out=cv.cvtColor(img,cv.COLOR\_BGR2GRAY)

import cv2 as cv  
img=cv.imread("download.jpeg")  
#output=cv.imshow("final\_1",img)  
out=cv.cvtColor(img,cv.COLOR\_BGR2GRAY)  
#output\_1=cv.imshow("final",out)  
print(img.shape)  
do\_heg=90  
do\_wid=100  
do\_pts=(do\_heg,do\_wid)  
resize\_do=cv.resize(img,do\_pts)  
up\_heg=200  
up\_wid=300  
up\_pts=(up\_heg,up\_wid)  
resize\_up=cv.resize(img,up\_pts)  
#output\_4=cv.imshow("final\_2",resize\_do)  
#output\_5=cv.imshow("final\_3",resize\_up)  
(b\_channel,g\_channel,r\_channel)=cv.split(img)  
#cv.imshow("final\_5",b\_channel)  
#cv.imshow("final\_6",g\_channel)  
#cv.imshow("final\_7",r\_channel)  
merged\_image=cv.merge((b\_channel,g\_channel,r\_channel))  
cv.imshow("final\_8",merged\_image)  
cv.waitKey(0)

import cv2 as cv  
cap=cv.VideoCapture(0)  
if (cap.isOpened()):  
 print("error")  
while(cap.isOpened()):  
 ret,frame=cap.read()  
 if ret==True:  
 cv.imshow("frame",frame)  
 if cv.waitKey(25) & 0xFF==ord('q'):  
 break  
 else:  
 break

import cv2 as cv  
cap=cv.VideoCapture(0)  
if (cap.isOpened()):  
 print("error")  
while(cap.isOpened()):  
 ret,frame=cap.read()  
 if ret==True:  
 cv.imshow("frame",frame)  
 gray=cv.cvtColor(frame,cv.COLOR\_BGR2GRAY)  
 cv.imshow("frame\_2",gray)  
  
 if cv.waitKey(25) & 0xFF==ord('q'):  
 break  
 else:  
 break

import cv2 as cv  
import numpy as np  
img=cv.imread("download.jpeg")  
print(img.dtype)  
print(img)  
img\_negative=255-img  
print(img\_negative)  
cv.imshow("negative",img\_negative)  
cv.imshow("positive",img)  
cv.waitKey(0)

import cv2 as cv  
import numpy as np  
img=cv.imread("download.jpeg")  
print(img.dtype)  
print(img)  
img\_negative=255-img  
#print(img\_negative)  
#cv.imshow("negative",img\_negative)  
#cv.imshow("positive",img)  
c=255/(np.log(1+np.max(img)))  
log\_2=c\*np.log(1+img)  
log\_3=np.array(log\_2,dtype=np.uint8)  
print(log\_3)  
cv.imshow("logi.jpeg",log\_3)  
cv.waitKey(0)

import cv2 as cv  
import numpy as np  
img=cv.imread("download.jpeg")  
print(img.dtype)  
print(img)  
img\_negative=255-img  
#print(img\_negative)  
#cv.imshow("negative",img\_negative)  
#cv.imshow("positive",img)  
c=255/(np.log(1+np.max(img)))  
log\_2=c\*np.log(1+img)  
log\_3=np.array(log\_2,dtype=np.uint8)  
#print(log\_3)  
#cv.imshow("logi.jpeg",log\_3)  
for gamma in [0.1,0.5,1.0,1,5]:  
 gamma\_1=255\*(img/255)\*\*gamma  
 gamma\_final=np.array(gamma\_1,dtype="uint8")  
 print(gamma\_final)  
 cv.imshow("gamma",gamma\_final)  
 cv.waitKey(0)

import cv2 as cv  
import numpy as np  
def pixelVal(pix,r1,s1,r2,s2):  
 if 0<=pix and pix<r1:  
 return (s1/r1)\*pix  
 elif r1<=pix and pix<r2:  
 return ((s2-s1)/(r2-r1))\*(pix-r1)+s1  
 else:  
 return ((255-s2)/(255-r2))\*(pix-r2)+s2  
img=cv.imread("download.jpeg")  
print(img.dtype)  
print(img)  
img\_negative=255-img  
#print(img\_negative)  
#cv.imshow("negative",img\_negative)  
#cv.imshow("positive",img)  
c=255/(np.log(1+np.max(img)))  
log\_2=c\*np.log(1+img)  
log\_3=np.array(log\_2,dtype=np.uint8)  
#print(log\_3)  
#cv.imshow("logi.jpeg",log\_3)  
#for gamma in [0.1,0.5,1.0,1,5]:  
 #gamma\_1=255\*(img/255)\*\*gamma  
 #gamma\_final=np.array(gamma\_1,dtype="uint8")  
 #print(gamma\_final)  
 #cv.imshow("gamma",gamma\_final)  
r1=100  
r2=150  
s1=200  
s2=250  
pixel\_val=np.vectorize(pixelVal)  
contrast=pixel\_val(img,r1,s1,r2,s2)  
contrast\_final=cv.imshow("contrast",contrast)  
cv.waitKey(0)

import numpy as np  
import cv2  
  
img = cv2.imread("download.jpeg",0)  
lst = []  
for i in range(img.shape[0]):  
 for j in range(img.shape[1]):  
 lst.append(np.binary\_repr(img[i][j],width=8))  
eight\_bit\_img = (np.array([int(i[0]) for i in lst],dtype = np.uint8) \*128).reshape(img.shape[0],img.shape[1])  
seven\_bit\_img = (np.array([int(i[1]) for i in lst],dtype = np.uint8) \*64).reshape(img.shape[0],img.shape[1])  
six\_bit\_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8) \*32).reshape(img.shape[0],img.shape[1])  
five\_bit\_img = (np.array([int(i[3]) for i in lst],dtype = np.uint8) \*16).reshape(img.shape[0],img.shape[1])  
four\_bit\_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8) \*8).reshape(img.shape[0],img.shape[1])  
three\_bit\_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8) \*4).reshape(img.shape[0],img.shape[1])  
two\_bit\_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) \*2).reshape(img.shape[0],img.shape[1])  
one\_bit\_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) \*1).reshape(img.shape[0],img.shape[1])  
  
finalr = cv2.hconcat([eight\_bit\_img,seven\_bit\_img,six\_bit\_img,five\_bit\_img])  
finalv = cv2.hconcat([four\_bit\_img,three\_bit\_img,two\_bit\_img,one\_bit\_img])  
final = cv2.vconcat([finalr,finalv])  
  
cv2.imshow("last",final)  
cv2.waitKey(0)

VIDEOWRITE import cv2  
  
  
def main():  
 # reading the input  
 cap = cv2.VideoCapture(r"C:\Users\ranji\PycharmProjects\pythonProject2\closeup\_of\_wild\_butterfly\_in\_nature\_6891908.mp4")  
  
 output = cv2.VideoWriter(  
 "output.avi", cv2.VideoWriter\_fourcc(\*'MPEG'), 30, (1080, 1920))  
  
 while (True):  
 ret, frame = cap.read()  
 if (ret):  
  
 # adding rectangle on each frame  
 cv2.rectangle(frame, (100, 100), (500, 500), (0, 255, 0), 3)  
  
 # writing the new frame in output  
 output.write(frame)  
 cv2.imshow("output", frame)  
 if cv2.waitKey(1) & 0xFF == ord('s'):  
 break  
  
 cv2.destroyAllWindows()  
 output.release()  
 cap.release()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

GRAYSCALE LiveVIDEOWRITE

import cv2  
  
  
def main():  
 # reading the input  
 cap = cv2.VideoCapture(0)  
  
 output = cv2.VideoWriter(  
 "output.avi", cv2.VideoWriter\_fourcc(\*'MPEG'), 30, (1080, 1920))  
  
 while (True):  
 ret, frame = cap.read()  
 if (ret):  
  
  
  
 frames=cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)  
 # writing the new frame in output  
 output.write(frames)  
 cv2.imshow("output",frames)  
 if cv2.waitKey(1) & 0xFF == ord('s'):  
 break  
  
 cv2.destroyAllWindows()  
 output.release()  
 cap.release()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Masking

import cv2  
import numpy as np  
  
# Specify the file path of the original image  
image\_path = 'img.png' # Replace with the actual path of your image file  
  
# Read the original image using OpenCV  
original\_image = cv2.imread(image\_path)  
  
# Check if the image was successfully loaded  
if original\_image is not None:  
 # Create a binary mask (white rectangle on a black background)  
 mask = np.zeros\_like(original\_image, dtype=np.uint8)  
 roi = (50, 50, 200, 200) # Define the region of interest (x, y, width, height)  
 cv2.rectangle(mask, (roi[0], roi[1]), (roi[0] + roi[2], roi[1] + roi[3]), (255, 255, 255), thickness=cv2.FILLED)  
  
 # Display the original image and mask  
 cv2.imshow('Original Image', original\_image)  
 cv2.imshow('Mask', mask)  
  
 # Apply masking  
 masked\_image = cv2.bitwise\_and(original\_image, mask)  
  
 # Display the masked image  
 cv2.imshow('Masked Image', masked\_image)  
  
 # Wait for a key event and then close the windows  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()  
  
else:  
 print(f"Error: Unable to load the image from {image\_path}")

masking the image

import cv2  
import numpy as np  
# read input image  
img = cv2.imread(r'C:\Users\ranji\PycharmProjects\pythonProject2\download.jpeg')  
# Convert BGR to HSV  
hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)  
# define range of blue color in HSV  
lower\_yellow = np.array([15,50,180])  
upper\_yellow = np.array([40,255,255])  
# Create a mask. Threshold the HSV image to get only yellow colors  
mask = cv2.inRange(hsv, lower\_yellow, upper\_yellow)  
# Bitwise-AND mask and original image  
result = cv2.bitwise\_and(img,img, mask= mask)  
# display the mask and masked image  
cv2.imshow('Mask',mask)  
cv2.waitKey(0)  
cv2.imshow('Masked Image',result)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

masking the video

import cv2  
import numpy as np  
# read input image  
vid = cv2.VideoCapture(r'C:\Users\ranji\PycharmProjects\pythonProject2\closeup\_of\_wild\_butterfly\_in\_nature\_6891908.mp4')  
while (True):  
 # Capture the video frame  
 # by frame  
 ret, img = vid.read()  
 # Display the resulting frame  
 cv2.imshow('frame', img)  
 hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)  
 # define range of blue color in HSV  
 lower\_yellow = np.array([15, 50, 180])  
 upper\_yellow = np.array([40, 255, 255])  
 # Create a mask. Threshold the HSV image to get only yellow colors  
 mask = cv2.inRange(hsv, lower\_yellow, upper\_yellow)  
 # Bitwise-AND mask and original image  
 result = cv2.bitwise\_and(img, img, mask=mask)  
 # the 'q' button is set as the  
 # quitting button you may use any  
 cv2.imshow('Mask', mask)  
 # desired button of your choice  
 cv2.imshow('Masked Image', result)  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
# Convert BGR to HSV  
# display the mask and masked image  
cv2.waitKey(0)  
cv2.destroyAllWindows()

masking the live video

import cv2  
import numpy as np  
# read input image  
vid = cv2.VideoCapture(0)  
while (True):  
 # Capture the video frame  
 # by frame  
 ret, img = vid.read()  
 # Display the resulting frame  
 cv2.imshow('frame', img)  
 hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)  
 # define range of blue color in HSV  
 lower\_yellow = np.array([15, 50, 180])  
 upper\_yellow = np.array([40, 255, 255])  
 # Create a mask. Threshold the HSV image to get only yellow colors  
 mask = cv2.inRange(hsv, lower\_yellow, upper\_yellow)  
 # Bitwise-AND mask and original image  
 result = cv2.bitwise\_and(img, img, mask=mask)  
 # the 'q' button is set as the  
 # quitting button you may use any  
 cv2.imshow('Mask', mask)  
 # desired button of your choice  
 cv2.imshow('Masked Image', result)  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
# Convert BGR to HSV  
# display the mask and masked image  
cv2.waitKey(0)  
cv2.destroyAllWindows()

blurring

# Importing cv2 module  
import cv2  
img = cv2.imread('download.jpeg')  
# make sure that you have saved it in the same folder  
# You can change the kernel size as you want  
blurImg = cv2.blur(img,(10,10))  
cv2.imshow('blurred image',blurImg)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

Paint and draw on the image

import numpy as np  
import cv2  
img = np.zeros((512, 512, 3), dtype='uint8')  
cv2.line(img,(20,400),(400,20),(255,255,255),3)  
cv2.rectangle(img,(200,100), (400,400), (0,255,0),-1)  
cv2.circle(img,(80,80), 55, (255,255,0), -1)  
cv2.ellipse(img, (300,425), (80, 20), 5, 0, 360, (0,0,255), -1)  
cv2.putText(img, "harini",(10,100), cv2.FONT\_HERSHEY\_SIMPLEX,  
2,(255,0,255),2,cv2.LINE\_AA)  
cv2.imshow('image',img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

for white background

import numpy as np  
import cv2  
img = np.full((500, 500, 3), 255, dtype = np.uint8)  
cv2.line(img,(20,400),(400,20),(255,255,255),3)  
cv2.rectangle(img,(200,100), (400,400), (0,255,0),-1)  
cv2.circle(img,(80,80), 55, (255,255,0), -1)  
cv2.ellipse(img, (300,425), (80, 20), 5, 0, 360, (0,0,255), -1)  
cv2.putText(img, "Vijayalakshmi",(10,100), cv2.FONT\_HERSHEY\_SIMPLEX,  
2,(255,0,255),2,cv2.LINE\_AA)  
cv2.imshow('image',img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

writing on the image

import numpy as numpy  
import cv2  
img = cv2.imread('download.jpeg',0)  
txt="Lenna"  
font = cv2.FONT\_HERSHEY\_SIMPLEX  
cv2.putText(img,txt,(90,100), font, 1,(0,0,255),1,cv2.LINE\_AA)  
cv2.line(img,(20,400),(400,20),(255,255,255),3)  
cv2.rectangle(img,(200,100),(80,80),(0,255,0),2)  
cv2.circle(img,(50,50), 25, (25,0,255), 1)  
cv2.ellipse(img, (30,40), (20,10), 3, 0, 360, (0,0,255), -1)  
cv2.imshow('image',img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

histogram

import numpy as numpy  
import cv2 as cv  
import matplotlib.pyplot as plt  
img = cv.imread('download.jpeg',0)  
histr=cv.calcHist([img],[0],None,[256],[0,256])  
plt.plot(histr)  
plt.show()

exp 4

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
# Specify the file path of the original image  
image\_path = 'download.jpeg' # Replace with the actual path of your image file  
  
# Read the original image using OpenCV  
original\_image = cv2.imread(image\_path)  
  
# Check if the image was successfully loaded  
if original\_image is not None:  
 # Define translation parameters for affine transformation  
 translation\_matrix\_affine = np.float32([[1, 0, 50], [0, 1, 30]])  
  
 # Define rotation parameters for affine transformation  
 angle\_affine = 45  
 rotation\_center\_affine = (original\_image.shape[1] // 2, original\_image.shape[0] // 2)  
 rotation\_matrix\_affine = cv2.getRotationMatrix2D(rotation\_center\_affine, angle\_affine, 1)  
  
 # Apply affine transformations (translation and rotation)  
 translated\_affine = cv2.warpAffine(original\_image, translation\_matrix\_affine, (original\_image.shape[1], original\_image.shape[0]))  
 rotated\_affine = cv2.warpAffine(translated\_affine, rotation\_matrix\_affine, (original\_image.shape[1], original\_image.shape[0]))  
  
 # Define translation parameters for perspective transformation  
 translation\_matrix\_perspective = np.float32([[1, 0, 50], [0, 1, 30], [0, 0, 1]])  
  
 # Define rotation parameters for perspective transformation  
 angle\_perspective = 45  
 rotation\_matrix\_perspective = cv2.getRotationMatrix2D(rotation\_center\_affine, angle\_perspective, 1)  
  
 # Apply perspective transformations (translation and rotation)  
 translated\_perspective = cv2.warpPerspective(original\_image, translation\_matrix\_perspective, (original\_image.shape[1], original\_image.shape[0]))  
 rotated\_perspective = cv2.warpAffine(translated\_perspective, rotation\_matrix\_perspective, (original\_image.shape[1], original\_image.shape[0]))  
  
 # Display the images using Matplotlib  
 plt.subplot(231), plt.imshow(cv2.cvtColor(original\_image, cv2.COLOR\_BGR2RGB)), plt.title('Original Image')  
 plt.subplot(232), plt.imshow(cv2.cvtColor(translated\_affine, cv2.COLOR\_BGR2RGB)), plt.title('Affine Translation')  
 plt.subplot(233), plt.imshow(cv2.cvtColor(rotated\_affine, cv2.COLOR\_BGR2RGB)), plt.title('Affine Rotation')  
 plt.subplot(234), plt.imshow(cv2.cvtColor(translated\_perspective, cv2.COLOR\_BGR2RGB)), plt.title('Perspective Translation')  
 plt.subplot(235), plt.imshow(cv2.cvtColor(rotated\_perspective, cv2.COLOR\_BGR2RGB)), plt.title('Perspective Rotation')  
  
 plt.show()  
  
else:  
 print(f"Error: Unable to load the image from {image\_path}")

thresholding

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
# Specify the file path of the original image  
image\_path = 'img.png' # Replace with the actual path of your image file  
  
# Read the original image using OpenCV  
original\_image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  
  
# Check if the image was successfully loaded  
if original\_image is not None:  
 # Grayscale conversion  
 gray\_image = original\_image  
  
 # Simple binary thresholding  
 \_, binary\_simple = cv2.threshold(original\_image, 127, 255, cv2.THRESH\_BINARY)  
  
 # Inverse binary thresholding  
 \_, binary\_inverse = cv2.threshold(original\_image, 127, 255, cv2.THRESH\_BINARY\_INV)  
  
 # Truncation thresholding  
 \_, trunc\_image = cv2.threshold(original\_image, 127, 255, cv2.THRESH\_TRUNC)  
  
 # Otsu's thresholding  
 \_, otsu\_image = cv2.threshold(original\_image, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  
  
 # Adaptive thresholding  
 adaptive\_image = cv2.adaptiveThreshold(original\_image, 255, cv2.ADAPTIVE\_THRESH\_MEAN\_C, cv2.THRESH\_BINARY, 11, 2)  
  
 # Display the images using Matplotlib  
 plt.subplot(231), plt.imshow(gray\_image, cmap='gray'), plt.title('Original Image (Grayscale)')  
 plt.subplot(232), plt.imshow(binary\_simple, cmap='gray'), plt.title('Simple Binary Thresholding')  
 plt.subplot(233), plt.imshow(binary\_inverse, cmap='gray'), plt.title('Inverse Binary Thresholding')  
 plt.subplot(234), plt.imshow(trunc\_image, cmap='gray'), plt.title('Truncation Thresholding')  
 plt.subplot(235), plt.imshow(otsu\_image, cmap='gray'), plt.title("Otsu's Thresholding")  
 plt.subplot(236), plt.imshow(adaptive\_image, cmap='gray'), plt.title('Adaptive Thresholding')  
  
 plt.tight\_layout()  
 plt.show()  
  
else:  
 print(f"Error: Unable to load the image from {image\_path}")

otsu thresholding

import cv2 as cv  
import numpy as np  
from matplotlib import pyplot as plt  
img = cv.imread('img.png', cv.IMREAD\_GRAYSCALE)  
ret1,th1 = cv.threshold(img,127,255,cv.THRESH\_BINARY)  
# Otsu's thresholding  
ret2,th2 = cv.threshold(img,0,255,cv.THRESH\_BINARY+cv.THRESH\_OTSU)  
# Otsu's thresholding after Gaussian filtering  
blur = cv.GaussianBlur(img,(5,5),0)  
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH\_BINARY+cv.THRESH\_OTSU)  
# plot all the images and their histograms  
images = [img, 0, th1,  
img, 0, th2,  
blur, 0, th3]  
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',  
'Original Noisy Image','Histogram',"Otsu's Thresholding",  
'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]  
for i in range(3):  
 plt.subplot(3,3,i\*3+1),plt.imshow(images[i\*3],'gray')  
 plt.title(titles[i\*3]), plt.xticks([]), plt.yticks([])  
 plt.subplot(3,3,i\*3+2),plt.hist(images[i\*3].ravel(),256)  
 plt.title(titles[i\*3+1]), plt.xticks([]), plt.yticks([])  
 plt.subplot(3,3,i\*3+3),plt.imshow(images[i\*3+2],'gray')  
 plt.title(titles[i\*3+2]), plt.xticks([]), plt.yticks([])  
 plt.show()

import cv2 as cv  
import numpy as np  
from matplotlib import pyplot as plt  
img = np.array([[100, 95, 57],  
 [150, 75, 37],  
 [157, 75, 153]],dtype=np.uint8)  
print('Original ',img)  
THRESHOLD = np.empty((3,3),dtype=np.uint8)  
for i in range(0,3):  
 for j in range(0,3):  
 if img[i][j]<=127:  
 THRESHOLD[i][j] = 0  
 else:  
 THRESHOLD[i][j]=255  
print('threshold matrix ',THRESHOLD)  
plt.hist(img)  
plt.show()  
plt.hist(THRESHOLD)  
plt.show()

edge detection

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
# Specify the file path of the original image  
image\_path = 'img.png' # Replace with the actual path of your image file  
  
# Read the original image using OpenCV  
original\_image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  
  
# Check if the image was successfully loaded  
if original\_image is not None:  
 # Sobel edge detection  
 sobel\_x = cv2.Sobel(original\_image, cv2.CV\_64F, 1, 0, ksize=3)  
 sobel\_y = cv2.Sobel(original\_image, cv2.CV\_64F, 0, 1, ksize=3)  
 sobel\_edges = np.sqrt(sobel\_x\*\*2 + sobel\_y\*\*2)  
  
 # Canny edge detection  
 canny\_edges = cv2.Canny(original\_image, 50, 150)  
  
 # Laplacian edge detection  
 laplacian\_edges = cv2.Laplacian(original\_image, cv2.CV\_64F)  
  
 # Prewitt edge detection  
 kernel\_prewitt\_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])  
 kernel\_prewitt\_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])  
 prewitt\_x = cv2.filter2D(original\_image, cv2.CV\_64F, kernel\_prewitt\_x)  
 prewitt\_y = cv2.filter2D(original\_image, cv2.CV\_64F, kernel\_prewitt\_y)  
 prewitt\_edges = np.sqrt(prewitt\_x\*\*2 + prewitt\_y\*\*2)  
  
 # Display the images using Matplotlib  
 plt.figure(figsize=(12, 8))  
  
 plt.subplot(231), plt.imshow(original\_image, cmap='gray'), plt.title('Original Image')  
 plt.subplot(232), plt.imshow(sobel\_edges, cmap='gray'), plt.title('Sobel Edge Detection')  
 plt.subplot(233), plt.imshow(canny\_edges, cmap='gray'), plt.title('Canny Edge Detection')  
 plt.subplot(234), plt.imshow(laplacian\_edges, cmap='gray'), plt.title('Laplacian Edge Detection')  
 plt.subplot(235), plt.imshow(prewitt\_edges, cmap='gray'), plt.title('Prewitt Edge Detection')  
  
 plt.tight\_layout()  
 plt.show()  
  
else:  
 print(f"Error: Unable to load the image from {image\_path}")

edge detection in different windows

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
# Specify the file path of the original image  
image\_path = 'img.png' # Replace with the actual path of your image file  
  
# Read the original image using OpenCV  
original\_image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)  
  
# Check if the image was successfully loaded  
if original\_image is not None:  
 # Sobel edge detection  
 sobel\_x = cv2.Sobel(original\_image, cv2.CV\_64F, 1, 0, ksize=3)  
 sobel\_y = cv2.Sobel(original\_image, cv2.CV\_64F, 0, 1, ksize=3)  
 sobel\_edges = np.sqrt(sobel\_x\*\*2 + sobel\_y\*\*2)  
 plt.imshow(sobel\_edges, cmap='gray')  
 plt.title('Sobel Edge Detection')  
 plt.show()  
  
 # Canny edge detection  
 canny\_edges = cv2.Canny(original\_image, 50, 150)  
 plt.imshow(canny\_edges, cmap='gray')  
 plt.title('Canny Edge Detection')  
 plt.show()  
  
 # Laplacian edge detection  
 laplacian\_edges = cv2.Laplacian(original\_image, cv2.CV\_64F)  
 plt.imshow(laplacian\_edges, cmap='gray')  
 plt.title('Laplacian Edge Detection')  
 plt.show()  
  
 # Prewitt edge detection  
 kernel\_prewitt\_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])  
 kernel\_prewitt\_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])  
 prewitt\_x = cv2.filter2D(original\_image, cv2.CV\_64F, kernel\_prewitt\_x)  
 prewitt\_y = cv2.filter2D(original\_image, cv2.CV\_64F, kernel\_prewitt\_y)  
 prewitt\_edges = np.sqrt(prewitt\_x\*\*2 + prewitt\_y\*\*2)  
 plt.imshow(prewitt\_edges, cmap='gray')  
 plt.title('Prewitt Edge Detection')  
 plt.show()  
  
 # Display the original image using Matplotlib  
 plt.imshow(original\_image, cmap='gray')  
 plt.title('Original Image')  
 plt.show()  
  
else:  
 print(f"Error: Unable to load the image from {image\_path}")

edge detection using video live camera

import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
  
# Function to perform Prewitt edge detection  
def prewitt\_edge\_detection(image):  
 kernel\_prewitt\_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])  
 kernel\_prewitt\_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])  
 prewitt\_x = cv2.filter2D(image, cv2.CV\_64F, kernel\_prewitt\_x)  
 prewitt\_y = cv2.filter2D(image, cv2.CV\_64F, kernel\_prewitt\_y)  
 prewitt\_edges = np.sqrt(prewitt\_x\*\*2 + prewitt\_y\*\*2)  
 return prewitt\_edges  
  
# Open a connection to the camera (0 is usually the built-in camera)  
cap = cv2.VideoCapture(0)  
  
while True:  
 # Read a frame from the camera  
 ret, frame = cap.read()  
  
 # Check if the frame was successfully captured  
 if not ret:  
 print("Error: Couldn't capture frame")  
 break  
  
 # Convert the frame to grayscale  
 gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # Sobel edge detection  
 sobel\_x = cv2.Sobel(gray\_frame, cv2.CV\_64F, 1, 0, ksize=3)  
 sobel\_y = cv2.Sobel(gray\_frame, cv2.CV\_64F, 0, 1, ksize=3)  
 sobel\_edges = np.sqrt(sobel\_x\*\*2 + sobel\_y\*\*2)  
  
 # Canny edge detection  
 canny\_edges = cv2.Canny(gray\_frame, 50, 150)  
  
 # Laplacian edge detection  
 laplacian\_edges = cv2.Laplacian(gray\_frame, cv2.CV\_64F)  
  
 # Prewitt edge detection  
 prewitt\_edges = prewitt\_edge\_detection(gray\_frame)  
  
 # Display the frames in separate windows  
 cv2.imshow('Original Frame', frame)  
 cv2.imshow('Sobel Edges', np.uint8(sobel\_edges))  
 cv2.imshow('Canny Edges', canny\_edges)  
 cv2.imshow('Laplacian Edges', np.uint8(laplacian\_edges))  
 cv2.imshow('Prewitt Edges', np.uint8(prewitt\_edges))  
  
 # Break the loop if 'q' key is pressed  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
# Release the camera and close all windows  
cap.release()  
cv2.destroyAllWindows()

edge dection for input video

import cv2  
import numpy as np  
  
# Function to perform Prewitt edge detection  
def prewitt\_edge\_detection(image):  
 kernel\_prewitt\_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])  
 kernel\_prewitt\_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])  
 prewitt\_x = cv2.filter2D(image, cv2.CV\_64F, kernel\_prewitt\_x)  
 prewitt\_y = cv2.filter2D(image, cv2.CV\_64F, kernel\_prewitt\_y)  
 prewitt\_edges = np.sqrt(prewitt\_x\*\*2 + prewitt\_y\*\*2)  
 return np.uint8(prewitt\_edges)  
  
# Open a connection to the video file (replace 'input\_video.mp4' with the path to your video file)  
cap = cv2.VideoCapture(r'C:\Users\ranji\PycharmProjects\pythonProject2\closeup\_of\_wild\_butterfly\_in\_nature\_6891908.mp4')  
  
while True:  
 # Read a frame from the video  
 ret, frame = cap.read()  
  
 # Break the loop if no more frames  
 if not ret:  
 break  
  
 # Convert the frame to grayscale  
 gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # Sobel edge detection  
 sobel\_x = cv2.Sobel(gray\_frame, cv2.CV\_64F, 1, 0, ksize=3)  
 sobel\_y = cv2.Sobel(gray\_frame, cv2.CV\_64F, 0, 1, ksize=3)  
 sobel\_edges = np.sqrt(sobel\_x\*\*2 + sobel\_y\*\*2)  
 cv2.imshow('Sobel Edges', np.uint8(sobel\_edges))  
  
 # Canny edge detection  
 canny\_edges = cv2.Canny(gray\_frame, 50, 150)  
 cv2.imshow('Canny Edges', canny\_edges)  
  
 # Laplacian edge detection  
 laplacian\_edges = cv2.Laplacian(gray\_frame, cv2.CV\_64F)  
 cv2.imshow('Laplacian Edges', np.uint8(laplacian\_edges))  
  
 # Prewitt edge detection  
 prewitt\_edges = prewitt\_edge\_detection(gray\_frame)  
 cv2.imshow('Prewitt Edges', prewitt\_edges)  
  
 # Display the original frame  
 cv2.imshow('Original Frame', frame)  
  
 # Break the loop if 'q' key is pressed  
 if cv2.waitKey(25) & 0xFF == ord('q'):  
 break  
  
# Release the video file and close all windows  
cap.release()  
cv2.destroyAllWindows()

single template matching

import cv2  
import numpy as np  
  
# Load the main image  
main\_image = cv2.imread('img\_3.png')  
main\_gray = cv2.cvtColor(main\_image, cv2.COLOR\_BGR2GRAY)  
  
# Load the template image  
template = cv2.imread('1234.jpg', 0)  
  
# Get the dimensions of the template  
template\_height, template\_width = template.shape  
  
# Choose the matching method (you can try different methods)  
method = cv2.TM\_CCOEFF\_NORMED  
  
# Perform template matching  
result = cv2.matchTemplate(main\_gray, template, method)  
min\_val, max\_val, min\_loc, max\_loc = cv2.minMaxLoc(result)  
  
# Draw a rectangle around the matched region  
top\_left = max\_loc  
bottom\_right = (top\_left[0] + template\_width, top\_left[1] + template\_height)  
cv2.rectangle(main\_image, top\_left, bottom\_right, (0, 255, 0), 2)  
  
# Display the result  
cv2.imshow('Template Matching Result', main\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

for input video

import cv2  
import numpy as np  
  
# Load the template image  
template = cv2.imread('img\_4.png', 0)  
  
# Get the dimensions of the template  
template\_height, template\_width = template.shape  
  
# Open a connection to the video file (replace 'input\_video.mp4' with the path to your video file)  
cap = cv2.VideoCapture('closeup\_of\_wild\_butterfly\_in\_nature\_6891908.mp4')  
  
while True:  
 # Read a frame from the video  
 ret, frame = cap.read()  
  
 # Break the loop if no more frames  
 if not ret:  
 break  
  
 # Convert the frame to grayscale  
 frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # Perform template matching  
 result = cv2.matchTemplate(frame\_gray, template, cv2.TM\_CCOEFF\_NORMED)  
 \_, \_, \_, max\_loc = cv2.minMaxLoc(result)  
  
 # Draw a rectangle around the matched region  
 top\_left = max\_loc  
 bottom\_right = (top\_left[0] + template\_width, top\_left[1] + template\_height)  
 cv2.rectangle(frame, top\_left, bottom\_right, (0, 255, 0), 2)  
  
 # Display the result  
 cv2.imshow('Template Matching Result', frame)  
  
 # Break the loop if 'q' key is pressed  
 if cv2.waitKey(25) & 0xFF == ord('q'):  
 break  
  
# Release the video file and close all windows  
cap.release()  
cv2.destroyAllWindows()

for live camera

import cv2  
import numpy as np  
  
# Load the template image  
template = cv2.imread(r'C:\Users\ranji\PycharmProjects\pythonProject2\img\_5.png', 0)  
  
# Get the dimensions of the template  
template\_height, template\_width = template.shape  
  
# Open a connection to the camera (0 is usually the built-in camera)  
cap = cv2.VideoCapture(0)  
  
while True:  
 # Read a frame from the camera  
 ret, frame = cap.read()  
  
 # Break the loop if no more frames  
 if not ret:  
 break  
  
 # Convert the frame to grayscale  
 frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # Perform template matching  
 result = cv2.matchTemplate(frame\_gray, template, cv2.TM\_CCOEFF\_NORMED)  
 \_, \_, \_, max\_loc = cv2.minMaxLoc(result)  
  
 # Draw a rectangle around the matched region  
 top\_left = max\_loc  
 bottom\_right = (top\_left[0] + template\_width, top\_left[1] + template\_height)  
 cv2.rectangle(frame, top\_left, bottom\_right, (0, 255, 0), 2)  
  
 # Display the result  
 cv2.imshow('Template Matching Result', frame)  
  
 # Break the loop if 'q' key is pressed  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
# Release the camera and close all windows  
cap.release()  
cv2.destroyAllWindows()

single template for multiple matching

import cv2  
import numpy as np  
  
# Load the main image  
main\_image = cv2.imread('img\_13.png')  
main\_gray = cv2.cvtColor(main\_image, cv2.COLOR\_BGR2GRAY)  
  
# Load the template image  
template = cv2.imread('hp.jpg', 0)  
  
# Get the dimensions of the template  
template\_height, template\_width = template.shape  
  
# Perform template matching  
result = cv2.matchTemplate(main\_gray, template, cv2.TM\_CCOEFF\_NORMED)  
threshold = 0.8 # Adjust the threshold as needed  
  
# Find locations where the result is above the threshold  
locations = np.where(result >= threshold)  
  
# Draw rectangles around all occurrences  
for loc in zip(\*locations[::-1]):  
 top\_left = loc  
 bottom\_right = (top\_left[0] + template\_width, top\_left[1] + template\_height)  
 cv2.rectangle(main\_image, top\_left, bottom\_right, (0, 255, 0), 2)  
  
# Display the result  
cv2.imshow('Template Matching Result', main\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

for live camera video

import cv2  
import numpy as np  
  
# Load the template image  
template = cv2.imread('img\_14.png', 0)  
  
# Get the dimensions of the template  
template\_height, template\_width = template.shape  
  
# Open a connection to the camera (0 is usually the built-in camera)  
cap = cv2.VideoCapture(0)  
  
while True:  
 # Read a frame from the camera  
 ret, frame = cap.read()  
  
 # Break the loop if no more frames  
 if not ret:  
 break  
  
 # Convert the frame to grayscale  
 frame\_gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # Perform template matching  
 result = cv2.matchTemplate(frame\_gray, template, cv2.TM\_CCOEFF\_NORMED)  
 threshold = 0.8 # Adjust the threshold as needed  
  
 # Find locations where the result is above the threshold  
 locations = np.where(result >= threshold)  
  
 # Draw rectangles around all occurrences  
 for loc in zip(\*locations[::-1]):  
 top\_left = loc  
 bottom\_right = (top\_left[0] + template\_width, top\_left[1] + template\_height)  
 cv2.rectangle(frame, top\_left, bottom\_right, (0, 255, 0), 2)  
  
 # Display the result  
 cv2.imshow('Template Matching Result', frame)  
  
 # Break the loop if 'q' key is pressed  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
# Release the camera and close all windows  
cap.release()  
cv2.destroyAllWindows()

multiple template matching

import cv2  
import numpy as np  
  
# Load the main image  
main\_image = cv2.imread('img\_13.png')  
main\_gray = cv2.cvtColor(main\_image, cv2.COLOR\_BGR2GRAY)  
  
# Load multiple template images (replace with the paths to your template images)  
templates = [cv2.imread('hp.jpg', 0), cv2.imread('ui.jpg', 0)]  
  
# Iterate over each template  
for template in templates:  
 # Get the dimensions of the template  
 template\_height, template\_width = template.shape  
  
 # Perform template matching  
 result = cv2.matchTemplate(main\_gray, template, cv2.TM\_CCOEFF\_NORMED)  
 threshold = 0.8 # Adjust the threshold as needed  
  
 # Find locations where the result is above the threshold  
 locations = np.where(result >= threshold)  
  
 # Draw rectangles around all occurrences  
 for loc in zip(\*locations[::-1]):  
 top\_left = loc  
 bottom\_right = (top\_left[0] + template\_width, top\_left[1] + template\_height)  
 cv2.rectangle(main\_image, top\_left, bottom\_right, (0, 255, 0), 2)  
  
# Display the result  
cv2.imshow('Template Matching Result', main\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()