PRACTICAL 1

Write a program to Implement common styles of image augmentation.

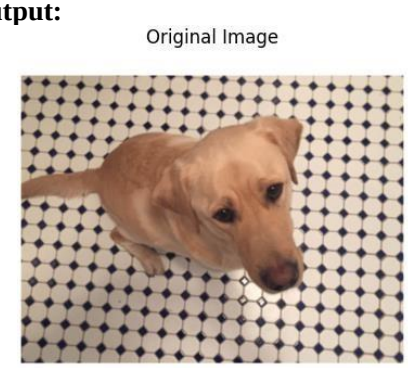
A. Random Image Warping Transformations

Code :

```
import cv2 as cv
import matplotlib.pyplot as plt
import albumentations as A
from skimage.io import imread
def display_image(title, image):
   plt.title(title)
   plt.imshow(image)
   plt.axis('off')
   plt.show()
imOriginal = cv.resize(imread("D:/Images_/kobi.png"),(0, 0),
                  fx=2, fy=2).astype('float32')/ 255

display_image("Original Image",imOriginal)
transform = A.Compose([
A. Affine(
rotate=(-45, 45),
scale=(0.8, 1.2),
translate_percent={'x': (-0.1, 0.1), 'y': (-0.1, 0.1)},
p=1.0
)
])
imAugmented = transform(image=imOriginal)['image']
display_image("Random Image Warping Transformation", imAugmented)
```

Output:



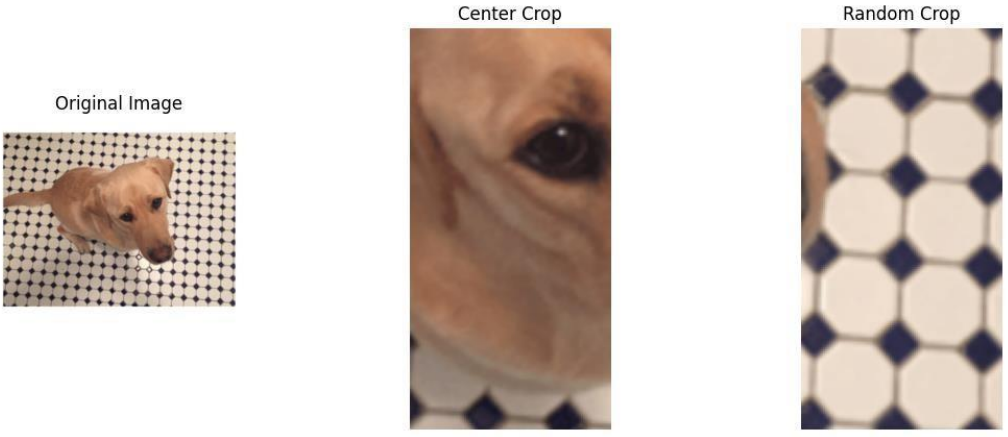
B. Cropping Transformations

Code:

```
import cv2 as cv
import matplotlib.pyplot as plt
from skimage.io import imread
import numpy as np
import torchvision.transforms as T
from PIL import Image

imOriginal =cv.resize(imread("D:/Images_/kobi.png"),(0, 0),
                      fx=2, fy=2).astype('float32')/ 255
imOriginal = Image.fromarray((imOriginal * 255).astype(np.uint8))
targetSize = (200, 100)
center_crop = T.CenterCrop(targetSize)
imCenterCrop = center_crop(imOriginal)
random_crop = T.RandomCrop(targetSize)
imRandomCrop = random_crop(imOriginal)
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(imOriginal)
axes[0].set_title("Original Image")
axes[0].axis('off')
axes[1].imshow(imCenterCrop)
axes[1].set_title("Center Crop")
axes[1].axis('off')
axes[2].imshow(imRandomCrop)
axes[2].set_title("Random Crop")
axes[2].axis('off')
plt.show()
```

Output;



C. Color Transformations

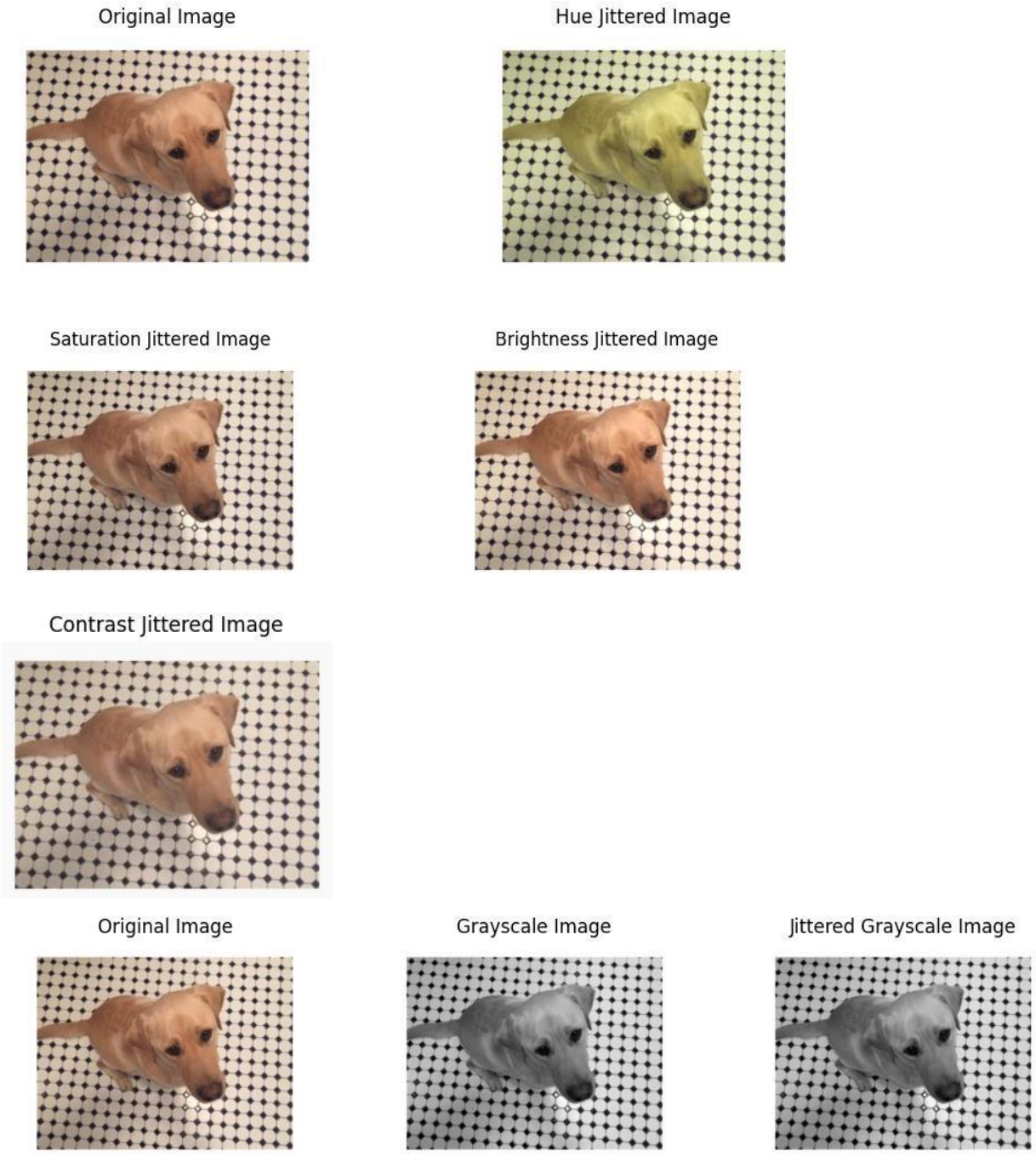
Code :

```
import cv2 as cv
import matplotlib.pyplot  as plt
from skimage.io import imread
from torchvision import transforms
from PIL import Image
import numpy as np

imOriginal = cv.resize(imread("D:/Images_/kobi.png"),(0, 0),fx=2, fy=2).astype('float32')/ 255
imOriginal = Image.fromarray((imOriginal * 255).astype(np.uint8))
def apply_color_jitter(image, brightness=0, contrast=0, saturation=0, hue=0):
    transform = transforms.ColorJitter(brightness=brightness, contrast=contrast,
saturation=saturation, hue=hue)
    jittered_image = transform(image)
    return jittered_image
imJittered = apply_color_jitter(imOriginal, hue=0.1)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(imOriginal)
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(imJittered)
plt.title('Hue Jittered Image')
plt.axis('off')
plt.show()
imJittered = apply_color_jitter(imOriginal, saturation=0.5)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(imJittered)
plt.title('Saturation Jittered Image')
plt.axis('off')
imJittered = apply_color_jitter(imOriginal, brightness=0.3)
plt.subplot(1, 2, 2)
plt.imshow(imJittered)
plt.title('Brightness Jittered Image')
plt.axis('off')
plt.show()
imJittered = apply_color_jitter(imOriginal, contrast=0.5)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(imJittered)
plt.title('Contrast Jittered Image')
plt.axis('off')
plt.show()
imGray = cv.cvtColor(np.array(imOriginal), cv.COLOR_RGB2GRAY).astype('float32') /255
contrastFactor = 1 - 0.2 * np.random.rand()
brightnessOffset = 0.3 * (np.random.rand() - 0.5)
imJittered = np.clip(imGray * contrastFactor + brightnessOffset, 0, 1)
imJittered = (imJittered * 255).astype(np.uint8)
fig, axes = plt.subplots(1, 3, constrained_layout=True,figsize=(10, 5))
axes[0].imshow(imOriginal)
```

```
axes[0].set_title("Original Image")
axes[0].axis('off')
axes[1].imshow(imGray, cmap='gray')
axes[1].set_title("Grayscale Image")
axes[1].axis('off')
axes[2].imshow(imJittered, cmap='gray')
axes[2].set_title("Jittered Grayscale Image")
axes[2].axis('off')
plt.show()
```

Output:

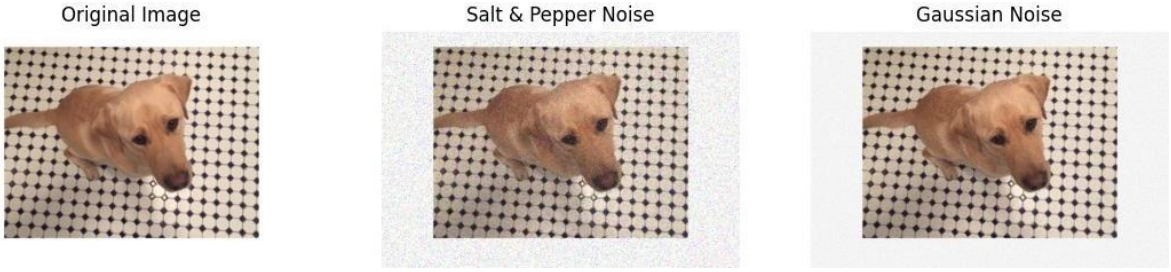


D. Synthetic Noise

Code:

```
import cv2 as cv
import matplotlib.pyplot as plt
from skimage.io import imread
from skimage.util import random_noise
import numpy as np
imOriginal = cv.resize(imread("D:/Images_/kobi.png"),(0, 0),fx=2, fy=2).astype('float32')/ 255
imSaltAndPepperNoise = random_noise(imOriginal, mode='s&p', amount=0.1)
imGaussianNoise = random_noise(imOriginal, mode='gaussian', var=0.01)
fig, axes = plt.subplots(1, 3, figsize=(25, 5))
axes[0].imshow(imOriginal)
axes[0].set_title("Original Image")
axes[0].axis('off')
axes[1].imshow(imSaltAndPepperNoise)
axes[1].set_title("Salt & Pepper Noise")
axes[1].axis('off')
axes[2].imshow(imGaussianNoise)
axes[2].set_title("Gaussian Noise")
axes[2].axis('off')
plt.show()
```

Output:

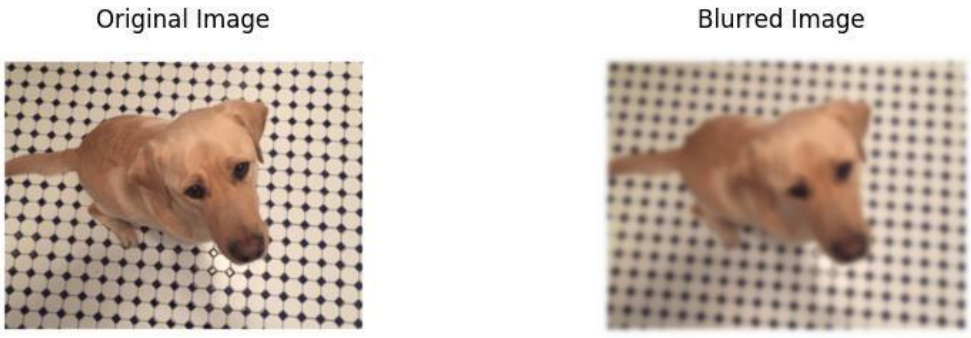


E. Synthetic Blur

Code:

```
import cv2 as cv
import matplotlib.pyplot as plt
from skimage.io import imread
from scipy.ndimage import gaussian_filter
from PIL import Image
import numpy as np
imOriginal = cv.resize(imread("D:/Images_/kobi.png"),(0, 0),fx=2, fy=2).astype('float32')/ 255
imOriginal = (imOriginal * 255).astype(np.uint8)
sigma = 1 + 5 * np.random.rand()
imBlurred = gaussian_filter(imOriginal, sigma=(sigma, sigma, 0))
plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.imshow(imOriginal)
plt.title('Original Image')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(imBlurred)
plt.title('Blurred Image ')
plt.axis('off')
plt.show()
```

Output:



`

PRACTICAL 2

Write a program to Implement Pose estimation

Code:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
input_image_path = "D:/Images -20240804T060317Z-001/Images_/person.jpg"

threshold = 0.2
inWidth = 368
inHeight = 368

BODY_PARTS = { "Nose": 0, "Neck": 1, "RShoulder": 2, "RElbow": 3, "RWrist": 4,
               "LShoulder": 5, "LElbow": 6, "LWrist": 7, "RHip": 8, "RKnee": 9,
               "RAnkle": 10, "LHip": 11, "LKnee": 12, "LAnkle": 13, "REye": 14,
               "LEye": 15, "REar": 16, "LEar": 17, "Background": 18 }
POSE_PAIRS = [ ["Neck", "RShoulder"], ["Neck", "LShoulder"], ["RShoulder",
"RElbow"],
               ["RElbow", "RWrist"], ["LShoulder", "LElbow"], ["LElbow", "LWrist"],
               ["Neck", "RHip"], ["RHip", "RKnee"], ["RKnee", "RAnkle"], ["Neck", "LHip"],
               ["LHip", "LKnee"], ["LKnee", "LAnkle"], ["Neck", "Nose"], ["Nose", "REye"],
               ["REye", "REar"], ["Nose", "LEye"], ["LEye", "LEar"] ]
net = cv.dnn.readNetFromTensorflow("D:/Images -20240804T060317Z-
001/Images_/graph_opt.pb")
frame = cv.imread(input_image_path)
if frame is None:
    raise ValueError(f"Image at path '{input_image_path}' could not be loaded.")

frameWidth = frame.shape[1]
frameHeight = frame.shape[0]
net.setInput(cv.dnn.blobFromImage(frame, 1.0, (inWidth, inHeight), (127.5, 127.5, 127.5),
swapRB=True, crop=False))
out = net.forward()
out = out[:, :19, :, :] # MobileNet output [1, 57, -1, -1], we only need the first 19 elements

assert(len(BODY_PARTS) == out.shape[1])

points = []
for i in range(len(BODY_PARTS)):
    heatMap = out[0, i, :, :]
    _, conf, _, point = cv.minMaxLoc(heatMap)
    x = (frameWidth * point[0]) / out.shape[3]
    y = (frameHeight * point[1]) / out.shape[2]
# Add a point if its confidence is higher than the threshold.
    points.append((int(x), int(y)) if conf > threshold else None)
for pair in POSE_PAIRS:
```

```
    partFrom = pair[0]
    partTo = pair[1]
    assert(partFrom in BODY_PARTS)
    assert(partTo in BODY_PARTS)

    idFrom = BODY_PARTS[partFrom]
    idTo = BODY_PARTS[partTo]

    if points[idFrom] and points[idTo]:
        cv.line(frame, points[idFrom], points[idTo], (0, 255, 0), 3)
        cv.ellipse(frame, points[idFrom], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
        cv.ellipse(frame, points[idTo], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
t, _ = net.getPerfProfile()
freq = cv.getTickFrequency() / 1000
cv.putText(frame, '%.2fms' % (t / freq), (10, 20), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0,
0, 0))
frame_rgb = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
plt.imshow(frame_rgb)
plt.title('OpenPose using OpenCV')
plt.axis('off')
plt.show()
```

Output:

PRACTICAL 3

Write a program to Implement Feature Based Panoramic Image Stitching.

Code:

```
import cv2
import matplotlib.pyplot as plt
# Load images individually
image1 = cv2.imread('D:/Images -20240804T060317Z-001/Images_/data/001.jpg')
image2 = cv2.imread('D:/Images -20240804T060317Z-001/Images_/data/002.jpg')
image3 = cv2.imread('D:/Images -20240804T060317Z-001/Images_/data/003.jpg')
image4 = cv2.imread('D:/Images -20240804T060317Z-001/Images_/data/004.jpg')
image5 = cv2.imread('D:/Images -20240804T060317Z-001/Images_/data/005.jpg')
if image1 is None or image2 is None or image3 is None or image4 is None or image5 is None:
    print("One or more images not found")
else:
    images = [image1, image2, image3, image4, image5]
    stitcher = cv2.Stitcher_create()
    status, stitched = stitcher.stitch(images)
    if status == cv2.Stitcher_OK:
        print("Stitching completed successfully.")
        stitched_rgb = cv2.cvtColor(stitched, cv2.COLOR_BGR2RGB)
        plt.imshow(stitched_rgb)
        plt.axis('off')
        plt.show()
        cv2.imwrite('stitched_image.jpg', stitched)
    else:
        print(f"Stitching failed with status {status}.")
```

Output:

PRACTICAL 4

Write a program to Implement Image Blending techniques
A) Simple blending B) Weighted blending C) Mask blending

Code:

```
import cv2
from matplotlib import pyplot as plt
from skimage.io import imread
image1 = cv2.resize(imread("D:\\Images -20240818T074730Z-001\\Images_\\forest.jpg"), (0,0),
fx=2, fy=2).astype('float32') / 255
image2 = cv2.resize(imread("D:\\Images -20240818T074730Z-001\\Images_\\birds.jpg"), (0,0),
fx=2, fy=2).astype('float32') / 255
image1=cv2.resize(image1,(image2.shape[1],image2.shape[0]))
plt.figure(figsize=(10,5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image1, cv2.COLOR_BGR2RGB))
plt.title('Image1')
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(image2, cv2.COLOR_BGR2RGB))
plt.title('Image2')
plt.tight_layout()
plt.show()
blended=cv2.add(image1,image2)
plt.imshow(cv2.cvtColor(blended, cv2.COLOR_BGR2RGB))
plt.title("Simple Blended Image")
plt.show()
alpha = 0.7
beta = 0.3
blended = cv2.addWeighted(image1, alpha, image2, beta, 0)
plt.imshow(cv2.cvtColor(blended, cv2.COLOR_BGR2RGB))
plt.title("Weighted Blended Image")
plt.show()
mask = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
_, mask = cv2.threshold(mask, 1, 255, cv2.THRESH_BINARY)
foreground = cv2.bitwise_and(image2, image2, mask=mask)
background = cv2.bitwise_and(image1, image1, mask=mask)
result = cv2.add(foreground, background)

plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
plt.title("Mask Blended Image")
plt.show()
```

Output:

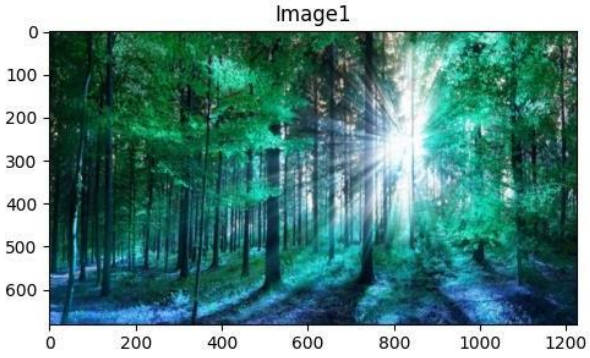


Image1
0
100
200
300
400
500
600
0
200
400
600
800
1000
1200

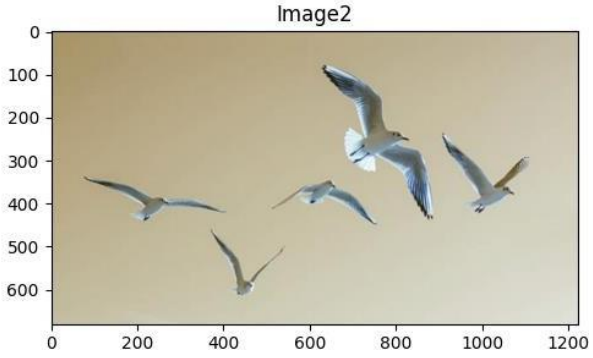
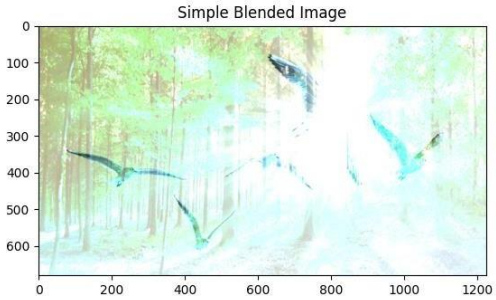
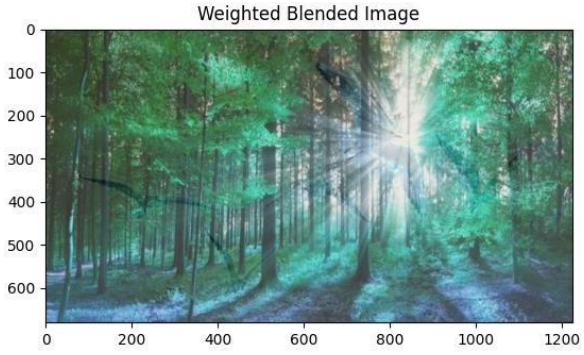


Image2
0
100
200
300
400
500
600
0
200
400
600
800
1000
1200


Simple Blended Image
0
100
200
300
400
500
600
0
200
400
600
800
1000
1200


Weighted Blended Image
0
100
200
300
400
500
600
0
200
400
600
800
1000
1200

PRACTICAL 5

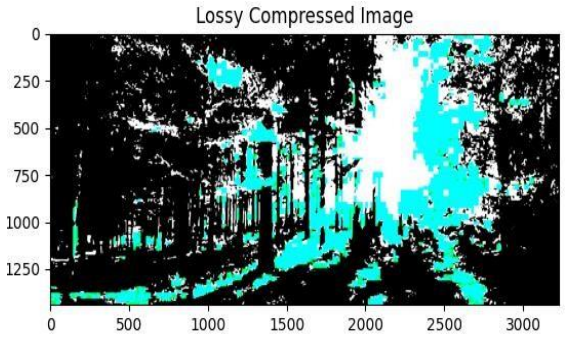
Write a program to Implement Image Compression.

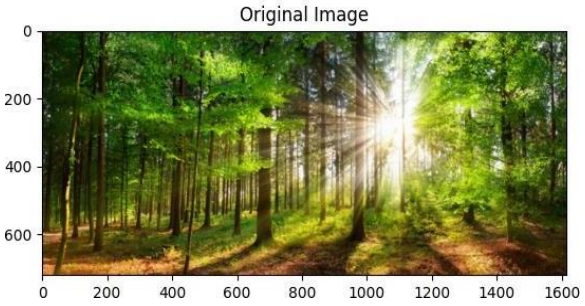
Code:

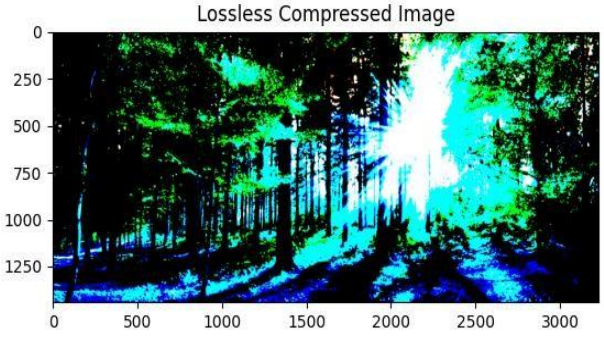
```
import cv2
from skimage.io import imread
import os
import matplotlib.pyplot as plt
img = cv2.resize(imread("D:\\Images -20240818T074730Z-001\\Images_\\forest.jpg"),
(0, 0), fx=2, fy=2).astype('float32') / 255
cv2.imwrite('lossless_compressed_image.png', img)
jpeg_quality = 90
cv2.imwrite('lossy_compressed_image.jpg', img, [cv2.IMWRITE_JPEG_QUALITY,
jpeg_quality])
original_size = os.path.getsize('D:\\Images -20240818T074730Z-001\\Images_\\forest.jpg')
lossless_size = os.path.getsize('lossless_compressed_image.png')
lossy_size = os.path.getsize('lossy_compressed_image.jpg')
print(f'Original image size: {original_size} bytes')
print(f'Lossless compressed image size: {lossless_size} bytes')
print(f'Lossy compressed image size: {lossy_size} bytes')
lossless_img = cv2.resize(imread("lossless_compressed_image.png"), (0, 0), fx=2,
fy=2).astype('float32')
lossy_img = cv2.resize(imread("lossy_compressed_image.jpg"), (0, 0), fx=2,
fy=2).astype('float32')
plt.imshow(img)
plt.title('Original Image')
plt.show()
plt.imshow(lossless_img)
plt.title('Lossless Compressed Image')
plt.show()
plt.imshow(lossy_img)
plt.title('Lossy Compressed Image')
plt.show()
```

Output:

```
Original image size: 173882 bytes
Lossless compressed image size: 148116 bytes
Lossy compressed image size: 26415 bytes
```







PRACTICAL 6

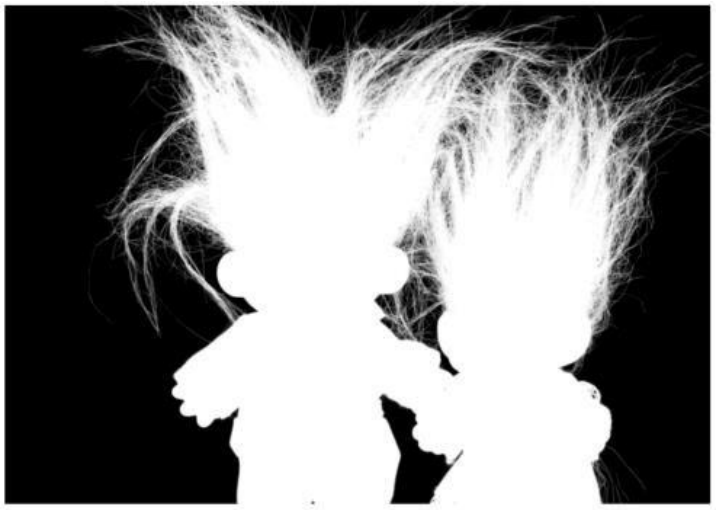
Write a program to Implement Image Matting technique.

Code:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
I = cv2.imread('D:\Matting-20240901T071408Z-001\Matting\matting1.png')
I=cv2.cvtColor(I,cv2.COLOR_BGR2RGB)/255
plt.imshow(I),plt.axis('off')
plt.show()
alpha_ex=cv2.imread('D:\Matting-20240901T071408Z-
001\Matting\matting2.png',cv2.IMREAD_GRAYSCALE)/255
plt.imshow(alpha_ex,cmap='gray'),plt.axis('off')
plt.show()
```

```
n_rows=I.shape[0]
n_cols=I.shape[1]
n_pixels=n_rows*n_cols
I=np.reshape(I,(n_pixels,3))
I[:,0]=I[:,1]
I[:,2]=I[:,1]
alpha_ex=np.reshape(alpha_ex,(n_pixels,1))
G_B=1
I=alpha_ex*I+(1-alpha_ex)*[0,G_B,0]
I = np.reshape(I, (n_rows, n_cols, 3))
plt.imshow(I), plt.axis('off')
plt.show();
```

```
R_I = I[:, :, 0]
G_I = I[:, :, 1]
B_I = I[:, :, 2]
alpha = (R_I - (G_I - G_B))/G_B
plt.imshow(alpha, cmap='gray'), plt.axis('off')
plt.show();
alpha = np.reshape(alpha, (n_pixels, 1))
error = np.linalg.norm(alpha - alpha_ex)/np.linalg.norm(alpha_ex)
print(f'Error (alpha): {error:.2e}')
K = cv2.imread('D:\\Matting-20240901T071408Z-001\\Matting\\bg.png')
K = cv2.cvtColor(K, cv2.COLOR_BGR2RGB)/255
K = K[:n_rows, :n_cols, :]
plt.imshow(K), plt.axis('off')
plt.show();
K = np.reshape(K, (n_pixels, 3))
R_I = np.reshape(R_I, (n_pixels, 1))
J = np.tile(R_I, 3) + (1 - alpha) * K
J = np.reshape(J, (n_rows, n_cols, 3))
plt.imshow(J), plt.axis('off')
plt.show();
```



```
Error (alpha): 3.53e-17
```

PRACTICAL 7

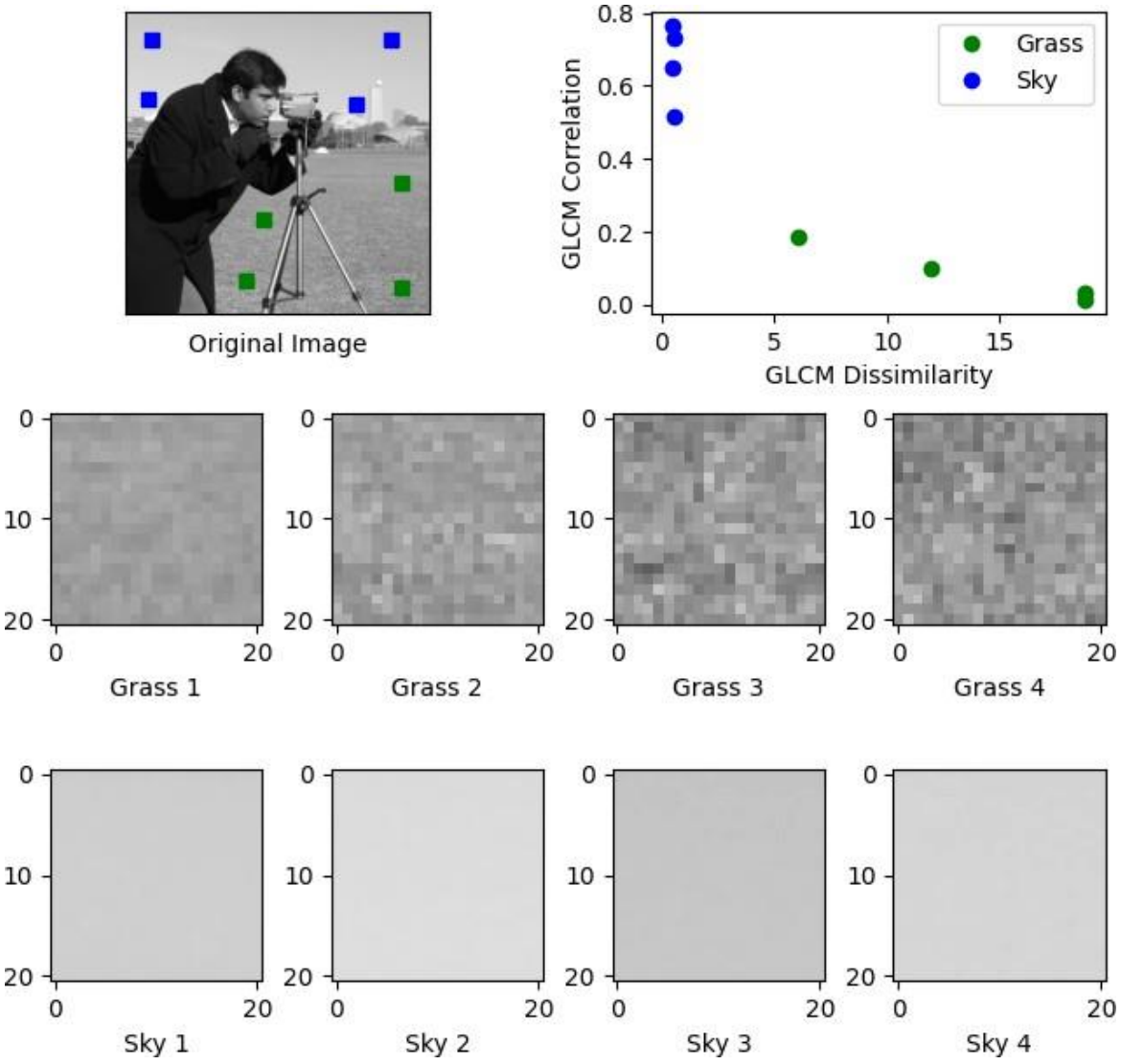
Write a program to Implement Texture analysis.

Code 1:

```
import matplotlib.pyplot as plt
from skimage.feature import graycomatrix, graycoprops
from skimage import data
PATCH_SIZE=21
image=data.camera()
grass_locations = [(280, 454), (342, 223), (444, 192), (455, 455)]
grass_patches = [image[loc[0]:loc[0] + PATCH_SIZE, loc[1]:loc[1] + PATCH_SIZE] for loc in
grass_locations]
sky_locations = [(38, 34), (139, 28), (37, 437), (145, 379)]
sky_patches = [image[loc[0]:loc[0] + PATCH_SIZE, loc[1]:loc[1] + PATCH_SIZE] for loc in
sky_locations]
xs=[]
ys=[]
for patch in grass_patches+sky_patches:
    glcm=graycomatrix(patch,distances=[5],angles=[0],levels=256,symmetric=True,normed=True)
    xs.append(graycoprops(glcm,'dissimilarity')[0,0])
    ys.append(graycoprops(glcm,'correlation')[0,0])
fig=plt.figure(figsize=(8,8))
ax = fig.add_subplot(3, 2, 1)
ax.imshow(image, cmap=plt.cm.gray, vmin=0, vmax=255)
for y, x in grass_locations:
    ax.plot(x+PATCH_SIZE/2,y+PATCH_SIZE/2,'gs')
for y,x in sky_locations:
    ax.plot(x+PATCH_SIZE/2,y+PATCH_SIZE/2,'bs')
ax.set_xlabel('Original Image')
ax.set_xticks([])
ax.set_yticks([])
ax.axis('image')


ax = fig.add_subplot(3, 2, 2)
ax.plot(xs[:len(grass_patches)], ys[:len(grass_patches)], 'go', label='Grass')
ax.plot(xs[len(grass_patches):], ys[len(grass_patches):], 'bo', label='Sky')
ax.set_xlabel('GLCM Dissimilarity')
ax.set_ylabel('GLCM Correlation')
ax.legend()
for i,patch in enumerate(grass_patches):
    ax = fig.add_subplot(3, len(grass_patches), len(grass_patches) * 1 + i + 1)
    ax.imshow(patch, cmap=plt.cm.gray, vmin=0, vmax=255)
    ax.set_xlabel(f"Grass {i + 1}")
for i, patch in enumerate(sky_patches):
    ax = fig.add_subplot(3, len(sky_patches), len(sky_patches) * 2 + i + 1)
    ax.imshow(patch, cmap=plt.cm.gray, vmin=0, vmax=255)
    ax.set_xlabel(f"Sky {i + 1}")
fig.suptitle('Gray Level Co-occurrence Matrix Features', fontsize=14, y=1.05)
plt.tight_layout()
plt.show()
```

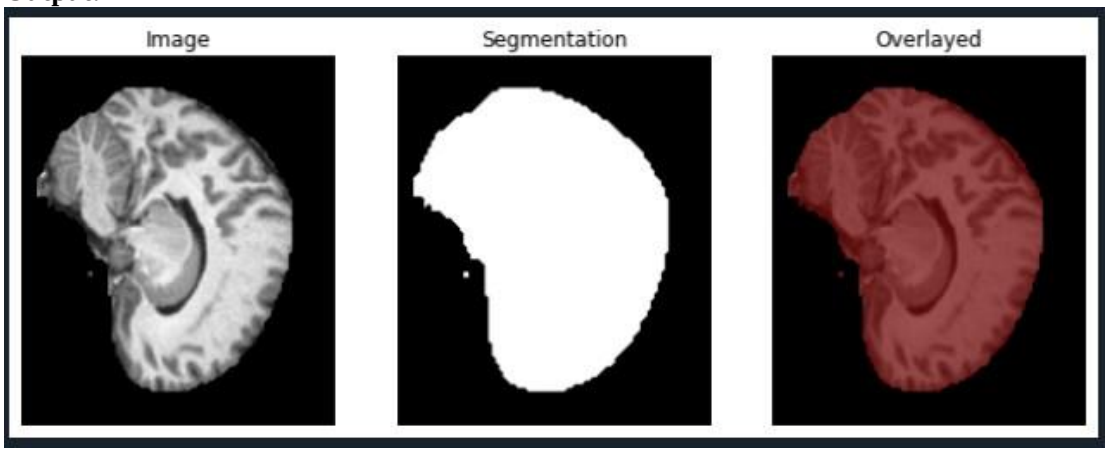
Output:


Original Image
GLCM Correlation
GLCM Dissimilarity
Grass
Sky
Grass 1
Grass 2
Grass 3
Grass 4
Sky 1
Sky 2
Sky 3
Sky 4

Code 2:

```
import seaborn as sns
from skimage.color import label2rgb
import numpy as np
import matplotlib.pyplot as plt
import h5py
slice_img = (h5py.File('C:/Users/Yash/Downloads/test/mri_00041615.h5')['image'][100][:, :,
0]/3200*255).astype('uint8')
slice_mask = slice_img>0
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(11, 5))
ax1.imshow(slice_img, cmap='gray')
ax1.axis('off')
ax1.set_title('Image')
ax2.imshow(slice_mask, cmap='gray')
ax2.axis('off')
ax2.set_title('Segmentation')
ax3.imshow(label2rgb(slice_mask > 0, slice_img, bg_label=0, ))
ax3.axis('off')
ax3.set_title('Overlayed')
plt.show()
```

Output:



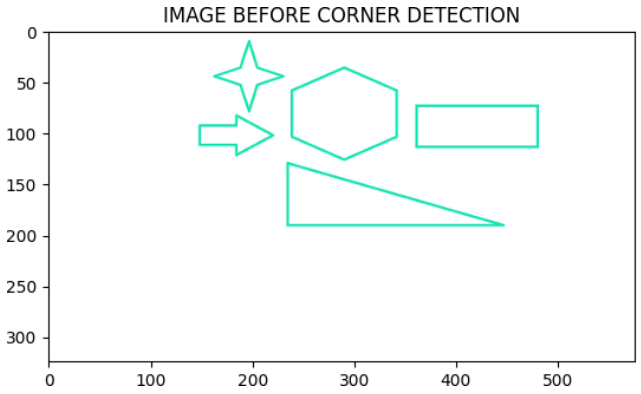
PRACTICAL 8

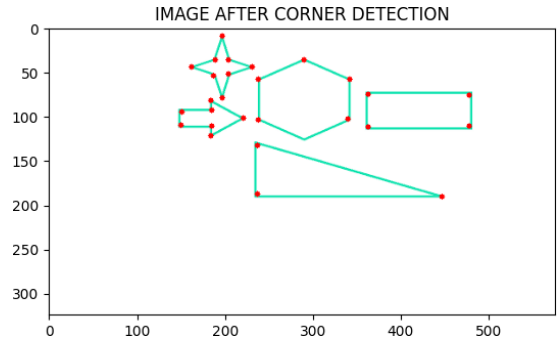
Write a program to Implement detect corner of an image using OpenCV.

Code:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
img=cv2.imread('D:\\Images -20240818T074730Z-001\\Images_\\corner.png')
plt.title('IMAGE BEFORE CORNER DETECTION')
plt.imshow(img), plt.show()
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
corners = cv2.goodFeaturesToTrack(gray, 27, 0.01, 10)
corners = np.int0(corners)
for i in corners:
    x,y=i.ravel()
    cv2.circle(img,(x,y),3,255,-1)
plt.title('IMAGE AFTER CORNER DETECTION')
plt.imshow(img),plt.show()
```

Output:





PRACTICAL 9

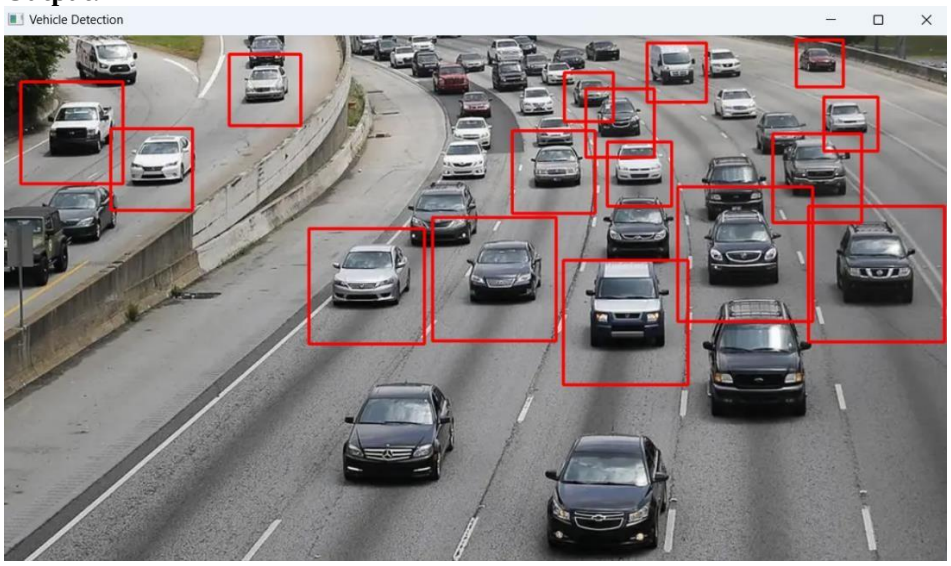
A. Write a program to Implement Vehicle detection using OpenCV Python.

Code:

```
import cv2
car_cascade = cv2.CascadeClassifier('C:/Users/Yash/Downloads/Images -20241020T073657Z-001/Images/haarcascades/cars.xml')
def detect_vehicles(image_path):

    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cars = car_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
    for (x, y, w, h) in cars:
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
    cv2.imshow('Vehicle Detection', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
image_path = 'C:/Users/Yash/Downloads/Images -20241020T073657Z-001/Images/cars.jpeg'
detect_vehicles(image_path)
```

Output:



B. License Plate Recognition with OpenCV and Tesseract OCR

Code:

```
import cv2
import matplotlib.pyplot as plt
haar_cascade = 'C:/Users/Yash/Downloads/Images -20241020T073657Z-
001/Images/haarcascades/cars.xml'
video = 'C:/Users/Yash/Downloads/Images -20241020T073657Z-001/Images/video.avi'
cap = cv2.VideoCapture(video)
car_cascade = cv2.CascadeClassifier(haar_cascade)
ret, frames = cap.read()
gray = cv2.cvtColor(frames, cv2.COLOR_BGR2GRAY)
cars = car_cascade.detectMultiScale(gray, 1.1,  1)
for (x,y,w,h) in cars:
cv2.rectangle(frames,(x,y),(x+w,y+h),(0,0,255),2)
cv2.imshow('video', frames)
while True:
    ret, frames = cap.read()
    gray = cv2.cvtColor(frames, cv2.COLOR_BGR2GRAY)
    cars = car_cascade.detectMultiScale(gray, 1.1, 1)
    for (x,y,w,h) in cars:
        cv2.rectangle(frames,(x,y),(x+w,y+h),(0,0,255),2)
    cv2.imshow('video', frames)
    if cv2.waitKey(33) == 27:
        break
cv2.destroyAllWindows()
```

Output:

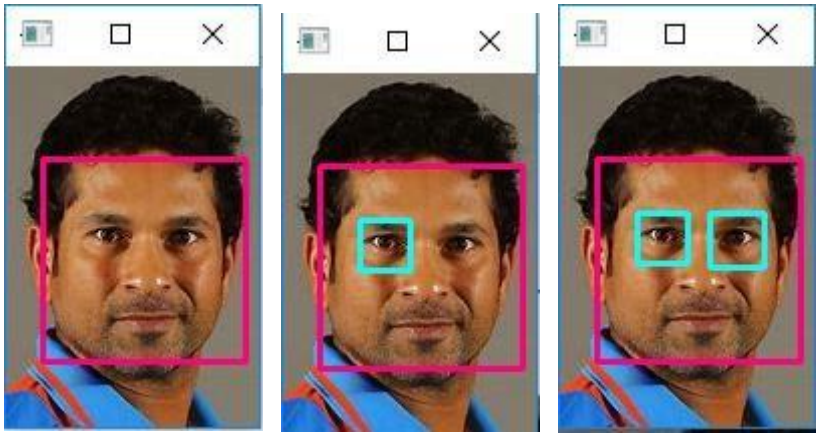
PRACTICAL 10

Write a program for Face Detection.

Code:

```
import numpy as np
import cv2
face_classifier=cv2.CascadeClassifier('D:\Images -20240818T074730Z-
001\Images_\haarcascades\haarcascade_frontalface_default.xml')
eye_classifier = cv2.CascadeClassifier('D:\Images -20240818T074730Z-
001\Images_\haarcascades\haarcascade_eye.xml')
img = cv2.imread('D:\Images -20240818T074730Z-001\Images_\Sachin.jpeg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_classifier.detectMultiScale(gray, 1.05, 3)
if faces is ():
    print("No Face Found")
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(127,0,255),2)
    cv2.imshow('img',img)
    cv2.waitKey(0)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_classifier.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(255,255,0),2)
        cv2.imshow('img',img)
        cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Output:



Code :

```
import cv2

img = cv2.imread('C:/Users/Yash/Downloads/Images -20241020T073657Z-
001/Images/womens-cricket-team.jpg')

gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Loading the required haar-cascade xml classifier file

haar_cascade = cv2.CascadeClassifier('C:/Users/Yash/Downloads/Images -20241020T073657Z-
001/Images/haarcascades/haarcascade_frontalface_default.xml')

faces_rect = haar_cascade.detectMultiScale(gray_img, 1.1, 9)

for (x, y, w, h) in faces_rect:
 cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
cv2.imshow('Detected faces', img)
cv2.waitKey(0)
```

Output: