BIRZEIT UNIVERSITY

Faculty of Engineering & Technology

Electrical & Computer Engineering Department

COMPUTER VISION

ENCS5343

Assignment 2

Student Name: Lama Naser 1200190

Instructor: Dr. Aziz Qaroush

Section: 1

Date: 31-12-2023

Abstract

This assignment aims to create a Content-Based Image Retrieval (CBIR) system using color histogram and color moment features. It tests how well these features represent image content for retrieval. The evaluation of the CBIR system involves using metrics like precision, recall, and F1 score. Experiments will be done on a standard image dataset to analyze the system's performance and compare different color features.

Table of Contents

List of Figures

List of Tables

1- Introduction

1.1- Content-Based Image Retrieval (CBIR)

Advances in data storage and image acquisition technologies have enabled the creation of large datasets of images. In order to deal with this data, it was necessary to develop appropriate information systems to manage these collections efficiently. Image search is considered one of the most important services that need support from such systems. In general, there are different methods have been implemented to allow searching across image collections: one based on image textual metadata and another based on image content information.

The commonest approaches use the so-called Content-Based Image Retrieval (CBIR) systems. Basically, these systems try to retrieve images similar to a user-defined specification or pattern (e.g., shape sketch, image example). Their goal is to support image retrieval based on content properties (e.g., shape, color, texture), usually encoded into feature vectors. One of the main advantages of the CBIR approach is the possibility of an automatic retrieval process, instead of the traditional keyword-based approach, which usuallyrequires very laborious and time-consuming previous annotation of database images. The CBIR technology has been used in several applications such as fingerprint identification, biodiversity information systems, digital libraries, crime prevention, medicine, historical research, among others.

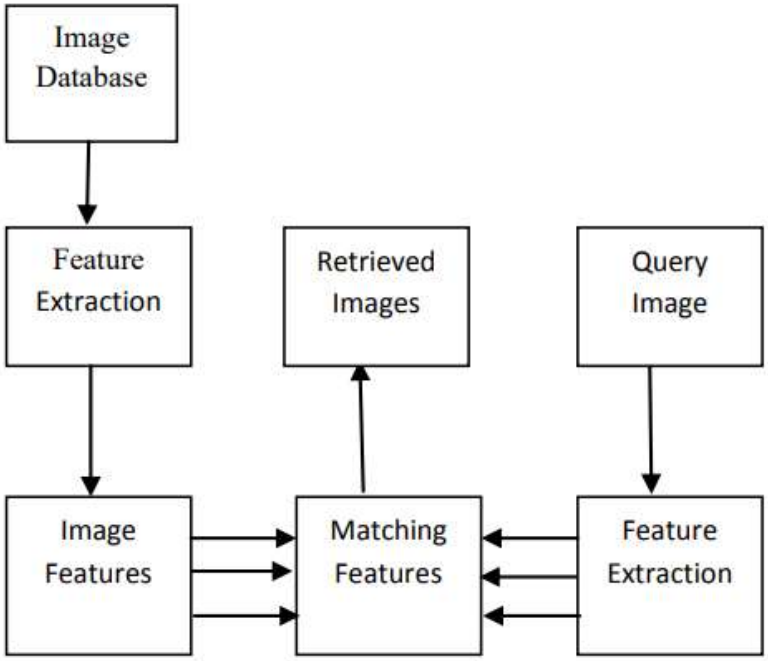


Figure 1: Block diagram of CBIR

The above figure shows the block diagram of a basic CBIR system. A feature vector is extracted from each image in the database and the set of all feature vectors is organized as a database index. At query time, a feature vector is extracted from the query image and it is matched against the feature vectors in the index.

1.2- theoretical background of color histograms

A color histogram represents the distribution of colors in an image, where each histogram bin corresponds to a color in the quantized color space. Color histograms are a set of bins where each bin represents a particular color of the color space being used. The number of bins depends on the number of colors there in the image. A color histogram for a given image is defined as a vector: H = {H[0], H[1], H[2], H[3],.....H[i].....H[n]} (1)

Where i represents the color bin in the color histogram and H[i] represents the number of pixels of color i in the image, and n is the total number of bins used in the color histogram. Typically, each pixel in an image will be assigned to a bin of a color histogram of that image, and so for the color histogram of an image, the value of each bin is the number of pixels that has the same corresponding color. In order to compare images of different sizes, color histograms should be normalized. The normalized color histogram H ' is defined as the following.

H '[i] = $\frac{H[i]}{p}$, where p is the number of pixels in the image. (2)

However, color histogram has its own drawbacks. If two images have exactly the same color proportion but the colors are scattered differently, then we can't retrieve correct images. There are two unrelated images in figure 2 below, but they have the same color histogram.

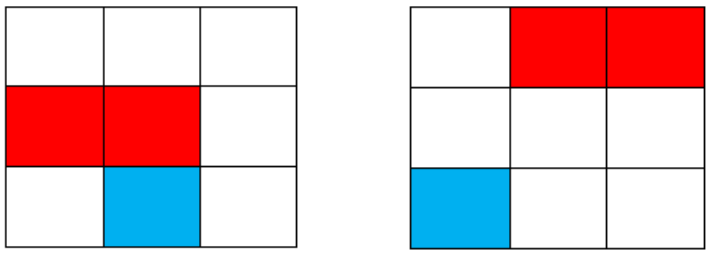


Figure 2: Example of two different images having same

We extract the color histogram of the image and we will get a 256-dimensional color feature vector: HQ = (h0,h1,h2,h3……h255) (3)

Histogram intersection method is used to measure the distance 'S1' between the query image Q and the image P in the image database.

S1 = 1 - $\sum_{i=0}^{255} \min(HQ[i], Hp[i])$ (4)

1.3- theoretical background of color moment

Color moments have been successfully used in content based image retrieval systems. It has been shown that characterizing one dimensional color distributions with the first three moments is more robust and runs faster than the histogram based methods. in order to improve the discriminating power of color indexing techniques, the image is divided horizontally into three equal non-overlapping regions and from each of the three regions, we extract from each color channel the first three moments of the color distribution and store the 27 floating point numbers in the index of the image. If the color distribution of an image is interpreted as a probability distribution, then the color distribution can be characterized by its moments. If the value of the ith color channel at the jth image pixel is Iij and the number of pixels is N, then the index entries related to this color channel and the region 'r' are:

$$E_{r,i} = \frac{1}{N}\sum_{j=1}^{N}\left(I_{ij}\right) \qquad (5)$$

$$\sigma_{r,i} = \sqrt{\left[\frac{1}{N}\sum_{j=1}^{N}\left(I_{ij} - E_{r,i}\right)^2\right]} \qquad (6)$$

$$s_{r,i} = \sqrt[3]{\left[\frac{1}{N}\sum_{j=1}^{N}\left(I_{ij} - E_{r,i}\right)^3\right]} \qquad (7)$$

The entries Er,i (1 <= i <=3) are the average color of the region r. The entries σr,i and sr,i are the variance and the skewness of each color channel in this region 'r'. The index entry for one image consists of Index size = number of regions x number of color Channels x 3 floating point numbers. The distance between Q and P using Euclidean distance is given as the following.

$$s_2 = \sqrt{\left[\sum_{i=1}^{27}(f^Q(i) - f^P(i))^2\right]} \qquad (8)$$

Such that f^Qdenote color moment feature vector of query image Q. Also, f^P denote color moment feature vector of database image P.

2- System Implementation

The CBIR system was implemented using Python language. For this assignment, three separate scripts were written using PyCharm program. Each script was for a specific type of image representation techniques.

2.1- CBIR system with Color Histogram representation

The main functionality of this system is that it takes a set of query images and a dataset of images. Then, it calculates color histogram for each image. After that, it computes the Euclidean distance between the queries histograms and dataset histograms. Following that, it ranks the images based on the distances. Finally, it evaluates the performance through some metrics which they are precision, recall, F1 score and Receiver Operating Characteristic (ROC) curve.

2.1.1- Programing languages and libraries

The implementation used Python language with some libraries. First, OpenCV (Open Source Computer Vision Library) this library was used for image reading and loading from files (“.jpg”,”.jpeg”, “.png”) using “cv2.imread()”. Also it was used for histogram calculation using “cv2.calcHist()” which is a function that is used for computing color histograms. Second, NumPy library which it is a numerical computing library for Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. Third, scikit-learn library which it was used for computing precision, recall, and ROC curve metrics. Fourth, time library was used to compute the total time required for each query Finally, Matplot library was used to create a Receiver Operating Characteristic (ROC) curve.

```
2   import numpy as np
3   import os
4   import time
5   from sklearn.metrics import precision_recall_curve, auc
6   import matplotlib.pyplot as plt
7   from sklearn.metrics import roc_curve
```

Figure 3: Libraries used for CBIR system with Color Histogram representation

2.1.2- Implementation Details

1. User interface: At the beginning of the program, it asks the user to enter the number of pins which is the number of bins used for each color channel when calculating color histogram. After that It asks for the threshold distance value that will determines which images that will be considered as relevant images to a specific query image.

```
181  #main function starts here
182  print("\n-------------------------")
183  print("Welcome to my CBIR System")
184  print("-------------------------\n")
185
186  dataset_path = "data_set"
187  queries_path = "quires_set"
188
189  while(1):
190      try:
191          pins = input("\nEnter the pins value: ")
192          pins = int(pins)
193          threshold = input("Enter the threshold: ")
194          threshold = float(threshold)
195          cbir(queries_path, dataset_path,pins,threshold)
196          break
197      except ValueError:
198          print("Invalid input...\n")
```

Figure 4: main function

2. Loading images: The way used for loading images was the same for queries or dataset images as shown below by looping over their folders' files then using readable images without storing them in lists since that was so bad for memory while running. When I tried it with lists, I faced a problem that made my device freeze.

```
for queryImage in os.listdir(queryset_path):
    start_time = time.time()
    if (queryImage.lower().endswith(('.jpg', '.jpeg', '.png'))):
        queryImage_path = os.path.join(queryset_path, queryImage)
        query_image = cv2.imread(queryImage_path)
        query_features = calculate_histogram(query_image,pins)
        query_name = os.path.basename(queryImage_path)

        distances = []
        for image_file in os.listdir(dataset_path):
            if image_file.lower().endswith(('.jpg', '.jpeg', '.png')):
                image_path = os.path.join(dataset_path, image_file)
                image = cv2.imread(image_path)

                db_features = calculate_histogram(image,pins)

                # Use Euclidean distance as the similarity measure
                distance = euclidean_distance(query_features,db_features)
```

Figure 5: Loading images

3. Inage feature representation: Features were represented by color histograms calculated using the “calculate_histogram” function as shown below. The function takes the image as an input and the number of pins to be used in each channel of the histogram. And then computes the histogram using “cv2. calcHist()” function. This function takes [0, 1, 2] channels (0 for blue, 1 for green and 2 for red). The next parameter was passed as ‘None’ which is represents the mask (no mask used). The next parameter is [pins, pins, pins] which represents the number of pins for each channel. The last one is [0, 256, 0, 256, 0, 256]: Range of pixel values for each channel (0 to 255 for 8-bit images). The last step normalizes the histogram using cv2.normalize. Then The resulting histograms were flattened into a one-dimensional array using “flatten()”.

```
def calculate_histogram(image, pins):
    hist = cv2.calcHist([image], [0, 1, 2], None, [pins, pins, pins], [0, 256, 0, 256, 0, 256])
    hist = cv2.normalize(hist, hist).flatten()
    return hist
```

Figure 6: calculate_histogram function

4. Distance Measure: The Euclidean distance used as a similarity measure between the query and dataset images as shown below. It calculates the Euclidean distance between the two histograms using NumPy's np.linalg.norm function.

```
def euclidean_distance(hist1, hist2):
    return np.linalg.norm(hist1 - hist2)
```

Figure 7: Euclidean distance function

5. Reference Labels: For each query that was chosen from the data set, the most similar images was collected manually. These collected images were used as a reference that tells which images are truly relevant for a specific query, so that the true positive and false positive labels can be calculated.

```
def get_true_labels(query_name,image_name,true_labels):
    # define reference true labels for each query
    query0 = ["0.jpg","22.jpg","98.jpg","129.jpg","139.jpg","826.jpg",
              "971.jpg","2218.jpg","2697.jpg","3516.jpg"]
    query1 = ["1.jpg","58.jpg","64.jpg","183.jpg","370.jpg","4492.jpg",
              "587.jpg","1447.jpg","1945.jpg","2351.jpg","3861.jpg"]
    query2 = ["25.jpg","217.jpg","1040.jpg","2164.jpg"]
    query3 = ["3.jpg","3592.jpg","3778.jpg","4694.jpg","4730.jpg"]
    query4 = ["4.jpg","3407.jpg","3180.jpg","2992.jpg"]
    query5 = ["5.jpg","3421.jpg"]
    query6 = ["6.jpg","3266.jpg","3417.jpg","3418.jpg"]
    query7 = ["7.jpg","3254.jpg"]
    query8 = ["8.jpg","3218.jpg"]
    query9 = ["9.jpg","3352.jpg"]

    #find the query
```

Figure 8: reference images

6. Image Ranking: Images are ranked based on their Euclidean distances to the query image histograms as shown below. It uses a built in Python method “sort” that sorts lists. The “key” argument was used to specify an implicit function (in this case, this function is ‘lambda’) that it is used to extract the sorting key from each element in the list. In this case, it sorts based on the second element (x[1]) which is the Euclidean distance between the query and image features in ascending way.

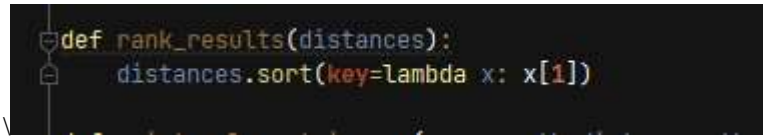


Figure 9: rank images function

7. Evaluation Metrics: This function uses “precision_recall_fscore_support” function from the scikit-learn library. This function computes precision, recall, and F1 score for each class (in this case, the binary classes 0 and 1).

Figure 10: calculate metrics function

8. Calculate Area Under the Curve (AUC): The function uses the “roc_curve” function from the scikit-learn library to compute the Receiver Operating Characteristic (ROC) curve that represents the trade-off between true positive rate and false positive rate at various thresholds. The “auc” function is then used to compute the area under the ROC curve.

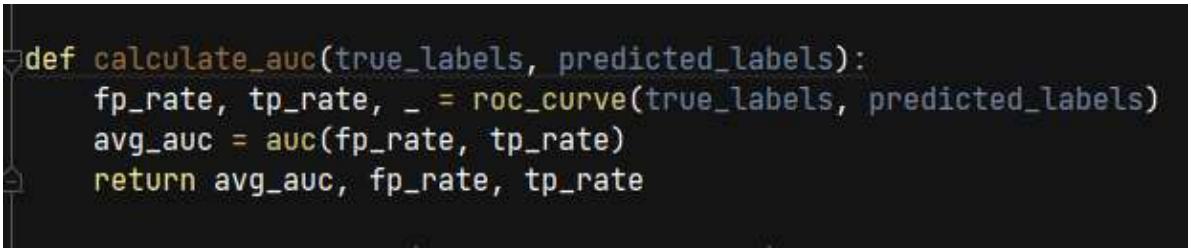


Figure 11: calculate auc function

9. Relevant Images: The following function displays for a specific query all relevant images that the Euclidean distance between the image and the query image is less than or equal a specific threshold provided from the user.

```
def print_relevant_images(query_path,distances,threshold):
    print("Related images for "+query_path)
    #sort distances based on the euclidean distances
    rank_results(distances)
    j = 0
    while (1):
        if (j < len(distances)):
            if (distances[j][1] <= threshold):
                print(f"{distances[j][0]} - Euclidean Distance: {distances[j][1]}")
            else:
                breaka
        else:
            break
        j += 1
    print("\n")
```

Figure 12: Show relevant images function

10. Display metrics results: The following function calculates the average precision, recall, F1 score, execution time, area under the curve (AUC) and the ROC. After that, it prints all these results to the user. Finally, it displays the ROC curve plot using "matplotlib.pyplot" library.

```
def print_metrics(true_labels,predicted_labels,execution_lists):
    # Calculate average metrics
    avg_precision, avg_recall, avg_f1 = calculate_metrics(true_labels, predicted_labels)
    # calculated average execution time (divide it by 60.0 to get a result in minutes)
    avg_execution = np.mean(execution_lists) / 60.0
    # Calculate AUC
    avg_auc, fp_rate, tp_rate = calculate_auc(true_labels, predicted_labels)
    # print results
    print("Average Precision:       " + str(avg_precision))
    print("Average Recall:          " + str(avg_recall))
    print("Average F1 Score:        " + str(avg_f1))
    print("Average AUC:             " + str(avg_auc))
    print("Average execution time:  " + str(avg_execution) + " minutes")

    # Plot ROC curve
    plt.figure()
    plt.plot(fp_rate, tp_rate, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(avg_auc))
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

Figure 13: show results function

2.1.3- Interaction of Modules

In The CBIR system, the user starts by entering the number of pins to be tested and the threshold value. The threshold value represents the distance that will determine which images to be considered as relevant. Then, the function “cbir” iterates over the query images and for each one it calculates the color histogram of it and then iterates over the dataset images. For each image in the dataset, it computes the Euclidean distance between this image histogram and the query histogram. Then, it labels that image as relevant (give it value one) if the Euclidean distance calculated is less than or equal the threshold distance and labels it as not relevant (give it value zero) if it is larger than the threshold value. After finish looping over the dataset images, the function sorts the distances using “rank_results” function. Finally after looping over query images, it finds the average execution time, precision, recall and F1 score using “calculate_metrics” function. Also it plots the ROC curve and calculates the area under the curve (AUC) and prints it along with the other metrics using “print_metrics” function.

2.2- CBIR system with Color Moments representation

The main functionality of this system is that it takes a set of query images and a dataset of images. Then, it calculates color moments as features for each image. After that, it computes the Euclidean distance between the queries and dataset images. Finally, it evaluates the performance through some metrics which they are precision, recall, F1 score and Area Under Curve (AUC) to measure the overall performance of the system.

2.2.1- Programing languages and libraries

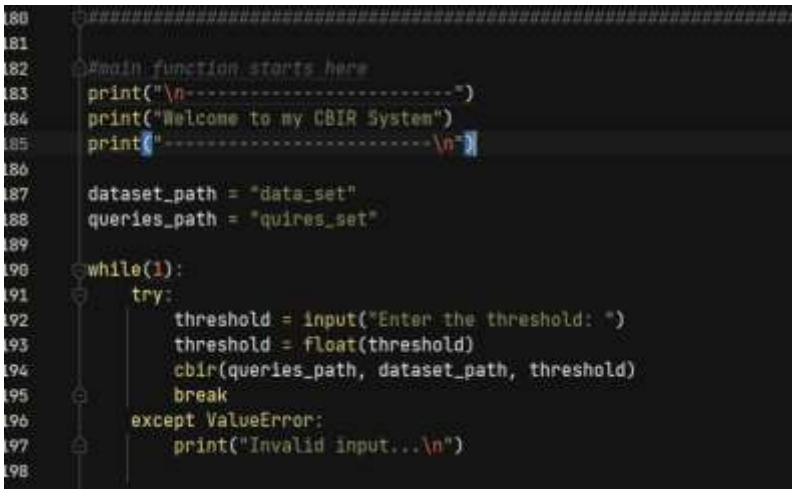
The implementation used Python language with some libraries. First, OpenCV (Open Source Computer Vision Library) this library was used for image reading and loading from files (“.jpg”,”.jpeg”, “.png”) using “cv2.imread()”. Also it was used to convert the images to HSV color space using “cv2. cvtColor ()”. Second, NumPy library which it is a numerical computing library for Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. Third, scikit-learn library which it was used for computing precision, recall, and AUC metrics. Fourth, time library was used to compute the total time required for each query. Finally, "scipy.stats" library is used for calculating the skewness of the image color moments.

```
1  import cv2
2  import os
3  import numpy as np
4  from sklearn.metrics import precision_recall_fscore_support, roc_curve, auc
5  from sklearn.metrics.pairwise import euclidean_distances
6  from scipy.stats import skew
7  import time
```

Figure 14: Libraries used for CBIR system with color moment representation

2.2.2- Implementation Details

1. User interface: At the beginning of the program, it requires from the user to enter the threshold distance value that will determine which images will be considered as related and which will considered as not.



```
#main function starts here
print("\n-----------------------")
print("Welcome to my CBIR System")
print("-----------------------\n")

dataset_path = "data_set"
queries_path = "quires_set"

while(1):
    try:
        threshold = input("Enter the threshold: ")
        threshold = float(threshold)
        cbir(queries_path, dataset_path, threshold)
        break
    except ValueError:
        print("Invalid input...\n")
```

Figure 15: main function

2. Loading images: The way used for loading images was the same for queries or dataset images as shown below by looping over their folders and using read image without storing them in list since that was so bad for memory while running. When I tried it with lists, I faced a a problem that made my device freeze.

```
def cbir(queryset_path, dataset_path, threshold):

    weights = [1.0, 3.0, 2.0, 4.0, 1.5, 2.5]
    true_labels = []
    predicted_labels = []
    execution_lists = []

    for queryImage in os.listdir(queryset_path):
        start_time = time.time()
        if (queryImage.lower().endswith(('.jpg', '.jpeg', '.png'))):
            queryImage_path = os.path.join(queryset_path, queryImage)
            query_image = cv2.imread(queryImage_path)
            query_features = calculate_color_moments(query_image,weights)
            query_name = os.path.basename(queryImage_path)

            distances = []
            for image_file in os.listdir(dataset_path):
                if image_file.lower().endswith(('.jpg', '.jpeg', '.png')):
                    image_path = os.path.join(dataset_path, image_file)
                    image = cv2.imread(image_path)

                    db_features = calculate_color_moments(image,weights)
```

Figure 16: Loading images

3. Inage feature representation: This function takes an image and a set of weights as input and calculates color moments based on the HSV (Hue, Saturation, Value) color space. It combines mean, standard deviation, and skewness values for each channel in the HSV color space. It uses weights to customize the contribution of each moment to the final feature vector.

```
def calculate_color_moments(image, weights):
    # Convert the image to HSV color space
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Calculate mean, standard deviation, and skewness for each channel (H, S, V)
    mean_values = np.mean(hsv_image, axis=(0, 1))
    std_dev_values = np.std(hsv_image, axis=(0, 1))
    skewness_values = skew(hsv_image, axis=(0, 1))  # Use skew from scipy.stats

    # Apply weights to each moment
    mean_values *= weights[0]
    std_dev_values *= weights[1]
    skewness_values *= weights[2]

    # Concatenate the weighted feature vectors
    color_moments = np.concatenate((mean_values, std_dev_values, skewness_values))

    return color_moments
```

Figure 17: Color moment calculation function

4. Image feature representation (part 2): In this assignment, another three moments were used to test the system with more moments. The following function calculates the moments which were explained in the above point as well as the new moments which they are “Kurtosis”, it is a measure of the "tailedness" of the probability distribution. Also “Median”, it is the middle value of the pixel intensities. Finally the “Mode”, it is most frequently occurring pixel intensity for each channel.

```
def calculate_color_moments(image, weights):
    # Convert the image to HSV color space
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # Calculate mean, standard deviation, skewness, kurtosis, median, and mode for each channel (H, S, V)
    moments = {
        'mean': np.mean(hsv_image, axis=(0, 1)),
        'std_dev': np.std(hsv_image, axis=(0, 1)),
        'skewness': skew(hsv_image, axis=(0, 1)),
        'kurtosis': kurtosis(hsv_image, axis=(0, 1)),
        'median': np.median(hsv_image, axis=(0, 1)),
        'mode': np.apply_along_axis(lambda x: np.argmax(np.bincount(x)), axis=0,
                                    arr=np.apply_along_axis(lambda x: np.argmax(np.bincount(x)),
                                                            axis=1, arr=hsv_image))
    }
    # Apply weights to each moment
    color_moments = np.concatenate([moments[key] * weights[i] for i, key in enumerate(['mean', 'std_dev',
                                                                                        'skewness', 'kurtosis',
                                                                                        'median', 'mode'])])
    return color_moments
```

Figure 18: Color moment calculation function with foure moments

5. Distance Measure: The Euclidean distance used as a similarity measure between the query and dataset images as shown below. It calculates the Euclidean distance between the two histograms using NumPy's np.linalg.norm function.

```
def euclidean_distance(features1, features2):
    return np.linalg.norm(features1 - features2)
```

Figure 19: Euclidean distance calculation function

6. Reference Labels: For each query that was chosen from the data set, the most similar images was collected manually. These collected images were used as a reference that tells which images are truly relevant for a specific query, so that the true positive and false positive labels can be calculated.

```
def get_true_labels(query_name,image_name,true_labels):
    # define referance true labels for each query
    query0 = ["0.jpg","22.jpg","98.jpg","129.jpg","139.jpg","826.jpg",
              "971.jpg","2218.jgp","2697.jpg","3516.jpg"]
    query1 = ["1.jpg","58.jpg","64.jpg","103.jpg","370.jpg","4492.jpg",
              "587.jpg","1447.jpg","1945.jpg","2351.jpg","3061.jpg"]
    query2 = ["25.jpg","217.jpg","1040.jpg","2164.jpg"]
    query3 = ["3.jpg","3592.jpg","3778.jpg","4694.jpg","4730.jpg"]
    query4 = ["4.jpg","3407.jpg","3180.jpg","2992.jpg"]
    query5 = ["5.jpg","3421.jpg"]
    query6 = ["6.jpg","3266.jpg","3417.jpg","3410.jpg"]
    query7 = ["7.jpg","3254.jpg"]
    query8 = ["8.jpg","3210.jpg"]
    query9 = ["9.jpg","3352.jpg"]

    #find the query
```

Figure 20: Reference images

7. Image Ranking: Images are ranked based on their Euclidean distances to the query image histograms as shown below. It uses a built in Python method “sort” that sorts lists. The “key” argument was used to specify an implicit function (in this case, this function is ‘lambda’) that it is used to extract the sorting key from each element in the list. In this case, it sorts based on the second element (x[1]) which is the Euclidean distance between the query and image features in ascending way.

```
def rank_results(distances):
    distances.sort(key=lambda x: x[1])
```

Figure 21: rank results function

8. Calculate metrics: This function uses “precision_recall_fscore_support” function from the scikit-learn library. This function computes precision, recall, and F1 score for each class (in this case, the binary classes 0 and 1).

```
def calculate_metrics(true_labels, predicted_labels):
    precision, recall, f1, _ = precision_recall_fscore_support(true_labels, predicted_labels, average='binary')
    return precision, recall, f1
```

Figure 22: calculate metrics function

9. Calculate Area Under the Curve: The function uses the “roc_curve” function from the scikit-learn library to compute the Receiver Operating Characteristic (ROC) curve that represents the trade-off between true positive rate and false positive rate at various thresholds. The “auc” function is then used to compute the area under the ROC curve.

```
def calculate_auc(true_labels, predicted_labels):
    fp_rate, tp_rate, _ = roc_curve(true_labels, predicted_labels)
    avg_auc = auc(fp_rate, tp_rate)
    return avg_auc
```

Figure 23: Calculate AUC function

10. Showing relevant images: The following function displays for a specific query all relevant images that the Euclidean distance between the image and the query image is less than or equal a specific threshold provided from the user.

```
def print_relevant_images(query_path,distances,threshold):
    print("Related images for " + query_path)
    # sort distances based on the euclidean distances
    rank_results(distances)
    j = 0
    while (1):
        if (j < len(distances)):
            if (distances[j][1] <= threshold):
                print(f"{distances[j][0]} - Euclidean Distance: {distances[j][1]}")
            else:
                break
        else:
            break
        j += 1
    print("\n")
```

Figure 24: print relevant images function

11. Displaying results: the following function calculates the average precision, recall, F1 score, execution time and area under the curve (AUC). Also it displays all these results to the user.

```
def print_metrics(true_labels,predicted_labels,execution_lists):
    # Calculate average metrics
    avg_precision, avg_recall, avg_f1 = calculate_metrics(true_labels, predicted_labels)
    # calculated average execution time (divide it by 60.0 to get a result in minutes)
    avg_execution = np.mean(execution_lists) / 60.0
    # Calculate AUC
    avg_auc = calculate_auc(true_labels, predicted_labels)
    # print results
    print("Average Precision: " + str(avg_precision))
    print("Average Recall: " + str(avg_recall))
    print("Average F1 Score: " + str(avg_f1))
    print("Average AUC: " + str(avg_auc))
    print("Average execution time: " + str(avg_execution) + " minutes")
```

Figure 25: print metrics results function

2.2.3- Interaction of Modules

The program starts when the user enters a threshold value. For the selected 10 queries, the program loops for each query. Each time calculates the color moment features for this query. Then it starts looping on the dataset images. For each one, it calculates the color moment features to compute the Euclidean distance between the query features and the image features. At this point, it the distance is less than or equal the entered threshold from the user then it handseled as relevant image to the query. Finally, after finish looping on all queries the metrics (precision, recall, f1 score and time) can be calculated as an average of all queries as well as the AUC to measure the overall performance of the CBIR system.

2.3- Another image representation techniques

For this section, I chose Local Binary Patterns (LBP) as an image representation technique. To implement the feature extraction function using technique, the following code was used to achieve this. First, the input image was converted to grey scale that will simplify the image to a single channel. Second, for each pixel in the gray scale image, its intensity was compared with he intensities of its neighboring pixels. Such that a binary pattern was created by assigning 1 if the intensity of the neighbor is greater than or equal to the center pixel, and 0 otherwise. After that, a histogram was created by counting the occurrences of different binary patterns. Finally, The created histogram was normalized to get a feature vector.

This technique was used because of its simplicity. Also, LBP is relatively robust to changes in illumination. It focuses on local patterns, making it less affected by global illumination variations. Moreover, LBP is computationally efficient, making it suitable for large datasets.

```python
def calculate_lbp(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    lbp = feature.local_binary_pattern(gray, P=5, R=1, method="uniform")
    (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, 10 + 1), range=(0, 10))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist
```

Figure 26: Features extraction using LBP representation

3- Experimental Setup and results

3.1- Evaluation Methodology

The dataset used in this section was downloaded from the following website: CBIR_dataset (kaggle.com) . It involves 4738 different images. From this dataset, I chose the first 10 images and used them as the queries (numbered from 0 to 9). For evaluation metrics, precision, recall, F1 score and time were used because these metrics provides a holistic view of the CBIR system's strengths and weaknesses. Precision and recall give insights into the accuracy and completeness of retrieval, F1 score balances the trade-off between the two, and time assesses the efficiency of the system. Moreover, ROC and AUC were also used as an evaluation metric since The ROC curve offers a visual representation of the system's performance and AUC is useful for assessing the overall discriminative power of the CBIR system.

3.2- CBIR with Color Histogram Representation

This section divided into the following parts in order to get good view about evaluation methodologies used and results that was obtained.

3.2.1- Results Presentation

3.2.1.1- Results with pins = 120

1. Threshold value = 0.85

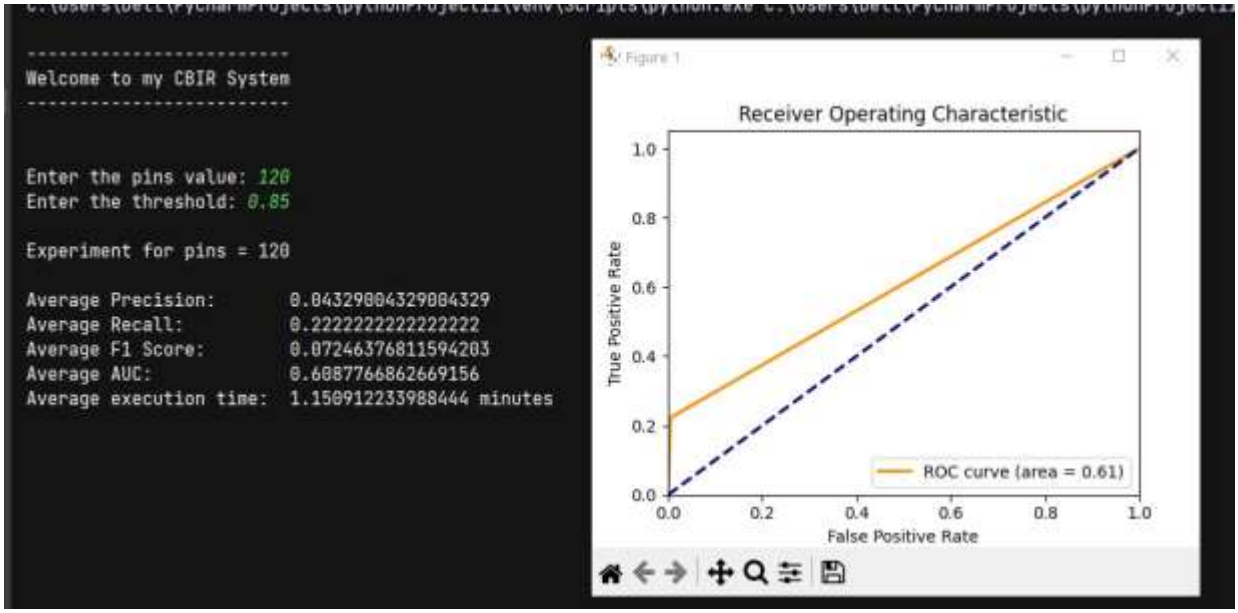


Figure 27: color histogram with pins = 120, threshold = 0.85

2. Threshold value = 0.95

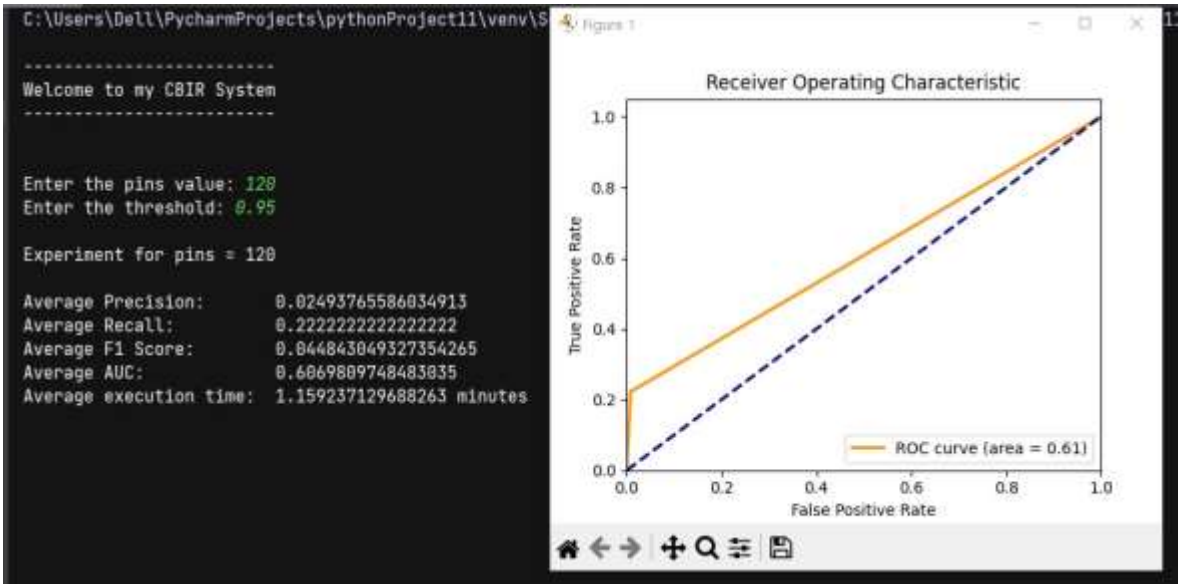


Figure 28: Color histogram with pins = 120, threshold = 0.95

As shown above, changing the threshold value affects the metrics used. Also this affects the number of relevant images. For example, the following query was used to see the relevant images (threshold value = 0.85)

Figure 29: test query image

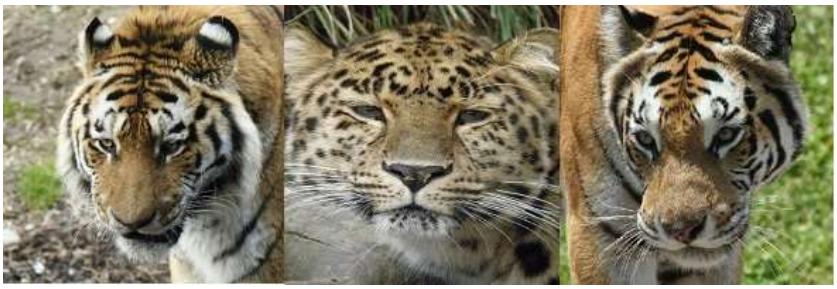


Figure 30: Relevant images to the test query image, pins = 120, threshold = 0.85

Figure 31: Some of relevant images to the test query image, pins = 120, threshold = 0.95

3.1.2.2- Results with pins = 180

1. Threshold = 0.85

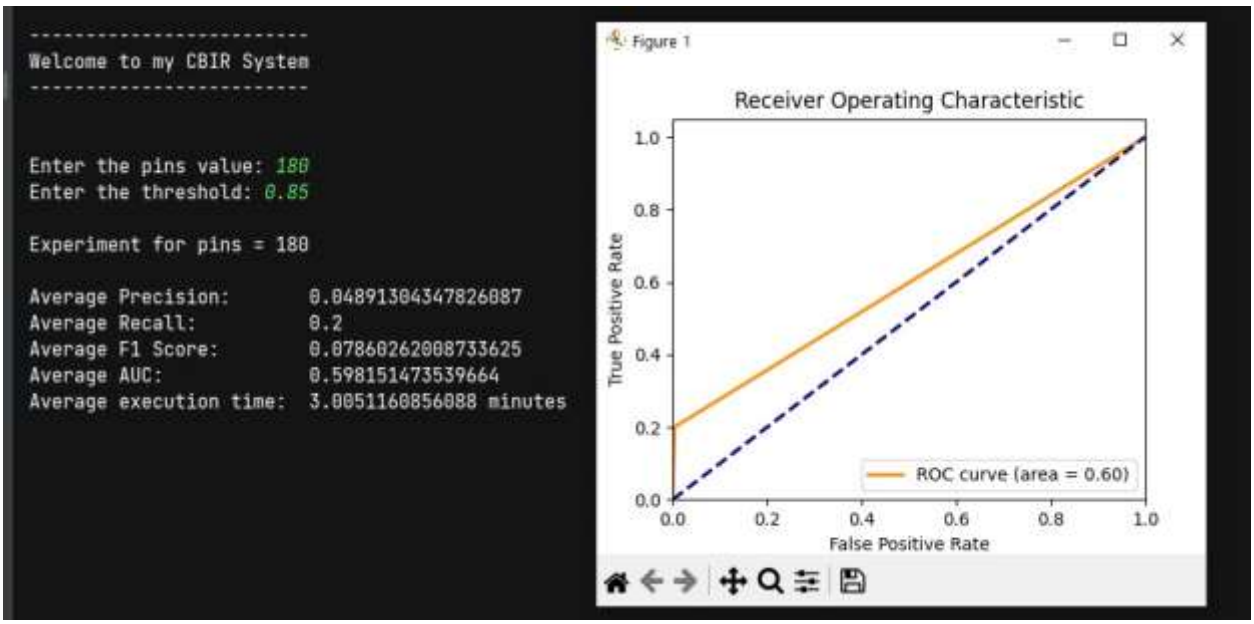


Figure 32: Color histogram results with pins = 180, threshold = 0.85

2. Threshold = 0.95

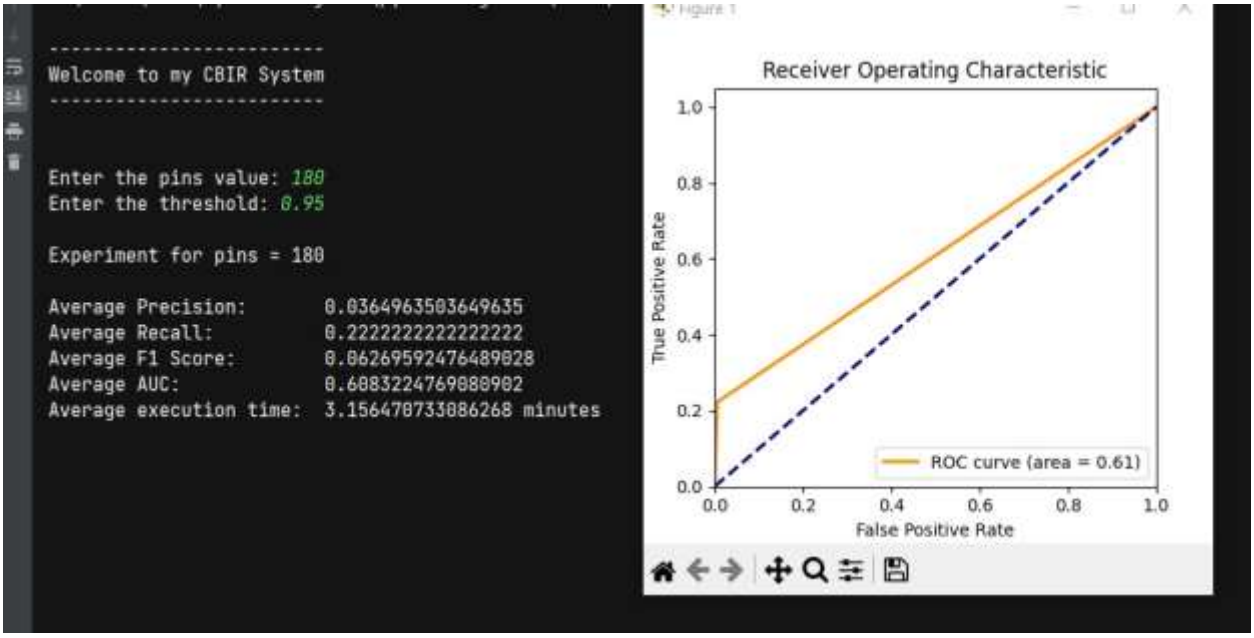


Figure 33: Color histogram results with pins = 180, threshold = 0.95

Using pins = 180 with threshold = 0.85 gave only one relevant images which is test query itself. While using it with threshold = 0.95 gave the following relevant images for the test query that was shown before.

Figure 34: related images using pins = 180 with threshold = 0.95

3.1.2.3- Results with pins = 360

1. Threshold = 0.85

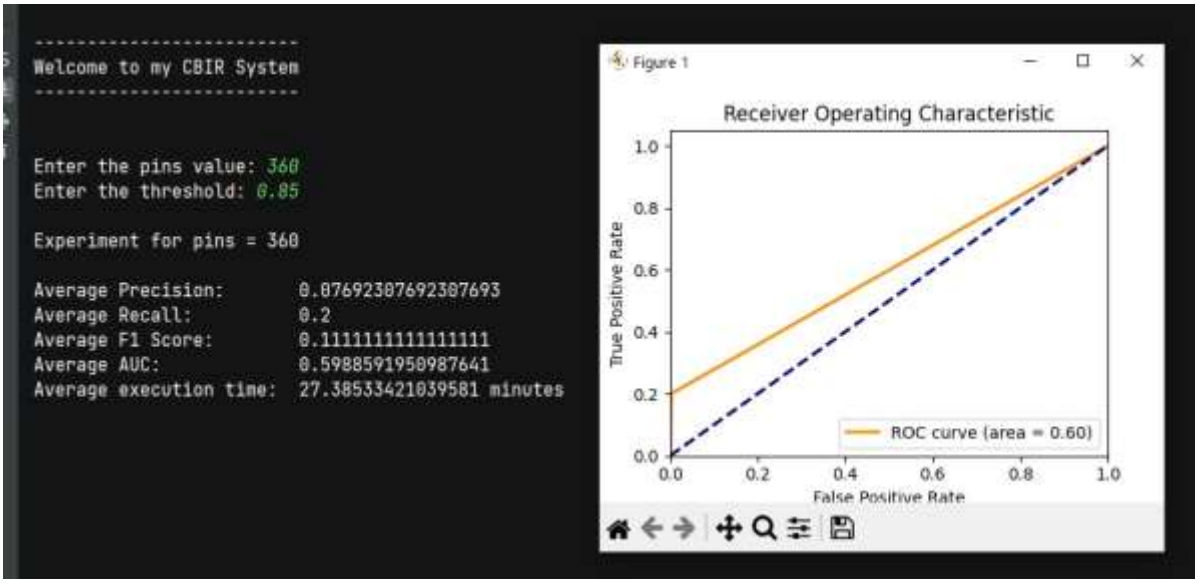


Figure 35: Color histogram results with pins = 360, threshold = 0.85

2. Threshold = 0.95

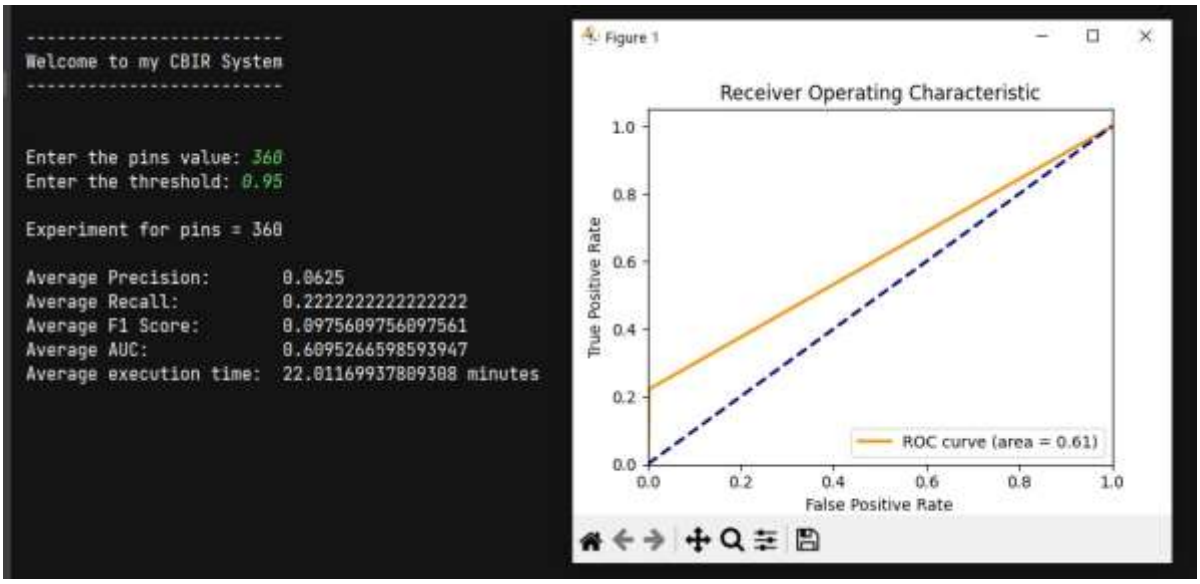


Figure 36: Color histogram results with pins = 360, threshold = 0.95

Using pins = 180 with threshold = 0.85 gave only one relevant images which is test query itself. While using it with threshold = 0.95 gave the following relevant images for the test query that was shown before.

Figure 37: related images using pins = 360 with threshold = 0.95

3.2.2- Comparisons and Discussion

All metrics results got from the previse parts were recorded in the following table.

Table 1: Color histogram final results

	Threshold = 0.85			Threshold = 0.95		
	Pins = 120	Pins = 180	Pins = 360	Pins = 120	Pins = 180	Pins = 360
Precision	0.0433	0.0489	0.0769	0.0249	0.0365	0.0625
Recall	0.2222	0.2	0.2	0.2222	0.2222	0.2222
F1 score	0.0725	0.0786	0.1111	0.0448	0.0627	0.0975
time (min)	1.1509	3.0051	27.385	1.1592	3.1565	22.011
AUC	0.6088	0.5981	0.5988	0.6069	0.6083	0.6095

It was noticed that using larger threshold leads to get more relevant images retrieved from the system but that led to a possibility of getting non similar images of the query image. Also it was noticed that there seems to be a trade-off between precision and recall. As precision increases, recall tends to decrease. Also, increasing the pins value leads to a longer execution time. However, increasing the pins led to better accuracy. Moreover, the AUC indicates moderate discriminative power across different pin and threshold values.

3.3- CBIR with Color Moments Representation

3.3.1- Results presentation

3.3.1.1- Color Moments with equal weights

This section shows the results got from using Color moments as an image representation technique for feature extraction with equal weights for each moment (here each weight was used equal to one) with different threshold values.

1. Threshold = 5.0

```
------------------------
Welcome to my CBIR System
------------------------

Enter the threshold: 5.0
Average Precision:       0.8333333333333334
Average Recall:          0.2222222222222222
Average F1 Score:        0.3508771929824561
Average AUC:             0.6110899850944216
Average execution time:  5.946614696979522 minutes

Process finished with exit code 0
```

Figure 38: Metrics results for color moments with equal wieghts, threshold = 5.0

2. Threshold = 10.0

```
------------------------
Welcome to my CBIR System
------------------------

Enter the threshold: 10.0
Average Precision:       0.37037037037037035
Average Recall:          0.2222222222222222
Average F1 Score:        0.27777777777777773
Average AUC:             0.61093153996925
Average execution time:  6.0542075081666304 minutes

Process finished with exit code 0
```

Figure 39: Metrics results for color moments with equal wieghts, threshold = 10.0

For the tested query that was shown in figure 29 before, it was again used to test this part. The following is the relative images got from using color moments with equal weights for both thresholds 5.0 and 10.0.

Figure 40: relative images with threshold = 5.0

Figure 41: relative images with threshold = 10.0

3.3.1.2- Color Moments with different weights

In this section, different weights were used for the moments such that the weight value is related to the importance of the moments. Here the higher weight was given to the skewedness (5.0), then the standard deviation (2.0) and the lower weight was for the Mean (0.2).

1. Threshold = 5.0

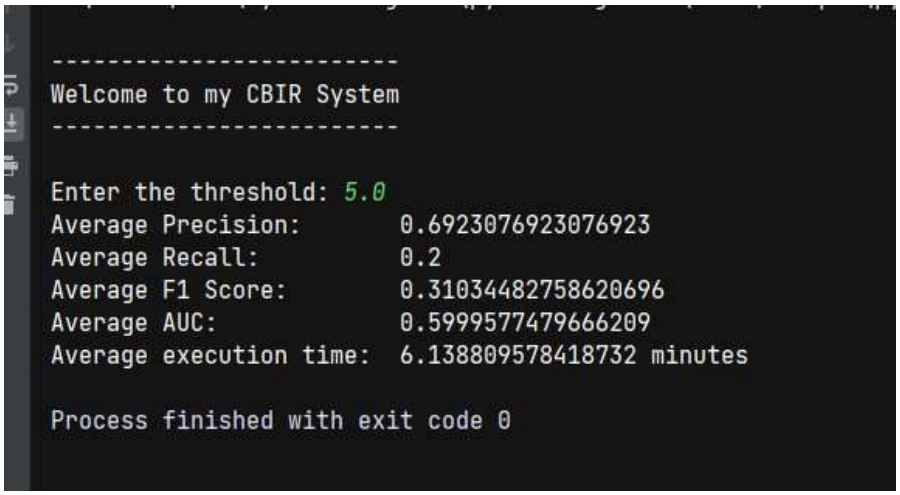


Figure 42: Metrics results for color moments with different weights, threshold = 5.0

2. Threshold = 10.0

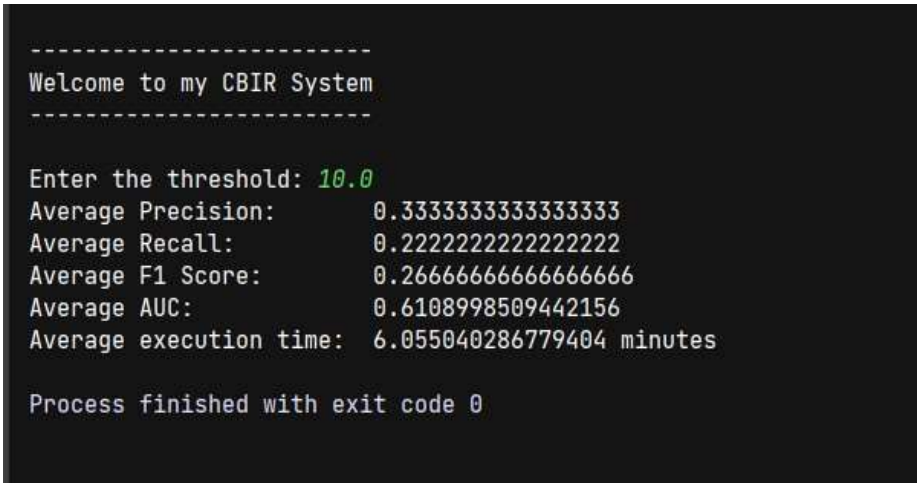


Figure 43: Metrics results for color moments with different weights, threshold = 10.0

For the tested query that was shown in figure 29 before, it was again used to test this part. The following is the relative images got from using color moments with different weights for threshold 10.0. While the relative images got from using threshold 5.0 were only the test query image itself.

Figure 44: relative images with threshold = 10.0

3.3.1.3- Color Moments with different weights with more moments

In this section, four moments were used with different weights. The first one is the mean with weight = 1.0 (no significant emphasis, contributes equally). Second, the standard deviation with weight = 3.0 (emphasizes variations in color intensity). Third, the skewness with weight = 2.0 (moderate emphasis on the asymmetry of the color distribution). Fourth, kurtosis with weight = 4.0 (strong emphasis on the "tailedness" of the distribution). Fifth, Median with weight = 1.5 (moderate emphasis on the middle value of pixel intensities). Finally Mode with weight = 2.5 (strong emphasis on the most frequently occurring pixel intensity).

1. Threshold = 5.0

```
------------------------
Welcome to my CBIR System
------------------------

Enter the threshold: 5.0
Average Precision:       0.9
Average Recall:          0.2
Average F1 Score:        0.32727272727272727
Average AUC:             0.5999894369916552
Average execution time:  10.582293638388315 minutes

Process finished with exit code 0
```

Figure 45: metrics results using four moments with threshold = 5.0

2. Threshold = 10.0

```
-------------------------
Welcome to my CBIR System
-------------------------

Enter the threshold: 10.0
Average Precision:       0.6
Average Recall:          0.2
Average F1 Score:        0.3
Average AUC:             0.5999366219499312
Average execution time:  10.30863928159078 minutes

Process finished with exit code 0
```

Figure 46: metrics results using four moments with threshold = 10.0

For the tested query that was shown in figure 29 before, it was again used to test this part. The following is the relative images got from using color moments with four moments each with different weights for threshold 10.0. While the relative images got from using threshold 5.0 were only the test query image itself.

Figure 47: relative images with threshold = 10.0

3.3.2- Comparisons and Discussion

All metrics results got from the previse parts were recorded in the following table.

Table 2: Color moments final results

	Threshold = 5.0			Threshold = 10.0		
	Equal weights	Different weights	More moments	Equal weights	Different weights	More moments
Precision	0.8333	0.6923	0.9	0.3704	0.3333	0.6
Recall	0.2222	0.2	0.2	0.2222	0.2222	0.2
F1 score	0.3509	0.3103	0.3273	0.2778	0.2666	0.3
time (min)	5.946	6.139	10.58	6.054	6.055	10.30
AUC	0.6111	0.5999	0.5999	0.6109	0.6109	0.5999

It was noticed that using larger threshold leads to get more relevant images retrieved from the system but that led to a possibility of getting non similar images of the query image. Also it was noticed that using color moments in general gave a better results than using color histograms as an images representation technique. As for the color moments, it was noticed that assigning different weights to the moments can improve the system performance but these weights are related to the nature of the images. Moreover, using more moments including Kurtosis, Median and Mode gave a precision = 0.9 means that out of all the instances predicted as positive, 90% of them are actually true positive instances which means the predicted relevant images were mostly true. However, using this method led to more computations as well as the time required for it.

3.4- CBIR with Another Image Representation

In this section, the performance tried to be improved by using LBP representation using P (number of points) = 5, R (radius) = 1 and threshold = 0.5.

```
-------------------------
Welcome to my CBIR System
-------------------------

Enter the threshold: 0.5
Average Precision:       0.6923076923076923
Average Recall:          0.2
Average F1 Score:        0.31034482758620696
Average AUC:             0.5999577479666209
Average execution time:  5.652507937351862 minutes

Process finished with exit code 0
```

Figure 48: Metrics results by using LBP technique

For the same test query used in figure 26 before, it was again used here to see the relevant images got from using this technique with threshold = 0.5.

Figure 49: Relevant images to the tested query using LBP with threshold = 0.5

It was concluded that using LBP gave a higher precision compared to color histogram but lower than color moments. While LBP has a lower recall compared to both color histogram and color moments. AUC values are comparable, with LBP being slightly lower. Finally, LBP has a moderate execution time, falling between color histogram and color moments.

4- Conclusion

In this assignment, it taught to create a Content-Based Image Retrieval (CBIR) system using color histogram and color moment features. It helped to learn how well these features represent image content for retrieval. The evaluation of the CBIR system involves using metrics like precision, recall, and F1 score.

The overall effectiveness of the color features for CBIR tasks was influenced by the color representation technique used. Color moments, with their ability to capture richer color information, seemed as a more strong representation for image retrieval tasks than using color histogram representation technique.

For insights and recommendations of the CBIR system and further researches, it is important to have most reliable reference images since the ones used in this assignment were selected manually and cannot cover all relative images for all queries because of the big size of the dataset. Also, experimenting more cases (more pins values or weights in the color moment) with more threshold values to get the best parameters values. Moreover, the system could be integrated with advanced machine learning techniques, such as deep learning architectures will improve he CBIR system's capabilities to recognize complex patterns and semantic information.