1.Blue , Green , Red , 1D , 2D convolution

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

image = cv2.imread('flower.png')
plt.subplot(241),plt.title('image'),plt.imshow(image),plt.axis('off')

gray=cv2.imread(r'C:\Users\Vishwas\Downloads\flower2.png',cv2.IMREAD_GRAYSCALE)
plt.subplot(242),plt.title('gray'),plt.imshow(gray,cmap='gray'),plt.axis('off')

b,g,r = cv2.split(image)

plt.subplot(243),plt.title('blue'),plt.imshow(b,cmap='gray'),plt.axis('off')

plt.subplot(244),plt.title('green'),plt.imshow(g,cmap='gray'),plt.axis('off')

plt.subplot(245),plt.title('red'),plt.imshow(r,cmap='gray'),plt.axis('off')

k1 = np.array([1,0,-1])
c1 = cv2.filter2D(image , -1 , k1)

plt.subplot(246),plt.title('conv1'),plt.imshow(c1,cmap='gray'),plt.axis('off')

k2 = np.array([[1,1,1],[1,-8,1],[1,1,1]])
c2 = cv2.filter2D(image , -1 , k2)

plt.subplot(247),plt.title('conv2'),plt.imshow(c2,cmap='gray'),plt.axis('off')

plt.show()
```

2.Arithmetic and logic operations

```
import cv2

image1=cv2.imread('flower2.png')
image2=cv2.imread('flower1.png')

if image1.shape != image2.shape:
    image2 = cv2.resize(image2 , (image1.shape[1],image1.shape[0]))

cv2.imshow('image1',image1)
cv2.imshow('image2',image2)

a = cv2.add(image1,image2)
s = cv2.subtract(image1,image2)

cv2.imshow('add',a)
cv2.imshow('sub',s)

m = cv2.multiply(image1,image2)
d = cv2.divide(image1,image2)

cv2.imshow('mul',m)
cv2.imshow('div',d)
```

```
not1 = cv2.bitwise_not(image1)
cv2.imshow('1not',not1)

not2 = cv2.bitwise_not(image2)
cv2.imshow('2not',not2)

or1 = cv2.bitwise_or(image1,image2)
cv2.imshow('or',or1)

and1 = cv2.bitwise_and(image1,image2)
cv2.imshow('and',and1)

xor1 = cv2.bitwise_xor(image1,image2)
cv2.imshow('xor',xor1)
```

3.Gray Level transformation

```
import cv2
import numpy as np

org_image = cv2.imread('flower2.png')
cv2.imshow('image',org_image)

image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)
cv2.imshow('gray',image)
```

```
neg = 255-image
cv2.imshow('neg',neg)

c = 255/np.log(1 + np.max(image))

log_t = c * np.log(1 + image)
log_t = np.uint8(log_t)
cv2.imshow('log',log_t)

gamma=0.5



cont = (image-min_int)*((b-a)/(max_int-min_int)) + a
cont = np.uint8(cont)
cv2.imshow('cont',cont)
```

4.Bit plane slicing

```
import cv2

def bps(image,bit):
    plane = cv2.bitwise_and(image , 2**bit)
    return plane

image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)
```

```
num_bits = 8


bit_plane = [bps(image,bit) for bit in range(num_bits)]


for bit,plane in enumerate(bit_plane):
    cv2.imshow(f'{bit}',plane)
```

5.Histogram equilisation

```
import cv2


image=cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)
cv2.imshow('image',image)


equi = cv2.equalizeHist(image)
cv2.imshow('equi',equi)
```

6.Low and High pass in frequency domain

```
import cv2
import numpy as np
from matplotlib import pyplot as plt


def low_pass(image,freq):
    rows,cols = image.shape
    crow , ccol = rows//2,cols//2
    dft = cv2.dft(np.float32(image),flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
    mask = np.zeros((rows,cols,2),np.uint8)
    mask[crow-freq:crow+freq , ccol-freq:ccol+freq]=1

    dft_shift_low = mask*dft_shift

    f_ishift = np.fft.ifftshift(dft_shift_low)

    img_back = cv2.idft(f_ishift)
    img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])
    return img_back
def high_pass(image,freq):
    rows,cols = image.shape
    crow , ccol = rows//2,cols//2

    dft = cv2.dft(np.float32(image),flags=cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
    mask = np.ones((rows,cols,2),np.uint8)
    mask[crow-freq:crow+freq , ccol-freq:ccol+freq]=0

    dft_shift_low = mask*dft_shift

    f_ishift = np.fft.ifftshift(dft_shift_low)
    img_back = cv2.idft(f_ishift)

    img_back = cv2.magnitude(img_back[:,:,0],img_back[:,:,1])

    return img_back



image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)

freq = 50

lp = low_pass(image,freq)

hp = high_pass(image,freq)

plt.subplot(131),plt.title('image'),plt.imshow(image,cmap='gray'),plt.axis('off')

plt.subplot(132),plt.title('lp'),plt.imshow(lp,cmap='gray'),plt.axis('off')

plt.subplot(133),plt.title('hp'),plt.imshow(hp,cmap='gray'),plt.axis('off')

plt.show()
```

7.High and low pass in spatial domain

```
import cv2

import numpy as np

from matplotlib import pyplot as plt


image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)

plt.subplot(131),plt.title('image'),plt.imshow(image,cmap='gray'),plt.axis('off')

k = 3

k1 = np.ones((k,k),np.float32)/(k*k)

lp = cv2.filter2D(image,-1,k1)
```

```
plt.subplot(132),plt.title('lp'),plt.imshow(lp,cmap='gray'),plt.axis('off')


k2 = np.array([[0,1,0],[1,-4,1],[0,1,0]])
hp = cv2.filter2D(image , -1,k2)
plt.subplot(133),plt.title('hp'),plt.imshow(hp,cmap='gray'),plt.axis('off')
plt.show()
```

8.salt and pepper using median filter

```
import cv2


def denoi(image):
    return cv2.medianBlur(image,5)


image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)
cv2.imshow('image',image)


denoise = denoi(image)
cv2.imshow('de',denoise)
```

9.Sobel , Laplacian , prewitt filter

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)
plt.subplot(221),plt.title('image'),plt.imshow(image,cmap='gray'),plt.axis('off')
```

```
sobel_x = cv2.Sobel(image , cv2.CV_64F , 1,0,ksize=5)
sobel_y = cv2.Sobel(image , cv2.CV_64F , 0 , 1 , ksize=5)
sobel = np.sqrt(sobel_x**2 + sobel_y**2)

plt.subplot(222),plt.title('sobel'),plt.imshow(sobel,cmap='gray'),plt.axis('off')

pxk = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
px = cv2.filter2D(image , -1 , pxk)

pyk = np.array([[-1,-1,-1],[0,0,0],[1,1,1]])
py = cv2.filter2D(image,-1,pyk)

p = np.sqrt(px**2 + py**2)

plt.subplot(223),plt.title('prewitt'),plt.imshow(p,cmap='gray'),plt.axis('off')

l = cv2.Laplacian(image , cv2.CV_64F)
plt.subplot(224),plt.title('laplacian'),plt.imshow(l,cmap='gray'),plt.axis('off')
plt.show()
```

10.Sharpening image

```
import cv2
import numpy as np

def shar(image ):
    fil = np.array([[0,1,0],[1,-4,1],[0,1,0]])
    sharp = cv2.filter2D(image , -1 , fil)
    sharp = cv2.addWeighted(image , 1 , sharp , -0.5 , 0)
    cv2.imshow('sharpened',sharp)

image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)
cv2.imshow('image',image)
sharpen = shar(image)
```

11.Morphological filters

```
import cv2
import numpy as np

image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)
cv2.imshow('image',image)
kernel=np.ones((5,5),np.uint8)
erosion = cv2.erode(image , kernel , iterations=1)
dilation = cv2.dilate(image , kernel,iterations=1)

cv2.imshow('erosion',erosion)
cv2.imshow('dilation',dilation)

opening = cv2.morphologyEx(image , cv2.MORPH_OPEN , kernel)
closing = cv2.morphologyEx(image , cv2.MORPH_CLOSE , kernel)
```

```
cv2.imshow('opening',opening)

cv2.imshow('closing',closing)
```

12.Segmentation

```
import cv2

import numpy as np


image = cv2.imread('flower2.png')

cv2.imshow('image',image)

pixels = image.reshape((-1,3))

pixels = np.float32(pixels)


criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER , 100 , 0.2)


k=3


_,labels,centers = cv2.kmeans(pixels , k , None , criteria , 10 , cv2.KMEANS_RANDOM_CENTERS)


centers = np.uint8(centers)

seg_img=centers[labels.flatten()]

seg_img = seg_img.reshape(image.shape)

cv2.imshow('seg',seg_img)
```

13.Image Watermarking

```
import cv2
import numpy as np

def add_text_watermark(file, out, mark, size, color, opacity, angle, space):

 # Load the image

 image = cv2.imread(file)

 (h, w) = image.shape[:2]

 # Create a blank image with the same dimensions for the watermark

 watermark = np.zeros((h, w, 3), dtype="uint8")

 # Set the font, scale, and thickness for the watermark text

 font = cv2.FONT_HERSHEY_SIMPLEX

 scale = size / 100

 color = tuple(int(color[i:i+2], 16) for i in (1, 3, 5)) # Convert hex color to BGR

 thickness = int(size / 20)

 # Calculate the text size

 (text_w, text_h), baseline = cv2.getTextSize(mark, font, scale, thickness)

 text_h += baseline

 # Create a transparent overlay

 overlay = watermark.copy()

 # Draw the watermark text repeatedly on the overlay

 for y in range(0, h, text_h + space):

 for x in range(0, w, text_w + space):

 cv2.putText(overlay, mark, (x, y), font, scale, color, thickness, cv2.LINE_AA)

 # Rotate the overlay if an angle is specified

 if angle != 0:

 M = cv2.getRotationMatrix2D((w // 2, h // 2), angle, 1)
 overlay = cv2.warpAffine(overlay, M, (w, h))
 # Blend the overlay with the original image
 cv2.addWeighted(overlay, opacity, image, 1 - opacity, 0, image)
 # Save the watermarked image
 cv2.imwrite(out, image)
 # Display the watermarked image
 cv2.imshow('Watermarked Image', image)
# Wait for a key press and close the image window
 cv2.waitKey(0)
 cv2.destroyAllWindows()
# Example usage
file = 'sunflower.jpg'
out = 'watermarked_image.png'
mark = 'UVCE'
size = 80
color = '#ffffff'
opacity = 0.2
angle = 30
space = 40
add_text_watermark(file, out, mark, size, color, opacity, angle, space)
```

14.Image restoration

```
import cv2
import numpy as np
```

```
image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)
cv2.imshow('image',image)

def res(image , kernel=3):
    res_img = cv2.medianBlur(image , kernel)
    cv2.imshow('restored',res_img)

kernel=3
res(image,kernel)
```

15.Block Truncation code

```
import cv2
import numpy as np

def btc(image , bs=8):
    h,w = image.shape
    num_h = h//bs
    num_w = w//bs

    compressed_image = np.zeros((h,w),np.float32)

    for i in range(num_h):
        for j in range(num_w):
            block = image[i*bs : (i+1)*bs , j*bs:(j+1)*bs]
            block_mean = np.mean(block)
        block_std = np.std(block)

        thres = block_mean


        compressed_block = np.where(block<=thres , 0 ,255)


        compressed_image[i*bs : (i+1)*bs , j*bs:(j+1)*bs] = compressed_block


        cv2.imshow('c',compressed_image)


image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)

cv2.imshow('image',image)


bs=8

btc(image,bs)
```

16.Edge detection

```
import cv2


image = cv2.imread('flower2.png',cv2.IMREAD_GRAYSCALE)

cv2.imshow('image',image)


edges = cv2.Canny(image , 200 , 100)

cv2.imshow('edges',edges)
```