Project 2 – Computer Vision – CS 6643 – Rajiev Timal - 0220052

1. **How my program works**
   1. The first thing that the program does is open up sample digits.bmp and get the bounding boxes of the letters, as well as the features for each number that is present in each box. This file is binarized first, even though it really doesn’t need to be because it’s already binarized. My algorithm scans from top to bottom, left to right. When it hits a number, it uses the floodfill algorithm to fill the shape. This algorithm operates on a 2D array called pixmap, which is a duplicate of the image, and contains 1’s for black and 0 for white. Floodfill returns the minx, maxx, miny, maxy value of the character in an array called bounds[4]. It does this until there are no more black pixels that are unfilled. The data structure that I use to store bounding boxes is :

struct Boundingbox{

int maxx; //max x coordinate

int minx;//min x coordinate

int maxy; //max y

int miny; //min y

int area;//area of box

int areacharacter;//area of character within box

float averager;//average r, used in rbar, cbar, the centroid - normalized for height/width of box

float averagec;//average c - normalized for height/width of box

//featurevector[0] is the row moment of the character

//featurevector[1] is the column moment of the character

//featurevector[2] is the mixed moment of the character

//featurevector[4] is the Euler number of the character

float featurevector[4];

string c; //c is the character that is contained within the box};

Every image has a C++ map of structs, e.g. map<int, Boundingbox> samplesboxes;

So for instance, samplesboxes[0].areacharacter would give the area of the character, and so on.

* 1. The 4 features that I used are the Row moment of the character, the Column moment of the Character, the Mixed Moment of the character, and the Euler number of the character. The first three require the calculation of the centroid which was done by dividing the average row value obtained with the height of the box and the average column value with the width of the box; this was done to normalize the centroid. The formulas for the first three features were taken directly from the course textbook. They are scale and translation invariant. The Euler number, the 4th feature, was found by using a scanline that goes horizontally across the middle of the characters. If the line intersected the character twice, this means that there was a hole, and the character had to be either a 0 or a 4. 0 and 4 Have Euler numbers of 1-1 = 0;
  2. Next, IMG\_1.bmp goes through a slightly different processes than the samples. It is binarized, and the bounding boxes are returned by floodfill. However, boxes that contain only a connected component greater than 40 pixels in area are returned. The number 40 was arbitrarily chosen as a lower threshold for the area of a bounding box. For any smaller bounding boxes, the contents of that bounding box in the image were erased by a function called removeartifacts(). This guaranteed that all that was contained in the bounding box returned was the connected character, in this case a number that we wanted to obtain. Once we have the bounding box with the features, we now call getcentroidinfo and extractFeatures to obtain our 4 features.
  3. The function :

void classifyandoutput(CImg<unsigned int> \*image, int numboxes,map<int, Boundingbox>& boundingboxes,map<int, Boundingbox>& sampleboxes,map<int, Boundingbox>& sampleboxes2);

Takes as its input a pointer to the image, the number of boxes in that image, the C++ map containing the bounding boxes of that image, the map containing the sample bounding boxes, and the map containing the second set of samples (5-9) which are not used in this assignment but maybe used if I can do the extra credit. A minimum distance classifier is used in this function to assign the right character to a boundingbox, based on it’s distance from the characters in the sample. The equation:

sqrt((X1-XS1)^2+(X2-XS2)^2))..etc is used, where X1 and XS1 are each feature vectors of the current bounding box and the bounding box of a sample character, respectively. (all the way to Xn and XSn, where N is the number for features). For each bounding box in IMG\_1, the minimum distance is found to all 5 of the initial sample characters (0-4). The sample bounding box character with the minimum distance to the IMG\_1 bounding box character has its character assigned to the IMG\_1 bounding box character.

This process is repeated for all the images, and in the end the r,c coordinates of the top left and bottom right of the bounding box of the characters, as well as numbers determined to be contained in the boxes, are outputted to a file images/classes.txt.

Summary of Results and problems encountered.

1. The only image that yielded 100% accuracy when classified was IMG\_1.bmp

The rest of them have less than 50 percent accuracy. This is partly due to the fact that I could not figure out more workable feature vectors. The degradation in image quality after binarization of the other shapes altered the shapes so much that my features did not work well to classify them.

1. Another issue is that the alphabet members of the reject class are not classified in the reject class. This is because the features that I chose classifies them too closely to the actual numbers, so there is no way based on the minimum distance that I can put them in the reject class.
2. Otsu’s algorithm from project 1 did not binarize the images well enough for those images to be used for segmentation. I picked manual threshholds (50 or 70) for all the images that worked well to binarize them and completely separate the characters from the background.

Compilation instructions:

The entire project folder with the “.sln” file is included. A library called CIMG is used to read/write pixel values. This library is also included in the project. All that is needed is to click the .sln file (OCR.SLN) in the Top –level “OCR” folder. This will open a vc++ project that is ready to run or debug. ‘OCR.cpp’ is the filename of the source code of the project and is contained in the lower level “OCR” folder that is included. There is a folder called images in OCR/OCR/images. sample digits.bmp and all of the test images up to IMG\_6.bmp should be added here for the program to run. These were included with the project, because they didn’t make the filesize too big. The program will also save binary versions of the image here. **There is a file called classes.txt with the results(bounding box top left, bottom right and character) that is also in the images folder. This file is appended to, so should be deleted if new results are to be obtained.** This file is also attached to the project email. The program console also outputs the letters that are detected by the program, as well as its minimum distance to the corresponding sample character.