**Number Characteristic Recognition**

Course: ECE 438, Digital Image Processing

By: Lucas Gillette & Chad Robke

Due Date: 12/15/2022

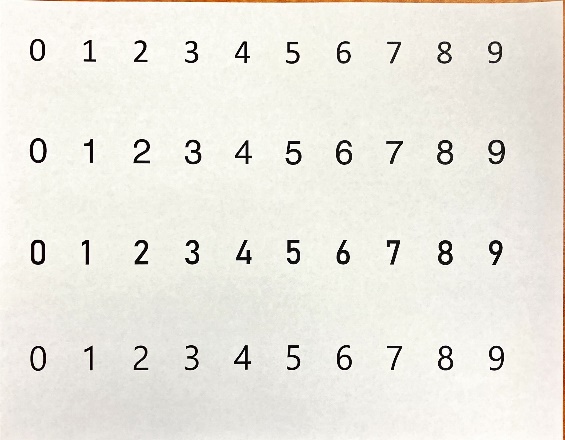
Project Outline

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 ##.

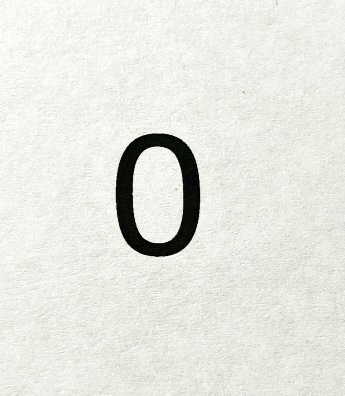
The image used as the example image is shown to the right and is labeled Figure 1. This image is one of our training images we used to help program our project. As one can see, it is a photo of a zero on a white piece of paper. The photo was taken on an iPhone SE gen 3. The surrounding environment was 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 hue while figure 2’s background is strictly gray.

Figure 2

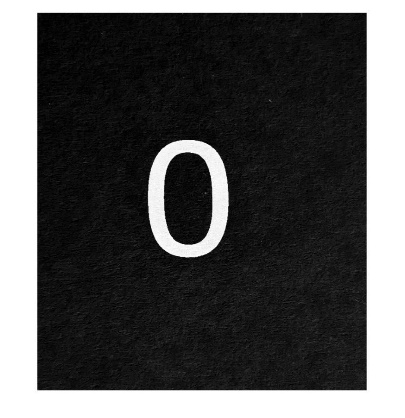
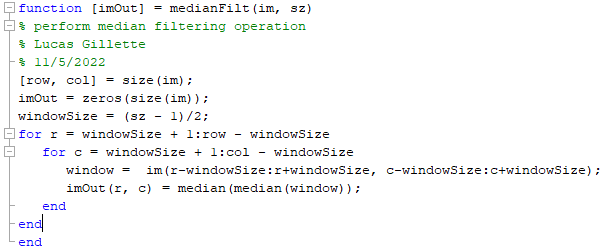
 The second stage is very simple, it takes the rgb2gray image and inverts the image. It does this by subtracting 255 by each pixel’s value. The reason 255 is used is because that’s the highest value an individual 8-bit pixel can have. Inverting the image is required because the larger the pixel’s value, 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 ‘uint8’ 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.

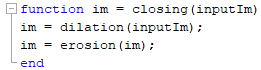
Figure 4

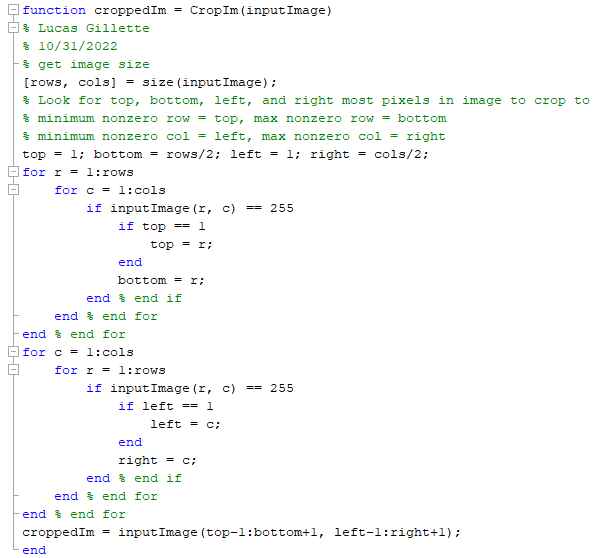
Text

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 pixels surrounding those above the threshold 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 function created in Lab 6 of the class followed by the erosion function which was also created previously in the semester. 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.

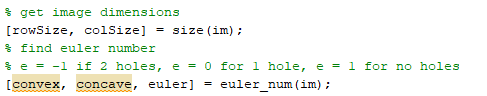
Logo

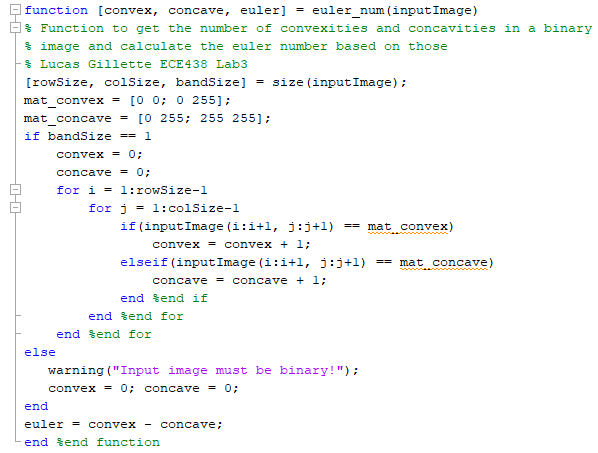
Description automatically generatedHow the cropping code works is, it starts scanning down the rows of the image. Once the code finds a pixel with the value of 255, that row number is saved to a variable labeled ‘top’. The code keeps scanning down the rows and is now writing the row value into the variable 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 across 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

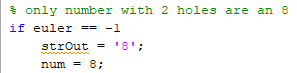
Characterization Processing

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 obtains the new dimensions of the image after cropping before 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, created in Lab 3. The euler\_num function is screenshot below.

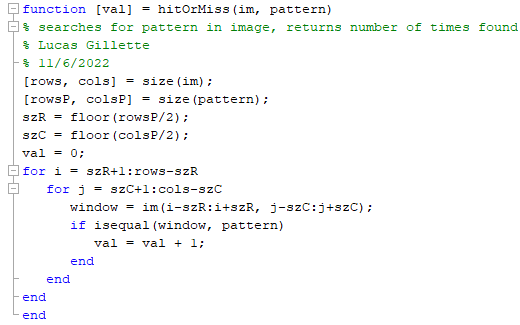
This function first counts the amount of convexities and concavities patterns and subtracts the two. A convex pattern looks like ] and a concavity pattern looks like ]. The subtraction of convexities minus the concavities equals the Euler number.

 Now that the Euler number has been calculated, the code is broken up into three sections. The three sections are based on the Euler number and are implemented by “if” statements. The first “if” statement section is the easiest character recognition process and it’s when the Euler number equals -1.

If the Euler number equals -1, the code writes an “8” to the strOut variable and then the MATLAB Command Window says its and 8.

Text

Description automatically generated Below is a screenshot of the code when the Euler number is equal to zero. When the Euler number equals zero, the code has to then decipher if the character in question is a zero, four, six, or nine.

The code starts out by taking the preprocessed image and breaking it up into 3 parts. The variable imBot is the bottom half of the image. The variable imTop is the top half of the image and the variable imMid is the middle third of the image. After the image is separated into 3 different variables, a new variable is created called “pattern”. Pattern is the key to the success of the characterization code, and is used in a hit-or-miss algorithm. First a pattern is created and then that pattern is compared against the 3 separated image variables. The number of times that pattern accurses in the sectioned image is found by a function called hitOrMiss. Below is a screenshot of the hitOrMiss function. The hitOrMiss function saves its count in the variable called “value”.

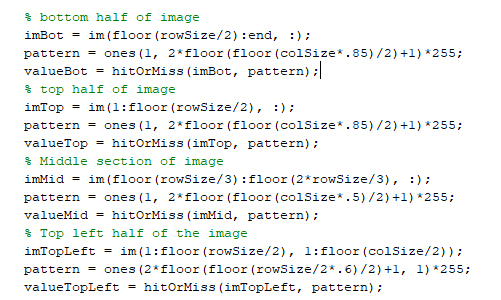
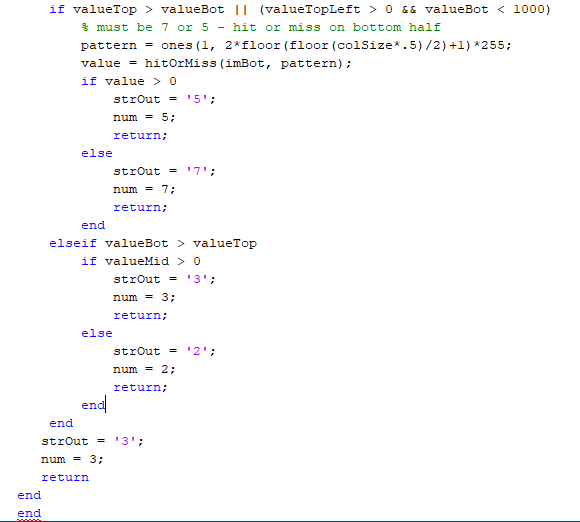
Back to the Euler number equaled to zero code. The pattern is initially set to look for a vertical column of pixels with the value of 255. The height of that vertical column is 65% of half the preprocessed images height. That pattern is then compared to the top right half of the image. If the variable “value” is greater than zero, this means that there is a vertical line in the top right of the image and is not a 6. If there is no vertical line in the top half, the number is likely a 6, but was sometimes confused with 0 when training. To determine if the image number is a 0 or a 6, the pattern is switched to be a horizontal row of pixels with a length of 50% of the preprocessed images size. Then that pattern is used in the hitOrMiss function and scans through the middle part of the preprocessed image. If it finds the pattern, a horizontal component in the center of the number, then the number in the image is a 6, if it does not find the pattern, the number in the image is a 0. If a vertical line is found in the top right half instead, the pattern will still be a vertical line but now it scans the top left half of the image. If both the top right and the top left half of the image have a vertical component, then the character recognition code moves on. If no vertical component is found in the top left of the image, the code determines that the number is a 4. The bottom left of the image is then scanned with the vertical line pattern. If this component is not found, the number is 9. If it is found, the number is a 0.

Text

Description automatically generatedNow that we discussed how the code determines the number on the image if the Euler number is a zero or a negative one, but what if the Euler number isn’t a zero or a negative one? Well to start, we need to determine what other numbers are left to look for. The numbers that are left are 1, 2, 3, 5, and 7. To begin, we will use the same pattern technique as before.

The code creates a vertical pattern that’s 90% the height of the preprocessed image and then scans the preprocessed image using the hitOrMiss function. If the program finds this pattern in the preprocessed image, then the number in the image is a 1.

Once again the code breaks down the preprocessed image into sections. After the image is broken down into sections, another pattern is created and the hitOrMiss function is once again used. This time the hitOrMiss fuction writes its value into a unique variable for each section. A screenshot of the code is below.

So, the four sections of the preprocessed image are the bottom half, the top half, the middle section, and the top left half of the image. The bottom half section’s pattern is a horizontal row that’s 85% of the preprocessed image’s size. The top half section uses the same pattern. The middle section uses a horizontal line that’s 50% of the preprocessed image’s size. Finally, the top left section uses a vertical pattern that is 60% of half the preprocessed image’s size. We are finally at the part of the code that determines the number on the image if the Euler Number is a one or greater. Below is the screenshot.

So if the top half of the image had its pattern accrue more than the bottom half, or the top left half had it’s pattern show up more than once and the bottom half had its pattern show up less than 1000 times, the number in the image is either a 5 or a 7. The top half pattern indicates the top line of a 5 or 7, while the top left vertical line indicates the line in the top left of a 5. If neither of those cases are true, then the number in the image is a 3 or a 2. To determine if the image’s number is a 5 or a 7, a new horizontal pattern is created and scanned through the bottom half of the image to look for the bottom line of a 5 which would not be present in a 7. If the pattern is found, then the image’s number is a 5, if there is no pattern found, then the image contains a 7. Now to determine if the image contains a 2 or a 3, the program looks at the previously mentioned middle section’s pattern, which looks for the center line of a 3. If the middle section found a part of the preprocessed image that contained the pattern, then the image is a 3. If no pattern was found in the middle section, then the image contains a 2.

Short Comings of the Program