CS5330: Project 2: Content-based Image Retrieval

Created by Sida Zhang, last modified on Feb 15, 2022

@Sida Zhang and @Hongyu Wan

Feb 15, 2021

Introduction

This project contains two interfaces including: CLI (Command line interface) and GUI (graphical user interface). Both of the interfaces allow user to find best matches of a the target image from an image dataset (oylmpus.zip). The project teaches us how to match images by accessing every pixels and compare histograms. Given a dataset of images, the goal is to find images in the set with similar content by adapting CBIR (content-based image retrieval): color and texture.

The environment for this project is Windows OS and the IDE is Visual Studio Code. We have also used cmake to make a makefile and an outside source lib, CVui, to build this project. I will now demonstrate each functionalities of the project in below content. Please scroll all the way down to **Extension(Extension0)** part to watch a quick video demonstration on the graphical user interface.

Feel free to post any question or suggestion in this page, and thank you for viewing our Wiki page.

Overall process

The biggest challenge of this project that I have encountered is data conversion from different dimensions of vectors.

The first few tasks (task 1, task2, and task3) were pretty straight forward and the results are identical to the Canvas results.

I used the Sobel Magnitude filter from Project 1 (CS5330: Project 1: Real-time filtering) to calculate the texture histogram for both task4 and task5. The results are kind of different than I expected because of weight averaging between texture and color histograms.

The three extensions that we implemented in this project are GUI executable, Gabor histogram, and HSV histogram.

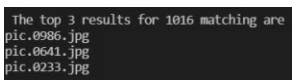
I will illustrate the process and results of each tasks and extensions in the following content now.

Tasks

Task1: Baseline Matching:

For this task, we first looped every image in the dataset and stored the 9x9 color values into the CSV file. Then we read the features from the CSV file and calculate the distance between the target image and each image by using sum-of-square-difference sum((ft - fi) * (ft - fi)). Sort the differences and find the top three closest-to-zero values as the top three matches.

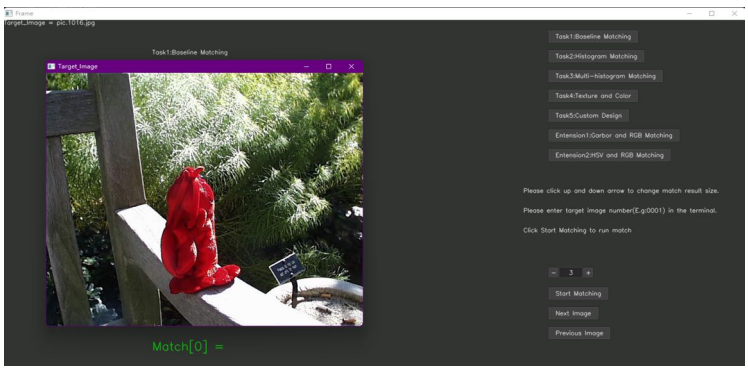
CLI (readfiles.exe): automatically find the top three matches of the target image pic.1016.jpg.



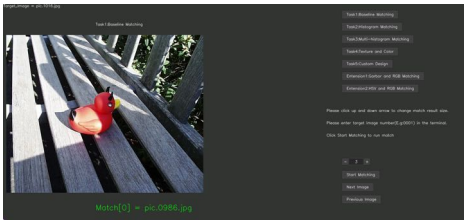
GUI (gui.exe): user will be asked to select which task to run, then the user needs to input the target image number and select how many matches to output.

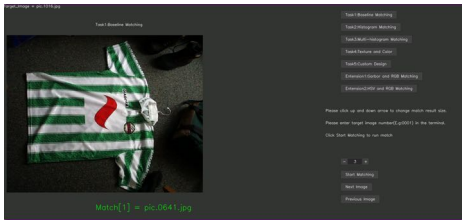
In this case, it would be "task1", "1016" and "3". Click "Start Match" and the output will be shown on the left section of the window.

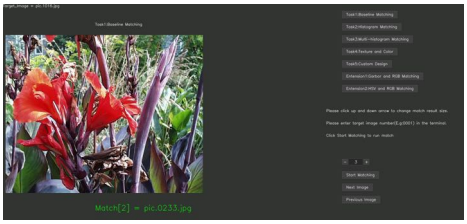
Input:



Output:







Results: pic.0986.jpg, pic.0641.jpg, and pic.0233.jpg

Task2: Histogram Matching:

For this task, we first looped every image in the dataset and stored the rg chromaticity histogram features into the CSV file. The process was done by calculating the value of rix ((r/(R+B+G+1)) * bins) and gix(((g/(R+B+G+1)) * bins). Self-increase the value of the histogram (or the grid), at position rix, gix. In our case, we had a 2-D histogram with 16 bins. Then we read the data from the CSV file and calculate the distance between the target image and each image by adapting histogram intersection sum(ft - min(ft, fi)). Sort the differences and find the top three closest-to-zero values as the top three matches.

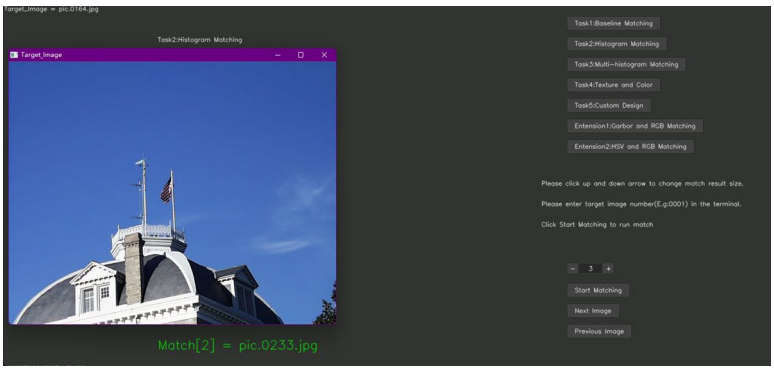
CLI (readfiles.exe): automatically find the top three matches of the target image pic.0164.jpg.

```
The top 3 results for 0164 matching are
pic.0080.jpg
pic.1032.jpg
pic.0461.jpg
```

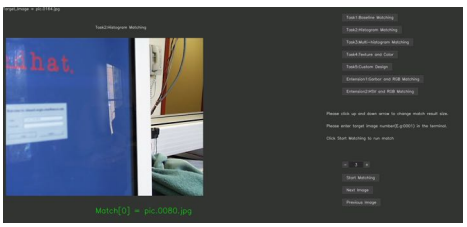
GUI (gui.exe): user will be asked to select which task to run, then the user needs to input the target image number and select how many matches to output.

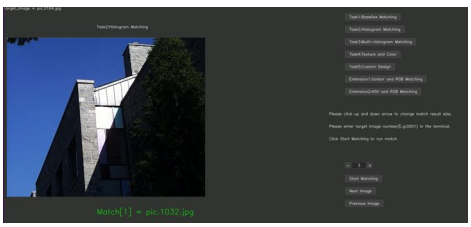
In this case, it would be "task2", "0164" and "3". Click "Start Match" and the output will be shown on the left section of the window.

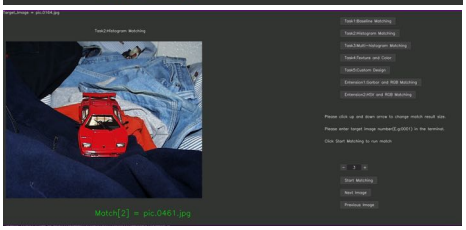
Input:



Output:







Results: pic.0080.jpg, pic.1032.jpg, and pic.0461.jpg

Task3: Multi-histogram Matching:

For this task, we looped every image in the dataset and stored two RGB histogram features into the CSV file. The process was done by calculating the RGB values (R/ buckets) where buckets are 256 divided by the bins, in this case, we have set bin number to 8 to make two 3-D histograms. The value of the histogram self

increases at position, R, G, and B. Then we read the data from the CSV file and calculate the distance between the target image and each image by using histogram intersection sum(ft - min(ft, fi)). Finally we sort the differences and find the top three closest-to-zero values as the top three matches.

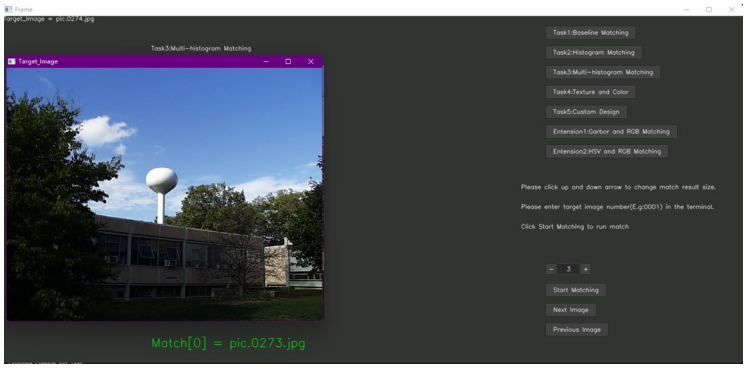
CLI (readfiles.exe): automatically find the top three matches of the target image pic.0274.jpg.

```
The top 3 results for 0274 matching are
pic.0273.jpg
pic.1031.jpg
pic.0409.jpg
```

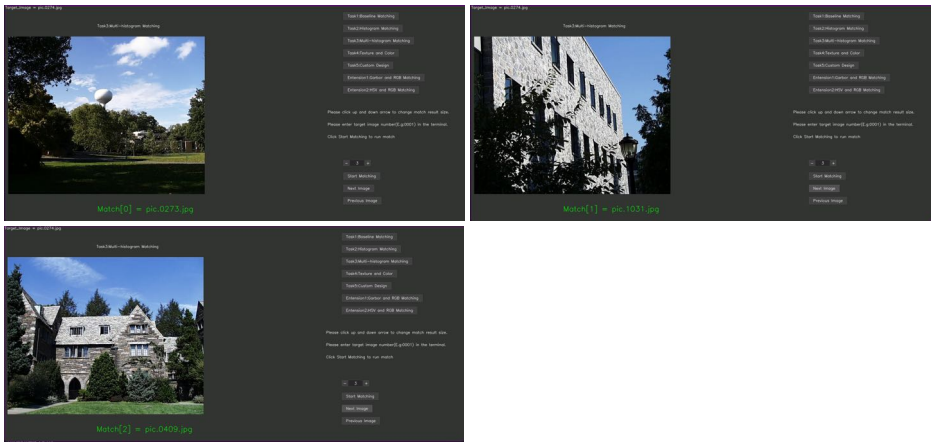
GUI (gui.exe): user will be asked to select which task to run, then the user needs to input the target image number and select how many matches to output.

In this case, it would be "task3", "0274" and "3". Click "Start Match" and the output will be shown on the left section of the window.

Input:



Output:



Results: pic.0273.jpg, pic.1031.jpg, and pic.0409.jpg

Task4: Texture and Color:

For this task, we looped every image in the dataset and stored one texture histogram and one color histogram into two different CSV files. The texture histogram we have used is magnitude (Sobel X and Sobel Y) from project 1, we included filter.h in the header, and inputted the value into a 1-d array to store the histogram data. At first, we had so much difficulty calculating the data because we weren't sure what value to put in. After Dr. Maxwell's explanation, we understood that we need to think of it as one color space since it is grayscale. We successfully did a self-increment at position(gray). Then we also stored the RGB value into a 3-D vector and stored both of the histograms into two separate CSV files. Finally we calculated the distance between the target image and each image by using histogram intersection sum(ft - min(ft, fi)) and outputted the top three closest-to-zero matches.

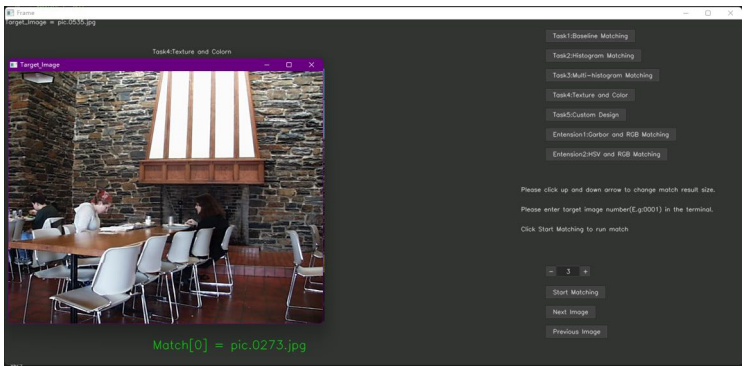
CLI (readfiles.exe): automatically find the top three matches of the target image pic.0535.jpg.

```
The top 3 results for 0535 matching are
pic.0355.jpg
pic.0975.jpg
pic.0628.jpg
```

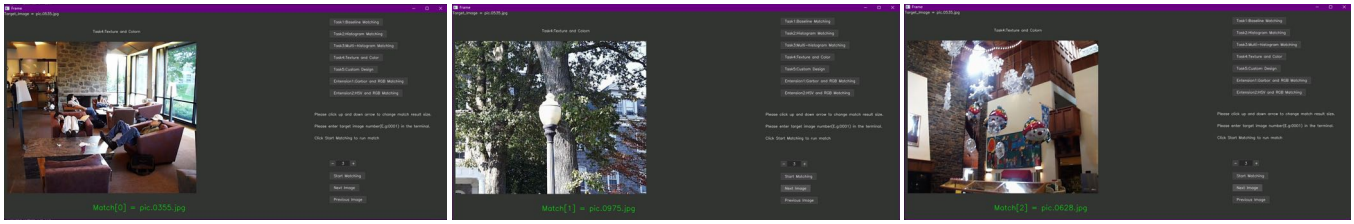
GUI (gui.exe): user will be asked to select which task to run, then the user needs to input the target image number and select how many matches to output.

In this case, it would be "task4", "0535" and "3". Click "Start Match" and the output will be shown on the left section of the window.

Input:



Output:



Results: pic.0355.jpg, pic.0975.jpg, and pic.0628.jpg

Thoughts on Task4: By comparing the processes of Task3 and Task4, we have concluded that two RGB histogarm focus on the intensity and color distribution whereas one texture and one color focus on one area's spatial and intensities.

Task5: Custom Design:

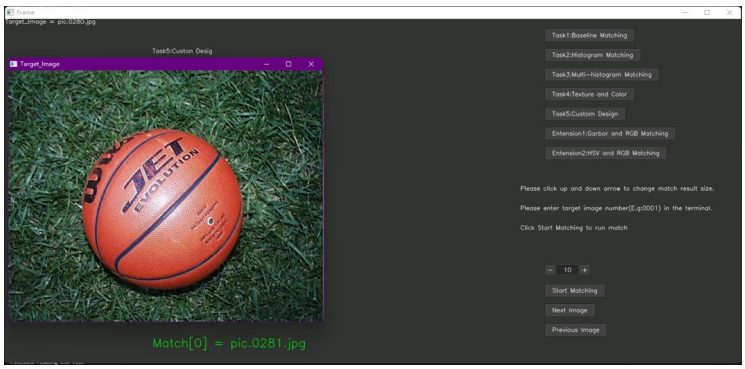
For this task, we created a subset of basketball and a subset of greens. We stored a texture histogram and a RGB histogram into two CSV files for both of the subset. Turns out the results are quite different than what I expected. Then I re-weighted the texture and color histogram to 0.1 to 0.9. The results are much better with more color intensities in this case. We compare two images with both 1-D magnitude histogram and a 3-D RGB histogram. We calculated the distance between the target image and each image by adapting histogram intersection: sum(ft - min(ft, fi)). Finally we sort the differences and find the top ten closest-to-zero values as the top ten matches.

CLI (readfiles.exe): automatically find the top ten matches of the target images pic.0280.jpg (basketball) and pic.0334.jpg (greens)

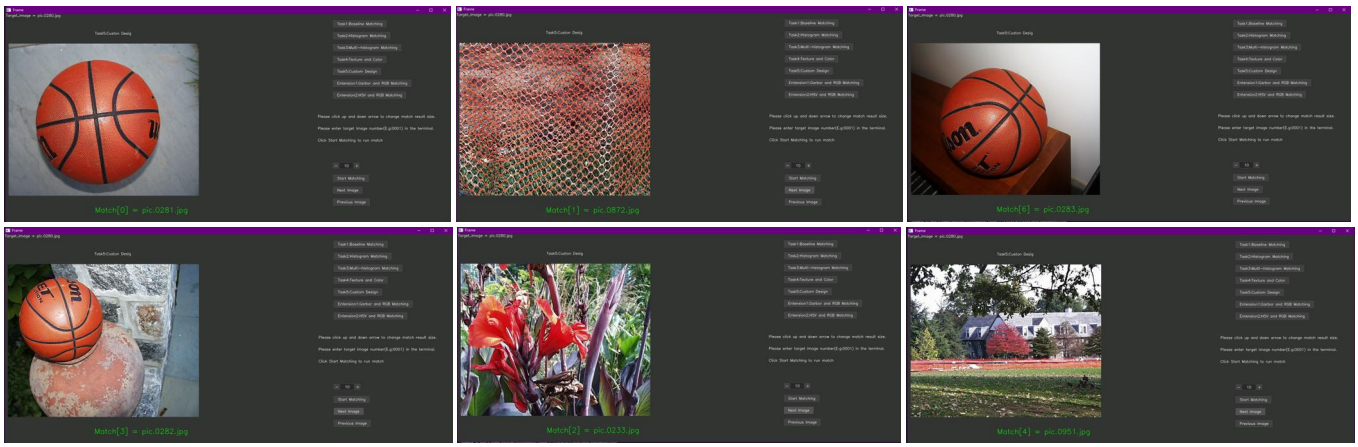
GUI (gui.exe): user will be asked to select which task to run, then the user needs to input the target image number and select how many matches to output.

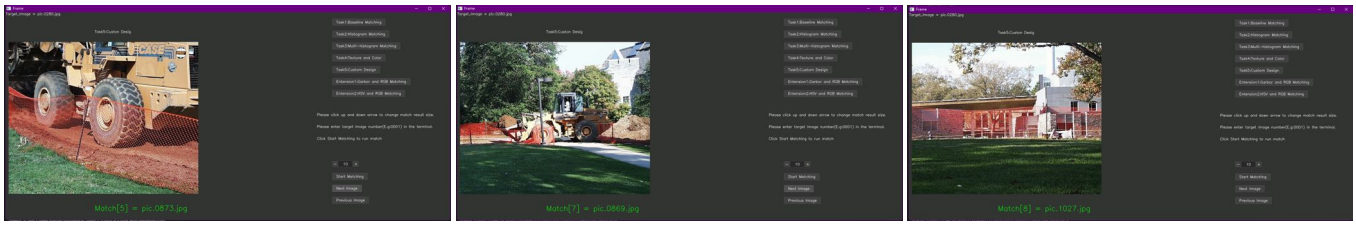
In this case, it would be "task5", "0280" and "10". Click "Start Match" and the output will be shown on the left section of the window.

Input (Basketball):

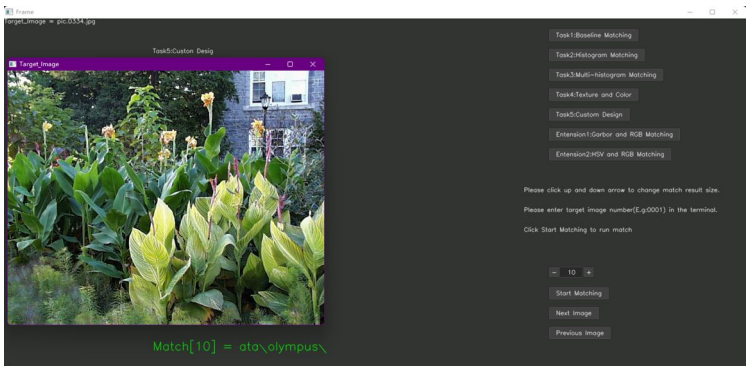


Output:





Input (Greens):



Output:

Results:

Basketball: pic.0281.jpg, pic.0872.jpg, pic.0233.jpg, pic.0282.jpg, pic.0951.jpg, pic.0873.jpg, pic.0283.jpg, pic.0869.jpg, pic.1027.jpg, and pic.0117.jpg.

Greens: pic.0405.jpg, pic.0824.jpg, pic.0251.jpg, pic.1068.jpg, pic.0964.jpg, pic.0458.jpg, pic.0272.jpg, pic.0701.jpg, pic.0154.jpg, and pic.0451.jpg.

Extensions

Extension0:

Video Demonstration on GUI:

Project-2-GUI-Demonstration.mp4

We have created a GUI version of the program. To match images with different tasks, user first needs to click on the corresponding task, then input image number (E.g: 0001), then select how many results to show. Finally click on "Start Matching" to start match images. By clicking "Next Image" and "Previous Image", you can see all the matches on the left side of the application.

Extension1:

For this task, we looped every image in the dataset and stored one texture histogram and one color histogram into two different CSV files. The texture histogram we have used is Gabor filter (citation in Knowledgement), we used OpenCV built-in function, getGaborKernelm(sigma, theta, lambd, gamma, psi), for Gabor Filter and input the value into a 1-d array to store the histogram data. we self-increased the value at position(gabor). Then we also stored the RGB value into a 3-D vector and stored both of the histograms into two separate CSV files. Finally we calculated the distance between the target image and each image by using histogram intersection sum(ft - min(ft, fi)) and outputted the top three closest-to-zero matches.

CLI (readfiles.exe): automatically find the top three matches of the target image pic.0535.jpg.

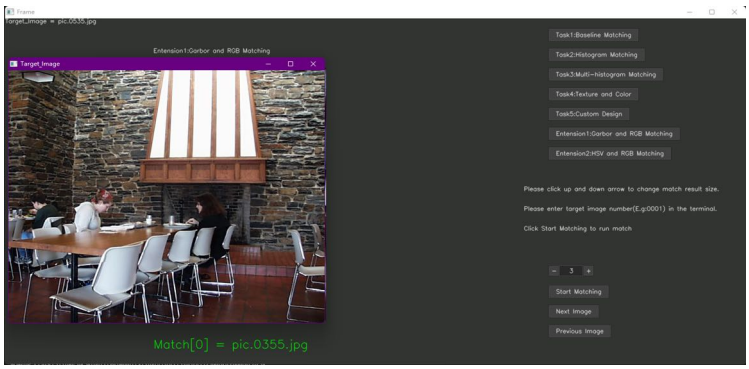
```
The top 3 results for 0535 matching are

pic.0355.jpg
pic.0739.jpg
pic.0554.jpg
```

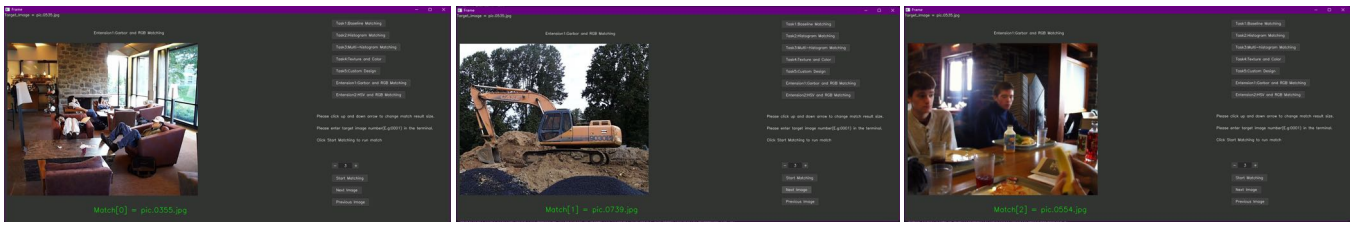
GUI (gui.exe): user will be asked to select which task to run, then the user needs to input the target image number and select how many matches to output.

In this case, it would be "extension1", "0535" and "3". Click "Start Match" and the output will be shown on the left section of the window.

Input:



Outputs:



Results: pic.0355.jpg, pic.0739.jpg, and pic.0554.jpg

Thoughts on extension1: this extension has very simlar results as task 4 since both of the histograms weighted equally.

Extension2:

For this task, we looped every image in the dataset and stored one HSV histogram and one RGB histogram into two different CSV files. We converted the value from RGB to HSV by using OpenCV built-in function, cvtColor(RGB, HSV, cv::COLOR_RGB2HSV), and stored the value into a 3-D histogram. Then we also stored the RGB value into a 3-D vector and stored both of the histograms into two separate CSV files. Finally we calculated the distance between the target image and each image by using histogram intersection sum(ft - min(ft, fi)) and outputted the top three closest-to-zero matches.

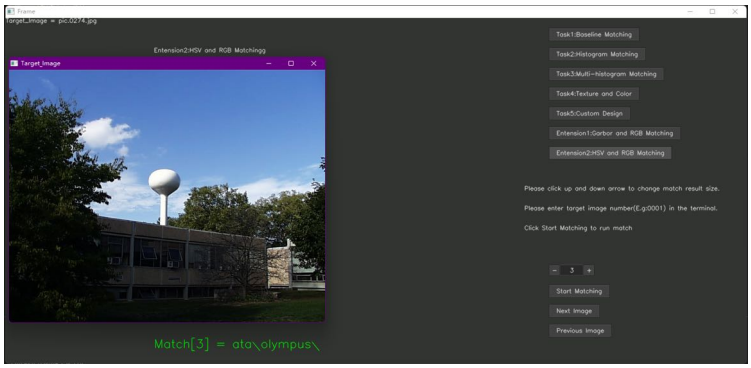
CLI (readfiles.exe): automatically find the top three matches of the target image pic.0274.jpg.

```
The top 3 results for 0274 matching are
pic.0273.jpg
pic.0409.jpg
pic.0213.jpg
```

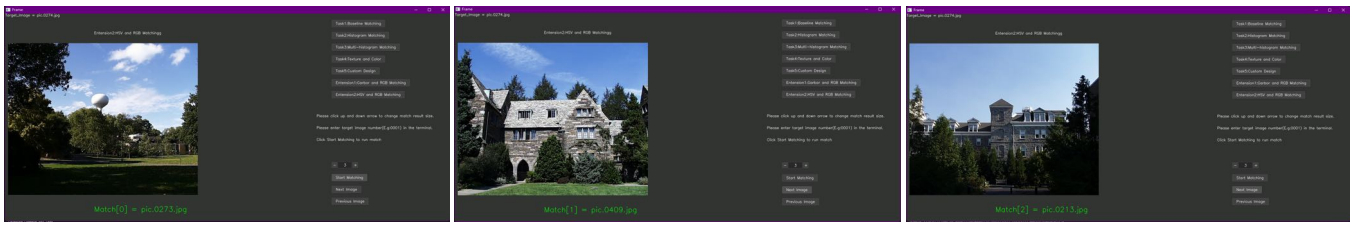
GUI (gui.exe): user will be asked to select which task to run, then the user needs to input the target image number and select how many matches to output.

In this case, it would be "extension2", "0274" and "3". Click "Start Match" and the output will be shown on the left section of the window.

Input:



Outputs:



Results: pic.0355.jpg, pic.0739.jpg, and pic.0554.jpg

Thoughts on extension2: we have equally weighted the distance between the different histograms in this extension and the histogram. The result of this HSV and RGB histogram are very similar to RGB+RGB histogram.

Project Reflection:

Compared to the previous project, this project was a lot easier. And I think it is because we have understood the concept of the matching mechanism.

Overall, I believe the purpose of this project is very important because it makes us familiar with the processing and manipulating images at a pixel level through matching and pattern recognition. It was very necessary to have different feature and distance metrics in order to compare their different functionalities. I think the project was very comprehensive for the computer vision field.

Acknowledgement:

Thanks to Dr. Maxwell and the Piazza posts and discussions.

Data conversion documents from Stackoverflow and OpenCV.

OpenCV tutorial:

https://docs.opencv.org/4.5.2/index.html

CVui:

https://github.com/Dovyski/cvui

https://dovyski.github.io/cvui/components/sparkline/

Gabor:

https://docs.opencv.org/4.5.2/d4/d86/group__imgproc__filter.html#gae84c92d248183bd92fa713ce51cc3599

https://en.wikipedia.org/wiki/Gabor_filter

HSV:

https://docs.opencv.org/4.5.2/de/d25/imgproc_color_conversions.html

https://docs.opencv.org/4.5.2/d8/d01/group__imgproc__color__conversions.html#gga4e0972be5de079fed4e3a10e24ef5ef0aa4a7f0ecf2e94150699e48c79139ee12

No labels