Created By:
Daniel LoPresti 100748818
Juan Gaviria 100738545
Prof. Shahryar Rahnamayan, PEng
Data Structures Course Project
OntarioTech University
April 11, 2021

Note: This code was written in a linux virtual machine thus it won't be able to run without errors on windows due to linux file system using '/' instead of '\'

A) Algorithm Explanation

barcode-gen.py:
The main purpose of the barcode generator is to take a path input of an image(MNIST_DS/x/xy.jpg), open that image, and create a barcode based on certain aspects of the image. When the program opens an image at the given path, it converts the image to grayscale, and then inputs the value of each pixel into a *numpy* array. Each pixel is assigned a value of 0 - 255 (0 = black, 255 = white). We then use the projection function that takes the sum of the values depending on the angle. We then calculate the threshold value, which is the average value of projections, and then we change each value depending whether it is above or below the threshold (above = 1, below = 0). We do this for each projection, and then we concatenate each string into a larger string in the projection format of 0,90,45,135. We then use a barcode generator class that takes this concatenated string, and outputs a barcode with a similar path to the original image.

search-algo.py:
The job of this algorithm is to take an input file path for a barcode from the user and compare it to every other barcode by calculating the hamming distance between the input barcode and each of the other barcodes. It is able to do this by accessing the updated MNIST_DS folder which now after running the barcode-gen.py algorithm, has the barcode images and by using the pyzbar and cv2 python packages to get data from the images. The algorithm creates a list of objects where the file path, barcode string, and hamming distance for each barcode is stored in said object which is then stored along with the other barcode objects, in the object list. The algorithm then compares all the hamming distances for all the barcodes and prints out the path to the inputted barcode along with the path of the most similar barcode.

B) Measurements and Analysis

C) Search Results

Input Barcode: b_01.jpg
Most Similar Barcode: b_04.jpg

Input Barcode: b_10.jpg
Most Similar Barcode: b_14.jpg

Input Barcode: b_23.jpg
Most Similar Barcode: b_22.jpg

Input Barcode: b_33.jpg
Most Similar Barcode:b_36.jpg

Input Barcode: b_42.jpg
Most Similar Barcode: b_40.jpg

Input Barcode: b_55.jpg
Most Similar Barcode: b_58.jpg

Input Barcode: b_60.jpg
Most Similar Barcode: b_65.jpg

Input Barcode: b_70.jpg
Most Similar Barcode: b_79.jpg

Input Barcode: b_85.jpg
Most Similar Barcode: b_83.jpg

Input Barcode: b_90.jpg
Most Similar Barcode: b_94.jpg

D) Conclusion Remarks

The process of both developing and testing the algorithms showed the potential use of such technologies in real world applications. The end results of comparing input barcodes to the other barcodes in the dataset showed that while results were mostly accurate, there is still room for improving the algorithm as some comparisons were not as accurate as they could have been. The most challenging part was figuring out how to not have issues while comparing barcode strings and calculating the hamming distance. This is because some strings were longer than others but by tweaking some trace values in the barcode generator algorithm, the problem was solved and allowed for more accurate results as well. The original image set proved to be an excellent data set for testing out computer vision algorithms, and it made it easier to determine where

tweaking needed to be done to the algorithms when the accuracy was off by significant amounts. In the end, the results produced by the algorithms showed that they were successful even if further testing and tweaking could have been done to make them more accurate.