** **

**COM466B**

**DIGITAL IMAGE PROCESSING**

**Practical Coursework as Midterm Exam**

*Instructor : Kurtuluş KÜLLÜ*

*Student Name&Surname: Tuğçe Nur ŞAHİN*

*Student Number: 15290122*

Visual Studio Code IDE was used during the homework preparation process. The code is written in Python programming language.

**1)**

import cv2

image=cv2.imread('pelikan.jpg',cv2.IMREAD\_GRAYSCALE)

cv2.imshow('GreyImage', image)

cv2.waitKey()

cv2.destroyAllWindows()

The cv2 library, OpenCv's library, was imported in the above code. Thanks to the **IMREAD\_GRAYSCALE** function inside the cv2, the photo has been changed from color to gray color. Had it replaced **0** with cv2.IMREAD\_GRAYSCALE, it would have performed the same function. It is used to reduce the channels divided into 3 as BGR (Blue-Green-Red) into a single channel.

In addition, The **imshow()** method is used to show the photo.

The **waitKey()** method specifies how long the photo will remain on the screen.

The **destroyAllWindows()** method closes all pages when page is closed.

**Figure 1.1** (Color photo) **Figure1.2**(TheIMREAD\_GRAYSCALE method is applied)

**2)**

import cv2

import numpy as np

image = cv2.imread("pelikan.jpg",0)

def mean(im,im1,x,y,i,j):

    im[(i,j)]=(int(im1[(x,y)])+int(im1[(x+1,y)])+int(im1[(x,y+1)])+int(im1[(x+1,y+1)]))/4

    return im[(i,j)]

def resizeToQuarter(image):

    boy= int(image.shape[0])

    en = int(image.shape[1])

    print(en,boy)

    boy1=int(boy/2)

    en1=int(en/2)

    image1=np.zeros((boy1,en1,1),np.uint8)

    i=0

    for x in range (0,boy-1,2):

     j=0

        for y in range(0,en-1,2):

            image1[(i,j)]=mean(image1,image,x,y,i,j)

            j=j+1

        i=i+1

    return image1

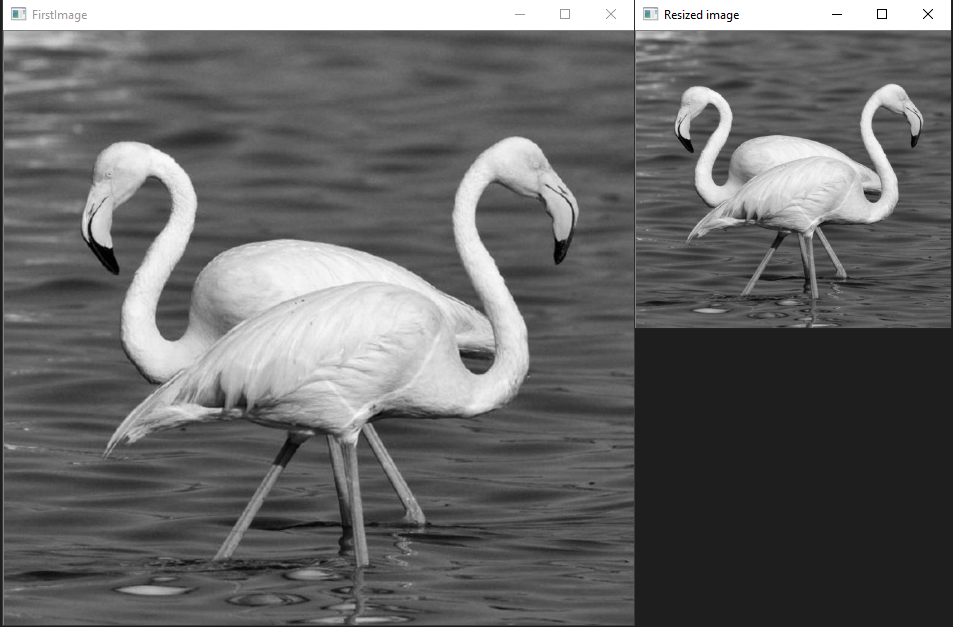
cv2.imshow("FirstImage",  image)

cv2.imshow("Resized image", resizeToQuarter(image))

cv2.waitKey(0)

cv2.destroyAllWindows()

The photograph, which is 630X594 under normal conditions, has been reduced to 1 in 4. In the **resizeToQuarter** function, the main functions are performed. For the ratio of 1 to 4, the width and length are divided into two. For these photographs (315X297), an empty array was created with the help of the Numpy library. (Np.zeros ((boy1, en1,1), np.uint8)) Each pixel was sent to the mean function for itself and the average of the neighboring pixels. Since the adjacent pixels enter the average, the values are skipped in duplicate in the photo(for loop). The received average pixel values are placed in the newly created empty array.



**Figure 2.1**

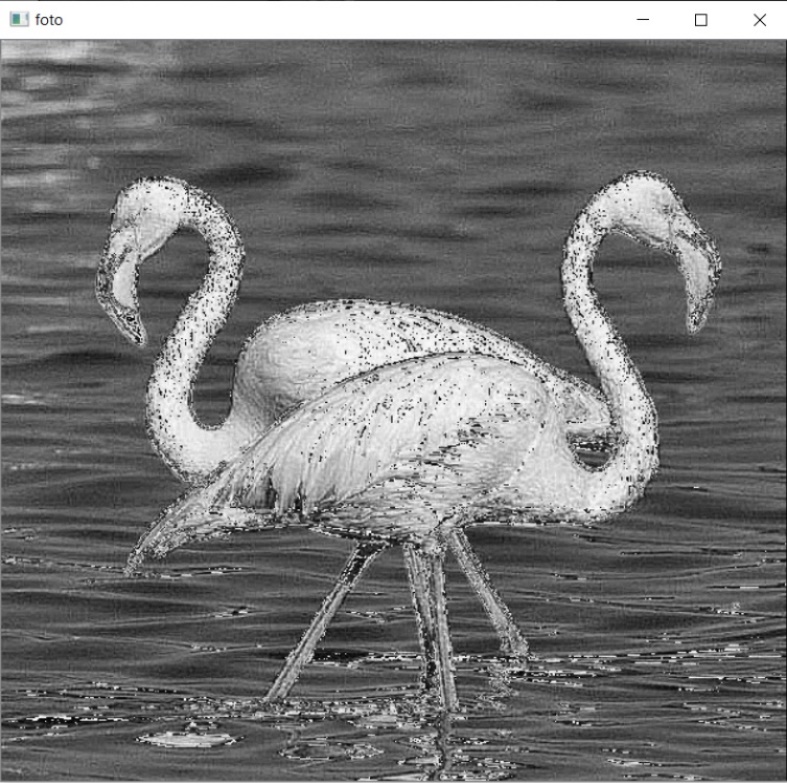
Figure 2.1 show that the photo has been reduced by 1 in 4.

**3)**

Since the code is long, the full code is provided at the end of the problem.

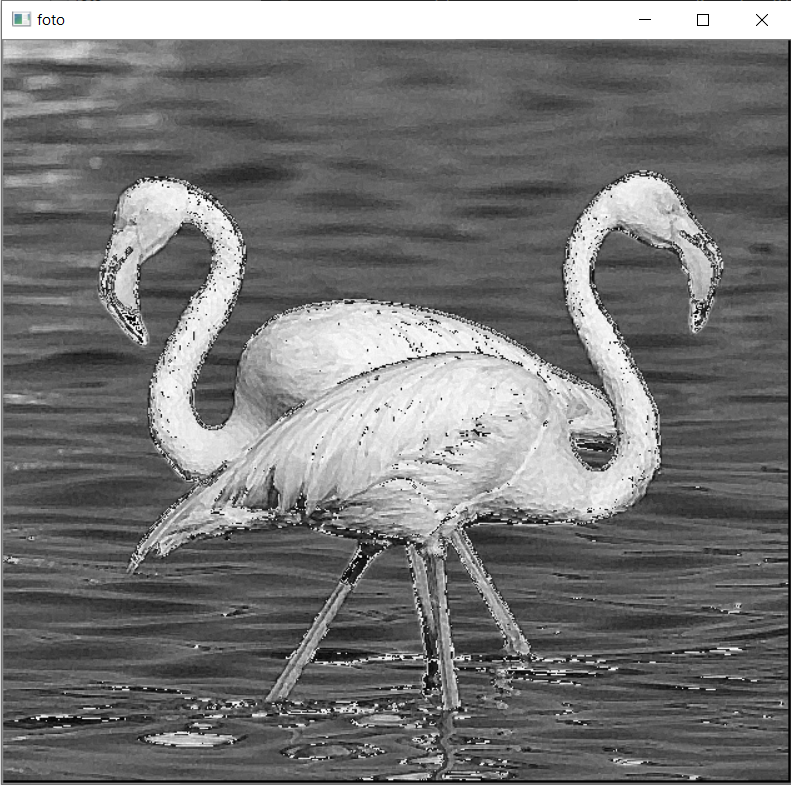
An array with the values given for the Laplacian filter has been created. The for loop has been started so that it can be navigated in each pixel. However, there are 9 if blocks in this loop. 4 of them are always for one corner, the other 4 are for edges, and the last if block is for the middle area.

There are some problems encountered. The most important of these is overflow in the range [0.255]. For this problem, 3 different filters were applied considering that they may be caused by noise.

****

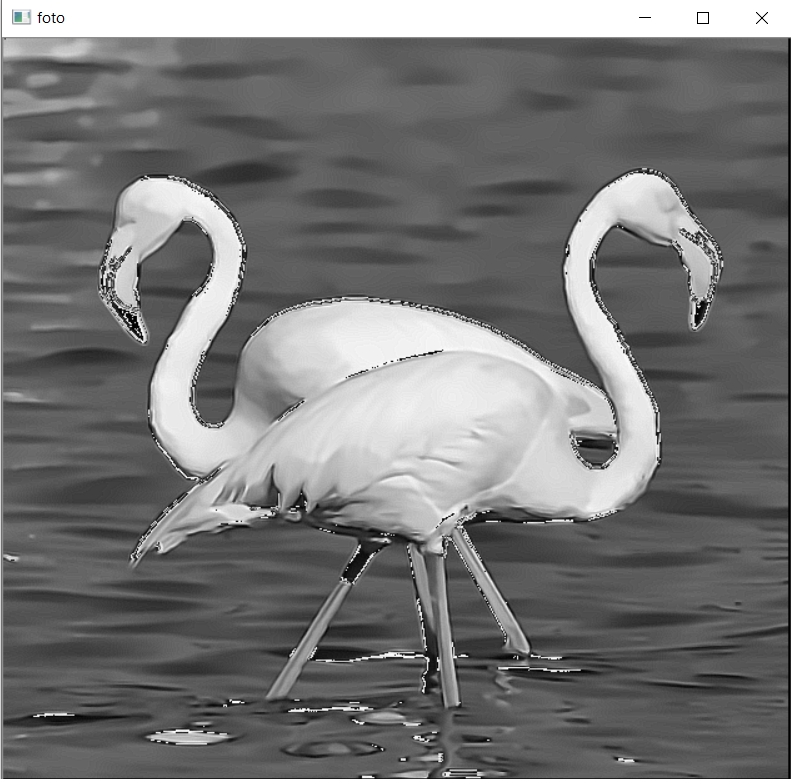
**Figure 3.1**

Figure 3.1 show that only the Laplacian function has been implemented. The overflow in the range [0.255] was seen as [631, -507].

****

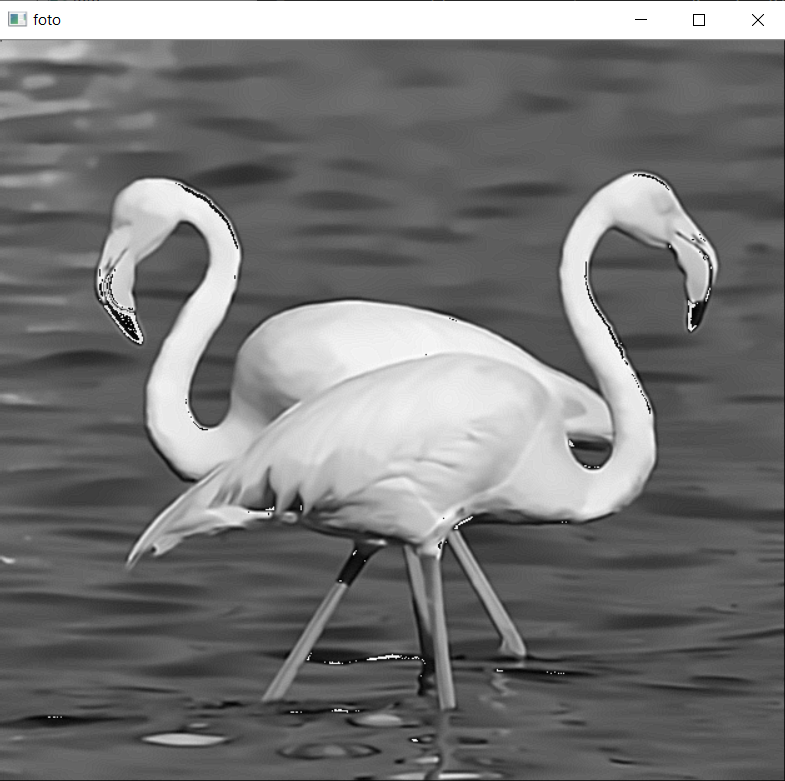
**Figure 3.2**

In figure 3.2, the medianBlur () filter was applied to reduce the value range in the previous photograph. It is aimed to get a better result by getting rid of the noises. However, in this photograph, the range [504, -311] was determined. A good result has not been achieved yet.

****

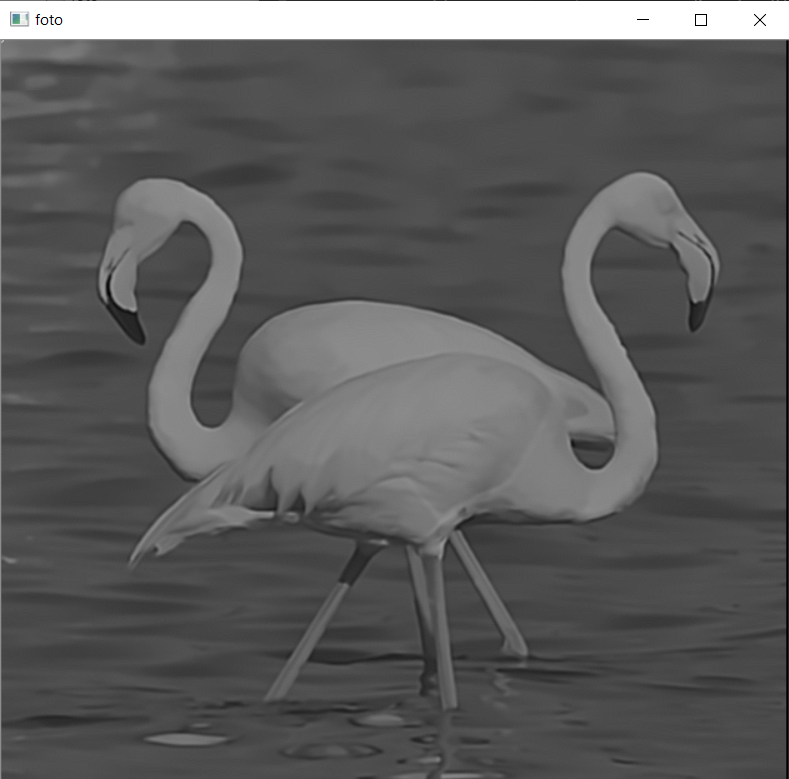
**Figure 3.3**

In Figure 3.3, the mean filter was applied to the photograph that has been applied a Median filter to reduce the value range in the previous photograph. It is aimed to achieve a better result by reducing the noises. However, in this photograph, the range [515, -299] was determined. A good result has not been achieved yet.

****

**Figure 3.4**

In Figure 3.4, Gaussian filter was applied in addition to the photo with two filters applied. In this photograph, the range is set to [515, -90]. Although the range has decreased, it is not at the desired size.

****

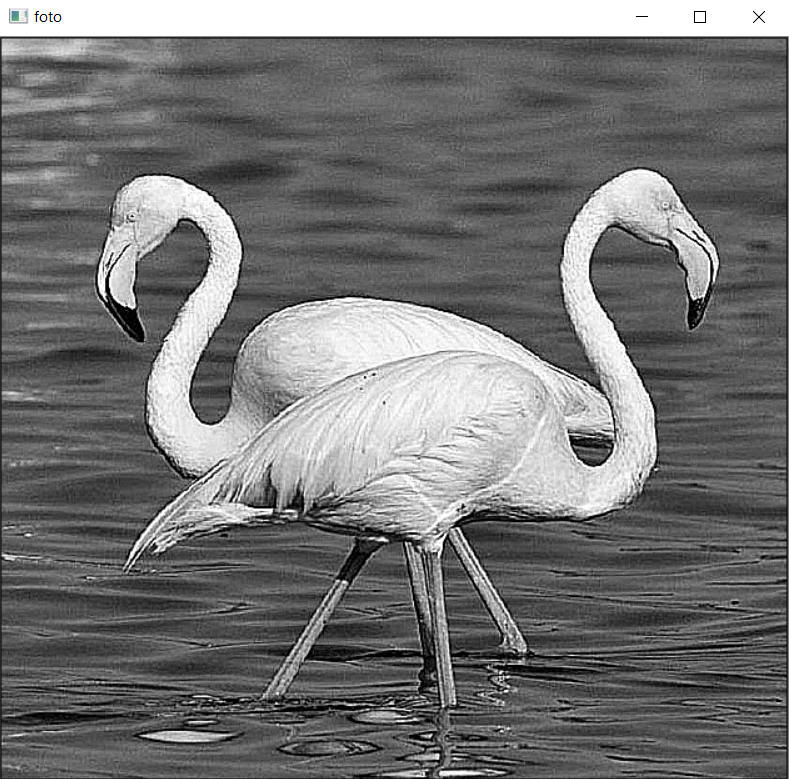
**Figure 3. 5**

In Figure 3.5, 3 filters were applied. The following code line has been added to be included in the specified range.

                sum1[a]=sum1[a]+90

                sum1[a]=int(sum1[a]/(2.37))

Thanks to this, we now have a range of [0.255], but the photo is grayscale because it is taken from a large range. When Laplacian is applied several times, the edges become clear but the image clarity is impaired. This did not achieve the desired result.



**Figure 3.6**

In Figure 3.6, the problem was made with another perspective in mind. In order to fix it to the [0,255] range, those over 255 are equal to 255 and those below 0 are equal to 0. This is seen in the lines of code given below.

            if(sum1[a] < 0):

               sum1[a] = 0

            elif(sum1[a] > 255):

                sum1[a] = 255

Previous works are included in the comment line as can be seen in the code below and this method was chosen because it gives better results.

import cv2

import numpy as np

img = cv2.imread('pelikan.jpg', 0)

cv2.imshow('ilk resim', img)

#cv2.imwrite("grayburhan.jpeg",img)

#img = cv2.medianBlur(img, ksize=3)

#img = cv2.GaussianBlur(img, (5,5), 0)

cv2.imshow('filtre', img)

def mySharpening(img):

    laplacian = np.array((

        [-1, -1, -1],

        [-1, 9, -1],

        [-1, -1, -1]), dtype="int")

    boy = int(img.shape[0])

    en = int(img.shape[1])

    print(img.shape)

    new = np.zeros((boy, en, 1), np.uint8)

    sum1 = np.zeros(((boy) \* (en), 1), 'int32')

    a = 0

    max1 = 0

    min1 = 0

    for i in range(0,boy - 2,1):

        for j in range(0,en - 2,1):

            if (j == en - 2 and i < 0):

                a1 = img[i + 1][j] \* laplacian[0][0]

                b = img[i + 1][j] \* laplacian[1][0]

                c = img[i + 2][j] \* laplacian[2][0]

                d = img[i + 1][j + 1] \* laplacian[0][1]

                e = img[i + 1][j + 1] \* laplacian[1][1]

                f = img[i + 2][j + 1] \* laplacian[2][1]

                g = img[i + 1][j + 1] \* laplacian[0][2]

                h = img[i + 1][j + 1] \* laplacian[1][2]

                z = img[i + 2][j + 1] \* laplacian[2][2]

                sum1[a] = int(a1 + b + c + d + e + f + g + h + z)

                #sum1[a]=sum1[a]+90

                #sum1[a]=int(sum1[a]/(2.37))

                new[i + 1][j + 1] = sum1[a]

            elif (j < 0 and i < 0):

                a1 = img[i + 1][j + 1] \* laplacian[0][0]

                b = img[i + 1][j + 1] \* laplacian[1][0]

                c = img[i + 2][j + 1] \* laplacian[2][0]

                d = img[i + 1][j + 1] \* laplacian[0][1]

                e = img[i + 1][j + 1] \* laplacian[1][1]

                f = img[i + 2][j + 1] \* laplacian[2][1]

                g = img[i + 1][j + 2] \* laplacian[0][2]

                h = img[i + 1][j + 2] \* laplacian[1][2]

                z = img[i + 2][j + 2] \* laplacian[2][2]

                sum1[a] = int(a1 + b + c + d + e + f + g + h + z)

                #sum1[a]=sum1[a]+90

                #sum1[a]=int(sum1[a]/(2.37))

                new[i + 1][j + 1] = sum1[a]

            elif (i == boy - 2 and j < 0):

                a1 = img[i][j + 1] \* laplacian[0][0]

                b = img[i + 1][j + 1] \* laplacian[1][0]

                c = img[i + 1][j + 1] \* laplacian[2][0]

                d = img[i][j + 1] \* laplacian[0][1]

                e = img[i + 1][j + 1] \* laplacian[1][1]

                f = img[i + 1][j + 1] \* laplacian[2][1]

                g = img[i][j + 2] \* laplacian[0][2]

                h = img[i + 1][j + 2] \* laplacian[1][2]

                z = img[i + 1][j + 2] \* laplacian[2][2]

                sum1[a] = int(a1 + b + c + d + e + f + g + h + z)

                #sum1[a]=sum1[a]+90

                #sum1[a]=int(sum1[a]/(2.37))

                new[i + 1][j + 1] = sum1[a]

            elif (j == en - 2 and i == en - 2):

                a1 = img[i][j] \* laplacian[0][0]

                b = img[i + 1][j] \* laplacian[1][0]

                c = img[i + 1][j] \* laplacian[2][0]

                d = img[i][j + 1] \* laplacian[0][1]

                e = img[i + 1][j + 1] \* laplacian[1][1]

                f = img[i + 1][j + 1] \* laplacian[2][1]

                g = img[i][j + 1] \* laplacian[0][2]

                h = img[i + 1][j + 1] \* laplacian[1][2]

                z = img[i + 1][j + 1] \* laplacian[2][2]

                sum1[a] = int(a1 + b + c + d + e + f + g + h + z)

                #sum1[a]=sum1[a]+90

                #sum1[a]=int(sum1[a]/(2.37))

                new[i + 1][j + 1] = sum1[a]

            elif (j < 0 and i > 0 and i < boy + 1):

                a1 = img[i][j + 1] \* laplacian[0][0]

                b = img[i + 1][j + 1] \* laplacian[1][0]

                c = img[i + 2][j + 1] \* laplacian[2][0]

                d = img[i][j + 1] \* laplacian[0][1]

                e = img[i + 1][j + 1] \* laplacian[1][1]

                f = img[i + 2][j + 1] \* laplacian[2][1]

                g = img[i][j + 2] \* laplacian[0][2]

                h = img[i + 1][j + 2] \* laplacian[1][2]

                z = img[i + 2][j + 2] \* laplacian[2][2]

                sum1[a] = int(a1 + b + c + d + e + f + g + h + z)

                #sum1[a]=sum1[a]+90

                #sum1[a]=int(sum1[a]/(2.37))

                new[i + 1][j + 1] = sum1[a]

            elif (i < 0 and j < en - 2 and j > 0):

                a1 = img[i + 1][j] \* laplacian[0][0]

                b = img[i + 1][j] \* laplacian[1][0]

                c = img[i + 2][j] \* laplacian[2][0]

                d = img[i + 1][j + 1] \* laplacian[0][1]

                e = img[i + 1][j + 1] \* laplacian[1][1]

                f = img[i + 2][j + 1] \* laplacian[2][1]

                g = img[i + 1][j + 2] \* laplacian[0][2]

                h = img[i + 1][j + 2] \* laplacian[1][2]

                z = img[i + 2][j + 2] \* laplacian[2][2]

                sum1[a] = int(a1 + b + c + d + e + f + g + h + z)

                #sum1[a]=sum1[a]+90

                #sum1[a]=int(sum1[a]/(2.37))

                new[i + 1][j + 1] = sum1[a]

            elif (j == en - 2 and i > 0 and i < boy - 2):

                a1 = img[i][j] \* laplacian[0][0]

                b = img[i + 1][j] \* laplacian[1][0]

                c = img[i + 2][j] \* laplacian[2][0]

                d = img[i][j + 1] \* laplacian[0][1]

                e = img[i + 1][j + 1] \* laplacian[1][1]

                f = img[i + 2][j + 1] \* laplacian[2][1]

                g = img[i][j + 1] \* laplacian[0][2]

                h = img[i + 1][j + 1] \* laplacian[1][2]

                z = img[i + 2][j + 1] \* laplacian[2][2]

                sum1[a] = int(a1 + b + c + d + e + f + g + h + z)

                #sum1[a]=sum1[a]+90

                #sum1[a]=int(sum1[a]/(2.37))

                new[i + 1][j + 1] = sum1[a]

            elif (i == en - 2 and j > 0 and j < en - 2):

                a1 = img[i][j] \* laplacian[0][0]

                b = img[i + 1][j] \* laplacian[1][0]

                c = img[i + 1][j] \* laplacian[2][0]

                d = img[i][j + 1] \* laplacian[0][1]

                e = img[i + 1][j + 1] \* laplacian[1][1]

                f = img[i + 1][j + 1] \* laplacian[2][1]

                g = img[i][j + 2] \* laplacian[0][2]

                h = img[i + 1][j + 2] \* laplacian[1][2]

                z = img[i + 1][j + 2] \* laplacian[2][2]

                sum1[a] = int(a1 + b + c + d + e + f + g + h + z)

                #sum1[a]=sum1[a]+90

                #sum1[a]=int(sum1[a]/(2.37))

                new[i + 1][j + 1] = sum1[a]

            else:

                a1 = img[i][j] \* laplacian[0][0]

                b = img[i + 1][j] \* laplacian[1][0]

                c = img[i + 2][j] \* laplacian[2][0]

                d = img[i][j + 1] \* laplacian[0][1]

                e = img[i + 1][j + 1] \* laplacian[1][1]

                f = img[i + 2][j + 1] \* laplacian[2][1]

                g = img[i][j + 2] \* laplacian[0][2]

                h = img[i + 1][j + 2] \* laplacian[1][2]

                z = img[i + 2][j + 2] \* laplacian[2][2]

                sum1[a] = (a1 + b + c + d + e + f + g + h + z)

                #sum1[a]=sum1[a]+90

                #sum1[a]=int(sum1[a]/(2.37))

                if(sum1[a] < 0):

                sum1[a] = 0

                elif(sum1[a] > 255):

                    sum1[a] = 255

                new[i + 1][j + 1] = sum1[a]

            if (sum1[a] > max1):

                max1 = sum1[a]

            if (sum1[a] < min1):

                min1 = sum1[a]

            a = a + 1

    return new

    print(max1, min1)

shape=mySharpening(img)

cv2.imwrite("greylap.png",shape)

cv2.imshow('foto',shape)

cv2.waitKey()

**4)**

The photo to be used is given below. It is tried to prevent the noises by processing this noisy photo.



**Figure 4.1**



****

**Figure 4.2**

Figure 4.2 show that **Median filter**, one of the most suitable filters for salt & pepper noises, has been applied. We look at the NxN region (kernel connected to you) around each pixel and the median value of these numbers is selected. Since the isolated pixels, in this case, have high values, taking the median value saves these values ​​and flattens our image. As it will appear in the code, a ready function for Median is used. In this function, the median value of the sequentially sorted pixel values ​​is taken.

output = cv2.medianBlur(image, ksize=3)



****

**Figure 4.3**

In Figure 4.3, the **Gaussian filter** is applied. However, it was observed that this filter was not sufficient to lose the Pepper appearance. The following line of code is used for Gaussian. (5,5) the more we enlarge the part determined, the more blurred.

img\_gaussian = cv2.GaussianBlur(image, (5,5), 0)





**Figure 4.4**

**Mean filter** is applied in Figure 4.4. This filter is obtained by averaging the pixels and the neighboring pixels. This filter alone was not able to remove all the noise. The code is given below.

def MeanFilter(image):

    boy= int(image.shape[0])

    en = int(image.shape[1])

    image1=np.zeros((boy,en,1),np.uint8)

    i=0

    for x in range (1,boy-1,1):

        j=-1

        for y in range(1,en-1,1):

            j=j+1

            image1[(i,j)]=(int(image[(x,y)])/9+int(image[(x+1,y)])/9+\

                    int(image[(x-1,y)])/9+int(image[(x,y+1)])/9+\

                    int(image[(x+1,y+1)])/9+int(image[(x-1,y+1)])/9+\

                    int(image[(x-1,y-1)])/9+int(image[(x,y-1)])/9+\

                    int(image[(x+1,y-1)])/9)

        i=i+1

    return image1

mean=MeanFilter(image)





**Figure 4.5**

A **bilateral filter** is a nonlinear, edge-protecting and noise-reducing smoothing filter for images. Replaces the density of each pixel with the average density values ​​it calculates from nearby pixels. It functions similar to Gaussian. However, as seen in the figure, the elephant image is in the background and the noise is in the foreground. This is almost the opposite of what we want, an erroneous image. The code is as shown below.

img\_bilateral = cv2.bilateralFilter(image, 13, 70, 50)





**Figure 4.6**

In this figure, **Blur** is used. It was tried to get rid of the Salt & Pepper image by losing the photograph's clarity. This loss of clarity eliminated noise, but the elephant lost clarity, so this filter was not considered suitable for these errors. Blur operation was carried out in two different ways below. The two perform the same function.

kernel = np.ones((5,5),np.float32)/25

blur = cv2.filter2D(image,-1,kernel)

**or**

blur= cv2.blur(image,(5,5))





**Figure 4.7**

As a result, **Median**, **Mean** and **Gaussian** filters gave the best results when looking at the best results out of 5 filters. Therefore, when a combination of these filters is made, the cleanest image will appear. For this, the code blocks are linked together and made as follows:

output = cv2.medianBlur(image, ksize=3)

img\_gaussian = cv2.GaussianBlur(output, (5,5), 0)

mean=MeanFilter(img\_gaussian)

All of the codes used for these filtering processes are shown below.

import cv2

import numpy as np

def MeanFilter(image):

    boy= int(image.shape[0])

    en = int(image.shape[1])

    image1=np.zeros((boy,en,1),np.uint8)

    i=0

    for x in range (1,boy-1,1):

        j=-1

        for y in range(1,en-1,1):

            j=j+1

            image1[(i,j)]=(int(image[(x,y)])/9+int(image[(x+1,y)])/9+\

                    int(image[(x-1,y)])/9+int(image[(x,y+1)])/9+\

                    int(image[(x+1,y+1)])/9+int(image[(x-1,y+1)])/9+\

                    int(image[(x-1,y-1)])/9+int(image[(x,y-1)])/9+\

                    int(image[(x+1,y-1)])/9)

        i=i+1

    return image1

image = cv2.imread("resim.jpg",0)

output = cv2.medianBlur(image, ksize=3)

img\_gaussian = cv2.GaussianBlur(output, (5,5), 0)

mean=MeanFilter(img\_gaussian)

img\_bilateral = cv2.bilateralFilter(image, 13, 70, 50)

blur= cv2.blur(image,(5,5))

#kernel = np.ones((5,5),np.float32)/25

#blur = cv2.filter2D(image,-1,kernel)

cv2.imshow('RealImage',image)

cv2.imshow('Combined',mean)

cv2.waitKey(0)