

Content Based Image Retrieval

A project submitted in partial fulfilment of the requirement for the award of the degree

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE and ENGINEERING

Submitted By

Gudena Dileep
(1210310424)

K. Abhinav
(1210310425)

Edukulla Harish
(1210310420)

Reyya Shiva Kumar
(1210310453)

Under the Esteemed Guidance of

Sri K. Narasimha Raju

Assistant Professor

Department Of Computer Science and Engineering



Department Of Computer Science and Engineering
GITAM Institute of Technology
(Est. u/s of UGC act 1956)
Visakhapatnam-530045
2010-2014

Department Of Computer Science and Engineering
GITAM Institute of Technology
GITAM University
(Est. u/s of UGC act 1956)
Visakhapatnam-530045



CERTIFICATE

This is to certify that the project work entitled “**CONTENT BASED IMAGE RETRIEVAL**” is the bonafide work done by GUDENA DILEEP(1210310424), K. ABHINAV(1210310425), EDUKULLA HARISH(1210310420), REYYA SHIVA KUMAR(1210310453) during 2010-2014, in the Department Of Computer Science and Engineering, GITAM, in the partial fulfilment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY (COMPUTER SCIENCE AND ENGINEERING).

Head of the Department

Project Guide

Dr. P.V. NAGESWARA RAO

K. NARASIMHA RAJU

Professor
CSE department
GITAM University
Visakhapatnam

Assistant Professor
CSE department
GITAM University
Visakhapatnam

DECLARATION

We, (GUDENA DILEEP, K. ABHINAV, EDUKULLA HARISH, REYYA SHIVA KUMAR) hereby declare that the project report entitled “**CONTENT BASED IMAGE RETRIEVAL**” is an original and authentic work done in the Department of Computer Science and Engineering, GITAM, Rushikonda, Visakhapatnam, submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering. The matter embodied in this project work has not been submitted earlier for award of any degree or diploma to the best of our knowledge.

Registration No	Name	Signature
1210310424	GUDENA DILEEP	
1210310425	K. ABHINAV	
1210310420	EDUKULLA HARISH	
1210310453	REYYA SHIVA KUMAR	

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without mentioning the people who made it possible and whose constant guidance and encouragement crown all the efforts with success.

We take this opportunity to express our profound gratitude and deep regards to our guide **Sri K. NARASIMHA RAJU**, of the Department of Computer Science and Engineering, for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry us a long way in the journey of life on which we are about to embark.

We feel elated to extend our gratitude to **Dr. P.V. NAGESWARA RAO**, Head of the Department, Computer Science and Engineering Department, for his constant guidance and encouragement all the way during this project.

We would also like to thank each and every faculty member who has been a part of our learning experience throughout this degree.

CONTENTS

1.ABSTRACT.....	02
2.INTRODUCTION.....	03
2.1.APPLICATIONS OF CBIR	
2.2.CBIR SYSTEMS	
3.SOFTWARE PROCESS MODEL.....	05
4.REQUIREMENTS ELICITATION.....	07
4.1.EXISTING SYSTEM	
4.2.PROBLEMS IN EXISTING SYSTEM	
4.3.PROPOSED SYSTEM	
4.4.FEASIBILITY REPORT	
5.SOFTWARE REQUIREMENTS SPECIFICATION...10	
6.OBJECT ORIENTED DESIGN.....	14
6.1.DESIGN PROCESS	
6.2.UML DIAGRAMS	
7.ALGORITHMS.....	28
8.BACKEND.....	31
9.CODE.....	32
10. TESTCASES.....	53
11. CONCLUSION.....	56
12. REFERENCES.....	57

1.ABSTRACT

There is a need to find a desired image from a collection of images shared by many professional groups including journalists, design engineers and art historians while the requirements of image users can vary considerably. It can be useful to characterize image queries into three levels of abstraction primitive features such as colour or shape, logical features such as the identity of object shown and abstract attributes such as significance of the scenes depicted. While Content Based Image Retrieval (CBIR) systems currently operate effectively at the lowest of these levels satisfying most users.

The project consists of two phases. The first phase is creating a database of an image collection. The inputs to the system are the collection of images. The system extracts colour properties from all the images and stores them in a database.

The second phase consists of presenting a query to the system. Here the query is an example image. The system extracts colour features from this query and compares with data in the database. The output of this phase retrieves top 5 best matching images for the given image. In the two phases, a histogram based method and a colour coherence vector are used for comparing images.

2.INTRODUCTION

As processors become increasingly powerful, and memories become increasingly cheaper, the deployment of large image databases for a variety of applications have now become realizable. Databases of art works, satellite and medical imagery have been attracting more and more users in various professional fields for example, geography, medicine, architecture, advertising, design, fashion, and publishing. Effectively and efficiently accessing desired images from large and varied image databases is now a necessity.

Definition

CBIR or **Content Based Image Retrieval** is the retrieval of images based on visual features such as color, texture and shape [2]. Reasons for its development are that in many large image databases, traditional methods of image indexing have proven to be insufficient, laborious, and extremely time consuming. These old methods of image indexing, ranging from storing an image in the database and associating it with a keyword or number, to associating it with a categorized description, have become obsolete. This is not C B I R . In CBIR, each image that is stored in the database has its features extracted and compared to the features of the query image. It involves two steps.

- **Feature Extraction** The first step in the process is extracting image features to a distinguishable extent.
- **Matching** The second step involves matching these features to yield a result that is visually similar.

2.1 Applications of CBIR

Examples of CBIR applications are

- **Crime prevention** Automatic face recognition systems, used by police forces.
- **Security Check** Finger print or retina scanning for access privileges.
- **Medical Diagnosis** Using CBIR in a medical database of medical images to aid diagnosis by identifying similar past cases.
- **Intellectual Property** Trademark image registration, where a new candidate mark is compared with existing marks to ensure no risk of confusing property ownership.

2.2 CBIR Systems

Several CBIR systems currently exist, and are being constantly developed.

Examples are

- QBIC or Query By Image Content was developed by IBM, Almaden Research Centre, to allow users to graphically pose and refine queries based on multiple visual properties such as colour, texture and shape. It supports queries based on input images, user-constructed sketches, and selected colour and texture patterns.
- VIR Image Engine by Virage Inc., like QBIC, enables image retrieval based on primitive attributes such as colour, texture and structure. It examines the pixels in the image and performs an analysis process, deriving image characterization features.
- VisualSEEK and WebSEEK were developed by the Department of Electrical Engineering, Columbia University. Both these systems support colour and spatial location matching as well as texture matching.
- NeTra was developed by the Department of Electrical and Computer Engineering, University of California. It supports colour, shape, spatial layout and texture matching, as well as image segmentation.
- MARS or Multimedia Analysis and Retrieval System are developed by the Beckman Institute for Advanced Science and Technology, University of Illinois. It supports colour, spatial layout, texture and shape matching.

3. Software Process Model

Introduction

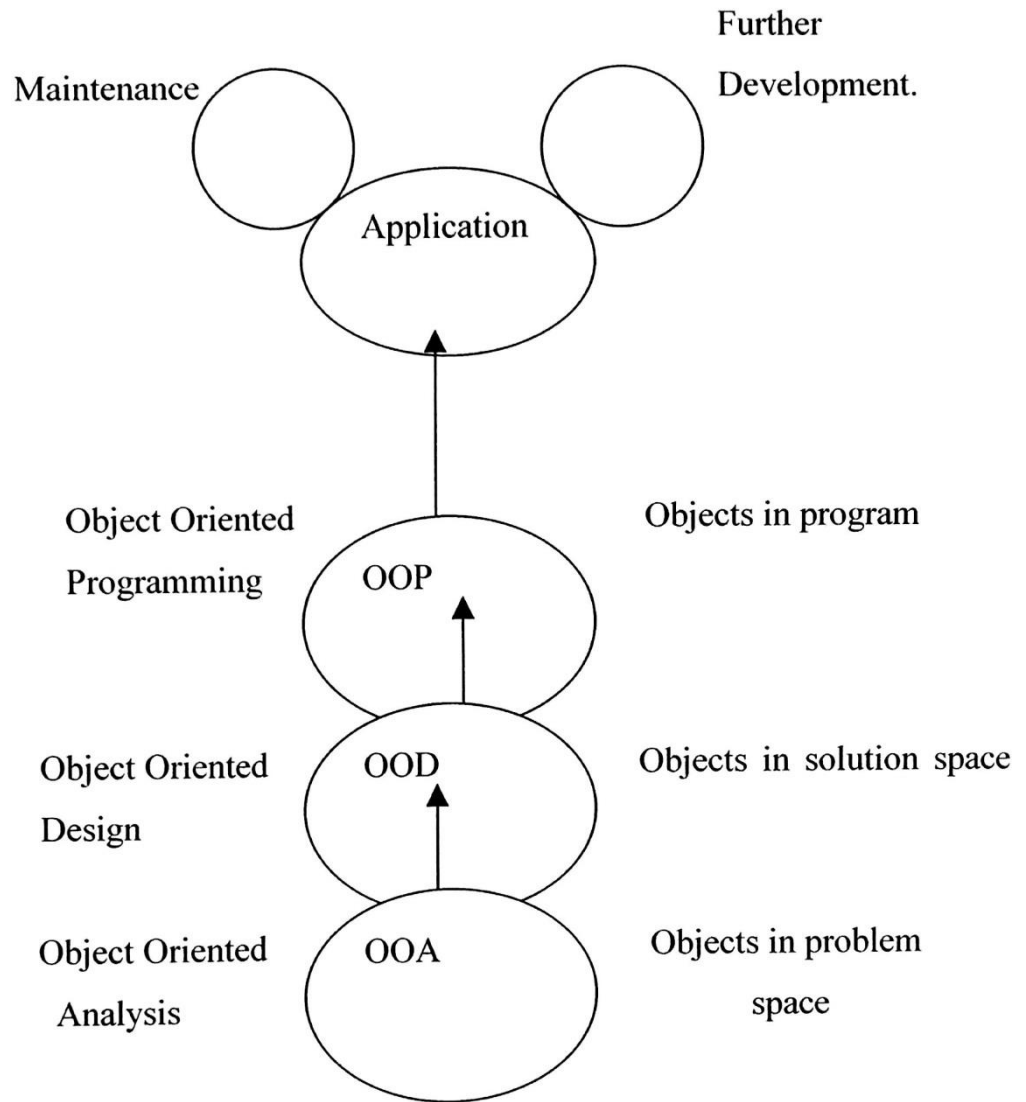
A software model is an abstract representation of software process. There are many models available for the user as object oriented model, procedure oriented model (waterfall model), evolutionary model, formal system based model, etc. These models are not the definite description of the software process; rather, they are useful abstractions, which can be used to explain different approaches to software development.

Fountain Model

The object oriented places greater emphasis on the objects that encapsulate data procedures. They play a vital role in all the stages of the software development. Therefore there exists a high degree of overlap and iteration between the stages. The entire development model is evolutionary in nature and the resulting model is called “Fountain model”. This model depicts that the development reaches a higher level only to fall back to a previous level and then again climbing up.

The Fountain model can be depicted from the following figure. Object oriented analysis (OOA) refers to the methods of specifying requirements of the software in terms of real world objects, their behaviour and their interactions. Object oriented design (OOD), on the other hand, turns the software requirements into specifications for objects and derives class hierarchies from which objects can be created. Finally, object oriented programming (OOP) refers to the implementation of the program using objects, in an object oriented programming language such as java, C++.

Fountain Model



4.REQUIREMENTS ELICITATION

4.1 EXISTING SYSTEM

Currently there are two existing systems image matching and retrieval techniques

- The conventional method of image indexing, ranging from storing an image in the database and associating it with a keyword or number, to associating it with a categorized description, has become obsolete.
- Colour histograms are used to compare images in many applications. Their advantages are efficiency, and insensitivity to small changes in camera viewpoint. However, color histograms lack spatial information, so images with very different appearances can have similar histograms. For example, a picture of fall foliage might contain a large number of scattered red pixels; this could have a similar color histogram to a picture with a single large red object.

4.2 PROBLEMS IN EXISTING SYSTEM

- If the keywords associated with stored images are not relevant in any way then the matching techniques would fail to function.
- Images with very different appearances can have similar histograms.

4.3 PROPOSED SYSTEM

We describe a colour histogram-based method for comparing images that incorporates spatial information. We classify each pixel in a given colour bucket as either coherent or incoherent, based on whether or not it is part a large similarly-coloured region. A colour coherence vector (CCV) stores the number of coherent versus incoherent pixels with each colour. By separating coherent pixels from incoherent pixels; CCV's provide finer distinctions than colour histograms. Intuitively, we define a colour's coherence as the degree to which pixels of that colour are members of large similarly-coloured regions. We refer to these significant regions as coherent regions, and observe that they are of significant importance in characterizing images.

ADVANTAGES

Advantages can be explained clearly by considering the following figures-



The images shown in the above figures have similar colour histograms, despite their rather different appearances. The colour red appears in both images in approximately the same quantities. In the left image the red pixels (from the flowers) are widely scattered, while in the right image the red pixels (from the golfer's shirt) form a single coherent region. Our coherence measure classifies pixels as either coherent or incoherent.

4.4 FEASIBILITY REPORT

For all new systems, the requirements engineering process should start with a feasibility study. The input to the feasibility study is an outline description of the system and how it will be used within an organization. Results of the feasibility study should be a report, which whether or not it is worth carrying on with the requirements engineering and system development process. Carrying out a feasibility study involves information assessment, information collection and report writing.

TECHNICAL FEASIBILITY

System should be equipped with PYTHON 2.7 and MYSQL 5.6 or their upgraded versions.

ECONOMIC FEASIBILITY

The system does not demand any expensive technologies. IT also does not require much of the computer resources. Hence, the implementation of the

system does not include any economic hurdles and making it feasible for implementation.

OPERATION FEASIBILITY

The proposed system does not require any knowledge of how the system works and no training is required to operate for general users.

5.SOFTWARE REQUIREMENT SPECIFICATION

Purpose

The purpose of the system is to retrieve images which are perceptually similar to the given query (here query is an example image). While performing the retrieval we give priority to the human perceptual tendency to recognize large areas of similar colour as dominant regions.

Scope

The scope of the system is storing of images into the databases by authorized users and querying the database with an example image.

Abbreviations

Authorized User

He is the one who has privileges to store the images in to the database.

General User

He is the one who queries the databases with an example image. Any user can come under this.

REQUIREMENTS

Software systems requirements are often classified as functional or non- functional.

Functional Requirements

These are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.

Non-Functional Requirements

These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process, standards, etc.

Scenario Technique for Requirement Elicitation

Scenario can be particularly useful for adding detail to an outline requirements description. The scenario starts with an outline of that interaction. A scenario may include

- 1 A system state description at the beginning of the scenario.
- 2 A description of the normal flow of events in the scenario.
- 3 A description of what can go wrong and how this is handled.
- 4 Information about other activities which might be going on at the same time.
- 5 A description of the state of the system after the completion of the scenario.

Use cases

Use cases are a scenario-based technique for requirements elicitation which were first introduced in the Objectory method. They have now become a fundamental feature of the UML notation for describing object-oriented system models.

NON-FUNCTIONAL REQUIREMENTS

Reliability

Our system is reliable. It cannot be prone to errors and not vulnerable to interceptors.

Efficiency

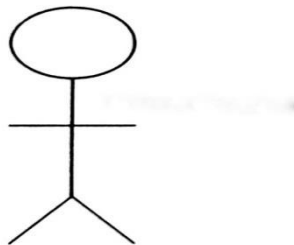
Our system takes linear time to search the entire database and responds very quickly to the user query.

REQUIREMENT MODEL

Actors

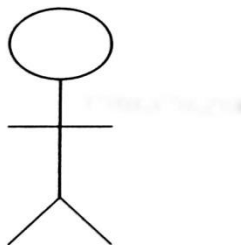
Authorized user

He is the one who has privileges to store the images into our databases.
He will give a collection of images to the system as input.



General User

He is the one who interacts with the system by giving an example Image as an input. The system will retrieve images which are best matched to the given image.



USE CASES

- 1. Read image.**
- 2. Coherence processing (calculating coherence vector(CCV)).**
- 3. Inserting**
- 4. Comparing**

1. Reading Image

The image given as input by the user is read into a buffer. The buffered image is blurred and entire range of colors is mapped into 64 colors.

2. Coherence processing

The coherent & in-coherent values for all the 64 colors of the buffered image are calculated and converting CCV into bytes.

3. Inserting

The CCV values extracted from the image along with the URL of the image are stored into the database.

4. Comparing

The CCV values of the query image are compared with the CCV values of image collection in the database and retrieve the best matched images.

6.OBJECT ORIENTED DESIGN

6.1 Design process

The general process used for object-oriented design has five stages.

1. Understanding and defining the context.
2. Design the system architecture.
3. Identify the principal objects in the system.
4. Develop design models.
5. Specify object interfaces.

6.1.1 Understanding and define the context

When we model the interactions of a system with its environment you should use an abstract approach that does not include too much detail of these interactions. The approach that is proposed in the UML is to develop a USE CASE model. Where each use case represents a interaction of the system. In use case models each possible interaction is named in an ellipse and the external entity involved in the interaction is represented by a stick figure.

6.1.2 Architectural Design

Design process of identifying the sub-systems and establishing a framework for sub-system control and communication is called architectural design. It is sub-divided into 3 parts. They are as follows

System Structuring

The system is structures into a number of principal sub-systems where a sub-system is an independent software unit. Communications between sub-systems are identified. The model that we are following to decompose a system into a set of interacting sub-systems is THE ABSTRACT MACHINE MODEL, also known as layered model. It organizes the system into a series of layers. Each of which provides a set of services. Each layer defines an abstract machine whose

machine language is used to implement the next level of abstract machine. The layered approach supports the incremental development of systems. This architecture is also changeable and portable.

Control Modelling

A general model of the control relationships between the parts of the system is established. The approach that we are using is event based control. The technique that we are using to implement event driven control model is BROADCASTING EVENT DRIVEN CONTROL MODEL. In this model, an event is, in principle, broadcast to all sub-system, which can handle that event, responds to it.

Modular Decomposition

Each identified sub-system is decomposed into modules. The architect must decide on the types of module and their interconnections. Here, we are using an object oriented model, by which we decompose the system into a set of communicating objects.

6.1.3 Object Identification

It is concerned with identifying object classes. The design is described in terms of these classes. Inevitably, we have to refine the object classes that we initially identify and revisit the stage of process as we develop deeper understanding of designing. Here we are using behavioural approach. The various behaviours are assigned to different parts of the system and an understanding is derived of who initiates and participates in this behaviour. Participants who play significant roles are recognized as objects.

6.1.4 Design Models

Design models show the objects or object classes in a system and, where appropriate, different kinds of relationships between these entities. These

models are the bridge between the requirements for the system and the system implementation. The various kinds of design models are

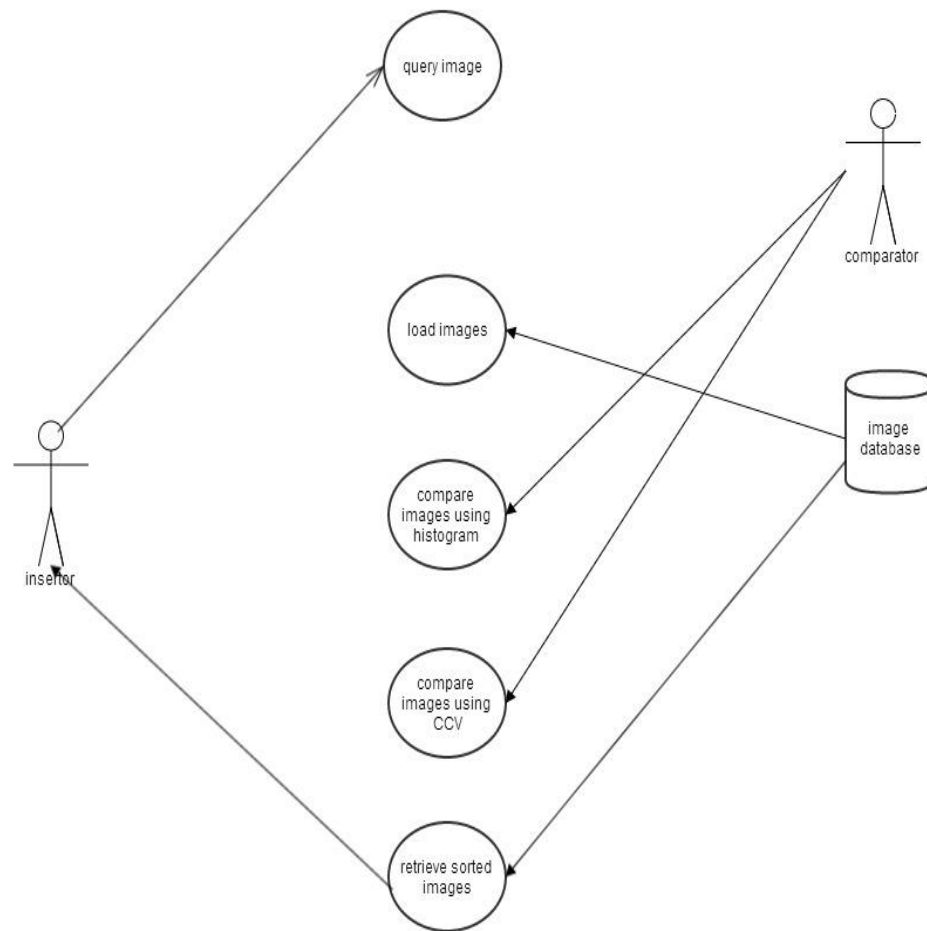
1. Sub-system models that show logical grouping of objects into coherent sub-system is shown as a package. These are static models.
2. Sequence models that show the sequence of object interactions. These are represented using a UML sequence or collaboration diagram. These are dynamic models.
3. State machine models that show how individual objects change the state in response to events. These are represented in UML using state chart diagrams. These are dynamic models.

6.1.5 Object interface specification

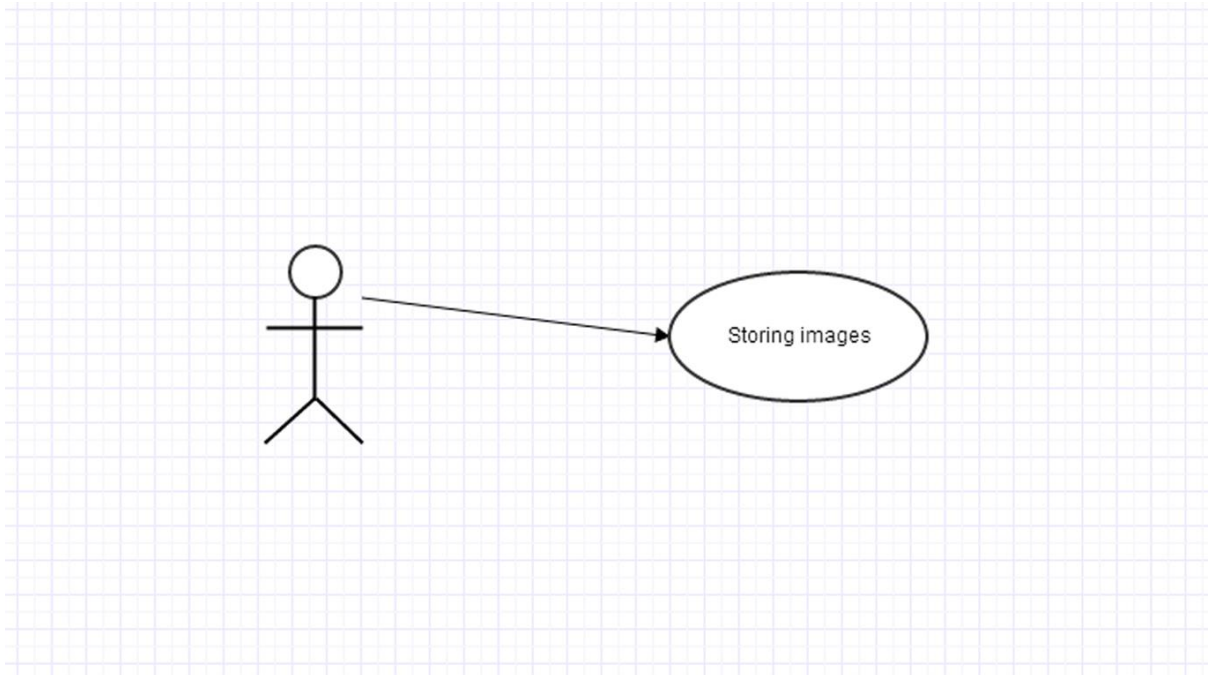
Object interface specification is important so that objects and other components can be designed in parallel. Interface representation information should be hidden and object operations provided to access and update the data. Hence it can be changed without affecting the objects that use these attributes making it more maintainable. Object interface design is concerned with specifying in detail of the interface to an object or to a group of objects. Interfaces in the UML can be specified using the same notation as in the class diagrams. Programming languages can also be used to define interface as an alternative approach.

6.2 UML DIAGRAMS

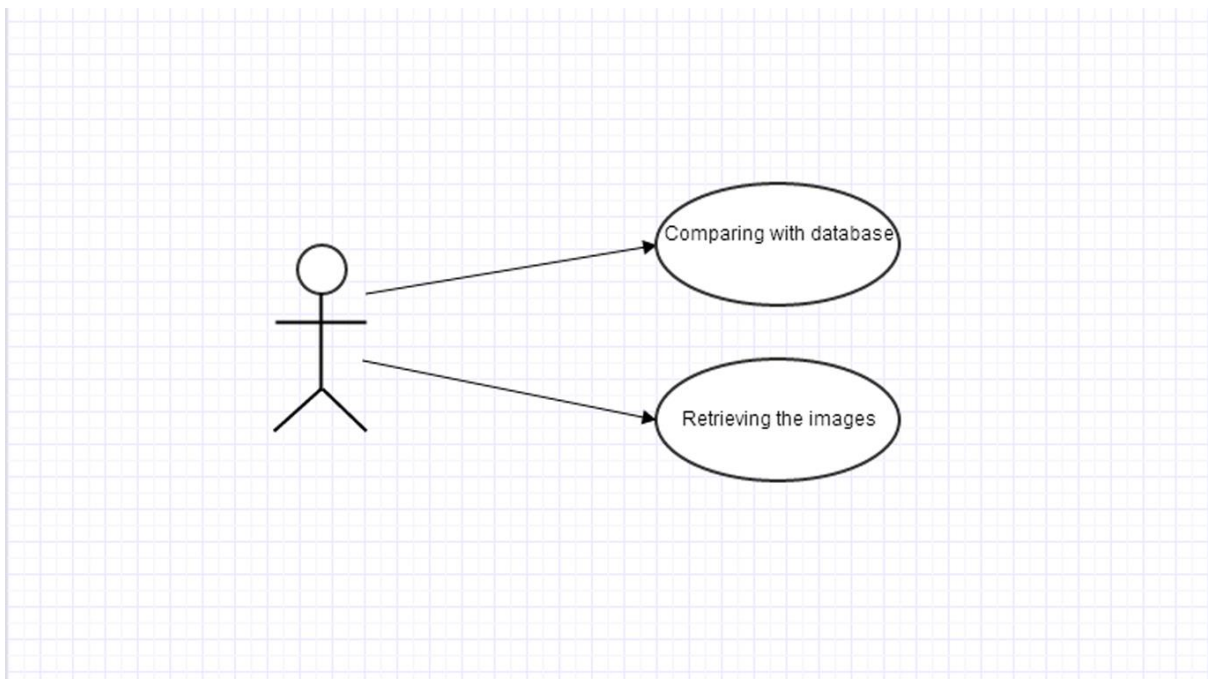
USECASE DIAGRAMS



INSERTOR



COMPARATOR



STATECHART DIAGRAMS

Detailed Description of State chart Diagrams

Read image

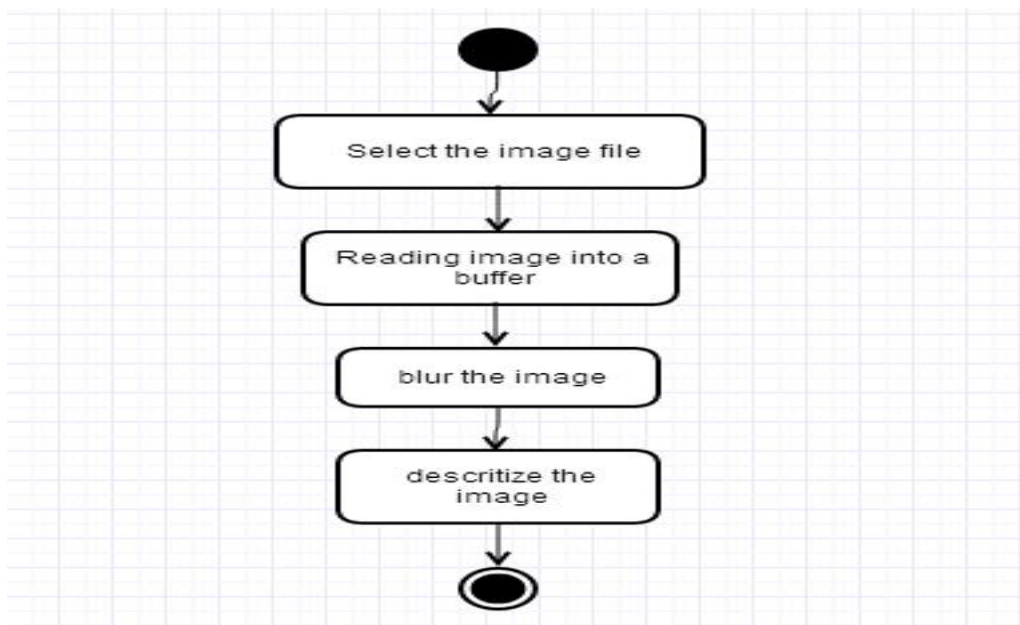
Pre-condition The program has to run on a machine having a python interpreter.

Main flow of events

1. The user browses for the query image.
2. If our system detects that the image selected is not supported by the system then the system prompts a message to select a valid image.
3. The image is read into a buffer.
4. Image is blurred.
5. Image is discretized.

Post-conditions The use case will end when the discretized image is read into a buffer.

State Chart diagram



Coherence Processing

Pre-conditions

Discretized image is in the buffer

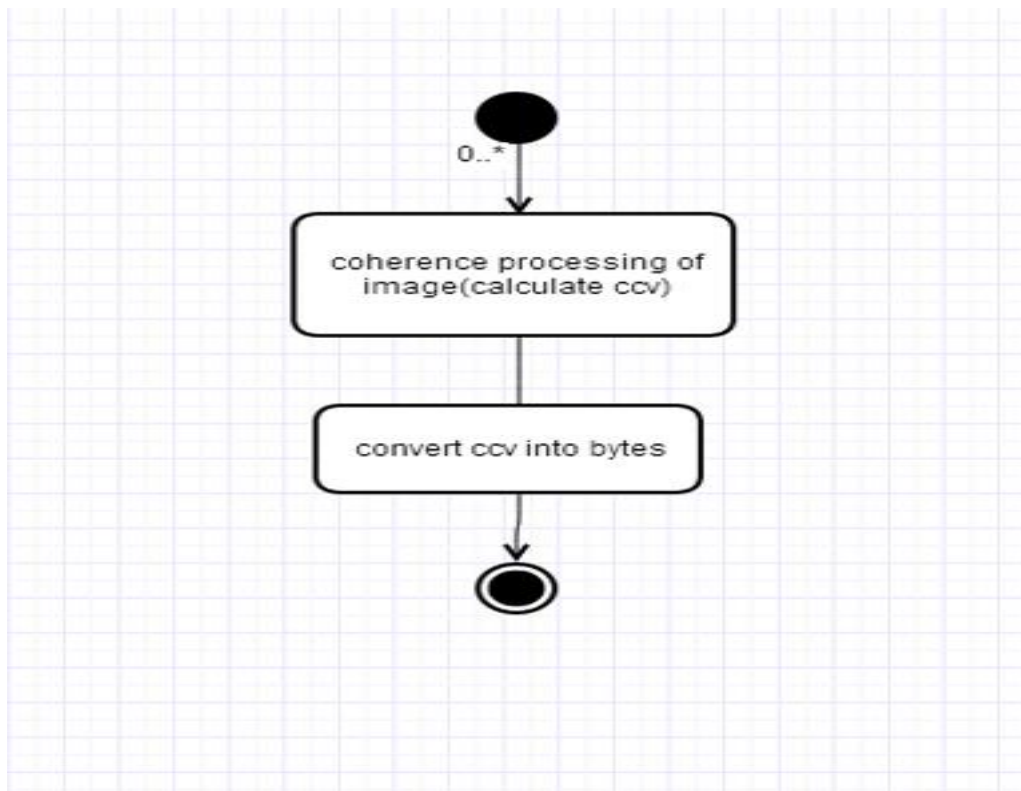
Main flow of events

1. Coherence processing of the image.
2. Convert CCV into bytes.

Post-conditions

The features extracted from the image are in the form of bytes.

State Chart diagram



Inserting

Pre-condition

CCV is calculated.

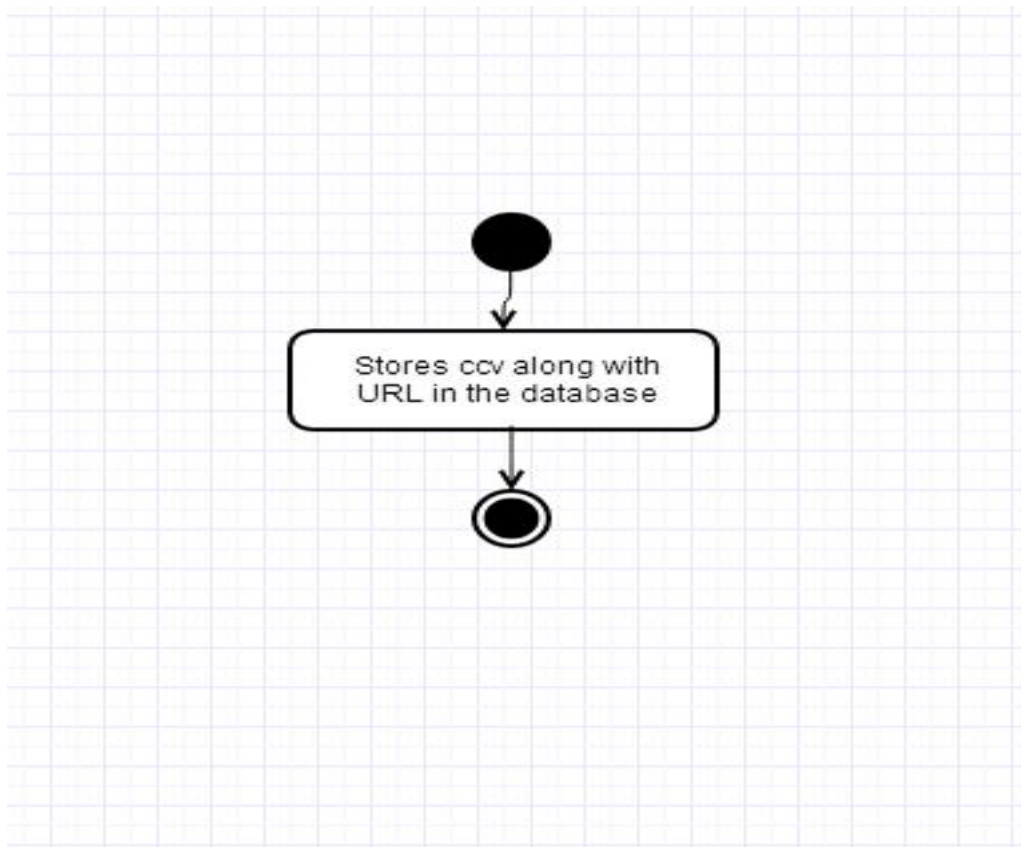
Main flow events

1. The CCV along with the image URL is stored in the database.

Post-condition

The features extracted from the image are stored into the database.

State Chart diagram



Comparing

Pre-condition

The program has to run on a machine having a java compiler.

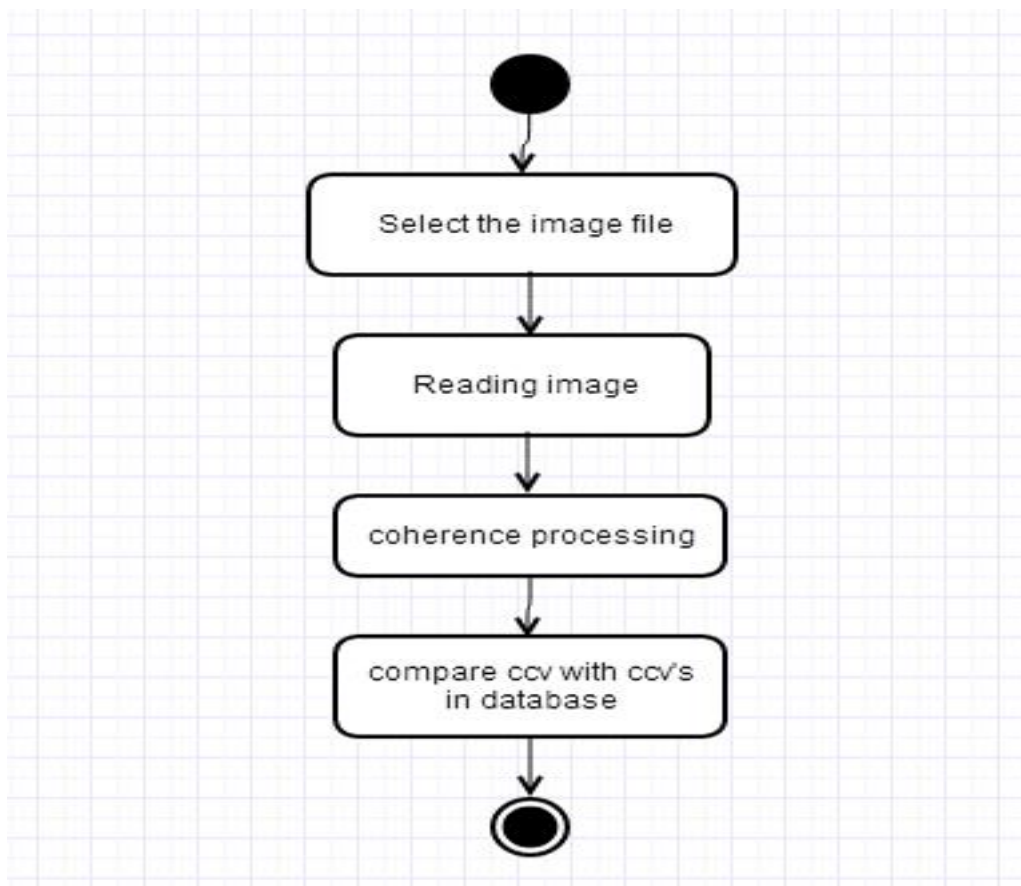
Main flow of events

1. The user browses for the query image.
2. Read image.
3. Coherence processing.
4. Compare the ccv with the ccv's in the database.
5. Retrieve the top matched images.

Post-conditions

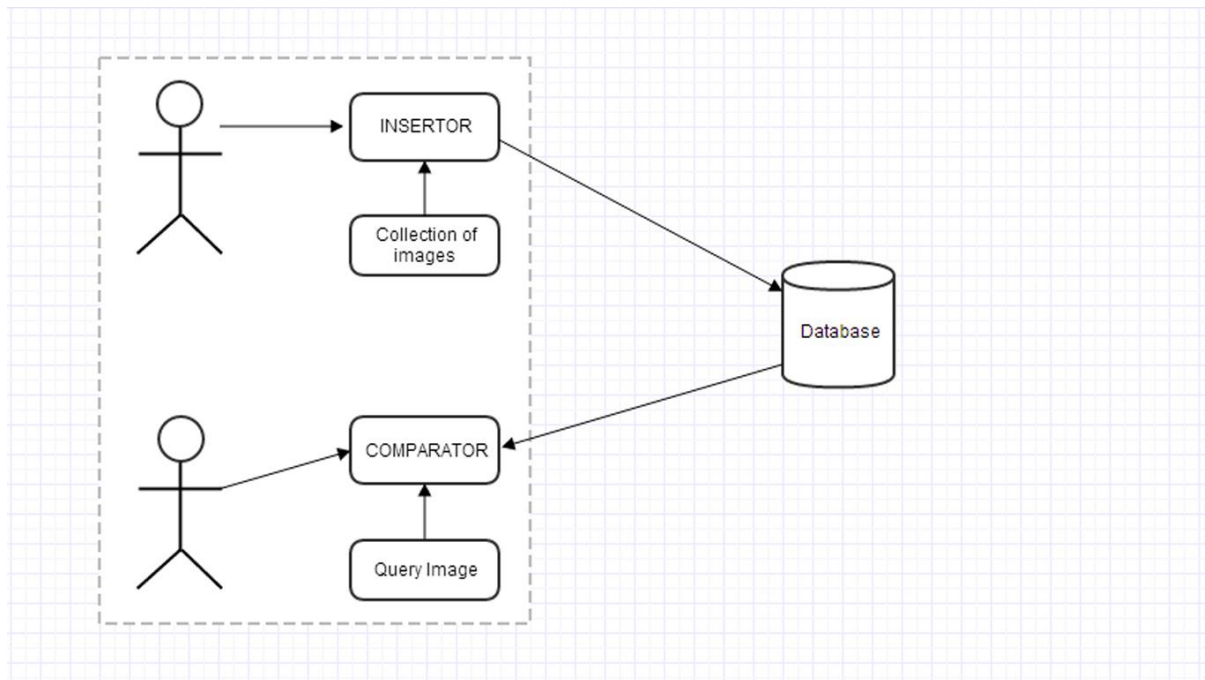
The best matched images are retrieved.

State chart diagram



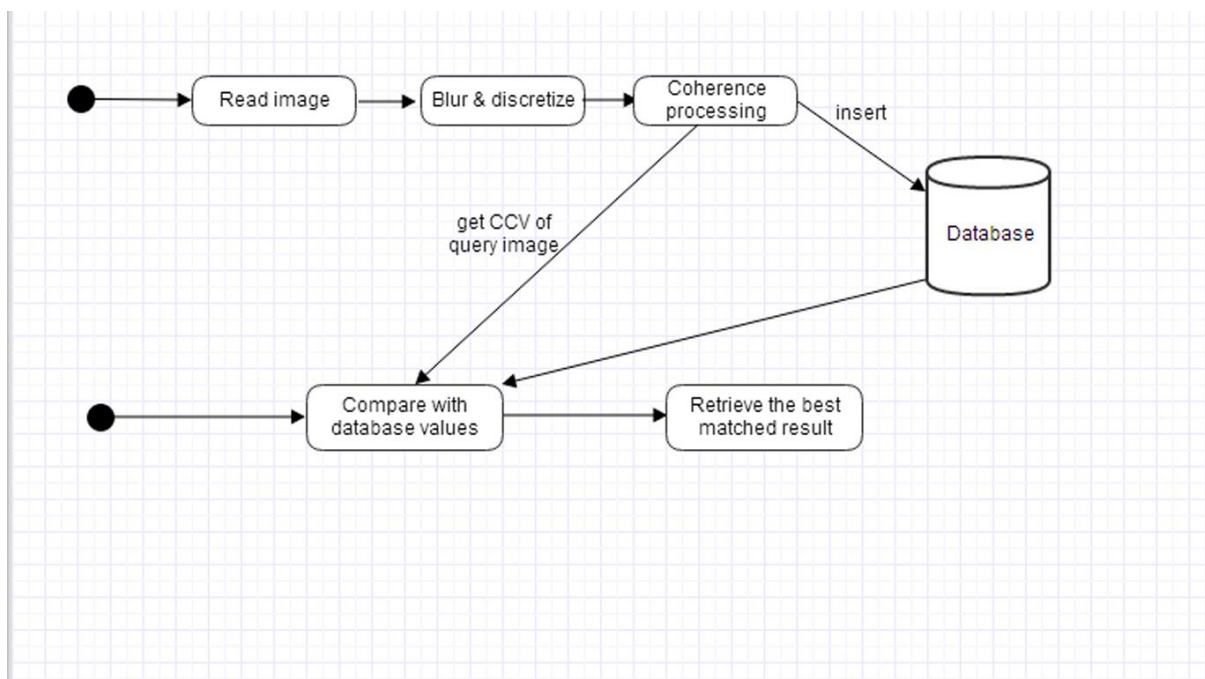
DESIGN MODEL

DEPLOYMENT MODEL

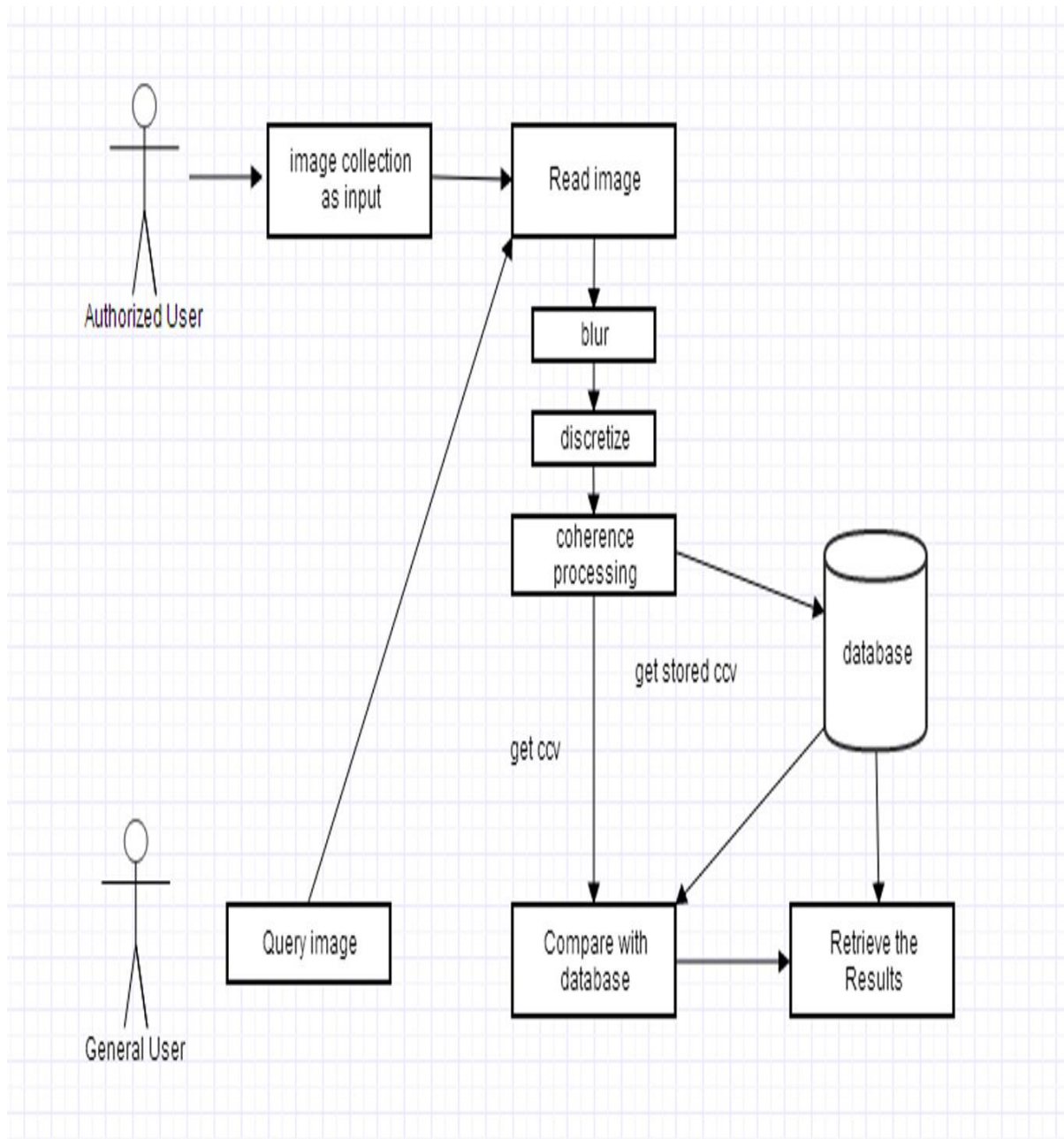


SUBSYSTEM INTERFACE

INSERTOR



GENERIC DESIGN MECHANISM



1. Input image file.
2. Read image.
3. Blur image.
4. Discretize.
5. Coherence processing.
6. Inserting.

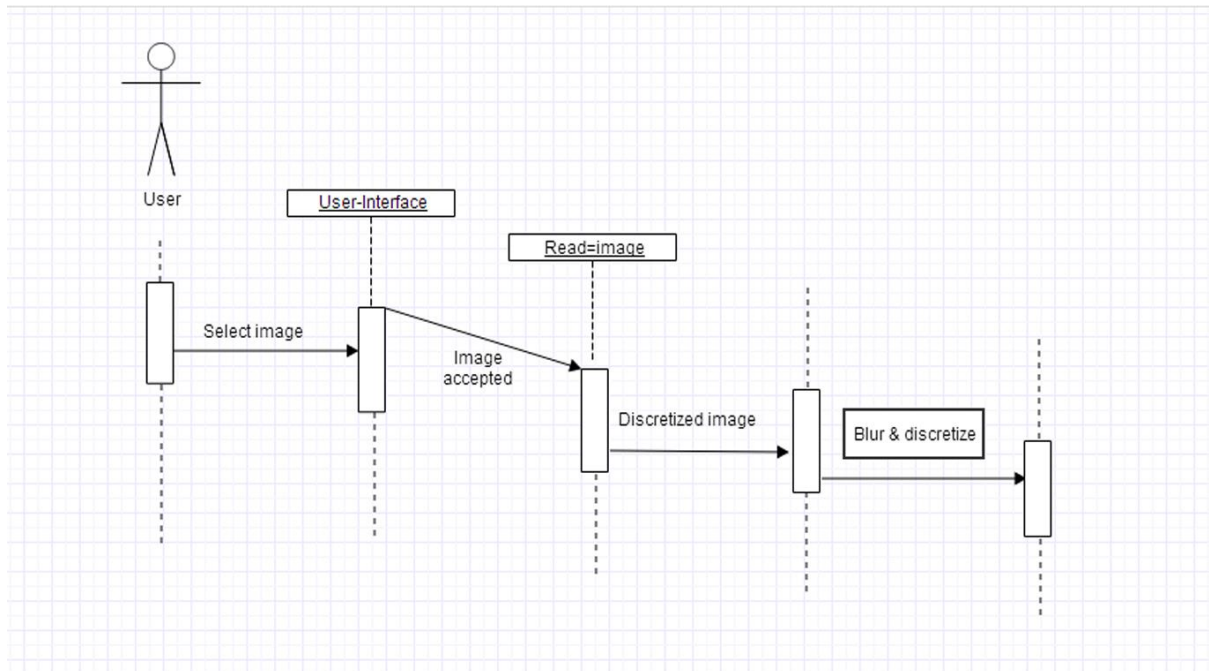
7. Comparing.
8. Retrieving.

The authorized user gives a collection of images as input. Read image function reads the image into buffer and now the image is blurred and discretized. Coherence processing function extracts the coherent vector values. The inserting function inserts these values into the database.

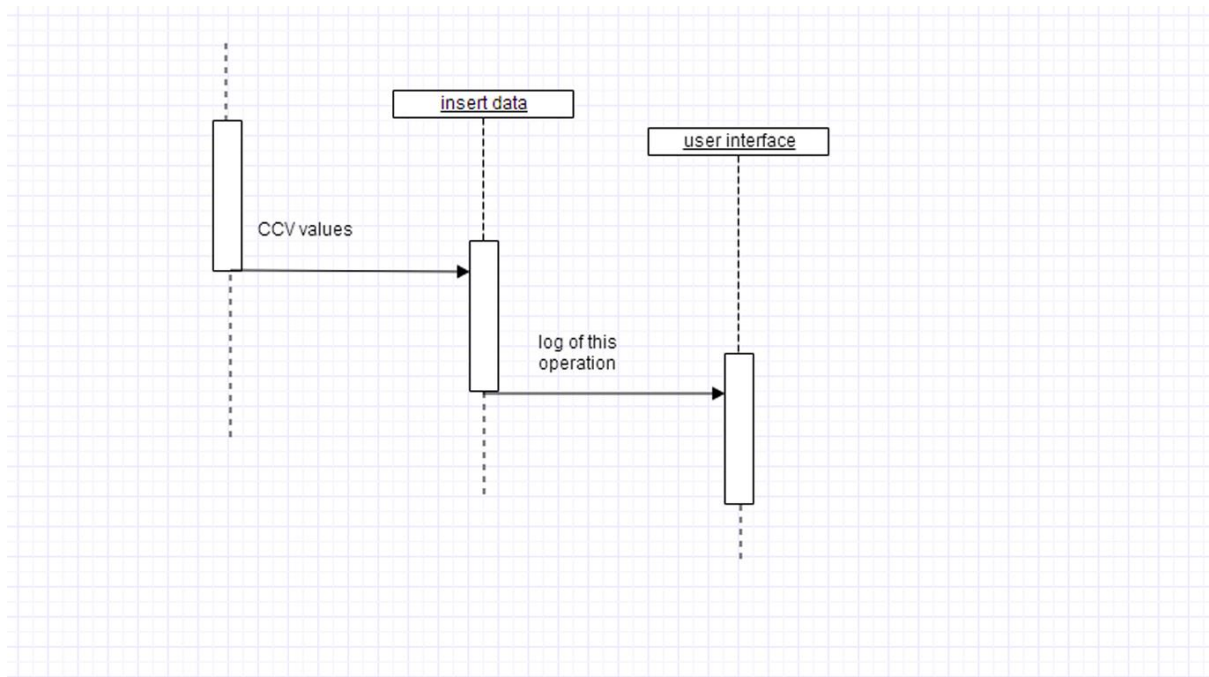
The general user gives a query image as an input. Read image function reads the image into buffer and now the image is blurred and discretized. Coherence processing function extracts coherent vector values. These values are compared with the values of images stored in the database and return an array of images according to their ranks. The retrieval function retrieves the images from the database and displays the results.

UML INTERACTION DIAGRAMS

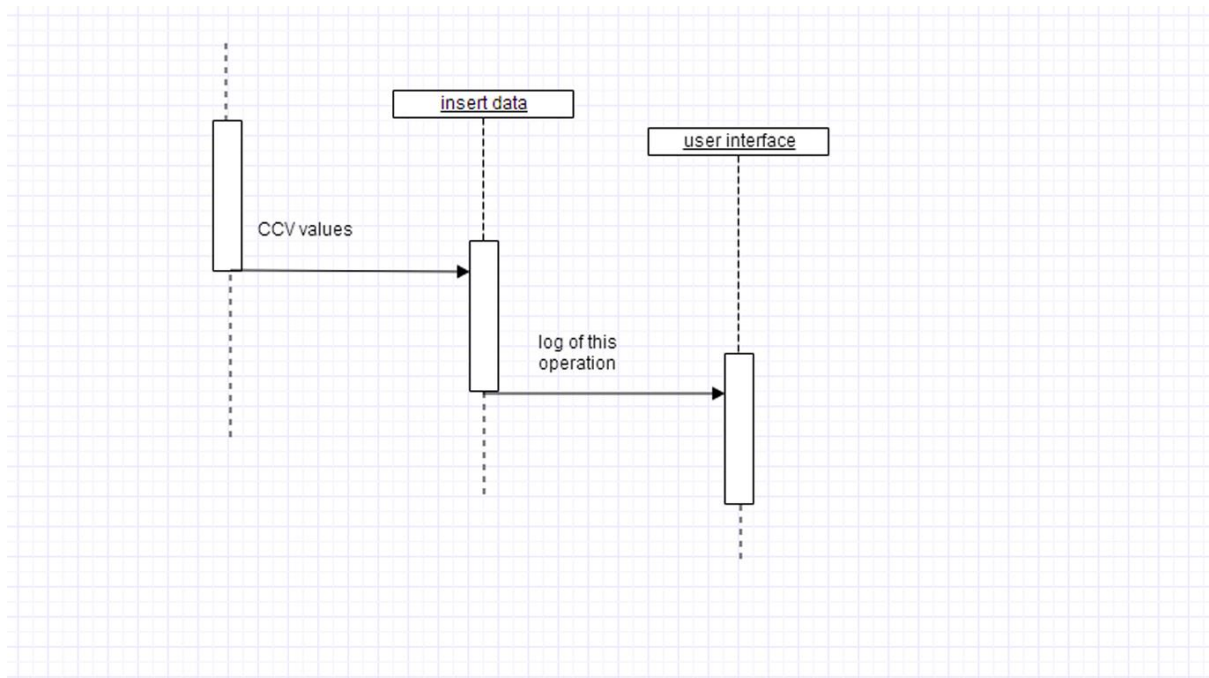
READ IMAGE



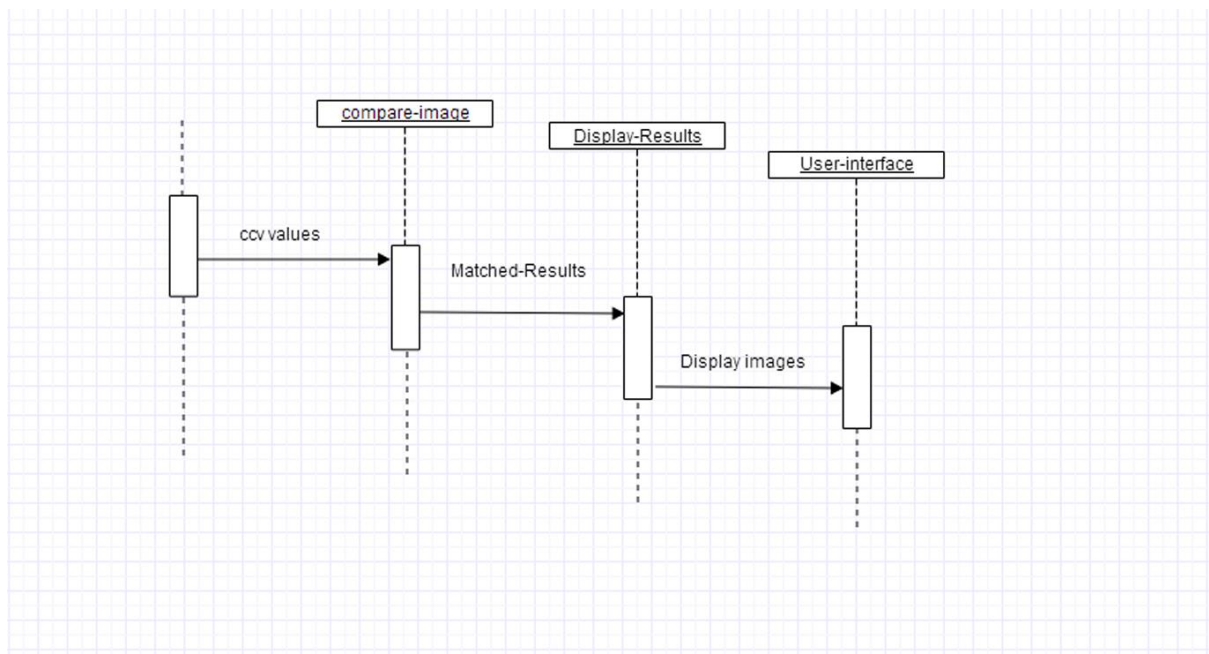
INSERTING



COHERENCE PROCESSING



COMPARING



7.ALGORITHMS

ALGORITHM FOR CALCULATING COLOUR HISTOGRAM

We discretize the colour space of the image such that there are n distinct (discretized) colours. A colour histogram H is a vector $\langle h_1, h_2, \dots, h_n \rangle$ in which each bucket h_j contains the number of pixels of colour j in the image. Typically images are represented in the RGB colour space, and a few of the most significant bits are used from each colour channel. For a given image I , the colour histogram HI is a compact summary of the image. A database of images can be queried to find the most similar image to I , and can return the image I' with the most similar colour histogram HI' .

Comparing Images Using Colour Histogram

Typically colour histograms are compared using the sum of squared differences (L2-distance) or the sum of absolute value of differences (L1-distance). So the most similar image to I would be the image I_0 minimizing

$$\|HI - HI'\| = \sum_{j=1}^n (HI[j] - HI'[j])^2$$

for L2-distance, or

$$\|HI - HI'\| = \sum_{j=1}^n |HI[j] - HI'[j]|$$

for the L1-distance. Note that we are assuming that differences are weighted evenly across different colour buckets for simplicity.

Algorithm for Calculating Colour Coherence Vector

Intuitively, we define a colour's coherence as the degree to which pixels of that colour are members of large similarly-coloured regions. We refer to these significant regions as coherent regions, and observe that they are of significant importance in characterizing images.

Our coherence measure classifies pixels as either coherent or incoherent. Coherent pixels are a part of some sizable contiguous region, while incoherent pixels are not. A

colour coherence vector represents this classification for each colour in the image. CCV's prevent coherent pixels in one image from matching incoherent pixels in another. This allows fine distinctions that cannot be made with colour histograms.

Computing CCV's

First blur the image slightly by replacing pixel values with the average value in a small local neighbourhood (currently including the 8 adjacent pixels). This eliminates small variations between neighbouring pixels. We then discretize the colour space, such that there are only n distinct colours in the image.

The next step is to classify the pixels within a given colour bucket as either coherent or incoherent.

A coherent pixel is part of a large group of pixels of the same colour, while an incoherent pixel is not. We determine the pixel groups by computing connected components. A connected component C is a maximal set of pixels such that for any two pixels $p, p_0 \in C$, there is a path in C between p and p_0 . (Formally, a path in C is a sequence of pixels $p = p_1; p_2; \dots; p_n = p_0$ such that each pixel p_i is in C and any two sequential pixels $p_i; p_{i+1}$ are adjacent to each other. We consider two pixels to be adjacent if one pixel is among the eight closest neighbours of the other; in other words, we include diagonal neighbours.) Note that we only compute connected components within a given discretized colour bucket. This effectively segments the image based on the discretized colour space. Connected components can be computed in linear time. When this is complete, each pixel will belong to exactly one connected component. We classify pixels as either coherent or incoherent depending on the size in pixels of its connected component. A pixel is coherent if the size of its connected component exceeds a fixed value T ; otherwise, the pixel is incoherent.

For a given discretized colour, some of the pixels with that colour will be coherent and some will be incoherent. Let us call the number of coherent pixels of the j 'th discretized colour α_j and the number of incoherent pixels β_j . Clearly, the total number of pixels with that colour is $\alpha_j + \beta_j$, and so a colour histogram would summarize an image as

$$\langle \alpha_1 + \beta_1 + \alpha_2 + \beta_2 + \dots + \alpha_n + \beta_n \rangle.$$

Instead, for each colour we compute the pair (α_j, β_j) which we will call the coherence pair for the j 'th colour. The colour coherence vector for the image consists of

$$\langle (\alpha_1, \beta_1), (\alpha_2, \beta_2), \dots, (\alpha_n, \beta_n) \rangle.$$

Comparing CCV's

Consider two images I and I_0 , together with their CCV's GI and GI_0 , and let the number of coherent pixels in colour bucket j be α_j (for I) and α_{0j} (for I_0). Similarly, let the number of incoherent pixels be β_j and β_{0j} . So

$$GI = \langle (\alpha_1, \beta_1), (\alpha_2, \beta_2), (\alpha_3, \beta_3), \dots, (\alpha_n, \beta_n) \rangle$$

and

$$GI' = \langle (\alpha_1', \beta_1'), (\alpha_2', \beta_2'), (\alpha_3', \beta_3'), \dots, (\alpha_n', \beta_n') \rangle$$

$$\Delta G = \sum_{j=1}^{64} |\alpha_j - \alpha_j'| + |\beta_j - \beta_j'|$$

8.BACKEND

The backend is a database which is a combination of operating system file system and RDBMS .The actual Images are stored in the file system .The features extracted from the images are stored in the database along with the Image.

Our database consists of only one table '**imgdata**'

Description of the table

Field	Type	Null	Key	Default
Images	LongBlob	No		NULL
Colhist	Varchar(5000)	No		NULL
Id	Int(11)	No	Yes	NULL
Key	LongText	No		NULL
Vector	LongText	No		NULL

9.CODE

Insertintodb.py:

```
import sys
from PIL import Image
import ImageFilter
import numpy
import PIL.Image
from numpy import array
import copy
import scipy
import os
import mysql.connector
import cStringIO
import base64

WIDTH = 0
HEIGHT = 0
SIZE = 0
TOU = 0

path='C:/Users/Abhi/Desktop/cbir-p/project/images'
ext='.jpg'
def main():
    data=1
    a=[]
    for i in range(1000):
        a.append(data)
```

```

    data=data+1
for k in a:
    imageFile =path+'/a ('+str(k)+').jpg'
    colorString=colorHistogram(imageFile)
    ccv=processColorCoherenceVector(imageFile)
    stringKey=""
    stringValues=""
    keys=ccv.keys()
    vectors=ccv.values()
    for key in keys:
        stringKey='#'.join(key for key in keys)
    for value in vectors:
        stringValues='$'.join(str(value) for value in vectors)
    insert(imageFile,colorString,k,stringKey,stringValues)
    print 'insert successful'

def colorHistogram(imageName):
    b=""
    xsize , ysize = Image.open(imageName).size
    im=Image.open(imageName)
    rgb_im = im.convert('RGB')
    redhist=[0]*256
    greenhist=[0]*256
    bluehist=[0]*256
    for y in range(ysize):
        for x in range(xsize):
            r,g,b= rgb_im.getpixel((x,y))
            redhist[r]=redhist[r]+1
            greenhist[g]=greenhist[g]+1
            bluehist[b]=bluehist[b]+1
    result=[]

```

```

result=redhist+greenhist+bluehist
for r in result:
    b=':'.join(str(r) for r in result)
return b

def insert(imgf,st,k,ke,ve):
    db = mysql.connector.connect(user='root', password='abhi',
                                host='localhost',
                                database='cbir')

    blob_value = open(imgf, 'rb').read()
    sql      =      "INSERT      INTO      imgdata(images,colhist,id,`key`,`vector`)
VALUES(%s,%s,%s,%s,%s)"
    args = (blob_value,st,k,ke,ve)
    cursor=db.cursor()
    cursor.execute(sql,args)
    db.commit()
    db.close()

def processColorCoherenceVector(im):
    matrix = convertImageToMatrix(im)
    computeImageDimensions(matrix)
    matrix = computeImageBlur(matrix)
    matrix = computeDiscretization(matrix)
    labelMatrix, connectedComponentTable = computeConnectedComponents(matrix)

    ccv = computeColorCoherenceVector(connectedComponentTable)
    return ccv

import matplotlib.pyplot as plt

def displayImage(image):
    plt.imshow(image)
    plt.show()

def computeColorCoherenceVector(connectedComponentTable):

```

```

tempCCT = {}
colorCoherenceVector = {}

for key, (label, color, size) in connectedComponentTable.iteritems():
    if(size >= TOU):
        tempCCT[key] = (label, color, size, True)
    else:
        tempCCT[key] = (label, color, size, False)

for (label, color, size, coherence) in tempCCT.values():
    if(coherence == True):
        clr='.'.join(str(r) for r in color)
        (alpha, beta) = colorCoherenceVector.get(clr, (0, 0))
        colorCoherenceVector[clr] = (alpha + size, beta)
    else:
        clr='.'.join(str(r) for r in color)
        (alpha, beta) = colorCoherenceVector.get(clr, (0, 0))
        colorCoherenceVector[clr] = (alpha, beta + size)
return colorCoherenceVector

```

```

def computeConnectedComponents(image):
    connectedComponentTable = {}
    labelImage = copy.deepcopy(image)
    for x in range(0, WIDTH):
        for y in range(0, HEIGHT):
            labelImage[x][y] = None
    listOfPixels = computeListOfPixelsInOrder()
    index = 0
    while index < len(listOfPixels):
        (x, y) = listOfPixels[index]
        if(labelImage[x][y] is None):

```

```

    for i in range(-1, 2):
        for j in range(-1, 2):
            nx = x + i
            ny = y + j
            if(nx >= 0 and nx < WIDTH and ny >= 0 and ny < HEIGHT and
labelImage[x][y] is None and
labelImage[nx][ny] is not None and image[x][y] == image[nx][ny]):
                labelImage[x][y] = labelImage[nx][ny]
                incrementConnectedComponentsTable(connectedComponentTable,
labelImage[nx][ny],
image[nx][ny])

```

```

    if(labelImage[x][y] is None):
        labelImage[x][y] = getNextLabel()
        incrementConnectedComponentsTable(connectedComponentTable,
labelImage[x][y], image[x][y])

```

```

    for i in range(-1, 2):
        for j in range(-1, 2):
            nx = x + i
            ny = y + j
            if(nx >= 0 and nx < WIDTH and ny >= 0 and ny < HEIGHT and
labelImage[nx][ny] is None and
image[x][y] == image[nx][ny]):
                labelImage[nx][ny] = labelImage[x][y]
                incrementConnectedComponentsTable(connectedComponentTable,
labelImage[nx][ny],
image[nx][ny])
                listOfPixels.insert(index+1, (nx, ny))

```

```

    index = index + 1

```



```

    return labelImage, connectedComponentTable
labelCount = 0;
def getNextLabel():
    global labelCount
    label = str(labelCount)
    labelCount = labelCount + 1
    return label
def incrementConnectedComponentsTable(connectedComponentTable, label, color):
    key = str(label) + "_" + str(color)
    (receivedLabel, receivedColor, receivedSize) = connectedComponentTable.get(key,
(label, color, 0))
    connectedComponentTable[key] = (receivedLabel, receivedColor, receivedSize +
1)
def computeListOfPixelsInOrder():
    listOfPixels = []

    for x in range(0, WIDTH):
        for y in range(0, HEIGHT):
            listOfPixels.append((x,y))

    return listOfPixels

def computeDiscretization(image):
    for x in range(0, WIDTH):
        for y in range(0, HEIGHT):
            for i in range (3):
                image[x][y][i]=image[x][y][i] / 4

    return image

```

```

def convertImageToMatrix(inputImageFilename):
    size = Image.open(inputImageFilename).size
    im=Image.open(inputImageFilename)
    rgb_im = im.convert('RGB')
    image=[]
    for x in range(size[0]):
        image.append([])
        for y in range(size[1]):
            inten=list(rgb_im.getpixel((x,y)))
            image[x].append(inten)

    return image

def computeImageDimensions(matrix):
    global WIDTH
    global HEIGHT
    global SIZE
    global TOU
    WIDTH = len(matrix)
    HEIGHT = len(matrix[0])
    SIZE = WIDTH * HEIGHT
    TOU = SIZE / 100

def printImage(image):
    for x in range(0, WIDTH):
        for y in range(0, HEIGHT):
            print image[x][y],
        print
    print

def printCCV(ccv):
    print "Color", "Alpha", "Beta"
    for color, (alpha, beta) in ccv.iteritems():

```

```

        print color, alpha, beta
    print
def printCCT(connectedComponentTable):
    print "Label", "Color", "Size"
    for (label, color, size) in connectedComponentTable.values():
        print label, color, size
def computeImageBlur(image):
    for x in range(0, WIDTH):
        for y in range(0, HEIGHT):
            image[x][y] = averageOfNeighbors(x, y, image)
    return image
def averageOfNeighbors(x, y, image):
    value = [0,0,0]
    count = 0
    for i in range(-1, 2):
        for j in range(-1, 2):
            nx = x + i
            ny = y + j
            if(nx >= 0 and nx < WIDTH and ny >= 0 and ny < HEIGHT):
                for li in range(3):
                    value[li] = value[li] + image[nx][ny][li]
                count = count + 1
    #print count
    for li1 in range(3):
        value[li1]=value[li1]/count
    return value
if __name__ == '__main__':
    main()

```

Retrieveandcompare.py

```

import mysql.connector
import sys

```

```
from PIL import Image
import cStringIO
import PIL.Image
import ImageFilter
import os
import copy
import scipy
import numpy
from numpy import array
```

```
path='C:/Users/Abhi/Desktop/cbir-p/project/retrieved/'
ext='.jpg'
```

```
def main():
```

```
    global path
```

```
    global ext
```

```
    for file in os.listdir(path):
```

```
        os.remove(path+file)
```

```
    imageFile='C:/Users/Abhi/Desktop/cbir-p/project/testing/test (45).jpg'
```

```
    xsize , ysize = Image.open(imageFile).size
```

```
    im=Image.open(imageFile)
```

```
    rgb_im = im.convert('RGB')
```

```
    redhist=[0]*256
```

```
    greenhist=[0]*256
```

```
    bluehist=[0]*256
```

```
    for y in range(ysize):
```

```
        for x in range(xsize):
```

```
            r,g,b= rgb_im.getpixel((x,y))
```

```
            redhist[r]=redhist[r]+1
```

```

        greenhist[g]=greenhist[g]+1
        bluehist[b]=bluehist[b]+1
result=redhist+greenhist+bluehist
compare(result)
def compare(r):
    global path
    global ext
    db = mysql.connector.connect(user='root', password='abhi',
                                host='localhost',
                                database='cbir')
    sql = 'SELECT colhist,id FROM imgdata;'
    key1=[]
    key2=[]
    cursor=db.cursor()
    cursor.execute(sql)
    data=cursor.fetchall()
    colid={ }
    retrieved={ }
    for item in data:
        colid[item[1]]=item[0]
    for key in colid.keys():
        r1=[]
        r1=r
        r2=[]
        st=str(colid[key])
        r2=st.split(':')
        r3=[]
        for i in r2:
            r3.append(int(i))
        dist=0
        for i in range(768):

```

```

        dist=dist+abs(r1[i]-r3[i])
    retrieved[dist]=key

for g in range(12):
    saveImage(retrieved[sorted(retrieved)[g]])
print "retrieved!!!"

def saveImage(x):
    global path
    global ext
    db = mysql.connector.connect(user='root', password='abhi',
                                host='localhost',
                                database='cbir')

    sql1='select images from imgdata where id='+str(x)
    cursor=db.cursor()
    cursor.execute(sql1)
    data1=cursor.fetchall()
    file_like=cStringIO.StringIO(data1[0][0])
    img=PIL.Image.open(file_like)
    img.save(path+str(x)+ext)

if __name__ == '__main__':
    main()

```

Retrieveandcomparewithcvv.py

```
import sys
from PIL import Image
import ImageFilter
import numpy
import PIL.Image
from numpy import array
import copy
import scipy
import os
import mysql.connector
import cStringIO
import base64

WIDTH = 0
HEIGHT = 0
SIZE = 0
TOU = 0

path='C:/Users/Harish/Desktop/abhi/project/retrieved/'
ext='.jpg'
def main():
    global path
    global ext

    imageFile='C:/Users/Harish/Desktop/abhi/project/testimg/test (45).jpg'
    result1=processColorCoherenceVector(imageFile)
    retr={}
    for file in os.listdir(path):
```

```

imageFile1 = path+file
result2=processColorCoherenceVector(imageFile1)
DISTANCE=compare(result1,result2)
retr[DISTANCE]=file
print retr
ki=sorted(retr)
for x in range(5) :

    saveImage(retr[ki[x]])

```

```

def saveImage(name):
    global path
    pa='C:/Users/Harish/Desktop/abhi/project/ccvretr/'
    ex='.jpg'
    img=PIL.Image.open(path+name)
    img.save(pa+name+ext)

```

```

def colorHistogram(imageName):
    b=""
    xsize , ysize = Image.open(imageName).size
    im=Image.open(imageName)
    rgb_im = im.convert('RGB')
    redhist=[0]*256
    greenhist=[0]*256
    bluehist=[0]*256
    for y in range(ysize):
        for x in range(xsize):
            r,g,b= rgb_im.getpixel((x,y))
            redhist[r]=redhist[r]+1

```



```

        greenhist[g]=greenhist[g]+1
        bluehist[b]=bluehist[b]+1
result=[]
result=redhist+greenhist+bluehist
#print len(result)
for r in result:
    b=':'.join(str(r) for r in result)
return b
def retrieve():
    db = mysql.connector.connect(user='root', password='abhi',
                                host='localhost',
                                database='cbir')
    sql = "SELECT `key`,`vector` FROM ccv;"
    cursor=db.cursor()
    cursor.execute(sql)
    data=cursor.fetchall()
    k= str(data[0][0])
    #print l
    r2=k.split('#')
    #print len(r2)
    v=str(data[0][1])
    r3=v.split('$')
    from ast import literal_eval
    r4 = [literal_eval(s) for s in r3]
    print r4

def insert(imgf,st,k,ke,ve):
    db = mysql.connector.connect(user='root', password='abhi',
                                host='localhost',
                                database='cbir')
    blob_value = open(imgf, 'rb').read()

```

```

sql      =      "INSERT      INTO      imgdata(images,colhist,id,`key`,`vector`)
VALUES(%s,%s,%s,%s,%s)"
args = (blob_value,st,k,ke,ve)
cursor=db.cursor()
cursor.execute(sql,args)
db.commit()
db.close()

```

```

def compare(r1,r2):
    dist=0
    for r in range(64):
        for g in range(64):
            for b in range(64):
                clr=str(r)+ ':' +str(g)+ ':' +str(b)
                dist=dist+abs(r1[clr][0]-r2[clr][0])+abs(r1[clr][1]-r2[clr][1])

```

```

    return dist
def processColorCoherenceVector(im):
    matrix = convertImageToMatrix(im)
    computeImageDimensions(matrix)

    matrix = computeImageBlur(matrix)
    matrix = computeDiscretization(matrix)
    labelMatrix, connectedComponentTable = computeConnectedComponents(matrix)

    ccv = computeColorCoherenceVector(connectedComponentTable)
    return ccv

```

```

import matplotlib.pyplot as plt
def displayImage(image):

```

```
plt.imshow(image)
plt.show()
```

```
def computeColorCoherenceVector(connectedComponentTable):
    tempCCT = {}
    colorCoherenceVector = {}
    for r in range(64):
        for g in range(64):
            for b in range(64):
                clr=str(r)+':' +str(g)+':' +str(b)
                colorCoherenceVector[clr]=(0,0)

    for key, (label, color, size) in connectedComponentTable.iteritems():
        if(size >= TOU):
            tempCCT[key] = (label, color, size, True)
        else:
            tempCCT[key] = (label, color, size, False)

    for (label, color, size, coherence) in tempCCT.values():
        if(coherence == True):
            clr=':'.join(str(r) for r in color)
            (alpha, beta) = colorCoherenceVector.get(clr, (0, 0))
            colorCoherenceVector[clr] = (alpha + size, beta)
        else:
            clr=':'.join(str(r) for r in color)
            (alpha, beta) = colorCoherenceVector.get(clr, (0, 0))
            colorCoherenceVector[clr] = (alpha, beta + size)

    return colorCoherenceVector
```

```

def computeConnectedComponents(image):
    connectedComponentTable = { }

    labelImage = copy.deepcopy(image)
    for x in range(0, WIDTH):
        for y in range(0, HEIGHT):
            labelImage[x][y] = None

    listOfPixels = computeListOfPixelsInOrder()
    index = 0
    while index < len(listOfPixels):
        (x, y) = listOfPixels[index]
        if(labelImage[x][y] is None):
            for i in range(-1, 2):
                for j in range(-1, 2):
                    nx = x + i
                    ny = y + j
                    if(nx >= 0 and nx < WIDTH and ny >= 0 and ny < HEIGHT and
labelImage[x][y] is None and labelImage[nx][ny] is not None and image[x][y] ==
image[nx][ny]):
                        labelImage[x][y] = labelImage[nx][ny]
                        incrementConnectedComponentsTable(connectedComponentTable,
labelImage[nx][ny], image[nx][ny])

            if(labelImage[x][y] is None):
                labelImage[x][y] = getNextLabel()

            incrementConnectedComponentsTable(connectedComponentTable,
labelImage[x][y], image[x][y])

```

```

    for i in range(-1, 2):
        for j in range(-1, 2):
            nx = x + i
            ny = y + j
            if(nx >= 0 and nx < WIDTH and ny >= 0 and ny < HEIGHT and
labelImage[nx][ny] is None and image[x][y] == image[nx][ny]):
                labelImage[nx][ny] = labelImage[x][y]
                incrementConnectedComponentsTable(connectedComponentTable,
labelImage[nx][ny], image[nx][ny])
                listOfPixels.insert(index+1, (nx, ny))

        index = index + 1

    return labelImage, connectedComponentTable

labelCount = 0;

def getNextLabel():
    global labelCount
    label = str(labelCount)
    labelCount = labelCount + 1
    return label

def incrementConnectedComponentsTable(connectedComponentTable, label, color):
    key = str(label) + "_" + str(color)

    (receivedLabel, receivedColor, receivedSize) = connectedComponentTable.get(key,
(label, color, 0))
    connectedComponentTable[key] = (receivedLabel, receivedColor, receivedSize +
1)

```

```
def computeListOfPixelsInOrder():
```

```
    listOfPixels = []
```

```
    for x in range(0, WIDTH):
```

```
        for y in range(0, HEIGHT):
```

```
            listOfPixels.append((x,y))
```

```
    return listOfPixels
```

```
def computeDiscretization(image):
```

```
    for x in range(0, WIDTH):
```

```
        for y in range(0, HEIGHT):
```

```
            for i in range (3):
```

```
                image[x][y][i]=image[x][y][i] / 4
```

```
    return image
```

```
def convertImageToMatrix(inputImageFilename):
```

```
    size = Image.open(inputImageFilename).size
```

```
    im=Image.open(inputImageFilename)
```

```
    rgb_im = im.convert('RGB')
```

```
    image=[]
```

```
    for x in range(size[0]):
```

```
        image.append([])
```

```

        for y in range(size[1]):
            inten=list(rgb_im.getpixel((x,y)))
            image[x].append(inten)

    return image

def computeImageDimensions(matrix):
    global WIDTH
    global HEIGHT
    global SIZE
    global TOU
    WIDTH = len(matrix)
    HEIGHT = len(matrix[0])
    SIZE = WIDTH * HEIGHT
    TOU = SIZE / 100

def printImage(image):
    for x in range(0, WIDTH):
        for y in range(0, HEIGHT):
            print image[x][y],
        print
    print

def printCCV(ccv):
    print "Color", "Alpha", "Beta"
    for color, (alpha, beta) in ccv.iteritems():
        print color, alpha, beta
    print

def printCCT(connectedComponentTable):
    print "Label", "Color", "Size"
    for (label, color, size) in connectedComponentTable.values():

```

```
print label, color, size
```

```
def computeImageBlur(image):  
    for x in range(0, WIDTH):  
        for y in range(0, HEIGHT):  
            image[x][y] = averageOfNeighbors(x, y, image)  
    return image
```

```
def averageOfNeighbors(x, y, image):  
    value = [0,0,0]  
    count = 0  
    for i in range(-1, 2):  
        for j in range(-1, 2):  
            nx = x + i  
            ny = y + j  
            if(nx >= 0 and nx < WIDTH and ny >= 0 and ny < HEIGHT):  
                for li in range(3):  
                    value[li] = value[li] + image[nx][ny][li]  
                count = count + 1  
    #print count  
    for li1 in range(3):  
        value[li1]=value[li1]/count  
    return value
```

```
if __name__ == '__main__':  
    main()
```

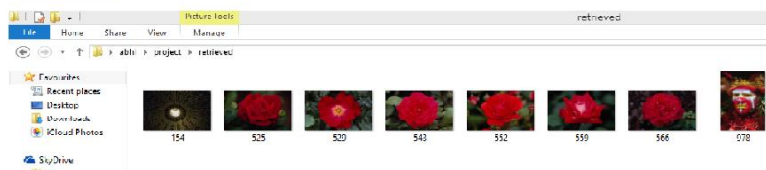

10. TESTCASES

QUERY 1

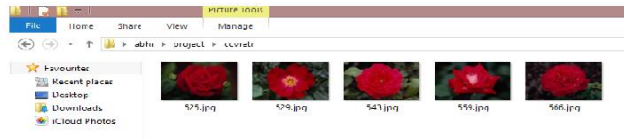
INPUT IMAGE:



IMAGES
RETRIVED
USING
HISTOGRAM:



RETRIEVED
USING CCV

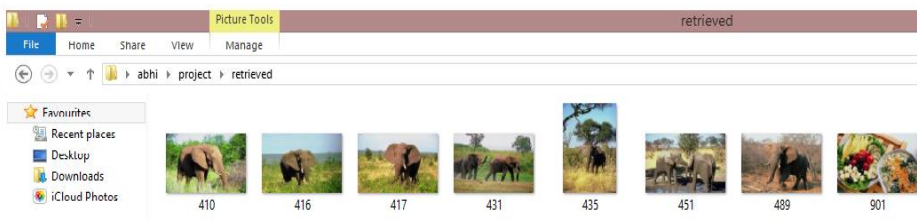


QUERY 2

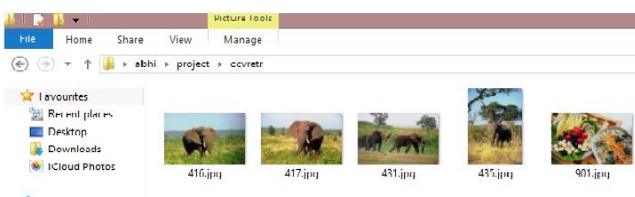
INPUT IMAGE:



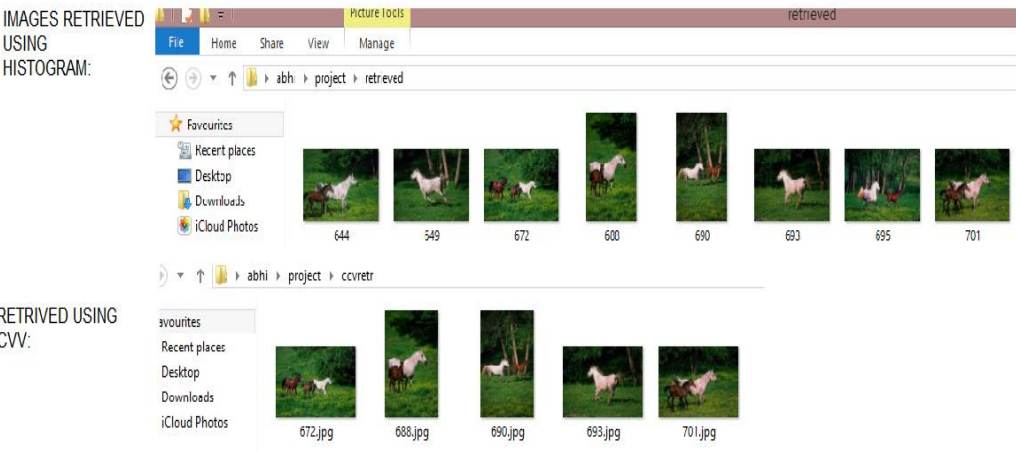
RETRIEVED
USING
HISTOGRAM:



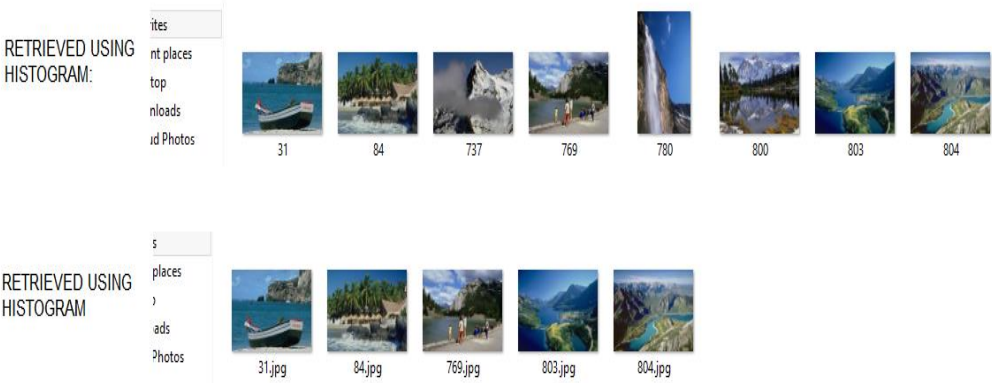
RETRIEVED
USING CCV:



QUERY 3




QUERY 4



QUERY5

INPUT IMAGE:











test (25)

RETRIEVED USING HISTOGRAM






Navigation: achi > project > retrieved

Left sidebar: Favourites, Recent places, Desktop, Downloads

Image	Filename
	315
	320
	328
	345
	346
	363
	364
	399

RETRIVED USING CCV:

Left sidebar: Images, Videos, Photos

Image	Filename
	320.jpg
	345.jpg
	346.jpg
	364.jpg
	399.jpg

11. CONCLUSION

Content Based Image Retrieval is an active and fast advancing research area since the 1990s. During the past decade, remarkable progress has been made in both theoretical research and system development. This project has discussed the colour histograms and colour coherence vector. Based on these features we have delivered similar images to that of the input image from the database and show how efficient colour coherence vector is when compared to colour histogram.

12. REFERENCES

Raghu Ram Krishna “*DATABASE MANAGEMENT SYSTEMS*” 4th edition

Rafael C.Gonzalez, Richard E.Woods “*Digital Image Processing*” 5th edition

Ian Sommerville “*Software Engineering*” 4th edition

Greg Pass ,Ramin Zabih, Justin Miller “*Comparing Images Using Color Coherence Vectors*”

Google’s Python Classes “*PYTHON*”

MySQL Documentation “*MySQL*”