

IMAGE PROCESSING

PRACTICAL FILE



2016UIT4019

AMIT KUMAR

IT-2

INDEX

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Name** | **Date** |
| 1. | Split image into the red, blue and green channels, and use them to convert the image into a grayscale image. | 03-01-2019 |
| 2. | Demonstrate different methods to measure the distance in an image. | 10-01-2019 |
| 3. | Illustrate the use of steganography by hiding data in an image and extracting it back. | 10-01-2019 |
| 4. | Show image connectivity with the different type of adjacencies. | 17-01-2019 |
| 5. | Implement component labeling algorithm. | 24-01-2019 |
| 6. | Show the zooming and shrinking of an image. | 31-01-2019 |
| 7. | Perform contrast stretching on an image. | 07-02-2019 |
| 8. | Perform an image transformation to create a negative image. | 14-02-2019 |
| 9. | Implement the low pass and high pass filter. | 21-02-2019 |
| 10. | Implement the Gaussian filter on a noisy image. | 07-03-2019 |
| 11. | Perform edge detection on an image and find angle as well as the direction of the edges. | 28-03-2019 |
| 12. | Demonstrate the histogram equalization method on an image. | 04-04-2019 |
| 13. | Perform the histogram specification between two images. | 04-04-2019 |
| 14. | Show the fundamental morphological operations - Dilation and Erosion. | 11-04-2019 |
| 15. | Perform the morphological operations - Opening and Closing, using the fundamental morphological operations. | 11-04-2019 |

1. **Split image into color channels, and convert the image into a grayscale image.**

Main.py >

import numpy as np

import cv2

import matplotlib.pyplot as plt

img = cv2.imread('images/cups.jpg')

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

r,g,b = img[:,:,2], img[:,:,1], img[:,:,0]

plt.figure("channels",figsize=(20,5))

plt.subplot(131)

plt.imshow(r,cmap='Reds')

plt.colorbar()

plt.subplot(132)

plt.imshow(r,cmap='Greens')

plt.colorbar()

plt.subplot(133)

plt.imshow(r,cmap='Blues')

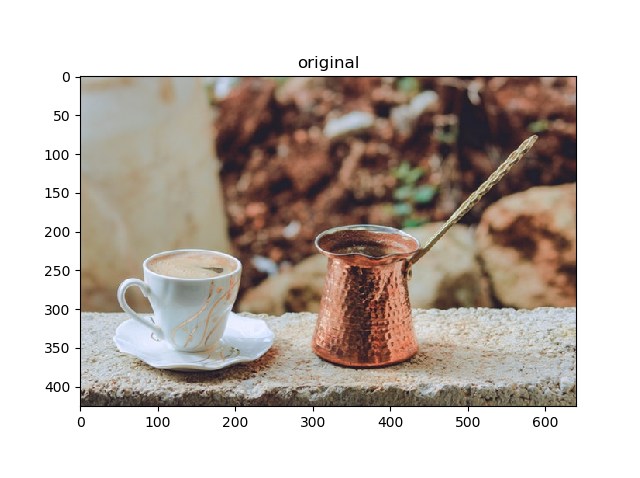
plt.colorbar()

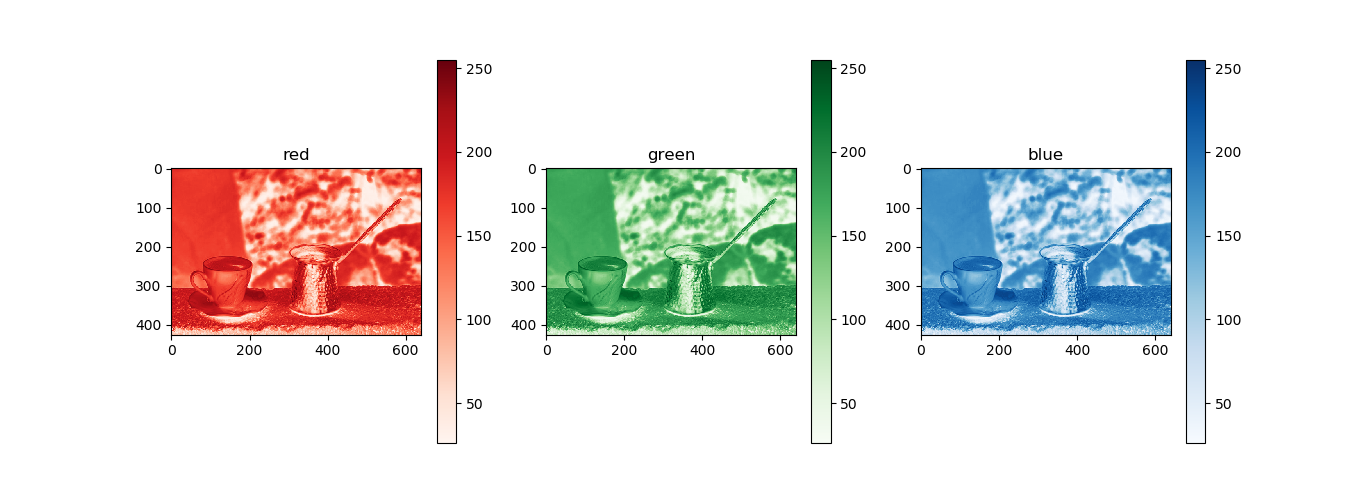
grey = val = 0.2989\*r + 0.5870\*g + 0.1140\*b

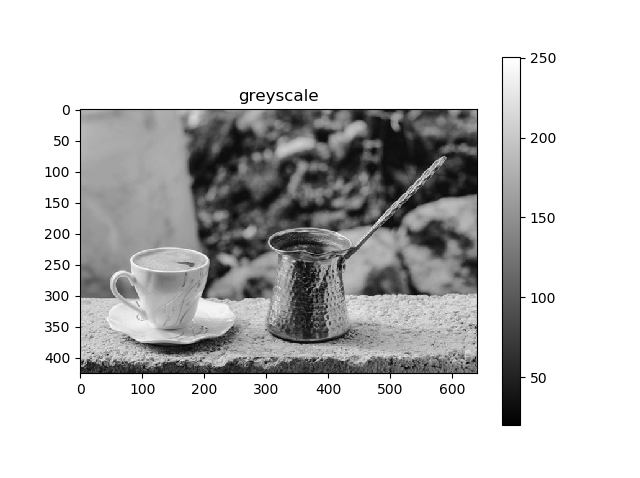
plt.imshow(grey,cmap='gray')

plt.colorbar()

Output:







1. **Demonstrate different methods to measure the distance in an image.**

Main.py >

px, py = map(int, input("enter coord of p: ").split())  
qx, qy = map(int, input("enter coord of q: ").split())  
  
print("eucledian distance between 2 pixels is:",((px-qx)\*\*2 + (py-qy)\*\*2)\*\*(0.5))  
print("manhattan distance between 2 pixels is:",abs(px-qx) + abs(py-qy))  
print("chess-board distance between 2 pixels is:",max(abs(px-qx),abs(py-qy)))

Output:

enter coord of p: 2 4

enter coord of q: 5 6

eucledian distance between 2 pixels is: 3.605551275463989

manhattan distance between 2 pixels is: 5

chess-board distance between 2 pixels is: 3

1. **Illustrate the use of steganography by hiding data in an image and extracting it back.**

Main.py>

import numpy as np

import cv2

import matplotlib.pyplot as plt

def encode(img,data,filename):

stream = img.ravel()

if len(stream) < len(data)\*9:

print("image is not big enough")

return

j=0

for i in range(0,len(stream),9):

if j >= len(data):

break

for k in range(8):

stream[i+k] &= 254

stream[i+k] |= (( (1<<k) & ord(data[j]) ) > 0)

stream[i+8] &= 254

stream[i+8] |= (j < len(data)-1)

j += 1

imgn = np.reshape(stream,img.shape)

print(stream[:(len(data)+1)\*9])

cv2.imwrite(filename.split('.')[0]+"\_encoded.png",imgn)

return imgn

def decode(img):

data = ''

stream = img.ravel()

print(stream[:27])

for i in range(0, len(stream), 9):

val = 0

for k in range(8):

val |= ( (1<<k)\*(stream[i+k]&1 > 0) )

print(val)

data += chr(val)

if stream[i+8]&1 == 0:

break

return data

path = 'images/cups.jpg'

img = cv2.imread(path)

plt.subplot(121)

plt.title("original image")

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

imgn=encode(img,"amit kumar",path)

plt.subplot(122)

plt.title("encoded image")

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

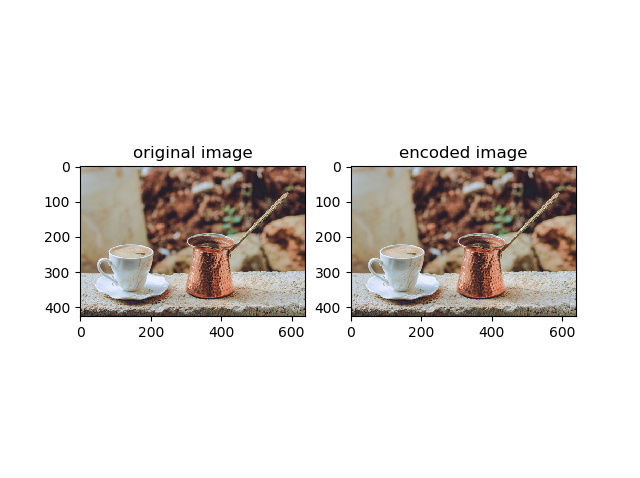
plt.show()

d\_path = 'images/cups\_encoded.png'

img\_d = cv2.imread(d\_path)

print(f"extracted data = '{decode(img\_d)}'")

Output:



extracted data = 'amit kumar'

1. **Show image connectivity with the different type of adjacencies**.

Main.py>

m, n = map(int, input("enter size of matrix: ").split())  
  
px, py = map(int, input("enter coord of p: ").split())  
qx, qy = map(int, input("enter coord of q: ").split())  
  
  
def is\_n8(px, py, qx, qy):  
 if (px - qx) \*\* 2 + (py - qy) \*\* 2 == 1 or (px - qx) \*\* 2 == 1 and (py - qy) \*\* 2 == 1:  
 return True  
 else:  
 return False  
  
  
def is\_n4(px, py, qx, qy):  
 if (px - qx) \*\* 2 + (py - qy) \*\* 2 == 1:  
 return True  
 else:  
 return False  
  
  
def is\_m(px, py, qx, qy):  
 if is\_n4(px, py, qx, qy) or ((px - qx) \*\* 2 == (py - qy) \*\* 2 and not is\_n8(px, py, qx, qy)):  
 return True  
 else:  
 return False  
  
  
if not (px >= 0 and px < m and qx >= 0 and qx < m and py >= 0 and py < n and qy >= 0 and qy < n):  
 print("coordinate out of range")  
 exit(0)  
  
print("is p and q are n4 adjancency", is\_n4(px, py, qx, qy))  
print("is p and q are n8 adjancency", is\_n8(px, py, qx, qy))  
print("is p and q are m adjancency", is\_m(px, py, qx, qy))

Output:

enter size of matrix: 10 10

enter coord of p: 2 4

enter coord of q: 3 4

is p and q are n4 adjancency True

is p and q are n8 adjancency True

is p and q are m adjancency True

1. **Implement component labeling algorithm.**

Main.py >

from PIL import Image

import matplotlib.pyplot as plt

%matplotlib qt

path = "images/objects\_small.bmp"

img = Image.open(path)

print("image size is:", img.size)

mat = []

for i in range(img.size[0]):

row = []

for j in range(img.size[1]):

if img.getpixel((j, i)) == 0:

row.append(1)

else:

row.append(0)

mat.append(row)

for r in mat:

print(r)

# plt.set\_cmap("gray")

plt.imshow(mat,cmap="gray")

plt.colorbar()

plt.title("objects")

plt.show()

print("-----------------First Pass-----------------")

val = 1

pairs = []

for i in range(len(mat)):

for j in range(len(mat[0])):

if mat[i][j]:

if i > 0 and mat[i - 1][j]:

mat[i][j] = mat[i - 1][j]

if j > 0 and mat[i][j - 1] and mat[i][j - 1] != mat[i][j]:

pairs.append((mat[i][j - 1], mat[i][j]))

elif j > 0 and mat[i][j - 1]:

mat[i][j] = mat[i][j - 1]

else:

mat[i][j] = val

val += 1

else:

mat[i][j] = 0

for r in mat:

print(r)

plt.imshow(mat)

plt.colorbar()

plt.title("connected components")

plt.show()

print("---------------Pairs-------------------")

print(pairs)

eq\_classes = {}

for i in range(val):

eq\_classes[i] = {i}

queue = [i]

state = [False for j in range(val)]

state[i] = True

while len(queue) > 0:

pop\_el = queue.pop(0)

for p in pairs:

if pop\_el == p[0]:

if state[p[1]] is False:

queue.append(p[1])

state[p[1]] = True

eq\_classes[i].add(p[1])

elif pop\_el == p[1]:

if state[p[0]] is False:

queue.append(p[0])

state[p[0]] = True

eq\_classes[i].add(p[0])

print("---------------Equivalent classes-------------------")

for k, v in eq\_classes.items():

print(k, ">", v)

for i in range(len(mat)):

for j in range(len(mat[0])):

if mat[i][j]:

mat[i][j] = sorted(eq\_classes[mat[i][j]])[0]

print("---------------objects-------------------")

for r in mat:

print(r)

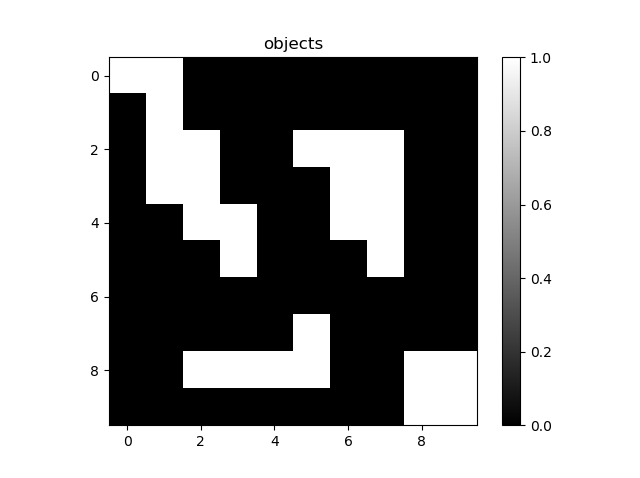
plt.imshow(mat)

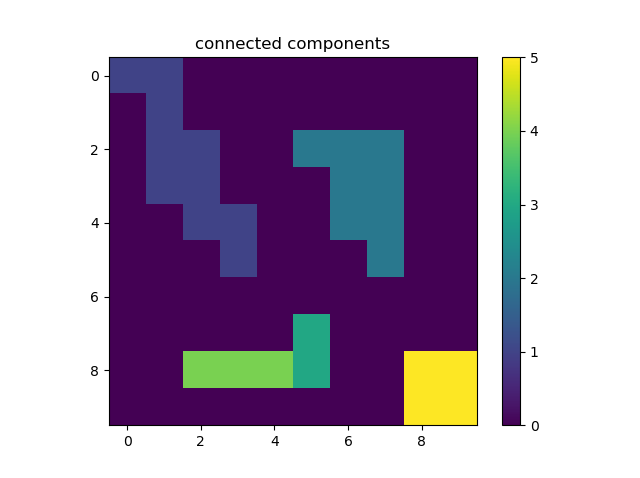
plt.colorbar()

plt.title("connected components")

plt.show()

Output:





---------------Pairs-------------------

[(4, 3)]

---------------Equivalent classes-------------------

0 > {0}

1 > {1}

2 > {2}

3 > {3, 4}

4 > {3, 4}

5 > {5}

---------------objects-------------------

[1, 1, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 1, 1, 0, 0, 2, 2, 2, 0, 0]

[0, 1, 1, 0, 0, 0, 2, 2, 0, 0]

[0, 0, 1, 1, 0, 0, 2, 2, 0, 0]

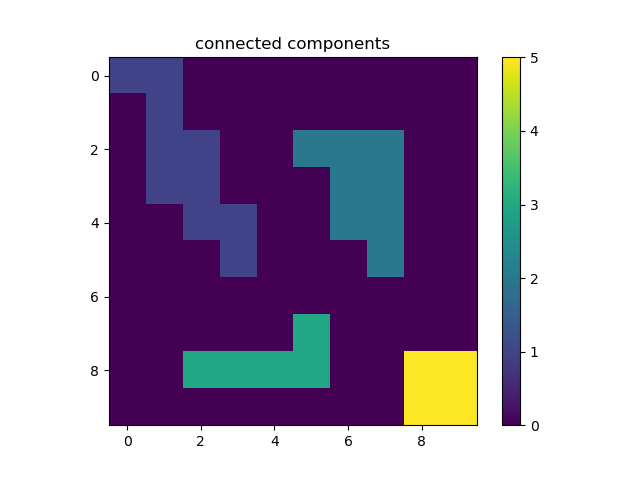
[0, 0, 0, 1, 0, 0, 0, 2, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 3, 0, 0, 0, 0]

[0, 0, 3, 3, 3, 3, 0, 0, 5, 5]

[0, 0, 0, 0, 0, 0, 0, 0, 5, 5]



1. **Show the zooming and shrinking of an image.**

Main.py >

import cv2

import matplotlib.pyplot as plt

import numpy as np

import math

%matplotlib qt

def calc\_sizes(img, ratio\_x, ratio\_y):

size = img.shape

n\_size = ((img.shape[0] - 1) \* ratio\_y + 1, (img.shape[1] - 1) \* ratio\_x + 1)

return size, n\_size

def nearest\_zoom(img, ratio\_x, ratio\_y):

size, n\_size = img.shape, (int(img.shape[0]\*ratio\_y), int(img.shape[1]\*ratio\_x))

n\_img = np.zeros(n\_size)

for i in range(n\_size[0]):

for j in range(n\_size[1]):

n\_img[i][j] = img[int(i / ratio\_y)][int(j / ratio\_x)]

return n\_img

def bilinear\_zoom(img, ratio\_x, ratio\_y):

size, n\_size = calc\_sizes(img, ratio\_x, ratio\_y)

b\_img = np.zeros(n\_size)

temp = np.zeros((size[0], n\_size[1]))

for i in range(size[0]):

for j in range(n\_size[1]):

temp[i][j] = (int(img[i][math.floor(j / ratio\_x)]) + int(img[i][math.ceil(j / ratio\_x)])) / 2

for i in range(n\_size[0]):

for j in range(n\_size[1]):

b\_img[i][j] = (int(temp[math.floor(i / ratio\_y)][j]) + int(temp[math.ceil(i / ratio\_y)][j])) / 2

return b\_img

def k\_zoom(img, ratio\_x, ratio\_y):

size, n\_size = calc\_sizes(img, ratio\_x, ratio\_y)

k\_img = np.zeros(n\_size)

temp = np.zeros((size[0], n\_size[1]))

for i in range(size[0]):

count = 0

op = 0

for j in range(n\_size[1]):

if count == 0:

temp[i][j] = img[i][int(j / ratio\_x)]

if j < n\_size[1] - 1:

op = (int(img[i][int(j / ratio\_x) + 1]) - int(img[i][int(j / ratio\_x)])) / ratio\_x

else:

temp[i][j] = int(temp[i][j - 1] + op)

count = (count + 1) % ratio\_x

for j in range(n\_size[1]):

count = 0

op = 0

for i in range(n\_size[0]):

if count == 0:

k\_img[i][j] = temp[int(i/ratio\_y)][j]

if i < n\_size[0] -1:

op = (temp[int(i/ratio\_y)+1][j] - temp[int(i/ratio\_y)][j])/ratio\_y

else:

k\_img[i][j] = int(k\_img[i-1][j] + op)

count = (count+1)%ratio\_y

# k\_img = temp

return k\_img

path = "images/daisy.jpg"

img = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

# img = np.array([[69,50,80],[45,60,66],[30,55,80]])

# img = np.array([[15,30,15],[30,15,30]])

# img = np.arange(100).reshape((10,10))

# print(img)

plt.figure(1,figsize=(5,5))

# plt.subplot(321)

plt.title("original")

plt.grid(True, color="gray")

plt.imshow(img, cmap="gray")

#

ratio\_y, ratio\_x = 3,3

#

n\_img = nearest\_zoom(img, ratio\_x, ratio\_y)

# print(n\_img)

plt.figure(2,figsize=(17,10))

plt.subplot(131)

plt.title("nearest neighbour")

plt.grid(True, color="gray", )

plt.imshow(n\_img, cmap="gray")

#

b\_img = bilinear\_zoom(img, ratio\_x, ratio\_y)

# print(b\_img)

plt.subplot(132)

plt.title("bilinear interpolation")

plt.grid(True, color="gray")

plt.imshow(b\_img, cmap="gray")

k\_img = k\_zoom(img, ratio\_x, ratio\_y)

# print(k\_img)

plt.subplot(133)

plt.title("k-zooming")

plt.grid(True, color="gray")

plt.imshow(k\_img, cmap="gray")

plt.show()

img2 = np.arange(400).reshape((20,20))

plt.figure(3,figsize=(15,10))

plt.subplot(121)

plt.title("original")

plt.grid(True, color="gray")

plt.imshow(img, cmap="gray")

r\_img = nearest\_zoom(img,1/2,1/2)

# print(r\_img)

plt.subplot(122)

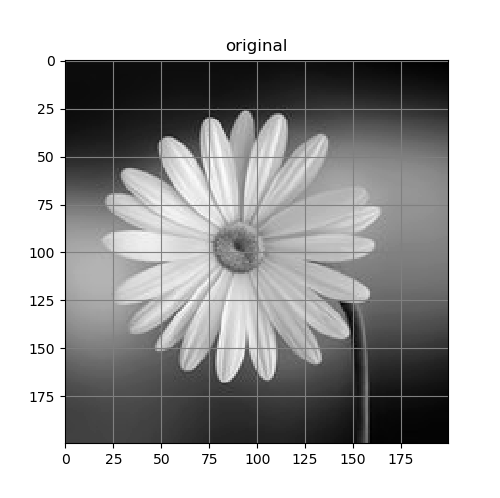
plt.title("reduced")

plt.grid(True, color="gray")

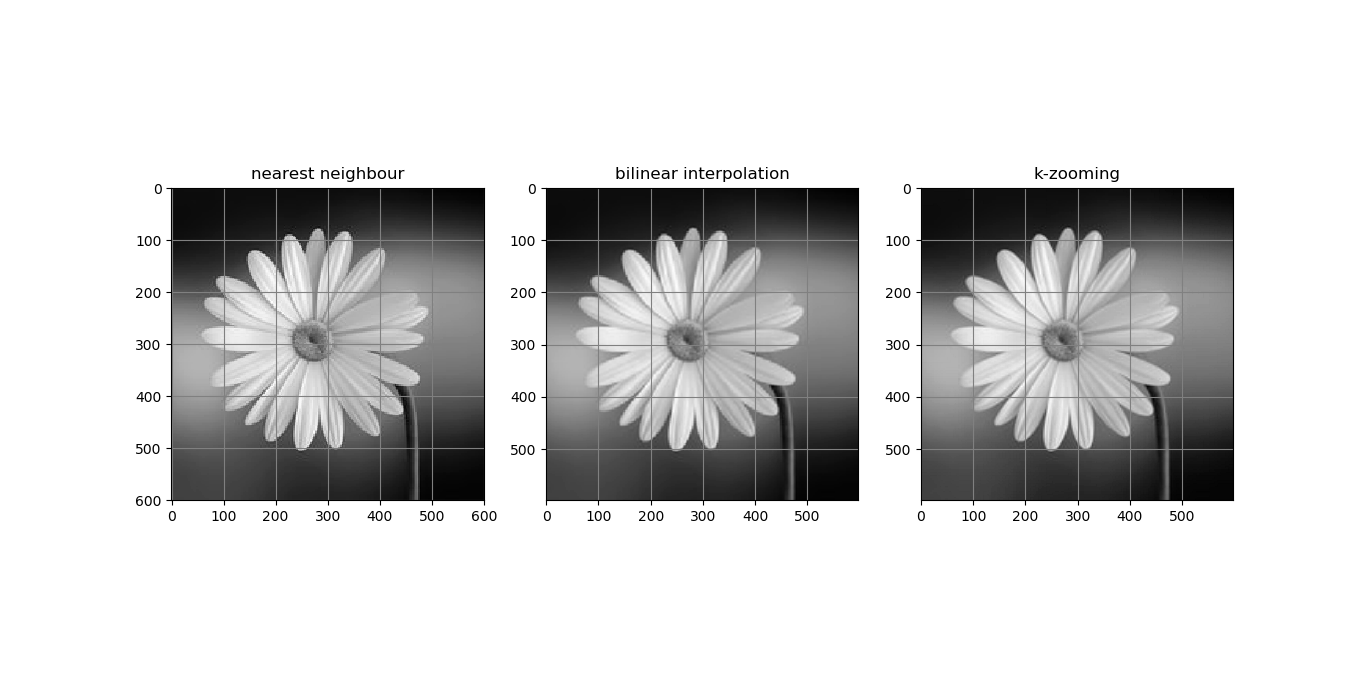
plt.imshow(r\_img, cmap="gray")

plt.show()

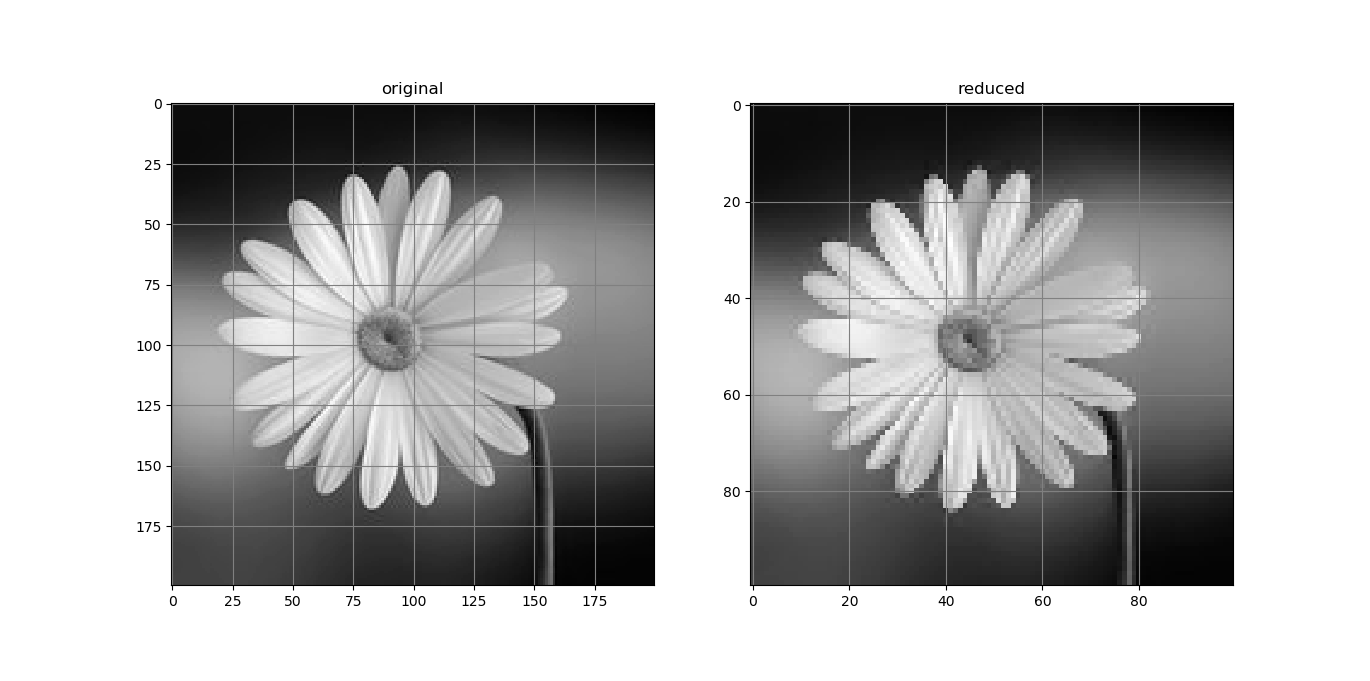
Output:



zooming



shrinking



1. **Perform contrast stretching on an image.**

Main.py >

import numpy as np

import cv2

import matplotlib.pyplot as plt

%matplotlib qt

path = "images/sat\_image.jpg"

img = cv2.imread(path,cv2.IMREAD\_GRAYSCALE)

# img = np.random.randint(50,200,(10,10))

# s1,s2 = 0, 255

# r1,r2 = np.min(img),np.max(img)

#

# img = s1 + ((s2-s1)/(r2-r1))\*(img-r1)

# print(np.min(img),np.max(img))

r1, r2, = 130, 200

s1, s2 = 200, 250

plt.figure(1,figsize=(5,5))

plt.grid(True)

plt.plot([0,r1,r2,255],[0,s1,s2,255])

plt.figure(2,figsize=(15,10))

plt.subplot(121)

plt.title("original")

plt.imshow(img,cmap="gray")

alpha = s1/r1

beta = (s2-s1)/(r2-r1)

gamma = (255-s2)/(255-r2)

for i in range(img.shape[0]):

for j in range(img.shape[1]):

if img[i][j] < r1:

img[i][j] = alpha\*img[i][j]

elif img[i][j] < r2:

img[i][j] = beta\*(img[i][j] - r1) + s1

else:

img[i][j] = gamma\*(img[i][j] - r2) + s2

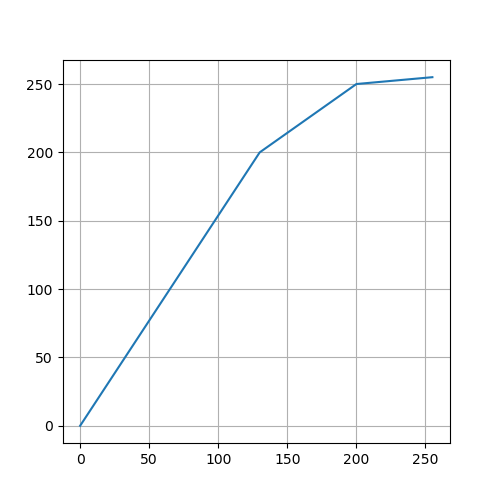
plt.subplot(122)

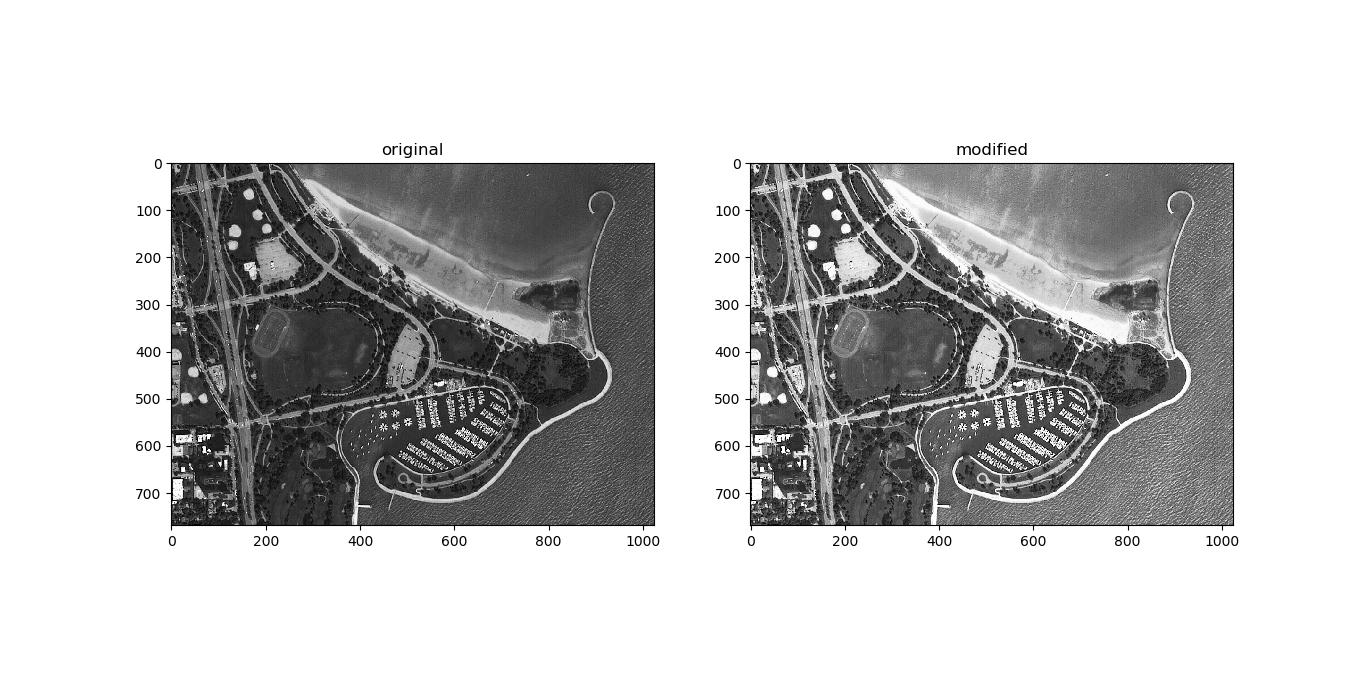
plt.title("modified")

plt.imshow(img,cmap="gray")

plt.show()

Output:





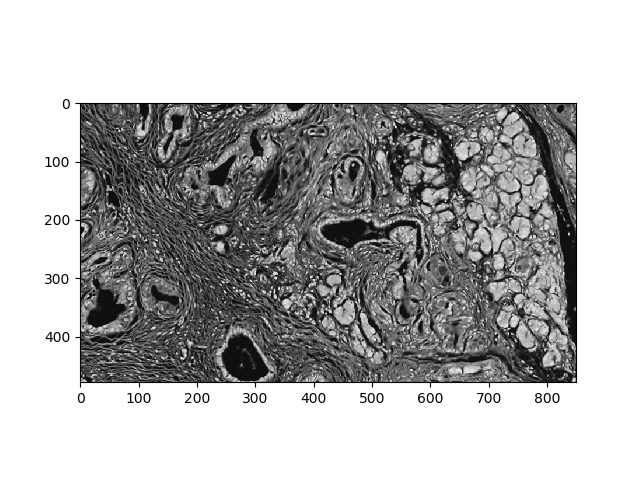
1. **Perform an image transformation to create a negative image.**

Main.py >

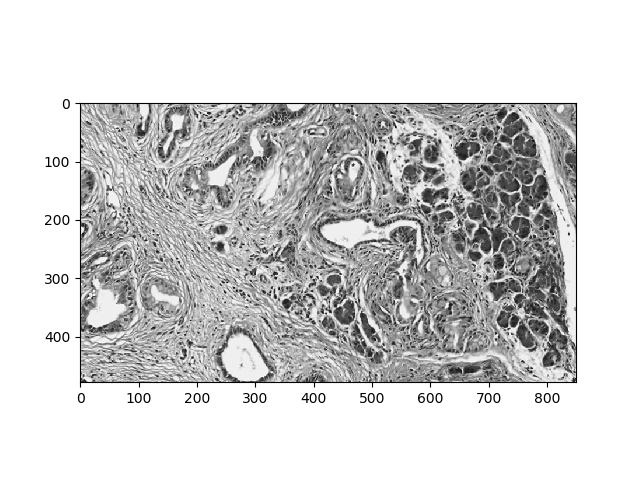
import cv2  
import numpy  
import matplotlib.pyplot as plt  
  
path = "images/pancreaticcancer.jpg"  
img = cv2.imread(path,cv2.IMREAD\_GRAYSCALE)  
img = 255 - img  
plt.figure(1)  
plt.imshow(img,cmap="gray")  
  
img = 255 - img  
plt.figure(2)  
plt.imshow(img,cmap="gray")  
  
plt.show()

Output:

Original image



Negative image



1. **Implement the low pass and high pass filter.**

Main.py >

import numpy as np

import cv2

from apply\_kernel import \*

import matplotlib.pyplot as plt

%matplotlib qt

# t\_img = np.random.randint(0,256,(20,30),dtype='uint8')

t\_img = cv2.imread('images/space\_view.jpg',cv2.IMREAD\_GRAYSCALE)

# t\_img = np.ones((10,10),dtype='uint8')

kernel = np.ones((7,7))/49

# kernel = np.array([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])

r\_img = apply\_kernel(t\_img,kernel)

plt.figure("low pass filter")

plt.subplot(121)

plt.imshow(t\_img,cmap='gray')

plt.title("original")

plt.subplot(122)

plt.imshow(r\_img,cmap='gray')

plt.title("blurred")

plt.show()

t\_img = cv2.imread('images/moon.jpg',cv2.IMREAD\_GRAYSCALE)

# t\_img = np.ones((10,10),dtype='uint8')

kernel = np.ones((3,3))\*-1

kernel[1,1] = 8

kernel /= 9

# kernel = np.array([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])

r\_img = apply\_kernel(t\_img,kernel)

plt.figure("high pass filter")

plt.subplot(131)

plt.imshow(t\_img,cmap='gray')

plt.title("original")

plt.subplot(132)

plt.imshow(r\_img,cmap='gray')

plt.title("edges")

plt.subplot(133)

sharped\_img = np.array(t\_img.astype(int)+r\_img.astype(int),dtype='int')

sharped\_img[sharped\_img >255] = 255

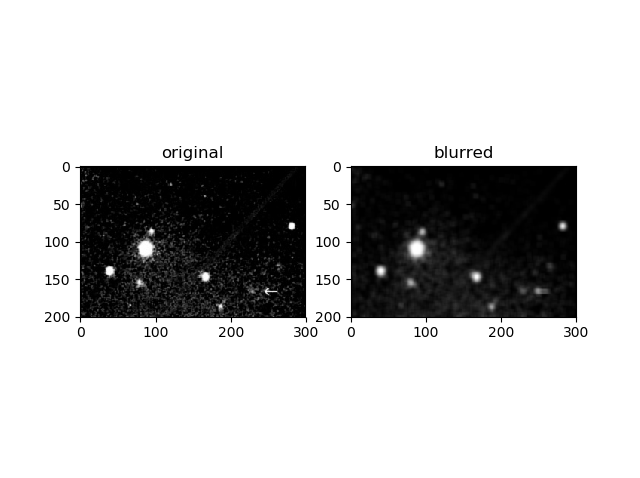
plt.imshow(sharped\_img,cmap='gray')

plt.title("sharped image")

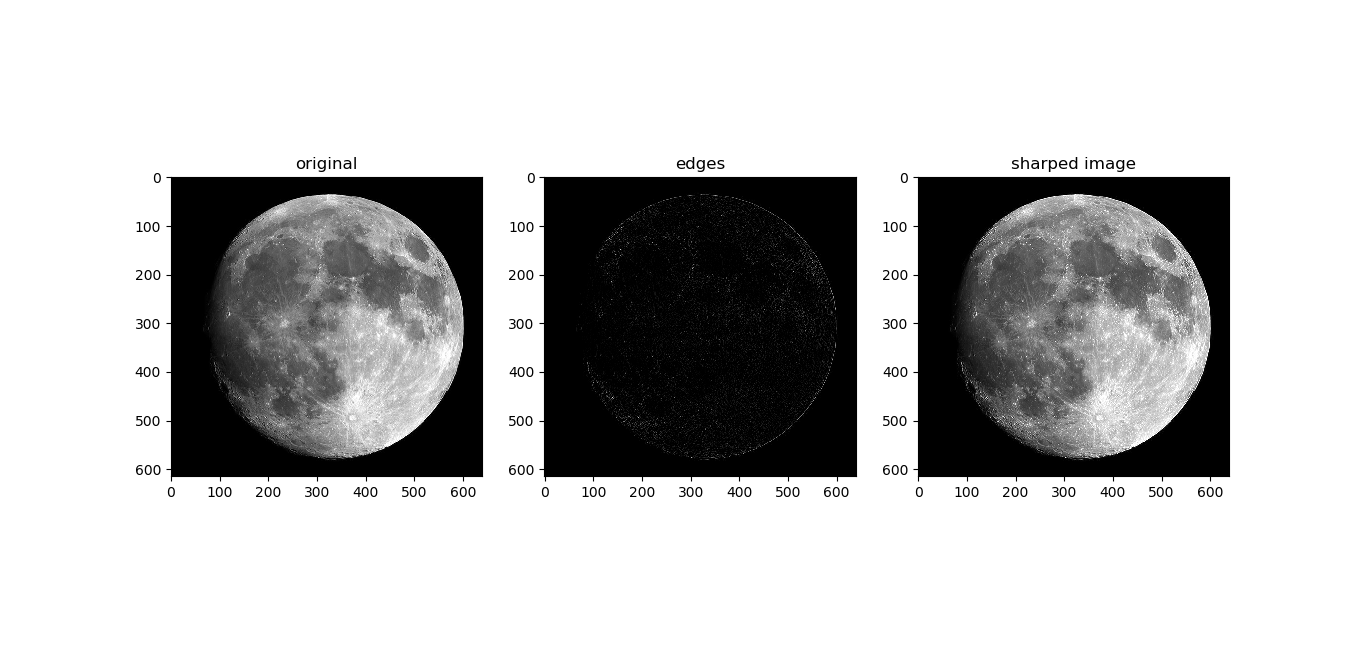
plt.show()

Output:

Low pass



High pass



1. **Implement the Gaussian filter on a noisy image.**

Main.py >

import numpy as np

import cv2

import matplotlib.pyplot as plt

from apply\_kernel import \*

%matplotlib qt

def get\_gaussian\_kernel(n,sigma2):

x, y = np.meshgrid(np.arange(-(n//2),n//2+1),np.arange(-(n//2),n//2+1))

kernel = np.exp(-(x\*\*2 + y\*\*2)/(2\*sigma2))

kernel \*= np.ceil(1/kernel[0,0])

kernel = kernel.astype(int)

# print(kernel)

return kernel/np.sum(kernel)

img = cv2.imread('images/daisy.jpg',cv2.IMREAD\_GRAYSCALE)

plt.figure("gaussian")

plt.subplot(121)

plt.imshow(img,cmap='gray')

plt.title("original")

kernel = get\_gaussian\_kernel(5,2)

r\_img = apply\_kernel(img, kernel)

plt.subplot(122)

plt.imshow(r\_img, cmap='gray')

plt.title("gaussian blurred")

plt.show()

Output:



1. **Perform edge detection on an image and find angle as well as the direction of the edges**

Main.py >

import numpy as np

import cv2

from apply\_kernel import \*

import matplotlib.pyplot as plt

%matplotlib qt

precuitt\_kernel\_x = np.array([[-1,0,1],

[-1,0,1],

[-1,0,1]])

precuitt\_kernel\_y = np.array([[-1,-1,-1],

[0,0,0],

[1,1,1]])

sobel\_kernel\_x = np.array([[-1,0,1],

[-2,0,2],

[-1,0,1]])

sobel\_kernel\_y = np.array([[-1,-2,-1],

[0,0,0],

[1,2,1]])

# img = np.array([[0,0,64],

# [0,64,0],

# [64,0,0]])

img = cv2.imread('images/duck2.jpg',cv2.IMREAD\_GRAYSCALE)

h\_pass\_k = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])/9

img = apply\_kernel(img, h\_pass\_k)

plt.subplot(121)

plt.imshow(img,cmap='gray')

gx = apply\_kernel(img,sobel\_kernel\_x,padding=True,allow\_neg=True)

gy = apply\_kernel(img,sobel\_kernel\_y,padding=True,allow\_neg=True)

magnitude = np.abs(gx) + np.abs(gy)

direction = 180\*np.arctan(gy/gx)/np.pi

# print(img)

print(gx)

# print(img)

print(gy)

print(magnitude)

print(direction)

plt.subplot(122)

plt.imshow(direction)

plt.colorbar()

plt.figure("magnitude")

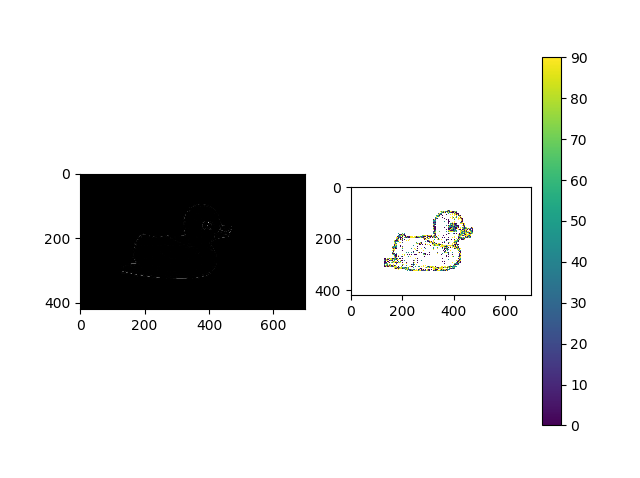
plt.imshow(magnitude,cmap='gray')

plt.title("magnitude")

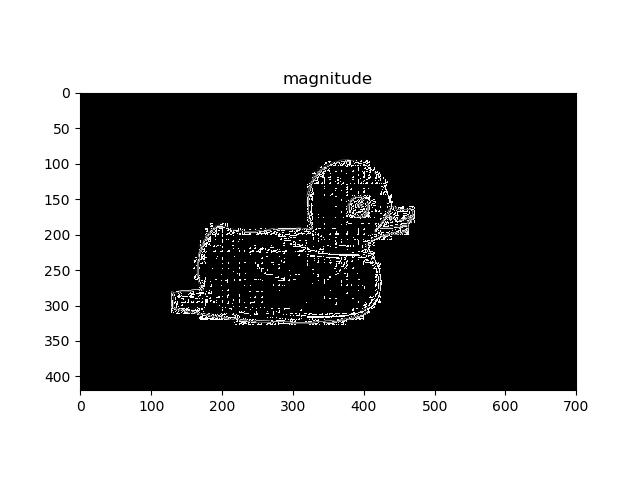
plt.show()

OUTPUT:

edges and direction



Magnitude



1. **Demonstrate the histogram equalization method on an image.**

Main.py >

import numpy as np

import cv2

import matplotlib.pyplot as plt

%matplotlib qt

img = cv2.imread('images/board.jpg',cv2.IMREAD\_GRAYSCALE)

# print(img.shape)

# des\_img = cv2.imread('images/board\_desired.jpg',cv2.IMREAD\_GRAYSCALE)

bins\_count = 256

# # plt.imshow(img,cmap='gray')

plt.figure("histogram")

plt.subplot(121)

plt.title("original histogram")

freq, bins, patches = plt.hist(img.ravel(),bins=bins\_count,color='red')

total\_pixels = np.sum(freq)

c1 = freq / total\_pixels

prev = 0

for i in range(len(c1)):

c1[i] = prev = prev + c1[i]

c1 \*= bins[-1]

c1 = np.round(c1)

imgn = np.array(img,copy=True)

for i in range(bins\_count):

imgn[img == i] = c1[i]

plt.subplot(122)

plt.title("equalized histogram")

plt.bar(np.arange(0,bins\_count),c1)

plt.show()

plt.figure("image")

plt.subplot(121)

plt.title("original")

plt.imshow(img,cmap='gray')

plt.colorbar()

plt.subplot(122)

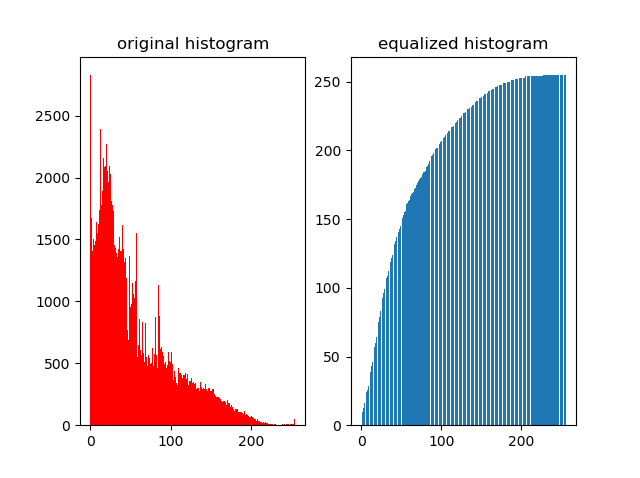
plt.title("equalized")

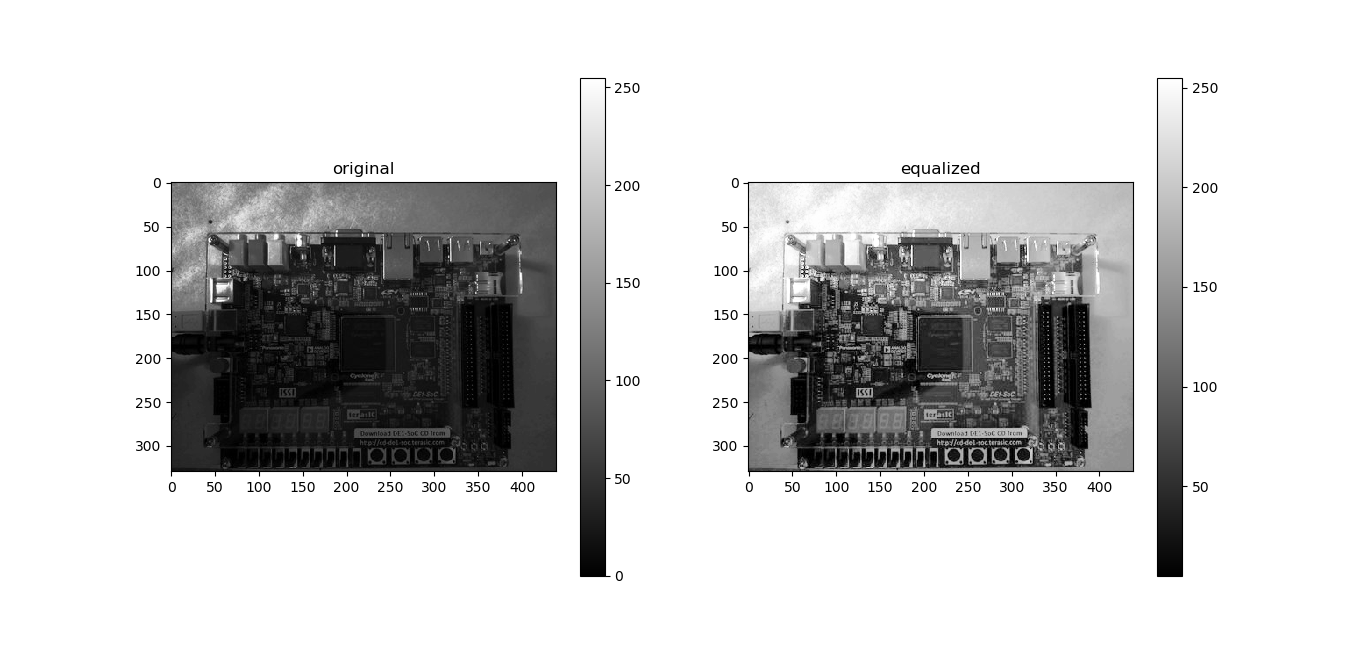
plt.imshow(imgn,cmap='gray')

plt.colorbar()

plt.show()

Output:





1. **Perform the histogram specification between two images**

Main.py >

import numpy as np

import cv2

import matplotlib.pyplot as plt

%matplotlib qt

img = cv2.imread('images/board.jpg',cv2.IMREAD\_GRAYSCALE)

des\_img = cv2.imread('images/board\_desired.jpg',cv2.IMREAD\_GRAYSCALE)

bins\_count = 256

# plt.imshow(img,cmap='gray')

plt.figure(0)

plt.subplot(131)

plt.title("original")

freq, bins, patches = plt.hist(img.ravel(),bins=bins\_count)

total\_pixels = np.sum(freq)

plt.subplot(132)

plt.title("desired")

dfreq, dbins, dpatches = plt.hist(des\_img.ravel(),bins=bins\_count)

c1 = freq/total\_pixels

prev = 0

for i in range(len(c1)):

c1[i] = prev = prev + c1[i]

c1 \*= bins[-1]

c1 = np.round(c1)

c2 = dfreq/total\_pixels

prev=0

for i in range(len(c2)):

c2[i] = prev = prev + c2[i]

c2 \*= bins[-1]

c2 = np.round(c2)

d = []

ic1, ic2 = 0, 0

while ic1 < len(c1):

while c1[ic1] > c2[ic2]:

ic2 += 1

d.append(bins[ic2])

ic1 += 1

d = np.array(d,dtype='int')

des\_hist = np.zeros(bins\_count)

ret\_img = np.array(img,copy=True)

for i in range(len(d)):

des\_hist[d[i]] += freq[i]

ret\_img[img == i] = d[i]

plt.subplot(133)

plt.title("retrived")

plt.bar(np.arange(0,256),des\_hist)

plt.figure(1)

plt.subplot(131)

plt.imshow(img,cmap='gray')

plt.title("original")

plt.subplot(132)

plt.imshow(des\_img,cmap='gray')

plt.title("desired")

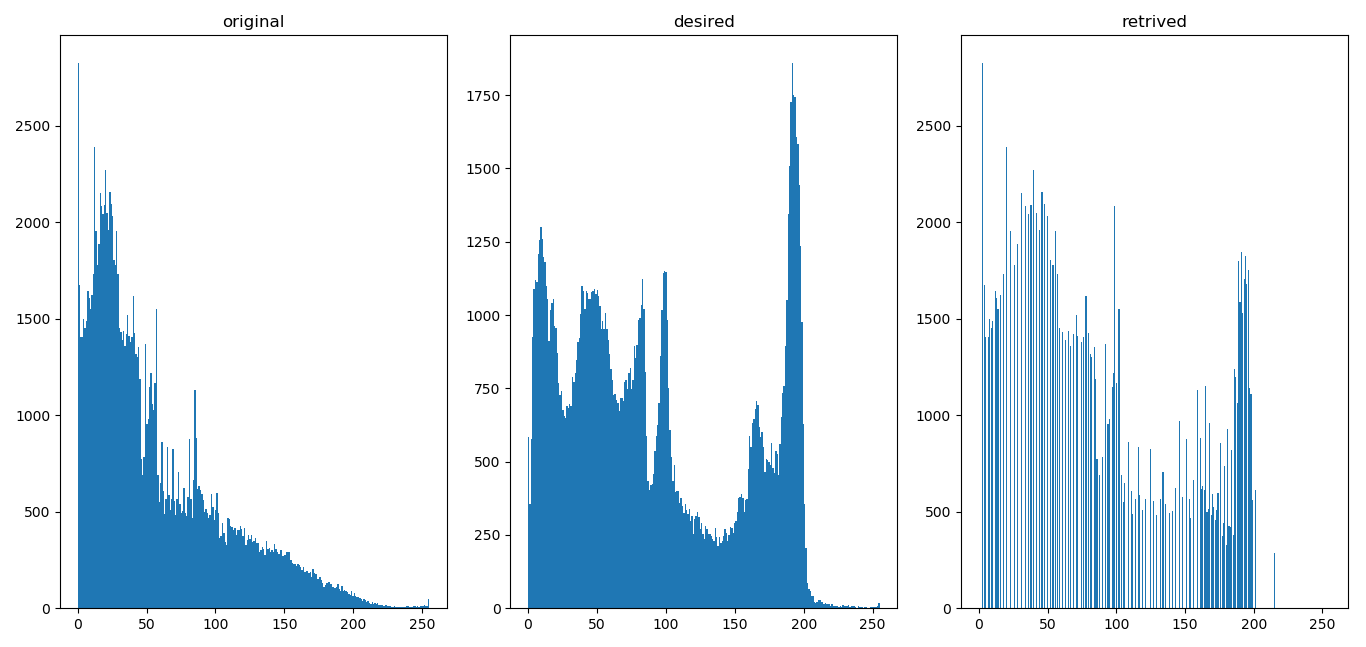
plt.subplot(133)

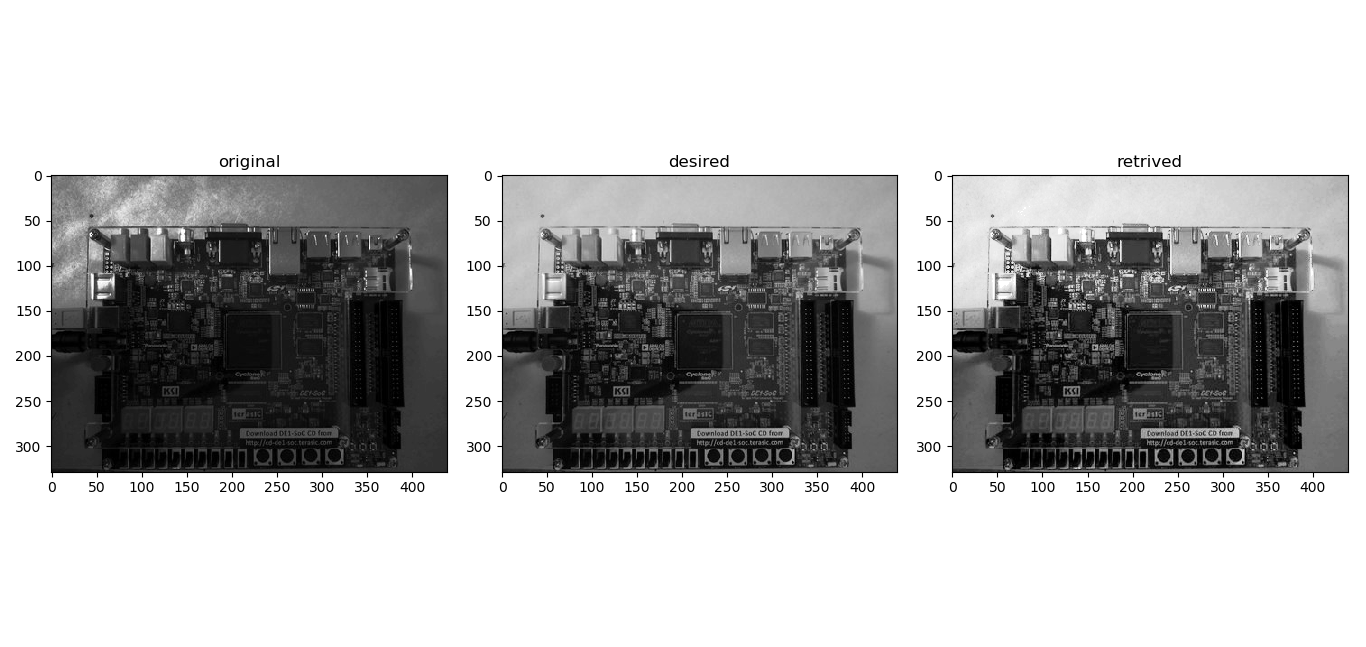
plt.imshow(ret\_img,cmap='gray')

plt.title("retrived")

plt.show()

Output:





1. **Show the fundamental morphological operations - Dilation and Erosion**.

Main.py >

import numpy as np

import cv2

from apply\_kernel import \*

import matplotlib.pyplot as plt

%matplotlib qt

def apply\_dilation(img,kernel,padding=True):

ksize = len(kernel)

pimg = img

if padding:

pimg = padd(img,ksize//2)

# print(pimg)

new\_img = np.array(pimg,copy=True)

for i in range(ksize//2,pimg.shape[0]-ksize//2):

for j in range(ksize//2,pimg.shape[1]-ksize//2):

# print(i,":",j)

if pimg[i,j] == kernel[ksize//2,ksize//2]:

new\_img[i-ksize//2:i+1+ksize//2,j-ksize//2:j+1+ksize//2] = kernel

return new\_img

def apply\_erosion(img,kernel):

ksize = len(kernel)

new\_img = np.array(img,copy=True)

for i in range(ksize//2,img.shape[0]-ksize//2):

for j in range(ksize//2,img.shape[1]-ksize//2):

if np.array\_equal(img[i-ksize//2:i+1+ksize//2,j-ksize//2:j+1+ksize//2], kernel):

new\_img[i,j] = 255

new\_img[new\_img != 255] = 0

new\_img[new\_img == 255] = 1

return new\_img

img = cv2.imread('images/shape2.bmp',cv2.IMREAD\_GRAYSCALE)

img[img==255]=1

plt.imshow(img,cmap='gray')

plt.colorbar()

plt.title("original")

plt.show()

kernel = np.ones((3,3))

d\_img = apply\_dilation(img, kernel)

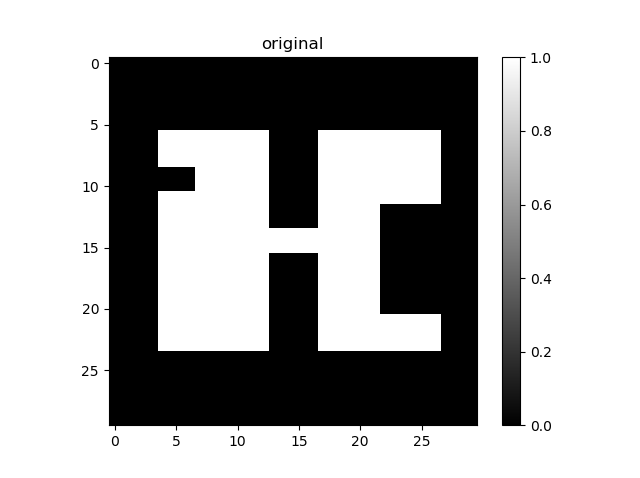
plt.imshow(d\_img,cmap='gray')

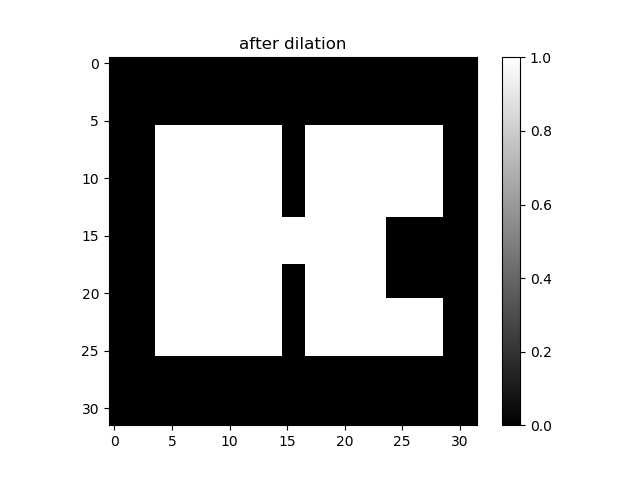
plt.colorbar()

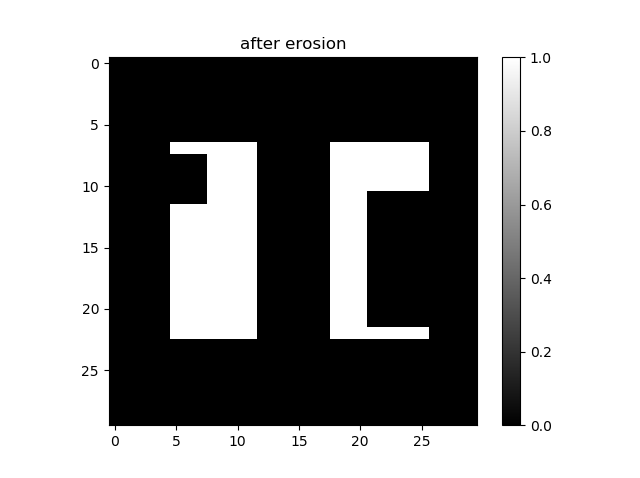
plt.title("after dilation")

plt.show()

Output:







1. **Perform the morphological operations - Opening and Closing, using the fundamental morphological operations.**

Main.py >

def apply\_opening(img, kernel):

erosed\_img = apply\_erosion(img, kernel)

dil\_erosed\_img = apply\_dilation(erosed\_img, kernel)

plt.figure("opening")

plt.subplot(131)

plt.imshow(img,cmap='gray')

plt.title("original")

plt.subplot(132)

plt.imshow(erosed\_img,cmap='gray')

plt.title("after erosion")

plt.subplot(133)

plt.imshow(dil\_erosed\_img,cmap='gray')

plt.title("after erosion & dilation")

plt.show()

return dil\_erosed\_img

def apply\_closing(img, kernel):

dilated\_img = apply\_dilation(img, kernel)

er\_dilated\_img = apply\_dilation(dilated\_img, kernel)

plt.figure("closing")

plt.subplot(131)

plt.imshow(img,cmap='gray')

plt.title("original")

plt.subplot(132)

plt.imshow(dilated\_img,cmap='gray')

plt.title("after dilation")

plt.subplot(133)

plt.imshow(er\_dilated\_img,cmap='gray')

plt.title("after dilation & erosion")

plt.show()

return er\_dilated\_img

img = cv2.imread('images/shape2.bmp',cv2.IMREAD\_GRAYSCALE)

img[img==255]=1

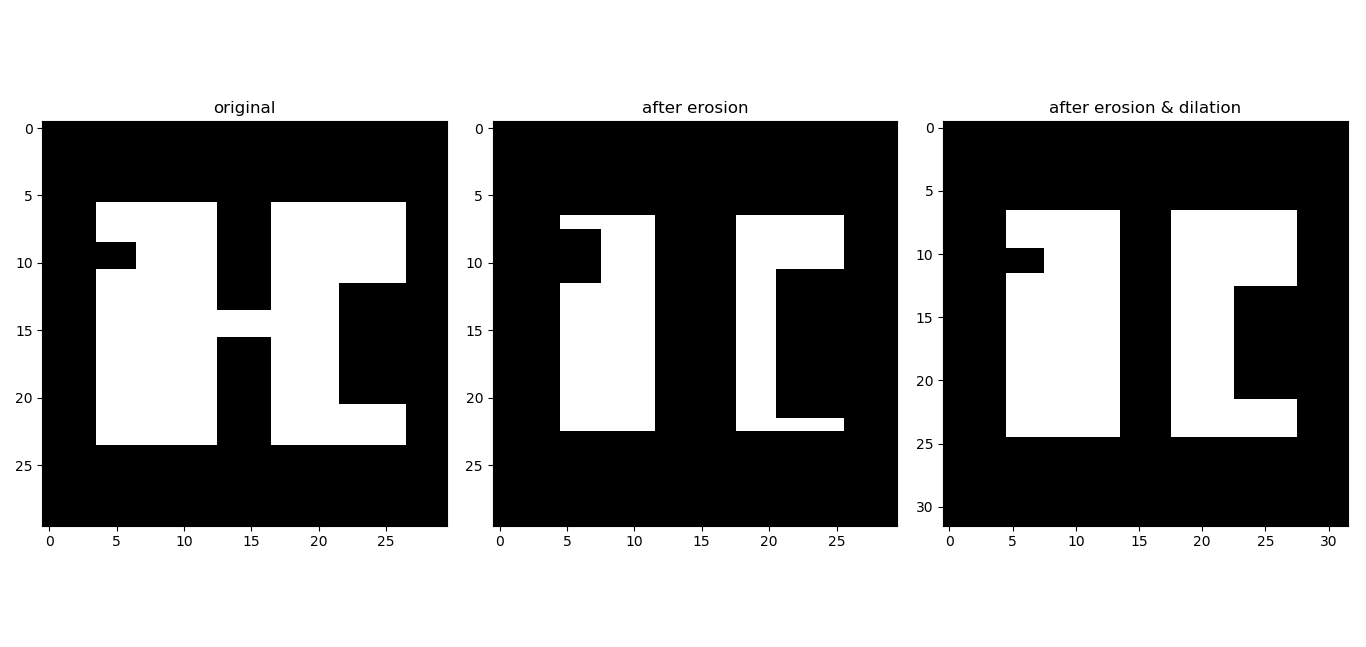
kernel = np.ones((3,3))

apply\_opening(img,kernel)

apply\_closing(img,kernel)

Output:

Opening



Closing

