```python
import matplotlib.pyplot as plt
import cv2
import numpy as np


def hist(img):
    d = dict()
    for i in range(256):
        d[i]=0
    for i in img:
        for j in i:
            d[j]+= 1
    return d

def main():
    img_path = "mountain.jpeg"
    rgb = plt.imread(img_path)
    gray = cv2.cvtColor(rgb,cv2.COLOR_RGB2GRAY)
    _,binary = cv2.threshold(gray,126,255,cv2.THRESH_BINARY)
    # histshow(rgb,gray,binary)
    # pointprocessing(gray)
    # neighbourprocessing(gray)
    # bitslici_masking(gray)
    # addnoise(gray)
    # shifting(gray)
    # morphological(binary)
    # histogramequalization(gray)
    # frequencydomain(gray)
    #jpg2png(img_path)

def imshow(img_set,x,y):
    for i in range(len(img_set)):
        plt.subplot(x,y,i+1)
        plt.imshow(img_set[i],'gray')
    plt.savefig('result')
    plt.show()

def plotshow(plt_list,x,y):
    for i in range(len(plt_list)):
        plt.subplot(x,y,i+1)
        plt.plot(plt_list[i],'red')
    plt.show()
```

```
def histshow(rgb,gray,binary):

    r,c = gray.shape
    binary2 = gray.copy()
    for i in range(r):
        for j in range(c):
            if(gray[i][j]<127):
                binary2[i][j]=0
            else:
                binary2[i][j]=1
    red = rgb[:,:,0]
    green = rgb[:,:,1]
    blue = rgb[:,:,2]

    r = cv2.calcHist([rgb],[0],None,[256],[0,256])
    g = cv2.calcHist([rgb],[1],None,[256],[0,256])
    b = cv2.calcHist([rgb],[2],None,[256],[0,256])
    g = cv2.calcHist([rgb],[0],None,[256],[0,256])
    bi = cv2.calcHist([binary2],[0],None,[256],[0,256])

    hist_list=[r,g,b,g,bi]
    img_set = [rgb,gray,binary,binary2,red,green,blue]
    imshow(img_set,4,2)
    plotshow(hist_list,3,2)

def pointprocessing(img):
    r,c = img.shape
    t1,t2 = 100,140
    #first condition:s = 100, if r >= T1 and r <= T2; otherwise s =
10.
    c1 = img.copy()
    for i in range(r):
        for j in range(c):
            if(img[i][j]>=t1 and img[i][j]<=t2):
                c1[i][j]=100
            else:
                c1[i][j]=10
    #second condition
    c2 = img.copy()
    for i in range(r):
        for j in range(c):
            if(img[i][j]<=t2 and img[i][j]>=t1):
                c2[i][j]= 100
```

```
            else:
                c2[i][j]=img[i][j]
    #third condition
    c3 = 3*np.log(img+1)
    c4 = 3*(1e-6+img) **3

    img_set = [img,c1,c2,c3,c4]
    imshow(img_set,3,2)

def neighbourprocessing(img):
    #filter
    hor = np.array([[-2,-2,-2],[0,0,0],[2,2,2]])
    ver = np.array([[2,0,-2],[2,0,-2],[2,0,-2]])
    lapla = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
    blur = np.ones((3,3))/9
    edge = np.array([[-1,-1,-1],[-1,10,-1],[-1,-1,-1]])
    #convulation operation
    h = cv2.filter2D(img,-1,hor)
    v = cv2.filter2D(img,-1,ver)
    lap = cv2.filter2D(img,-1,lapla)
    blurim = cv2.filter2D(img,-1,blur)
    #custom convulation
    r,c = img.shape
    x,y = edge.shape
    e = np.zeros((r-x-1,c-y-1),dtype=np.uint8)
    for i in range(r-x-1):
        for j in range(c-y-1):
            sum = np.sum(np.multiply(img[i:i+x,j:j+y],edge))
            if(sum<0):
                e[i][j]=0
            elif(sum>255):
                e[i][j]=255
            else:
                e[i][j]=sum

    img_set = [h,v,lap,blurim,e]
    imshow(img_set,2,3)

def bitslici_masking(img):
    r,c = img.shape

    b16 = img&16
    b32 = img&32
```

```
    b64 = img&64
    b128 = img&128
    bx = b64+b128+b32
    mask = np.zeros((r,c),dtype=np.uint8)
    mask[40:120,50:250]=255
    mask = mask & img

    img_set=[bx,b16,b32,b64,b128,mask]
    imshow(img_set,3,2)

def addnoise(x):
    img = x.copy()
    r,c = img.shape

    t = r*c//50
    for i in range(t):
        x = np.random.randint(0,r)
        y = np.random.randint(0,c)
        if(i%2==0):
            img[x][y]=255
        else:
            img[x][y]=0
    plt.subplot(1,2,1)
    plt.imshow(img,'gray')
    plt.show()

def shifting(img):
    rs = img.copy()
    ls = img.copy()
    ns = img.copy()
    rs = rs -80
    ls = ls+50
    r,c = img.shape
    for i in range(r):
        for j in range(c):
            if(ns[i][j]<=30):
                ns[i][j]=30
            elif(ns[i][j]>=60):
                ns[i][j]=80
    imgh = cv2.calcHist([img],[0],None,[256],[0,256])
    rsh = cv2.calcHist([rs],[0],None,[256],[0,256])
    lsh = cv2.calcHist([ls],[0],None,[256],[0,256])
    nsh = cv2.calcHist([ns],[0],None,[256],[0,256])
```

```
    plt_list=[imgh,rsh,lsh,nsh]
    plotshow(plt_list,2,2)

def morphological(binary):
    r,c = binary.shape
    kernal = np.ones((3,3),np.uint8)
    x,y = kernal.shape

    img_erosion = cv2.erode(binary,kernal)
    img_dialation = cv2.dilate(binary,kernal)

    img_opening = cv2.dilate(img_erosion,kernal)
    img_closing = cv2.erode(img_dialation,kernal)

    #custom morphological operation
    cus_ero = np.zeros((r-x-1,c-y-1))
    cus_dia = np.zeros((r-x-1,c-y-1))
    for i in range(r-x-1):
        for j in range(c-y-1):
            sum = np.sum(np.multiply(binary[i:i+x,j:j+y],kernal))
            if(sum==2295):
                cus_ero[i][j]=255
            elif(sum>=255):
                cus_dia[i][j]=255

img_set=[img_erosion,img_dialation,img_opening,img_closing,cus_ero,cu
s_dia]
    imshow(img_set,3,2)

def histogramequalization(img):
    equlize_hist = cv2.equalizeHist(img)
    n_hist = cv2.calcHist([img],[0],None,[256],[0,256])
    e_hist = cv2.calcHist([equlize_hist],[0],None,[256],[0,256])
    plt_list=[n_hist,e_hist]
    plotshow(plt_list,1,2)

def frequencydomain(img):

    r,c = img.shape
    #fd = frequency domain
    #fds = centered frequency
    #h = filter
    #fdsh = filtered in frequency domain
```

```
#fdh = invert shifted
#sdh = invert in spatial domain
fd = np.fft.fft2(img)
fds = np.fft.fftshift(fd)
#for printing fds_abs=np.log1p(np.abs(fds))
#making low pass filter

#butterworth filter
h = np.zeros((r,c),dtype=np.float32)
d0 = 20
for u in range(r):
    for v in range(c):
        d = np.sqrt((u-r/2)**2+(v-c/2)**2)
        h[u,v]=1/(1+(d/d0)**2)
#for printing
h_abs=np.log1p(np.abs(h))
#filtered in frequency domain
fdsh = fds*h_abs
#for printing fdsh_abs=np.log1p(np.abs(fdsh))
#inverse in spatial domain
#inverse shifting
fdh = np.fft.ifftshift(fdsh)
#spatial domain
sdh = np.abs(np.fft.ifft2(fdh))

#high pass filter
hp = 1 - h
hp_abs = np.log1p(np.abs(hp))
#filtered in frequency domain
fdshp =  fds*hp
fdhp = np.fft.ifftshift(fdshp)
sdhp = np.abs(np.fft.ifft2(fdhp))

img_set = [img,h_abs,sdh,hp_abs,sdhp]
imshow(img_set,2,3)
#gaussain filter
h = np.zeros((r,c),dtype=np.float32)
d0 = 40
for u in range(r):
    for v in range(c):
        d = np.sqrt((u-r/2)**2+(v-c/2)**2)
        h[u,v] = np.exp(-(d**2)/(2*d0*d0))
```

```
    h_abs = np.log1p(np.abs(h))
    fdsh = fds*h
    fdh = np.fft.ifftshift(fdsh)
    sdh = np.abs(np.fft.ifft2(fdh))
    #highpass
    hp = 1 - h
    hp_abs = np.log1p(np.abs(hp))
    fdshp = fds*hp
    fdhp = np.fft.ifftshift(fdshp)
    sdhp = np.abs(np.fft.ifft2(fdhp))

    img_set = [img,h_abs,sdh,hp_abs,sdhp]
    imshow(img_set,2,3)

def jpg2png(img_path):
    name,format = img_path.split(".")
    if(format == 'jpeg' or format=='jpg'):
        img = cv2.imread(img_path)
        cv2.imwrite(name+".png",img)
    else:
        img = cv2.imread(img_path)
        cv2.imwrite(name+".jpg",img)



if __name__=="__main__":
    main()
```