Programming Assignment #2

Name: 林穎志 Student ID: 411551007

Introduction / Objectives:

The purpose of this program is to implement Canny Edge Detection to find the particular edge points in an image. One can observe the difference of results by applying different settings and experiences (filter, threshold… etc).

Method:

Noise reduction: Gaussian filter, Bilateral filter

Canny edge detection

Adaptation Threshold of Canny Edge Detection

Review of the methods I have implemented:

I. Noise reduction

1. Gaussian filter: A Gaussian has the advantage of being smooth in both the spatial and frequency domains.

2. Bilateral filter: A non-linear, edge-preserving, and noise-reducing smoothing filter for images, based on Gaussian distribution.

II. Sobel filter: A filter for calculating image gradients in horizontal and vertical direction to find the details of edge.

III. Non maximum suppression: Keep only the "strongest" edge points along the gradient direction.

IV. Double thresholding: Retain all the "strong" edge points as well as the "weak" edge points that are connected to the "strong" edge points.

3. Adaptive Threshold of Canny Edge Detection:

- Calculate the gradient and the max value(maxv) of a gray scale image
- Set a histogram of gradient image ranged in [0, maxv] and calculate the histogram.
- Set the ratio of non-edge pixels to the entire image pixels as Percent Of Pixels Not Edges.
- Total threshold = img.height * img.width * PercentOfPixelsNotEdges.
- Traverse the histogram, the number of pixels corresponding to each gradient value, and save the sum in a variable x.
- If x is greater than the value of total, exit the loop of hist traversal.
- Calculate lower and upper thresholds (High = 2.5*Low).
- Double Thresholding

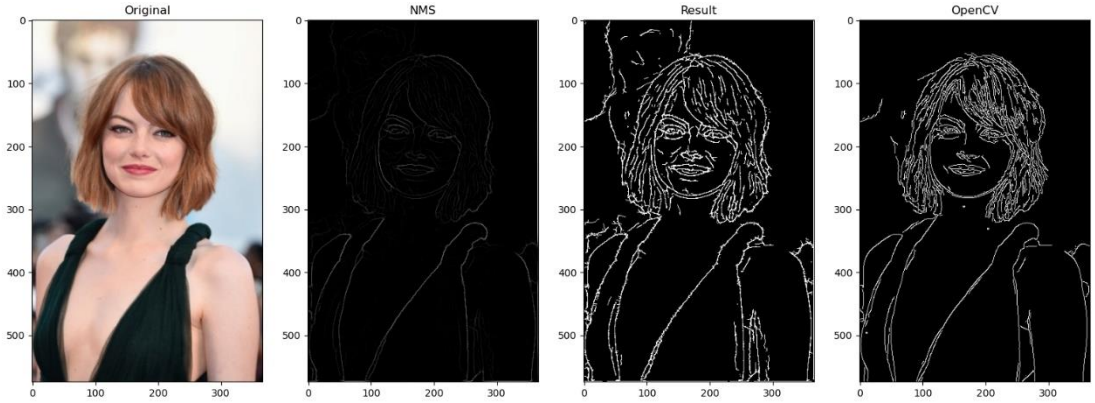
Picture 1:

I. Original Picture:

II. Implement Method: Canny Edge Detection, Bilateral Filter

III. Experiments & Results:

1. Implement and comparison



IV. Discussions:

The original picture is a RGB image. The result of non maximum suppression is the same image with thinner edges. Double threshold then links the strong edge points and weak edge points which is the neighbor of strong edge points. The comparison of OpenCV and our result shows that our implementation can retain more edges in the same criteria of thresholding.

V. Final Output:

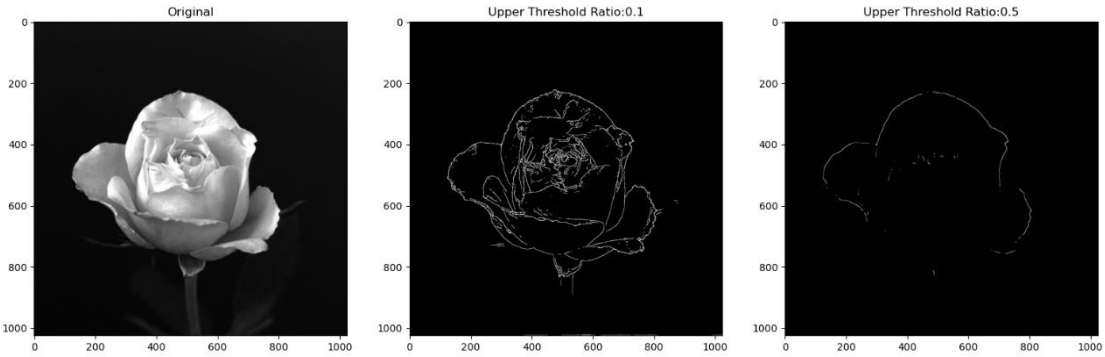
Picture 2:

I. Original Picture:

II. Implement Method: Canny Edge Detection, Bilateral Filter

III. Experiments & Results:

1. Experient different settings of upper threshold



IV. Discussions:

The original picture is a gray scale image. To test the performance of different upper threshold, we can observe that when the value of upper threshold becomes higher, the less the ‘strong’ edge points it can retain. The contour of rose become more blurred.

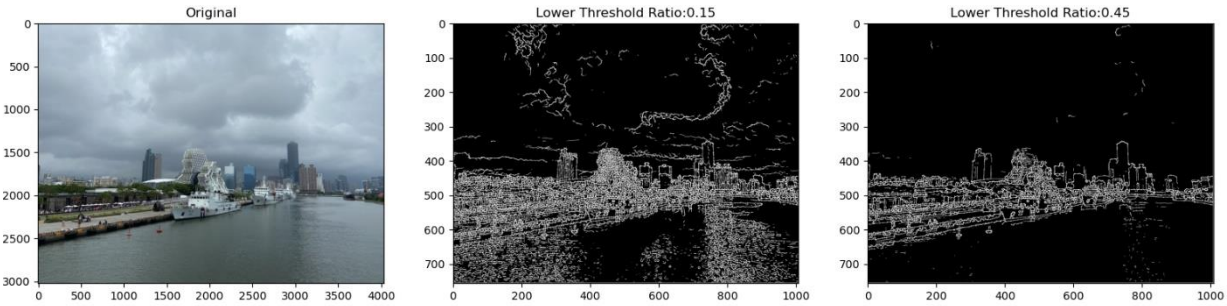
Picture 3:

I. Original Picture:

II. Implement Method: Canny Edge Detection, Bilateral Filter

III. Experiments & Results:

1. Experient different settings of lower threshold



IV. Discussions:

The original picture is a RGB image. To test the performance of different lower threshold, we can observe that when the value of lower threshold becomes higher, it can reduce more ‘weak’ edge points. The image has less edge points.

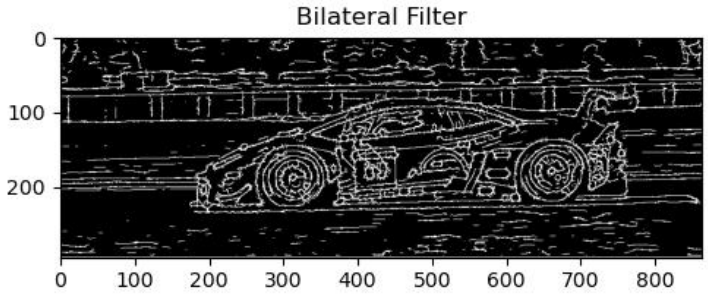
Picture 4:

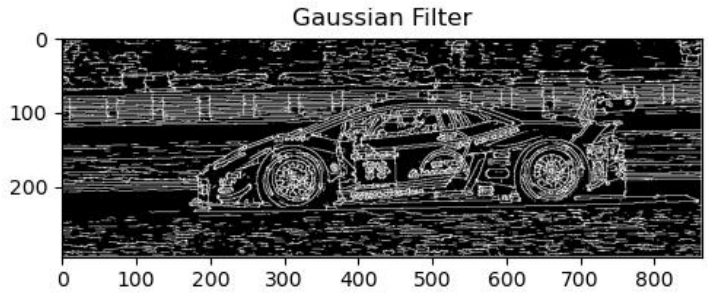
I. Original Picture:

II. Implement Method: Canny Edge Detection, Bilateral Filter, Gaussian Filter

III. Experiments & Results:

1. Experient different smoothing filters with the same criteria of upper and lower threshold





IV. Discussions:

The original picture is a RGB image. To test the performance of different smoothing filter, we can observe that the result of using Gaussian filter can retain more edge points than bilateral filter. The result of Canny edge detection using Gaussian filter is more detailed.

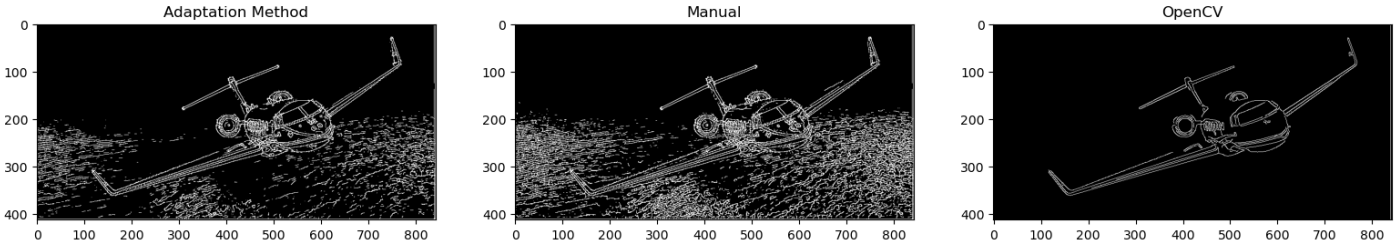
Picture 5:

I. Original Picture:

II. Implement Method: Adaptive threshold of Canny Edge Detection, Gaussian Filter

III. Experiments & Results:

1. Experient adaptive threshold of Canny Edge Detection



IV. Discussions:

The original picture is a RGB image. To test the performance of adaptive threshold of Canny Edge Detection, we can observe that the result of adaptive method had a better edge retain at the part of ocean. One can output a high performance of edge detection in the circumstance of do not need to set the upper and lower bound of threshold.

Code:

1. HW2.py

```
import cv2
import math
import numpy as np
import time
from scipy.signal import convolve2d as conv2
from matplotlib import pyplot as plt
from ProcFunc import *

class CannyEdgeDetection:
    def __init__(self, img, smooth_method = 'Bilateral',
                 threshold_mode = 'auto', l_threshold_ratio = 0.05,
u_threshold_ratio = 0.15):
        self.img = img
        #Obtain the time
        self.stime = time.time()
        self.angles = [0,45,90,135]
        self.nang = len(self.angles)
        self.smooth_method = smooth_method
        self.threshold_mode = threshold_mode
        #self.l_threshold = l_threshold
        #self.u_threshold = u_threshold
        self.highThreshold = u_threshold_ratio
        self.lowThreshold = l_threshold_ratio
        self.result = {}

    def deroGauss(self, w=5, s=1, angle=0):
        wlim = (w-1)/2
        y,x = np.meshgrid(np.arange(-wlim,wlim+1),np.arange(-wlim,wlim+1))
        G = np.exp(-
np.sum((np.square(x),np.square(y)),axis=0)/(2*np.float64(s)**2))
        G = G/np.sum(G)
        dGdx = -np.multiply(x,G)/np.float64(s)**2
        dGdy = -np.multiply(y,G)/np.float64(s)**2
        angle = angle*math.pi/180
        dog = math.cos(angle)*dGdx + math.sin(angle)*dGdy
```

```
        return dog

    def bilatfilt(self, I, w, sd, sr):
        dim = I.shape
        Iout= np.zeros(dim)
        wlim = (w-1)//2
        y,x = np.meshgrid(np.arange(-wlim,wlim+1),np.arange(-wlim,wlim+1))
        g = np.exp(-np.sum((np.square(x),np.square(y)),axis=0)/(2*(np.float64(sd)**2)))
        Ipad = np.pad(I,(wlim,),'edge')
        for r in range(wlim,dim[0]+wlim):
            for c in range(wlim,dim[1]+wlim):
                Ix = Ipad[r-wlim:r+wlim+1,c-wlim:c+wlim+1]
                s = np.exp(-np.square(Ix-Ipad[r,c])/(2*(np.float64(sr)**2)))
                k = np.multiply(g,s)
                Iout[r-wlim,c-wlim] = np.sum(np.multiply(k,Ix))/np.sum(k)
        return Iout

    # define function
    # Use the graussian filter to denoise
    def get_edges(self, I, sd):
        dim = I.shape
        Idog2d = np.zeros((self.nang,dim[0],dim[1]))
        for i in range(self.nang):
            dog2d = self.deroGauss(5,sd, self.angles[i])
            Idog2dtemp = abs(conv2(I,dog2d,mode='same',boundary='fill'))
            Idog2dtemp[Idog2dtemp<0]=0
            Idog2d[i,:,:] = Idog2dtemp
        return Idog2d

    # compute the gradient
    def calc_sigt(self, I,threshval):
        M,N = I.shape
        ulim = np.uint8(np.max(I))
        N1 = np.count_nonzero(I>threshval)
        N2 = np.count_nonzero(I<=threshval)
        w1 = np.float64(N1)/(M*N)
        w2 = np.float64(N2)/(M*N)
```

```
            try:
                u1 = sum(i*np.count_nonzero(np.multiply(I>i-0.5,I<=i+0.5))/N1 for i
in range(threshval+1,ulim))
                u2 = sum(i*np.count_nonzero(np.multiply(I>i-0.5,I<=i+0.5))/N2 for i
in range(threshval+1))

                uT = u1*w1+u2*w2
                sigt = w1*w2*(u1-u2)**2
            except:
                return 0
            return sigt

        # NMS
        def nonmaxsup(self, I,gradang):
            dim = I.shape
            Inms = np.zeros(dim)
            xshift = int(np.round(math.cos(gradang*np.pi/180)))
            yshift = int(np.round(math.sin(gradang*np.pi/180)))
            Ipad = np.pad(I,(1,),'constant',constant_values = (0,0))
            for r in range(1,dim[0]+1):
                for c in range(1,dim[1]+1):
                    maggrad = [Ipad[r-xshift,c-yshift],Ipad[r,c],Ipad[r+xshift,c+yshift]]
                    if Ipad[r,c] == np.max(maggrad):
                        Inms[r-1,c-1] = Ipad[r,c]
            return Inms

        # Obtain the best threshold
        def get_threshold(self, I):
            max_sigt = 0
            opt_t = 0
            ulim = np.uint8(np.max(I))
            print(ulim,)
            for t in range(ulim+1):
                sigt = self.calc_sigt(I,t)
                if sigt > max_sigt:
                    max_sigt = sigt
                    opt_t = t
            print ('optimal high threshold: ',opt_t,)
```

```
        return opt_t

# Threshold
def threshold(self, I, uth):
        if self.threshold_mode == 'auto':
                lth = uth/2.5
                Ith = np.zeros(I.shape)
                Ith[I>=uth] = 255
                Ith[I<lth] = 0
                Ith[np.multiply(I>=lth, I<uth)] = 100
        else:
                highThreshold = I.max() * self.highThreshold
                lowThreshold = highThreshold * self.lowThreshold

                Ith = np.zeros(I.shape)
                Ith[I >= highThreshold] = 255
                Ith[I < lowThreshold] = 0
                Ith[np.multiply(I>=lowThreshold, I<highThreshold)] = 100

        return Ith

# hysteresis
def hysteresis(self, I):
        r,c = I.shape
        Ipad = np.pad(I,(1,),'edge')
        c255 = np.count_nonzero(Ipad==255)
        imgchange = True
        for i in range(1,r+1):
                for j in range(1,c+1):
                        if Ipad[i,j] == 100:
                                if np.count_nonzero(Ipad[r-1:r+1,c-1:c+1]==255)>0:
                                        Ipad[i,j] = 255
                                else:
                                        Ipad[i,j] = 0
        Ih = Ipad[1:r+1,1:c+1]
        return Ih

def detector(self):
```

```
        self.result['Original'] = self.img
        # Resize the image
        while self.img.shape[0] > 1100 or self.img.shape[1] > 1100:
            self.img = cv2.resize(self.img,None, fx=0.5,fy=0.5,interpolation =
cv2.INTER_AREA)
        # translate into gray scale
        gimg = cv2.cvtColor(self.img,cv2.COLOR_BGR2GRAY)
        #Obtain the size of image
        dim = self.img.shape

        ## Start canny
        #Smoothing
        if self.smooth_method == 'Bilateral':
            #Bilateral filtering
            print ('Bilateral filtering...')
            gimg = self.bilatfilt(gimg, 5, 3, 10)
            print ('after bilat: ',np.max(gimg))
        elif self.smooth_method == 'Gaussain':
            print ('Gaussain filtering...')
            gimg = gs_filter(gimg)
            print ('after bilat: ',np.max(gimg))

        self.result['Smoothed'] = gimg

        #Gradient of Image
        print ('Calculating Gradient...')
        img_edges = self.get_edges(gimg,2)
        print ('after gradient: ',np.max(img_edges))

        #Non-max suppression：（NMS）
        print ('Suppressing Non-maximas...')
        for n in range(self.nang):
            img_edges[n,:,:] = self.nonmaxsup(img_edges[n,:,:], self.angles[n])

        print ('after nms: ', np.max(img_edges),)

        img_edge = np.max(img_edges,axis=0)
        lim = np.uint8(np.max(img_edge))
```

```
            self.result['NMS'] = img_edge

            # Compute the threshold
            print ('Calculating Threshold...')
            th = self.get_threshold(gimg)
            the = self.get_threshold(img_edge)

            # Obtain the best threshold
            print ('Thresholding...')
            img_edge = self.threshold(img_edge, the*0.25)

            print ('Applying Hysteresis...')
            img_edge = self.nonmaxsup(self.hysteresis(img_edge),90)

            print( 'Time taken :: ', str(time.time()-self.stime)+' seconds...')

            self.result['Result'] = img_edge
            # cv2.imwrite(img_name+'3'+'.jpg', img_edge)
            # cv2.imwrite(img_name+'4'+'.jpg', img_canny)

            return self.result


if __name__=='__main__':
        parser = argparse.ArgumentParser()
        parser.add_argument("-image1" , type = str , default='Face.jpg')
        parser.add_argument("-image2" , type = str , default='rose.tif')
        parser.add_argument("-image3" , type = str , default='Koa2.jpg')
        parser.add_argument("-image4" , type = str , default='car2.jpg')
        parser.add_argument("-image5" , type = str , default='jet.jpg')
        args = parser.parse_args()

        image = cv2.imread(args.image4)

        result = {}
        # #My implementation
        detector = CannyEdgeDetection(image, smooth_method = 'Gaussain',
```

```
                    threshold_mode = 'auto', l_threshold_ratio = 0.15,
u_threshold_ratio = 0.10)
    result_auto = detector.detector()
    result['Adaptation Method'] = result_auto['Result']

    #Manual
    detector = CannyEdgeDetection(image, smooth_method = 'Gaussain',
                    threshold_mode = 'Manual', l_threshold_ratio = 0.15,
u_threshold_ratio = 0.10)
    result_m = detector.detector()
    result['Manual'] = result_m['Result']

    # #OpenCV
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # blurred = cv2.medianBlur(gray, 11)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    # blurred = cv2.bilateralFilter(gray, 5, 3, 10)
    canny_blurred = cv2.Canny(blurred, 30, 150)
    result['OpenCV'] = canny_blurred

    show_all_img(result)
```

2. ProcFunc.py

```
from copy import copy
from pickletools import uint8
import cv2
import argparse
import random
import numpy as geek
import skimage.util.noise as noise
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import ndimage

#Resize
def img_resize(image, resize_height, resize_width):

    image_shape=np.shape(image)
    height=image_shape[0]
    width=image_shape[1]

    if (resize_height is None) and (resize_width is None):
        return image

    #Resize height
    if resize_height is not None:
        resize_width=int(width* (resize_height/height) )
    #Resize width
    elif resize_width is not None:
        resize_height=int(height* (resize_width/width) )

    img = cv2.resize(image, dsize=(resize_width, resize_height))

    return img

#RGB -> Gray
def RGB2GRAY(image):
    return cv2.cvtColor(image , cv2.COLOR_BGR2GRAY)
```

```
#Padding
def padding(image , kernel_size):

    padding_size = (kernel_size[0] // 2, kernel_size[1] // 2)
    height = image.shape[0]
    width = image.shape[1]
    output_h = height + padding_size[0] * 2
    output_w = width + padding_size[1] * 2
    output = np.zeros( (output_h , output_w), dtype="uint8")

    for i in range(height):
        for j in range(width):
            output[i + padding_size[0]][j + padding_size[1]] = image[i][j]

    return output

#Convolution
def convolution(image, kernel):

    kernel_size = kernel.shape
    output = np.zeros(image.shape, dtype = "uint8")
    pad_img = padding(image , kernel_size)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            temp = []
            result = 0
            temp.append( pad_img[ i:i+kernel_size[0], j:j+kernel_size[1] ] )
            image_data = np.row_stack(temp).flatten()
            kernel_data = np.row_stack(kernel).flatten()

            #Calculate
            for k in range(len(kernel_data)):
                result += image_data[k] * kernel_data[k]
            output[i][j] = result

    return output
```

```
#Sobel Filter
def sobel_filter(image):
    Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)
    Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)

    gx = convolution (image, Kx)
    gy = convolution (image, Ky)

    G = np.hypot(gx, gy)
    G = G / G.max() * 255
    theta = np.arctan2(gx, gy)

    return (G, theta)


#Smoothing
#Average
def box_filter(image, kernel_size = (3, 3)):

    all = kernel_size[0]* kernel_size[1]
    kernel = np.array([1]*all).reshape(kernel_size)

    avg_kernel = (1/ all)*kernel

    if len(image.shape) == 2:
        output = convolution (image, avg_kernel)
    else :
        image_r = image[:, :, 0]
        image_g = image[:, :, 1]
        image_b = image[:, :, 2]

        output_r = convolution (image_r , avg_kernel)
        output_g = convolution (image_g , avg_kernel)
        output_b = convolution (image_b , avg_kernel)
        output = np.dstack((output_r, output_g, output_b))

    return output
```

```
#Gaussian Filter
def gs_filter(image, kernel_size = (3, 3)):

    #Gassian filter
    x1 = 1 - math.ceil(kernel_size[0] / 2)
    x2 = math.ceil(kernel_size[0] / 2)
    y1 = 1 - math.ceil(kernel_size[1] / 2)
    y2 = math.ceil(kernel_size[1] / 2)

    x, y = np.mgrid[x1:x2, y1:y2]

    gaussian_kernel = np.exp(-(x**2+y**2))

    #Normalization
    gaussian_kernel = gaussian_kernel / gaussian_kernel.sum()

    if len(image.shape) == 2:
        output = convolution (image, gaussian_kernel)
    else :
        image_r = image[:, :, 0]
        image_g = image[:, :, 1]
        image_b = image[:, :, 2]

        output_r = convolution (image_r , gaussian_kernel)
        output_g = convolution (image_g , gaussian_kernel)
        output_b = convolution (image_b , gaussian_kernel)
        output = np.dstack((output_r, output_g, output_b))

    return output


#Median filter
def median_img_filter(image, kernel_size = (3, 3)):

    def median_filter(image, kernel_size):
        height = image.shape[0]
        width = image.shape[1]
```

```
            output = np.zeros((height, width), dtype = "uint8")

            #Padding
            padding_image = padding(image , kernel_size)

            #Find Median
            for i in range(height):
                for j in range(width):
                    temp = []
                    temp.append( padding_image[ i:i+kernel_size[0],
j:j+kernel_size[1] ] )
                    #Median
                    median = np.median(np.row_stack(temp).flatten())
                    output[i][j] = median
            return output

        if len(image.shape) == 2:
            output = median_filter(image, kernel_size)
        else:
            image_r = image[:, :, 0]
            image_g = image[:, :, 1]
            image_b = image[:, :, 2]

            output_r = median_filter(image_r , kernel_size )
            output_g = median_filter(image_g , kernel_size )
            output_b = median_filter(image_b , kernel_size )
            output = np.dstack((output_r, output_g, output_b))

        return output

# Histogram Equalization
def hist_equa_img(image):

        def hist_equa(image):
            data = np.zeros(256).astype(np.int64)
            image_f = image.flatten()
            for i in image_f:
                data[int(i)] += 1
```

```
            p = data/image_f.size
            p_sum = geek.cumsum(p)
            equal = np.around(p_sum * 255).astype('uint8')
            output = equal[image_f].reshape(image.shape)

            return equal[image]

        if len(image.shape) == 2:
            output = hist_equa(image)
        else:
            image_r = image[:, :, 0]
            image_g = image[:, :, 1]
            image_b = image[:, :, 2]

            output_r = hist_equa(image_r )
            output_g = hist_equa(image_g )
            output_b = hist_equa(image_b )
            output = np.dstack((output_r, output_g, output_b))

            return output

#Bilateral Filter
def bilatfilt(I, w, sd, sr):
    dim = I.shape
    Iout= np.zeros(dim)
    wlim = (w-1)//2
    y,x = np.meshgrid(np.arange(-wlim,wlim+1),np.arange(-wlim,wlim+1))
    g = np.exp(-np.sum((np.square(x),np.square(y)),axis=0)/(2*(np.float64(sd)**2)))
    Ipad = np.pad(I,(wlim,),'edge')
    for r in range(wlim,dim[0]+wlim):
        for c in range(wlim,dim[1]+wlim):
            Ix = Ipad[r-wlim:r+wlim+1,c-wlim:c+wlim+1]
            s = np.exp(-np.square(Ix-Ipad[r,c])/(2*(np.float64(sr)**2)))
            k = np.multiply(g,s)
            Iout[r-wlim,c-wlim] = np.sum(np.multiply(k,Ix))/np.sum(k)
    return Iout
```

```
#Show Image
def show_all_img(result, columns = None, rows = 1):

    if columns is None:
        columns = len(result)
    fig = plt.figure(figsize=(20, 20))

    for i, img_n in enumerate(result):
        ax = fig.add_subplot(rows, columns, i+1)
        ax.title.set_text(img_n)
        # ax.set_title(i)
        img = result[img_n]
        if len(img.shape) == 2:
            plt.imshow(img, cmap='gray')
        else:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            plt.imshow(img)
    #plt.tight_layout()
    plt.show()
```