Content-Based Image Retrieval(CBIR) Using Barcode

Date:13th April,2022

Authors (Student ID):
Syed Airaj Hussain (100789134)
Nived Leju Ramachandran Sonia (100782317)
Prince Lucky-Worluh(100791723)
Aarez Ansari (100787149)

Introduction

The Content-Based Image Retrieval Using Barcode, is a structure that handles an image from a given database and searches for the closest image.

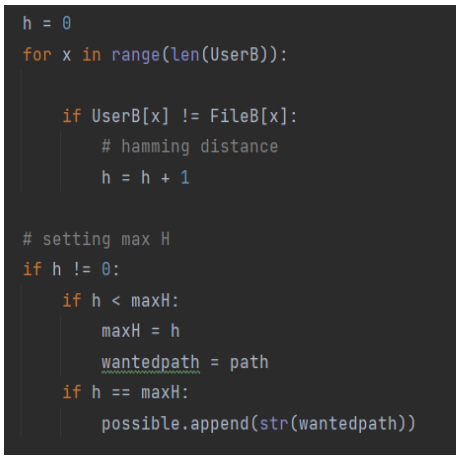
Briefly, within this paper it is demonstrated by using a image database (MNIST) holding 10 images of each digit from 0 through 9, and later finding the binarization of the images through the use of projections, and is compared to other images using their binary and comparing the two using their hamming distance(s).

1. **Algorithms Explained**
   - **Barcode Generator:**
     - The barcode generator works using these following steps:
     - Initialize directory
     - Allocate the image database to variable “classes”
     - Allocate specific image folders to variable “images_path”
     - Allocate unique image path to variable “path”
     - Images are then stored into the array
     - Using the four basic projection angles (0,45,90 and 135) store these angles with unique variables
     - For each projection the average of the vector components are stored in 4 respective variables
     - Using a for loop and an if statement within, the projections are stored into variables to form the total code fragments
     - All the code fragments combine to create barcode
     - The barcodes are stored in file “Barcode.txt” and the respective addresses are stored in “Address.txt”

   - **Search Algorithm:**
     The search algorithm works using the following steps:
     - Set a temporary variable called “h” to store hamming distance and initialize as 0
     - Using for loop in range from 0 to the length of array storing the barcode, compare digits with original image barcode to find hamming distance
     - “h” is incremented if digits at an index is different when compared
     - If hamming distance is lower than previous hamming distance (starting max hamming is preset to 100), the new hamming distance is set as max

- If the hamming distance is 0, the image is not considered as it is most likely the same image as the original
- The path of the image with new lowest hamming distance is stored
- Repeat all of the above steps till the the entire database has been searched and output the path stored for the image with the lowest hamming distance would be the closest matching
- If there are hamming distances that match the lowest hamming distance, the previous lowest is stored in an array and the new max is set as the current image

Fig 1. Search Execution Code

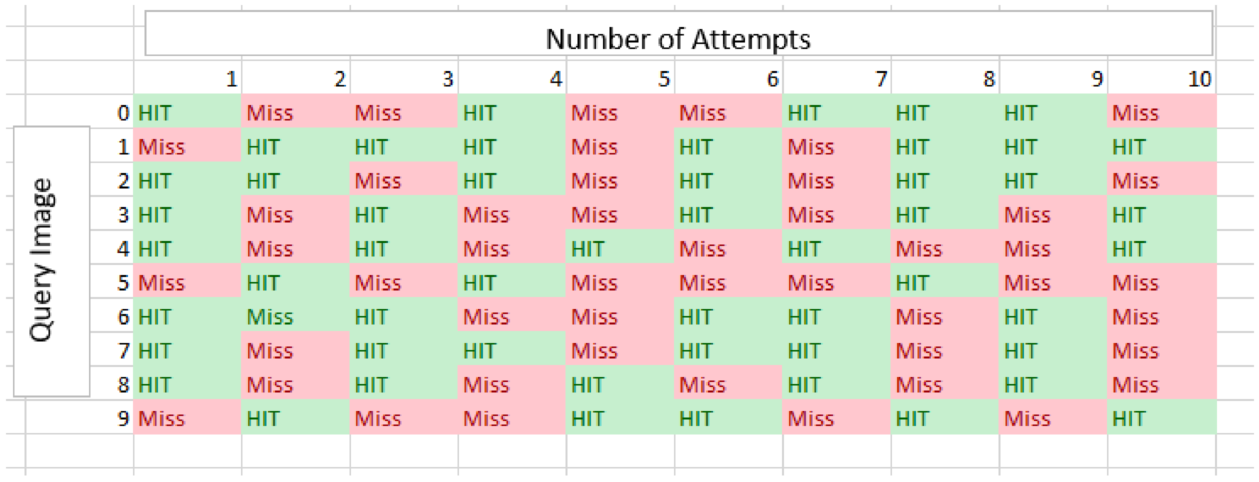


```
h = 0
for x in range(len(UserB)):

    if UserB[x] != FileB[x]:
        # hamming distance
        h = h + 1

# setting max H
if h != 0:
    if h < maxH:
        maxH = h
        wantedpath = path
    if h == maxH:
        possible.append(str(wantedpath))
```

2. Measurements and Analysis

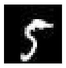
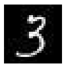
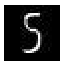
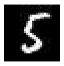
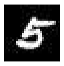
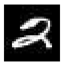
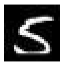
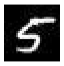
It was observed that after running the program 10 times for each query image folder, that there were a large amount of misses within the image 5 folder, within Fig 2 , at row 5 there is a HIT accuracy of 30% . It was noticed that mostly with the image 5 in Fig 2.1 , the program would recognize the closest image to it as either a 3 or 2

Figure 2. Comparing Query Image to Found Image



		Number of Attempts									
		1	2	3	4	5	6	7	8	9	10
Query Image	0	HIT	Miss	Miss	HIT	Miss	Miss	HIT	HIT	HIT	Miss
	1	Miss	HIT	HIT	HIT	Miss	HIT	Miss	HIT	HIT	HIT
	2	HIT	HIT	Miss	HIT	Miss	HIT	Miss	HIT	HIT	Miss
	3	HIT	Miss	HIT	Miss	Miss	HIT	Miss	HIT	Miss	HIT
	4	HIT	Miss	HIT	Miss	HIT	Miss	HIT	Miss	Miss	HIT
	5	Miss	HIT	Miss	HIT	Miss	Miss	Miss	HIT	Miss	Miss
	6	HIT	Miss	HIT	Miss	Miss	HIT	HIT	Miss	HIT	Miss
	7	HIT	Miss	HIT	HIT	Miss	HIT	HIT	Miss	HIT	Miss
	8	HIT	Miss	HIT	Miss	HIT	Miss	HIT	Miss	HIT	Miss
	9	Miss	HIT	Miss	Miss	HIT	HIT	Miss	HIT	Miss	HIT

Figure 2.1 Comparing Query Image (5) to Found Image

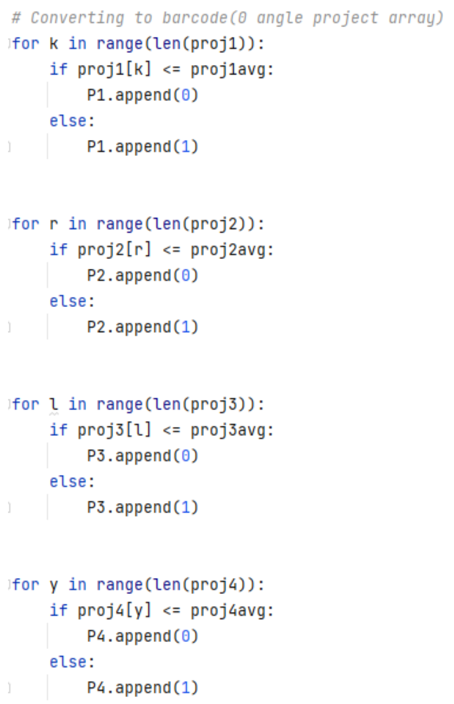
Current Attempt(s)	**Inserted Image**	**Most identical image**
1		
2		
3		
4		

Big O Analysis

The Barcode Generation Algorithm has a complexity of O(n^3). This is because the code first loops through the folders, then through each image, and finally loops to get each projection. n is dependent on the size of the image.

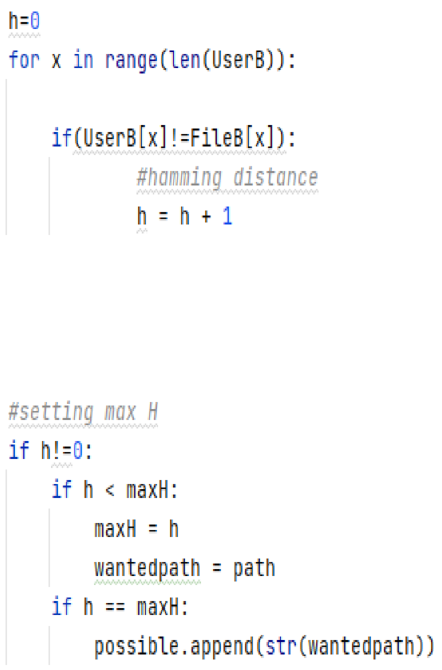
The Search Algorithm has Linear complexity as it has no nested for loops and increases by length of array which depends on the size of the image. Therefore, the Big O Complexity is O(n)

Barcode Generation algorithm Snippet



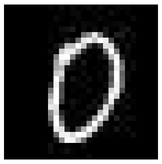
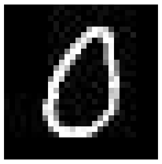
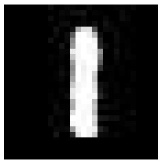
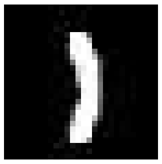
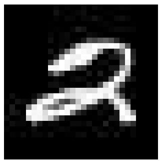
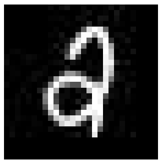
```
# Converting to barcode(0 angle project array)
for k in range(len(proj1)):
    if proj1[k] <= proj1avg:
        P1.append(0)
    else:
        P1.append(1)


for r in range(len(proj2)):
    if proj2[r] <= proj2avg:
        P2.append(0)
    else:
        P2.append(1)


for l in range(len(proj3)):
    if proj3[l] <= proj3avg:
        P3.append(0)
    else:
        P3.append(1)


for y in range(len(proj4)):
    if proj4[y] <= proj4avg:
        P4.append(0)
    else:
        P4.append(1)
```

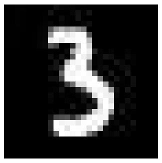
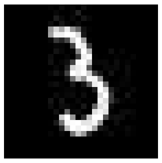
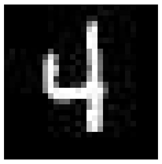
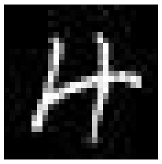
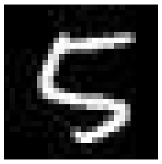
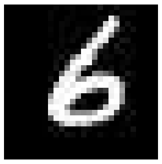
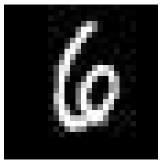
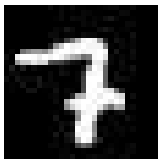
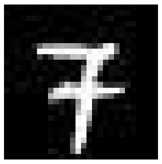
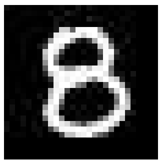
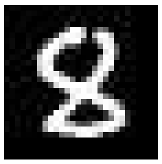
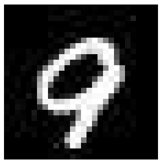
Search Algorithm



```
h=0
for x in range(len(UserB)):

    if(UserB[x]!=FileB[x]):
            #hamming distance
            h = h + 1




#setting max H
if h!=0:
    if h < maxH:
        maxH = h
        wantedpath = path
    if h == maxH:
        possible.append(str(wantedpath))
```

3. Search Results

Image folder	Inserted Image	Most identical image
0		
1		
2		

3		
4		
5		
6		
7		
8		
9		

4. **Ideas the group tried to improve the retrieval accuracy**
    - Increase the number of projection angles from 4 to a multiple of 4 to increase the accuracy of the binarization. This would provide a more precise barcode generation which can then be compared to the images in the database to have more accurate results.
    - Increase resolution of image from 28*28 to a much higher size to improve the accuracy of the resulting barcode which would provide more accurate results. This would increase the Big O complexity of the program as well but would have a greater impact on the retrieval accuracy
    - Adding more images to the database would provide a larger accuracy as more images are likely to be closer to the original.
    - Allowing the user to input the image directly instead of the path would reduce the space required to store the txt file that stores the query image path. This is a minor improvement but an improvement to the efficiency regardless

Conclusion

To wrap up this project, we have proven that the Barcode Generation and Searching algorithms are functional. The accuracy rate of the searches are at 53% with the current structure of the algorithms; mainly due to some images in the database looking the same. The algorithms can be improved by increasing the number of projections, increasing the number of images in the database and asking for higher resolution images for input and to compare against. The Barcode Generation algorithm has a Big O complexity of O(n^3), while the Searching algorithm has a Big O complexity of O(n). This project allowed us to creatively explore solutions to content-based retrieval problems that have real world applications.