Depatment of Computer Science
Rutgers University
CS443 – Digital Imaging and Multimedia
Assignment 4
Due Apr 15th, 2008

This assignment is supposed to be a tutorial assignment that will lead you step by step to use Matlab image processing toolbox and other Matlab functions to build a hand-written character recognition system as a practice on binary image analysis. You are given a set of 9 images, each with different instances of one character in it. You are also given a test image for evaluation. Also you are given code for some functions that you will need. The images and related functions are available at ftp://ftp.cs.rutgers.edu/pub/elgammal/CS443/HW4

Create a directory for your work and put all images and code in it. In Matlab command prompt use `cd` to change directory to your work directory.

I- Reading Images and binarization.

1- You can open an image using the function `imread().` For example to open the image with character ‘a’ use

```
im=imread('a.jpg');
```

The variable `im` contains the image as a matrix. Check out the size of the image using

```
size(im)
```

Now, you can visualize the image using `imshow(im);`

2- In many cases we need to convert the image from a matrix into a vector. This can be done using matlab function `reshape`. Check out the help for reshape to see how it works (use `help reshape` in the command prompt). Use `reshape` to convert the image into dx1 vector where d= image rows * image cols.

```
im1=reshape(im,prod(size(im)),1);
```

Use `size()` to check out the size of the vector `im1`.

3- Now we will look into the histogram of the image intensity using matlab `hist()` function. You can specify the bins for the histogram as a parameter for the function. for example to make pins from 0 to 255 you can use

```
h=hist(double(im1),[0:1:255]);
```
Notice that `hist()` takes the input data as a vector. That's why we use `im1` here. Now, you can visualize the histogram in a new figure:

```
figure
plot(h)
```

Because the image is mostly background, there will be a very high peak at high intensity values that dominate the graph. You can visualize a smaller range of the histogram. e.g.,

```
plot(h(1:255))
```

4- Given this histogram we can choose a threshold to binarize the image. It's up to you to choose a suitable threshold. You can try different values and see the effect on the resulting binary image. To do this, first we define a variable called `th` for the threshold and set it to a certain value, say 200. Then, we create a new image, `im2` , with the same size as the original image. Then, we use logical operation to find intensity values greater(smaller) than `th` and assign these pixels to 0 (1).

```
th=200;
im2=im;
im2(im>=th)=0;
im2(im<th)=1;
```

To visualize the binary image use `imagesc()`. `imagesc()` scales the image from 0-1 to proper range for visualization:

```
figure
imagesc(im2)
```

You need to change the colormap if you like to see the image in black and white:

```
colormap gray
```

Here we use 0 (black) for the background and 1 (white) for the foreground. For printing it is always better to have white as the background. In such case, you can use the logical complement operator '~' to visualize the logical complement of the image, i.e.,

```
imagesc(~im2)
```

II- Extracting characters and their features

1- Given the binary image we have, we can now run connected component analysis to label each character with a unique label. To do this, we can use matlab `bwlabel()` function which performs connected component analysis on the image and return a labeled image where all the pixels in each component are given an integer label 0,1,2,... where 0 is the background.

```
L=bwlabel(im2);
```

2- You can visualize the resulting component image:

```
figure
imagesc(L)
```

In this figure each component has a different color since it has a different label. To find out how many connected components are in the image, you can find the maximum label used

```
max(max(L))
```

In fact you can find that number of components is actually more than the number of characters in the page. This is due to small isolated components that are mainly noise. Usually this is called salt and pepper noise. This can be removed using mathematical morphology. Or you can try to omit small size components from further analysis by simply comparing their height and width to certain threshold.

3- For each component you can find out and visualize the bounding box containing it using the following piece of code that loops through the components and find the maximum and minimum of their coordinates. To run this code, create an .m file and copy this code into it, give it some name, and call it from the command prompt after performing all the previous steps (The code assumes that the labeled image, `L,` exists in the memory).

```
Nc=max(max(L));
figure
imagesc(L)
hold on;
for i=1:Nc;
     [r,c]=find(L==i);
     maxr=max(r);
     minr=min(r);
     maxc=max(c);
     minc=min(c);
     rectangle('Position',[minc,minr,maxc-minc+1,maxr-minr+1], 'EdgeColor','w');
end
```

4- In this step we will compute the Hu moments and other statistical measures for each character. Provided with this assignment is a function `moments()` to perform this task. Put the function in your directory and type help moments to see help synopsis.

Usage: `[centroid, theta, roundness, inmo] = moments(im, plotchoice)`

You need to insert this function into the previous loop and pass into it a cropped image for each character as:

```
cim=im2(minr-1:maxr+1,minc-1:maxc+1);
[centroid, theta, roundness, inmo] = moments(cim, 0);
```

Here `inmo` is a four dimensional vector containing the Hu moments.

It would be useful to omit small size noise components as mentioned above before calling the moment function. Just add an ‘if’ statement to compare components height and width with a given threshold.

5- The next step is to modify the above code in order to store the resulting moments for each character to be used in recognition. To do this, simply create an empty matrix before the loop, let's call it '`Features`' using

```
Features=[]
```

Then, inside the loop you need to concatenate each character features to the existing feature matrix. At the end, '`Features`' will contain a row for each character with 6 features in each row. The concatenation can be done using

```
Features=[Features; theta, roundness, inmo];
```

III- Build Character Features Database for Recognition:

1- The final part of this project is to extract the features for all the characters in all the images given to you to create a database of character features to be used in recognition.
You will need to use the above steps to process all the character images and to extract features for all characters and put them into one big features matrix as above. Modify the above code by adding all the necessary steps from above for reading the image, binarizing, extracting components, etc. into one .m file where you can call it for different images.

Of course, you need a way to remember what is the character class for each row in the Features matrix. One way to do that is use another array with the same number of rows as `Features` where in each row you keep the corresponding class label, i.e., 1 for 'a', 2 for 'd', 3 for 'm' etc., or any appropriate class labels.

2- Once you create the big Features matrix and the corresponding class labels you are ready to do recognition. In this project will just use a simple nearest neighbor approach to find the closest character in the database for a given test character.

One problem is that different features have different ranges, so that a large numerical difference in a feature with high variability may swamp the effects of smaller, but perhaps more significant, differences in features that have very small variability. The standard solution is to transform all of the feature distributions to a standard distribution with 0 mean and variance of 1.0. This is done by first computing the mean and standard deviation of each feature (this is done over the entire set of training characters,
and not for one character type at a time), and then normalizing the features by subtracting the mean and dividing by the standard deviation for each feature.
3- To evaluate the recognition rate on the training data you can find the nearest neighbor for each character in the training data and check if its class matches the correct class. Here is an example to do that: we will use a function provided with this assignment called `dist2()` which returns the squared Euclidean distance between two sets of points. we will use it to evaluate the distance between each character and all other characters, i.e., the distance between the row vectors in the Normalized Features matrix:

```
D=dist2(Features,Features);
```

The resulting `D` is an NxN matrix where N is the number of characters (number of rows of Features). Typically `D` is called affinity matrix. you can visualize `D` as an image using `imagesc(D)`.

Obviously `D` will have zeros on the diagonal since the distance between each character and itself is 0. To find the nearest neighbor for each character (excluding itself) you need to find the second smallest distance in each row in the `D` matrix. One way to do this is to sort the columns of `D` along each row and to find the index of the second smallest distance in each row. To sort along the rows use:

```
[D_sorted,D_index]=sort(D,2);
```

The `D_index` matrix contains the index of the columns in `D` sorted according to the distances. So, the second column of `D_index` will contain the index of the closest match to each character (excluding itself). Find the class for this closest match.

4- You can compute the confusion matrix between character classes given the provided function `ConfusionMatrix()`, which takes as input, the correct classes (as a vector) , the resulting classes (as vector), and the Number of classes.

IV- Testing

For evaluation you are given a test image (test.jpg) with some characters. you will need to do the whole processing for this image and extract the features for each character. You will need to normalize the extracted features using the same means and standard deviations which were computed from the training data. Then, using the character features database obtained above and the function `dist2()`, find the nearest neighbor match for each character in the test image.

Deliverables:

In matlab, you can dump your entire command window work into a text file using the command `diary`. Use `diary <filename>` at the beginning of your work and all subsequenct commands will be dumbed into that file till you turn off the diary using
`diary off`

1- do parts I and II using ***one*** of the images and submit the diary file containing script of your work. Also submit all the figures you generated. For printing binary images always use white as the background. ***Only submit your script and figures for one of the images***

2- For part III: submit your code for processing the images and submit a figure visualizing your D matrix. Report your recognition rate. Print the Confusion Matrix.

3- For part IV: submit your code for recognizing the characters in the test image as well as print a list of each component and the resulting class. Your result should be in this format: for each component print its label, coordinates (minr, minc, maxr maxc), and the recognized class. What is your correct recognition rate?