

# **Content-Based Image Retrieval Using Barcode**

Group 26

Hanzalla Naveed, Merrill Monteiro, Calla Wilson, Waddah Saleh

### **a) Introduction**

There are a hundred handwritten images provided, with ten images representing each digit of a number (0 to 9). The aim is to retrieve a content-based image using a radon barcode. There is a singular barcode that represents each image, which will then be used to search for other barcodes that are the most similar to it. Then, through many trials, the accuracy of retrieving a similar image will then be acquired.

### **b) Algorithms Explanation**

#### **Barcode Generator**

There are points where the algorithm can diverge depending on if the program is being run for all images or one image. These instances will be separated with different paragraphs.

This is for one image. A text file is opened with the intention to write to it. The chosen image is represented with an array. A function named, “projection” is called. After the projection function is executed, the file is closed.

This is for all 100 images. An empty numpy array is declared with the intention to hold all 100 barcodes of size 162. A for loop that loops ten times is written, the following is in the for loop. An array that holds a class for each iteration of the for loop is declared called folder. This array holds the name of each image name for the given class. A while is written that runs if there is an item in folder. An array is declared to represent the given image which pops items from the folder. The projection function is called for the image that was popped. The for and while loop is exited. The array is dumped to a json file.

This for both, the projection function. One array is declared for each projection, which is of size 28 for horizontal/vertical and size 53 for diagonal. A barcode array is declared for the current image. A variable is declared for the threshold. A for loop within a for loop is written for the range of 28 each. This is to represent incrementing through each pixel in the image for projection 1 and 2. Projection 1 adds all the values in a row together, and projection 2 does the same for the columns. After each nested for loop, the row sum is added to the threshold. The for loop is exited. The threshold is divided by 28 to get the average. A for loop to determine the barcode for projection 1 and 2 is started. There are if statements that compare if the values are higher than the threshold if so, the value becomes a 1. The threshold is multiplied by a value determined in the ideas for improvement section. Exit for loop. A for loop is started for the long diagonal in projection 3 and 4. The diagonal is summed by increment through the for loop. Exit for loop and start a for loop for all the other diagonals. From the long diagonal the problem is broken into two parts for each direction, this is done for each projection. A function named, “diagonal\_projection” is called for each part of the projections. Diagonal\_projection: The length of the diagonal is determined by if statements and the diagonal values are summed using a for loop. The diagonal sum is then returned. Exit for loop, and the average for projection 3 and for is

then calculated. A for loop to determine the barcode for projection 3 and 4 is started. There are if statements that compare if the values are higher than the threshold if so, the value becomes a 1. The threshold is multiplied by a value determined in the ideas for improvement section. Exit for loop. If the program is being run for 1 image the barcode is written to the file. If the program is written for all images the barcode is returned.

### **Search Algorithm**

A variable to represent the hit ratio is declared. A json file is open with the intention to read. An array is loaded with a barcode for each image from a json file. If the program is being run for one search the search function is called and the hit ratio is outputted. If the program is being run for all searches a nested for loop runs through each image and then outputs the hit ratio.

The search function. An array is declared representing the image you want to match. A variable representing most similar image and the lowest hamming distance is declared. Nested for loops run through each barcode to compare them. An if statement makes sure we are not comparing the barcode to itself. A variable is declared for the image we are comparing to and the hamming distance. A for loop is run to compare each bit in the barcodes, If the bits do not match hamming distance is incremented. The for loop is exited. If the lowest hamming distance so far is higher than the current hamming distance, the hamming distance is updated, the location of the barcode in the array is saved. The for loops are exited. The query image name and the search result named is outputted. A 0 or 1 is returned depending on if the comparison was correct.

#### **c) Required Measurements and Analysis**

##### **Barcode Generator Big-O Complexity Analysis: $O(n^2)$**

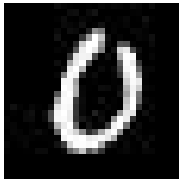
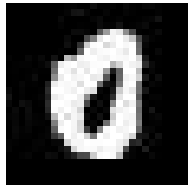
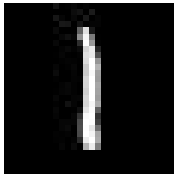
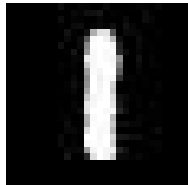
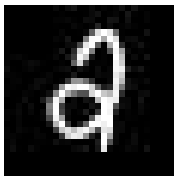
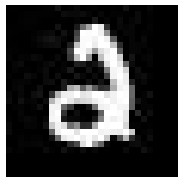


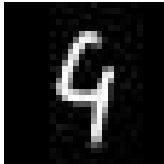
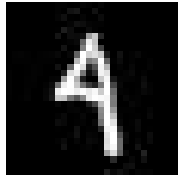

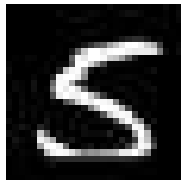
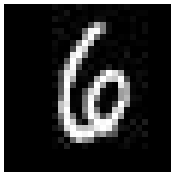
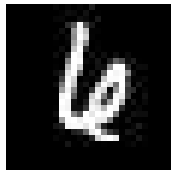


To be able to determine the Big-O complexity of this first algorithm the for loops must be taken into account. As there is only one nested for loop, in the worst case scenario, it would cause the longest delay in the runtime. Therefore we assign that nested for loop the big-O complexity of  $O(n^2)$ .




##### **Search Algorithm Big-O Complexity Analysis: $O(n^3)$**

To be able to determine the Big-O complexity of the second algorithm, the for loops are important. Thus, in order to determine the complexity of this algorithm the for loop which contains two more for loops will lead to a longer runtime in the worst case scenario. Therefore the assigned big-O complexity will be  $O(n^3)$ .

#### **d) Search Results**

Number	Search	Result
--------	--------	--------

0		
1		
2		
3		
4		
5		
6		
7		

8		
9		

Search results for each class.

#### e) Ideas to improve retrieval accuracy

We started with a base retrieval accuracy of 49% with the threshold values being the average of the projection.

We grouped the threshold values for the diagonal projections into one and did the same for horizontal and vertical projections. This allowed us to reduce the number of thresholds to deal with and it seemed reasonable as the grouped projections were similar and had the same length.

Our first idea was to try multiplying and dividing the threshold values by various numbers and see how it affected the retrieval accuracy. This method allowed us to increase the retrieval accuracy to 61% by multiplying the horizontal/vertical threshold by 1.05 and dividing the diagonal threshold by 2.

Vertical/Horizontal Threshold	Diagonal Threshold	Outcome on retrieval accuracy
Average	Average	0.49
Average	Average/2	0.52
Average/2	Average/2	0.44
Average*1.1	Average/2	0.59
Average*1.05	Average/2	0.61
Average*1.06	Average/2	0.58
Average*1.04	Average/2	0.61

Average*1.04	Average/3	0.56
Average*1.04	Average/2.5	0.58
Average*1.04	Average/2.1	0.60
Average*1.04	Average/1.9	0.59

This a list of some of the alterations we made to the threshold values to try to improve the accuracy, and the resulting outcome on the retrieval accuracy.

Our second idea was to write a program that would test multiplying the thresholds by numbers ranging from 0 to 3 with increments of 0.01. The idea behind this was to brute force the multiplier of the thresholds. This method increased our retrieval accuracy to 71%. The program determined the best multiples were 1.26 and 2.54 for horizontal/vertical and diagonal projections, respectively. The program will be provided with the submission named, “pythonProject”, this program took about 20 hours to run.

```
Hit ratio: 0.28
Hit ratio: 0.34
Hit ratio: 0.36
Hit ratio: 0.37
Hit ratio: 0.38
Hit ratio: 0.4
Hit ratio: 0.43
Hit ratio: 0.45
Hit ratio: 0.46
Hit ratio: 0.5
Hit ratio: 0.51
Hit ratio: 0.52
Hit ratio: 0.54
Hit ratio: 0.55
Hit ratio: 0.56
Hit ratio: 0.58
Hit ratio: 0.61
Hit ratio: 0.63
Hit ratio: 0.66
Hit ratio: 0.67
Hit ratio: 0.68
Hit ratio: 0.69
Hit ratio: 0.71
[1.26, 2.54]

Process finished with exit code 0
```

The output from the brute force program.

## f) Conclusion Remarks

Therefore, using radon barcodes to retrieve an image is a very useful method. First, a Barcode Generator algorithm was created that generates a barcode for an image. Then, a Search Algorithm is started that compares the dataset with the chosen barcode to search for a similar image. The image with the lowest hamming distance is selected as the most similar image. The algorithm complexity was analyzed based on Big-O Notation. A nested for loop was assigned to the Barcode Generator Big-O Complexity Analysis of  $O(n^2)$ . The assigned Search Algorithm Big-O complexity analysis was  $O(n^3)$ . The search results were deemed successful as the images were retrieved as expected. Starting with a base retrieval accuracy of 49%, the first idea was to multiply and divide the horizontal/vertical threshold by 1.05 and dividing the diagonal threshold by 2. This allowed for an improved retrieval accuracy of 61%. Also, by multiplying the thresholds by numbers ranging from 0 to 3 with increments of 0.01, a retrieval accuracy of 71% was acquired. This can be concluded to be a great improvement from 49%. Thus, content-based image retrieval using a barcode was a success.