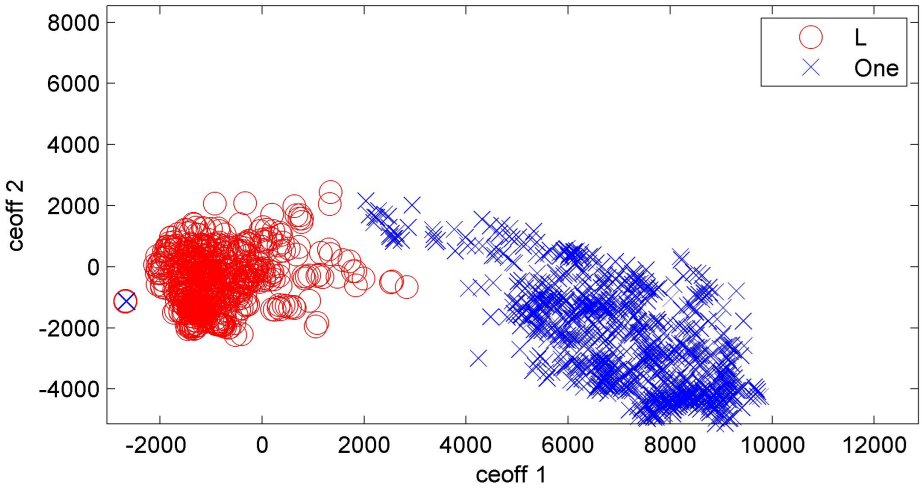


Figure 4.13: Distribution of 'L' and 'One' samples

vectors which may have negative effect on performance.

In order to have better performance (i.e., reduce number of support vectors), we may use more complex kernel such as cubic polynomial kernel. However, here we have a trade-off between complexity of the kernel and the number of support vectors: increasing both may affect negatively on performance. Figure 4.15 explains the separation

between 'L' and 'One' data samples with polynomial kernel SVM.

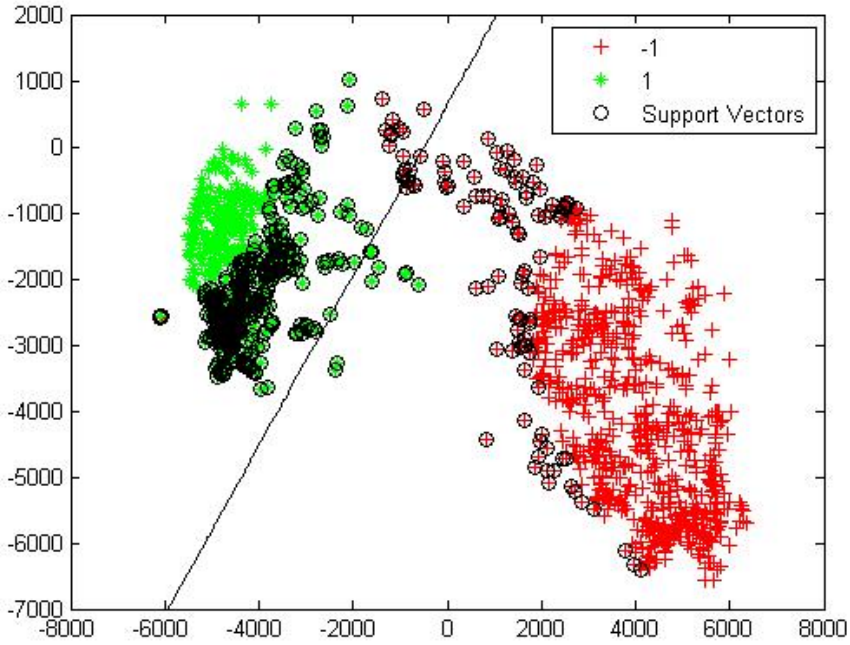


Figure 4.14: 'L' v.s. 'One' binary classification using SVM with linear kernel

Some other pairs cannot easily be separated using this simple linear kernel, but a more complex classifier like a neural network or SVM with polynomial kernel in order to have a polynomial decision boundary. Some of such pairs are (C,Y) and (V,Y).

The first two components are not always enough to separate between labels. Due to limitation of first two components in representing each image, some pairs of signs may be separated using more than two components. For example, (C,Five) ـ shown in figure 4.16 looks easier to separate using three components instead of two.

If we check the distribution of 'C' and 'Five' in the $(coeff1, coeff2)$

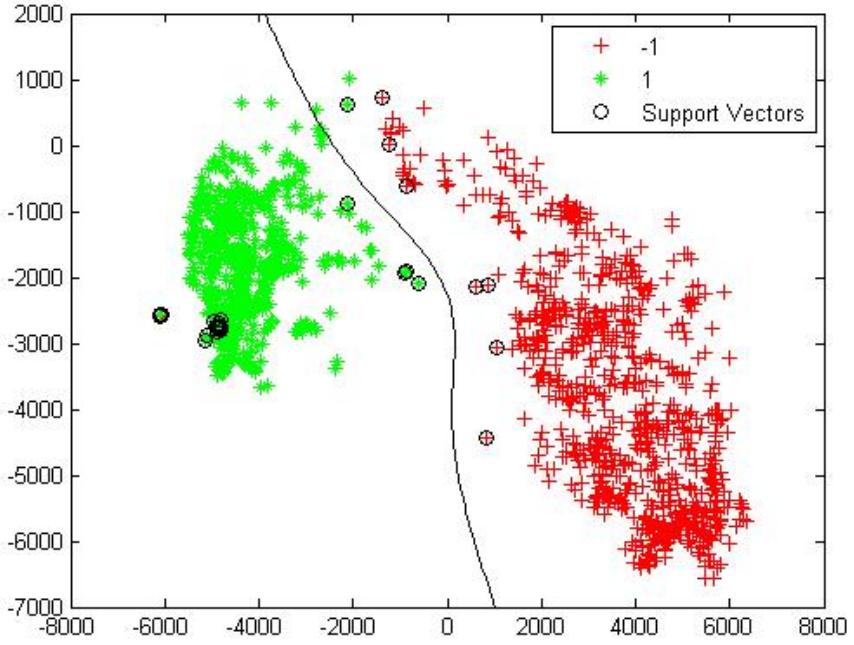


Figure 4.15: 'L' v.s. 'One' binary classification using polynomial kernel

space figure 4.17, we find that such separation is infeasible because of the high interference between the samples. However, if the three components are used instead of two, the separation seems to be much easier. Figure 4.18 illustrates the separation of the two clusters in a 3-D space $(coeff1, coeff2, coeff3)$ even though they look much overlapped in figure 4.17.

Finally, some pairs are not separable easily even if we use three, four or five components such as the pair (L,Y) whose distribution throughout $(coeff1, coeff2)$ and $(coeff1, coeff2, coeff3)$ spaces are shown in figures 4.19 and 4.20 respectively.

Figure 4.16: (Left) 'C' binary image, (middle) 'Five' binary image, (right) difference between 'C' and 'Five'

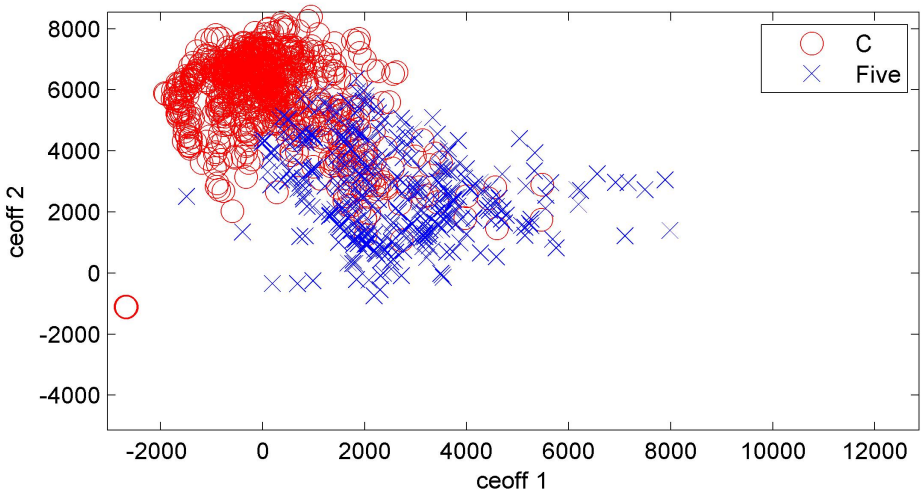


Figure 4.17: Distribution of 'C' and 'Five' samples in $(coeff1, coeff2)$ space

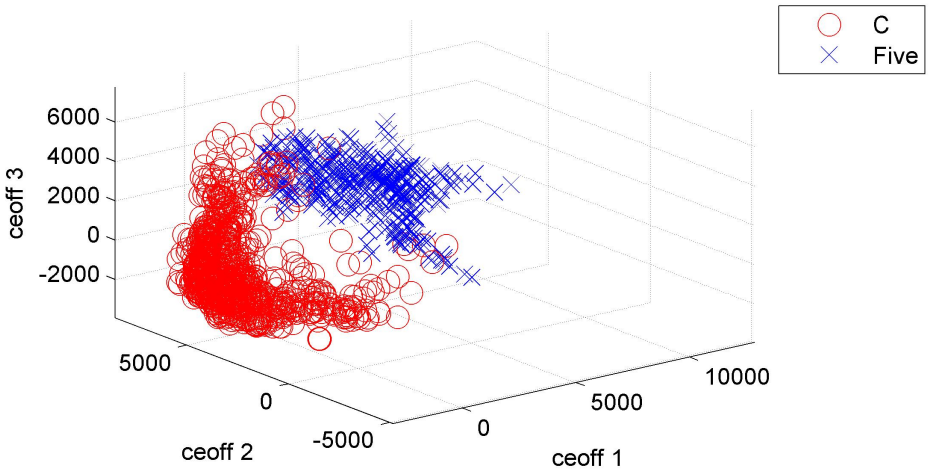


Figure 4.18: Distribution of 'C' and 'Five' samples in $(coeff1, coeff2, coeff3)$ space

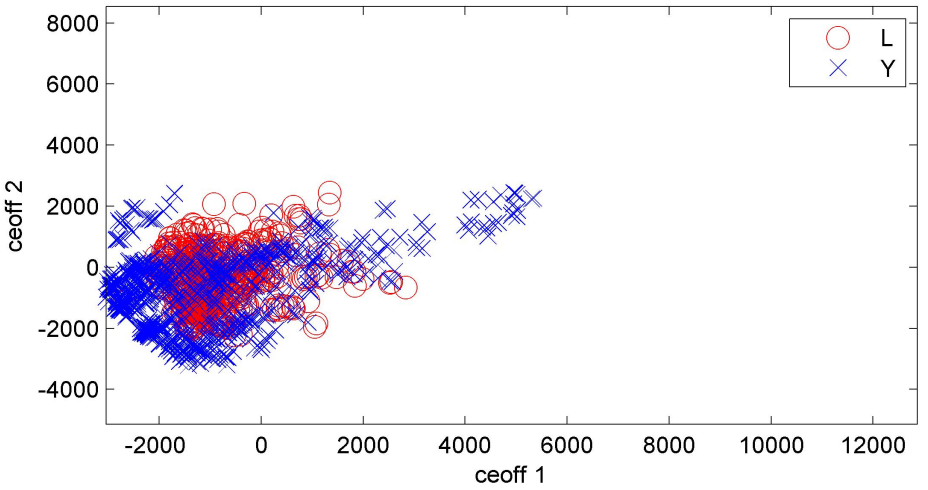


Figure 4.19: Distribution of 'L' and 'Y' samples in $(coeff1, coeff2)$ space

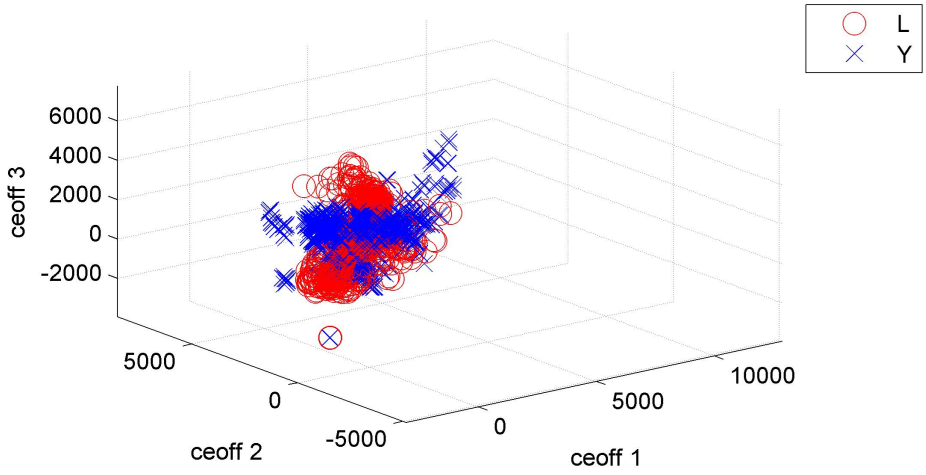


Figure 4.20: Distribution of 'L' and 'Y' samples in $(coeff1, coeff2, coeff3)$ space

5 Classification

After extracting the features from the image and in order to obtain the right hand-shape that being presented, we have to use a classification model based on machine learning techniques, which would recognize different binary shapes depending on several scalar properties (i.e., the features extracted in chapter 4).

5.1 Feed-Forward Back-propagation Neural Networks

A Neural network (also called an ANN or an Artificial Neural Network) is an artificial system made up of virtual abstractions of neuron cells. Based on the human brain, neural networks are used to solve computational problems by imitating the way neurons are fired or activated in the brain. During a computation, many computing cells work in parallel to produce a result. This is usually seen as one of the possible ways artificial intelligence can work. Most neural networks can still operate if one or more of the processing cells fail.

Neural networks can learn, by themselves, an ability which sets them apart from normal computers.

As shown in figure 5.1, a neural network basically consists of many layers, input layer, output layer, and zero or more hidden layers. The input layer contains the properties or features that are captured from the problem being studied and proposed to the model, which will depend on them to decide on a class among multiple classes. Each class is represented with an output node in the output layer.

The input and output nodes are interconnected by links which

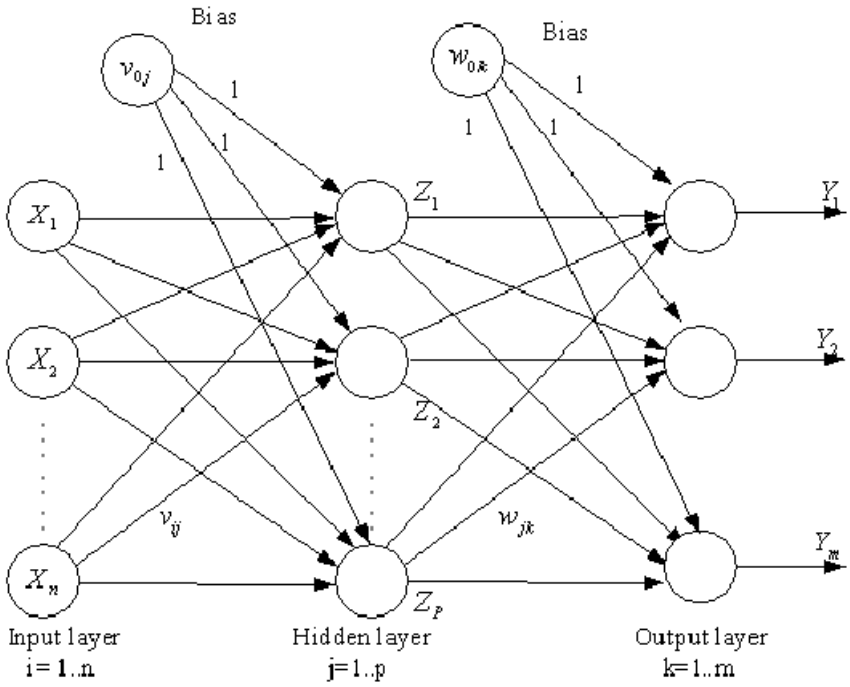


Figure 5.1: Neural Network Model

have weights. In the training phase, the network learns to perform a particular function by adjusting these weights.

Back-propagation neural network works in a way to be trained on a data set of some input so that it adjusts the network's weights every time a new data sample is processed. Then this output is compared to the known output and a mean-square error is calculated. Error value is used to adjust weights from the output layer through hidden layers, until input layer is reached. This process is repeated until the overall error get minimized.

Finally, a well trained network can predict an output class for an unknown or unseen input data examples using the learned model.

In general, a data set is divided up into three sub-datasets, train-

ing, validation and test sets, each sub-dataset is used for a different purpose.

A Training subset basically used for training a classification model by adjusting the weights on the neural network, that is done with variety of labelled examples, each example is represented as a feature vector along with a label class. This label tells the classifier what class to predict for this set of features.

A Validation subset used for controlling a model parameters, such as regularization, that eventually implies minimizing over-fitting.

A validation set also may be used to choose the best architecture (i.e. number of nodes in a hidden layer).

A Test subset is a set of examples used only to assess the performance of the specified classifier.

Usually, 60% of the entire dataset would be dedicated for training set, and 20% for each of validation and test sets.

5.1.1 Our Proposed Model

As previously mentioned, a neural network model consists of several layers; each layer consists of several nodes. Depending on our suggested features (explained in in chapter 4), the input layer designed to accept 27 features which implies 27 input node in the input layer, then one hidden layer with 10 nodes (see section 5.1.2), and finally an output layer with 8 nodes representing 8 signs as figure 5.2 shows.

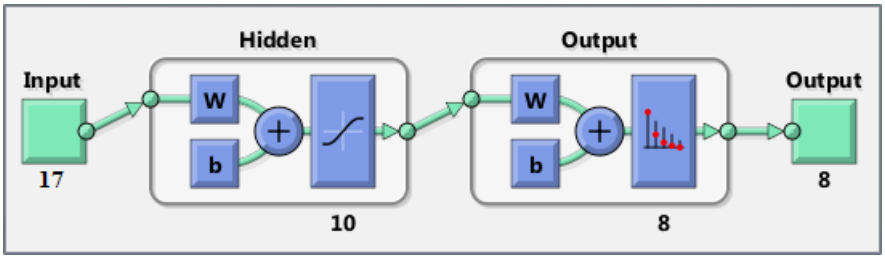


Figure 5.2: Model architecture

5.1.2 Training procedure

The network is trained with initial random weights using back-propagation algorithm, a supervised learning algorithm for multilayer feed forward network, the performance of the network is evaluated calculating the MSE (Mean Squared Error) between network outputs and corresponding target outputs.

After results are achieved, the accuracy has been calculated. Current results are promising having in mind that few improvements could provide better results. After combination of building network with different nodes, we got the results shown in figures 5.3 and 5.4.

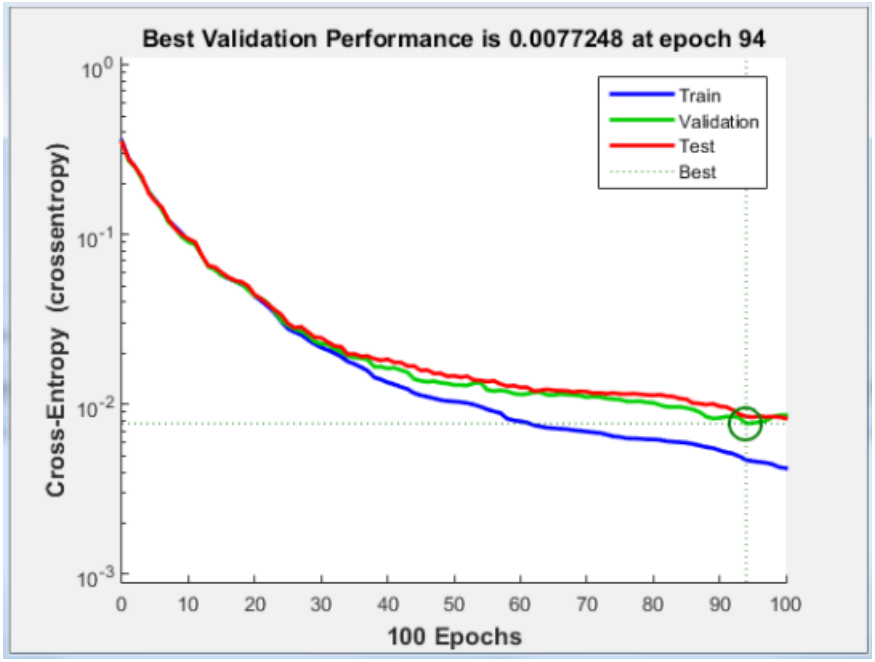


Figure 5.3: Model performance

As shown in 5.3, a plot of the training errors, validation errors and test errors appears. After 100 epochs of training, the result is reasonable because of the following considerations:

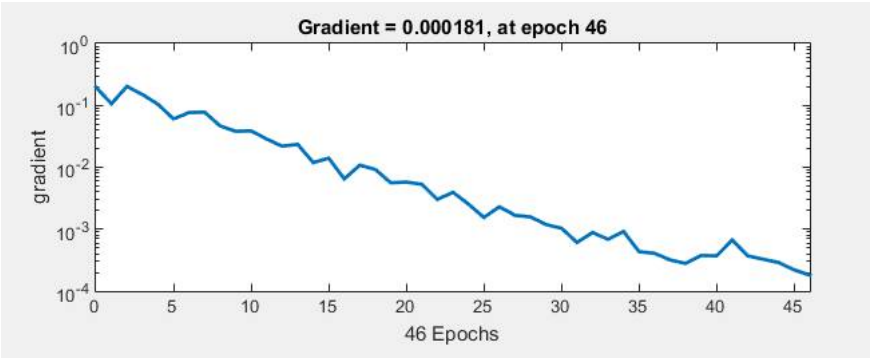


Figure 5.4: Gradient convergence

- The green circle refers to the iteration at which the validation performance reached the minimum. Clearly, the final mean-square error is relatively small.
- The test set error and the validation set error have similar characteristics, that is, the two curves are very similar, indicating that no significant over-fitting has occurred (especially where the best validation performance occurs).

Note: *Validation checks are used to stop training early if the network performance fails to improve or remains the same for (max_fail) epochs in a row.*

Training stops when any of these conditions occurs

- The maximum number of epochs (iterations) is reached.
- The maximum amount of time is exceeded.
- Performance is minimized to the goal.
- The performance gradient falls below min_grad.
- Validation performance has increased more than max_fail times since the last time it decreased.

Figure 5.4 shows variation in gradient coefficient with respect to number of epochs, the final value of gradient coefficient is approximate near to zero, and in general, minimum the value of gradient is, better will be the training and testing of networks.

5.1.3 Training functions comparison

Network training and testing is done using several training functions, considering that we seek for the best performance possible for our model. As seen in 5.5, a comparison between two common training functions the Levenberg-Marquardt algorithm (TRINLM) and the Resilient backpropagation algorithm (TRAINRP).

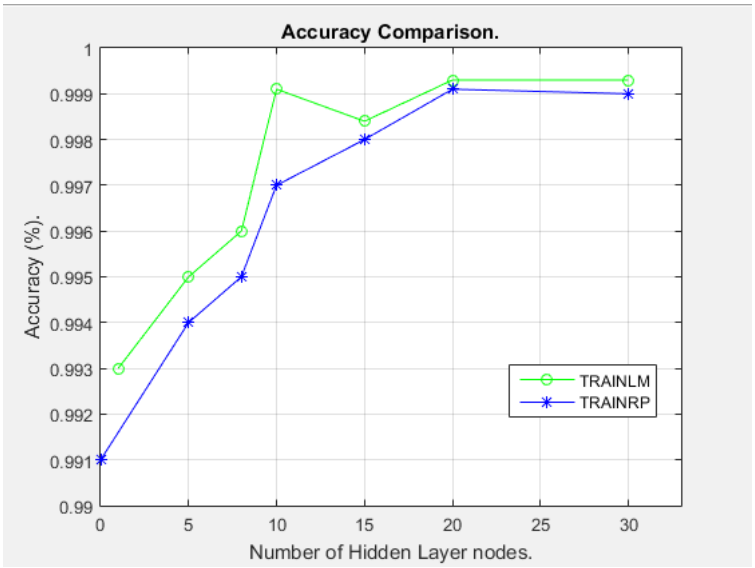


Figure 5.5: Training Functions Accuracy Comparison

The curves shows the effect of number of hidden layer's nodes in increasing the accuracy of the proposed classifier, however, it appears that we achieve better accuracy when increasing the hidden nodes, and that actually very true because in this way the network will have

more parameters to modify the final curve in such a way it fits the data well, in contrast, increasing the hidden layer's nodes very much would make a complex curve which may over-fit the data.

Another observation appears in the figure is that a model trained using Marquardt algorithm gives better accuracies, and that because this algorithm is designed to always make a successful step towards the optimal solution, it may also use a small step size when converging, which results in minimized error, and thus Marquardt optimization is more powerful than conventional gradient techniques.

5.2 Support Vector Machine

Support Vector Machine (or SVM) is a machine learning structure that is capable of classifying by providing optimal decision hyperplane which separates two data classes.

The hyperplane (also known as decision surface) is a subspace of the original features space that contains one dimension less than the original one, for example for 2-D feature space, we will have 1-D line that separate the data classes.

Support vectors are the data points that lie closest to the decision surface, they are the most difficult to classify and have direct bearing on the optimum location of the decision surface [31].

By optimal decision (figure 5.6) hyperplane we mean that the algorithm choose the plane with maximum margin around this plane.

Until now, this means that SVM is only a binary linear classifier but actually it can do polynomial classification using "kernels" and we can generalize binary classification to multiple-class classification using one versus all or one versus one methods.

5.2.1 Brief preface

First, we will **shortly** explain the theory behind kernels and multi-class classification, and then we will introduce our SVM model.

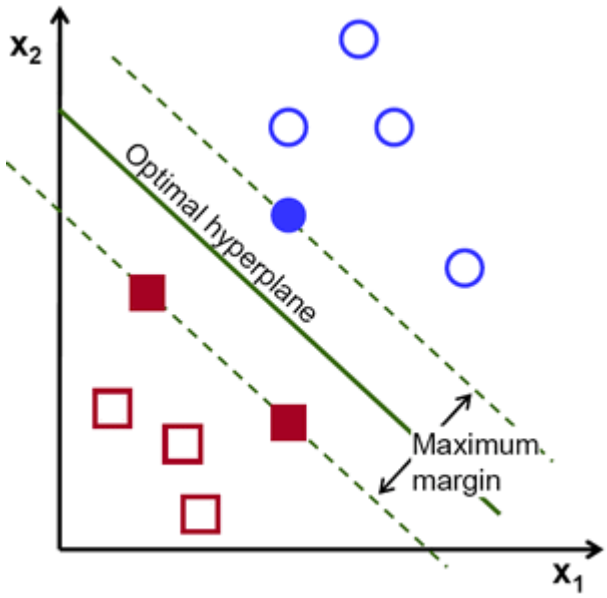


Figure 5.6: Optimal hyperplane from [1]

5.2.1.1 Kernels

Kernel is simply a function $K(p1, p2)$ that measures similarity between two points (samples) in the features space.

Assuming that we have two classes (positive and negative) and a set of points P which represents points from 2-D features space where each point $p = (x1, x2)$, where $x1 \in X1$ and $x2 \in X2$, and we want to classify a new point p_{test}, then SVM algorithm classifies p_{test} as positive if

$$\sum_{p \in P} (K(p_{test}, p) * \theta_p) \geq 0$$

whereθ_p is a scalar weight assigned to point $p \in P$

Actually, points of interest (support vectors) are chosen from P by assigning non-zero values to their weight.

Finally there are several types of kernels, discussing them is not covered by our study.

Software we are using for that is `libsvm`, for more about kernels and minimization objective read [32].

We have used three types of kernels: linear kernel (no kernel), polynomial kernel and Gaussian kernel.

5.2.1.2 Multi-class classification

Since SVM is two-class classifier and our project requires multi-class classifier, we need to generalize two-class classifier to multi-class classifier; there are two well-known methods [33]:

- **One versus all** (also known as one versus rest)

  Done by building several binary classifiers by number of classes we have, and then train each one by the samples of this class as positives and all other samples as negatives. Finally, classify the test sample by all of these classifiers and pick the class with positive classification, in case of several positive classifications pick one with the highest margin.

- **One versus one**

  Done by building $\frac{N(N-1)}{2}$ classifiers for each pair of classes then to after training process to classify a sample it is classified by all of these classifiers and the class with highest number of votes is selected.

In our tests and application, we are using one versus one classification only .

Note: according to `libsvm` library each SVM parameter is fixed for every binary classifier during training and cross validation test, in order to avoid over-fitting.

5.2.2 Model selection

Model selection is done by scaling all features discussed in chapter 4, then we try to find a combination of SVM parameters that provide

best performance among all combinations of parameters values in specific ranges.

Testing is done using n-folds validation with 6 validation folds.

n-folds validation: splits the data randomly into n pieces and for n times it trains the classifier with all pieces except a specific one and test performance against this piece, then the average accuracy is taken among all of the n tests.

5.2.2.1 Selection of SVM parameters

Linear kernel actually do not have specific parameters so we will only try one SVM related parameter, which is c regularization parameter

$$log_2(c) \in [-1, 0, 1, 2, 3]$$

This comparison is shown in table 5.1.

$log_2(c)$	-1	0	1	2	3
Accuracy %	99.8556	99.9098	99.9549	99.9459	99.9549

Table 5.1: Linear kernel SVM accuracy

Polynomial kernel has the parameters $gamma$, $coeff0$, $degree$ where it is given by the formula

$$K(u, v) = (gamma \times u^T \times v + coeff0)^{degree}$$

Values we are trying:

$$log_2(c) \in [-1, 0, 1, 2, 3]$$
$$log_2(gamma) \in [-4, -3, -2, -1, 0, 1]$$
$$coeff0 \in [0, 1, 2, 3, 4, 5]$$
$$degree \in [2, 3, 4, 5]$$

Best value in polynomial kernel was

$degree$	2
$log_2(c)$	-1
$log_2(gamma)$	-4
$coeff0$	4
Accuracy%	96.6522

Table 5.2: Polynomial kernel SVM accuracy

Gaussian kernel (radial basis function) has the parameter $gamma$, it is given by the formula

$$K(u, v) = e^{-gamma \times \|u-v\|^2}$$

Values we are trying

$$log_2(gamma) \in [-4, -3, -2, -1, 0, 1]$$
$$log_2(c) \in [-1, 0, 1, 2, 3]$$

Best value for Gaussian kernel was for

$log_2(gamma)$	-1
$log_2(c)$	1
Accuracy%	99.9820

Table 5.3: Polynomial kernel SVM accuracy

So from above we can see that Gaussian kernel provides best performance among all of our tests.

6 Evaluations

In this chapter we describe the training and testing processes, we also describe the used data set in details. Besides, we show the results of the proposed models (i.e., their accuracy) and compare them to a number of other related works.

6.1 Dataset

First, let's view the input specification: for this implementation we've chosen 8 different signs of the ASL shown in figure 6.1. Note that the only exception is letter 'L' which is rotated 90 degrees.

For training, a new data set is collected using for 8 different signs of the ASL from two different signers with different hand shapes, each subject performs about 65 different poses for the same sign, so the result is about 1040. Each case consists of a binary image of the signer's hand, no other information was included because the models expect the input to be isolated from its environment.

Another dataset is collected for testing using same signs from 3 other different signers as specified before.

6.2 Features sets

As explained in chapter 4, we have 4 different sets of features. We will label these sets as follows

- S1: Geometric shape descriptors
- S2: Invariant Moments

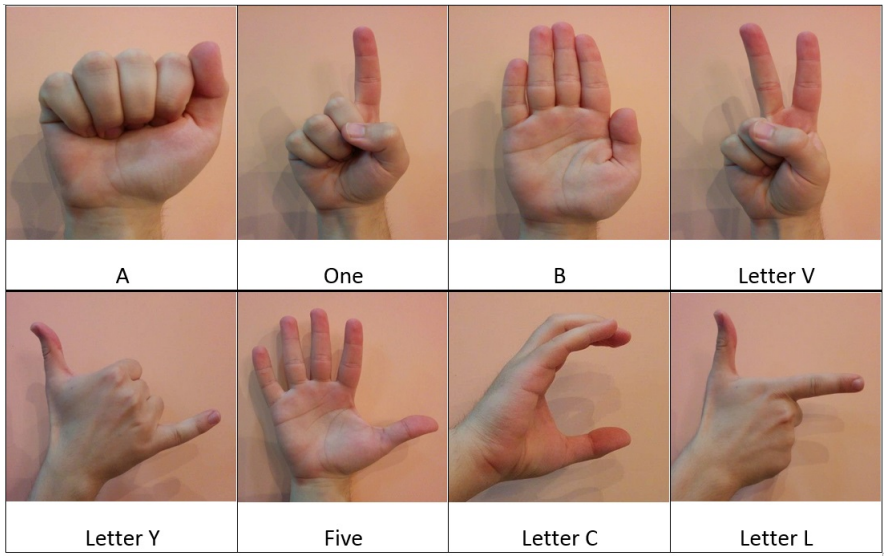


Figure 6.1: Chosen signs and their labels

- S3: Principal Components Analysis
- S4: Active Fingers Properties

Combinations of these features sets will be used in the next step in section 6.3.

6.3 Accuracy evaluation

In order to evaluate the accuracy, we conducted several experiments on models (ANN and SVM) for each of which, different combinations of features sets are tried, and the accuracy of each experiment is shown in figure 6.2.

Comments on ANN accuracy A confusion matrix, also known as an error matrix is shown in 6.3.

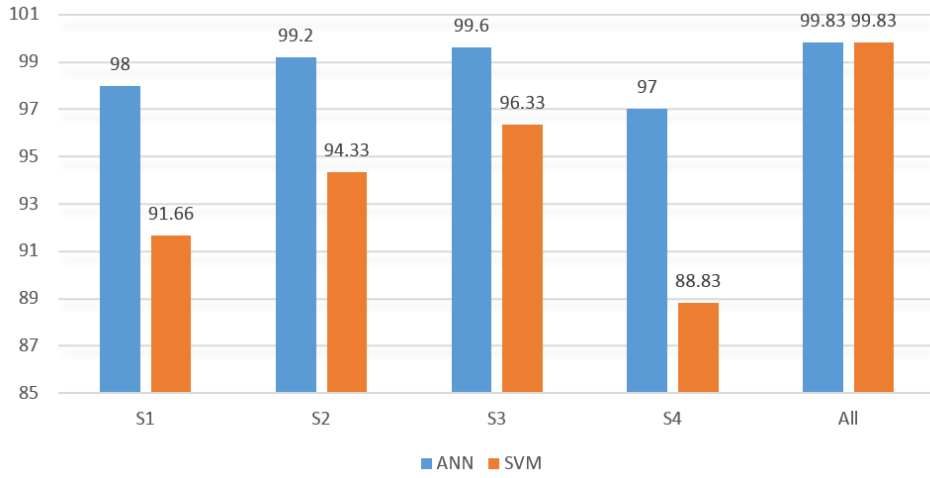


Figure 6.2: Different features sets comparison

Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class (or vice-versa).

The diagonal cells in each table show the number of cases that were correctly classified, and the off-diagonal cells show the misclassified cases. The blue cell in the bottom right shows the total percent of correctly classified cases (in green) and the total percent of misclassified cases (in red).

The results for testing dataset show very good recognition results.

6.4 Comparison with other works

Table 6.1 gives a brief comparative study between our work with the other related works. It is obvious that different accuracies have been reached using different approaches. In fact, each one of them has a limit preventing it from being used on a large scale. Our approach succeed to increase the number of gestures the and preserve a high accuracy in the same system and is shown in table 6.2 [18].

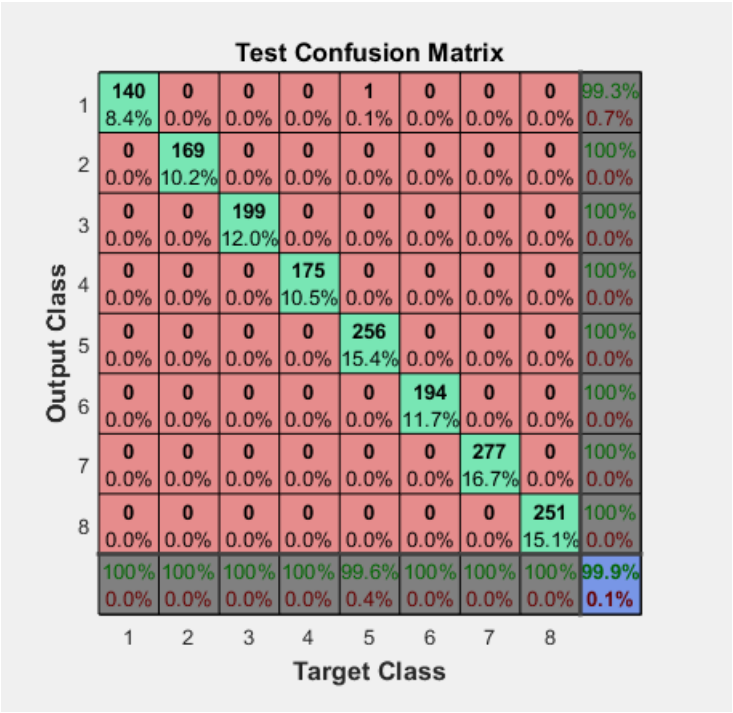


Figure 6.3: Confusion matrix of ANN using all features

6.5 Performance evaluation

In this section we view execution times at each phase in our system.

The image capturing process is done using a regular camera (Nexus 5's rear camera which is an 8 MP Sony Exmor IMX179) to acquire frames, and each frame will be processed separately. The captured images are of size 1024×768.

The image processing, feature extraction and classification processes are conducted on: Intel core i5-3210 CPU 2.50GHz $\times$ 2, running Linux Mint 17.3 Rosa platform using MATLAB R2015a.

First we review running times for the image processing and hand localization phase (table 6.3).

Name of the technique used	Success rate	Difficulties faced
Back propagation Neural Network	80.28%	Neural network requires input feature vectors to have integer values for learning
Hit-Miss Operation, HMM	97.83%	Poor skin color detection
Perceptual Color Space	100%	dealt with only 5 gestures
Contour based	91%	Use of hand gloves
ANN based	94%	Use of hand gloves with 13 sensors
Kinematic Chain Theory based	100%	3 simple gestures Data gloves used

Table 6.1: Comparison with other works

Our Work	Success Rate
Back propagation Neural Network	99.83%
SVM	99.83%

Table 6.2: Comparison with other works

Sub-phase	Average running time (sec)
Image acquisition	0.1093
Face detection	0.8470
Skin filtering (thresholding: hysteresis)	0.1286
Skin filtering (thresholding: Otsu)	0.1274

Table 6.3: Image processing phase running times

Results

- The high image acquisition time shows the difficulty of dealing with weak equipment.
- Since the two thresholding techniques have similar running times, we'd better use the most accurate one which is hysteresis thresholding (see section 3.2.3.2).
- If we wanted to run a real-time application, the face might not be detected for each frame; it has to be detected either at the start of the application (the processing time of a frame would be 0.2377 seconds) or once for every number of (let's say 10) frames (therefore, face detection would take an average of 0.0847 seconds and the processing of a frame would become 0.3226 seconds instead of 1.0849 seconds).

Second, we show running times for extracting each set of the proposed features with the average accuracy they could achieve for both classification models (section 6.3).

Feature set	Average running time (sec)	Accuracy% in ANN	Accuracy% in SVM
Geometric shape descriptors	1.7168e-05	98.00	91.66
Invariant moments	2.4362e-05	99.2	94.33
PCA	32.942e-05	99.60	96.33
Active fingers attributes	392.22e-05	97.00	88.30

Table 6.4: Feature sets extraction running times

Results From table 6.4 we deduce

- Since the running time of the active fingers features is extremely high, we can't use this type of features in a real-time application

At last, the running times of prediction for both classification models.

Feature set	ANN average running time (sec)	SVM average running time (sec)
Geometric shape descriptors	0.0117	3.0621e-04
Invariant moments	0.0115	2.5986e-04
PCA	0.0132	1.9591e-04
Active fingers attributes	0.0113	3.1162e-04
All features	0.0119	2.0909e-04

Table 6.5: Feature sets extraction running times

Results

- ANN running times on each set of features barely differ
- For ANN, the choice of using PCA features is the best since the total running time would be 0.3361 seconds and the accuracy would be 99.60%
- SVM prediction is much faster ANN's

6.6 Interactive application

As an addition to our project, we present a simple application which summarizes the work.

This application is supposed to make predictions on images snapshots acquired from a streaming video, comprising face detection, skin detection, features extraction classification, and that is done for each snapshot separately each 0.3 second (since the run time of the best combination is less than 0.3 second).

As seen in Figure 6.4, a prediction is made for each hand, labelled with its word/letter from the ASL sub-dictionary and presented as the final result.

And because some of the features are noise-resistant, such as invariant moments and PCA, this will allow our application to correctly classify some noisy binary input.

It is also clear that the skin detection and hand localization proposed method is robust and works fine even in a challenging environment.

Figure 6.4: On-line Predictions

7 Results, challenges and future work

7.1 Results

1. Providing a complete system for recognizing 8 hand signs which enables us of building a dictionary of ($8 \times 8 = 64$) unique commands, words, etc..
2. Reaching an accurate and efficient system through discussing and comparing several approaches for each of the main phases.
3. Classifying signs (out of 8 classes) with an accuracy of 99.83% for both ANN and SVM.
4. Getting positive results in comparing our work to other works.
5. Achieving high speed and low memory consuming models which takes 0.3361 seconds to process a sign with good accuracy (99.60%)
6. Proposing method for skin detection (skin distance, applying hysteresis thresholding for skin-color filtering).
7. Visualizing the distribution of samples using PCA.

7.2 Challenges

1. Low hardware capabilities, which implies:

a) Incapability to acquire images in depth, due to depth sensors' high costs which creates obstacles in hand localization process

b) Acquiring low resolution images which means results would be significantly affected.

c) High image acquisition time which represents the bottleneck in the process running time.

d) Difficulties in building a real-time recognition system.

e) Consuming much time while training models, extracting PCA features.

2. Insufficient time period to immerse enough through the theoretical concepts and details.

3. Difficulty in accessing scientific works published previously.

7.3 Real-life applications

Gesture-based technology is becoming the next generation in industry even though there are still some limitations. It is spreading all around the world and implemented for usage in different domains.

Some examples applications for this technology are

1. Control electronic devices in hotels or houses, where gesture recognition system uses dedicated pre-assigned signs to switch lights on/off, control television, etc.

2. Teach hearing impaired how to perform their own sign languages gestures, by correcting hand-shapes to be as much close to standard hand-shapes as possible.

3. Entertainment, gestures recognition technologies gives new gaming experience, interactive platforms ,virtual reality and more.

7.4 Future Work

In the future we have several steps we may take in order to improve our project, some of them are:

- Extending number of signs we are classifying to include all of the ASL (or maybe the Arabic Sign Language).
- Making our application more interactive by adding video processing instead of static still images processing.
- Enhance skin detection and hand localization process and we may also try machine learning concepts in hand localization too.
- Studying and testing all possible combinations of features we have extracted so far in order to get smaller set of features that has a better balancing between computing time and prediction accuracy.

Acknowledgements

From all of us, we'd like to thank:

- Our parents who were always on our side
- Dr. Mohsen Hussain for teaching us the academic writing and helping us in making the right decision every time
- Our friends who helped us capturing and collecting the data sets
- Dr. Shaheen, Dr. Atassi and Eng. Nasser for directing us and making this work look good

Bibliography

[1] "Introduction to Support Vector Machines (OpenCV Tutorials) http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html," 2016.

[2] N. SARAWATE, M. C. LEU, and C. ÖZ, "A real-time American Sign Language word recognition system based on neural networks and a probabilistic model," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 23, pp. 2107–2123, 2015.

[3] T. Johnson and A. Schembri, *Australian Sign Language (Auslan): An introduction to sign language linguistics.* Cambridge University Press, 2007.

[4] "National Center on Birth Defects and Developmental Disabilities, Atlanta, GA, USA http://www.cdc.gov/ncbddd/index.html," 2008.

[5] P. W. Vamplew, "Recognition of Sign Language Using Neural Networks by," no. May, pp. 27–33, 1996.

[6] Z. Ren, J. Yuan, J. Meng, and Z. Zhang, "Robust part-based hand gesture recognition using kinect sensor," *IEEE Transactions on Multimedia*, vol. 15, no. 5, pp. 1110–1120, 2013.

[7] H.-D. Yang, "Sign Language Recognition with the Kinect Sensor Based on Conditional Random Fields," *Sensors*, vol. 15, no. 1, pp. 135–147, 2014.

[8] S. Lang, M. Block, and R. Rojas, "Sign Language Recognition Using Kinect," *Artificial Intelligence and Soft Computing*, vol. 7267, no. September, pp. 394–402, 2012.

[9] X. Chai, G. Li, Y. Lin, Z. Xu, Y. Tang, and X. Chen, "Sign Language Recognition and Translation with Kinect," *The 10th IEEE International Conference on Automatic Face and Gesture Recognition*, pp. 22–26, 2013.

[10] A. BEN JMAA, W. MAHDI, Y. BEN JEMAA, and A. BEN HMADOU, "A new approach for digit recognition based on hand gesture analysis," *(IJCSIS) International Journal of Computer Science and Information Security, Vol. 2, No. 1, 2009*, 2009.

[11] J. Singha and K. Das, "Indian Sign Language Recognition Using Eigen Value Weighted Euclidean Distance Based Classification Technique," *arXiv preprint arXiv:1303.0634*, vol. 4, no. 2, pp. 188–195, 2013.

[12] A. Mittal, A. Zisserman, and P. Torr, "Hand detection using multiple proposals," *22nd British Machine Vision Conference*, pp. 75.1–75.11, 2011.

[13] R. Khan, Z. Khan, M. Aamir, and S. Q. Sattar, "Static Filtered Skin Detection," *IJCSI International Journal of Computer Science Issues*, vol. 9, no. 2, pp. 257–261, 2012.

[14] D. Chai and K. N. Ngan, "Face segmentation using skin-color map in videophone applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 4, pp. 551–564, 1999.

[15] J. A. M. Basilio, G. A. Torres, G. S. Pérez, L. K. T. Medina, and H. M. P. Meana, "Explicit Image Detection using YCbCr Space Color Model as Skin Detection," *Applications of Mathematics and Computer Engineering (CEMATH)*, pp. 123–128, 2011.

[16] P. Mekala, Y. Gao, J. Fan, and A. Davari, "Real-time sign language recognition based on neural network architecture," *43rd IEEE Southeastern Symposium on System Theory (SSST)*, pp. 197–201, 2011.

[17] L. Bretzner, I. Laptev, and T. Lindeberg, "Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering," *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition*, pp. 423–428, 2002.

[18] S. Nagarajan and T. S. Subashini, "Static Hand Gesture Recognition for Sign Language Alphabets using Edge Oriented Histogram and Multi Class SVM," *International Journal of Computer Applications*, vol. 82, no. November, pp. 975–8887, 2013.

[19] B. C. Bedregal, A. C. R. Costa, and G. P. Dimuro, "Fuzzy Rule-Based Hand Gesture Recognition," pp. 1–10, 2004.

[20] K. Sobottka and I. Pitas, "A novel method for automatic face segmentation, facial feature extraction and tracking," *Signal Processing: Image Communication*, vol. 12, no. 3, pp. 263–281, 1998.

[21] M. B. Hmid and Y. B. Jemaa, "Fuzzy Classification, Image Segmentation and Shape Analysis for Human Face Detection," *Signal Processing, 2006 8th International Conference on*, vol. 4, no. FEBRUARY 2006, 2006.

[22] N. Otsu, "A Threshold Selection Method from Gray Level," {*IEEE*} *Transaction on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[23] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. I—-511—-I—-518, 2001.

[24] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[25] G. Liu and R. M. Haralick, "LiuHaralick00.pdf," *Proceedings 15th International Conference on Pattern Recognition*, vol. 3, no. 3-7, pp. 1052–1055, 2000.

[26] M. K. Hu, "Visual Pattern Recognition by Moment Invariants," *IRE Transactions on Information Theory*, vol. 8, no. 2, pp. 179–187, 1962.

[27] L. Keyes and A. Winstanley, "Using moment invariants for classifying shapes on large-scale maps," *Computers, Environment and Urban Systems*, vol. 25, no. 1, pp. 119–130, 2001.

[28] D. Austin, "We Recommend a Singular Value Decomposition http://www.ams.org/samplings/feature-column/fcarc-svd," 2015.

[29] G. Knese, "Operator theory and the singular value decomposition of its fundamental tools," *Snapshots of modern mathematics from Oberwolfach*, vol. 2014, no. 9, pp. 1–7, 2014.

[30] "Linear versus nonlinear classifiers http://nlp.stanford.edu/IR-book/html/htmledition/linear-versus-nonlinear-classifiers-1.html," 2009.

[31] R. Berwick, "An Idiot ' s guide to Support vector machines (SVMs) SVMs : A New Generation of Learning Algorithms Key Ideas," pp. 1–28, 2008.

[32] C. W. Hsu, C. C. Chang, and C. J. Lin, "A Practical Guide to Support Vector Classification," *BJU international*, vol. 101, no. 1, pp. 1396–400, 2008.

[33] J. Milgram, M. Cheriet, and R. Sabourin, ""One Against One" or "One Against All": Which One is Better for Handwriting Recognition with SVMs?," *Tenth International Workshop on Frontiers in Handwriting Recognition*, pp. 4–6, 2006.