Department of Computer Science, Rutgers University

CS334 – Computer Vision

Assignment 4

Due Sunday Dec 11th, 2016

1. Introduction

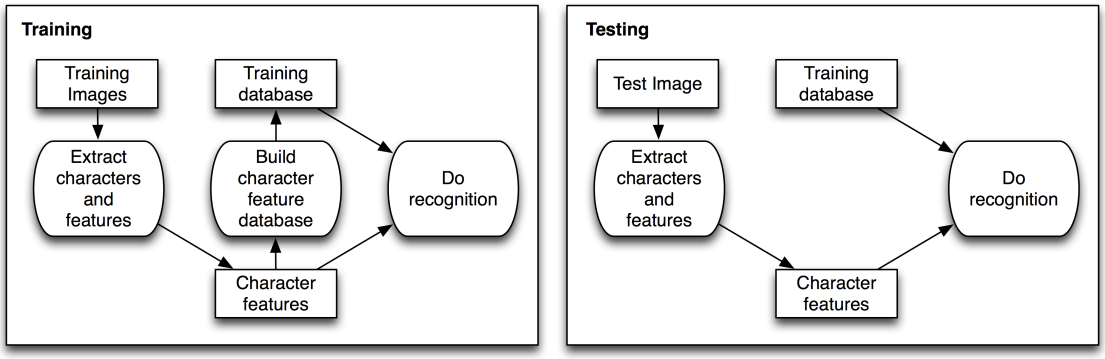
This assignment is supposed to be a tutorial assignment that will lead you step by step to use the Matlab image processing toolbox and other Matlab functions to build an optical character recognition (OCR) system for hand-written characters, as a practice on binary image analysis. You can also use Octave which is an open-source software similar to Matlab

Please note, 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 subsequent commands will be dumped into that file till you turn off the diary using

```
diary off
```

you will need to save your work into a diary as part of the deliverables for this assignment

1.1. Problem Overview



In this assignment you will be given grayscale images of hand-written characters, where you will need to **identify** (extract) and **recognize** (classify) each character.

The assignment has two phases: Training and recognition. After completing these, you will be asked to improve your recognition rates by providing your own ideas, which is the enhancement part.

For both training and recognition, you will need to convert the grayscale images to binary images, identify each separate character instance appearing in the image and extract some visual features from these character instances.

In the training phase, you will build (learn) a 'database' of features that you extracted from a given set of character images. These features will be put inside a matrix that will be later used as a recognition database. At this phase you will know what each character is (its label) and you will also store this information along with the extracted features.

In the recognition phase, you will use the built database of features, the associated character labels and the features you extracted from your test image to recognize each character in this test image. Features extracted for each character instance in the test image will be compared to all the features in the database and the label

of the 'best' matching character instance in the database will be the recognition result for this test character instance (this is called nearest neighbor classifier).

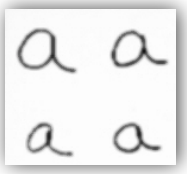
For crosscheck you will also be performing the same recognition step for the training images and measure the how well you have performed. This way you will know if your features are descriptive enough for your characters and you will also have the chance to catch possible errors in your implementation earlier.

1.2. Given Files

There are several images and code files we supplied with the assignment.

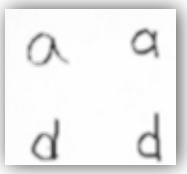
1.2.1. Images

Training Image Set: You are given a set of 16 images ('a.bmp', 'd.bmp',… , 'z.bmp'), each with different instances of one character in it. Images can be of different sizes.



Part of the training image 'a.jpg'

Test Image: You are also given two test images for evaluation ('test1', 'test2'), which contains different instances of the hand-written characters in the training set, combined in one image.



A part of the test image 'test1'

1.2.2. Code

Also you are given code for some functions that you will need. You will NOT change these files or re-implement their functionality when doing this assignment. See the files for their usage.

ConfusionMatrix.m: Computes the confusion matrix given ground truth data and classification results.

dist2.m: Calculates squared distance between two sets of points.

moments.m: Calculates the invariant moments, centroid, angle of minimum inertia and roundness of an object in a binary image.

1.2.3. Downloading the Files

The images and function code files are available at sakai resources.

2. Implementation

2.1. Training

2.1.1. Reading Images and Binarization

Reading an Image File

You can read an image file into a matrix using the function `imread()`. For example, to open the image with character 'a' use:

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

Now, the variable `im` contains the image as a matrix. Check out the size of the image using:

```
size(im)
```

Note: *Observe that Matlab stores matrices/images, where first index is the row index and second is the column index. This is the reverse order of the most of the programming languages and common mathematical notations.*

Note: *Observe that all the given images are grayscale images. The size of a grayscale image only consists of the height and the width of the image in pixels (HxW, e.g. 750x600). If an image is a color image then the size consists of height, width and the number of color channels (HxWxC, i.e. HxWx3 for an 3 channel/RGB image, i.e. 750x600x3).*

Visualizing an Image/Matrix

You can create a new figure using Matlab `figure()` function:

```
figure
```

Matlab always renders the desired visualization to the currently selected figure. If you want to keep a previous visualization, a plot, an image and continue from another figure you can create a new figure and use this figure for the rest of your operations.

Now, you can visualize the image using:

```
imshow(im)
```

You can also try other image/matrix visualization tools in Matlab to see the differences:

`imagesc(im)` and `imtool(im)`

You can name your figures using Matlab function `title()` so you will know what image they are displaying when you go back and look at them:

```
title('Character a')
```

Image Histogram

Now we will look into the histogram of the image pixel intensities using Matlab `imhist()` function, which is a specialized histogram function that works on images:

```
h = imhist(im);
```

Alternatively, you can use the more flexible but more complex Matlab `hist()` function. Here you need to specify the bins for the histogram as a parameter for the function and also you need to supply a one-

dimensional vector of intensity values instead of a 2-dimensional matrix. For example to make bins from 0 to 255 you can use:

```
h = hist(double(reshape(im, numel(im), 1),[0:1:255]);
```

Matlab `reshape()` function converts a multi-dimensional matrix to another shape without changing its content. Here specifically, `reshape(im, numel(im), 1)` command converts our 2-dimensional image into a one dimensional vector by putting all the pixel values one after the other. Matlab `numel()` function returns the number of elements in a matrix. This can also be achieved by multiplying all dimensions with each other by `prod(size(x))` .

Visualizing a Histogram/Vector

You can visualize the histogram in a new figure:

```
figure
plot(h)
title('Intensity Histogram')
```

Because the image is mostly background (white pixels), 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., to display the first 255 elements out of existing 256, you can use:

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

Note: *Observe that Matlab vector/array/matrix indexing starts from 1, where in most of the other programming languages its starts from 0. For example, the first element in a Matlab vector v is v(1), where the first element in a C/C++ array u is u[0]. Asking for v(0) in Matlab will give you an indexing error, unlike C/C++, these errors will be more descriptive and you will be able catch them easily.*

Binarization by Thresholding

Given this histogram we can choose a threshold to obtain binary image from the grayscale image (binarize the image). It is 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;
```

Actually a much easier ways to do this operation is to use:

```
im2 = uint8(im1 < th);
```

This generates a binary (logical) image and converts it into type uint8 (same as unsigned char in C/C++ or byte in Java).

To obtain the threshold automatically, try the triangle algorithm, described in class. Use the implementation provided, which can be used as follows

```
level=triangle_th(h,number of bins in h);
th=level*255     % level is in [0,1]
```

You can also try Matlab threshold function `graythresh()` if you have access to it (it is available as part of the image processing toolbox)

Describe which technique you decided to use and why. You can visualize the resulting binary images and compare them as described next.

Displaying Binary Image

To visualize the binary image, use `imagesc()`, which scales the image from its minimum and maximum values 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)
```

2.1.2. Extracting Characters and Their Features

Connected Component Analysis

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);
```

In newer versions of Matlab image processing toolbox, you can instead use `regionprops(im2, 'BoundingBox')`

Visualize Component Image

You can visualize the resulting component image:

```
figure
imagesc(L)
title('Connected Components')
```

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 value appearing in the labeled image using:

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

In fact you can find that the 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.

A sample character instance from character 'o' and its thresholded version. You can see the small component that will be identified as another letter instance in the labeling process and thus has to be removed before feature extraction.

Displaying Component Bounding Boxes

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 the name **BoundingBox.m** to the file and call it from the command prompt after performing all the previous steps.

```
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
hold off
```

Note: *This kind of Matlab files, which lists the commands as you would call them from Matlab command prompt are called script files. The other type of Matlab code file is a function file, which starts with a function declaration with the list of input and output parameters and the function name. Function files can only use parameters and local variables and return results in output variables, where script files can only access and generate global variables.*

Note: *Most of the times it is better to try a code using a script file and when debugging is finished convert it into a function file. Creating a function for every reusable piece of code can help avoid code repetitions and reduce the amount of code you need to write by a significant amount.*

Once the bounding box drawing works, try to convert it into a function file. Later on you may need to use this function in other places of the assignment.

Computing Hu Moments and Removing Small Components

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 size threshold. Experiment and visualize your bounding boxes until you find a good size threshold.

Storing Features

The next step is to modify the above code in order to store the resulting features 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 one row for each character with 6 features in each row (1 theta, 1 roundness and 4 from Hu moments). The concatenation can be done using:

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

2.1.3. Building Character Features Database for Recognition

Creating a File to Process Each Image

Now that we have extracted features for one image, the final part of training phase 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. Make this file a function file with the name **MyExtractorFunction.m**. The file should start with a function declaration and would need to get all the necessary information as input parameters and should use output parameters to return the outputs. It would also be nice to have a parameter that controls whether all the plots are displayed or not.

Of course, you need a way to remember what is the character class for each row in the Features matrix, since there may not be equal number of character instance for each character. One way to do that is to 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. Another way is to keep the labels in the Features matrix as the first or the last column. If you follow this way, you will need to pay a lot of attention not to include the labels column in any of the normalization, training or recognition steps. You can also come up with your own method to associate the labels to the features.

Normalization

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. It would be nice to store these means and variances so that you can reuse them in the recognition phase.

Recognition on Training Data

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. Since the particular character instance itself was in the training, its feature vector will also be in the feature database and definitely this will be the closest feature vector (with distance zero). For the case of recognition on the training data we will need to find the second closest row to the feature vector we are looking for. Hopefully, the next best match will correspond to another instance of the same letter, but sometimes other letter instances can be closer.

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. The distance between the row vectors in the Normalized Features matrix can be found using:

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

The resulting `D` is an NxN matrix where N is the number of characters (number of rows of Features). You can visualize `D` as an image using:

```
figure
imagesc(D)
title('Distance Matrix')
```

For example `D(1, 5)` will give you the distance between character instance 1 and character instance 5. 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 and `D_sorted` contains the sorted distance values. 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 by looking at the label corresponding to this instance in the label vector you constructed earlier.

Confusion Matrix

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. The confusion matrix elements will contain a normalized measure of how much each class is confused (recognized wrongly as) with another. This matrix is not necessarily symmetric, since i.e. ‘a’ can be recognized as ‘o’, but ‘o’ is recognized as ‘u’. The diagonal elements in the matrix are the cases where character classes are recognized correctly. Visualize this matrix and try to understand the reasons of very obvious confusions between several classes:

```
figure
imagesc(conf)
title('Confusion Matrix')
```

2.2. Recognition

For evaluation you are given a test image (test.jpg) with instances from all 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. As opposed to training phase, this time find the best match instead of the second best.

2.2.1. Reading Image and Binarization

Read the test image and binarize it for feature extraction. You should be using exactly the same process you followed in the training phase; you cannot change any of the processing steps or any of the parameters specifically for testing. Normally, you should not be using the test data to improve your algorithms, which may cause over tailoring you algorithms for this particular test data; but for this assignment you can change your algorithms by looking at how they perform on your test data.

2.2.2. Extracting Characters and Their Features

At this step make sure you fix all the problems you can fix related to small pieces of characters that may appear as different components.

When storing extracted features from the test image, this time you will not be saving any label for the characters since this is the information you want to obtain from the our recognition system.

Make sure you performed the normalization using the mean and variance stored for each feature dimension in the training phase. Do NOT normalize test data using mean and variance of the test data features themselves.

In the recognition phase, you will be calculating a distance matrix, but not a confusion matrix.

2.3. Enhancements

After fully completing the training and recognition parts, you can experiment with different ideas in order to improve your results. Try to fix any problems associated with the previous parts of your code before attempting to enhance it. If an enhancement is a successful one, it should increase both character recognition rate of the test image and the training images. An enhancement that increases one recognition rate, but decreases the other is not a good one.

Improving your results using different enhancements will contribute up to 20% of the assignment grade.

An enhancement can be completely experimental or it can be a remedy for a shortcoming that you observed in the current setting. If you are specifically testing an enhancement that should fix a problem in the current setting, try to observe/measure the resulting improvement on the problem you were targeting, instead of only measuring the recognition rate improvement. It may be a good idea to test improvements independently first, then combined together. You would not want to test ten different enhancements and be not sure which one made an improvement and which made it worse.

Make sure you document every thing in your report, even the unsuccessful ideas and trials.

In your report we would like to see:

- What you observed: A problem or an unsatisfactory point.
- What you deducted from it: Why this is happening, what is the main reason of this.
- What solution you came up with: A candidate solution designed to solve this particular problem.
- How this solution worked: Improvements in recognition rate from this specific enhancement and how the problem is solved, maybe with sample images or measurements. If failed to fix the problem or it introduced another problem, why do you think it failed.

2.3.1. Enhancement Ideas

- Automate the threshold selection process: instead of a fixed hard-coded threshold, find methods that analyze the images and find the best threshold.
- Use binary morphology: Using binary morphology might be useful if you have fragmented characters that you cannot get rid of solely by improving the thresholding. There could be many other uses for binary morphology.
- Investigate different shape and region descriptors to extract invariant features to be used in the recognition: More invariant moments, shape profiles, contour descriptors, etc. Make sure you perform a meaningful normalization on each new feature you included. Sometimes you will not need to normalize a feature using mean and variance, but maybe using its known bounds.
- Use a better classifier: Instead of the nearest neighbor (closest distance) from the features' database for recognition, you can find the k-nearest neighbors (k is small number 3, 5, 7) and do a majority vote.

You don't have to investigate all the above possibilities. These are just some ideas. You need to come up with a way to enhance the recognition rate. You should use the training data for validating your proposed approach and the test image for testing only (no training done on it). Keep in mind that your final code will be evaluated with a similar image, which will not be given to you. Your grade in this part will be based on how much improvement you can do over the base line established in previous parts.

3. Expected Deliverables

Report Document

Put all your deliverables in the report with the code and **submit a soft copy to Sakai**. The document must be either in **PDF**, **DOC** or **DOCX** format; reports in any other format **will NOT be graded**. Embed all figures in your report in correct order and properly label them. You do NOT need to submit your figures as separate image files.

The report should contain the following.

3.1. Diary

For parts 2.1.1 and 2.1.2: do these parts using ***one*** of the images and submit the diary file containing the step-by-step script of your work and the matlab outputs (using matlab diary as mentioned above). Also submit all the figures you generated in these steps. For printing binary images always use white as the background. ***Only submit your script and figures for one of the images. Please use the image that corresponds to the firstletter(your last name). if there is no such image, use the one corresponding to the closest letter prior to the firstletter(your last name) that has an image***

Along with the diary you should include the following resulting figures

- Grayscale image (`im`)
- Intensity histogram (`h`)
- Trials of different thresholding functions, including `triangle_th`, (and `graythresh` if available to you)
- Binary images (`im2`) after thresholding with different techniques,
- Connected component image with bounding boxes (`L`)
- Distance matrix (`D`)
- Confusion matrix (`conf`)

3.2. Code Files

You should submit all the code that you have written. You will NOT change any of the given code files hence you do NOT need to submit these files. Do NOT use other people's code or code files you found from Internet; however if it is absolutely necessary to use some external code, make sure you submit these files too. Submitting a non-working assignment may affect your score. You can use any Matlab toolbox function that exists in/upto R2014.

RunMyOCRRecogiton.m

Submit your final version of your code so that the TA can run it on some other images to evaluate performance. Create a function that takes an image filename as an input, runs your training algorithm on the training data, runs your recognition algorithm on the given test image and reports the training and testing recognition rates. The function should also create all the images appearing in the report, but the most important figure is the final recognition figure on the given test image. This figure should be the connected component test image with detected bounding boxes and corresponding recognized character classes.

To be able to generate recognition rates for any given test image, you need to have access to the groundtruth character locations and their correct classes. Your function needs to have this information as two separate

matrices: **Locations** and **classes**. Locations will be an Nx2 matrix of rough center coordinates of characters and classes will be an Nx1 matrix of the corresponding class label for each character, where N is the number of characters in the image. You will need to understand that labeling process may change the order of how the components are labeled and because of this, you cannot just feed class label corresponding to each component number directly as groundtruth. To compare to this groundtruth, you can go over your detected and recognized components and see if they contain any one of the given groundtruth character centers. If a center is in a detected component, then you can compare its groundtruth label and the recognized label. This way you can create a recognition rate finding mechanism that does not require that you have the same number of components as your actual characters. You will need to find the correct groundtruth points for the test image yourself. You can make use of Matlab function `imtool()` for this purpose.

Since you have access to groundtruth, you can also visualize each correct character label next to the recognized one. You can use Matlab function `text()` to print text on to a figure.

The result image should have the components, bounding boxes and correct labels and recognized labels. Function signature:

```
function result = RunMyOCRRecognition(filename, locations, classes)
```

Please use this coding for the class labels:

a->1

d->2

f->3

h->4

k->5

m->6

n->7

o->8

p->9

q->10

r->11

s->12

u->13

w->14

x->15

z->16

3.3. Results

For All Training Images

- Connected component image with bounding boxes and recognition results

For Recognition Phase

- Distance matrix (`D`)
- Test image connected components with bounding boxes and recognition results
- Test image connected components with bounding boxes and recognition results - For every (successful) enhancement
- Test image connected components with bounding boxes and recognition results - For all enhancements combined.

Recognition Rates and Other Values

In your report list the following values:

- Threshold value you have picked, or any algorithm you used to find a threshold
- Number of components you obtained for test image
- Recognition rate for each training image
- Recognition rate for the test image
- Recognition rate for test image after each enhancement
- Recognition rate for test image after all enhancement combined

Tested/Used Features

List every different feature that you have used in your code. Try to justify why you think they will work. Document your experiments in the report.

4. Grading

Base Recognition Rates

Without any enhancements, the base recognition rate for the given procedure is limited. You still can improve it by picking a better threshold. You are expected to achieve close to this recognition rate before applying any enhancements. This part will be up to **80%** of your grade.

Enhancements

After obtaining a recognition rate close to the base recognition rate, you will try different methods to improve this result. This part will be up to **20%** of your grade.

Report

Reporting everything clearly and putting all necessary figures in to your report.

Grade Penalties

- **10%** for late submission up to two days.

- Up to **10%** if the code is not running for some reason and/or not giving significantly different results than the reported ones. Do not forget to submit all the files you have written and any external files that you have used.
- Up to 20% if your report lacks necessary figures or details of your experiments.
- Up to **50%** for group work and/or code sharing, downloading. See Academic Integrity section.

4.1. Academic Integrity

This is an **individual assignment** and it has to be completed by each person independently and **not with groups**. Sharing code or/and specific ideas is not allowed. The code, results and report that you have submitted will be compared with other people's submissions as well as online code repositories, and your grade will be severely affected if a clear resemblance exists. Instead of consulting to your classmates, please contact the TA or the Professor to clear out any confusing points about the assignment.