Introduction

Your first project involves building a vision system that can locate and recognize hand printed characters in a digital image. There are three stages to this project:

1. Segmentation of characters from a digital image of a page of text.
2. Connected component analysis to identify the individual characters in the segmentation.
3. Property measurement and classifier design to recognize the different character types, as well as to reject any connected components which do not match sufficiently well against any character model.

You will be given a set of 10 training images. Each image will contain roughly 50 instances of a single hand written character (for example, an ‘‘a’’), although the different instances of these characters might have been written by different people. The characters will all be more or less the same size, and they will be written in the same orientation (so, none will be upside down or rotated). You will apply your segmentation and connected component algorithm to this set of training images in order to construct the recognition models that will be applied, subsequently, to a testing set of images.

In what follows, I will provide details about each stage of the project.

Segmentation

The training images will all be 8 bit gray scale images, which will be given to you as simple files, the format of which is described in more detail below. You will segment these images using a combination of thresholding and binary noise cleaning:

a. Ideally, images of hand printed pages should be composed mostly of very dark pixels, corresponding to the ink of the characters, and very bright pixels, corresponding to the white paper on which the characters are drawn. However, since the character edges will not align with the pixel boundaries, and the pen will not have been applied with constant pressure while the character was drawn, there will always be many pixels of intermediate gray level. Camera noise, for these images, will not be a significant problem. You will segment the images by finding the two peaks, corresponding to the very large background population and the distinct, but much smaller, ink population. There will be a broad valley between these two peaks, and it can be important to choose the threshold carefully in this valley. The simplest solution is to find the gray level with smallest frequency between the two peaks, and use it as a threshold. A better method chooses the threshold that maximizes the total contrast along the boundaries of the segmented objects. You do this as follows:

1. First, create a contrast image. Let I be the original image and C the contrast image we are trying to construct. For each pixel in I, compute the largest absolute difference in gray level between that pixel and its four neighbors. Place this absolute difference into the corresponding location of the contrast image, C. At the boundaries of I just use the available neighbors to construct C.
2. Now, using a single pass through I we will determine the threshold that gives maximal total contrast along the 1/0 boundaries of the corresponding binary image (i.e., at those pixels that are 1 in the binary image generated by

threshold t and that have at least one 8-adjacent neighbor that is 0 in that binary image). This program will construct a table, S, that will contain one entry for every threshold under consideration. The program will work as follows:
i. consider each pixel in I and determine the subrange of thresholds that would make this pixel a 1/0 boundary in the corresponding thresholded image. This is easy to do. Let g be the gray level at a point (i,j) in I. Let M be the maximum gray level in the 3x3 neighborhood of (i,j). Now, any threshold in the interval (g, M] will make (i,j) a 1/0 boundary point in the corresponding thresholded image. Note that if g >= M then there is no threshold that would make (i,j) a 1/0 boundary pixel.
ii. For each t in this interval, you increment St by C(i,j). After you have processed all of the pixels in I, St will contain the sum of all the contrast values for pixels that are 1/0 boundary pixels in the binary image corresponding to t.
iii. Choose the t that gives you maximal St.

b. Noise cleaning. It is possible that the thresholding step will produce small components of 0’s and 1’s, or slightly ragged edges on some objects. These can be often eliminated by applying 1-2 iterations (each) of a simple shrink/expand operation. You shrink a binary image by replacing all 1’s that are adjacent to 0’s by 0’s. You expand a binary image by replacing all 0’s that are adjacent to 1’s by 1’s. Be careful of the type of adjacency used in the two cases! It is dangerous to shrink an image for too many iterations, because thin characters will be annihilated altogether. But for our data a single iteration of shrinking (applied to the thresholded image!) followed by a single iteration of expanding should be sufficient. But you need to check the results of the noise cleaning, since it might actually reduce the quality of the original segmentation.

Connected component analysis

In the second stage you will use a run-based, two-pass connected component algorithm to find and measure the properties of all of the objects in the binary image. Make sure that you use 4 connectedness for the background (0’s) and 8 connectedness for the objects (1’s). For each object, compute its:

1. centroid, as a way to locate the character (this is not used for recognition).
2. compactness - ratio if its perimeter squared to area
3. ratio of area of holes in object to total area of object
4. aspect ratio of minimum bounding rectangle.

You can choose other features in an attempt to obtain good classification. Note that the characters vary somewhat in size (i.e., not all ‘‘a’’s are the same size) so that using simple features like area, perimeter will not work well.

Recognition

Your recognition algorithm will be based on a k--nearest neighbor rule. There are two subproblems that have to be addressed here:

1. How to weigh the different features in computing the distance between an unknown character and a character from the training set. The 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, u, and standard deviation, s, of each feature type (this is done over the entire set of training characters, and not for one character type at a time), and then replacing a feature measurement, x, by (x-u)/s.
2. Selection. Since we have 10 classes and about 50 samples per class, a brute force nearest neighbor decision procedure would involve comparing the feature vector of an unknown character against 500 feature vectors for the known characters from the training set. Now, for 500 total characters this is not much of a burden, but generally we would have thousands of objects in our training set, and we would like to edit the training set to a smaller size that would give us as good a classification as using the entire training set. I am going to leave the problem of developing an algorithm for effectively editing the training set up to you to solve individually! In order to measure how well your solution to this problem works, you will apply the brute force nearest neighbor algorithm to the test set as well as your clever solution to the editing problem. To see how well you are doing before we release the test set, you should employ the following jackknifing procedure to the training set:

* leave one character out of the training set
* build your classification model with the remaining data
* apply the classification model to the single character left out
* repeat this for all characters in the training set.

This is really not so bad because it only involves renormalizing the data sets (since the means and standard deviations change slightly when you take one element out of the set and put one back in and there are clever algorithms to do this) and then performing one classification test.

Image Files

This section provides information about the format of the image files containing the training images. The file format we will use is PGM ((P}ortable {G}rey {M}ap). PGM files can be viewed on an X--workstation using the program xv. You may obtain these images from the course ftp site

ftp.umiacs.umd.edu/pub/lsd/

Each file has a pgm extension, and the file name is the name of the character who training samples are contained within that file. A (binary) PGM file is formatted as follows.

1. The first line in the file is a ‘‘magic number,’’ a two digit string, which you can ignore. (It should be P5). You can read this using fscanf function call in C.
2. The second line of the file contains pair of integers, formatted in ASCII, containing the width (number of pixel columns) and the height (number of pixel rows).
3. The third line of the file contains the highest grey-scale value. For our purposes, this should be 255, and can be ignored.
4. The remainder of the file is a set of character values, written in binary. The values are the pixel intensities of each row written in normal English reading order. A value of 0 means black, and the maximum value (255) means white. Because these values are binary, you might want to use the getc function call to read them.

Each training character in the image is contained in a 75 x 75 square. Thus the upper left character is contained in the square with upper left coordinate (0,0) and lower right coordinate (74,74). Please note that when you are given the test image it will NOT have characters laid out in the same 75 x 75 grid. Only the training images are organized this way - so, your programs should not have these numbers hard coded!

Submitting your Project

You will hand in your program and a report which should include the following:

1. A description of your solution to the editing problem
2. Results of the jacknifing procedure on the training set. This should include a 10 x 10 confusion matrix. The (i,j)’th entry of this matrix is the number of times that a character in class i is classified as a character in class j. Ideally, you would only have nonzero entries on the main diagonal (all correct), but this is unlikely to happen. Results using both the edited and unedited data sets should be provided.
3. Similar results on the test set.

Additionally, the TA will be testing the connected component part of your program; details on this will be forthcoming.