Keerthi Raj Nagaraja (nagaraj1@purdue.edu)

11-05-2014

1 Character recognition

In this assignment, we implement a simple character recognition method using the concepts of Component labeling, harris corner detection and by extracting feature descriptor using the detected corners. We then try to recognize the characters from the test images and report the accuracies.

1.1 Segmentation and Component labeling

The otsu's algorithm is used for segmenting out the given image such that we get a binary image with characters as having pixel value of 255 and background is made 0. The otsu's algorithm has been explained in the previous assignment.

Followed by segmentation, we apply a component labeling algorithm to the segmented image. The goal here is to extract out each character as a separate image so that corners can be detected later. To extract out the characters, we assign different labels to each of them using connected component labeling technique since all the pixels belonging to a character will be connected as they have same pixel values and are separated by other characters by the black background.

I have used a 4-connectivity based 2-pass component labeling algorithm. In the first pass we assign labels to foreground pixels based on the neighborhood connectivity. In the second pass we resolve the equivalencies by re-labeling the pixels by the minimum-valued label in a given equivalence class. The algorithm has following steps:

First Pass:

For each pixel (x, y) in the segmented binary image:

(i) Check if the pixel's value is 255 or 0. If it is 0, go to next pixel

(ii) If the pixel value is 255, then check if the left and top neighboring pixels $(x-1, y)$ and $(x, y-1)$ have same non-zero value (same component label). If true, then assign the same component label to current pixel (x, y).

(iii) If both left and top pixels have different non-zero values (different component labels), then assign one of the component label to the current pixel and record that both those labels are equivalent.

(iv) If one among the left and top pixels is zero-valued and the other has a non-zero label, then assign the label to the current pixel.

(v) If both the left and top pixels are zero-valued, then create a new label (increment the label count) and assign the new label to the current pixel. Repeat from step 1 for next pixel.

Second Pass:

For each pixel (x, y) in the labeled image:

(i) Get the label of the pixel. If its zero, proceed to next pixel.

(ii) If its non-zero, get all the labels which are equivalent to this label in the equivalency data-structure. Re-assign the label of the current pixel to the minimum value among the equivalent labels. Repeat from step 1 for next pixel.

Once the component labeling is complete, pixels belonging to each character will have same label. Now, we crop out pixels belonging to same label as a new image with appropriate padding. The result is a set of images each containing a individual characters.

1.2 Harris corner detection

The corners in each character are detected using a harris corner detector algorithm. This method has been explained in assignment 4. The same program is used with following minor modifications:

(1) We pre-define the number of corners (N) we want to extract.

(2) Instead of choosing a threshold on the corner strength and extracting remaining corners, we extract top N corners with maximum corner strengths.

This gives us N corners for each character which ideally defines the shape of the character. N has been chosen as 13 since 13 is the maximum number of true corners (manually counted for each character and found 13 for 'M' and 'W'). This captures the complete shape of the character. If there are lesser than 13 true corners in any character (true for many), the harris corner just picks up the next best corner which will be close to one of the true corners so that the shape of the character is still captured by the corners.

1.3 Feature Extraction

In this stage, given N corners in the character image, we find the feature descriptor which represents the shape of the character based on the corner positons. Since the descriptor has to be scale invariant and rotation invariant, we cannot simply use positions of the corners as the features. We want a metric which captures the relative position of the corners.

Simple technique which was proposed in the assignment handout has been used to find the feature descriptor. The idea is to visualize a circle of some radius, $rpixels$ (with center as the center of the character image) around the character and project the corners on to the circumference of the circle. Then measure the arc lengths between each pair of projected corner points. We know that arc length $s = r\theta$ where θ is the angle between lines joining each corner point from origin. Since r is same for both testing and training images, it becomes a proportional constant and we can just measure angles. These N angles are used as a feature descriptor for a particular character.

Below are the details of the feature extraction method:
For each character image,

(1) Find the center of the image which is at half of the width and half of the height of the image.

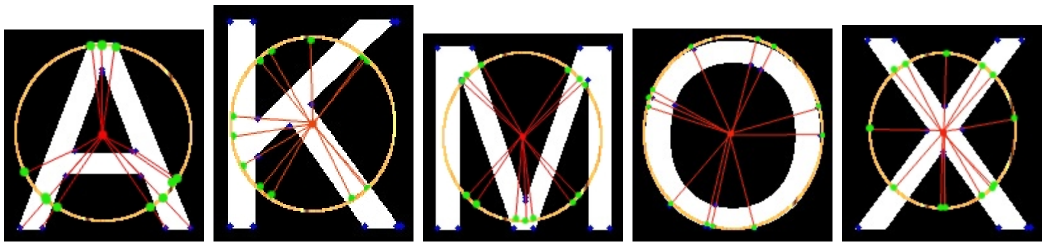
(2) Find the positions of the corners with respect to the center (or lets call it origin) by a translation.

(3) For each corner point, find the line passing through origin and the corner by using cross product method in homogeneous coordinates. If $(0, 0, 1)$ and $(x_c, y_c, 1)$ are the homogeneous coordinates of the origin and a corner respectively, then the line l joining them is given by the $l = (0, 0, 1) \times (x_c, y_c, 1)$

(4) To get angle between each line, we need to first sort them based on their angle with respect to $x-axis$. This is needed because without sorting we would end up finding angles between non-neighboring corners on the perimeter of the circle. So, if x-axis line is given by $(0, 1, 0)$ in homogeneous coordinates, then the angle between x-axis and line l is obtained using the dot product formula: $\theta = cos^{-1}(\frac{l \cdot (0,1,0)}{|l|})$

(5) We sort the corners based on their orientation with respect to x-axis and then take pair-wise difference between the angles made by their neighboring corner with respect to x-axis. This would give the angle (arc length) between each corners.

The feature extraction is shown in the figure below.

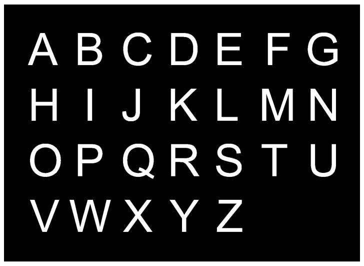


(a) A (b) K (c) M (d) O (e) X

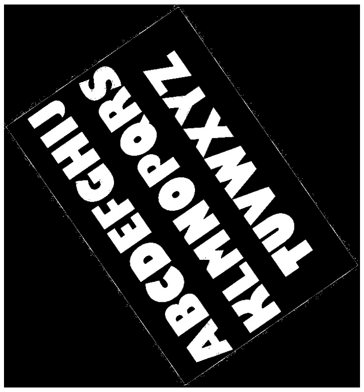
Figure 1: Feature Extraction - Blue points are detected corners, Red lines are lines joining origin to the corners, Green points are projected points of corners on the shown circle

The segmentation results are shown below for training image and all the test images

Some of the corner detection results are shown below for some training images and some test images. To see all the images, please check the attached zip file.



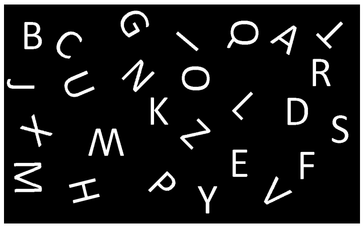
(a) Training



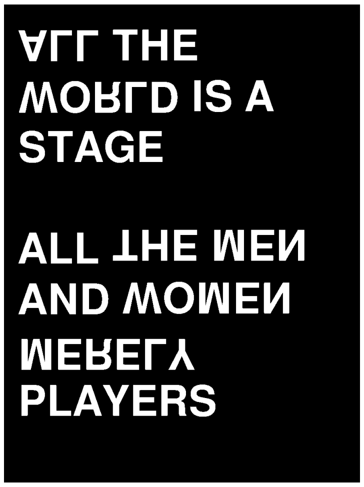
(b) Test Image 1

ABCDE
FGHIJK
LMNOP
QRSTU
VWXYZ

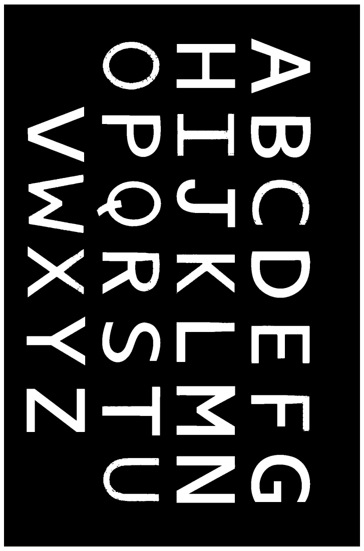
(c) Test Image 2



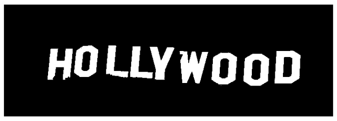
(d) Test Image 3



(e) Test Image 4

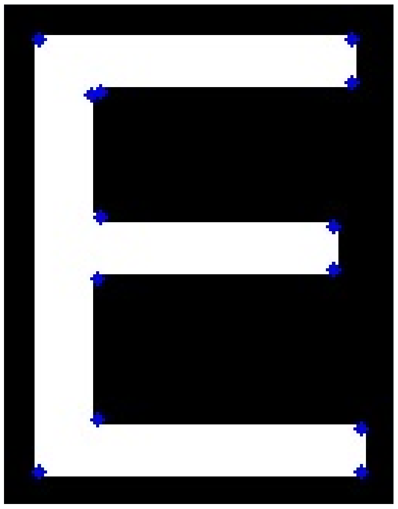


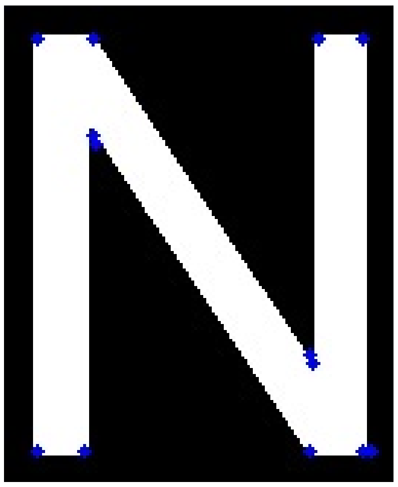
(f) Test Image 5

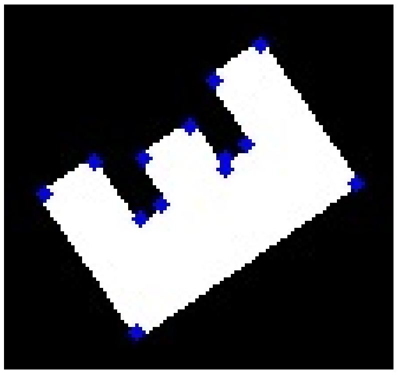


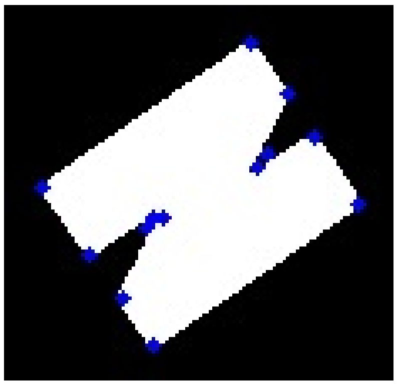
(g) Test Image 6

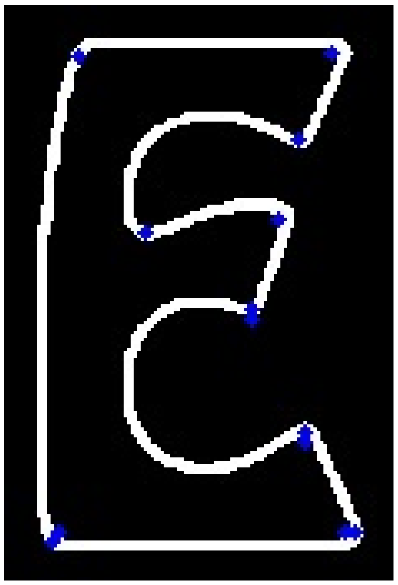
Figure 2: Segmented Images


(a) Training E


(b) Training N


(c) Image 1 E


(d) Image 1 N


(e) Image 2 E

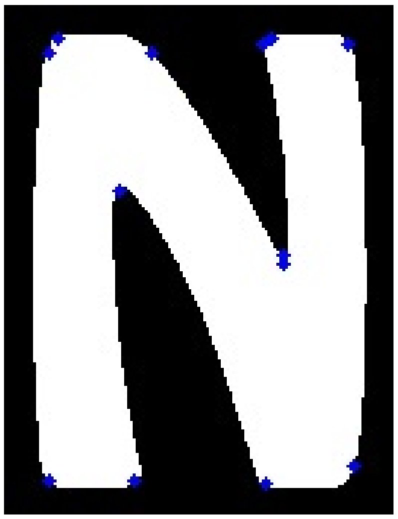
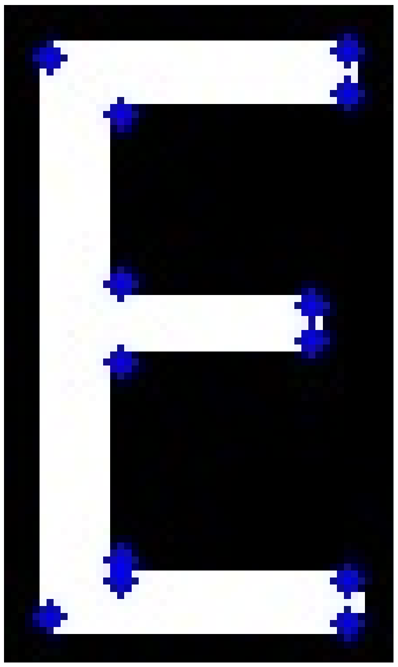
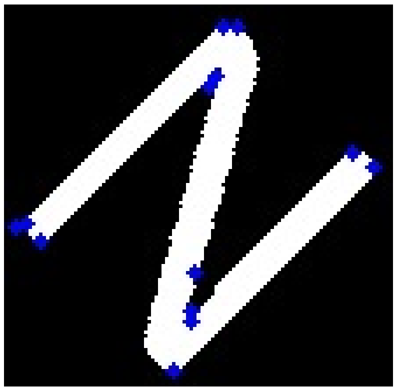
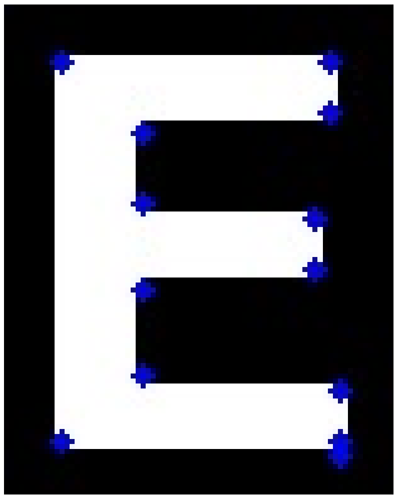
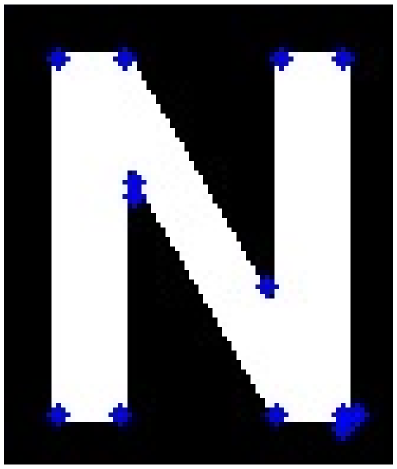

(f) Image 2 N

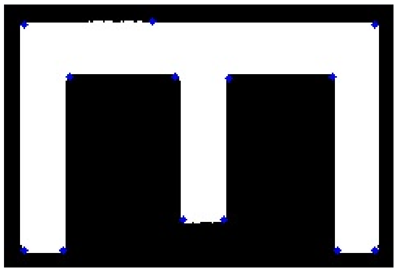
Figure 3: Some Corner Detection Results


(a) Image 3 E


(b) Image 3 N


(c) Image 4 E


(d) Image 4 N


(e) Image 5 E

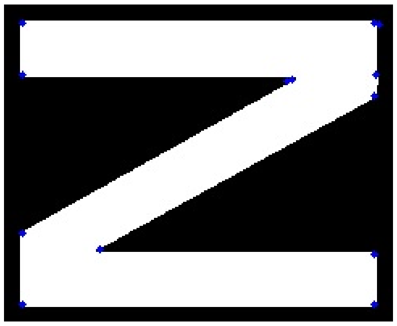
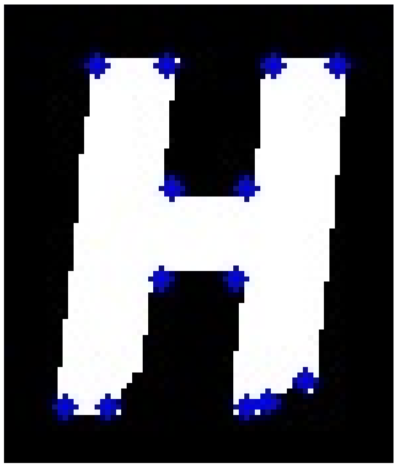

(f) Image 5 N

Figure 4: Some Corner Detection Results


(a) Image 6 H

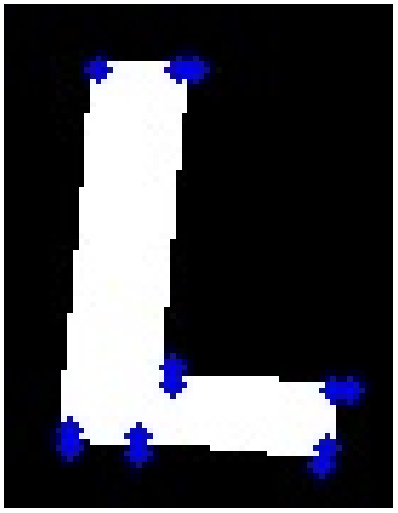

(b) Image 6 L

Figure 5: Some Corner Detection Results

2 Results

Actual Letter	Image 1 Detected	Image 2 Detected	Image 3 Detected	Image 4 Detected	Image 5 Detected	Image 6 Detected	Letter Accuracy
A	K	X	F	T, A, A, A, A, A	A	-	6/10 (60%)
B	F	F	O	-	J	-	0%
C	H	H	V	-	E	-	0%
D	B	T	W	D, D	W	Y	2/7 (28.57%)
E	B	M	H	H, H, H, H, Z, H, H	H	-	0%
F	C	Z	O	-	W	-	0%
G	X	F	K	G	G	-	2/5 (40%)
H	Y	Y	H	H, H	H	U	4/7 (57.14%)
I	Y	I	V	I	T	-	2/5 (40%)
J	J	V	T	-	J	-	2/4 (50%)
K	X	K	Z	-	B	-	1/4 (25%)
L	T	A	I	I, I, I, L, L, I, L	T	J, J	3/13 (23.04%)
M	X	E	Z	T, M, T	H	-	1/7 (14.28%)
N	Y	H	N	D, D, N	M	-	2/7 (28.57%)
O	X	F	X	H, H	C	M, U, K	0%
P	D	T	B	P	R	-	1/5 (20%)
Q	K	C	M	-	P	-	0%
R	O	O	H	X, X, H	E	-	0%
S	E	M	H	Z, S, F	K	-	1/7 (14.28%)
T	W	J	J	J, T, P	P	-	1/7 (14.28%)
U	Y	G	K	-	F	-	0%
V	J	J	M	-	S	-	0%
W	F	O	Z	W, W	M	T	2/7 (28.57%)
X	H	X	X	-	J	-	2/4 (50%)
Y	M	R	P	P, I	B	I	0%
Z	O	T	D	-	W	-	0%
Image Accuracy	1/26 (3.84%)	3/26 (11.53%)	3/26 (11.53%)	21/49 (42.8%)	4/26 (15.38%)	0/9 (0%)	**14.18%**

2.1 Observations

The performance of the method was not good compared to any existing character recognition methods. After some analysis of what exactly is causing lower performance, I came up with few reasons why the method may not work. They are listed below.

(1) The shape of the Character can't be captured uniquely for each character by angles

or arc lengths because there can be multiple corners along a straight line which gets projected to same point on the circle. Through this we are actually loosing the information about relative position of the corners. In the test images, the same phenomenon may not happen because of different font type.

(2) The center of the circle on which we are projecting the corners cannot be same for all the images because of their change in height and width of the segmented blobs. If the center of circle is not relatively same from all the corners, the angles will completely differ. They may even change from 60 degrees (in test image) to 100 degrees (in training). One can argue that we can resize all the images to the same size. This works only when there is no rotation between training and testing images. If there is a rotation, the aspect ratio of the rotated character will never be proportional to aspect ratio of the training character. This will not allow us to resize properly, especially if the character is non-symmetric. Example: L

(3) Feature vectors (angle between corners) are in such a way that they are dependent on each other (since they have to sum to 360 degrees). If one corner is not detected or detected at a wrong place in the test image, two feature vectors are changed, that too with lot of error. So, a simple euclidean distance measure cannot capture this inter-dependency between the features.

(4) Apart from these, the corners may not be detected at the same edges since different fonts will create different edge gradients and hence different corners. This is not a major problem though. Euclidean distance should still be reasonable for this.

Improvement?

I was thinking about using angles between each pair of corners and not just the neighboring ones. This will yield a NxN distance matrix (symmetric ofcourse). We can then use the 2-Norm or 1-Norm of the matrix as a measure to match the characters. This may take care of the dependency problem since we are calculating all possible angles and basically the feature vector is more expressive and discriminative. However, this will still suffer as long as there is a shift in the center of the circle with respect to the other corners from characeter to charcter. I haven't implemented this and hence cannot assure it will provide good results.