CHAPTER 5

FEATURE EXTRACTION

As mentioned earlier feature extraction is one of the subtasks of recognition process. Classification and post processing being the other two. The glyphs extracted from each of the zones are to be recognized independently. The first step in this process is feature extraction. In this chapter we present mathematical concepts used for feature extraction process during this study and also describe its exact application to the problem of Gujarati character recognitoin. There are two major types of the features used in the recognition process: (1) Structural (2) Mathematical / Statistical.

In this chapter we present a detailed study of the effectiveness of some of the features of each type. We have carried out experiments with mathematical features like Discrete Cosine Transform, Wavelets, Fringe Maps and structural features such as aspect ratio, zone information etc..

5.1 Introduction

Feature extraction is a process in which we transform an image from space of all images to a new space where, it is hoped, the pattern recognition problem will be easier to solve [7]. Purpose of this transformation is many fold : to reduce the variability among the images of the same class, to make data less sensitive towards noise, reducing the dimensionality etc.. The new space of transformed images are called *feature space* and elements in this space (set of features for image) are called *feature vectors*. Mathematically, let

Feature space
Feature vector

$$\mathcal{I} = \text{Space of all images of dimension } m \times n \text{ and}$$

$\mathbf{F}(\mathbf{i})$ = Feature extractor(vector) of dimension p for image $\mathbf{i} \in \mathcal{I}$, then,

the mapping $\mathbf{F}$ transforms $m \cdot n$ dimension image vector $\mathbf{i}$ in image space into a p-dimension feature vector in feature space.

For example, if we take height and width of a connected component as two features then the resultant feature space will be a 2-dimensional feature space. Therefore, the connected component, which can be considered as a vector / point in $m \cdot n$ dimensional image space, is mapped on to a point in two dimensional feature space as shown in Figure 5.1.

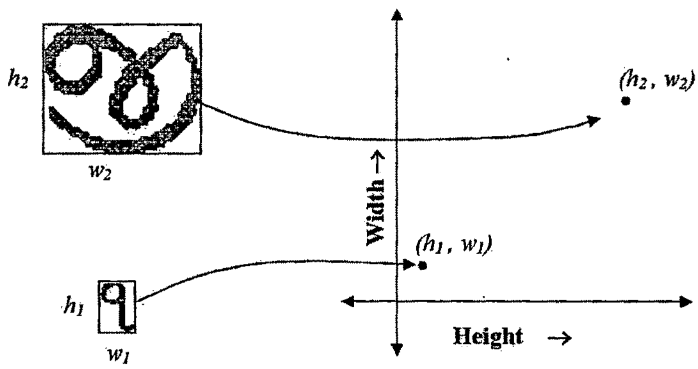

Figure 5.1: Image Represented in Feature Space

Several Mathematical and statistical techniques have been used by researchers for finding the "best" features. Fourier Descriptors, Discrete Cosine Transform, Principal Component Analysis, Moments, Projection Profile, Edge profile, Gabor filters, Pixel Density, Aspect Ratio are a few examples of the mathematical techniques used for feature extraction.

5.2 Mathematical Preliminaries

In this section we briefly describe the mathematical techniques applied to the feature extraction process.

5.2.1 Distance Measure[19]

For pixels $p(x, y)$,$q(s, t)$ and $z(v, w)$ of a digital image, D is a *distance function* or *metric* if Distance / Metric

a. $D(p, q) \geq 0 \quad (D(p, q) = 0 \text{ iff } p = q)$,

b. $D(p, q) = D(q, p)$, and

c. $D(p,q) \leq D(p,z) + D(z,q)$.

A special type of distance called D_4 *distance* between p and q is defined as

D_4 Distance

$$D_4(p,q) = |x - s| + |y - t|. \tag{5.1}$$

This is also known as *city-block distance.* In this case, the pixels having D_4 disance from (x, y) less than or equal to some value r form a diamond centered at (x, y). For example, the pixels with D_4 distance ≤ 2 from (x, y) (center point) form the following contours of constant distance :

City-block Distance

$$\begin{array}{ccccc} & & 2 & & \\ & 2 & 1 & 2 & \\ 2 & 1 & 0 & 1 & 2 \\ & 2 & 1 & 2 & \\ & & 2 & & \end{array}$$

5.2.2 Discrete Cosine Transform

Discrete Cosine Transform(DCT), $D(u, v)$ of an image, say $I(x, y)$ of size $n \times n$ is given by equation (5.2). It is a very important tool in image compression [19] where reduction of the memory requirement is the main goal. In the case of recognition also we try to reduce the number of elements in feature vectors as it directly implies faster calculation at the time of classification with less storage required. Considering this fact we have selected this as feature extractor.

$$\begin{aligned} D(u,v) &= C(u)C(v)\sum_{y=0}^{n-1}\sum_{x=0}^{n-1} I(x,y)cos\left[\frac{(2x+1)u\pi}{2n}\right] cos\left[\frac{(2y+1)v\pi}{2n}\right] \\ &\text{where } C(u) = \sqrt{\frac{1}{n}} \text{ for } u = 0 \text{ and } C(u) = \sqrt{\frac{2}{n}} \text{ otherwise,} \\ &u, v = 0, 1, 2, \ldots, n-1. \end{aligned} \tag{5.2}$$

5.3 Image Scaling

The type of features we have used for this work and not scale invariant. Hence, we have to scale all the connected components to a common size, exapt one, before we perform any feature extraction from them. We have used the following scaling algorithm for this purpose.

Algorithm 5.1 Scaling Connected Component

Input: Array of 0s and 1s, say $IM[i][j]$, with m rows and n columns corresponding to a binarized connected component,

Output: Scaled array, say $scaledIM[i][j]$, with $newM$ rows and $newN$ columns

Process: Scaling

$hr \leftarrow m/newM$
$wr \leftarrow n/newN$
for $i = 0$ to $newM$ **do**
  $ti \leftarrow round(i * hr)$;
  for $j = 0$ to tow **do**
    $tj \leftarrow round(j * wr)$;
    $scaledIM[i][j] \leftarrow IM[ti][tj]$
  end for
end for

5.4 Features

The attempts for Gujarati character recognition so far was using Hu-moments [1] and wavelets coefficients [38], [46], [47]. As a part of this research we have investigated effectiveness of three types of feature extractors viz. Fringe Map, Discrete Cosine Transform Coefficients and aspect ratio of a connected component as features. In addition to these three we have also used zone information as features.

5.4.1 Fringe Map

Template based recognition is a known method and Fringe map has been used as template earlier for Telugu script recognition[29]. *Fringe map* of a binary image is generated by replacing each pixel by its distance from nearest black pixel. The distance here is the D_4-Distance or the citi-block distance discussed eariler in this chapter. Figure 5.2 shows an example of fringe map for character /tha/.

Fringe Map

Algorithm 5.2 Compute Fringe Map

Input: Array of 0s and 1s, say $glyph[i][j]$, with m rows and n columns corresponding to a binarized connected component.

Output: Fring map, say $fring[i][j]$ of the same size as IM.

Process:

for $i = 0$ to m **do**
  $DistFromBlack \leftarrow -1$

```
    for j = 0 to n do
        if glyph[i][j] ≠ 0 and cDistFromBlack = −1 then
            fringe[i][j] ← 9999
        end if
        if glyph[i][j] = 0 then
            fringe[i][j] ← 0
            cDistFromBlack ← 0
        end if
        if glyph[i][j] ≠ 0 and cDistFromBlack ≠ −1 then
            cDistFromBlack ← cDistFromBlack + 1
            fringe[i][j] ← cDistFromBlack
        end if
    end for
    cDistFromBlack ← −1
    for j = n to 0 do
        if glyph[i][j] = 0 then
            cDistFromBlack ← 0
        end if
        if glyph[i][j] ≠ 0 and cDistFromBlack ≠ −1 then
            cDistFromBlack ← cDistFromBlack + 1
            if fringe[i][j] > cDistFromBlack then
                fringe[i][j] ← cDistFromBlack
            end if
        end if
    end for
end for
for j = 0 to n do
    cDistFromBlack ← −1
    for i = 0 to m do
        if glyph[i][j] = 0 then
            cDistFromBlack ← 0
        end if
        if glyph[i][j] ≠ 0 and cDistFromBlack ≠ −1 then
            cDistFromBlack ← cDistFromBlack + 1
            if fringe[i][j] > cDistFromBlack then
                fringe[i][j] ← cDistFromBlack
            end if
        end if
    end for
```

```
    cDistFromBlack ← −1
    for i = m to 0 do
        if glyph[i][j] = 0 then
            cDistFromBlack ← 0
        end if
        if glyph[i][j] ≠ 0 and cDistFromBlack ≠ −1 then
            cDistFromBlack ← cDistFromBlack + 1
            if fringe[i][j] > cDistFromBlack then
                fringe[i][j] ← cDistFromBlack
            end if
        end if
    end for
end for
```

```
2 1 1 0 0 0 0 0 1 1 1 1 2 3 4 5 5 4 3 2 1 1 1 2 2 3 4 5 6 7 8 9
1 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 4 3 2 1 0 0 0 1 1 2 3 4 5 6 7 8
1 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 3 2 1 0 0 0 0 0 1 2 3 4 5 6 7
0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 2 3 3 2 1 0 0 0 0 0 1 2 3 4 5 6 7
0 0 0 0 0 1 2 2 2 2 2 1 0 0 0 1 2 3 3 2 1 0 0 0 0 1 2 3 4 5 6 7
0 0 0 0 0 1 2 3 3 3 2 1 0 0 0 1 2 3 2 1 0 0 0 0 0 1 2 3 4 5 6 7
0 0 0 0 0 1 2 3 3 2 1 0 0 0 0 0 1 2 3 2 1 0 0 0 0 1 2 3 4 5 6 7
1 1 0 0 0 0 1 2 2 2 1 0 0 0 0 0 1 2 3 2 1 0 0 0 0 1 2 3 4 5 6 7
2 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 2 2 1 0 0 0 0 0 1 2 3 4 5 6 7
3 2 1 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 3 2 1 0 0 0 0 1 2 3 4 5 6 7
4 3 2 1 0 0 0 0 0 0 0 0 0 0 0 1 2 3 3 2 1 0 0 0 0 1 2 3 4 5 6 7
4 3 2 2 1 1 0 0 0 0 0 0 0 0 1 2 3 4 3 2 1 0 0 0 0 1 2 3 4 5 6 7
3 2 1 1 1 0 0 0 0 0 0 0 0 1 2 3 4 4 3 2 1 0 0 0 0 1 2 3 4 5 6 7
2 1 0 0 0 0 0 0 0 0 0 0 1 2 3 4 4 3 2 1 0 0 0 0 0 1 2 3 4 5 6 7
1 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 5 4 3 2 1 0 0 0 0 1 2 3 4 5 6 7
1 0 0 0 0 0 0 0 0 1 1 2 3 4 5 6 5 4 3 2 1 0 0 0 0 1 2 3 4 5 6 7
1 0 0 0 0 0 0 0 1 2 2 3 4 5 5 5 4 3 2 1 0 0 0 0 0 1 2 3 4 5 6 7
1 0 0 0 0 0 1 1 2 3 3 4 4 4 4 4 3 2 1 0 0 0 0 0 0 1 2 3 4 5 6 7
2 1 0 0 0 1 2 2 3 3 3 4 3 3 3 3 2 1 0 0 0 0 0 0 0 1 2 3 4 5 6 7
3 2 1 0 0 0 1 2 2 2 2 3 2 2 2 2 2 1 0 0 0 0 0 0 0 1 2 3 4 5 6 7
4 3 2 1 0 0 0 1 1 1 1 2 1 1 1 1 1 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7
5 4 3 2 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7
6 5 4 3 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 4 5 6
7 6 5 4 3 2 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 2 3 3 3 4 5
8 7 6 5 4 3 2 2 2 1 1 1 1 1 1 1 1 1 1 2 1 0 0 0 0 1 2 3 2 2 3 4
9 8 7 6 5 4 3 3 3 2 2 2 2 2 2 2 2 2 2 2 1 0 0 0 0 1 2 2 1 1 2 3
0 9 8 7 6 5 4 4 4 3 3 3 3 3 3 3 3 3 3 2 1 0 0 0 0 1 2 1 0 0 1 2
11 10 9 8 7 6 5 5 5 4 4 4 4 4 4 4 4 4 3 2 1 0 0 0 0 1 2 1 0 0 0 1
12 11 10 9 8 7 6 6 6 5 5 5 5 5 5 5 5 4 3 2 1 0 0 0 0 1 1 0 0 0 0 0
13 12 11 10 9 8 7 7 7 6 6 6 6 6 6 6 5 4 3 2 1 0 0 0 0 0 1 0 0 0 0 0
14 13 12 11 10 9 8 8 8 7 7 7 7 7 7 7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 0 1
15 14 13 12 11 10 9 9 9 8 8 8 8 8 8 8 7 6 5 4 3 2 1 1 0 0 0 0 0 1 1
```

Figure 5.2: Fringe Map for Alphabet /tha/

As can be seen from the method of template generation, this feature is not size invariant. Hence, we scale all our recognizable components to 32×32. All these 1024 elements constitute the feature vector for this alphabet /tha/.

5.4.2 Discrete Cosine Transform (DCT) Coefficients

For a connected component DCT coefficients are computed using eq. 5.2. Obviously we get as many coefficients as the number of elements in the input matrix. It is clear from the definition that the DCT coeffiecients are real numbers. In [43] DCT is reported to be one of the best transformations in terms of energy compaction. DCT coefficients are used in JPEG image compression where the main goal is to reduce the storage requirements while preserving important features of the image. The same property is useful in character recognition for reducing the size of the feature vector and thereby reducing the storage requirement for features of all possible connected components. This also helps in optimizing the searching for a class to which the connected component belongs.

It is clear from the theory of DCT that it is also not scale invariant and the process of calculating DCT can be optimized in terms of time and space if the dimension of input matrix is some power of 2. Hence, here too we scale the images to a common size before subjecting them to DCT. Although the total energy remains the same, the energy distribution changes with most of the energy being compacted to the low frequecy coefficients. That is the original image can be approximated by only few low frequecency coefficients. In case of 2-D DCT of an image, such coefficients lie on the top left corner of the matrix. Therefore we select top left coefficients in the zig-zag order as shown in Fig.5.3 to constitute feature vector.

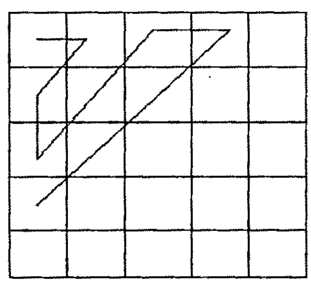

Figure 5.3: Zig-Zag Direction for DCT

5.4.3 Aspect Ratio

Aspect Ratio

The *Aspect Ratio* of an image is the ratio of the width of the image to its height, expressed as two numbers separated by a colon. That is, for an $x : y$ aspect ratio, no matter how big or small the image is, if the width is divided into x units of equal length and the height is measured using this same length unit, the height will be measured to be y units.

One of the most frequently occurring glyphs in Gujarati is a bar (vertical line) sign corresponding to vowel modifier for /AA/. The aspect ratio of this glyph is very much different from other glyphs and also the scaled version of this glyph will cover entire 32×32 matrix with black pixel and it can generate match for any random glyph when compared using any of the conventional features. Hence, for each of the glyph we calculate aspect ratio and we use this information to classify glyph corresponding to vowel modifier /AA/.

5.4.4 Zone Information

As mentioned in section 2.5 zone identification is done before extracting unit of recognition. Hence, we use the relative position of the glyph viz. its zone as one of the features.

This reduces the complexity of classifier design as classification of upper and lower zone glyphs will involve less than 10 classes. Also, for middle zone glyphs search will be limited to only features of middle zone glyphs. In other words, we will build three classifiers, one for classifying glyphs from each of the three zones.

5.4.5 Conclusions

Feature extraction is an important task. Features should be selected such that the glyphs of the same class have similar features and the features for the glyphs of the different classes should be much different. Use of DCT and fringe map as features has been discussed in this chapter. Further, some of other structural features like Aspect ratio and Zone information are also shown. It is also discussed that use of zone information reduces the classifier complexity.