**Practical 1**

**Aim:- Face Detection**

**A) Code:-**

# detect face from input image and save it on the disk.

import cv2

# Load the pre-trained Haar Cascade model for face detection

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

# load the image where you want to detect face

image\_path = 'image.jpg' # path to your image

image = cv2.imread(image\_path)

# convert the image to grayscale

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# detect faces in the image

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30,30))

# draw rectangles around each face

for (x,y,w,h) in faces:

cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)

# save the image with faces highlighted

output\_path = 'faces\_detected.jpg' # corrected file extension

cv2.imwrite(output\_path, image)

print(f"Faces Detected: {len(faces)}. Output saved to {output\_path}")

**B) Code:-**

# detect faces and show it on screen

import cv2

import matplotlib.pyplot as plt

# initialize the Haar Cascade face detection model

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

# start capturing video from the camera (default camera is usually at index 0)

cap = cv2.VideoCapture(0)

# Capture a single frame

ret, frame = cap.read()

if ret: # check if the frame was successfully captured

# convert the captured frame to grayscale

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# detect faces in the grayscale frame

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30,30))

# draw rectangles around the detected faces

for (x,y,w,h) in faces:

cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0), 2)

# Display the resulting frame with faces highlighted using matplotlib

plt.imshow(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

plt.axis('off') # turn off axis labels

plt.show()

# Release the capture

cap.release()

print('Number of faces detected: ',len(faces))

**Practical 2**

**Aim:- Pedestrians Detection**

**A) Code for Ideal Script:-**

import cv2

import matplotlib.pyplot as plt

import matplotlib.animation as animation

# Initialize the Haar Cascade pedestrian detection model

pedestrians\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_fullbody.xml')

# Load the video

video\_path = 'C:/Users/DELL/OneDrive/Desktop/MSc IT-1/video.mp4'

cap = cv2.VideoCapture(video\_path)

# Check if the video is opened successfully

if not cap.isOpened():

print("Error: Unable to open the video")

exit()

# Function to detect pedestrians and draw bounding boxes

def detect\_pedestrians(frame):

# Convert the frame to grayscale

gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Detect pedestrians in the grayscale frame

pedestrians = pedestrians\_cascade.detectMultiScale(gray\_frame, scaleFactor=1.1, minNeighbors=5, minSize=(30,30))

# Draw rectangle around the detected pedestrians

for (x,y,w,h) in pedestrians:

cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 2)

return frame

# Create a subplot for displaying the video frame

fig, ax = plt.subplots()

# Function to update the animation

def update(frame):

# Read a frame from the video

ret, frame = cap.read()

if not ret:

ani.event\_source.stop()

return

# Detect pedestrians and draw bounding boxes

frame\_with\_pedestrians = detect\_pedestrians(frame)

# Display the frame

ax.clear()

ax.imshow(cv2.cvtColor(frame\_with\_pedestrians, cv2.COLOR\_BGR2RGB))

ax.axis('off') # Turn off axis labels

# Create the animation

ani = animation.FuncAnimation(fig, update, interval=50)

plt.show()

# Release the video capture object

cap.release()

**B) Code for Jupyter:-**

import cv2

import matplotlib.pyplot as plt

# Initialize the Haar Cascade pedestrian detection model

pedestrian\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_fullbody.xml')

# Load the video

video\_path = 'C:/Users/DELL/OneDrive/Desktop/MSc IT-1/video.mp4'

cap = cv2.VideoCapture(video\_path)

# Check if the video is opened successfully

if not cap.isOpened():

print("Error: Unable to open the video")

exit()

# Loop through the video frames

while True:

# Read a frame from the video

ret, frame = cap.read()

if not ret:

break # Break the loop if no frame is captured

# Convert the frame to grayScale

gray\_frame = cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)

# Detect pedestrians in the grayscale frame

pedestrians = pedestrian\_cascade.detectMultiScale(gray\_frame, scaleFactor=1.1, minNeighbors=5, minSize=(30,30))

# Draw rectangle around the detected pedestrians

for (x,y,w,h) in pedestrians:

cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 2)

# Display the resulting frame with pedestrians highlighted using Matplotlib

plt.imshow(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

plt.axis('off') # Turn off the axis

plt.show()

# Check if the loop should be exited

# In Jupyter Notebook you might need to manually interrupt the kernel to stop the loop

# Release the video capture object

cap.release()

**Practical 3**

**Aim:- Object Detection**

**Code:-**

import argparse

import numpy as np

import cv2

import matplotlib.pyplot as plt

args = argparse.Namespace(

image="dog.jpg",

weights="yolov3.weights",

config="yolov3.cfg",

classes="yolov3.txt"

)

def get\_output\_layers(net):

layer\_names = net.getLayerNames()

try:

output\_layers = [layer\_names[i-1] for i in net.getUnconnectedOutLayers()]

except:

output\_layers = [layer\_names[i-1] for i in net.getUnconnectedOutLayers()]

return output\_layers

def draw\_prediction(img, class\_id, confidence, x, y, x\_plus\_w, y\_plus\_h):

label = str(classes[class\_id])

color = COLORS(class\_id)

cv2.rectangle(img, (x,y),(x\_plus\_w, y\_plus\_h), label, color,2)

cv2.putText(img, label, (x-10,y-10),cv2.FONT\_HERSHEY\_COMPLEX, color, 2)

# read input image

image = cv2.imread(args.image)

Width = image.shape[1]

Height = image.shape[0]

scale = 0.00392

classes = None

# read class names from text file

classes = None

with open(args.classes, 'r') as f:

classes = [line.strip() for line in f.readlines()]

# generate different colors for different classes

COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

# read pre-trained model and config file

net = cv2.dnn.readNet(args.weights, args.config)

# create input blob

blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)

# set input blob for the network

net.setInput(blob)

# function to get the output layer names

# in the architecture

# function to draw bounding box on the detected object with class name

def draw\_bounding\_box(img, class\_id, confidence, x, y, x\_plus\_w, y\_plus\_h):

label = str(classes[class\_id])

color = COLORS[class\_id]

cv2.rectangle(img, (x,y), (x\_plus\_w,y\_plus\_h), color, 2)

cv2.putText(img, label, (x-10,y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

# run inference through the network

# and gather predictions from output layers

outs = net.forward(get\_output\_layers(net))

# initialization

class\_ids = []

confidences = []

boxes = []

conf\_threshold = 0.5

nms\_threshold = 0.4

# for each detetion from each output layer

# get the confidence, class id, bounding box params

# and ignore weak detections (confidence < 0.5)

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x = int(detection[0] \* Width)

center\_y = int(detection[1] \* Height)

w = int(detection[2] \* Width)

h = int(detection[3] \* Height)

x = center\_x - w / 2

y = center\_y - h / 2

class\_ids.append(class\_id)

confidences.append(float(confidence))

boxes.append([x, y, w, h])

# apply non-max suppression

indices = cv2.dnn.NMSBoxes(boxes, confidences, conf\_threshold, nms\_threshold)

# go through the detections remaining

# after nms and draw bounding box

for i in indices:

try:

box=boxes[i]

except:

i=i[0]

box=boxes[i]

x = box[0]

y = box[1]

w = box[2]

h = box[3]

draw\_bounding\_box(image, class\_ids[i], confidences[i], round(x), round(y), round(x+w), round(y+h))

# display output image

#cv2.imshow("object detection", image)

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off') # turn off axis labels

plt.show()

# wait until any key is pressed

#cv2.waitKey()

# save output image to disk

#cv2.imwrite("object-detection.jpg", image)

# release resources

#cv2.destroyAllWindows()

**Practical 4**

**Aim:- Image Stitching**

**Code:-**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load images

img\_ = cv2.imread('right.jpg')

img1 = cv2.cvtColor(img\_, cv2.COLOR\_BGR2GRAY)

img = cv2.imread('left.jpg')

img2 = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

#Initialize SIFT detector

sift = cv2.SIFT\_create()

kp1, des1 = sift.detectAndCompute(img1,None)

kp2, des2 = sift.detectAndCompute(img2,None)

# Create a BFMatcher object with distance measurement cv2.NORM\_L2

bf = cv2.BFMatcher(cv2.NORM\_L2, crossCheck = False)

# Perfomr the matching between the SIFT descriptors of the images

matches = bf.knnMatch(des1,des2,k=2)

# Apply the ratio test to find good matches

good = []

for m,n in matches:

if m.distance < 0.75\*n.distance:

good.append(m)

# Atleast 4 matches are to be there to find the homography

if len(good)>4:

# prepare source and destination points

src\_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1,1,2)

dst\_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1,1,2)

# Compute Homography

H, status = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC, 5.0)

# Use homography to warp image

dst = cv2.warpPerspective(img\_,H, (img.shape[1]+img\_.shape[1],img.shape[0]))

# Convert warped image from BGR to RGB for matplotlib

dst\_rgb = cv2.cvtColor(dst, cv2.COLOR\_BGR2RGB)

# Display the warped image

plt.subplot(122), plt.imshow(dst\_rgb), plt.title('Warped Image')

plt.show()

# Place the left image on the appropriate position

dst[0:img.shape[0], 0:img.shape[1]] = img

# Convert the combined image from BGR to RGB for matplotlib

combined\_rgb = cv2.cvtColor(dst, cv2.COLOR\_BGR2RGB)

# Save the stitched image as output.jpg in the BGR format

cv2.imwrite('output.jpg',dst)

# Display the stitched image

plt.imshow(combined\_rgb)

plt.title('Stitched Image')

plt.show()

else:

raise AssertionError('Not enough matches are found - {}/{}'.format(len(good),4))

**IP Practicals**

**Practical 2A**

i. Program to perform Image negation.

Code:

clc;

clear all;

A = imread('D:\MSC IT\Part I\Sem II\Image

Processing\IP\_Practical\IP\_Practical\_Images\negimg.jpg');

subplot(2,1,1);

imshow(A);

title('Orignial Image');

R = A(:,:,1);

G = A(:,:,2);

B = A(:,:,3);

[row col]=size(A);

for x=1:row

for y=1:col

R(x,y)=255-R(x,y);

G(x,y)=255-G(x,y);

B(x,y)=255-B(x,y);

end

end

A(:,:,1)=R;

A(:,:,2)=G;

A(:,:,3)=B;

subplot(2,1,2);

imshow(A);

title('Image after negation');

ii. Program to perform threshold on an image.

Code:

clc;

a=imread('D:\MSC IT\Part I\Sem II\Image Processing\IP\_Practical\Honey.jpg');

b=double(a);

[m,n]=size(b);

T=100;

for i=1:m

for j=1:n

if(b(i,j)<T)

c(i,j)=0;

else

c(i,j)=255;

end

end

end

imshow(uint8(c));

iii. Program to perform Log transformation.

Code:

//LOG

a=imread('D:\MSC IT\Part I\Sem II\Image Processing\IP\_Practical\Honey.jpg');

b=rgb2gray(a);

//Log operator

c=edge(b,'log');

imshow(c)

iv. Power-law transformations

Code:

//Power Law transformation

clear all;

clc;

close all;

i=imread('D:\MSC IT\Part I\Sem II\Image

Processing\IP\_Practical\IP\_Practical\_Images\flower.jpg');

subplot(2,1,1);

imshow(i);

title('Original Image');

i=im2double(i);

c=1;

[row col]=size(i);

for x=1:row

for y=1:col

i(x,y)=c\*i(x,y)^0.5; //1.5

end

end

i=im2uint8(i);

subplot(2,1,2);

imshow(i);

title('Image after power-law transformation');

v. Piecewise linear transformations

a. Contrast Stretching

Code:

clc

a = imread ("D:\MSC IT\Part I\Sem II\Image

Processing\IP\_Practical\IP\_Practical\_Images\lena.png");

a = rgb2gray ( a ) ;

b = double ( a ) \*0.5;

b = uint8 ( b );

c = double ( b ) \*2;

c = uint8 ( c );

subplot(1,3,1)

imshow(a) ;

title ( "Original Image " )

subplot(1,3,2)

imshow(b) ;

title ( "Decrease in Contrast" )

subplot(1,3,3)

imshow(c) ;

title ( "Increase in Contrast")

b. Gray-level slicing with and without background

Code with background:

clc;

clear all;

a=imread('D:\MSC IT\Part I\Sem II\Image

Processing\IP\_Practical\IP\_Practical\_Images\lena.png');

a1=58; // This value is user defined

b1=158; // This value is user defined

[r,c]=size(a);

figure(2);

subplot(2,1,1);

imshow(a);

for i=1:r

for j=1:c

if (a(i,j)>a1 & a(i,j)<b1)

x(i,j)=255;

else

x(i,j)=a(i,j);

end

end

end

x=uint8(x);

subplot(2,1,2);

title('Gray level slicing with background')

imshow(x);

Code without background:

clc;

clear all;

a=imread('D:\MSC IT\Part I\Sem II\Image

Processing\IP\_Practical\IP\_Practical\_Images\lena.png');

a1=50; // This value is user defined

b1=150; // This value is user defined

[r,c]=size(a);

figure(1)

subplot(2,1,1);

imshow(a);

for i=1:r

for j=1:c

if (a(i,j)>a1 & a(i,j)<b1)

x(i,j)=255;

else

x(i,j)=0;

end

end

end

x=uint8(x);

subplot(2,1,2);

title('Gray level slicing without background');

imshow(x);

c. Bit-plane slicing

Code:

clc;

clear all;

f=imread('D:\MSC IT\Part I\Sem II\Image

Processing\IP\_Practical\IP\_Practical\_Images\lenag.jpeg');

f=double(f);

[r,c]=size(f);

com=[128 64 32 16 8 4 2 1];

for k=1:1:length(com);

for i=1:r

for j=1:c

new(i,j)=bitand(f(i,j),com(k));

end

subplot(2,4,k);

imshow(new);

end

end