ITMO University

Image Processing:
Lab3

Prepared by

Bassel Alshawareb

April 15, 2023

1. Introduction

In this lab, we are applying multiple types of noises on an image, and applying filters to restore the original image. The noises we are considering in this lab are the gaussian, salt and pepper, Poisson, and peckle. For removing these noises, we are applying multiple filters and checking the performance of each. The filters we are applying are gaussian, and counter harmonic. On the other hand, we apply non-linear filters including, Weiner, median, and the adaptive median. It is worth mentioning that the adaptive median algorithm was implemented from scratch.

2. Applying noise

The image chosen in this lab is an image of coins, with clear distinct values and white background. This give good contrast and help to see the effect of noise on the image.

Figure 1 a) Image of coins. b) Histogram of the image.

Figure 2 Applying noise on image. a) Gaussian noise. b) Salt and Pepper. c) Speckle. d) Poisson.

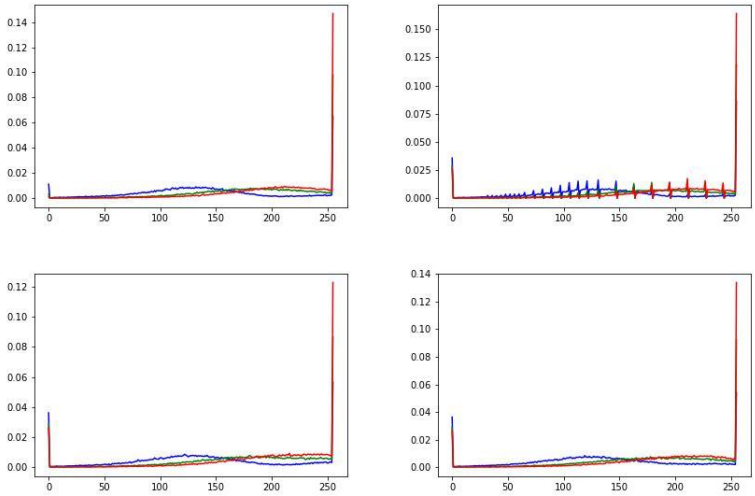

Figure 3 Histogram of the distorted images. a) Gaussian noise. b) Salt and Pepper. c) Peckle. d) Poisson.

3. Low-pass filters

3.1. Removing Salt and Pepper:

Recalling the image on which we applied salt and pepper noise

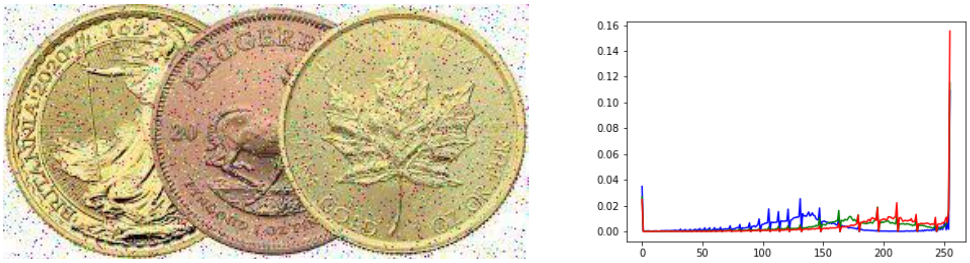

Figure 4 Coins with Salt and Pepper noise. a) The coins with noise. b) the histogram of the noised image.

3.1.1. Counter Harmonic filter:

Figure 5 The coins with pepper noise after applying the counter harmonic filer. Images from a-i) are the results of filtering with Q values range from -3 to 3 respectively.

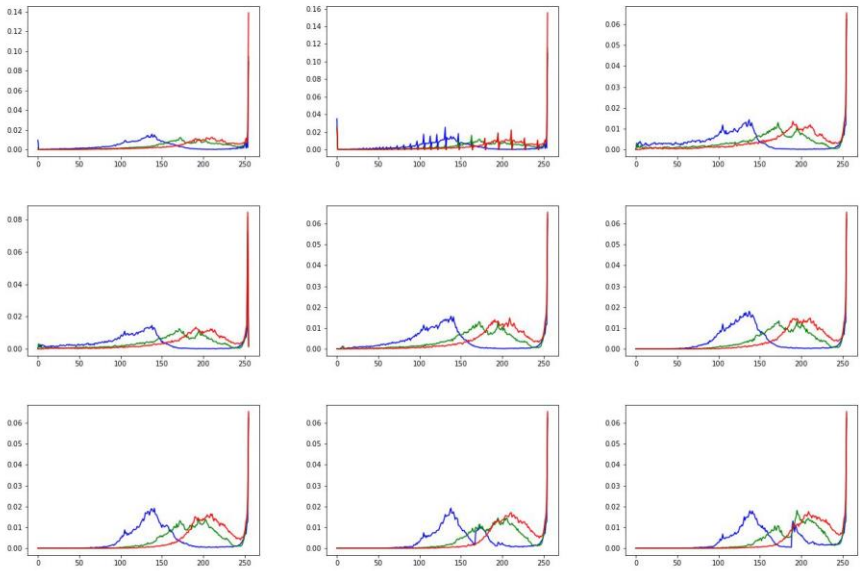

Figure 6 Histograms of the coins with pepper noise after applying the counter harmonic filer. A)The histogram of the original image. B) The histogram of the image with salt and pepper noise. Images from c-i) are the histograms of the filtered images with Q values range from -3 to 3 respectively.

We can see that with all values of Q from -3 to 3, the filter was able to remove the salt and pepper noise. It could be realized that with negative values of Q, the image gets sharper, while for positive values, the image gets higher values, and becomes brighter. For Q = 0, the counter harmonic mean works like an arithmetic mean. We can see that Q= {-2,-1, 0} have given the best restorations.

3.1.2. Using Gaussian filter

Figure 7 Applying gaussian filter on images with salt and pepper noise. a) the original image. b) The image with noise. C-F) The images after applying the gaussian filter with sigma values from 0.1 to 0.7 with 0.2-value increment.

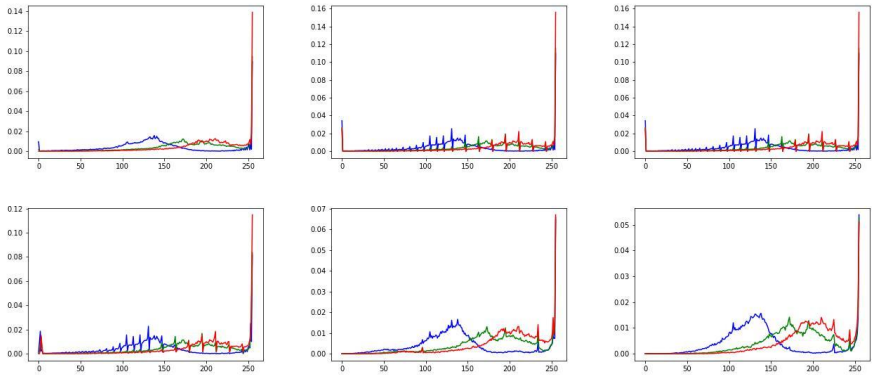

Figure 8 Histograms after applying gaussian filter on images with salt and pepper noise. a) the original image histogram. b) The histogram of image with noise. C-F) The histograms images after applying the gaussian filter with sigma values from 0.1 to 0.7 with 0.2-value increment.

We can see that we could almost restore the original image with a sigma value equals to 0.5.

3.2. Removing Gaussian Noise:

Recalling the image on after which are have applied the gaussian noise

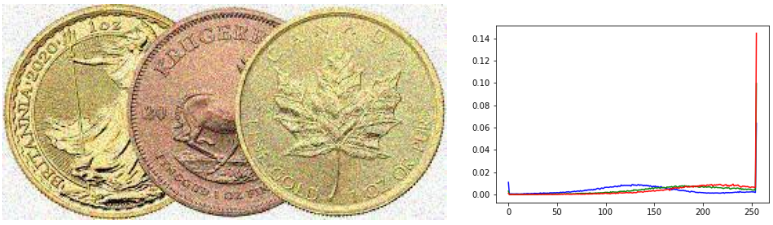

Figure 9 Image and histogram of coins with gaussian noise.

3.2.1. Using Counter Harmonic Filter

Figure 10 The coins with gaussian noise after applying the counter harmonic filer. Images from a-i) are the results of filtering with Q values range from -3 to 3 respectively.

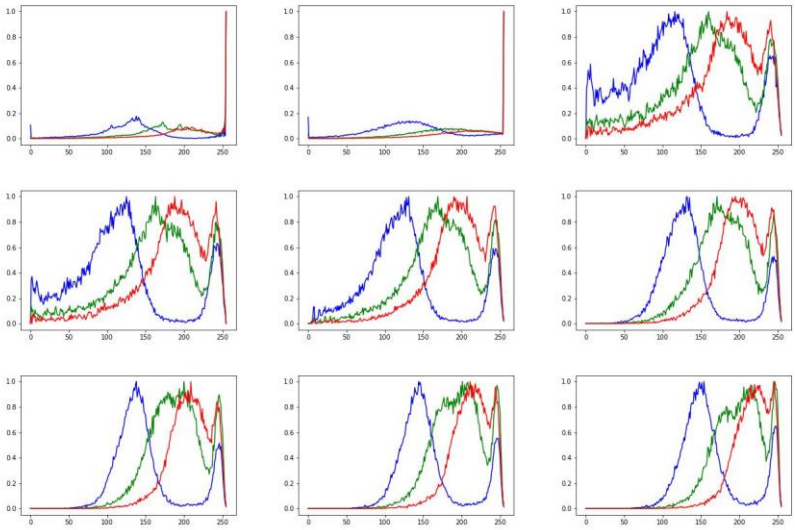

Figure 11 Histograms of the coins with gaussian noise after applying the counter harmonic filer. Images from a-i) are the histograms of the filtered images with Q values range from -3 to 3 respectively.

3.2.2. Using Gaussian filter

Figure 12 Applying gaussian filter on images with gaussian noise. a) the original image. b) The image with noise. C-f) The images after applying the gaussian filter with sigma values from 0.1 to 0.7 with 0.2-value increment.

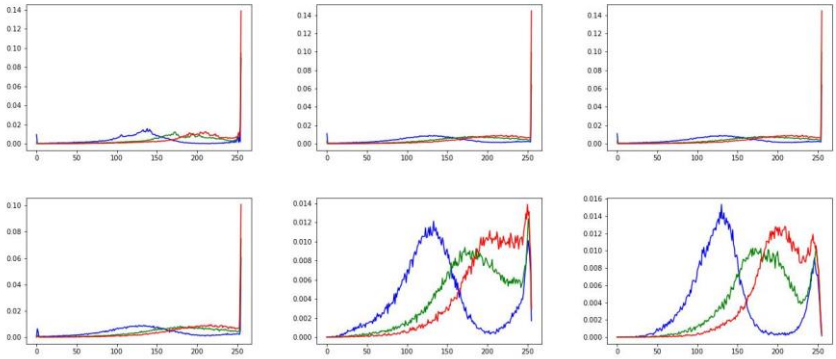

Figure 13 Histograms of the images after applying gaussian filter on images with gaussian noise. a) The histogram of the original image. b) The histogram of the image with noise. C-f) The histograms of the images after applying the gaussian filter with sigma values from 0.1 to 0.7 with 0.2-value increment.

3.3. Removing Poisson noise

3.3.1. Using Counter Harmonic filter

Figure 14 Applying counter harmonic filter on image with Poisson noise.

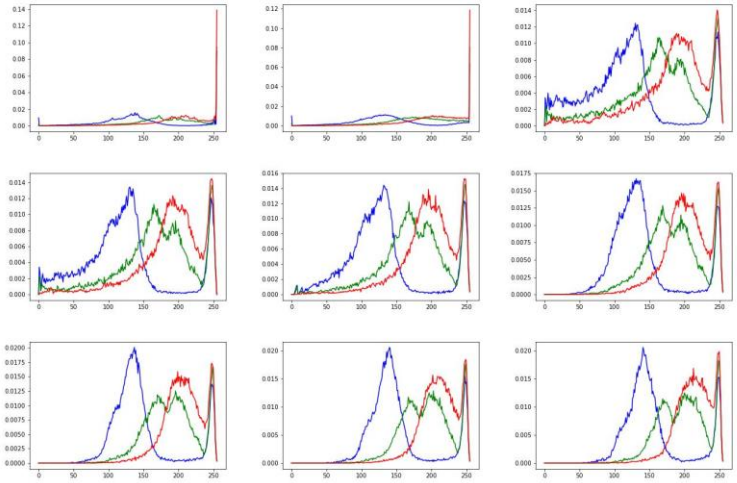

Figure 15 Histogram of images after applying counter harmonic filter on image with Poisson noise.

3.4. Removing Speckle noise

3.4.1. Using Counter Harmonic filter

Figure 16 Applying counter harmonic filter on images with speckle noise.

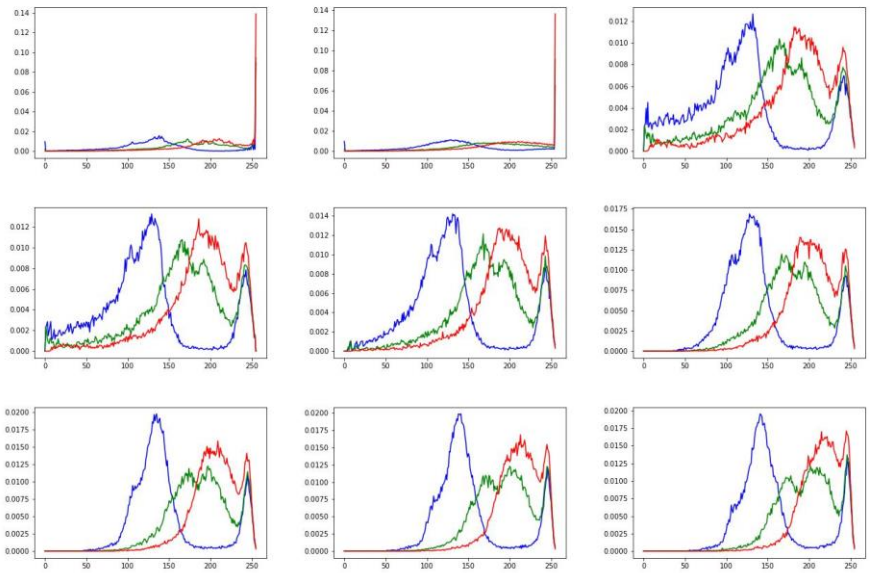

Figure 17 Histograms of images after applying counter harmonic filter on images with speckle noise.

4. Non-linear filters:

4.1.1. **Adaptive median filter:**

The algorithm was implemented from scratch. Here we discuss 2 variants of the algorithm. When the expanded window reaches the maximum size, we consider the case of returning the current value (as is suggested in the task description) and the case of returning the median value. For simplicity of explanation, we will refer to the algorithm when returns the current value as algorithm one, when the one which return the median value as algorithm 2.

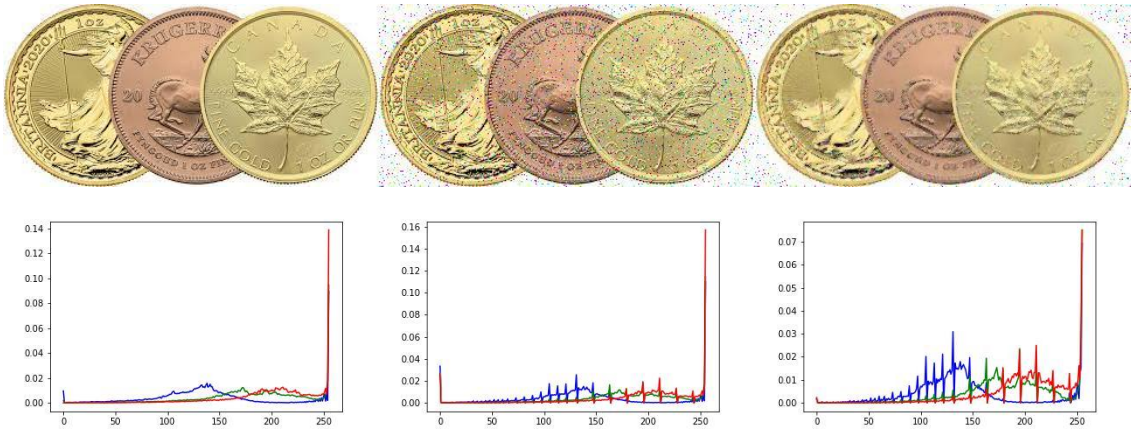

Figure 18 The adaptive median filter (algorithm 1) applied on image with salt and pepper noise. A) The original image. B) The image with salt and pepper noise. C) the resulting image. D-G) show the histograms of these images respectively.

We can see that algorithm 1 does very well removing salt and pepper noise on the coins. However, it doesn't remove the noise from the background, and this is due to the fact that the background has only one color-intensity. This means, that when the window is expanding, it will keep expanding until it reaches size limit, because as long as there is only 2 pixel intensity values, the window will keep expanding.
Now let's see the results of algorithm 2.

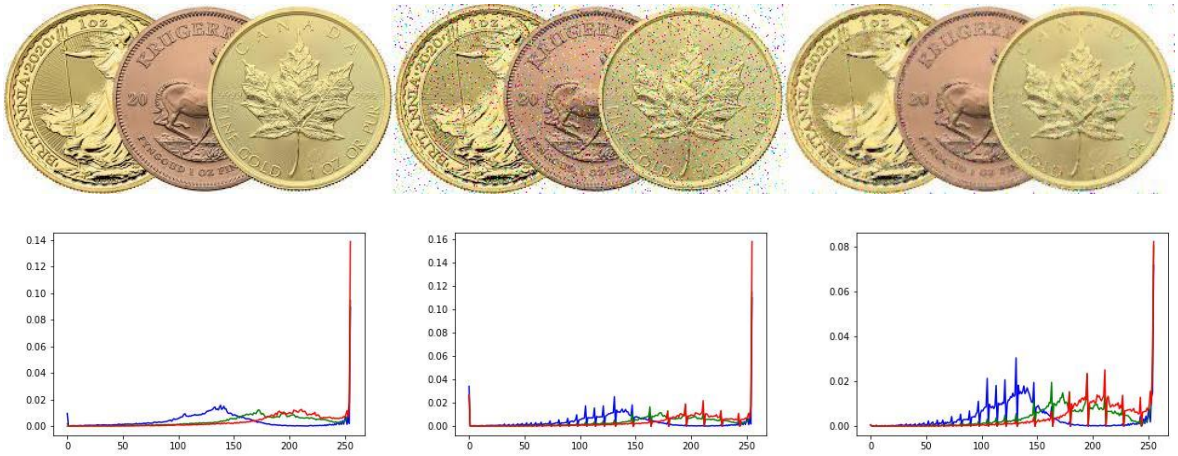

Figure 19 a) The original image. b) The image with salt and pepper noise. c) The restored image after applying the adaptive median filter (algorithm 2). Followed are the histograms of the images respectively.

Although the histogram of the resulting image doesn't really differ from the one obtained by algorithm 1, we can see clearly, that algorithm 2 has dealt with the salt and pepper noise applied on the background. In other words, algorithm 2 converts the pepper noise to white points to fit with the background. We can see this affect on the resulting histogram.

4.1.2. Geometric Mean:

4.1.2.1. On Gaussian noise:

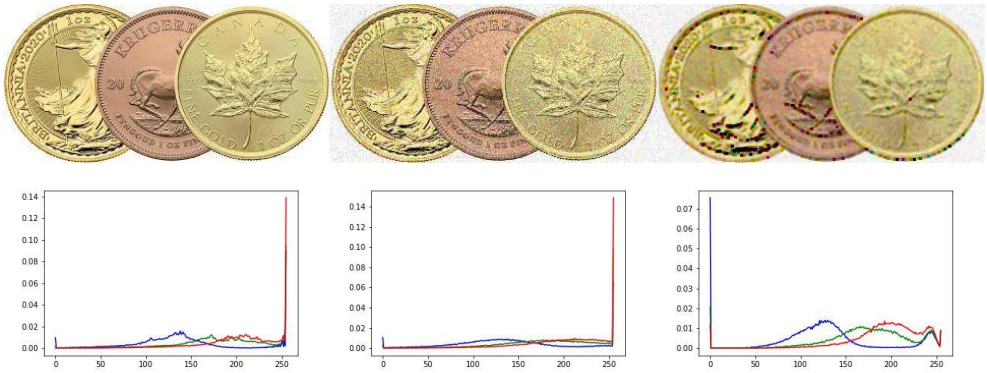

Figure 20 Applying geometric mean on Image with gaussian noise. A) Original Image. B) Image with Gaussian noise. C) Image after applying the gaussian filter

1.1.1.1. On salt and pepper noise:

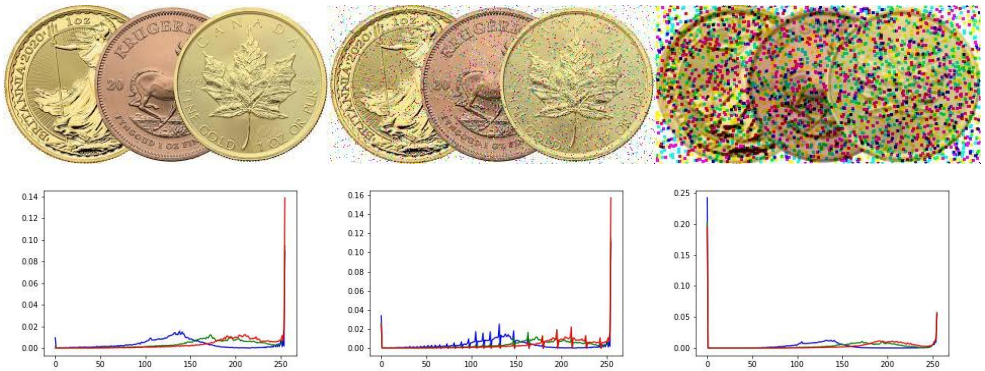

Figure 21 Applying geometric mean on image with salt and pepper noise.

We can see that applying the geometric filter on an image with salt and pepper noise gives very bad results. This is due to the effect of outliers on the multiplication operator. Multiplying by zeros gets all the values multiplied by it to zero. This mean that whenever there is a pepper noise, all the neighbors will go to zero.

4.1.3. Wiener filter:

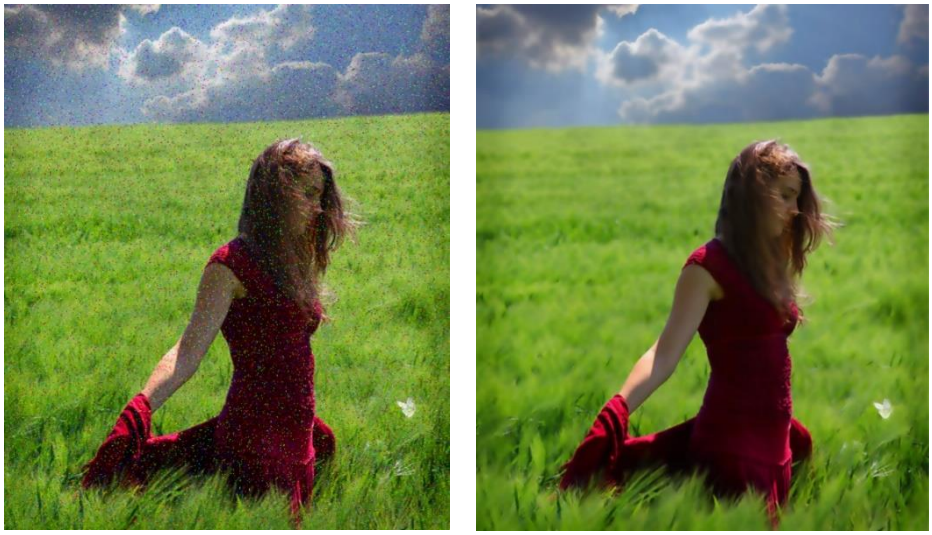

Figure 22 Applying Weiner filter on image with salt and pepper noise. A) The image with salt and pepper noise.. B) The image after applying Weiner filter.

We experience the effectiveness of Wiener filter in removing noise. We chose noise of type salt and pepper. We can realize that the filter could remove all the noise.

Wiener filter was implemented from scratch. The algorithm starts by padding the image with the half of the width and height of the kernel. Then we calculate the mean and the variance and calculate the resulting image according to weiner equations.

Code:

```
def wiener(I):
    I=cv2.imread("R.jfif")
    K=7
    kernel=np.ones((K,K), dtype=np.float64)
    pad=int((K-1)/2)
    I_copy=cv2.copyMakeBorder(I, bottom=pad, top=pad,
right=pad, left=pad, borderType=cv2.BORDER_REPLICATE)
    rows=I.shape[0]
    cols=I.shape[1]
    imgs=[]

    if I.shape[-1]==3:
        for k in range(3):
            m=np.zeros(I.shape[0:-1], dtype=np.float64)
            seg2=np.zeros(I.shape[0:-1], dtype=np.float64)
            for i in range(K):
                for j in range(K):
                    m=m+(I_copy[i:i+rows, j:j+cols,
k]).astype(np.float64)
                    seg2= seg2+ ((I_copy[i:i+rows,
j:j+cols,k]).astype(np.float64))**2

            seg2=seg2/(K**2)
            print("first",seg2)
            m=m/(K**2)
```

```
                print("THen",m)
                seg2=seg2-m**2
                v=np.sum(seg2)/(rows*cols)
                print("v",v)
                res=m+((seg2-v)/seg2)*(I[...,k]-m)
                print("res", res)
                plt.imshow(seg2<v, cmap="gray")
                f_res=np.where(seg2<v,m,res)
                imgs.append(f_res.astype(np.uint8))
        print(len(imgs))
        out=cv2.merge(imgs)
        return out
```

5. High-pass filters

5.1. Prewitt:

Figure 23 Perwitt filter.

Code:

```
@dec_filt
def Gx(I):
    kernel=np.array([[-1,0,1], [-1, 0, 1], [-1,0,1]])
    return cv2.filter2D(I,-1,kernel)

@dec_filt
def Gy(I):
    kernel=np.array([[-1,0,1], [-1, 0, 1], [-1,0,1]]).T
    return cv2.filter2D(I,-1,kernel)

gx=Gx(I)
```

```
gy=Gy(I)
res=gx+gy
```

5.2. Sobel

Figure 24 Sobel filter.

```
@dec_filt
def sobel_x(I):
    kernel=np.array([[-1,0,1], [-2, 0, 2], [-1,0,1]])
    return cv2.filter2D(I,-1,kernel)

@dec_filt
def sobel_y(I):
    kernel=np.array([[-1,0,1], [-2, 0, 2], [-1,0,1]]).T
    return cv2.filter2D(I,-1,kernel)

I=cv2.imread("coins.jpg")
sx=sobel_x(I)
sy=sobel_y(I)
res=sx+sy
```

5.3. Laplacian

Figure 25 Laplacian filter.

Code:

```
def L(I):
    kernel=np.array([[0,-1,0],[-1,4,-1],[0,-1,0]])
    return cv2.filter2D(I,-1,kernel)
```

5.4. Canny

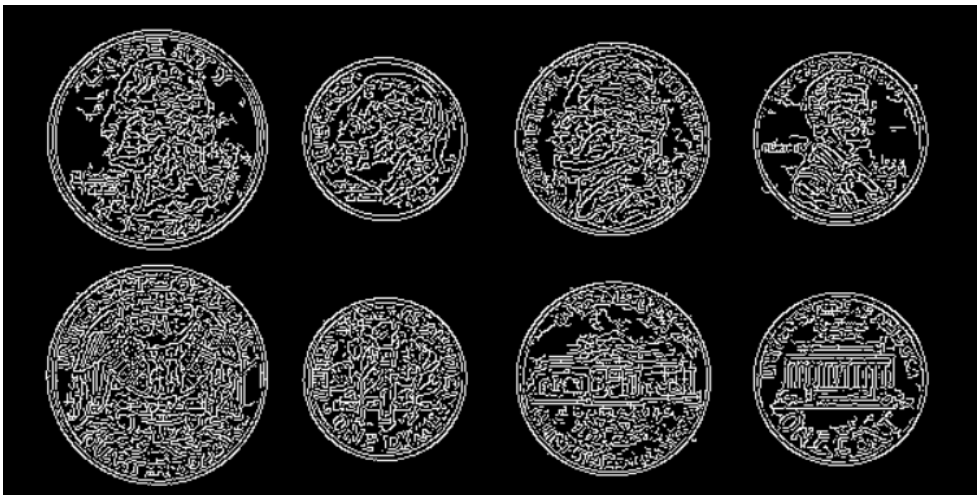

Figure 26 Canny filter.

```
res=cv2.Canny(I, 50,150)
```

Questions:

1. **What are the main disadvantages of adaptive image filtering methods?**
   The biggest disadvantage is that it is computationally expensive: Adaptive filtering is a non-convolutional processing, in the sense that it cannot be performed using the usual optimized convolution algorithms. The convolution process is optimized and could be applied with very high-performance using GPUs. The convolution is usually implemented as matrix multiplication. While in every adaptive filter, this property cannot be applied.
2. **For what values of the parameter Q will the counter harmonic filter work as an arithmetic filter, and for what values as a harmonic one?**
   When Q = 0: Arithmetic mean
   When Q = 1: Harmonic
3. **What operators can be used to detect edges in the image?**
   Any kernel that uses derivative could be used. Some examples are: Sobel, Laplacian, and Canny.
4. **Why, as a rule, is low-pass filtering performed at the first step of edge detection?**
   To remove noise: if there is salt and pepper noise, or in general any extreme value, then the derivative will give amplify the noise.

Complete code of the lab:

```
File: Histogram_processing
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 25 13:21:23 2023

@author: Bassel
"""

import cv2
import numpy as np
import matplotlib.pyplot as plt
import os

class Image:
    id_plot=0
    path_input = None
    path_output = None
    def __init__(self, img=None,histSize=256, histRange=(0,256),
CONFIG="BGR", EQUALIZE = True, pth = "default", sequence = True):
        self.img_on = None
        self.img_org = None
        self.sequence = sequence
        self.img_org_hist = None
        self.history = []
        self.history_hist = []
        self.history_hist_figure = []
        self.operations = []
        self.n_operations = 0
        if pth == "default":
            path = os.getcwd()
        else:
            path = pth
        Image.path_input = os.path.join(path,"inputs")
        Image.path_output = os.path.join(path , "outputs")
        self.last_executed = ""
        self.histSize=256
        self.EQUALIZE = EQUALIZE
        if CONFIG=="BGR":
            a=[0,1,2]
        else:
            a=[2,1,0]
        self.order=a
        Image.histSize=histSize
        Image.histRange=histRange
        if img is not None:
            self.set_img(img)
        else:
            self.img=None

    def copy_img(self):
        if self.sequence == False:
            return self.img_on.copy()
        else:
            return self.img.copy()
```

```
    def get_img(self):
        return self.img

    def set_img(self, img, text = None):
        self.n_operations = self.n_operations +1
        if img is None:
            return
        if text is not None:
            print(text)
        else:
            text = str(self.n_operations)
        if self.img_org is None:
            print("added origin")
            self.img_org = img
            self.img_on = img
        self.img = img
        self.rows, self.cols = self.img.shape[0:2]
        self.history.append(self.img)
        self.operations.append(text)
        self.calc()


    def on_current(self):
        self.img_on = self.img
        self.sequence = False
    def on_sequence(self):
        self.sequence = True

    def calc(self,img=None):
        print("Calculating Histogram")
        if img is None:
            if self.img is None:
                print("error")
                return
            img=self.img
        img_s=cv2.split(img)
        bHist=cv2.calcHist(img_s,[self.order[0]],None,
[self.histSize], (0, 256))
        gHist=cv2.calcHist(img_s,[self.order[1]],None,
[self.histSize], (0, 256))
        rHist=cv2.calcHist(img_s,[self.order[2]],None,
[self.histSize], (0, 256))
        self.img=img
        self.bH=bHist
        self.gH=gHist
        self.rH=rHist
        if self.EQUALIZE:
            self.equalize()
        else:
            self.history_hist.append((self.bH, self.gH, self.rH))

    def equalize(self):
        if self.last_executed == "":
            self.last_executed = "equalized"
        print("Equalizing")

        self.bH_not_normalized = self.bH
        self.gH_not_normalized = self.gH
```

```
            self.rH_not_normalized = self.rH
            max_b = np.sum(self.bH)
            max_g = np.sum(self.gH)
            max_r = np.sum(self.rH)
            self.bH = self.bH/max_b
            self.gH = self.gH/max_g
            self.rH = self.rH/max_r
            self.EQUALIZE = True
            self.history_hist.append((self.bH, self.gH, self.rH))
            if self.img_org_hist is None:
                self.img_org_hist = (self.bH, self.gH, self.rH)


        def show(self, image = "current", name=None):
            breakpoint()
            if image == "org":
                I = self.img_org

            else:
                I = self.img

            if name is None:
                name="number"+str(Image.id_plot)
            image_name = name + "_" + self.last_executed
            image_path = Image.path_output + "/" + "images"
            hist_path = Image.path_output + "/" + "Histograms"

            try:
                os.mkdir(image_path)
            except IOError:
                pass
            try:
                os.mkdir(hist_path)
            except IOError:
                pass

            self.__show_hist(self.history_hist[-1])
            plt.suptitle(name)
            #plt.savefig(hist_path + "/" + image_name  + "_Histogram" +
".png")
            plt.show()
            self.__show_img(I)
            #plt.imsave(image_path + "/" + image_name + ".jpg", I)

        def __show_img(self, I):
            Image.id_plot=Image.id_plot+1
            I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
            fig = plt.figure(Image.id_plot)

            plt.imshow(I)
            plt.show()
        def __show_hist(self, hist_tuple):
            bH, gH, rH = hist_tuple
            Image.id_plot=Image.id_plot+1
            fig = plt.figure(Image.id_plot)
            self.history_hist_figure.append(fig)
            t=range(256)
            plt.plot(t,bH, color="blue")
```

```
        plt.plot(t,gH, color="green")
        plt.plot(t,rH, color="red")

    def show_original(self):
        self.show(image = "org")

    def show_history(self):
        #Image.id_plot=Image.id_plot+1
        for i in range(len(self.history)):
            self.__show_hist(self.history_hist[i])
            plt.show()
            self.__show_img(self.history[i])
            plt.show()
    def save_history(self, folder = None):

        if len(self.operations) == 0:
            names = list(map(str,list(range(len(self.history)))))
        else:
            names = self.operations

        if folder is None:
            image_path = Image.path_output + "/" + "History"
        else:
            image_path = Image.path_output + "/" + "History/" +
folder

        try:
            os.mkdir(image_path)
        except IOError:
            pass
        ignor = 0
        length = len(self.history)-ignor
        rows, cols = self.history[0].shape[0:2]
        n_cols = 3
        n_rows = int(np.ceil(length/n_cols))

        mat = np.ones((rows*n_rows, cols*n_cols, 3 ), dtype =
np.uint8)*255
        hist_rows, hist_cols = (288, 432)
        hist_mat = np.ones((hist_rows*n_rows, hist_cols*n_cols, 3),
dtype = np.uint8)*255
        Image.id_plot=Image.id_plot+1
        j = 0
        k = 0

        for i in range(len(self.history)):
            plt.figure(self.history_hist_figure[i])
            plt.savefig(image_path  + "/"   + names[i] + "_Histogram"
+ ".png")
            hist = cv2.imread(image_path  + "/"   + names[i] +
"_Histogram" + ".png")
            hist = cv2.cvtColor(hist, cv2.COLOR_BGR2RGB)
            I = cv2.cvtColor(self.history[i], cv2.COLOR_BGR2RGB)
            plt.imsave(image_path + "/" + names[i] + ".jpg", I)
            if i>=ignor:
                mat[k*rows:rows*(k+1), j*cols:cols*(j+1),:] = I
                hist_mat[k*hist_rows:hist_rows*(k+1),
j*hist_cols:hist_cols*(j+1),:] = hist
```

```
                if j ==(n_cols-1):
                    j=-1
                    k = k+1
                j = j +1
        plt.imsave(image_path + "/ALL_images" + ".jpg", mat)
        plt.imsave(image_path + "/ALL_images_histogram" + ".jpg",
hist_mat)
        plt.imshow(mat)
        plt.imshow(hist_mat)

    def calc_error_hist(self):
        bH0, gH0, rH0 = self.img_org_hist
        for i in range(1,len(self.history)):
            bH, gH, rH = self.history_hist[i]
            Eb = np.sum((bH - bH0)**2)
            Eg = np.sum((gH- gH0)**2)
            Er = np.sum((rH- rH0)**2)
            Eavg = ( Eb + Eg + Er)/3
            print(Eavg)
def profile(img, x):
    return img[x,:]

def project_(img,xy):
    return np.sum(img,xy)/(img.shape[(xy+1)%2])



if __name__ == "__main__":
    pass
File: Filters
Code:
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 13 15:27:54 2023

@author: Bassel
"""
from Histogram_processing import Image
from skimage.util import random_noise
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt

class filters(Image):
    def __init__(self, img=None,histSize=256, histRange=(0,256),
CONFIG="BGR", EQUALIZE = True, pth = "default", sequence = True ):
        super().__init__(img ,histSize, histRange, CONFIG, EQUALIZE,
pth, sequence)
        self.window_size = 3

    def noise_saltnpepper(self):
        I = self.copy_img()
        I_noise = random_noise(I, mode = 's&p')
        I_noise = np.clip(255*I_noise, 0, 255)
        I_noise = np.asarray(I_noise, dtype = np.uint8)
        self.set_img(I_noise, "pepper_noise")
```

```
    def noise_gaussian(self):
        I = self.copy_img()
        I_noise = random_noise(I, mode = "gaussian")
        I_noise = np.clip(255*I_noise, 0, 255)
        I_noise = np.asarray(I_noise, dtype = np.uint8)
        self.set_img(I_noise, "gaussian_nosie")

    def noise_speckle(self):
        I = self.copy_img()
        I_noise = random_noise(I, mode = 'speckle')
        I_noise = np.clip(255*I_noise, 0, 255)
        I_noise = np.asarray(I_noise, dtype = np.uint8)
        self.set_img(I_noise, "speckle_nosie")

    def noise_poisson(self):
        I = self.copy_img()
        I_noise = random_noise(I, mode = 'poisson')
        I_noise = np.clip(255*I_noise, 0, 255)
        I_noise = np.asarray(I_noise, dtype = np.uint8)
        self.set_img(I_noise, "poisson_noise")

    def counterharmonic_mean_filter(self, Q=1):
        I = self.copy_img().astype(np.float64)
        size=(3,3)
        kernel = np.full(size, 1.0)

        num = np.power(I, Q + 1, where = I!=0)
        denum = np.power(I, Q, where = I!=0)

        denum_filtered = cv2.filter2D(denum, -1, kernel)
        num_filtered = cv2.filter2D(num, -1, kernel)

        result =  np.where(denum_filtered == 0, 0,
num_filtered/denum_filtered)
        Iout = np.asarray(result, dtype = np.uint8)
        self.set_img(Iout, "coutnerHarmonic" + "Q"+ str(Q))

    def gaussian_filter(self, sigma = 1):
        I = self.copy_img()
        size = 6*sigma + 1
        size = 7
        #size = 3
        Iout = cv2.GaussianBlur(I, (size,size), sigmaX = sigma,
sigmaY = sigma)
        self.set_img(Iout)

    def __adapt(self, I, i, j, max_size):
        size = 3
        #I = I.astype(np.float64)
        rows, cols = I.shape[0:2]
        med = 0
        while True:

            k = int((size-1)/2)
            if size >= max_size or (i-k)<0 or (i+k) >= rows or (j-k)
< 0 or (j+k) >= cols:
                return med
            crop= I[i-k:i+k+1, j-k:j+k+1]
```

```
            Imax = crop.max()
            Imin = crop.min()
            med = np.median(crop)
            A1 = med - Imin
            A2 = med - Imax
            if A1>0 and A2<0:
                Z1 = I[i,j] - Imin
                Z2 = I[i,j] - Imax
                if Z1>0 and Z2<0:
                    return I[i,j]
                else:
                    return med
            else:
                size = size + 2
    def adaptive_median(self):
        I = self.copy_img().astype(np.float64)
        Ib = I[...,0]
        Ig = I[...,1]
        Ir = I[...,2]
        IbOut = Ib
        IgOut = Ig
        IrOut = Ir
        max_size = 11
        half_w = int((max_size-1)/2)
        for i in range(1, I.shape[0] - 1,1):
            for j in range(1, I.shape[1] -1,1):
                print("i,j=", i, j)
                IbOut[i,j] = int(self.__adapt(Ib, i, j, max_size))
                IgOut[i,j] = int(self.__adapt(Ig, i, j, max_size))
                IrOut[i,j] = int(self.__adapt(Ir, i, j, max_size))
        IbOut = IbOut.astype(np.uint8)
        IgOut = IgOut.astype(np.uint8)
        IrOut = IrOut.astype(np.uint8)
        Iout = cv2.merge([IbOut, IgOut, IrOut])
        self.set_img(Iout, "adaptive")

    def __geo(self, I,r,i,j,size):
        mat = I[i-r:i+r+1, j-r:j+r+1]
        return int(255*np.power(np.prod(mat), 1/(size*size)))

    def geometric_mean(self):
        I = self.copy_img().astype(np.float64)
        Iout= I.copy().astype(np.uint8)
        I = I/255
        Ib = I[...,0]
        Ig = I[...,1]
        Ir = I[...,2]
        rows, cols = I.shape[0:2]
        size  = 3
        vl = 0
        r = int((size-1)/2)
        for i in range(r+vl, rows-r-vl):
            for j in range(r+vl, cols-r-vl):
                Iout[i,j,0] = self.__geo(Ib,r,i,j,size)
                Iout[i,j,1] = self.__geo(Ig,r,i,j,size)
                Iout[i,j,2] = self.__geo(Ir,r,i,j,size)
        Iout = Iout.astype(np.uint8)
        self.set_img(Iout, "Geometric_mean")
```

```
if __name__ == "__main__":
    path = os.getcwd()
    path_input = os.path.join(path,"inputs")
    path_output = os.path.join(path , "outputs")



    I=cv2.imread(path_input + '/money.jpg')
    ob = filters(I, sequence = True)

    ###Applying noise###
    n = 2
    if n == 1:
        ob.noise_gaussian()
    if n == 2:
        ob.noise_saltnpepper()
    if n == 3:
        ob.noise_poisson()
    if n == 4:
        ob.noise_speckle()

    ###Apply on the previous image###
    ob.on_current()

    ######Removing Noise######
    c = 5
    if c == 1:
        for Q in range(-3, 4):
            ob.counterharmonic_mean_filter(Q)
    if c == 2:
        for i in range(1, 8, 2):
            sigma = i/10
            ob.gaussian_filter(sigma)
    if c == 3:
        ob.adaptive_median()
    if c == 4:
        ob.geometric_mean()
    if c == 5:
        ob.wiener()

    ####plotting####
    ob.show_history()

    ob.save_history("adaptive_pepper")
```