جامعة بيرزيت
BIRZEIT UNIVERSITY
Faculty of Engineering and Technology
Electrical and Computer Engineering Department

ENCS5343 Computer Vision

Student name : Zainab Shaabneh

Student ID : 1182820

Instructor name : Ismail Khater

Introduction

Content-Based Image Retrieval (CBIR) is a crucial component of computer vision, enabling the retrieval of images based on their visual content rather than relying on metadata or textual descriptions. CBIR has various applications, including object detection, image recognition, and multimedia retrieval .However, CBIR faces several challenges that affect its performance and usability. Here is an overview of the CBIR system, its challenges, and potential solutions: CBIR involves feature extraction, where low-level visual features are extracted from images. These features are used to build a representation of the image content. CBIR algorithms compare these feature representations to retrieve similar images from a database. The retrieval can be based on a query image or specified visual features. CBIR algorithms compare the query image or features with the feature representations of the images in the database to retrieve the most similar ones.

Theory

Color moments

Color moments are measures that can be used differentiate images based on their features of color. Once calculated, these moments provide a measurement for color similarity between images. These values of similarity can then be compared to the values of images indexed in a database for tasks like image retrieval.

The basis of color moments lays in the assumption that the distribution of color in an image can be interpreted as a probability distribution. Probability distributions are characterized by a number of unique moments (e.g. Normal distributions are differentiated by their mean and variance). It therefore follows that if the color in an image follows a certain probability distribution, the moments of that distribution can then be used as features to identify that image based on color.

Stricker and Orengo [1] use three central moments of a image's color distribution. They are Mean, Standard deviation and Skewness. A color can be defined by 3 or more values. (Here we will restrict ourselves to the HSV scheme of Hue, Saturation and brightness, although alternative encoding could just as easily be used.) Moments are calculated for each of these channels in an image. An image therefore is characterized by 9 moments 3 moments for each 3 color channels. We will define the ith color channel at the jth image pixel as p_{ij} . The three color moments can then be defined as:

MOMENT 1 – Mean :

$$E_i = \sum_{N}^{j=1} \frac{1}{N} p_{ij}$$

Mean can be understood as the average color value in the image.

MOMENT 2 Standard Deviation :

$$\sigma_i = \sqrt{\left(\frac{1}{N}\sum_{N}^{j=1}(p_{ij} - E_i)^2\right)}$$

The standard deviation is the square root of the variance of the distribution.

MOMENT 3 – Skewness :

$$s_i = \sqrt[3]{\left(\frac{1}{N}\sum_{N}^{j=1}(p_{ij} - E_i)^3\right)}$$

Skewness can be understood as a measure of the degree of asymmetry in the distribution.

A function of the similarity between two image distributions is defined as the sum of the weighted differences between the moments of the two distributions. Formally this is:

$$d_{mom}(H, I) = \sum_{i=1}^{r} w_{i1}\left|E_i^1 - E_i^2\right| + w_{i2}\left|\sigma_i^1 - \sigma_i^2\right| + w_{i3}\left|s_i^1 - s_i^2\right|$$

Precision and Recall

PRECISION is the ratio of the number of relevant images retrieved to the total number of irrelevant and relevant images retrieved. It is usually expressed as a percentage.

$$Precision = \frac{[relevantimages] \cap (retrievedimages)}{(retrievedimages)} \quad (15)$$

RECALL is the ratio of the number of relevant records retrieved to the total number of relevant records in the database. It is usually expressed as a percentage.

$$Recall = \frac{[relevantimages] \cap (retrievedimages)}{(relevantimages)} \quad (16)$$

Color histogram

The histogram describes the gray-level or color distribution for a given image. It is a global feature which can be used to perform a fast but no so reliable indexing process. The histogram feature can be used as a preliminary step for database indexing in order to reduce the number of candidate images for the next steps which could use other features (e.g. shape, texture, orientation) to compare the database images with a given query image. The major advantage offered by the histogram feature consists in its small sensitivity to scale, rotation and translation [1]. An appropriate color space, a color quantization scheme, a histogram representation, and a similarity metric are the main ingredients required for the design of a histogram based retrieval system [2]. The RGB color space is inappropriate for image retrieval due to the fact that it is not related with the way humans perceive colors. Other color spaces like opponent color space [1], HSI is generally used for retrieval proposes .

Steps to do Color Histograms

Step 1

To create our histograms we first need images. Feel free to use any images you like, but, if you'd like to follow along with the same images, you can download them using Hugging Face Datasets.

```
1  from datasets import load_dataset
2  data=load_dataset('pinecone/image-set',split='train',revision='e7d39fc')
3  data
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:72: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
```

Figure 1: dataset

Step 2

Inside the image_bytes feature of this dataset we have base64 encoded representations of 21 images. We decode them into OpenCV compatible Numpy arrays like so:

```
[]  1  from base64 import b64decode
    2  import cv2
    3  import numpy as np
    4  def process(sample):
    5      image_bytes=b64decode(sample['image_bytes'])
    6      image =cv2.imdecode(np.frombuffer(image_bytes,np.uint8),cv2.IMREAD_COLOR)
    7      return image
    8  images=[process(sample) for sample in data]
```

Figure 2: decode images

Step 3

This code leaves us with the images in the list images. We can display them with matplotlib.

```
1  import matplotlib.pyplot as plt
2  plt.imshow(images[11])
```

```
<matplotlib.image.AxesImage at 0x7af761b54a30>
```

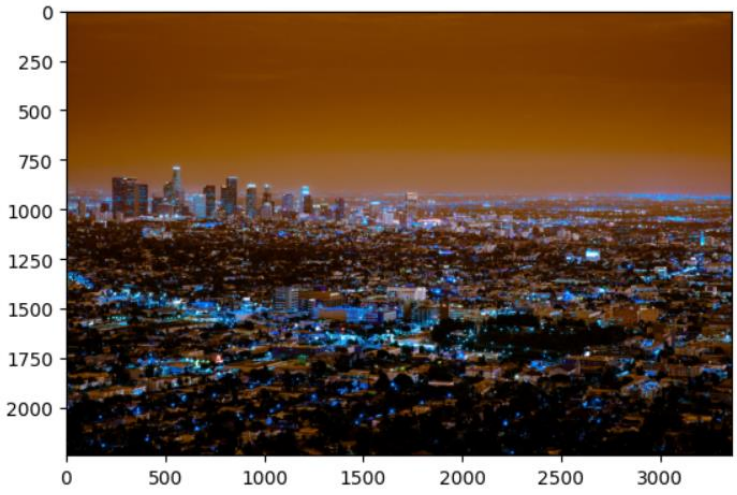


Figure 3: plot image

OpenCV loads images in a Blue Green Red (BGR) format. Matplotlib expected RGB, so we must flip the color channels of the array to get the true color image.

Step 4

Note that while the shape of the array has remained the same, the three values have reversed order. Those three values are the BGR-to-RGB channel values for a single pixel in the image. As shown, after flipping the order of these channels, we can display the true color image.

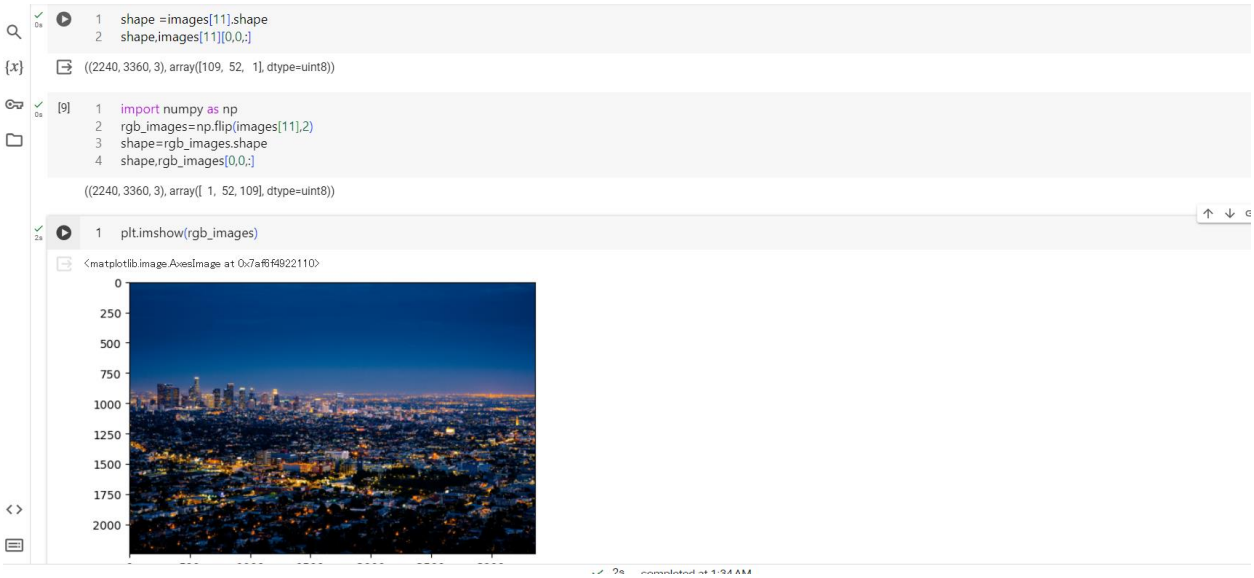


Figure 4: BGR-RGB

Step 5

Every pixel in each image has three BGR color values like this that range on a scale of 0 (no color) to 255 (max color).

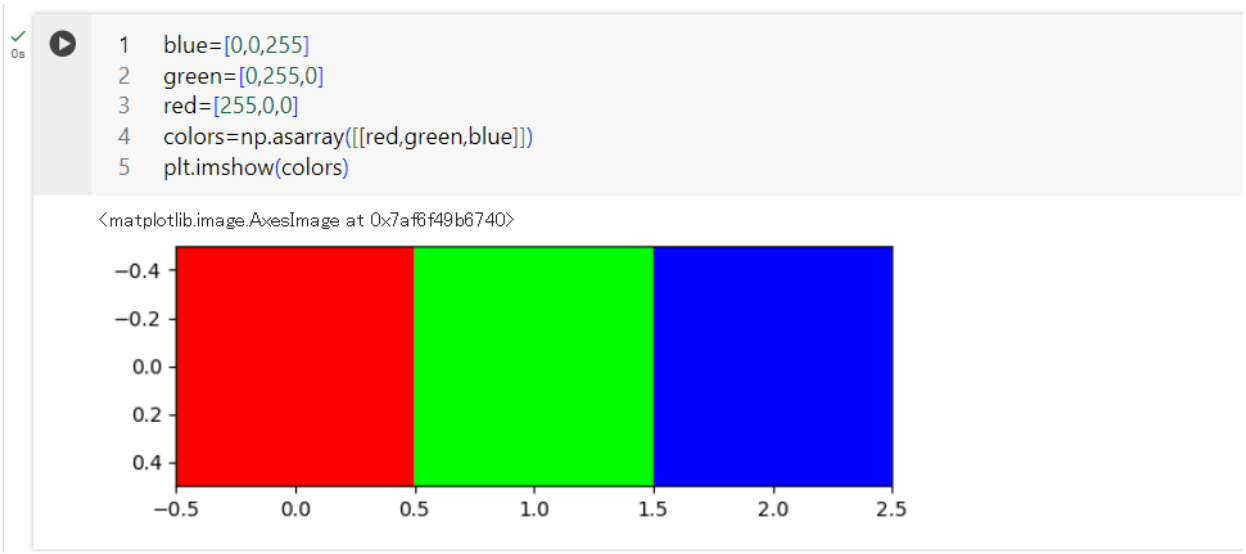


Figure 5 RGB Color

Step 6

we visualize each with a histogram

```
red=image_vector[0,:,0]
green=image_vector[0,:,1]
blue=image_vector[0,:,2]
red.shape ,green.shape , blue.shape
```

```
((7526400,), (7526400,), (7526400,))
```

```
#visualize each with histogram
fig, axs=plt.subplots(1,3,figsize=(15,4),sharey=True)
axs[0].hist(red,bins=256,color='r')
axs[1].hist(green,bins=256,color='g')
axs[2].hist(blue,bins=256,color='b')
plt.show()
```

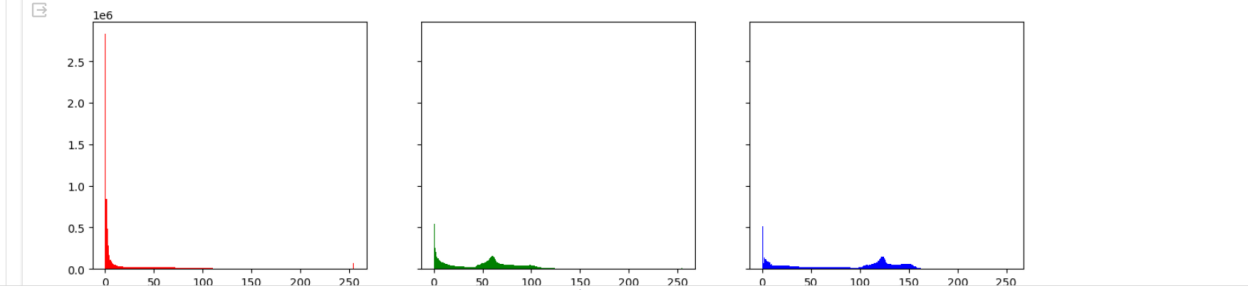


Figure 6: visualize histogram

Here we can see the three color channels RGB. On the x-axis we have the pixel color value from 0 to 255 and, on the y-axis, is a count of the number of pixels with that color value.

Step 7

discretize the histograms into a smaller number of bins. We will add this to a function called build_histogram that will take our image array image and a number of bins and build a histogram for us.

```
def color_histogram(image,pins=256):
    rgb_images=np.flip(image,2) #convert from BGR to RGB
    #show the image
    plt.imshow(rgb_images)
    #convert to a vector
    image_vector=rgb_images.reshape(1,-1,3)
    #beak into given number of pins
    div=256/pins
    pins_vector=(image_vector/div).astype(int)
    #get the red,green,blue channels
    red=pins_vector[0,:,0]
    green=pins_vector[0,:,1]
    blue=pins_vector[0,:,2]
    #bulid the histogram
    fig, axs=plt.subplots(1,3,figsize=(15,4),sharey=True)
    axs[0].hist(red,bins=256,color='r')
    axs[1].hist(green,bins=256,color='g')
    axs[2].hist(blue,bins=256,color='b')
    plt.show()
```

Figure 7: build histogram

Step 8

I apply some images to get an idea of how the color profile of an image can change the histograms.

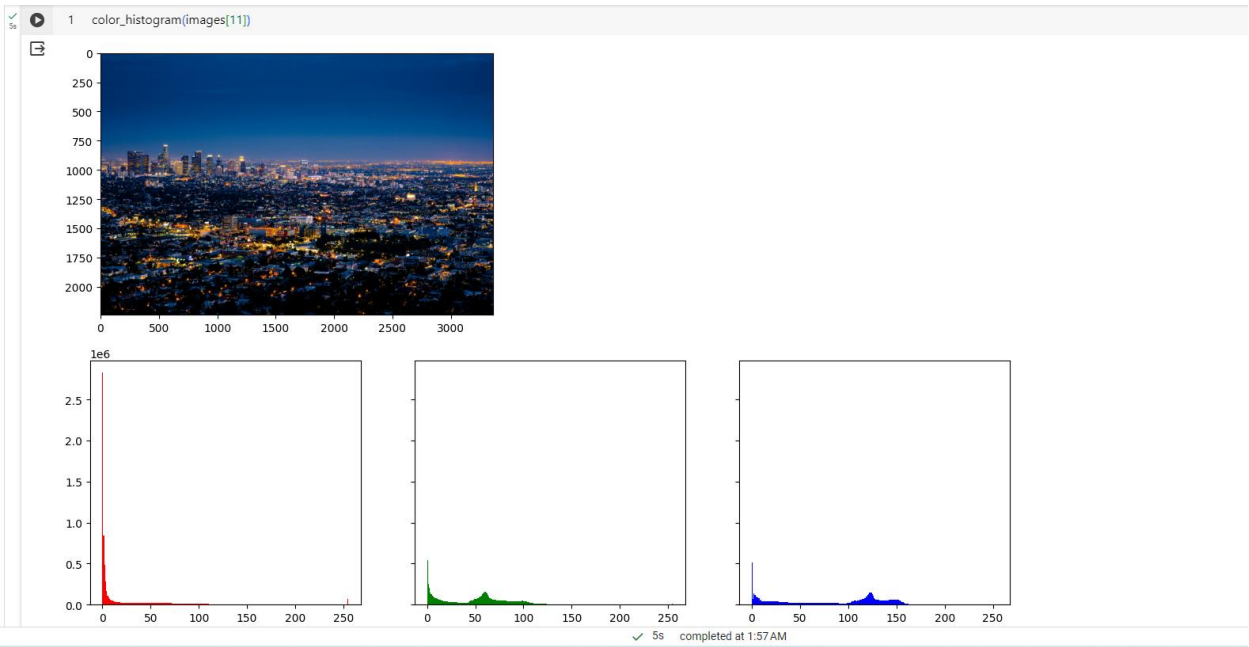


Figure 8: color histogram for image

Step 9

Using 120 pins means that the color spectrum is divided into 120 ranges. Each pin represents a range of colors .

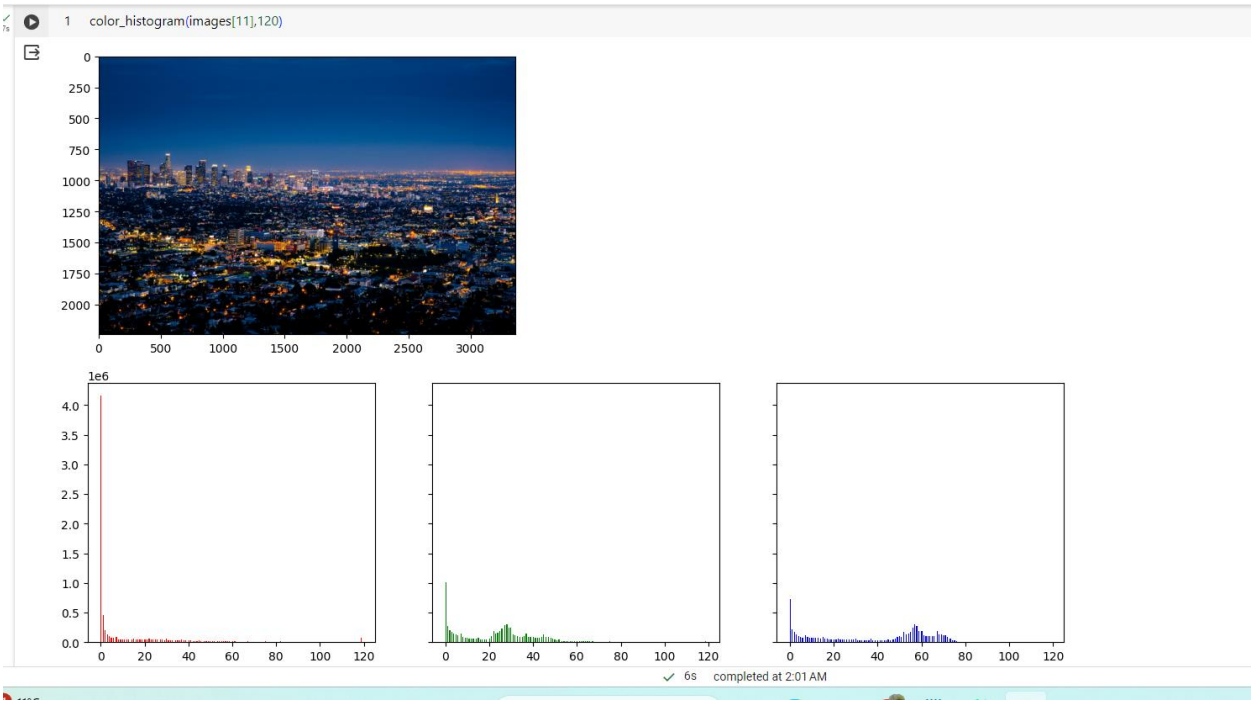


Figure 9: image on 120 pin

Using 180 pins means that the color spectrum is divided into 180 ranges This might lead to a more accurate representation at the cost of increased computational resources.

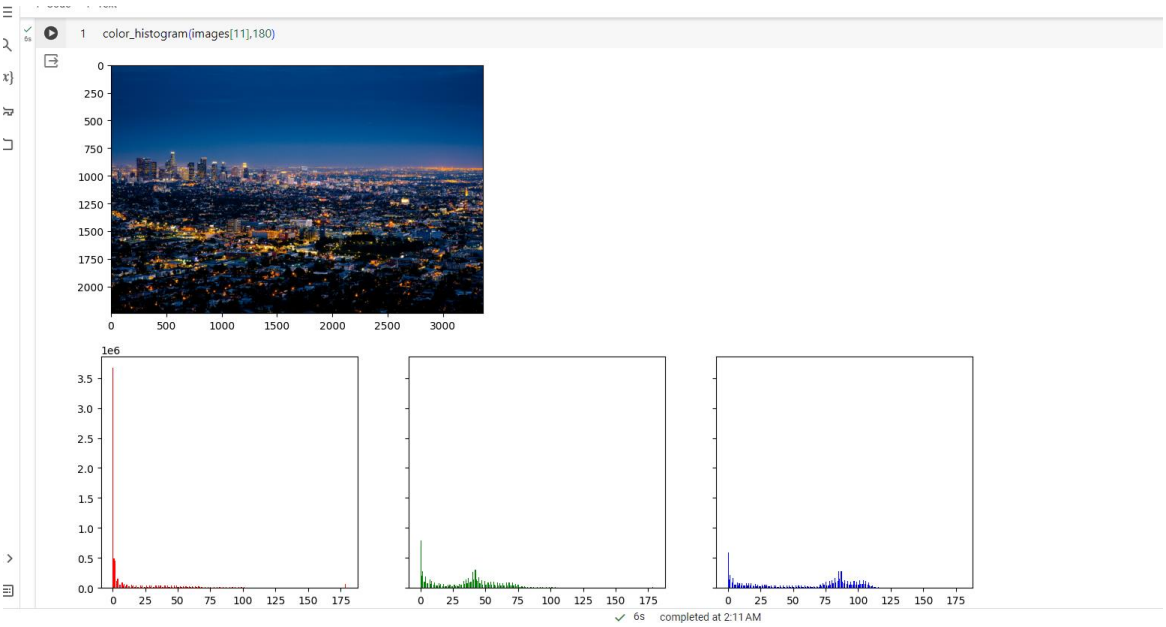


Figure 10: image on 180 pin

Using 256 pins can provide a very detailed color representation, which is useful for precise image retrieval but can be computationally more intensive.

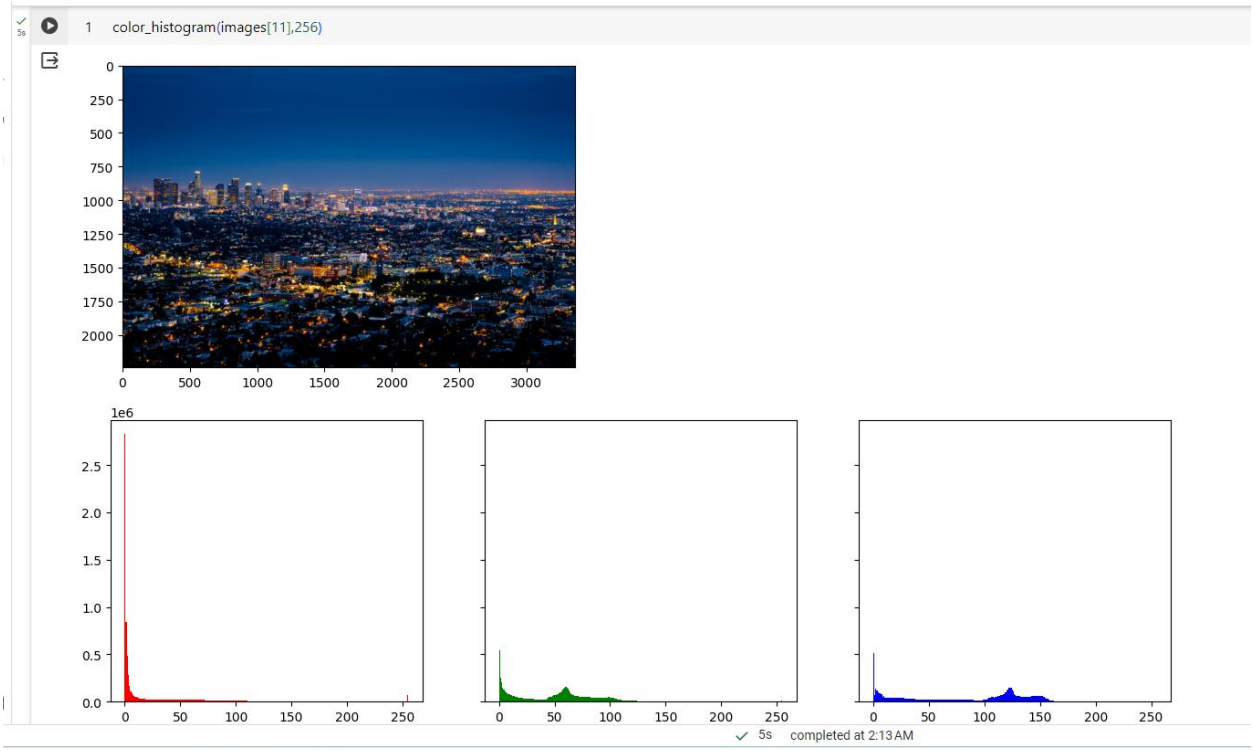


Figure 11: image on 256 pin

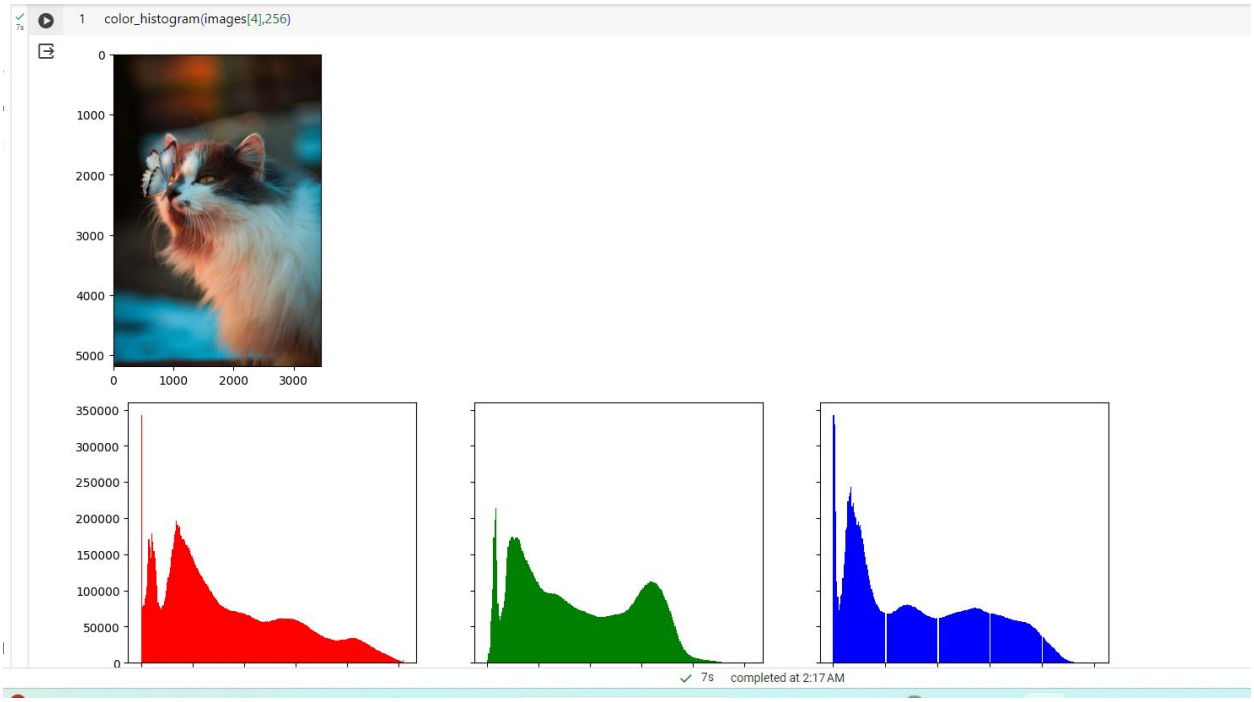


Figure 12: another image on 256 pin

In summary compare the performance and accuracy of the CBIR system with these different bin sizes to determine the optimal configuration for your specific application.

Step 10

Building histograms can be abstracted to be done more easily using the OpenCV library. OpenCV has a function called calcHist specifically for building histograms. We apply it like so:

```
[25]  1  histRed = cv2.calcHist(images[5], [2], None, [256], [0, 256])
      2  histGreen = cv2.calcHist(images[5], [1], None, [256], [0, 256])
      3  histBlue = cv2.calcHist(images[5], [0], None, [256], [0, 256])
      4  histRed.shape

(256, 1)

[26]  1  help(cv2.calcHist)

Help on built-in function calcHist:

calcHist(...)
    calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]]) -> hist
.   @overload
.
.   this variant supports only uniform histograms.
.
.   ranges argument is either empty vector or a flattened vector of histSize.size()*2 elements
.   (histSize.size() element pairs). The first and second elements of each pair specify the lower and
.   upper boundaries.
```

Figure 13: calcHist

images is our cv2 loaded image with a BGR color channel. This argument expects a list of images which is why we have placed a single image inside square brackets [].

channels is the color channel (BGR) that we'd like to create a histogram for; we do this for a single channel at a time.

mask is another image array consisting of 0 and 1 values that allow us to mask (e.g. hide) part of images if wanted. We will not use this so we set it to None.

pins is the number of buckets/histogram bars we place our values in. We can set this to 256 if we'd like to keep all of the original values.

hist_range is the range of color values we expect. As we're using RGB/BGR, we expect a min value of 0 and max value of 255, so we write [0, 256] (the upper limit is exclusive).

Step 11

After calculating these histogram values we can visualize them again using plot.

```
fig, axs=plt.subplots(1,3,figsize=(15,4),sharey=True)
axs[0].plot(histRed,color='r')
axs[1].plot(histGreen,color='g')
axs[2].plot(histBlue,color='b')
plt.show()
```

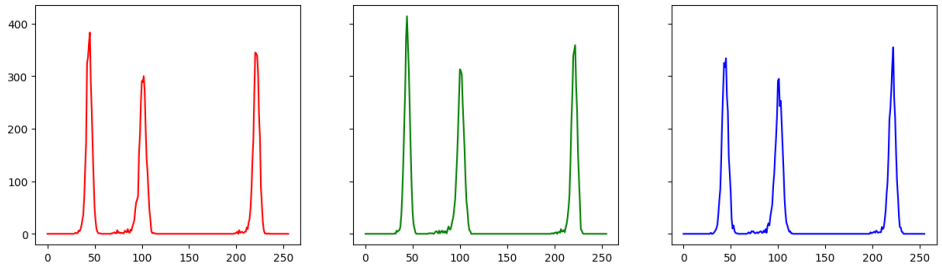


Figure 14: histogram values

Step 12

We have to built a function for transforming our images into three vectors representing the three color channels. Before comparing our images we must concatenate these three vectors into a single vector. We will pack all of this into get_vector:

Using the default pins=256 this function will return a vector with 768 dimensions, where values [0, … 256] are red, [256, … 512] are green, and [512, … 768] are blue.

```
[29]
def get_vector(image,pins=256):
    histRed = cv2.calcHist([image], [0], None, [pins], [0, 256])
    histGreen = cv2.calcHist([image], [1], None, [pins], [0, 256])
    histBlue = cv2.calcHist([image], [2], None, [pins], [0, 256])
    vector=np.concatenate([histRed,histGreen,histBlue],axis=0)
    vector=vector.reshape(-1)
    return vector
```

```
vec=get_vector(images[0])
vec.shape
```

```
(768,)
```

Figure 15: get_vector

```
import matplotlib.pyplot as plt
plt.figure(figsize=(18,6))
plt.plot(list(range(256)),vec[:256],color='red')
plt.plot(list(range(256,512)),vec[256:512],color='green')
plt.plot(list(range(512,768)),vec[512:768],color='blue')
plt.xlim(0,768)
plt.show()
```

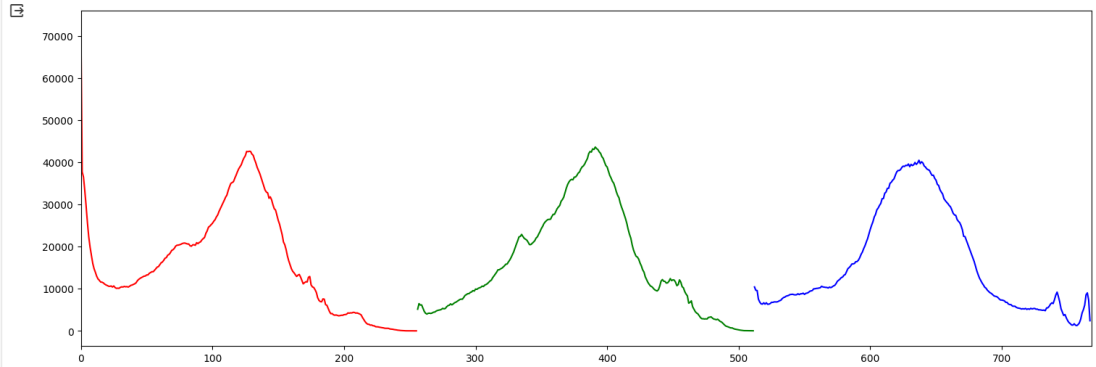


Figure 16: plot get_vector

Step 13

Once we have these vectors we can compare them using typical similarity/distance metrics such as Euclidean distance and cosine similarity. To calculate the cosine similarity we use the formula:

Using our cosine function we can calculate the similarity which varies from 0 (highly dissimilar) to 1 (identical).

```
[33] dist=np.linalg.norm(image_vector[0]-image_vector[1])
     dist
```

```
744737.75
```

```
def euclidean(a,b):
    return np.linalg.norm(a-b)
def cos(a,b):
    return np.dot(a,b)/(np.linalg.norm(a) *np.linalg.norm(b))
```

Figure 17: Euclidean & cos

Step 14

We can apply this alongside everything else we have done so far to create another search function that will return the top_k most similar images to a particular query image specified by its index idx in images.

```
[35]
#during visualization we use this array of rgb images
rgb_images = np. array ([np. flip(image, 2) for image in images])
def search(idx, top_k=5):
    query_vector = image_vector [idx]
    distances = []
    for _,vector in enumerate(image_vector):
        distances. append (cos(query_vector, vector))
     # get top k most similar images
    top_idx = np.argpartition(distances, -top_k) [-top_k:]
    return top_idx
```

```
<ipython-input-35-e5b666666ba9>:2: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarra
 rgb_images = np. array ([np. flip(image, 2) for image in images])
```

```
search(0)
```

```
array([ 3, 18, 14, 15,  0])
```

Figure 18: similarity function

Step 15

This function to display a query image and a set of result images along with their respective color histograms. It helps in visually comparing the color distribution of the query image against each of the result images. However, it contains a few errors and inefficiencies that need correction and optimization.

```
def visualize(query_i,result_i):
    query=rgb_images[query_i]
    span=int(len(image_vector[0])/3)
    results=rgb_images[result_i]
    top_k=len(results)
    fig, axs=plt.subplots(1,2,figsize=(18,10) )
    axs[0].imshow(query)
    axs[0].axis('OFF')
    axs[1].plot(image_vector[query_i][:span],'r',label='red')
    axs[1].plot(image_vector[query_i][span:span*2],'g',label='green')
    axs[1].plot(image_vector[query_i][span*2:],'b',label='blue')
    plt.show()
    #visualize top k images and thier color histogram
    fig0,axs0=plt.subplots(1,top_k,figsize=(top_k*5,5))
    fig1,axs1=plt.subplots(1,top_k,figsize=(top_k*5,5),sharey=True)
    for i in range(top_k):
     axs0[i].imshow(results[i])
     axs0[i].axis('OFF')
     axs[1].plot(image_vector[result_i[i]][:span],'r',label='red')
     axs[1].plot(image_vector[result_i[i]][span:span*2],'g',label='green')
     axs[1].plot(image_vector[result_i[i]][span*2:],'b',label='blue')
```

```
[38]
result_i=search(0)
```

```
result_i
```

```
array([ 3, 18, 14, 15,  0])
```

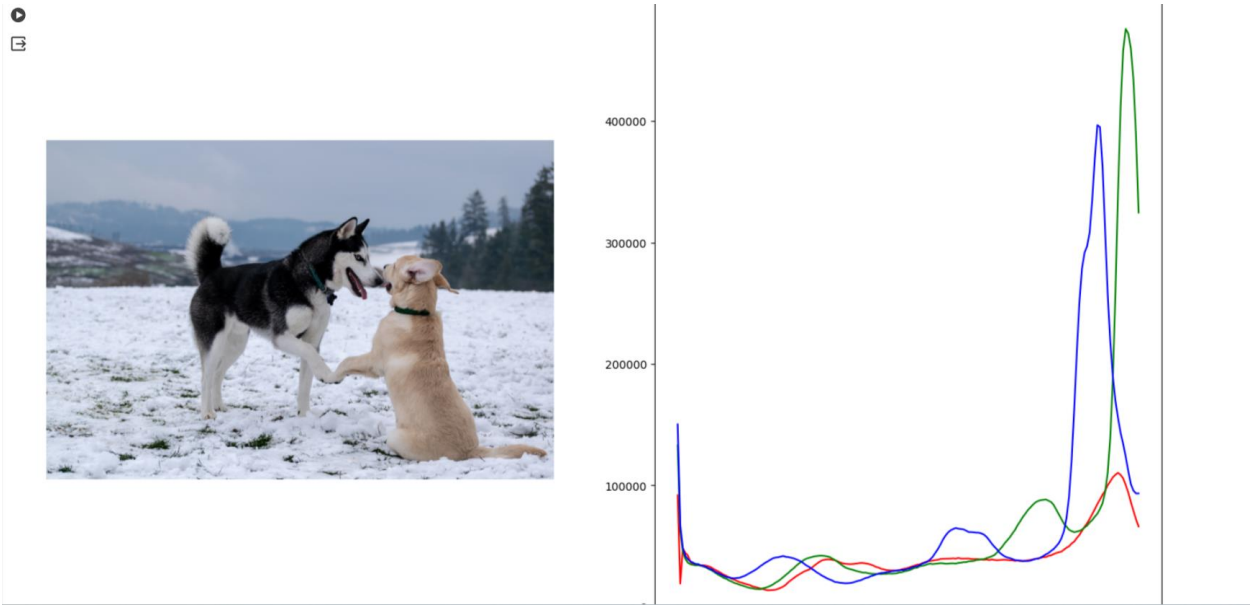


Figure 19: Query Image

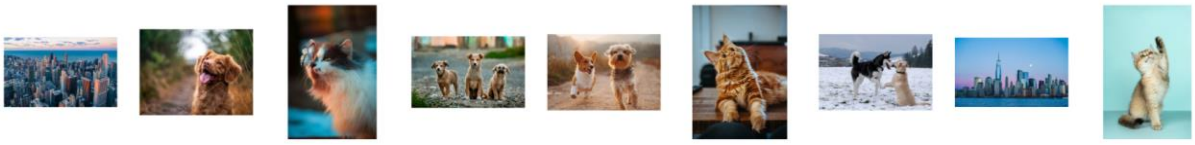


Figure 20: images retrieval

Using the top query image we return the top five most similar images (including the same image) based on their color profiles.
These examples demonstrate the core idea of color histograms. That is, we take an image, translate it into color-based histograms, and use these histograms to retrieve images with similar color profiles.

Task1 :

Task 1.1: color moment

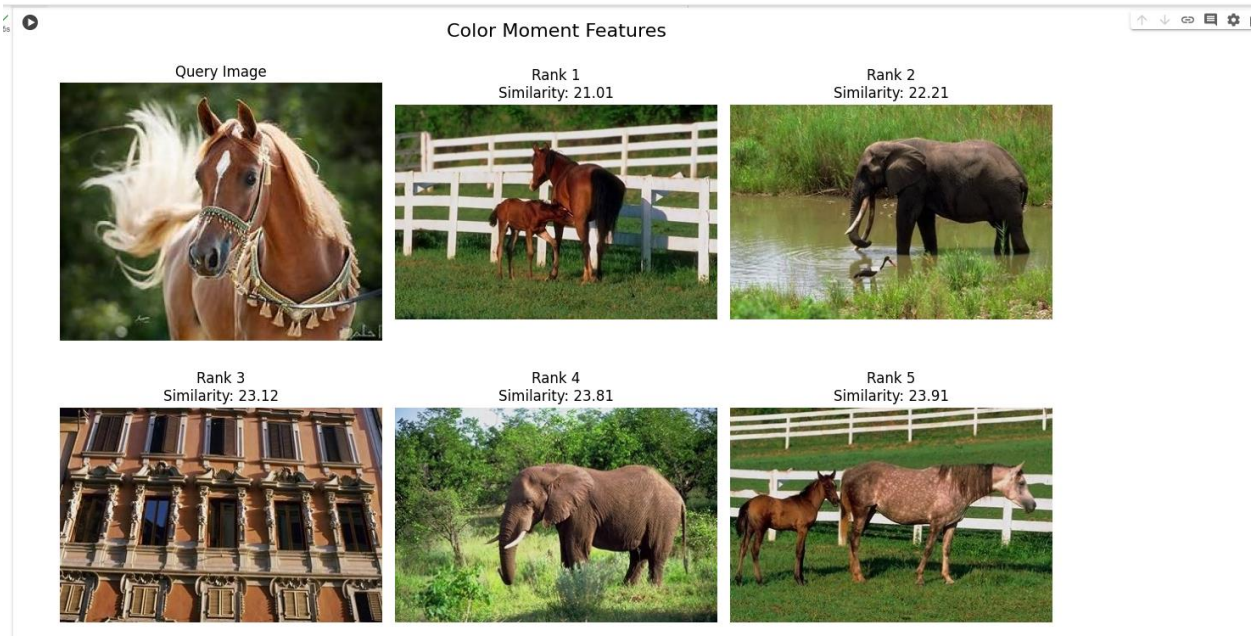


Figure 21: color moment calculation

As we shown the similarity is based on statistical properties. Images with similar mean, standard deviation, and skewness in each color channel will have lower Euclidean distances.

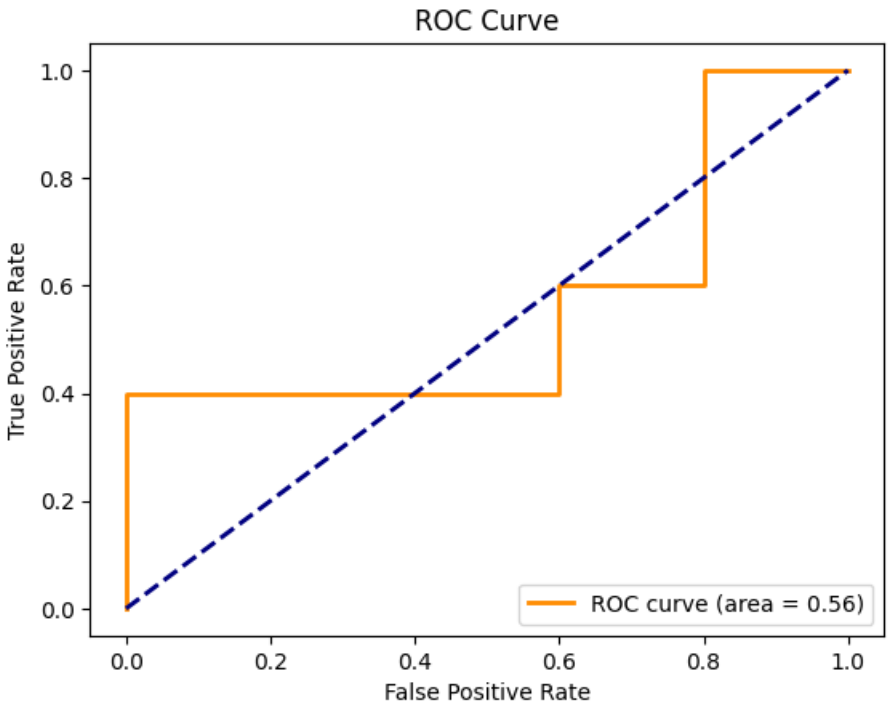
Task 1.2: color histogram

Figure 22: color moment histogram

Task 2:

Task 2.1: bin=120

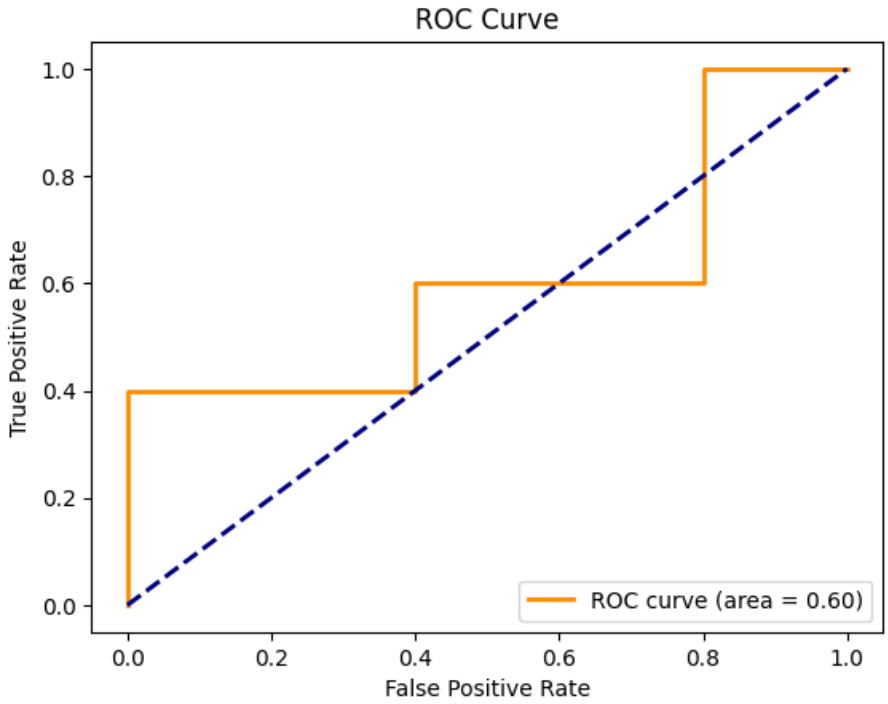
For bin size=120 and filename=9.jpg
Inner average precision = 0.12189724723625447
Inner average recall = 0.45199999999999996
Inner average f1 score = 0.14483814415339838



Area under curve = 0.56

Figure 23 image9(120)

For bin size=120 and filename=7.jpg
Inner average precision = 0.12606391390292115
Inner average recall = 0.45999999999999996
Inner average f1 score = 0.15061592193117618

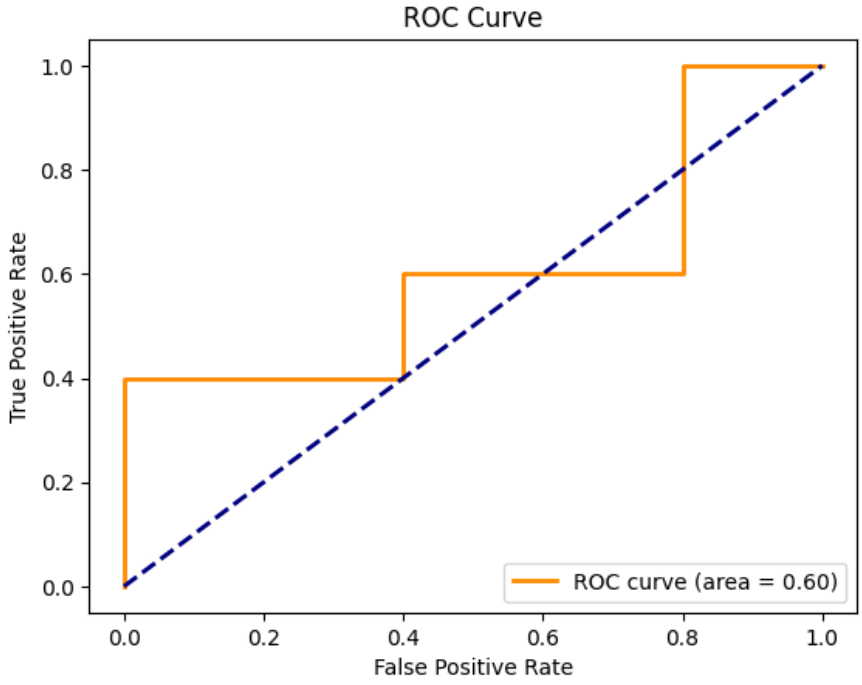


Area under curve = 0.6

Figure 24 image 7(120)

Task 2.2: bin=180

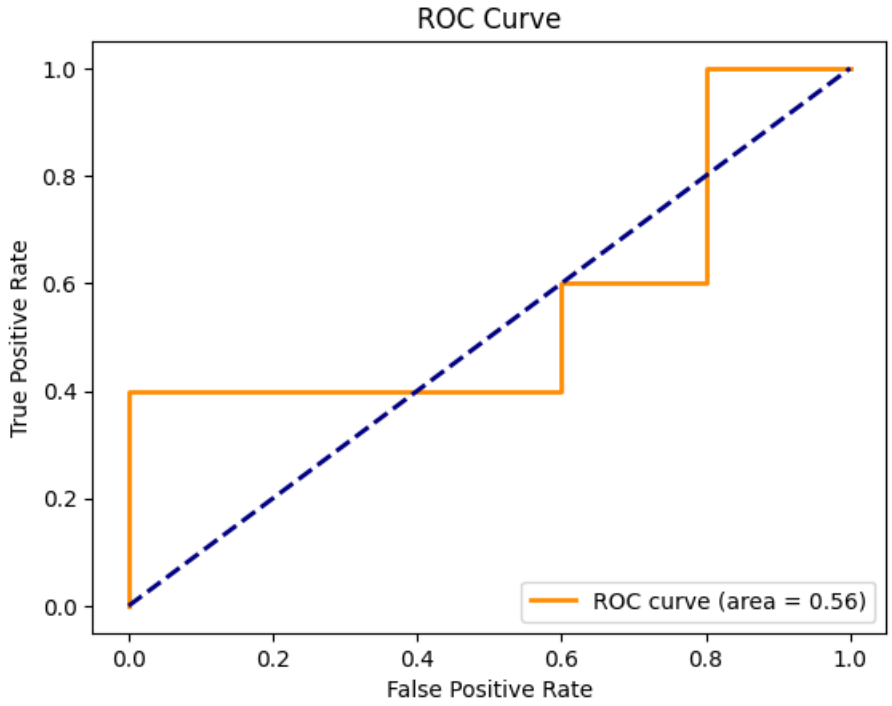
For bin size=180 and filename=7.jpg
Inner average precision = 0.15273058056958783
Inner average recall = 0.4800000000000001
Inner average f1 score = 0.17083814415339844



Area under curve = 0.6

Figure 23 image 7 (bin=180)

For bin size=180 and filename=9.jpg
Inner average precision = 0.12806391390292113
Inner average recall = 0.45600000000000007
Inner average f1 score = 0.14990481082006507



Area under curve = 0.56

Figure 24 image 9 (bin=180)

Task 2.3: bin=256

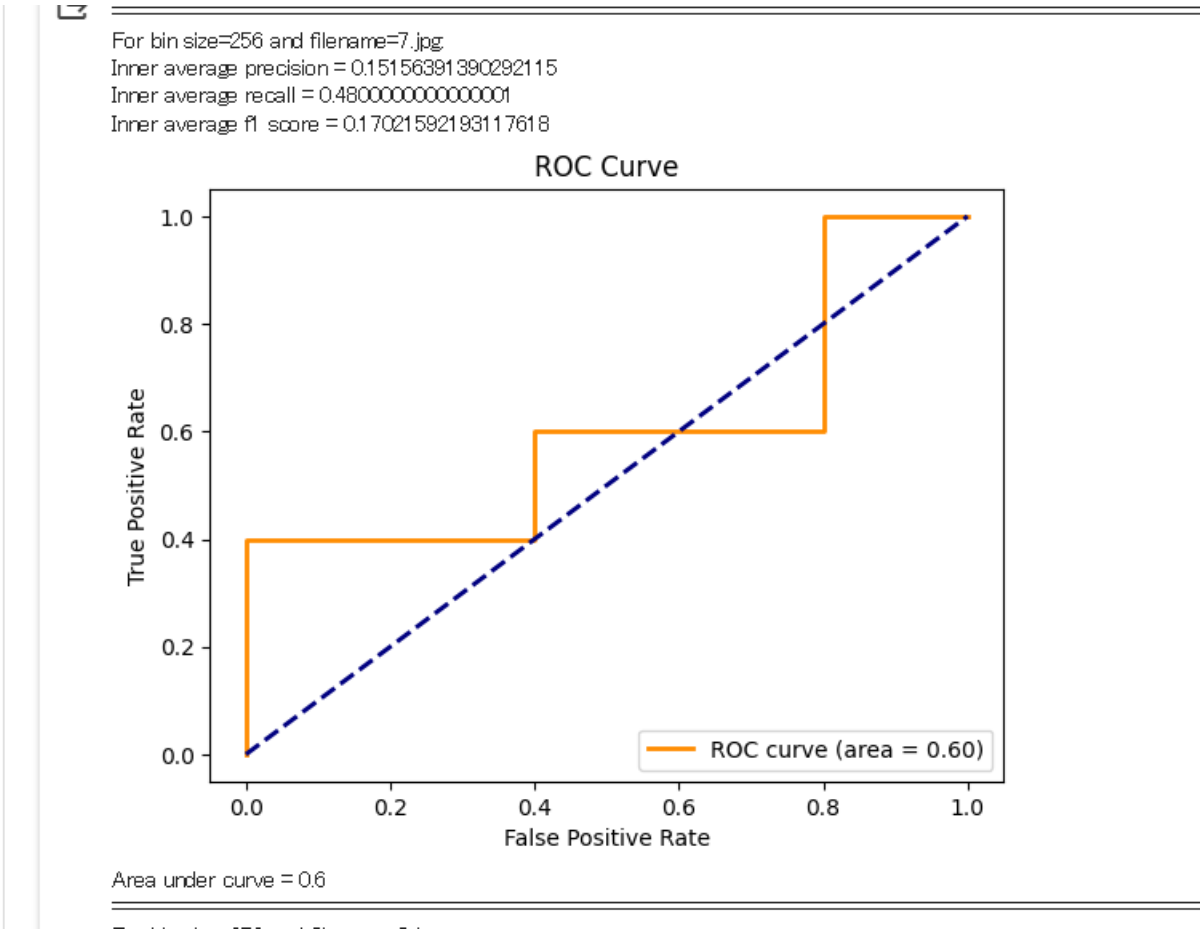


Figure 25 image 7(bin=256)

As we shown used different bins effects on each precision, recall, f1 So that it improves them .

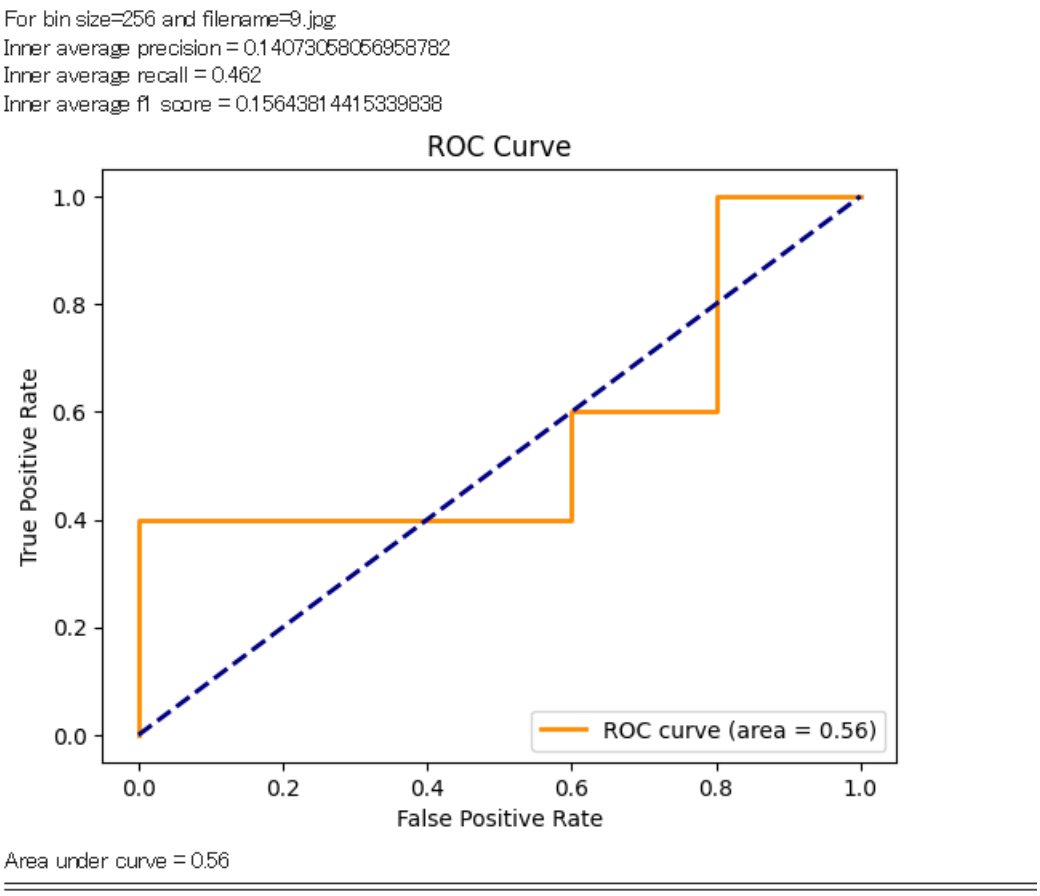


Figure 26 image 9 (bin=256)

As we shown the similarity is computed based on the distribution of pixel intensities. Images with similar color distributions will have lower Euclidean distances.

So in the selection of threshold values and bin sizes significantly influences performance metrics. Higher thresholds generally increase recall but may reduce precision, while optimal bin sizes strike a balance between capturing color details and avoiding noise. Precision benefits from smaller bins, especially in datasets where fine color nuances matter. Larger bins enhance recall by generalizing color features, but they may miss subtle differences. The F1 score seeks a balance between precision and recall, peaking at an optimal threshold. The AUC, less sensitive to threshold changes, gauges overall model performance.

Task 3

Task 3.1

```
Inner average precision = 0.34994756632256635
Inner average recall = 0.404
Inner average f1 score = 0.36893567354558066
Area under curve = 0.006649305884491469
area under curve=0.03799802177171629
==============
Inner average precision = 0.005614468864468865
Inner average recall = 0.008
Inner average f1 score = 0.006583784331610418
Area under curve = 0.03799802177171629
area under curve=0.0009861031418886
==============
Inner average precision = 0.04652097902097903
Inner average recall = 0.05600000000000001
Inner average f1 score = 0.050039730888754984
Area under curve = 0.0009861031418886
area under curve=0.006649305884491469
==============
Inner average precision = 0.0
Inner average recall = 0.0
Inner average f1 score = 0.0
Area under curve = 0.006649305884491469
area under curve=0.0
==============
Inner average precision = 0.01805841380841381
Inner average recall = 0.022
Inner average f1 score = 0.019557866639425398
Area under curve = 0.0
area under curve=0.0027281430911055247
==============
Inner average precision = 0.004066849816849817
Inner average recall = 0.006
Inner average f1 score = 0.004841360089186176
Area under curve = 0.0027281430911055247
area under curve=0.0006979184046751895
==============
Inner average precision = 0.11358186258186258
Inner average recall = 0.13599999999999998
Inner average f1 score = 0.12199204273837468
Area under curve = 0.0006979184046751895
area under curve=0.015188031415115606
```

Figure 27: task3.1

```
==============
Inner average precision = 0.5875500055500056
Inner average recall = 0.647
Inner average f1 score = 0.6053862222311273
Area under curve = 0.015188031415115606
area under curve=0.07387702639309843
==============
Inner average precision = 0.002630952380952381
Inner average recall = 0.004
Inner average f1 score = 0.003171794871794872
Area under curve = 0.07387702639309843
area under curve=0.0004097336674617791
==============
Inner average precision = 0.07741583416583415
Inner average recall = 0.094
Inner average f1 score = 0.0837341408012427
Area under curve = 0.0004097336674617791
area under curve=0.01085063051560543
==============
for weights=[1, 1, 1, 0, 0, 0]: average precision=0.12053869325119326
        average recall=0.1377
        average f1 score=0.12642426161370973
        average time=24.9527188539505
        average AUC=0.0
```

Figure 28: task 3.1

which are actually relevant. It is a measure of exactness Precision means that an algorithm returned most of the relevant results. More is the value of precision, better is the prediction made for the code which explains the reduction of semantic gap. Recall is the percentage of all relevant documents which show up in the retrieved set. Recall is a measure of completeness or quantity. The performance of the system is evaluated after analyzing the retrieved results we concluded that proposed method is effective to reduce the semantic gap and indexed the retrieved images also two histogram the same for two different images

Task 3.2

```
area under curve=0.04667654867846143
==============
Inner average precision = 0.0020059523809523813
Inner average recall = 0.003
Inner average f1 score = 0.0024025641025641024
Area under curve = 0.04667654867846143
area under curve=0.0003543735616693965
==============
Inner average precision = 0.0306494338994339
Inner average recall = 0.036
Inner average f1 score = 0.032577785871567
Area under curve = 0.0003543735616693965
area under curve=0.0044415077641414644
==============
Inner average precision = 0.000625
Inner average recall = 0.001
Inner average f1 score = 0.0007692307692307692
Area under curve = 0.0044415077641414644
area under curve=6.639645056289265e-05
==============
Inner average precision = 0.014207473082473082
Inner average recall = 0.017
Inner average f1 score = 0.015224265457998126
Area under curve = 6.639645056289265e-05
area under curve=0.0021176389012382084
==============
Inner average precision = 0.0
Inner average recall = 0.0
Inner average f1 score = 0.0
Area under curve = 0.0021176389012382084
area under curve=0.0
==============
Inner average precision = 0.1263616244866245
Inner average recall = 0.148
Inner average f1 score = 0.13407016691897564
Area under curve = 0.0
area under curve=0.015927757134632757
==============
```

Figure 29: task 3.2

```
Inner average f1 score = 0.0
Area under curve = 0.07281065582703034
area under curve=0.0
==============
Inner average precision = 0.06262635975135974
Inner average recall = 0.076
Inner average f1 score = 0.06780062758617073
Area under curve = 0.0
area under curve=0.008984140056168809
==============
for weights=[0.2, 0.3, 0.5, 0, 0, 0]: average precision=0.13053043761793762
        average recall=0.1479
        average f1 score=0.1363798692290495
        average time=25.01741898059845
        average AUC=0.0
```

Figure 30: task 3.2

As we notice the different between task 3.1 and 3.2 the average recall , f1 score and time increase behind the task3.1 also the skewness Improvement works better than others.

Task 3.3

```
Inner average precision = 0.43935051060051056
Inner average recall = 0.497
Inner average f1 score = 0.4588576343863731
Area under curve = 0.008984140056168809
area under curve=0.04876090683257019
==============
Inner average precision = 0.0
Inner average recall = 0.0
Inner average f1 score = 0.0
Area under curve = 0.04876090683257019
area under curve=0.0
==============
Inner average precision = 0.03476268176268176
Inner average recall = 0.046
Inner average f1 score = 0.039280054067468255
Area under curve = 0.0
area under curve=0.005363688558739869
==============
Inner average precision = 0.0019166666666666668
Inner average recall = 0.003
Inner average f1 score = 0.0023384615384615384
Area under curve = 0.005363688558739869
area under curve=0.0002657451119085272
==============
Inner average precision = 0.07157726995226996
Inner average recall = 0.09000000000000001
Inner average f1 score = 0.07893166962261053
Area under curve = 0.0002657451119085272
area under curve=0.010619552121039617
==============
```

Figure 31: task 3.3

```
==============
Inner average precision = 0.7143777611277612
Inner average recall = 0.792
Inner average f1 score = 0.7387087768126939
Area under curve = 0.04124758159994205
area under curve=0.08285036491697234
==============
Inner average precision = 0.024658383283383285
Inner average recall = 0.034999999999999996
Inner average f1 score = 0.028841399180529616
Area under curve = 0.08285036491697234
area under curve=0.0035797488373989564
==============
Inner average precision = 0.2619688922188922
Inner average recall = 0.303
Inner average f1 score = 0.2766300659974775
Area under curve = 0.0035797488373989564
area under curve=0.03146046476745475
==============
for weights=[0.1, 0.4, 0.3, 0.1, 0.1, 0.2]: average precision=0.1914246864246864
        average recall=0.2186
        average f1 score=0.20083295147009106
        average time=25.48657968044281
        average AUC=0.0
```

As we notice when we add Median, Mode, and Kurtosis. Into task 3.2 the value become improvement compare with task3.2

Task 4

The Scale-Invariant Feature Transform (SIFT) algorithm can be integrated into image retrieval system to potentially enhance the performance of the evaluate_performance function. SIFT is particularly effective in detecting and describing local features in images, which can improve the accuracy of image comparison and retrieval.

```
Average Precision: 0.0867
Average Recall: 0.1386
Average F1 Score: 0.10063238095238095
Average Time: 56.58829426765442
Average AUC: 0.036166520904161094
```

Figure 32 task4

As we shown the Integrating SIFT into the image retrieval system enhances its ability to identify similar images under more challenging conditions, potentially improving precision, recall, F1,time, auc and overall robustness. However, this comes at the cost of increased computational complexity. The choice between SIFT and simpler methods like color moments depends on the specific requirements and constraints of the application, including the nature of the image dataset and the computational resources available.

Conclusion

In this novel approach to Content-Based Image Retrieval (CBIR) systems, a distinctive method is proposed by seamlessly integrating histogram and color moment techniques. The synergy of these two features in the retrieval process demonstrates significantly improved precision compared to previous methodologies, particularly when dealing with clear target images. The retrieval outcomes not only showcase enhanced accuracy but also serve as a valuable indicator of the algorithm's overall efficiency. Notably, the integration of Color Moments and Color Histogram proves to be particularly effective, leveraging the perceptual relevance of the HSV (Hue, Saturation, Value) histogram to the human vision system. Overall, this innovative combination of features contributes to a more advanced and refined CBIR system, setting a new standard for image retrieval precision.

References

[1] M.J.Swain, D.H.Ballard, ‘‘Color indexing’’, International Journal of Computer Vision, vol. 7, no. 1, pp. 11-32, 1991.

[2] A.Vellaikal and C.C.J.Kuo, ‘‘Content based image retrieval using multiresolution histogram representation’’, SPIE - Digital Image Storage and Archiving Systems, vol. 2606, pp. 312-323, 1995.