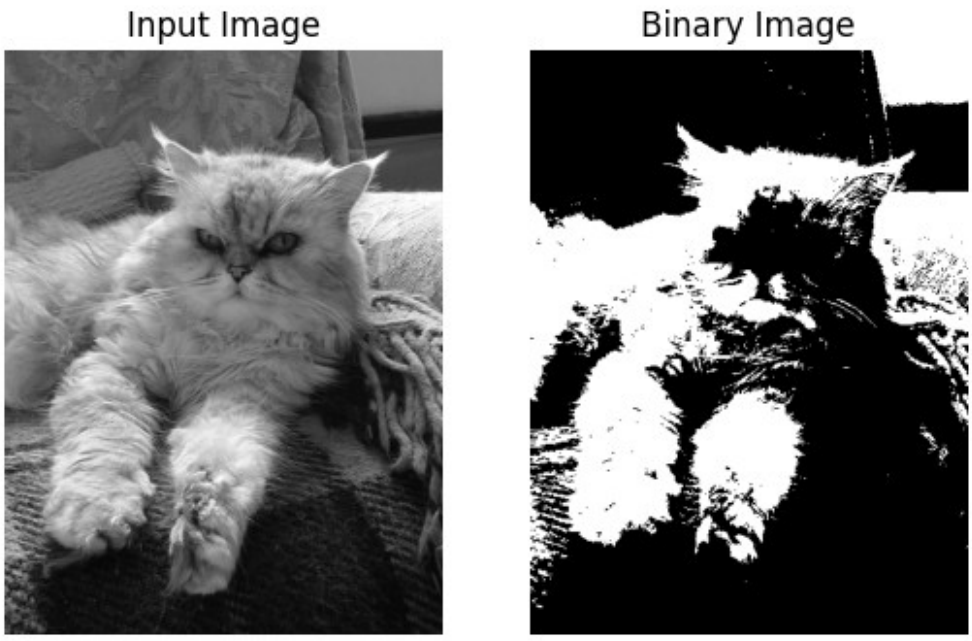
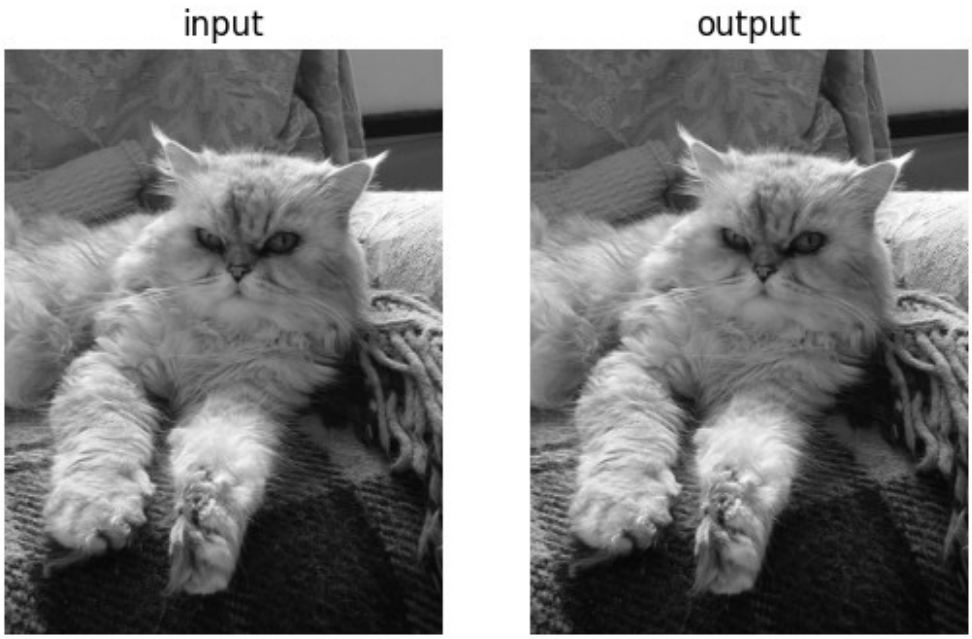
```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
image = cv.imread("cat1.jpg",0)
plt.imshow(image,cmap='gray')
plt.axis(False)
plt.show()
```

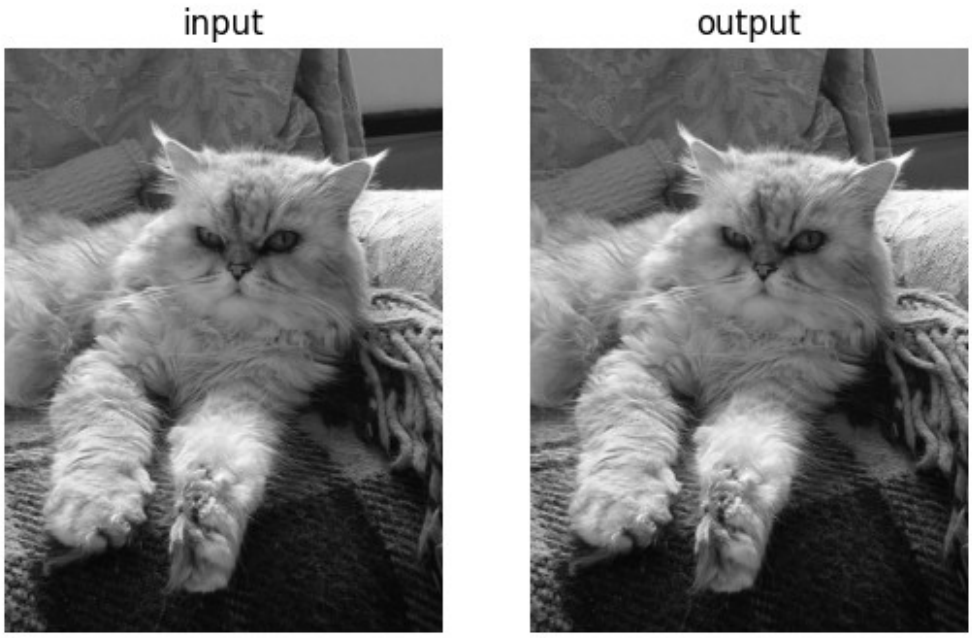
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import os
image = cv.imread("cat1.jpg")
image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
plt.subplot(1,2,1)
plt.imshow(image, cmap='gray')
plt.title('Input Image')
plt.axis('off')
threshold=127
image.shape
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        if image[i][j] >= threshold:
            image[i][j]=255
        else:
            image[i][j]=0

plt.subplot(1,2,2)
plt.imshow(image, cmap='gray')
plt.title('Binary Image')
plt.axis('off')
plt.show()
```



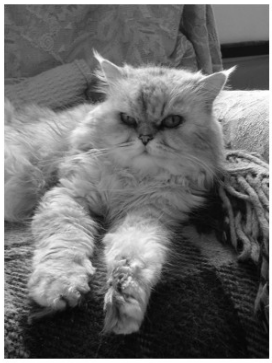
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
Image = cv.imread("cat1.jpg",0)
plt.subplot(1,2,1)
plt.imshow(Image, cmap='gray')
plt.axis('off')
plt.title('input')
def get_boundary_sum(img):
    horizontal_top = img[0,:]
    horizontal_bottom = img[-1,:]
    vertical_left = img[:,0]
    vertical_right = img[:,-1]
    boundary_sum = np.sum(horizontal_top) + np.sum(horizontal_bottom)
+ np.sum(vertical_left) + np.sum(vertical_right)-horizontal_top[0]-
horizontal_top[-1]-horizontal_bottom[0]-horizontal_bottom[-1]
    return boundary_sum
get_boundary_sum(img)
img2 = np.copy(img)
#replace middle
img2[img.shape[0]//2, img.shape[1]//2] = get_boundary_sum(img)
plt.subplot(1,2,2)
plt.imshow(img2, cmap='gray')
plt.axis('off')
plt.title('output')
plt.show()
```

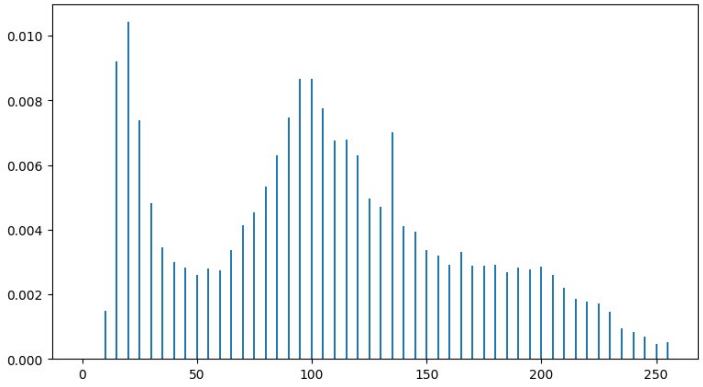


```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import os
img= cv.imread("cat1.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
plt.subplot(1,2,1)
plt.title("input")
plt.imshow(img, cmap='gray')
plt.axis('off')
def get_diagonal_sum(img):
    diagonal_1 = np.trace(img)
    diagonal_2 = np.trace(np.fliplr(img))
    diagonal_sum = diagonal_1 + diagonal_2 - img[img.shape[0]//2,
img.shape[1]//2]
    return diagonal_sum
get_diagonal_sum(img)
img[img.shape[0]//2, img.shape[1]//2] = get_diagonal_sum(img)
plt.subplot(1,2,2)
plt.title("output")
plt.imshow(img,cmap='gray')
plt.axis('off')
plt.show()
```



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

image = cv.imread("cat1.jpg")
img = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.imshow(img, cmap='gray')
plt.axis('off')
img2 = np.copy(img)
row = img2.shape[0]
column = img2.shape[1]
img2 = np.reshape(img2,img2.shape[0]*img2.shape[1])
img2
img2=np.sort(img2)
img2.shape
hist = np.zeros(256)
for i in img2:
    hist[i] = hist[i]+1
hist = hist/(row*column)
hist_difference = np.array([x if i%5 == 0 else 0 for i,x in
enumerate(hist)])
plt.subplot(1,2,2)
plt.bar(np.arange(256),hist_difference)
plt.show()
```

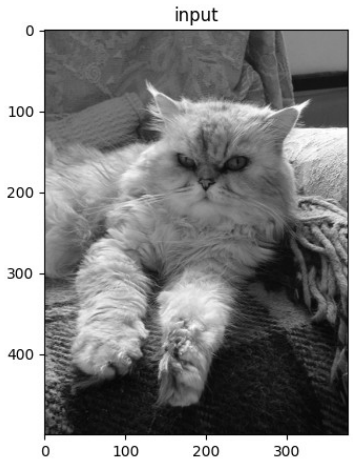




```
import cv2 as cv
import matplotlib.pyplot as plt

image = cv.imread("cat1.jpg")
img = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.title('input')
plt.imshow(img,cmap='gray')
plt.axis('off')
def image_negative(img):
    negative = 255-img
    return negative
img_neg = image_negative(img)
img_neg
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(img_neg, cmap="gray")
plt.axis("off")
plt.show()
```

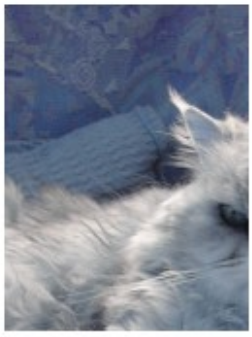
```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
#image loading
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.title('input')
image = cv.imread("cat1.jpg",0)
plt.imshow(image,cmap='gray')

def formula(x):
    c = 255/np.log(1+255)
    return c*np.log(1+x)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        image[i][j] = formula(image[i][j])
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(image,cmap="gray")
plt.axis(False)
plt.show()
```

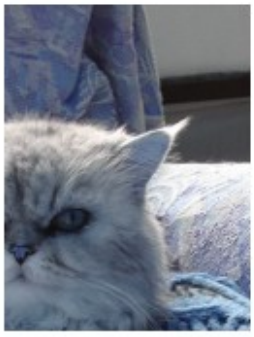
```
C:\Users\nazmu\AppData\Local\Temp\ipykernel_9600\3636713417.py:13:
RuntimeWarning: overflow encountered in scalar add
  return c*np.log(1+x)
C:\Users\nazmu\AppData\Local\Temp\ipykernel_9600\3636713417.py:13:
RuntimeWarning: divide by zero encountered in log
  return c*np.log(1+x)
C:\Users\nazmu\AppData\Local\Temp\ipykernel_9600\3636713417.py:16:
RuntimeWarning: invalid value encountered in cast
  image[i][j] = formula(image[i][j])
```

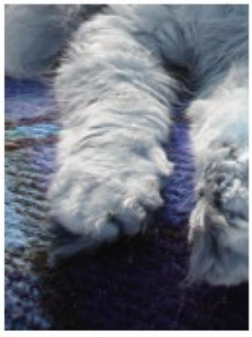


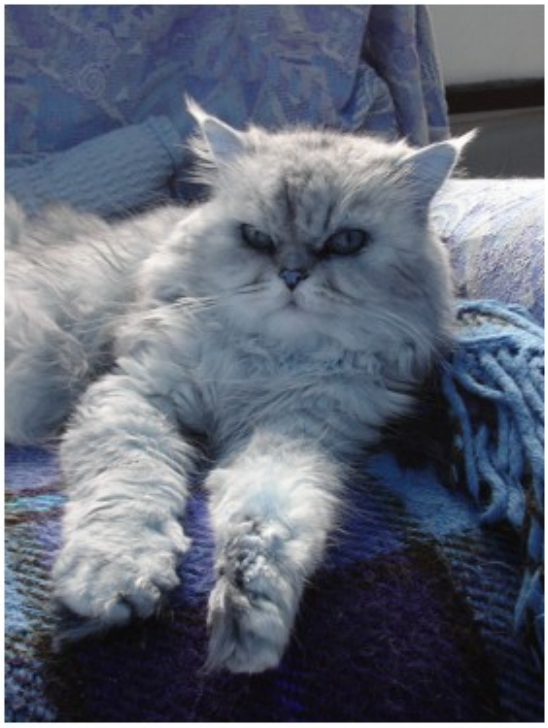
```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
#image loading
image = cv.imread("cat1.jpg",0).astype("float")
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.title('input')
plt.imshow(image,cmap='gray')
plt.axis(False)
def formula(x):
    gamma = 1.8
    c = 255/np.log(1+255)
    return c*(x**gamma)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):

        image[i][j] = float(image[i][j])/float(255)
        image[i][j] = formula(image[i][j])
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(image,cmap="gray")
plt.axis(False)
plt.show()
```

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("cat1.jpg")
def split(img):
    img1 = img[:img.shape[0]//2, :img.shape[1]//2]
    img2 = img[:img.shape[0]//2, img.shape[1]//2:]
    img3 = img[img.shape[0]//2:, :img.shape[1]//2]
    img4 = img[img.shape[0]//2:, img.shape[1]//2:]
    return img1, img2, img3, img4
img1, img2, img3, img4 = split(img)
plt.subplot(2,2,1)
plt.imshow(img1, cmap='gray')
plt.axis("off")
plt.subplot(2,2,2)
plt.imshow(img2, cmap='gray')
plt.axis("off")
plt.subplot(2,2,3)
plt.imshow(img3, cmap='gray')
plt.axis("off")
plt.subplot(2,2,4)
plt.imshow(img4, cmap='gray')
plt.axis("off")
plt.show()
img_merge = np.concatenate((np.concatenate((img1, img2), axis=1),
np.concatenate((img3, img4), axis=1)), axis=0)
plt.imshow(img_merge, cmap='gray')
plt.axis('off')
plt.show()
```

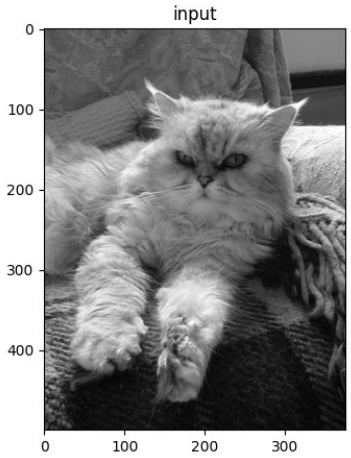




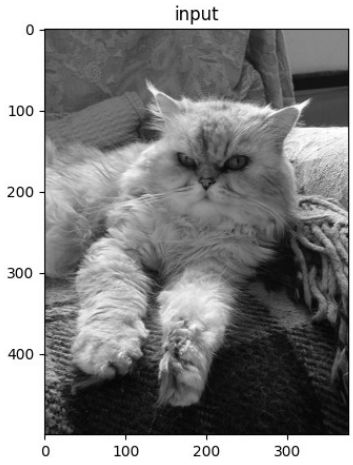


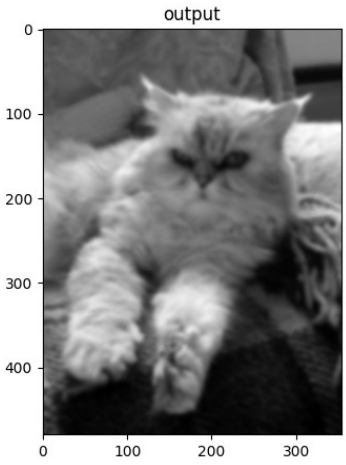


```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img2 = cv.imread('cat1.jpg')
img2 = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.title('input')
plt.imshow(img2, cmap='gray')
left = np.zeros((img2.shape[0],50))
img2 = np.concatenate((img2, left), axis=1)
img2 = np.concatenate((left, img2), axis=1)
up = np.zeros((50, img2.shape[1]))
img2 = np.concatenate((img2, up), axis=0)
img2 = np.concatenate((up, img2), axis=0)
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(img2, cmap='gray')
plt.axis('off')
plt.show()
```

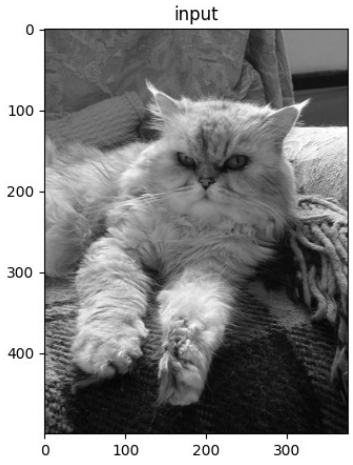


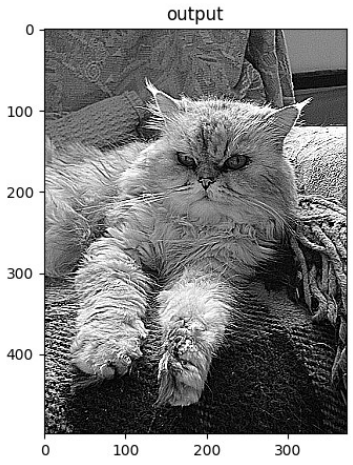
```
import numpy as np
def matrix_sum(mat_1,mat_2):
    sum = 0
    for i in range(mat_1.shape[0]):
        for j in range(mat_1.shape[1]):
            sum = sum + mat_1[i][j] * mat_2[i][j]
    return sum
def filter_operation(image,kernel):
    #must use a odd size of filter
    kernel_center = (kernel.shape[0]-1)//2
    kernel_dimension = kernel.shape[0]
    image_height = image.shape[0]
    image_width = image.shape[1]
    out_image_height = int(image_height-kernel_dimension+1)
    out_image_width = int(image_width-kernel_dimension+1)
    out_image = np.zeros((out_image_height,out_image_width))
    for row in range(out_image_height):
        for column in range(out_image_width):
            mat =
image[row:row+kernel_dimension,column:column+kernel_dimension]
            out_image[row,column] =
matrix_sum(mat,kernel)/kernel_dimension/kernel_dimension
    return out_image
kernel = np.array([[1,2,1],[2,4,2],[1,2,1]])/16
import cv2
import matplotlib.pyplot as plt
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.title('input')
image = cv2.imread("cat1.jpg")
image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
plt.imshow(image, cmap="gray")
filtered_image = image
for i in range(10):
    filtered_image = filter_operation(filtered_image,kernel)
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(filtered_image, cmap="gray")
plt.show()
```


input
0
100
200
300
400
0
100
200
300


output
0
100
200
300
400
0
100
200
300

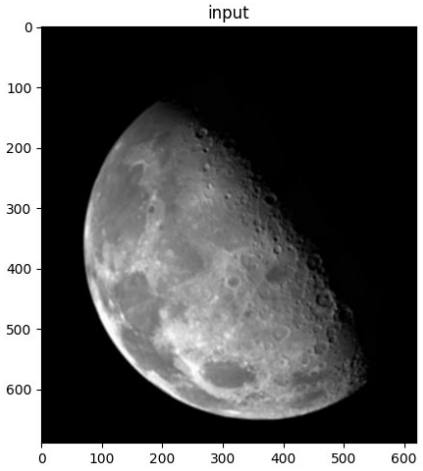
```
import numpy as np
def matrix_sum(mat_1,mat_2):
    sum = 0
    for i in range(mat_1.shape[0]):
        for j in range(mat_1.shape[1]):
            sum = sum + mat_1[i][j] * mat_2[i][j]
    return sum
def filter_operation(image,kernel):
    #must use a odd size of filter
    kernel_center = (kernel.shape[0]-1)//2
    kernel_dimension = kernel.shape[0]
    image_height = image.shape[0]
    image_width = image.shape[1]
    out_image_height = int(image_height-kernel_dimension+1)
    out_image_width = int(image_width-kernel_dimension+1)
    out_image = np.zeros((out_image_height,out_image_width))
    for row in range(out_image_height):
        for column in range(out_image_width):
            mat =
image[row:row+kernel_dimension,column:column+kernel_dimension]
            out_image[row,column] = matrix_sum(mat,kernel)
+image[row+kernel_center,column+kernel_center]
    return out_image
sharpening_kernel = np.array([[-1, -1, -1],
                              [-1, 8, -1],
                              [-1, -1, -1]])
import cv2
import matplotlib.pyplot as plt
image = cv2.imread("cat1.jpg")
image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.title('input')
plt.imshow(image, cmap="gray")
filtered_image = filter_operation(image,sharpening_kernel)
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(filtered_image ,cmap='gray',vmin=0,vmax=255)
plt.show()
```

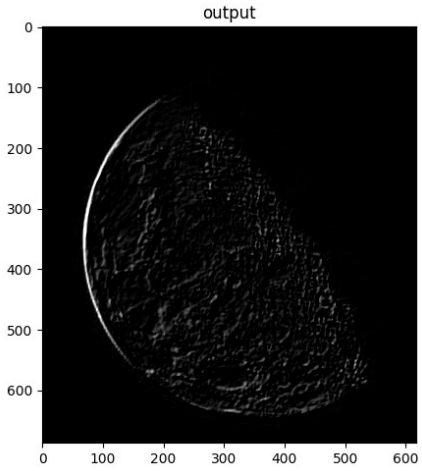

input
0
100
200
300
400
0
100
200
300


output
0
100
200
300
400
0
100
200
300

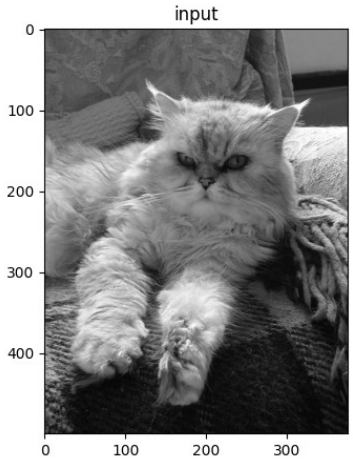
```
#filtering funtion
import numpy as np
import cv2
import matplotlib.pyplot as plt
def matrix_sum(mat_1,mat_2):
    sum = 0
    for i in range(mat_1.shape[0]):
        for j in range(mat_1.shape[1]):
            sum = sum + mat_1[i][j] * mat_2[i][j]
    return sum
def filter_operation(image,kernel):
    #must use a odd size of filter
    kernel_center = (kernel.shape[0]-1)//2
    kernel_dimension = kernel.shape[0]
    image_height = image.shape[0]
    image_width = image.shape[1]
    out_image_height = int(image_height-2*kernel_center)
    out_image_width = int(image_width-2*kernel_center)
    out_image = np.zeros((out_image_height,out_image_width))
    for row in range(out_image_height):
        for column in range(out_image_width):
            mat =
image[row:row+kernel_dimension,column:column+kernel_dimension]
            #print(mat)
            out_image[row,column] = matrix_sum(mat,kernel)
    return out_image
def padd_image(img2,n):
    left = np.zeros((img2.shape[0],n))
    left = left + 255
    img2 = np.concatenate((img2, left), axis=1)
    img2 = np.concatenate((left, img2), axis=1)

    up = np.zeros((n,img2.shape[1]))
    up = up + 255
    img2 = np.concatenate((img2, up), axis=0)
    img2 = np.concatenate((up, img2), axis=0)
    return img2
blur_kernel = np.array([
    [1,2,1],
    [2,4,2],
    [1,2,1]
])/16
line_detection_1 = np.array([
    [-1,-2,-1],
    [0,0,0],
    [1,2,1]
])
line_detection_2 = np.array([
    [-1,0,1],
```

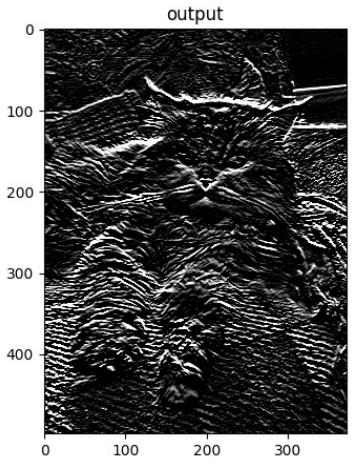
```
    [-2,0,2],
    [-1,0,1]
])
image = cv2.imread('image.png')
image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
filtered_image1 = filter_operation(image,line_detection_1)
filtered_image2 = filter_operation(image,line_detection_2)
filtered_image = filtered_image1+filtered_image2

# filtered_image2 = padd_image(filtered_image1,1)
# filtered_image2 = image+filtered_image2
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.title('input')
plt.imshow(image,cmap='gray')
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(filtered_image, cmap="gray",vmin=0,vmax=255)
plt.tight_layout()
plt.show()
```



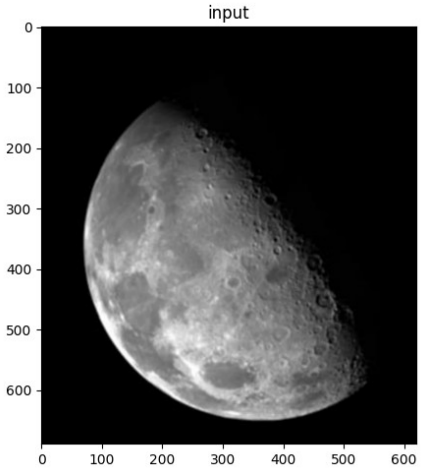


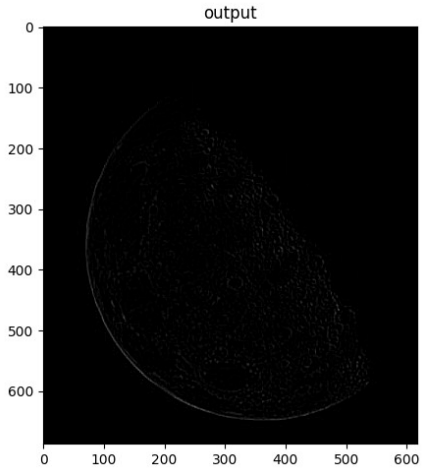
```
import numpy as np
def matrix_sum(mat_1,mat_2):
    sum = 0
    for i in range(mat_1.shape[0]):
        for j in range(mat_1.shape[1]):
            sum = sum + mat_1[i][j] * mat_2[i][j]
    return sum
def filter_operation(image,kernel):
    #must use a odd size of filter
    kernel_center = (kernel.shape[0]-1)//2
    kernel_dimension = kernel.shape[0]

    image_height = image.shape[0]
    image_width = image.shape[1]
    out_image_height = int(image_height-kernel_dimension+1)
    out_image_width = int(image_width-kernel_dimension+1)
    out_image = np.zeros((out_image_height,out_image_width))
    #print(image.shape)
    #print(out_image.shape)

    for row in range(out_image_height):
        for column in range(out_image_width):
            mat =
image[row:row+kernel_dimension,column:column+kernel_dimension]
            #print(mat)
            out_image[row,column] = matrix_sum(mat,kernel)
    return out_image
kernel = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
blur_kernel = np.array([[1,2,1],[2,4,2],[1,2,1]])/16
kernel
import cv2
import matplotlib.pyplot as plt
image = cv2.imread("cat1.jpg")
image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.title('input')

plt.imshow(image, cmap="gray")
filtered_image = image
for i in range(20):
    filtered_image = filter_operation(filtered_image,blur_kernel)
filtered_image = filter_operation(image,kernel)
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(filtered_image, cmap="gray",vmin=0,vmax=100)
plt.show()
```


input
0
100
200
300
400
0
100
200
300


output
0
100
200
300
400
0
100
200
300

```python
#filtering funtion
import numpy as np
import cv2
import matplotlib.pyplot as plt
def matrix_sum(mat_1,mat_2):
    sum = 0
    for i in range(mat_1.shape[0]):
        for j in range(mat_1.shape[1]):
            sum = sum + mat_1[i][j] * mat_2[i][j]
    return sum
def filter_operation(image,kernel):
    #must use a odd size of filter
    kernel_center = (kernel.shape[0]-1)//2
    kernel_dimension = kernel.shape[0]
    image_height = image.shape[0]
    image_width = image.shape[1]
    out_image_height = int(image_height-2*kernel_center)
    out_image_width = int(image_width-2*kernel_center)
    out_image = np.zeros((out_image_height,out_image_width))
    for row in range(out_image_height):
        for column in range(out_image_width):
            mat =
image[row:row+kernel_dimension,column:column+kernel_dimension]
            #print(mat)
            out_image[row,column] = matrix_sum(mat,kernel)
    return out_image
edge_detection_kernel_2 = np.array([
    [-1,-1,-1],
    [-1,8,-1],
    [-1,-1,-1]
])
image = cv2.imread('image.png')
image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
filter_image = filter_operation(image, edge_detection_kernel_2)
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.title('input')
plt.imshow(image,cmap='gray')
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(filter_image, cmap="gray", vmin=0, vmax=255)
plt.tight_layout()
plt.show()
```


input
0
100
200
300
400
500
600
0
100
200
300
400
500
600

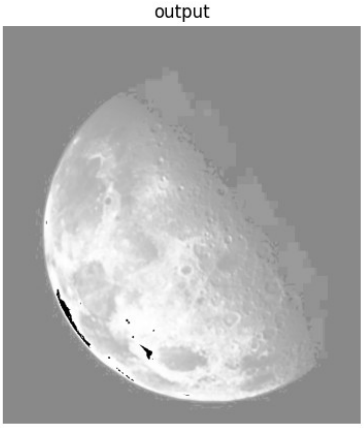

output
0
100
200
300
400
500
600
0
100
200
300
400
500
600

```
import numpy as np
def calculate_probability(image,hist):
    total_pixels = image.shape[0]
    print(total_pixels)
    for i in range(len(hist)):
        hist[i]=hist[i]/total_pixels
    return hist
def histogram(image,l):
    image = image.reshape(image.shape[0] * image.shape[1])
    histogram = np.zeros(l)
    for i in image:
        histogram[i] = histogram[i]+1
    histogram = calculate_probability(image,histogram)
    return histogram
def round_of_values(hist,l):
    #running_sum
    running_sum = np.zeros_like(hist)
    sum = 0
    for i in range(len(running_sum)):
        sum = sum + hist[i]
        running_sum[i] = sum*l
    round_of_values = np.round(running_sum)
    return round_of_values
def histogram_eualization(image,l):
    hist = histogram(image,l)
    round_of = round_of_values(hist,l)
    image_2 = np.zeros_like(image)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            value = image[i][j]
            image_2[i][j] = round_of[value]
            #print(round_of)
    return image_2
import cv2 as cv
import matplotlib.pyplot as plt
image= cv.imread("image.png")
img= cv.cvtColor(image,cv.COLOR_BGR2GRAY)
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
plt.title('input')
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.subplot(1,2,2)
plt.title('output')
img2 = histogram_eualization(img,256)
plt.imshow(img2, cmap='gray',vmin=0,vmax=255)
plt.axis('off')
plt.show()
```

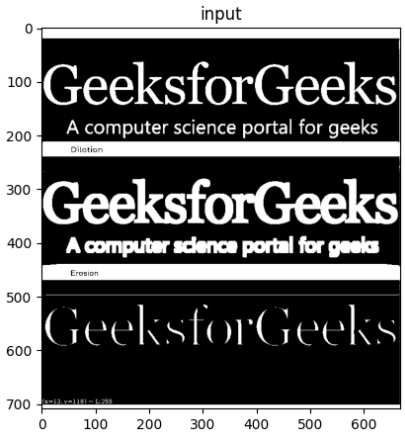
```
427180
```

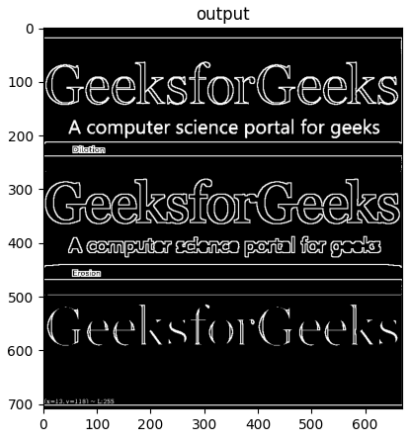
input

output

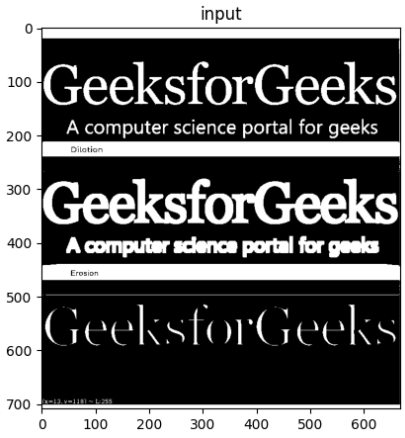


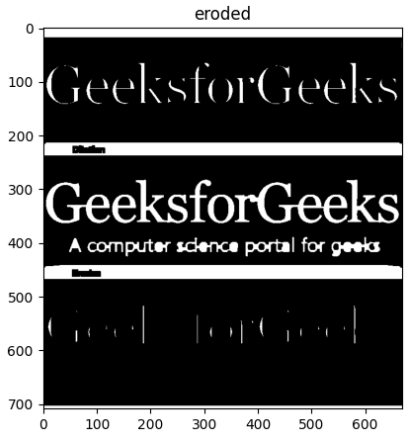
```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
image = cv.imread("ErosionDilation.jpg",0)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        if image[i][j]>200:
            image[i][j]=0
        else:
            image[i][j]=1
kernel = np.ones((5,5),np.uint8)
erode = cv.erode(image,kernel=kernel,iterations=1)
boundary_extraction = image-erode
plt.figure(figsize=(20,5))
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(boundary_extraction,cmap="gray")
plt.subplot(1,2,1)
plt.imshow(image,cmap="gray")
plt.title("input")
plt.show()
```



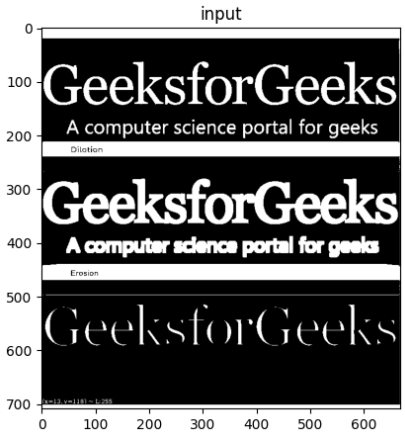


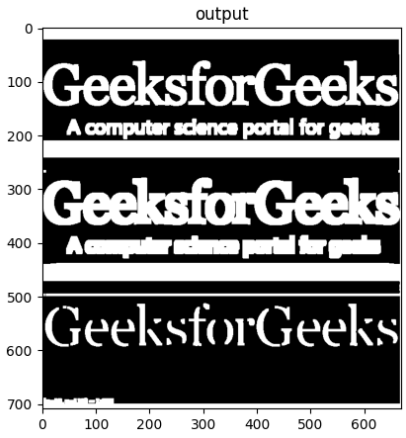
```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
image = cv.imread("ErosionDilation.jpg",0)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        if image[i][j]>200:
            image[i][j]=0
        else:
            image[i][j]=1
kernel = np.ones((5,5),np.uint8)
erode = cv.erode(image,kernel=kernel,iterations=1)
plt.figure(figsize=(20,5))
plt.subplot(1,2,2)
plt.title("eroded")
plt.imshow(erode,cmap="gray")
plt.subplot(1,2,1)
plt.imshow(image,cmap="gray")
plt.title("input")
plt.show()
```





```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
image = cv.imread("ErosionDilation.jpg",0)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        if image[i][j]>200:
            image[i][j]=0
        else:
            image[i][j]=1
kernel = np.ones((5,5),np.uint8)
erode = cv.dilate(image,kernel=kernel,iterations=1)
plt.figure(figsize=(20,5))
plt.subplot(1,2,2)
plt.title("output")
plt.imshow(erode,cmap="gray")
plt.subplot(1,2,1)
plt.imshow(image,cmap="gray")
plt.title("input")
plt.show()
```





```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
image = cv.imread("ErosionDilation.jpg",0)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        if image[i][j]>200:
            image[i][j]=0
        else:
            image[i][j]=1
kernel = np.ones((5,5),np.uint8)
erode = cv.erode(image,kernel=kernel,iterations=1)
dilate = cv.dilate(image,kernel=kernel,iterations=1)
open = cv.dilate(erode,kernel=kernel,iterations=1)
plt.figure(figsize=(20,5))
plt.subplot(1,2,2)
plt.title("output")
plt.imshow(open,cmap="gray")
plt.axis(False)
plt.subplot(1,2,1)
plt.imshow(image,cmap="gray")
plt.title("input")
plt.axis(False)
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
image = cv.imread("ErosionDilation.jpg",0)
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        if image[i][j]>200:
            image[i][j]=0
        else:
            image[i][j]=1
kernel = np.ones((5,5),np.uint8)
erode = cv.erode(image,kernel=kernel,iterations=1)
dilate = cv.dilate(image,kernel=kernel,iterations=1)

open = cv.erode(dilate,kernel=kernel,iterations=1)
plt.figure(figsize=(20,5))
plt.subplot(1,2,2)
plt.title('output')
plt.imshow(open,cmap="gray")
plt.axis(False)
plt.subplot(1,2,1)
plt.imshow(image,cmap="gray")
plt.title("input")
plt.axis(False)
plt.show()
```

```
# https://youtu.be/cDP_4VbC_sE
"""
Locating objects in large images using template matching

Need a source image and a template image.
The template image T is slided over the source image (as in 2D
convolution),
and the program tries to find matches using statistics.
Several comparison methods are implemented in OpenCV.
It returns a grayscale image, where each pixel denotes how much does
the
neighbourhood of that pixel match with template.

Once you get the result, you can use cv2.minMaxLoc() function
to find where is the maximum/minimum value.
Take it as the top-left corner of the rectangle and take (w,h) as
width and height of the rectangle.
That rectangle can be drawn on the region of matched template.

If the template image is larger than its size in the large image, we
can perfrom
the same exercise by resizing the template image to multiple sizes.
We can then extract the match with best score.
"""
### Template matching, single object in an image.


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

img_rgb = cv2.imread('main.png',0)  #Large image
plt.figure(figsize=(20,10))
plt.subplot(1,3,1)
plt.imshow(img_rgb,cmap='gray')
plt.axis('off')

template = cv2.imread('template.png', 0)  #Small image (template)
h, w = template.shape[::]

#methods available: ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED',
'cv2.TM_CCORR',
#            'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF',
'cv2.TM_SQDIFF_NORMED']

res = cv2.matchTemplate(img_rgb, template, cv2.TM_SQDIFF)
# For TM_SQDIFF, Good match yields minimum value; bad match yields
large values
# For all others it is exactly opposite, max value = good fit.
plt.subplot(1,3,2)
```

```
plt.imshow(res,cmap='gray')
plt.axis('off')

min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

top_left = min_loc  #Change to max_loc for all except for TM_SQDIFF
bottom_right = (top_left[0] + w, top_left[1] + h)

cv2.rectangle(img_rgb, top_left, bottom_right, (0, 0, 0), 2)  #Red
rectangle with thickness 2.
plt.subplot(1,3,3)
cv2.imwrite('matched.jpg', img_rgb)
plt.imshow(img_rgb,cmap='gray')
plt.axis('off')
plt.show()
```