CS5330: PROJECT 2
CONTENT-BASED IMAGE RETRIEVAL

Team Members:
Name: Ravi Shankar Sankara Narayanan (NUID: 001568628)
Name: Vishaq Jayakumar (NUID: 002737793)

Short description:

This C++ program uses OpenCV to compare a target image with a set of images in a directory using different feature extraction methods (like baseline features, color histogram, multi-histogram, color and texture features, Fourier transform features, co-occurrence matrix features, etc.). It then sorts the images based on their similarity to the target image and prints the filenames of the top N matching images.

Project Outputs:

We have split the project into four parts (four main files).The file and their contents are provided below:

- main.cpp - This file contains code for Tasks 1,2,3,4 and Extension 1 and 2.
- Task5.cpp - This file contains code for Task 5.
- Task7.cpp - This file contains code for Task 7.
- Ext.cpp - This file contains code for the Extension 3.

Our project is structured in the way that the main directory has a 'src' and ' include' folder where the main cpp files are in src and the header files are in include folders. The images provided by the professor were stored in a separate folder called data. And inside the data folder multiple subfolders were created. Each subfolder contained the target and input images for each of these tasks. For executing our program, we used a command line terminal. The command has the following structure:

```
.\patternmatching.exe E:\MSCS\CVPR\projects\project2\data\sample1\target\pic.1016.jpg E:\MSCS\CVPR\projects\project2\data\sample1\input b 5
```

The arguments are explained as follows:

- The first argument is the executable file of our program.
- The second argument is the path to target image.
- The third argument is the directory that holds the input images (which is a subset of the original dataset)
- The fourth argument specifies the matching method.
- The last argument mentions the number of matches that the program must return.

In this case, the program will read the target and input images from their path and directory respectively and use baseline matching method to return the path of 5 images that are most similar to the target image.

The key corresponding to the matching method is given below:

1. Baseline matching: b
2. Histogram matching: h
3. Histogram matching: h2
4. Multi-histogram matching: m
5. Texture and color-based matching: t
6. Texture and color-based matching using co-occurrence matrix(extension): c
7. Texture and color-based matching using Fourier Transform features(extension): f
8. Deep Network Embeddings: d

Task 1 Baseline Matching:

Our program was able to replicate the results mentioned in the project. The result of running the command line in the terminal gave the following results:

```
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug> .\patternmatching.exe E:\MSCS\CVPR\projects\project2\data\sample1\t
arget\pic.1016.jpg E:\MSCS\CVPR\projects\project2\data\sample1\input b 5
E:\MSCS\CVPR\projects\project2\data\sample1\input/pic.0641.jpg
E:\MSCS\CVPR\projects\project2\data\sample1\input/pic.0986.jpg
E:\MSCS\CVPR\projects\project2\data\sample1\input/pic.0547.jpg
E:\MSCS\CVPR\projects\project2\data\sample1\input/pic.0457.jpg
E:\MSCS\CVPR\projects\project2\data\sample1\input/pic.0435.jpg
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug>
```

The images mentioned in the result are attached below:

Target image:

Matched images:

Task 2 Histogram Matching:

We created two separate functions for histogram matching one with 16 bins and another with 8 bins.

For both the histogram matching we used a Euclidean function for measuring the distance between two histograms. The ‘euclideanDistance’ function computes the Euclidean distance between two histograms by summing the squared differences for each bin and taking the square root of the result.

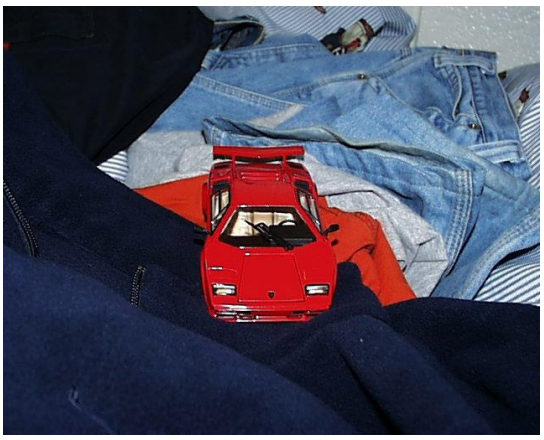
On executing the 16-bin function, we were also able to replicate the results mentioned in canvas:

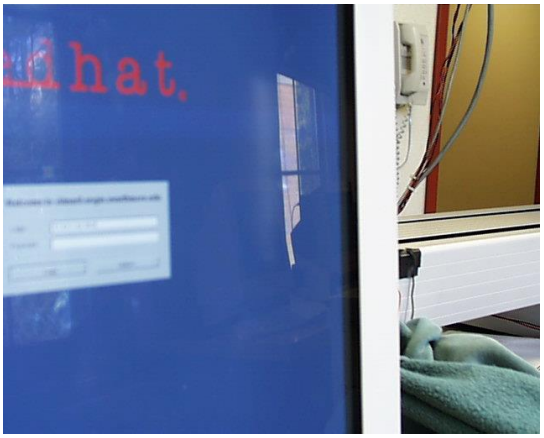
```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS E:\MSCS\CVPR\projects\project2\task1\build\Debug> .\patternmatching.exe E:\MSCS\CVPR\projects\pro
arget\pic.0164.jpg E:\MSCS\CVPR\projects\project2\data\sample2\input h 5
E:\MSCS\CVPR\projects\project2\data\sample2\input/pic.0080.jpg
E:\MSCS\CVPR\projects\project2\data\sample2\input/pic.1032.jpg
E:\MSCS\CVPR\projects\project2\data\sample2\input/pic.0110.jpg
E:\MSCS\CVPR\projects\project2\data\sample2\input/pic.0461.jpg
E:\MSCS\CVPR\projects\project2\data\sample2\input/pic.1013.jpg
```

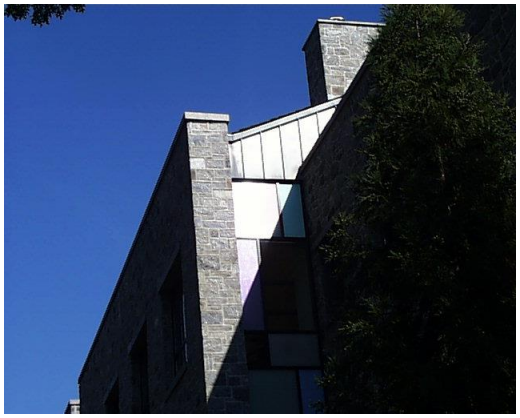
The target and matched images are attached below:

Target image:

Matched images:







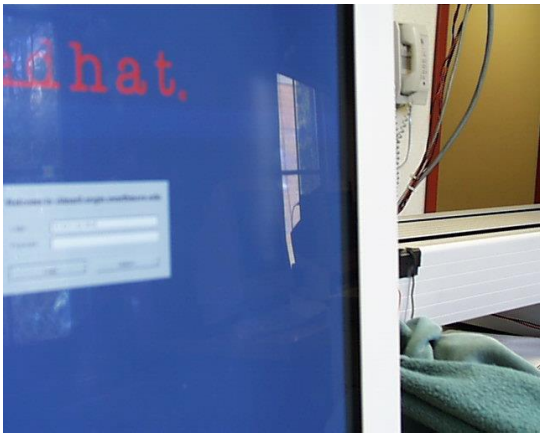
On executing the 8-bin histogram function we were able to replicate the results from the canvas:

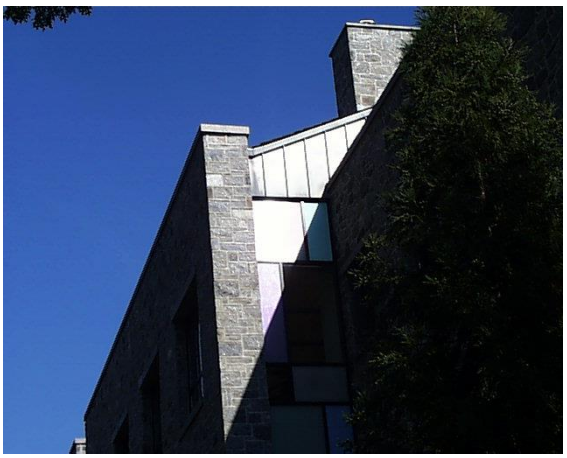
```
Windows PowerShell
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug> .\patternmatching.exe E:\MSCS\CVPR\projects\project2\data\sample3\t
arget\pic.0164.jpg E:\MSCS\CVPR\projects\project2\data\sample3\input h2 5
E:\MSCS\CVPR\projects\project2\data\sample3\input/pic.0461.jpg
E:\MSCS\CVPR\projects\project2\data\sample3\input/pic.0080.jpg
E:\MSCS\CVPR\projects\project2\data\sample3\input/pic.0092.jpg
E:\MSCS\CVPR\projects\project2\data\sample3\input/pic.1032.jpg
E:\MSCS\CVPR\projects\project2\data\sample3\input/pic.0110.jpg
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug>
```

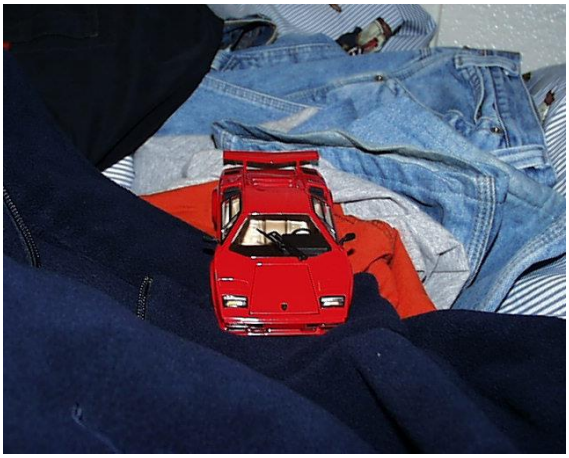
The target and matched images are attached below:

Target image:

Matched Images:







On looking at the results of both versions, we can observe that the results are more or less similar.

Task 3 Multi-histogram Matching:

The ‘computeMultiHistogram’ function in our program calculates the multi-histogram of an image by computing the color histogram for the whole image and its center. The ‘multiHistogramIntersectionDistance’ function calculates the intersection distance between two multi-histograms by summing the intersection distances for each histogram with different weights.

The result of executing this function is attached below:

```
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug> .\patternmatching.exe E:\MSCS\CVPR\projects\project2\data\sample4\target\pic.0274.jpg
 E:\MSCS\CVPR\projects\project2\data\sample4\input m 5
E:\MSCS\CVPR\projects\project2\data\sample4\input/pic.0311.jpg
E:\MSCS\CVPR\projects\project2\data\sample4\input/pic.0304.jpg
E:\MSCS\CVPR\projects\project2\data\sample4\input/pic.1044.jpg
E:\MSCS\CVPR\projects\project2\data\sample4\input/pic.0301.jpg
E:\MSCS\CVPR\projects\project2\data\sample4\input/pic.0303.jpg
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug>
```

The target images and the matched images are given below:

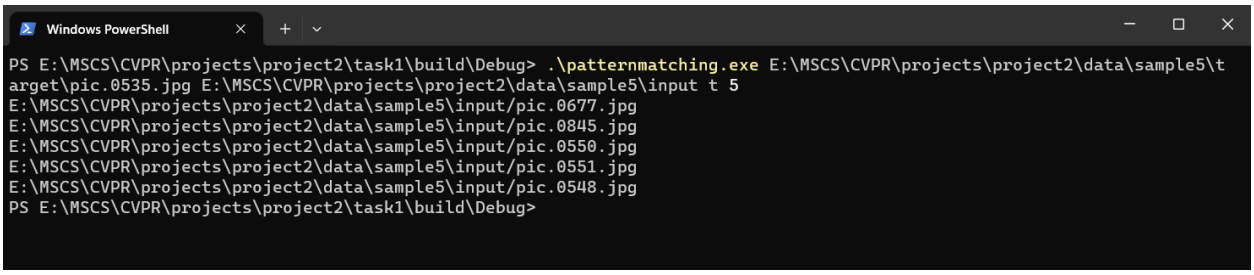
Target Image:

Matched Image:

Task 4 Texture and color matching:

We implemented the Texture and color matching by computing feature vectors for images and then comparing these vectors using a distance function. The feature vector for the image is a concatenation of a color histogram and a texture histogram. The color histogram is computed directly from the image, while the texture histogram is computed from the magnitude of the Sobel gradient of the image, this function was reused from the previous assignment. The Sobel gradient is a measure of the intensity change in the image, which can be used to capture texture information. The distance between two feature vectors is computed using the Euclidean distance. The smaller the distance, the more similar the two images are in terms of color and texture.

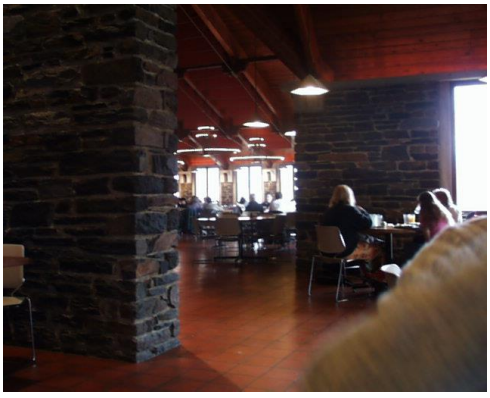
The result of executing this function is attached below:

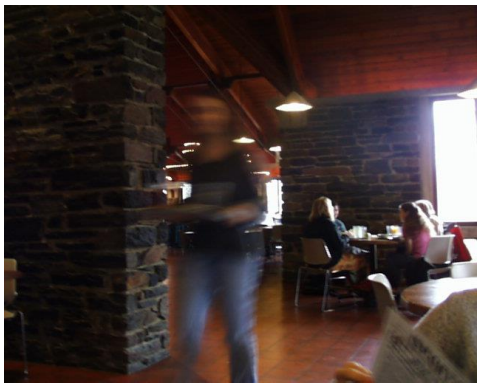


The target and matched images are attached below.

Target Image:

Matched Image:





In all the above matched images, we can see a pattern or color similar to the target image. Especially in the last four images, the program matched these with the target because of the presence of brick wall.

Task 5 Deep Network Embeddings:

The program takes four command-line arguments: the path to a target image, the path to a CSV file containing feature vectors for multiple images, a distance metric ("sum_square" or "cosine"), and an integer N representing the number of top matches to retrieve. The code reads the target image and feature vectors from the CSV file, computes the distance between the target image's feature vector and those of other images using either sum-of-square or cosine distance metrics, and then sorts the images based on their distances. Finally, it outputs the top N matches with their corresponding distances. The feature vectors are represented as 512-dimensional vectors, and OpenCV is used for image and matrix operations throughout the code.

The result of executing the function is given below:

```
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug> .\patternmatching.exe E:\MSCS\CVPR\projects\project2\data\sample6\target\pic.0164.jpg
 E:\MSCS\CVPR\projects\project2\data\sample6\input d 5
Target features read
E:\MSCS\CVPR\projects\project2\data\sample6\input/pic.0426.jpg
E:\MSCS\CVPR\projects\project2\data\sample6\input/pic.0274.jpg
E:\MSCS\CVPR\projects\project2\data\sample6\input/pic.0511.jpg
E:\MSCS\CVPR\projects\project2\data\sample6\input/pic.0275.jpg
E:\MSCS\CVPR\projects\project2\data\sample6\input/pic.0873.jpg
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug>
```

The target and the matched images are given below.

Target image:

The matched images are:

Task 6 Compare DNN Embeddings and Classic Features:

We have taken pic.0002.jpg for comparing the performance of both DNN and matching using conventional features like Texture and Color.

Firstly, the result of using DNN is attached below:

Result:

```
vishaqj@Vishaqs-Laptop build % ./MyProject ../../olympus/pic.0002.jpg ../../ResNet18_olym.csv sum_square 5
Match 1: pic.0002.jpg - Distance: 0
Match 2: pic.0321.jpg - Distance: 183.556
Match 3: pic.0606.jpg - Distance: 186.65
Match 4: pic.0006.jpg - Distance: 190.963
Match 5: pic.0397.jpg - Distance: 193.794
vishaqj@Vishaqs-Laptop build %
```

Target Image:

Matched Images:

.

Keeping the same image as target, we execute the texture and color matching function. The result is attached below.

```
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug> .\patternmatching.exe E:\MSCS\CVPR\projects\project2\data\sample7\target\pic.0002.jpg
 E:\MSCS\CVPR\projects\project2\data\sample7\input t 5
E:\MSCS\CVPR\projects\project2\data\sample7\input/pic.0006.jpg
E:\MSCS\CVPR\projects\project2\data\sample7\input/pic.0044.jpg
E:\MSCS\CVPR\projects\project2\data\sample7\input/pic.0004.jpg
E:\MSCS\CVPR\projects\project2\data\sample7\input/pic.0003.jpg
E:\MSCS\CVPR\projects\project2\data\sample7\input/pic.0065.jpg
```

The matched images are:

From comparing the results of both the methods we can observe that using deep network embeddings proves to be more effective in pattern matching.

Deep learning models, such as the ResNet18 mentioned in Canvas, can learn complex, hierarchical features from data. In the lower layers, these models learn simple features like edges and textures, while in the higher layers, they learn more complex features like shapes and object parts. This allows them to capture more complex patterns and variations in the images.

When trained on a large and diverse dataset like ImageNet, these models learn a rich set of features that are very effective at capturing the content of images. The final average pooling layer of a deep network provides a compact and robust representation of these features (which is provided in the csv file).

On the other hand, conventional features are handcrafted and may not capture all the relevant information in the images. They are often based on simple statistics and do not have the ability to learn complex patterns in the data.

Furthermore, deep learning models can be trained end-to-end, meaning they can learn the best features for the task at hand directly from the data. This contrasts with conventional methods, where the features are fixed and may not be optimal for the specific task.

However, it's important to note that deep learning models require large amounts of data and computational resources to train and can also be more difficult to interpret than conventional methods.

Task 7 Custom Design:

The program takes as input a CSV file containing a list of image file names, reads these images, computes Gabor features for each image, and stores the resulting feature vectors in a collection. The Gabor features are extracted from the grayscale version of each image using a specified set of parameters for the Gabor filter (e.g., kernel size, sigma, theta, lambda, and gamma). Subsequently, the code reads a target image, computes its Gabor features, and compares these features with those of the images from the CSV file using Euclidean distance. The top N closest matches, based on Euclidean distance, are then sorted, and printed, providing a ranked list of similar images.

Result:

```
vishaqj@Vishaqs-Laptop build % ./Part6_2 ../../ResNet18_olym.csv 5
Match 1: pic.0672.jpg - Distance: 0.0205243
Match 2: pic.0427.jpg - Distance: 0.0571985
Match 3: pic.0920.jpg - Distance: 0.0636477
Match 4: pic.0516.jpg - Distance: 0.0831722
Match 5: pic.0031.jpg - Distance: 0.0992076
vishaqj@Vishaqs-Laptop build %
```

Target Image:

Matched Images:

Extensions:

For the first couple of extensions, we added two more texture and color based matching functions based on the instructions from the professor and we added another method to match images based on edge histogram method.

Extension 1: Texture and color-based matching using co-occurrence matrix:

The ‘computeCoOccurrenceMatrixFeatures’ function is designed to extract texture features from an image using a co-occurrence matrix. The function first converts the input image to grayscale to calculate the occurrence matrix. It then computes a histogram of the grayscale image using the ‘calcHist’ function. This histogram serves as the co-occurrence matrix. The function then iterates over the co-occurrence matrix to calculate two features: energy and entropy. Energy is the sum of squared elements in the matrix, and entropy is a measure of randomness or disorder in the image, calculated as the sum of the product of each element and the log2 of each element. The function returns these two features as a vector. These features can be used to compare images based on their texture.

The distance between two feature vectors is computed reusing the Euclidean distance. The smaller the distance, the more similar the two images are in terms of color and texture.

The output of executing this function is provided below:

```
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug> .\patternmatching.exe E:\MSCS\CVPR\projects\project2\data\sample5\target\pic.0535.jpg E:\MSCS\CVPR\projects\project2\data\sample5\input c 5
[ INFO:0@0.016] global registry_parallel.impl.hpp:96 cv::parallel::ParallelBackendRegistry::ParallelBackendRegistry core(parallel): Enabled backends(3, sorted by priority): ONETBB(1000); TBB(990); OPENMP(980)
[ INFO:0@0.016] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16\bin\opencv_core_parallel_onetbb490_64d.dll => FAILED
[ INFO:0@0.017] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_onetbb490_64d.dll => FAILED
[ INFO:0@0.018] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16\bin\opencv_core_parallel_tbb490_64d.dll => FAILED
[ INFO:0@0.019] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_tbb490_64d.dll => FAILED
[ INFO:0@0.019] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16\bin\opencv_core_parallel_openmp490_64d.dll => FAILED
[ INFO:0@0.020] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_openmp490_64d.dll => FAILED
E:\MSCS\CVPR\projects\project2\data\sample5\input/pic.0681.jpg
E:\MSCS\CVPR\projects\project2\data\sample5\input/pic.0844.jpg
E:\MSCS\CVPR\projects\project2\data\sample5\input/pic.0677.jpg
E:\MSCS\CVPR\projects\project2\data\sample5\input/pic.0532.jpg
E:\MSCS\CVPR\projects\project2\data\sample5\input/pic.0726.jpg
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug>
```

The target and matched images are attached below:

Target image:

Matched image:

We can observe that three of the top 5 results have brick structure in them which would have matched with the target image.

Extension 2: Texture and color-based matching using Fourier Transform features:

The 'computeFourierTransformFeatures' function extracts feature from an image using the Fourier Transform. The function first converts the input image to grayscale. It then pads the image to the optimal size for the Discrete Fourier Transform (DFT). The DFT is then applied to the image, resulting in a complex image that contains both magnitude and phase information. The magnitude of the DFT is then computed and normalized to a range between 0 and 1. The magnitude image is then resized to a 16x16 image, effectively reducing the dimensionality of the feature vector. The pixel values of the resized image are then stored in a vector, which is returned by the function. These features can be used to compare images based on their frequency content, which can be useful for texture analysis.

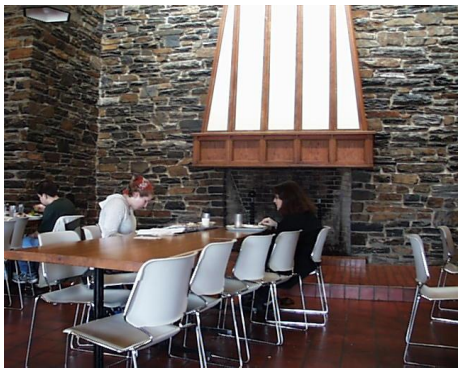
Again, the distance between two feature vectors is computed reusing the Euclidean distance. The smaller the distance, the more similar the two images are in terms of color and texture.

The output of executing this function is provided below:

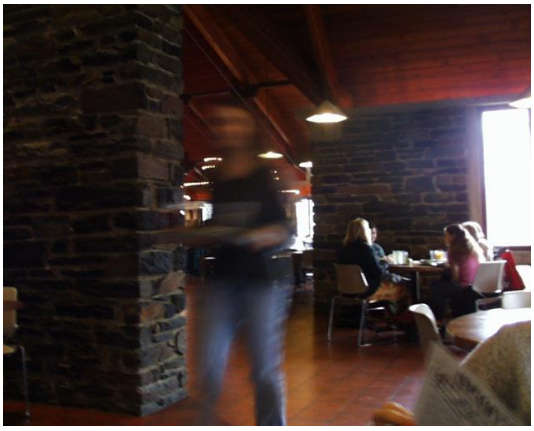
```
PS E:\MSCS\CVPR\projects\project2\task1\build\Debug> .\patternmatching.exe E:\MSCS\CVPR\projects\project2\data\sample5\target\pic.0535.jpg E:\MSCS\CVPR\projects\project2\data\sample5\input f 5
[ INFO:0@0.027] global registry_parallel.impl.hpp:96 cv::parallel::ParallelBackendRegistry::ParallelBackendRegistry core(parallel): Enabled backends(3, sorted by priority): ONETBB(1000); TBB(990); OPENMP(980)
[ INFO:0@0.031] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16\bin\opencv_core_parallel_onetbb490_64d.dll => FAILED
[ INFO:0@0.032] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_onetbb490_64d.dll => FAILED
[ INFO:0@0.032] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16\bin\opencv_core_parallel_tbb490_64d.dll => FAILED
[ INFO:0@0.033] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_tbb490_64d.dll => FAILED
[ INFO:0@0.033] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16\bin\opencv_core_parallel_openmp490_64d.dll => FAILED
[ INFO:0@0.034] global plugin_loader.impl.hpp:67 cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_openmp490_64d.dll => FAILED
E:\MSCS\CVPR\projects\project2\data\sample5\input/pic.0726.jpg
E:\MSCS\CVPR\projects\project2\data\sample5\input/pic.0677.jpg
E:\MSCS\CVPR\projects\project2\data\sample5\input/pic.0570.jpg
E:\MSCS\CVPR\projects\project2\data\sample5\input/pic.0550.jpg
E:\MSCS\CVPR\projects\project2\data\sample5\input/pic.0681.jpg
```

The target and matched images are attached below:

Target Images:



Matched Images:



Observation: Using a Fourier Transform for Texture and color matching proved to be much more precise compared to the other two texture and color matching methods. All 5 images seem to share the brick wall elements with the target image.

Extension 3: Using Edge Histogram Feature:

The program reads a CSV file containing image file names, loads corresponding images, and performs edge detection on their grayscale versions using the Canny edge detector. The edge histogram is then computed from the detected edges, and the resulting histogram is converted into a feature vector. These feature vectors are stored alongside their respective image file names. The program reads a target image, computes its edge histogram features, and compares these features with those of the images from the CSV file using cosine distance. The top N closest matches are then sorted and printed, providing a ranked list of similar images based on edge histogram features.

Result:

```
vishaqj@Vishaqs-Laptop build % ./Ext1 ../../ResNet18_olym.csv 5
Match 1: pic.0522.jpg - Distance: 3.67067e-08
Match 2: pic.0387.jpg - Distance: 6.2362e-08
Match 3: pic.0033.jpg - Distance: 7.13407e-08
Match 4: pic.0796.jpg - Distance: 8.64152e-08
Match 5: pic.0228.jpg - Distance: 1.15018e-07
vishaqj@Vishaqs-Laptop build %
```

Reflections:

Ravi Shankar Sankara Narayanan:

This project was very helpful in learning about the various methods in matching images based on conventional features and from the feature vector obtained from Deep neural networks. Working on this project has provided me with more insights on how deep learning networks are much better at this and the principle behind their effectiveness. This project has provided a valuable learning experience in understanding the practical applications of these techniques.

Vishaq Jayakumar:

I have implemented image feature extraction using Gabor features and edge histogram, employing Euclidean and cosine distances for similarity comparison. I have also incorporated robust file handling, including CSV-based image file name retrieval, error handling for empty/grayscale images, and command-line argument validation. Overall, this was a big learning experience for me in terms of code debugging and error handling. Matching the matrix dimensions for various operations involved frequent checking and correction of the code. Printing out the dimensions to frequently check for errors in the right line of code was crucial.