EXPERIMENT 1

Digital Image Processing

M.W. Nethmi N. Muthugala - 那娜

School of AI

W2010816010

31th October, 2022

CONTENT

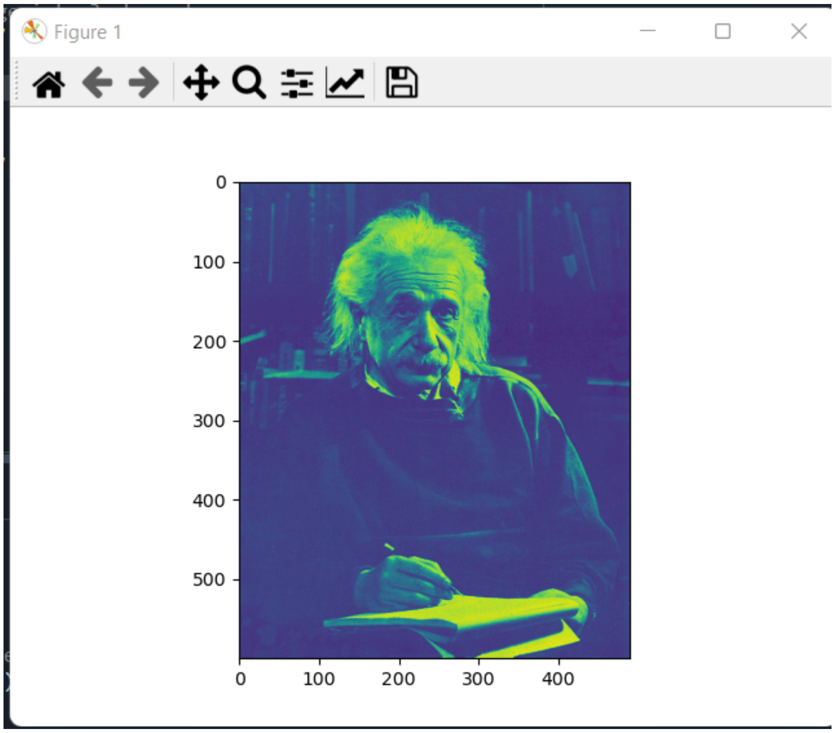
1. Single step experiments
    a. Load images with opencv function into the ipython environment.
    b. Draw histogram of an image, try histogram equalization.
    c. Add salt and pepper noise to an image, and remove the noise with a median filter.
    d. Add gaussian noise to an image, and remove the noise with a gaussian filter.
    e. Apply intensity normalization to an image.
    f. Apply gamma correction to an image.
    g. Apply any kind of linear intensity transformation of an image

2. Compound operations
    a. Implement a 2d spatial filter with python and compare it with conv2d filter from opencv library from the perspective of efficiency and effect.
    b. Do image enhancement, especially edge enhancement to ‘img7.tif’.
    c. Find edges and feature points from images

Single step experiments

1.1) Load images with opencv function into ipython environment

```
import cv2 as cv

#reading after splitting the image into 3 channels.

im1 = cv.imread('./pics/img1.tif')

im1b, im1g, im1r = cv.split(im1)

#reading as a gray image

im2 = cv.imread('./pics/img2.tif', 0)
```



Output image - im2

1.2) Draw histogram of an image, try histogram equalization

```
import cv2 as cv
import numpy as np
im = cv.imread('./pics/img6.tif', 0)
hist = cv.calcHist([im],[0],None,[256],[0,256])
```

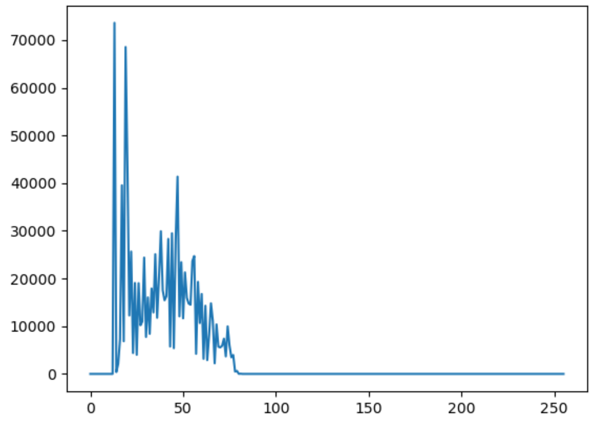
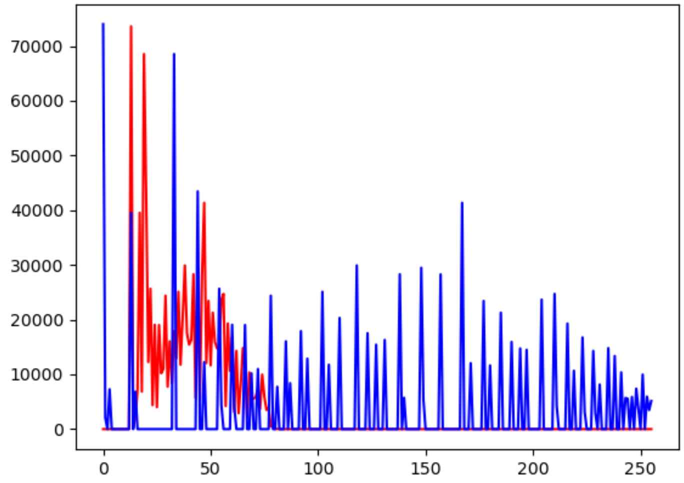


Image: hist

```
#min intensity is 10, max intensity is 81
equ = cv.equalizeHist(im)
hist2 = cv.calcHist([equ],[0],None,[256],[0,256])


import matplotlib.pyplot as plt
plt.figure()
plt.plot(hist, 'r-') #red curve-original
plt.plot(hist2, 'b-') #blue curve- equalized
```



Red - Original histogram, Blue- equalized histogram

```
res = np.hstack((im,equ)) #stacking images side-by-side

cv.imwrite('res.png',res)
```

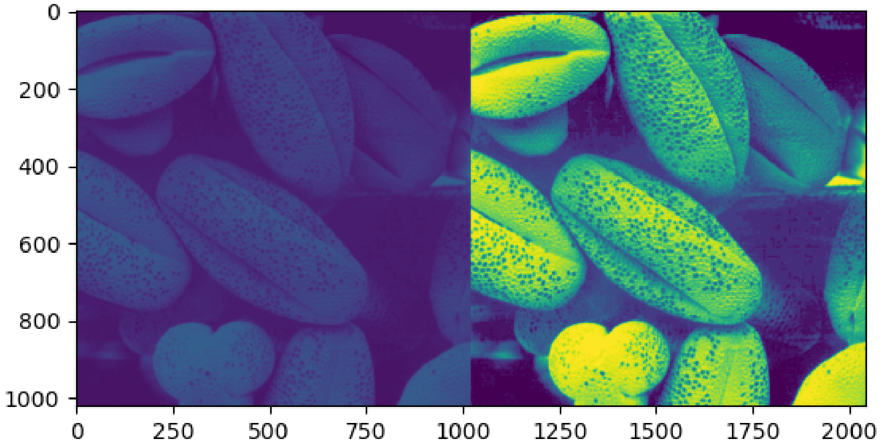
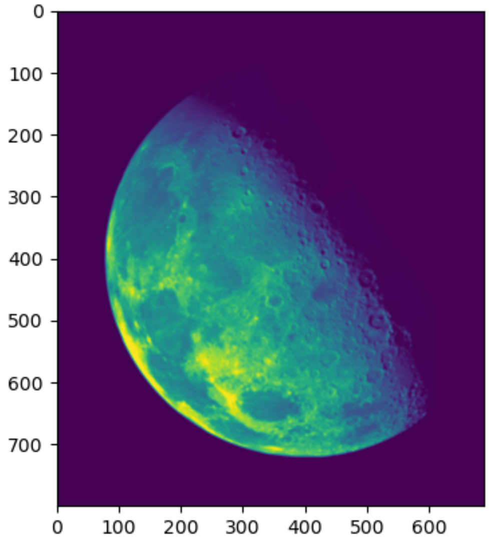


Image: res

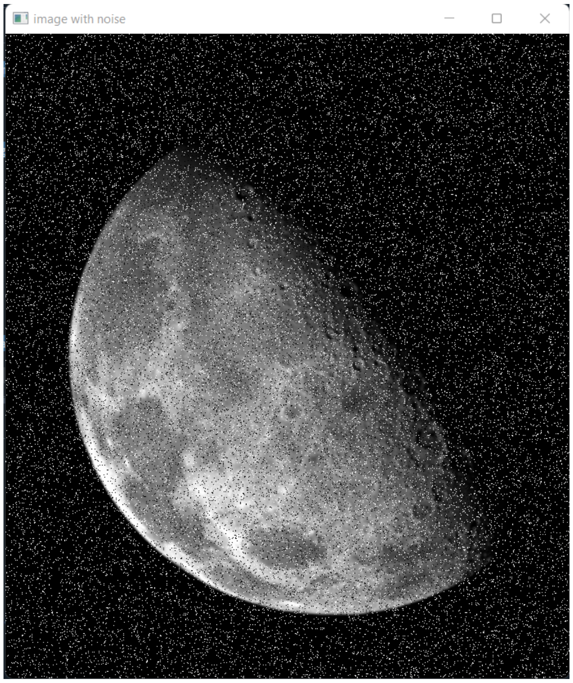
1.3) Add salt and pepper noise to an image, and remove the noise with median filter

```
#adding salt and pepper to an image

import cv2 as cv

import numpy as np


# 1) Open the original image

img = cv.imread('./pics/img4.tif', 0)

img = img/255
```



```
# 2) Create a blank image

x,y = img.shape

g= np.zeros((x,y), dtype=np.float32)

# 3) randomly filling the blank image

 # salt and pepper amount
```

```
pepper = 0.05

salt = 1 - pepper

#create salt and pepper noise image

for i in range(x):

    for j in range(y):

        rdn = np.random.random()

        if rdn < pepper:

            g[i][j] = 0

        elif rdn > salt:

            g[i][j] = 1

        else:

            g[i][j] = img [i][j]

        # 5% pepper noise and 5% alt noise

cv.imshow('image with noise', g)
```



```
#removing salt and pepper noise

import cv2 as cv

import numpy as np


im = cv.imread('./pics/img20.tif', 0)
```

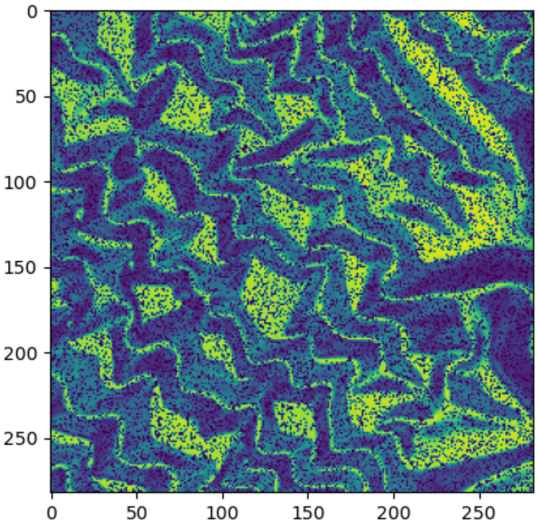


Image: im

```
imm = cv.medianBlur(im, 3)    #using median filter

im7 = cv.medianBlur(im, 7)
```

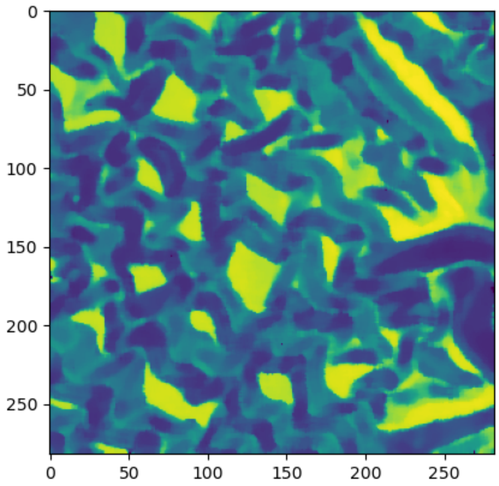


Image: im7

```
immm = cv.medianBlur(imm, 3) # use twice  of the same kernal
```

#use mean filter

im_mean = cv.filter2D(im, -1, np.ones((7,7))/49)

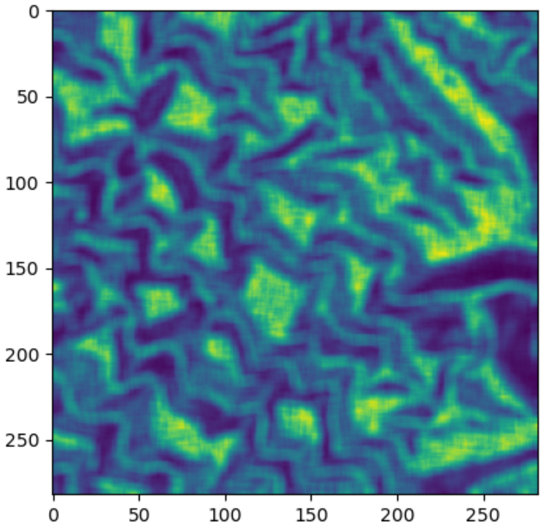


Image: im_mean

1.4) Add gaussian noise to an image, and remove the noise with gaussian filter

#ADDING GAUSSIAN NOISE

```
import cv2 as cv

import numpy as np

im = cv.imread('./pics/img5.tif', 0)

im = im/255
```

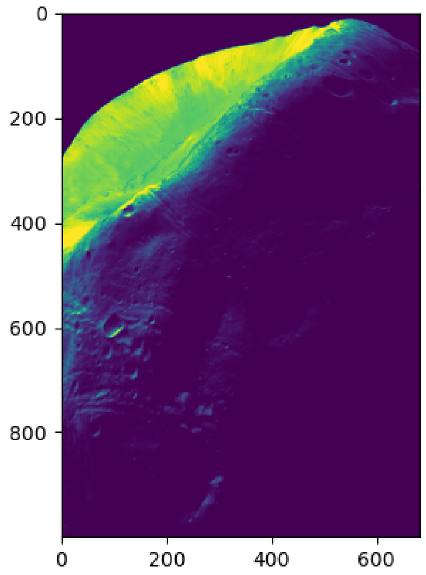
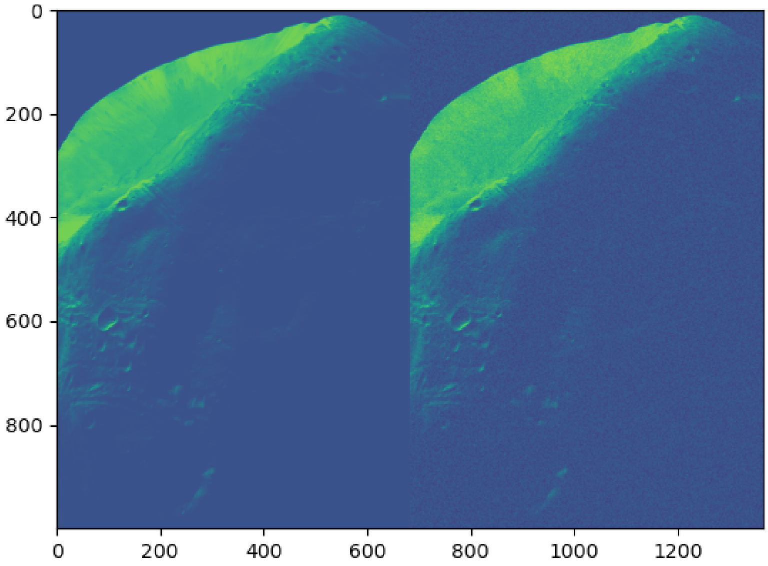


Image: im

```
#create gaussian noise

x, y = im.shape

mean = 0

var = 0.01

sigma = np.sqrt(var)

n= np.random.normal(loc = mean,

                scale = sigma,

                size = (x, y))
```

```
#add gaussian noise

g = im + n

res = np.hstack((im, g)) #stacking images side-by-side

cv.imwrite('res.png',res)
```



#GAUSSIAN DENOISING

```
import cv2 as cv

import numpy as np

im = cv.imread('./pics/img20.tif', 0)
```

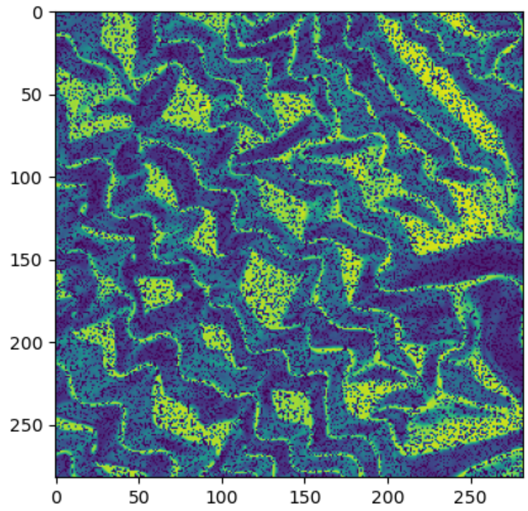


Image: im

```
imm = cv.GaussianBlur(im, (7,7), 0, borderType = cv.BORDER_CONSTANT)

res = np.hstack((im, imm)) #stacking images side-by-side

cv.imwrite('res.png',res)
```

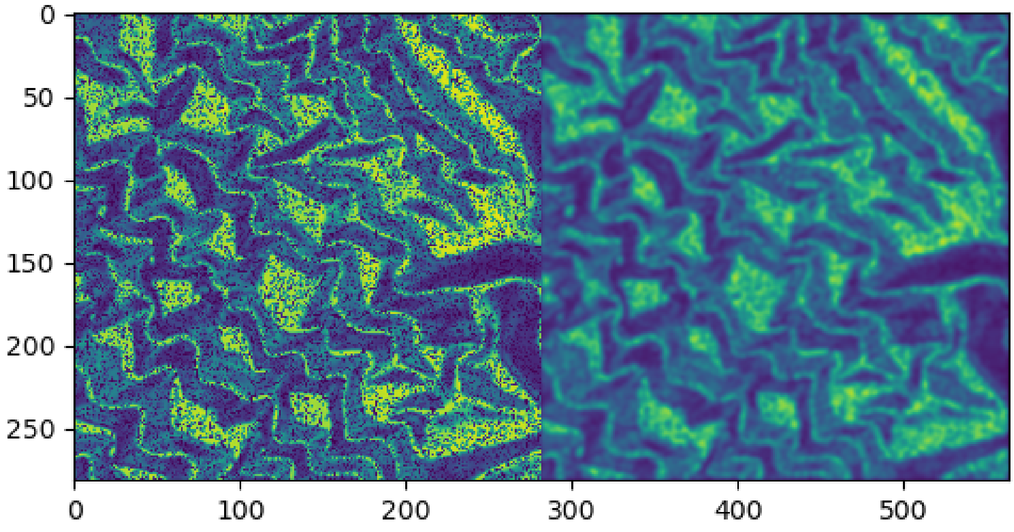


Image:res

1.5) Apply intensity normalization to an image

import cv2 as cv

import numpy as np

img = cv.imread('./pics/img8.tif', 0) #original image

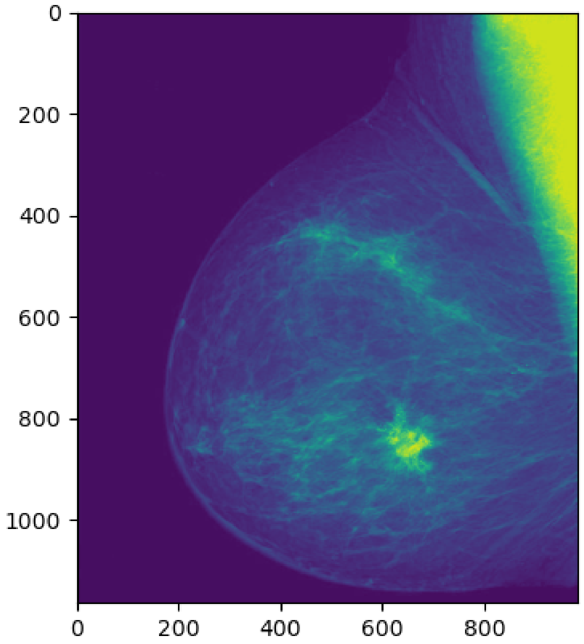


Image: img

n_img = np.zeros((800,800))

final_img = cv.normalize(img, n_img, 0, 255, cv.NORM_MINMAX)

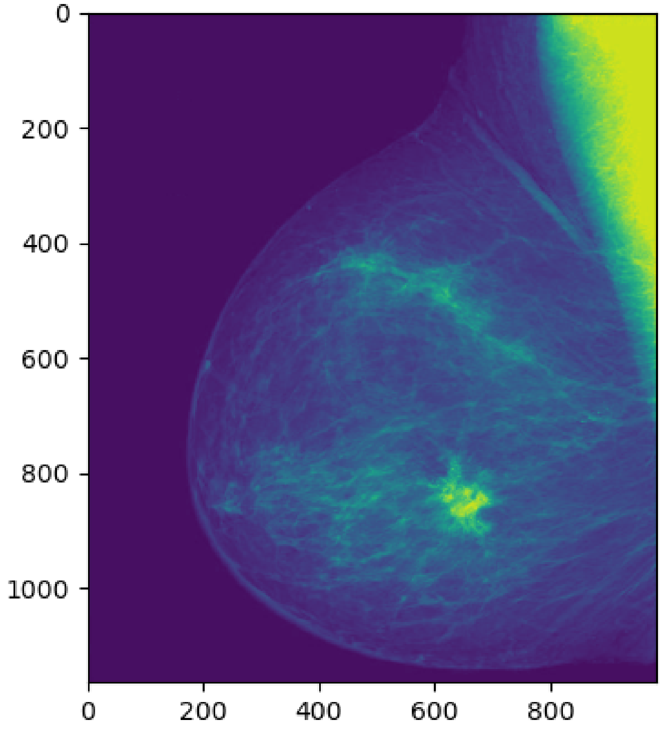


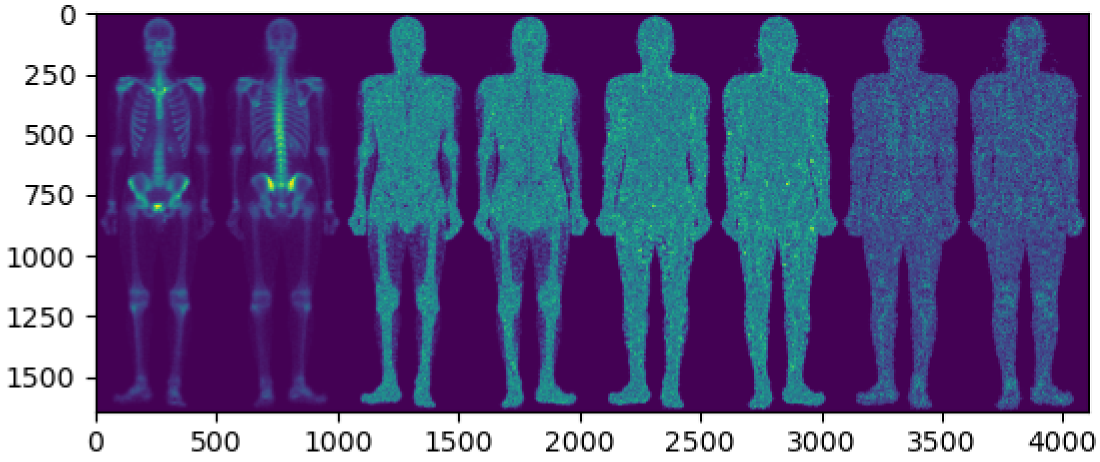
Image: final_img

res = np.hstack((img, final_img)) #stacking images side-by-side

cv.imwrite('res.png',res)

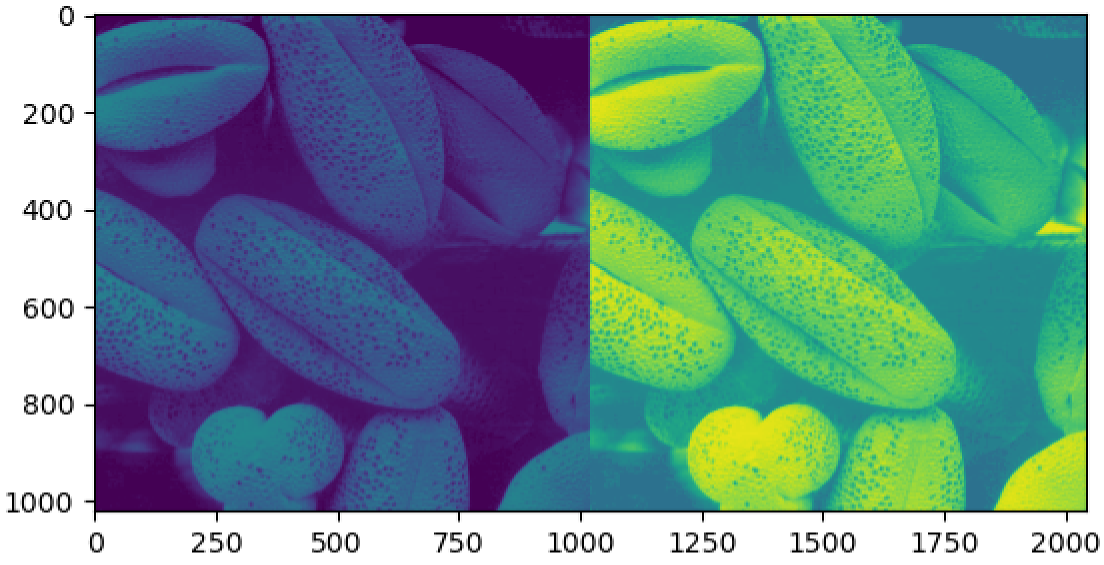
1.6) Apply gamma correction to an image

```
# GAMMA TRANSFORMATION

import cv2 as cv

import numpy as np

im = cv.imread('./pics/img7.tif', 0)

gamma = 2

im2 = np.power(im, gamma)


gamma = 3

im3 = np.power(im, gamma)


gamma = 4

im4 = np.power(im, gamma)


res = np.hstack((im, im2, im3, im4)) #stacking images side-by-side

cv.imwrite('res.png',res)
```



# GAMMA CORRECTION

```
import cv2 as cv

import numpy as np


def gammaCorrection(src, gamma):

    invGamma = 1 / gamma


    table = [((i / 255) ** invGamma) * 255 for i in range(256)]

    table = np.array(table, np.uint8)

    return cv.LUT(src, table)

im = cv.imread('./pics/img6.tif', 0)

gammaImg = gammaCorrection(im, 2.2)


res = np.hstack((im, gammaImg)) #stacking images side-by-side

cv.imwrite('res.png',res)
```



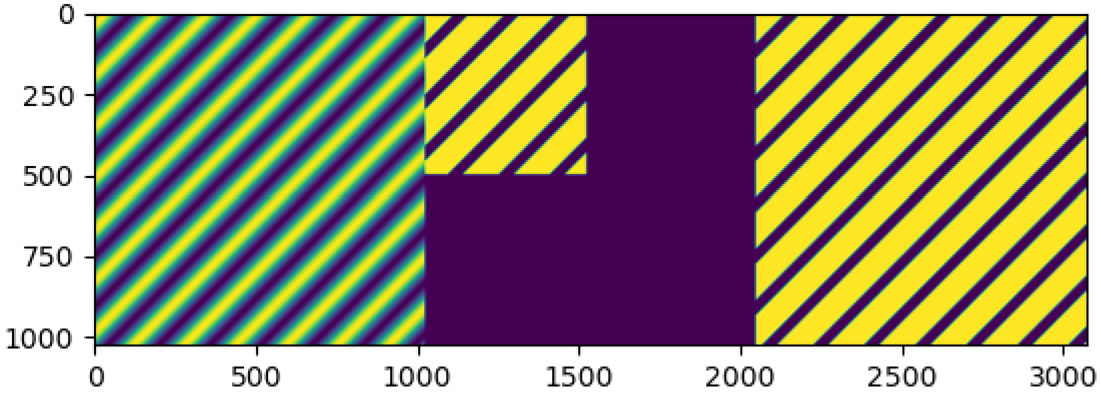
1.7) Apply any kind of linear intensity transformation of an image

```
import cv2 as cv
import numpy as np
im = cv.imread('./pics/img1.tif', 0)

imb = np.zeros_like(im)
thresh = 50 #threshhold of intensity

#loops all the pixels in the image
for row in range (500): #loop along row
    for col in range (500): #loop along column
        #print(im[row, col])

#transform with T - like binarization
        if im[row, col] > thresh:
            imb[row, col] = 255

#real life scenario
#numpy vectorization tech
imb2 = (im > thresh)
imb2 = np.uint8(imb2)*255

#compare 3 images pixel by pixel
```

```
diff = np.abs(imb-imb2)

print('The maximum of difference is %lg.'%np.max(diff))
```

Output: The maximum of difference is 1.

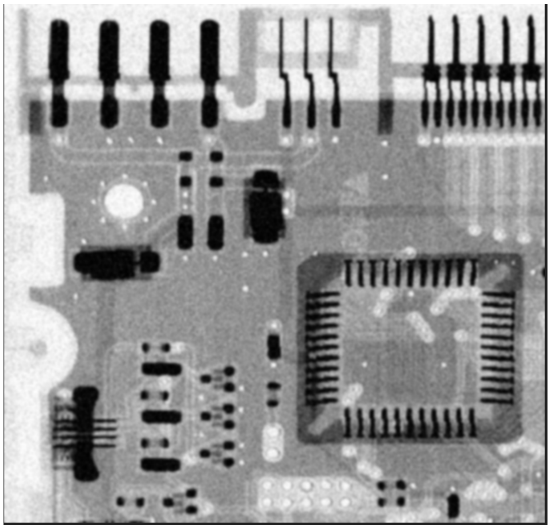
```
res = np.hstack((im, imb, imb2)) #stacking images side-by-side

cv.imwrite('res.png',res)
```



Compound operations

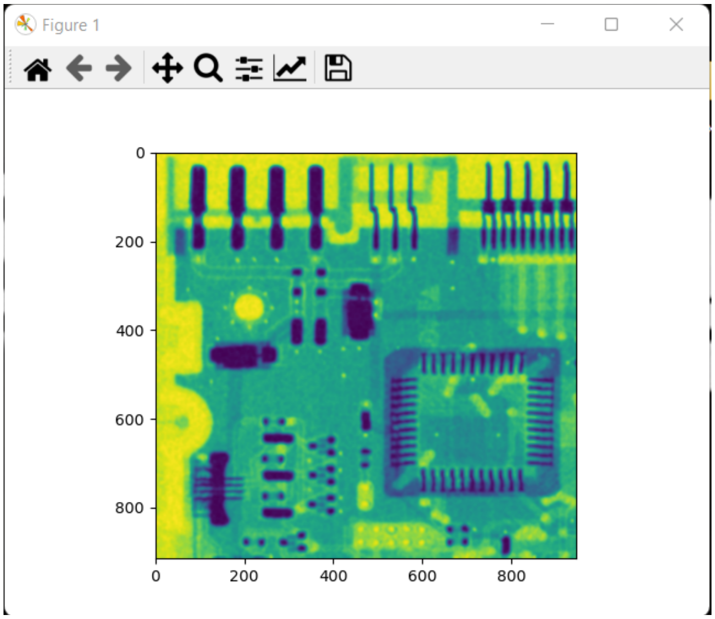
2.1) Implement a 2d spatial filter with python and compare it with conv2d filter from opencv library from the perspective of efficiency and effect

```
# ImageFilter for using filter() function

from PIL import Image, ImageFilter

# Opening the image

# (R prefixed to string in order to deal with '\' in paths)

image = Image.open(r"./pics/img30.tif")



# Blurring image by sending the ImageFilter.

# GaussianBlur predefined kernel argument

image = image.filter(ImageFilter.GaussianBlur)

# Displaying the image

image.show()
```



#USING FILTER2D OPTION

```
import cv2 as cv

import numpy as np


img = cv.imread('./pics/img30.tif', 0)

k = cv.getGaussianKernel(17, 3) #getting gaussian kernal

# 17*17 kernal in 2D

#(k*1) * (1*k) matrices

k2d = np.dot(k, k.T)

imgfilter2d = cv.filter2D(img, -1, k2d)

cv.imshow('filter2d method', imgfilter2d)
```

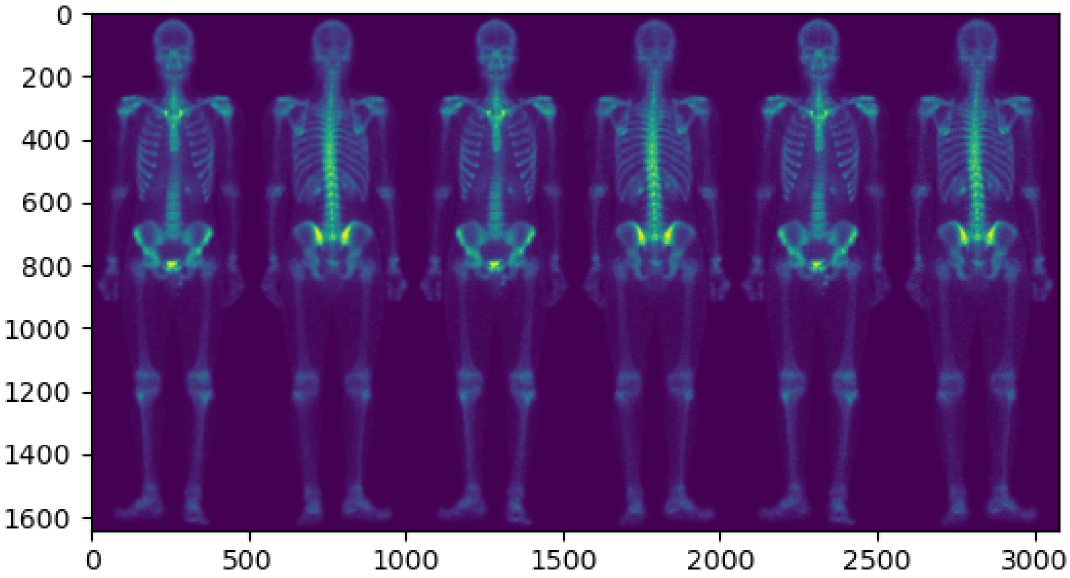


```
res = np.hstack((image, imgfilter2d)) #stacking images side-by-side

cv.imwrite('res.png',res)
```

2.2) Do image enhancement especially edge enhance to ‘img7.tif

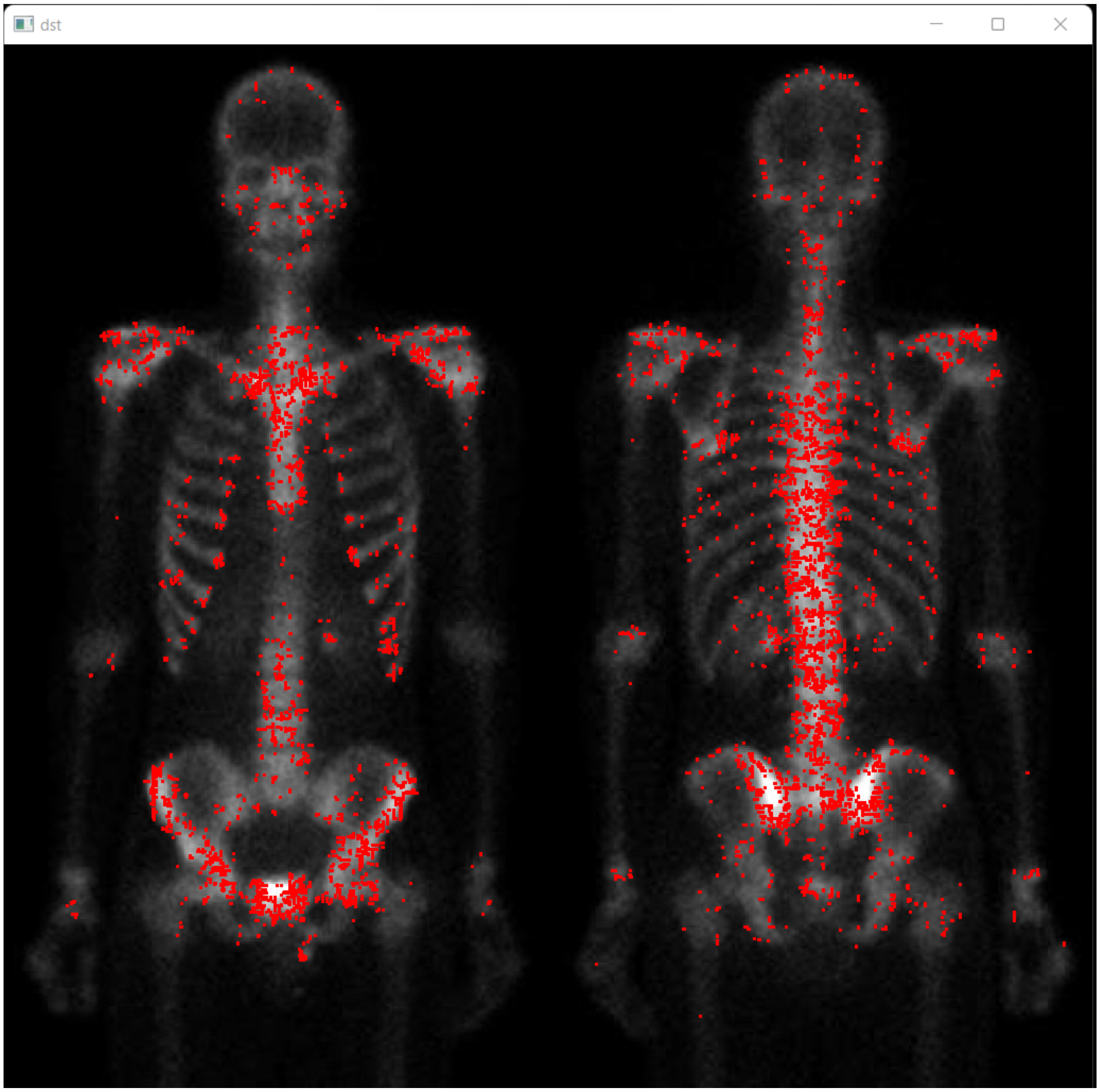
```
# import image module

from PIL import Image

from PIL import ImageFilter


# Open an already existing image

img = Image.open('./pics/img7.tif')


# Apply edge enhancement filter

edgeEnahnced = img.filter(ImageFilter.EDGE_ENHANCE)


# Apply increased edge enhancement filter

moreEdgeEnahnced = img.filter(ImageFilter.EDGE_ENHANCE_MORE)


# Show original image - before applying edge enhancement filters

img.show()


# Show image - after applying edge enhancement filter

edgeEnahnced.show()

# Show image - after applying increased edge enhancement filter

moreEdgeEnahnced.show()
```

```
import cv2 as cv

import numpy as np


res = np.hstack((img, edgeEnahnced, moreEdgeEnahnced)) #stacking images side-by-side

cv.imwrite('res.png',res)
```



2.3) Find edges and feature points from images

```
import cv2 as cv
import numpy as np

filename = './pics/img7.tif'
img = cv.imread(filename)
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
#img = cv.imread('./pics/img7.tif', 0)

gray = np.float32(gray)
dst = cv.cornerHarris(gray,2,3,0.04)

#result is dilated for marking the corners, not important
dst = cv.dilate(dst,None)

# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,0,255]

cv.imshow('dst',img)
```



dst