**Number Characteristic Recognition**

Course: ECE 438, Digital Image Processing

By: Lucas Gillette & Chad Robke

Due Date: 12/15/2022

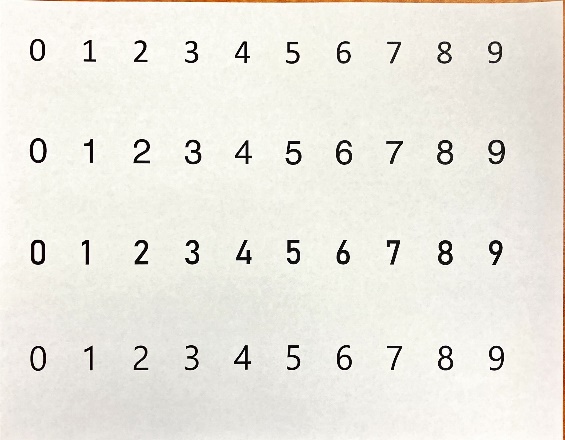
Project Outline

The goal of our project is to accurately determine a number that is on an image. How the program works is, a user will take a photo of a number. That photo is uploaded to a computer and then ran through a MATLAB program that was designed by us. The MATLAB program runs its code and determines the number that is presented on the image. The code currently only works on characters from 0 to 9.

Testing & Training Images

Calendar

Description automatically generatedText

Description automatically generated Before any code was written, a method of selecting training and testing images had to be established. We settled on picking out random fonts of numbers using Microsoft Word. To try to keep it as random as possible, we went down the list of font names in alphabetical order and selected the first font from each letter group. Once we had ten random fonts, we simply split the list in half. After we had the training set and test set, we printed out the numbers and took individual pictures of each number. Each photo was labeled and saved accordingly. Below are the fonts we originally used, the first 5 rows are the training set, and the second 5 rows are the test set.

After we built the program and tested the original test and training sets, we did create more training images to test the program farther. The other training numbers will be discussed later in the report.

Preprocessing

The best way to show how the program works is to post screenshots of the image after each preprocessing step or stage, along with screenshots of the MATLAB code. To see the code in full, along with all the functions, please refer to page ##.

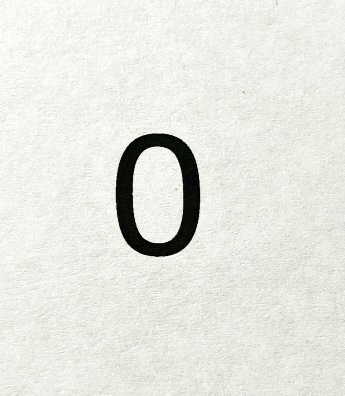
The image used as the example image is shown to the right and is labeled Figure 1. This image is one of our training images we used to help program our project. As one can see, it is a photo of a zero on a white piece of paper. The photo was taken on an iPhone SE gen 3. The surrounding environment was not changed or altered in the process of taking the photo. The photo was simply taken in a well-lit room with a very low spec’ ed iPhone.

Figure 1

Text, letter

Description automatically generated Below is a screenshot of the preprocessing part of the code. As one can see, there are 6 individual stages that the image will go through before the MATLAB code even tries to determine what number is on the image.

The first stage converts the image from having 3 color bands into one color band. This is done using the “rgb2gray” MATLAB command. This is a crucial step in the preprocessing stages. Figure 2 shows what happens to the original image (Figure 1) after the rgb2gray command.

A picture containing text

Description automatically generatedEssentially what the “rgb2gray” command does is create a gray scale version of the original image. If one looks closely, they can spot the difference between figure 1 and figure 2. Figure 1’s background has a slight tan hew while figure 2’s background is strictly gray.

Figure 2

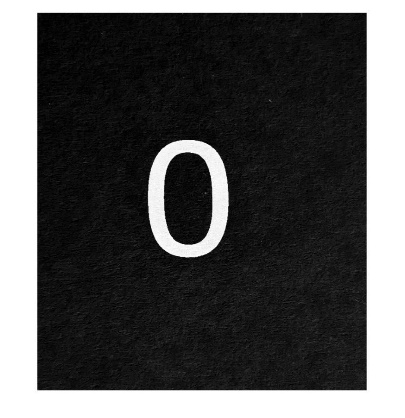
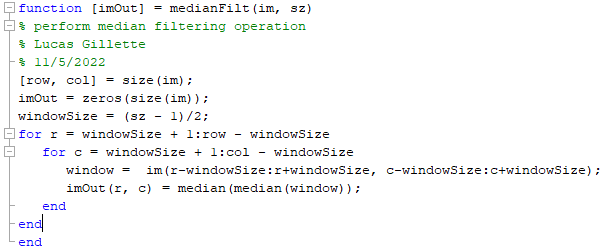
 The second stage is very simple, it takes the rgb2gray image and inverts the image. It does this by subtracting 255 by each pixel’s value. The reason 255 is used is because that’s the highest value an individual 8-bit pixel can have. Inverting the image is required because the larger the pixel’s value, the whiter the pixel is. The smaller the pixel’s value, the darker the pixel is. The pixels values will make sense later on.

Figure 3

 The third stage has two steps in one command. The first step is to send the image into a function called medianFilt. Below is a screenshot of the function.

The medianFilt function’s purpose is to smooth out the image and get rid of any random white spots in the background of the inverted image. The cast command converts the image into a ‘unit8’ class. Figure 4 displays the image after being run through stage 3.

Icon

Description automatically generatedComparing figure 4 to figure 3, one can see some significant changes. It almost looks like the image was converted into a binary image, but it was not. Each pixel still has a value range of 0 to 255.

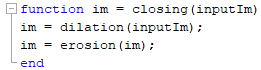
Figure 4

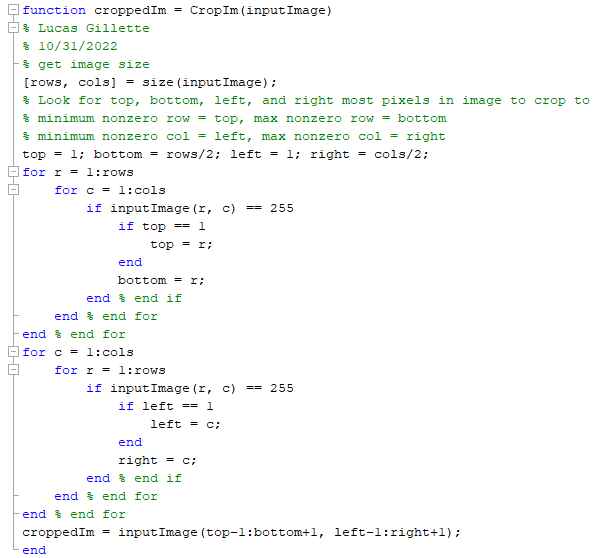
Text

Description automatically generated Stage 4 converts the newly filtered image into a binary image. A hysteresis threshold function was created to complete this task. Below is the screenshot of the function.

How a hysteresis threshold works is, any pixel with a value larger than the upper threshold gets a new value of 255 assigned to it. Then the surrounding pixels are examined using the 8-connectivity concept. If any of the surrounding pixels have a number higher than the lower threshold, that pixel is also assigned a value of 255. For our project, the upper threshold is set to 200 and the lower threshold is set to 100.

There is no noticeable difference between the image before and after the hysteresisThresh function, but the new pixel values are now only 0 or 255. The value 255 is the equivalent of a value of a binary 1.

Stage 5 is a simple closing function. The function first uses the dilation MATLAB command on the image and then does the erosion MATLAB command on the dilated image. The purpose of this stage is to close spots missed by the hysteresisThresh function. Once again, one cannot see the differences between the image before or after the closing function. Below is a screenshot of the closing function code.

The 6th and final stage of the preprocessing is the cropping stage. The purpose of the cropping stage is to get rid of all the unnecessary information that is part of the background and only keep the number in the image. Below is a screenshot of the CropIm function.

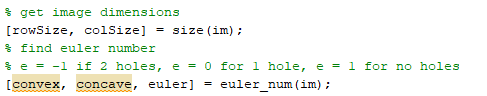
Logo

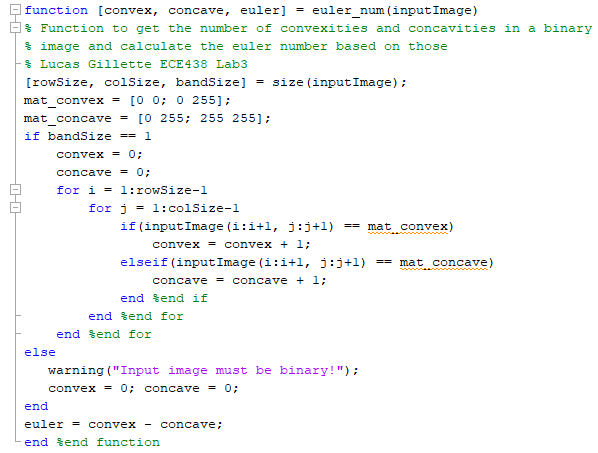
Description automatically generatedHow the cropping code works is, it starts scanning down the rows of the image. Once the code finds a pixel with the value of 255, that row number is saved to a variable labeled ‘top’. The code keeps scanning down the rows and is now writing the row value into the variable labeled ‘bottom’. It keeps overwriting the bottom variable with the new row value until the code no longer sees a pixel with the value of 225. To get the left and right sides of the cropped image, the process is the same, except it scans down the columns from left to right. Once the top, bottom, left, and right side of the character are found, the program then extends the outer bounds of the cropped image out by one row and one column on each side. Figure 5 is the example image after being cropped.

Figure 5

Characterization Processing

The characterization process lies heavily on the Euler number of each character. The Euler number’s definition is the total number of objects in an image minus the total number of holes in those objects. For example, the numbers 0, 4, 6 and 9 all have a Euler number of 0. Numbers 1, 2, 3, 5, and 7 all have a Euler number of 1. The number 8 is the only number that has a Euler number of -1. The characterization process can be broken down into 3 main sections. Each section starts by looking at the images Euler number and goes from there.

 The very first part of the characterization process is not actually finding the Euler number, it is sizing the image after it’s been cropped by the preprocessing part of the code. Then it is finding the Euler number. Below is a screenshot of the code.

 As one can see from the screenshot the Euler number is actually calculated in a separate function called euler\_num. The euler\_num function is screenshot below.

This function first counts the amount of convexities and concavities patterns and subtracts the two. A convex pattern looks like ] and a concavity pattern looks like ]. The subtraction of convexities minus the concavities equals the Euler number.