ANNA UNIVERSITY
REGIONAL CAMPUS, COIMBATORE

LABORATORY RECORD

2023 – 2024

NAME : ______________________________

REGISTER NUMBER : ______________________________

BRANCH : B.E. - COMPUTER SCIENCE AND ENGINEERING

SUBJECT CODE : CCS349

SUBJECT : IMAGE AND VIDEO ANALYTICS LABORATORY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ANNA UNIVERSITY-REGIONAL CAMPUS
COIMBATORE - 641 046

ANNA UNIVERSITY
REGIONAL CAMPUS, COIMBATORE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of Practical done in CCS349 – IMAGE AND VIDEO ANALYTICS LABORATORY by_Register No. ____________________in Third Year - Sixth Semester during 2023 - 2024.

STAFF IN-CHARGE HEAD OF THE DEPARTMENT

University Register No: ………………………………………

Submitted for the University Practical Examination held on………………...

INTERNAL EXAMINER EXTERNAL EXAMINER

INDEX

EX NO.	DATE	TITLE	PAGE NO.	MARKS	SIGN

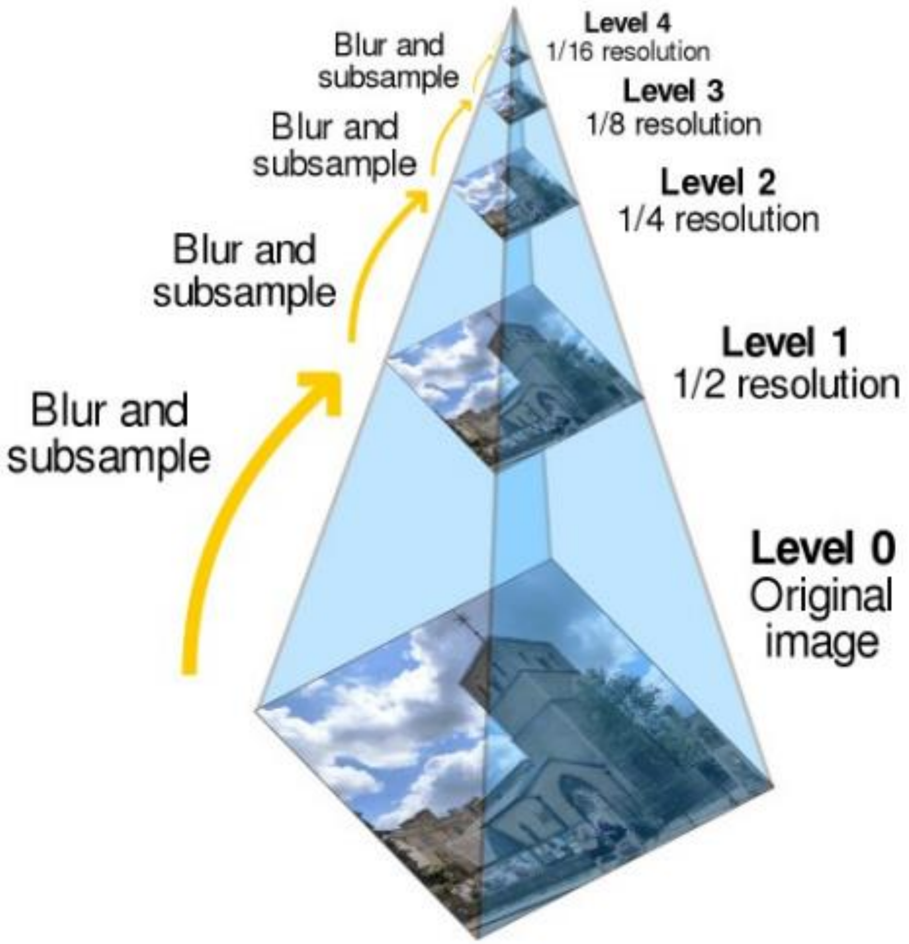
EX NO : 01	**T-PYRAMID OF AN IMAGE**
DATE :	

AIM:

To write python program for T- pyramid of an image.

ALGORITHM:

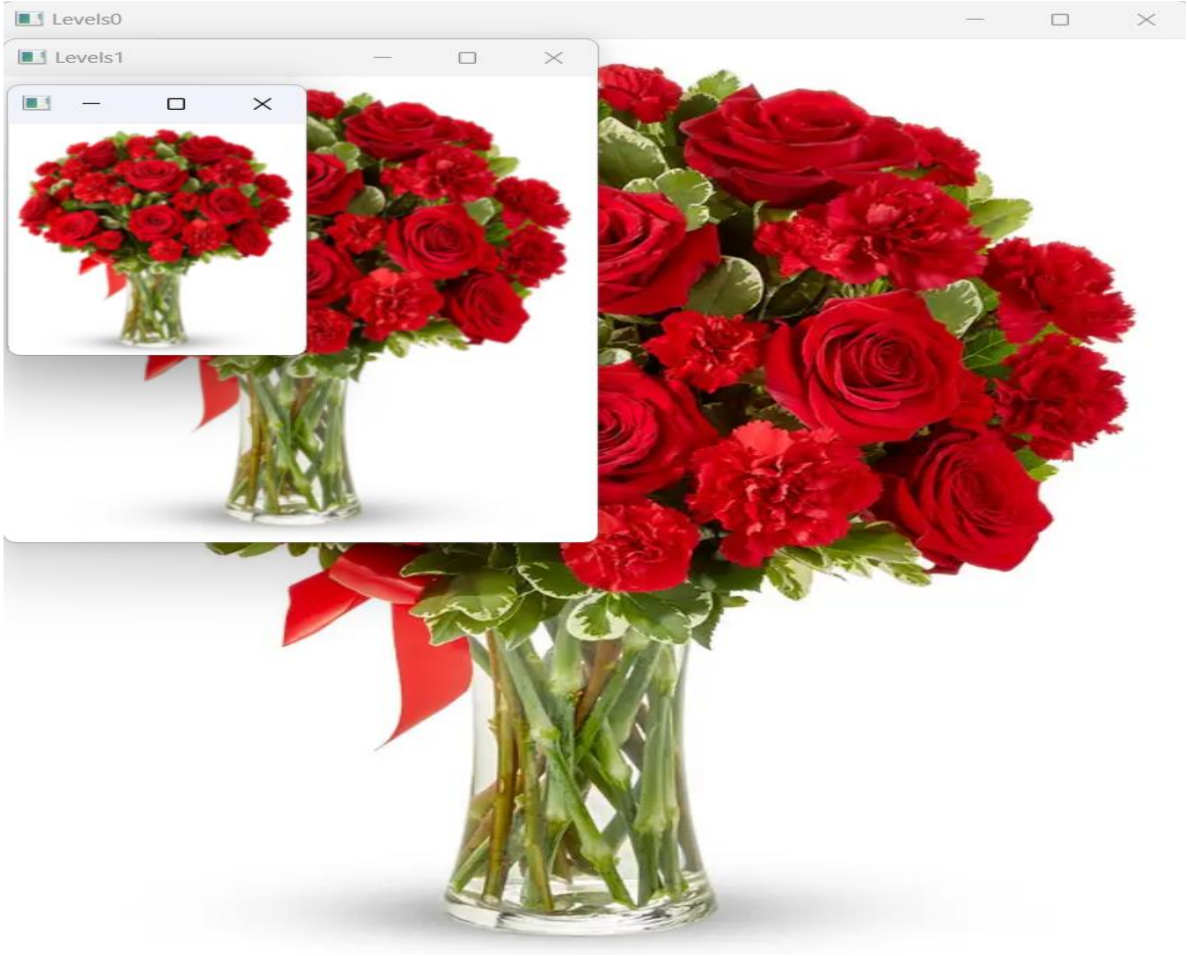
- First load the image
- Then construct the Gaussian pyramid with 3 levels.
- For the Laplacian pyramid, the topmost level remains the same as in Gaussian. The remaining levels are constructed from top to bottom by subtracting that Gaussian level from its upper expanded level.



PROGRAM:

```
import cv2
import numpy as np
def build_t_pyramid(image, levels):
 pyramid = [image]
 for _ in range(levels - 1):
            image = cv2.pyrDown(image)
            pyramid.append(image)
 return pyramid
def main():
 image_path = "IMG_8366.jpg"
 levels = 5
 original_image = cv2.imread(image_path)
 if original_image is None:
            print("Error: Could not load the image.")
            return
 t_pyramid = build_t_pyramid(original_image, levels)
 for i, level_image in enumerate(t_pyramid):
            cv2.imshow(f"Level {i}", level_image)
 cv2.waitKey(0)
 cv2.destroyAllWindows()
if __name__ == "__main__":
 main()
```

OUTPUT:



RESULT:

Thus the python program for T-pyramid implemented and the output is obtained successfully.

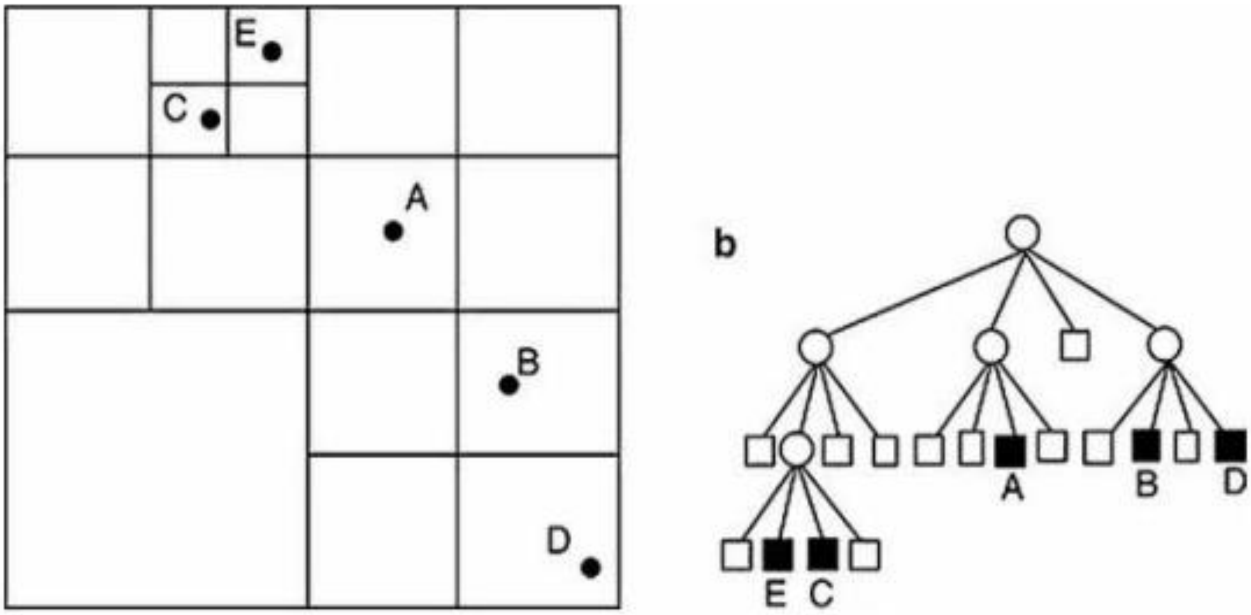
EX NO : 02	**QUAD TREE REPRESENTATION**
DATE :	

AIM:

To write a python program for quad tree representation of an image using the homogeneity criterion of equal intensity.

ALGORITHM:

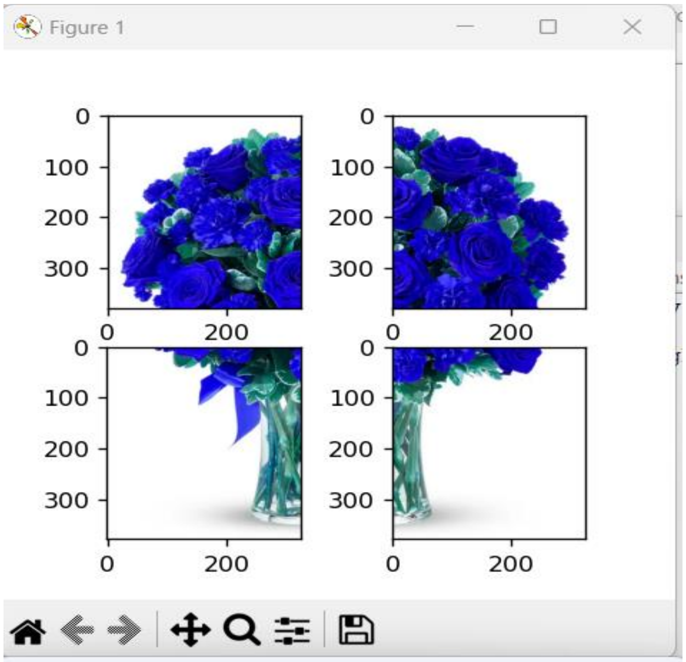
1. Divide the current two dimensional space into four boxes.
2. If a box contains one or more points in it, create a child object, storing in it the two dimensional space of the box
3. If a box does not contain any points, do not create a child for it
4. Recurse for each of the children.

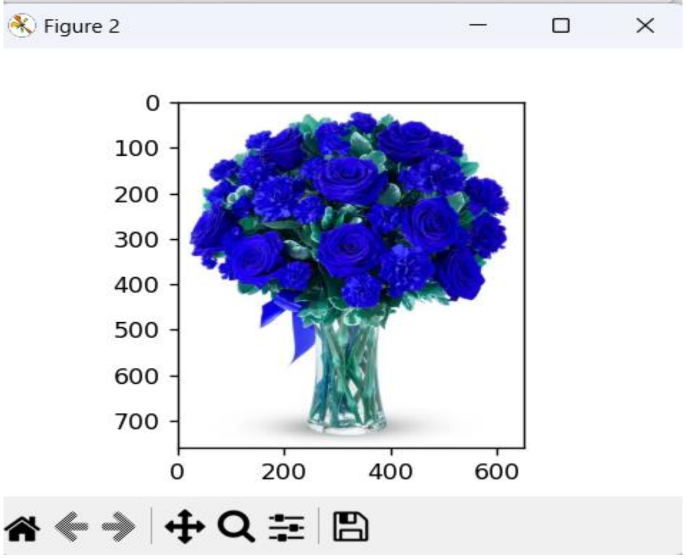


PROGRAM:

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
img = cv2.imread("IMG_8366.JPG")
from operator import add
from functools import reduce
def split4(image):
 half_split = np.array_split(image, 2)
 res = map(lambda x: np.array_split(x, 2, axis= 1), half_split)
 return reduce(add, res)
split_img = split4(img)
split_img[0].shape, split_img[1].shape
fig, axs = plt.subplots(2, 2)
axs[0, 0].imshow(split_img[0])
axs[0, 1].imshow(split_img[1])
axs[1, 0].imshow(split_img[2])
axs[1, 1].imshow(split_img[3])
def concatenate4(north_west, north_east, south_west, south_east):
 top = np.concatenate((north_west, north_east), axis=1)
 bottom = np.concatenate((south_west, south_east), axis=1)
 return np.concatenate((top, bottom), axis=0)
full_img = concatenate4(split_img[0], split_img[1], split_img[2], split_img[3])
plt.imshow(full_img)
```

OUTPUT:





RESULT:

Thus the python program for quad tree representation was implementation and output is obtained successfully.

EX NO : 03	**GEOMETRIC TRANSFORMS**
DATE :	

AIM:

To Develop programs for the following geometric transforms: (a) Rotation, (b) Change of scale, (c) Skewing, (d) Affine transform calculated from three pairs of corresponding points, (e)Bilinear transform calculated from four pairs of corresponding points.

ALGORITHM:

TRANSFORMATION MATRICES:

For each desired transformation, create a corresponding transformation matrix. For example:

 Translation: Create a 3×3 matrix with a 1 in the diagonal and the translation values in the last column.

 Rotation: Compute the rotation matrix using trigonometric functions (sin and cos) and the given rotation angle.

 Scaling: Create a 3×3 matrix with scaling factors along the diagonal and 1 in the last row and column.

 Shearing: Create an affine transformation matrix with shear factors in the offdiagonal elements.

COMBINE TRANSFORMATION MATRICES:

 Multiply the individual transformation matrices in the order you want to apply them. Matrix multiplication is not commutative, so the order matters. The combined matrix represents the sequence of transformations.

APPLY THE COMBINED TRANSFORMATION MATRIX:

In image processing, you can use libraries like OpenCV or Pillow to apply the combined transformation matrix to the image. For example, in OpenCV:

 Convert the 3×3 matrix to a 2×3 matrix by removing the last row.

 Use cv2.warpAffine() for affine transformations or cv2.warpPerspective() for projective transformations.

 Provide the combined transformation matrix and the input image as arguments to apply the transformations.

PROGRAM:

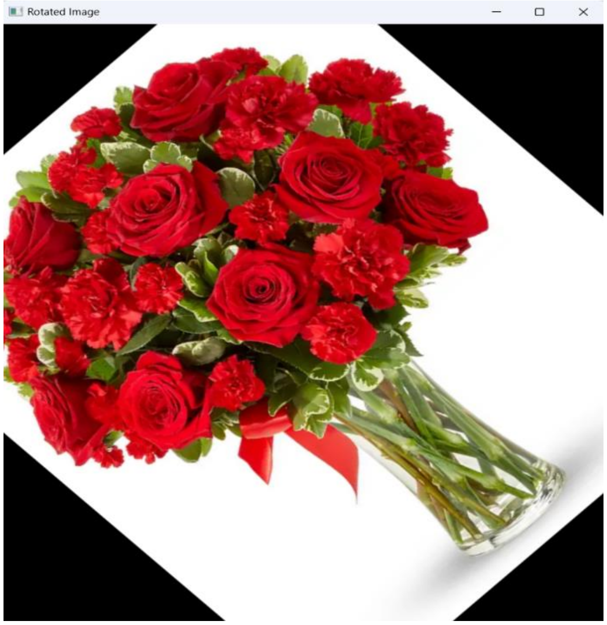
```
import cv2
import numpy as np
def rotate_image(image, angle):
 height, width = image.shape[:2]
 rotation_matrix = cv2.getRotationMatrix2D((width / 2, height / 2), angle, 1)
 rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))
 return rotated_image

# Usage
image = cv2.imread("img.jpg")
angle_degrees = 45
rotated = rotate_image(image, angle_degrees)
cv2.imshow("Rotated Image", rotated)
cv2.waitKey(0)
cv2.destroyAllWindows()

def scale_image(image, scale_x, scale_y):
 scaled_image = cv2.resize(image, None, fx=scale_x, fy=scale_y)
 return scaled_image
```

```
# Usage
image = cv2.imread("img.jpg")
scale_factor_x = 1.5
scale_factor_y = 1.5
scaled = scale_image(image, scale_factor_x, scale_factor_y)
cv2.imshow("Scaled Image", scaled)
cv2.waitKey(0)
cv2.destroyAllWindows()

def skew_image(image, skew_x, skew_y):
 height, width = image.shape[:2]
 skew_matrix = np.float32([[1, skew_x, 0], [skew_y, 1, 0]])
 skewed_image = cv2.warpAffine(image, skew_matrix, (width, height))
 return skewed_image

# Usage
image = cv2.imread("img.jpg")
skew_factor_x = 0.2
skew_factor_y = 0.1
skewed = skew_image(image, skew_factor_x, skew_factor_y)
cv2.imshow("Skewed Image", skewed)
cv2.waitKey(0)
cv2.destroyAllWindows()

def affine_transform(image, pts_src, pts_dst):
 matrix = cv2.getAffineTransform(pts_src, pts_dst)
 transformed_image = cv2.warpAffine(image, matrix, (image.shape[1],
image.shape[0]))
```

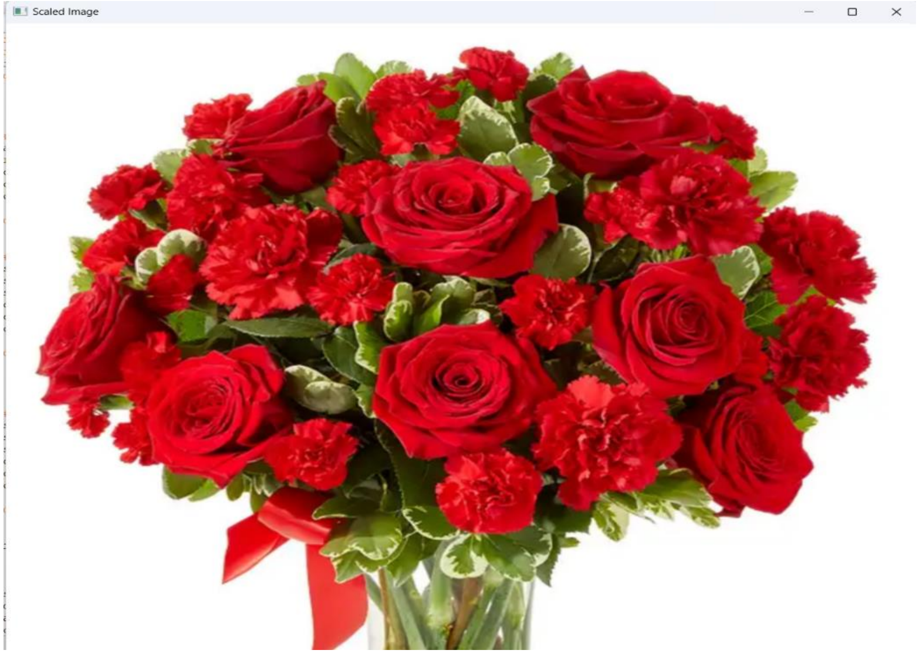
```
 return transformed_image

image = cv2.imread("img.jpg")
src_points = np.float32([[50, 50], [200, 50], [50, 200]])
dst_points = np.float32([[10, 100], [200, 50], [100, 250]])
affine_transformed = affine_transform(image, src_points, dst_points)
cv2.imshow("Affine Transformed Image", affine_transformed)
cv2.waitKey(0)
cv2.destroyAllWindows()

def bilinear_transform(image, pts_src, pts_dst):
 matrix = cv2.getPerspectiveTransform(pts_src, pts_dst)
 transformed_image = cv2.warpPerspective(image, matrix, (image.shape[1],image.shape[0]))
 return transformed_image

image = cv2.imread("img.jpg")
src_points = np.float32([[56, 65], [368, 52], [28, 387], [389, 390]])
dst_points = np.float32([[0, 0], [300, 0], [0, 300], [300, 300]])
bilinear_transformed = bilinear_transform(image, src_points, dst_points)
cv2.imshow("Bilinear Transformed Image", bilinear_transformed)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

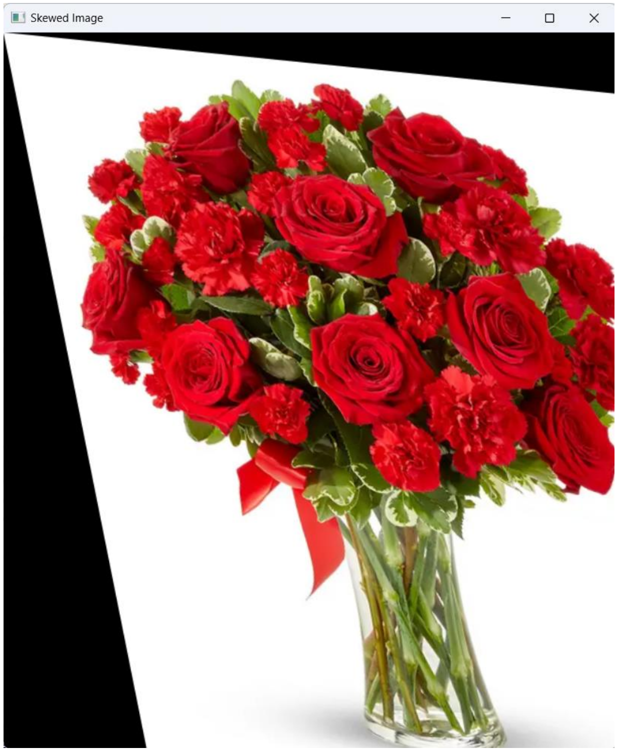
#ROTATED IMAGE



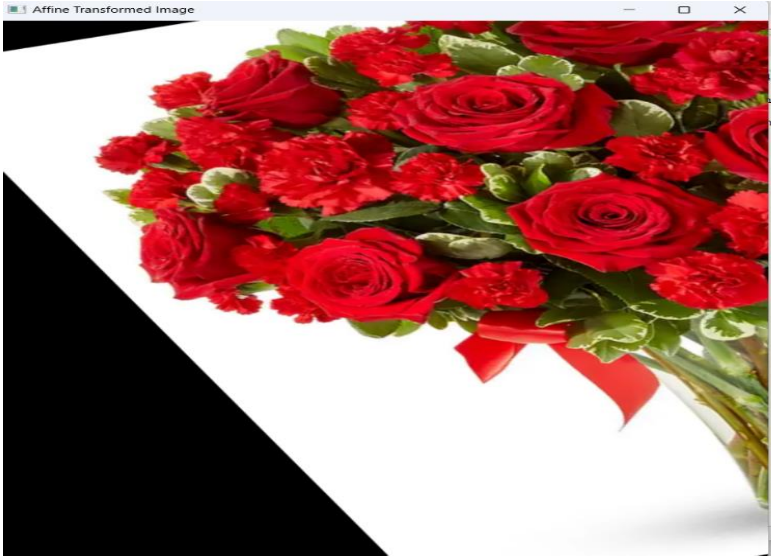
#SCALED IMAGE



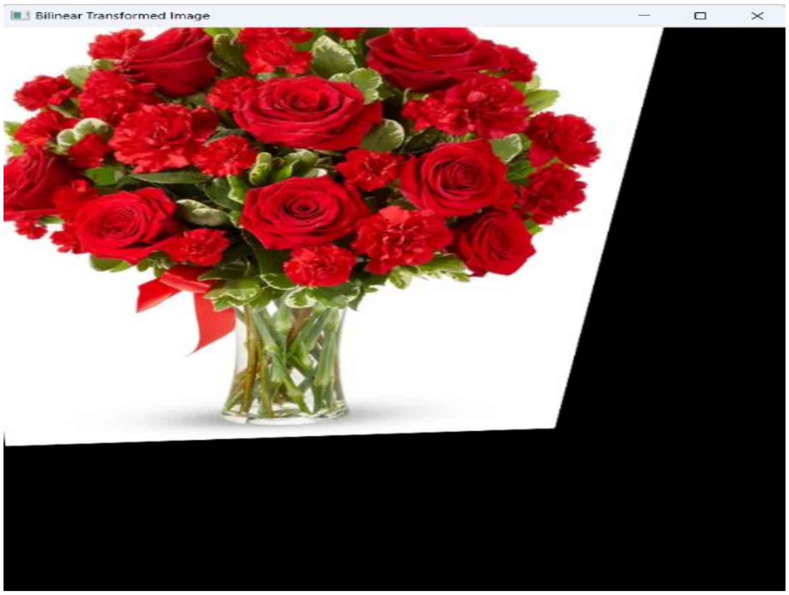
#SKEWED IMAGE



#AFFINE TRANSFORM



#BILINEAR IMAGE



RESULT:

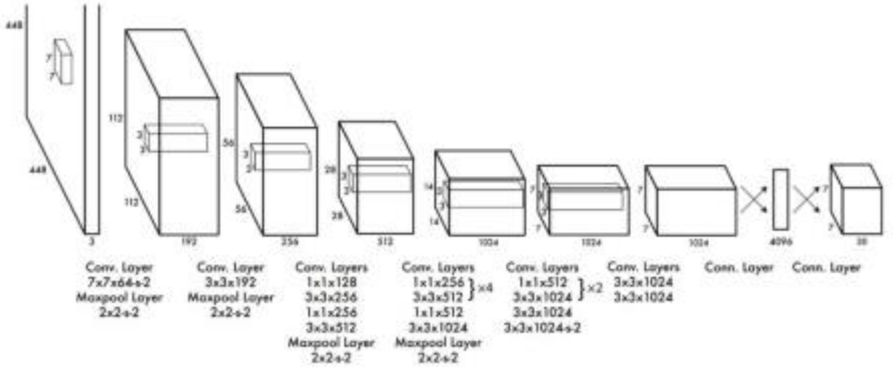
Thus the python program for geometric transforms implemented and output is obtained Successfully.

EX NO : 04	**OBJECT DETECTION AND RECOGNITION**
DATE :	

AIM:

To Develop a program to implement Object Detection and Recognition.

ALGORITHM:(ARCHITECTURE):

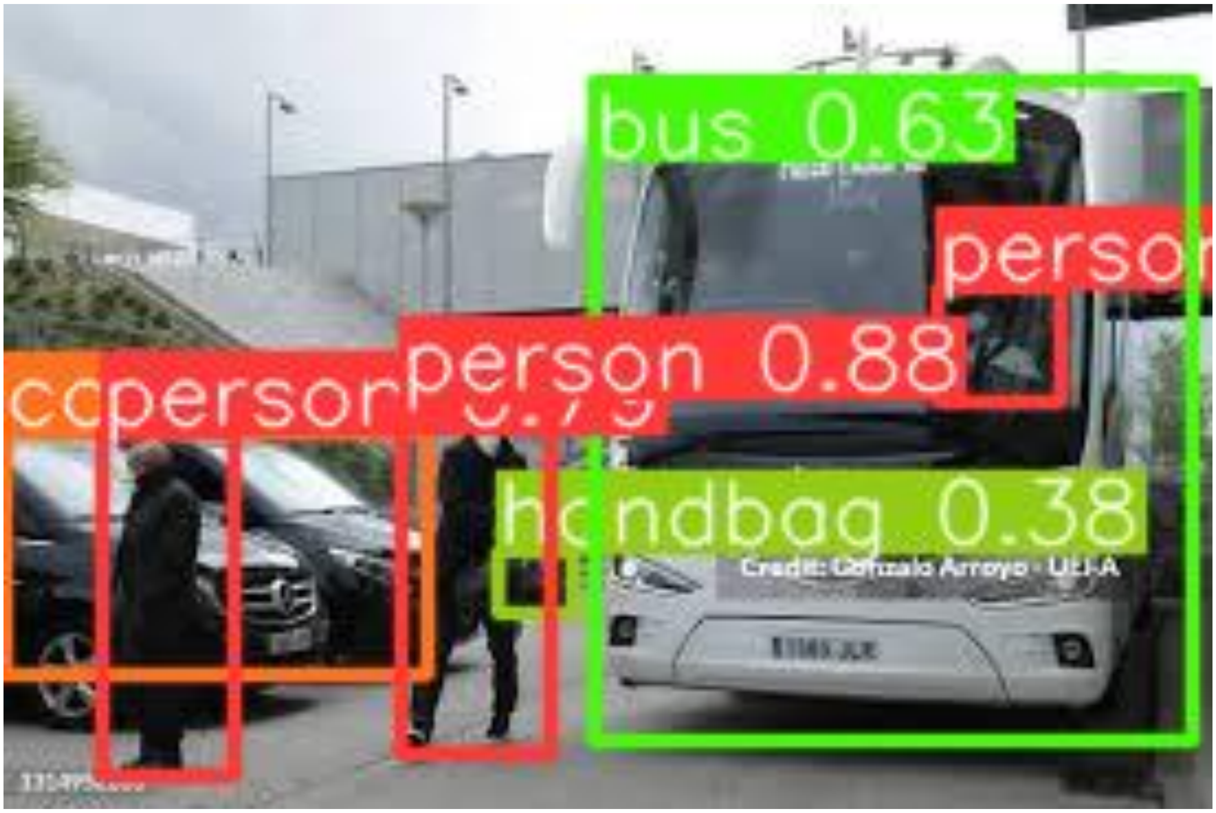


The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 × 1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224 × 224 input image) and then double the resolution for detection.

PROGRAM:

```
import torch
from pathlib import Path
from PIL import Image
import numpy as np
import cv2
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
def detect_objects(image_path):
    img = Image.open(image_path)
    results = model(img)
    return results
image_path = '/content/pic.jpg'
results = detect_objects(image_path)
results.show()
results.save()
```

OUTPUT:



RESULT:

Thus the python program for Object Detection and Recognition implemented and output is obtained successfully.

EX NO : 05

DATE :

MOTION ANALYSIS USING MOVING EDGES

AIM:

To Develop a program for motion analysis using moving edges, and apply it to your image sequence.

ALGORITHM:

Objective

Creating automated Laban movement annotation:

 Training four different machine learning algorithms through supervised learning on existing human motion datasets of video and skeletal sequences

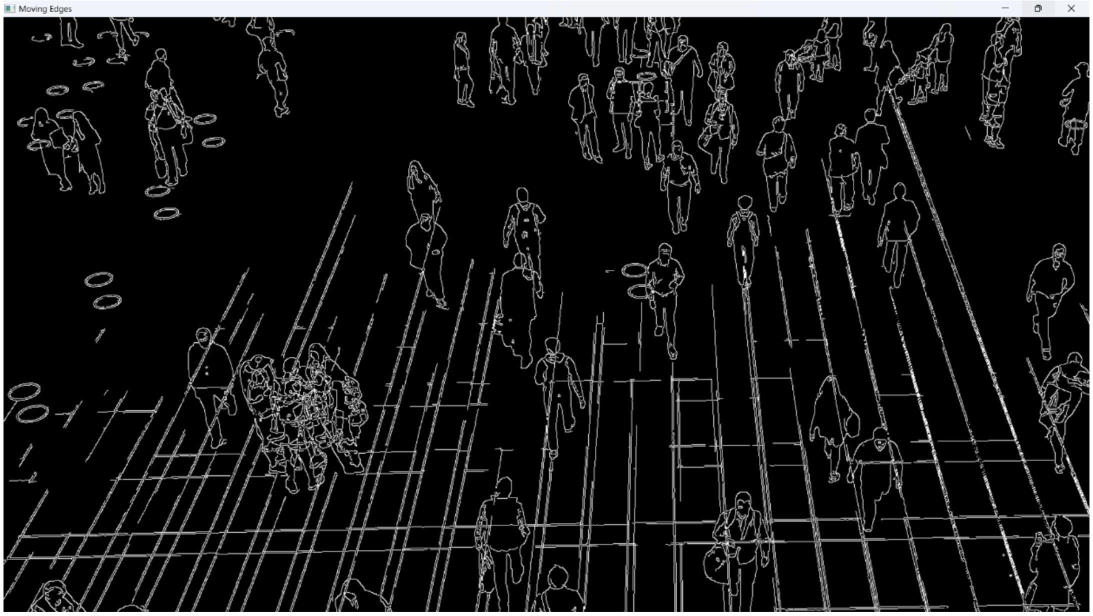
 Test feature extraction methods (within and across frames) to improve the annotation accuracy

 Input raw videos and export Laban annotated videos

PROGRAM:

```
import cv2
import numpy as np
def motion_analysis(video_path):
 cap = cv2.VideoCapture(video_path)
 ret, prev_frame = cap.read()
 prev_gray = cv2.cvtColor(prev_frame,cv2.COLOR_BGR2GRAY)
 while cap.isOpened():
     ret, frame = cap.read()
     if not ret:
         break
     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
     edges_prev = cv2.Canny(prev_gray, 50, 150)
     edges_curr = cv2.Canny(gray, 50, 150)
     frame_diff = cv2.absdiff(edges_prev, edges_curr)
     cv2.imshow('Moving Edges', frame_diff)
     if cv2.waitKey(30) & 0xFF == ord('q'):
        break
 prev_gray = gray.copy()
 cap.release()
 cv2.destroyAllWindows()
video_path = "input_video.mp4"
motion_analysis(video_path)
```

OUTPUT:



RESULT:

Thus the python program for motion analysis using moving edge was implemented and output is

EX NO : 06	**FACIAL DETECTION AND RECOGNITION**
DATE :	

AIM:

To Develop a program for Facial Detection and Recognition.

ALGORITHM:

Face Detection: The very first task we perform is detecting faces in the image or video stream. Now that we know the exact location/coordinates of face, we extract this face for further processing ahead.

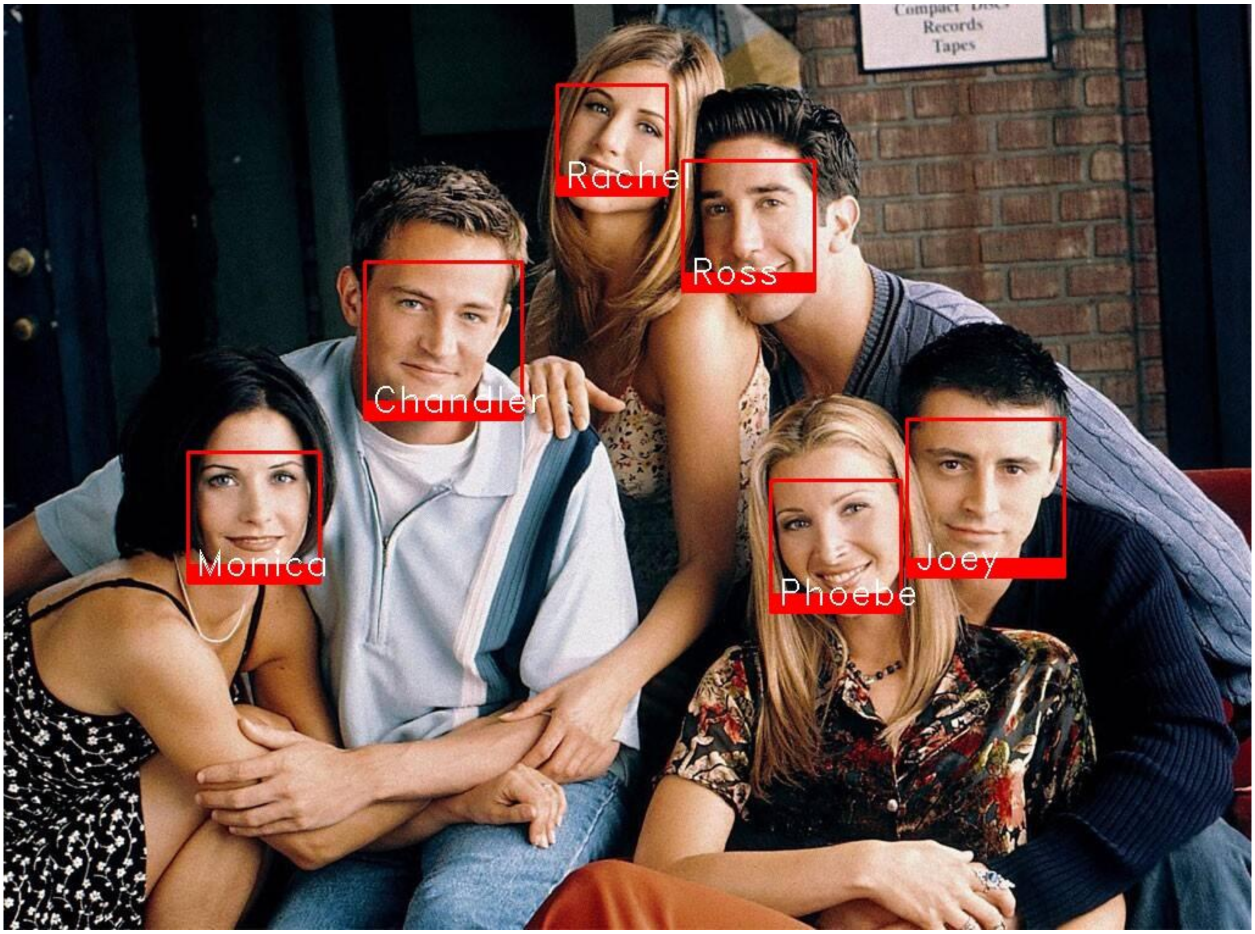
Feature Extraction: Now that we have cropped the face out of the image, we extract features from it. Here we are going to use face embeddings to extract the features out of the face. A neural network takes an image of the person's face as input and outputs a vector which represents the most important features of a face. In machine learning, this vector is called embedding and thus we call this vector as face embedding.

PROGRAM:

```
!pip install dlib

!pip install face_recognition

from google.colab.patches import cv2_imshow

import face_recognition as fr

import cv2

import numpy as np

import os

path = "/content/drive/MyDrive/Colab Notebooks/train/"

known_names = []

known_name_encodings = []

images = os.listdir(path)

for _ in images:
    image = fr.load_image_file(path + _)
    image_path = path + _
    encoding = fr.face_encodings(image)[0]

    known_name_encodings.append(encoding)
    known_names.append(os.path.splitext(os.path.basename(image_path))[0].capitalize())
print(known_names)
test_image = "/content/drive/MyDrive/Colab Notebooks/test/test.jpg"
image = cv2.imread(test_image)
if image is None:
    raise ValueError("Image is not loaded.")
face_locations = fr.face_locations(image, number_of_times_to_upsample=1)
face_encodings = fr.face_encodings(image, face_locations)
for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
   matches = fr.compare_faces(known_name_encodings, face_encoding)
   name = ""

   face_distances = fr.face_distance(known_name_encodings, face_encoding)
   best_match = np.argmin(face_distances)

   if matches[best_match]:
       name = known_names[best_match]

   cv2.rectangle(image, (left, top), (right, bottom), (0, 0, 255), 2)
   cv2.rectangle(image, (left, bottom - 15), (right, bottom), (0, 0, 255), cv2.FILLED)

   font = cv2.FONT_HERSHEY_DUPLEX
   cv2.putText(image, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)
```

```
cv2_imshow(image)
cv2.imwrite("/content/drive/MyDrive/Colab Notebooks/output/output.jpg", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



RESULT:

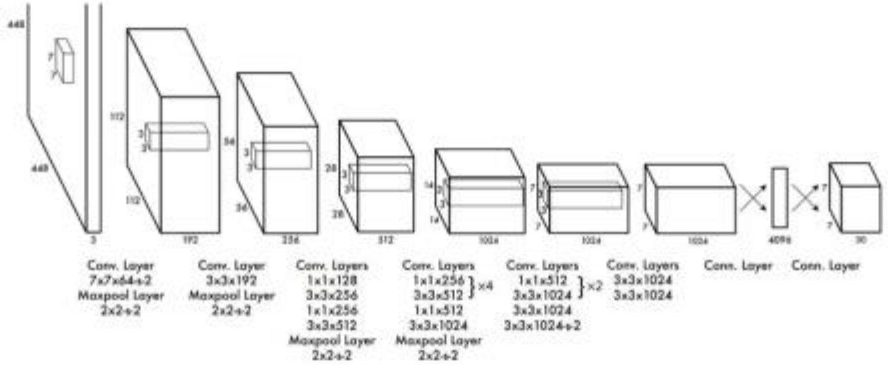
Thus the python program for Facial Detection and Recognition was implemented and output is obtained successfully.

EX NO : 07	**EVENT DETECTION IN VIDEO SURVEILLANCE SYSTEM**
DATE :	

AIM:

To Write a program for event detection in video surveillance system

ALGORITHM:(ARCHITECTURE):

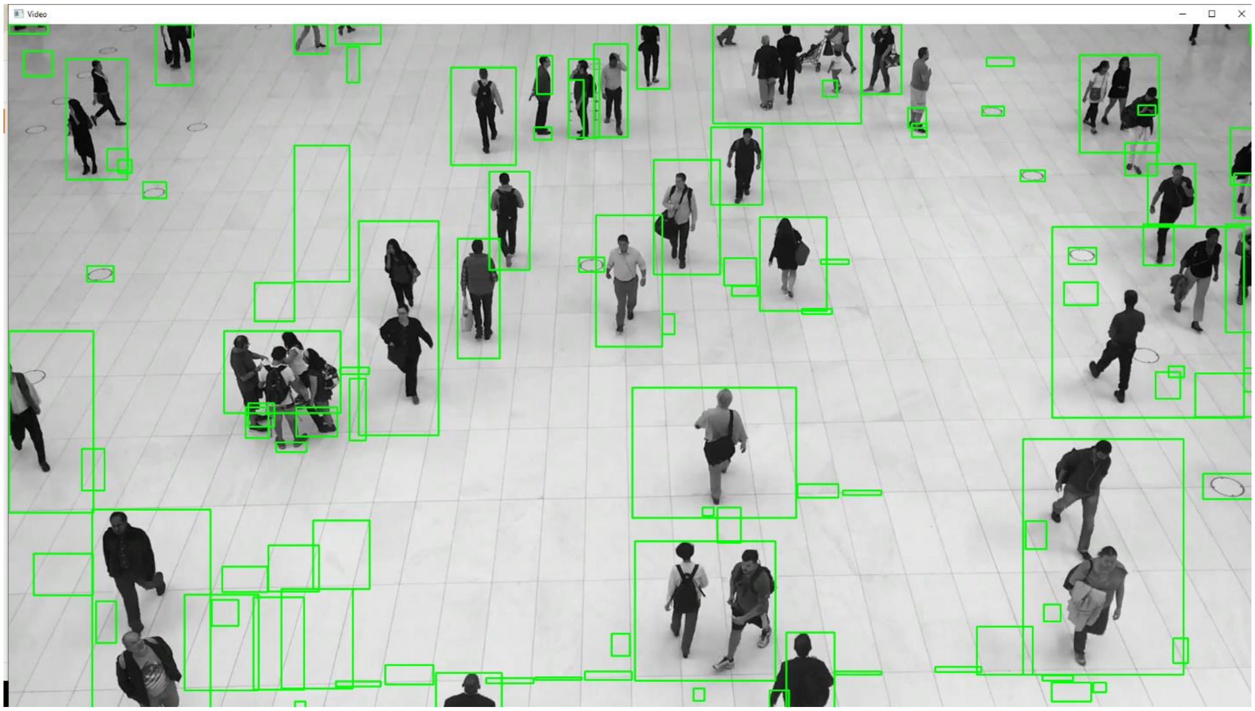


The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 × 1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224 × 224 input image) and then double the resolution for detection.

PROGRAM:

```
import cv2
# Initialize video capture
video_capture = cv2.VideoCapture("background-video-people-walking-1080-
ytshorts.savetube.me.mp4") # Replace with your video file
# Initialize background subtractor
bg_subtractor = cv2.createBackgroundSubtractorMOG2()
while video_capture.isOpened():
 ret, frame = video_capture.read()
if not ret:
 break
 # Apply background subtraction
 fg_mask = bg_subtractor.apply(frame)
 # Apply thresholding to get a binary mask
 _, thresh = cv2.threshold(fg_mask, 50, 255, cv2.THRESH_BINARY)

 # Find contours
 contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

 for contour in contours:
 # Filter contours based on area (adjust the threshold as needed)
 if cv2.contourArea(contour) > 100:
 # Draw a bounding box around detected objects or events
 x, y, w, h = cv2.boundingRect(contour)
 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

 # Display the processed frame
 cv2.imshow('Video', frame)
 if cv2.waitKey(1) & 0xFF == ord('q'):
 break
# Release video capture and close OpenCV windows
video_capture.release()
cv2.destroyAllWindows()
```

OUTPUT:



RESULT:

Thus the python program for event detection in video surveillance system was implemented and output is obtained successfully.