DIGITAL IMAGE PROCESSING

ASSIGNMENT - 2

Image Restoration

By : Prafful Mishra [RA2011003010802]
Subham Gayen [RA2011003010798]

Section : CSE - B2
Submitted To : Dr. K. Alice

Image Restoration Techniques:

Introduction:

Image restoration techniques refer to a set of methods and algorithms used to recover or improve the quality of an image that has been degraded or corrupted by noise, blur, or other factors. Some commonly used image restoration techniques:

Image Denoising: This technique is used to remove noise from an image without significantly affecting the image details. Methods used for image denoising include Gaussian filtering, median filtering, and wavelet-based denoising.

Image Deblurring: This technique is used to remove blur from an image caused by motion or other factors. Methods used for image deblurring include blind deconvolution, Wiener filtering, and Richardson-Lucy deconvolution.

Image Restoration using Deep Learning: This technique is used to restore an image using a deep neural network that is trained on a large dataset of degraded and corresponding clean images.

Image Super-Resolution: This technique is used to increase the resolution of an image. Methods used for image super-resolution include interpolation-based methods, multi-frame super-resolution, and deep learning-based methods.

Adaptive Filter:

An adaptive median filter is a type of image filtering technique used to remove noise from images. It is a non-linear filter that adjusts its filtering kernel size based on the local image statistics.

The idea behind the adaptive median filter is to calculate the median of the pixel values within a sliding window of increasing size. The filter starts with a small window and increases its size until a pixel within the window is found that satisfies a certain criterion.

CODE:

```
import numpy as np
import pandas as pd
from PIL import Image, ImageFilter
image_org = Image.open("originalpic.jpg")

def rgb2gray(rgb):
    if(len(rgb.shape) == 3):
        return np.uint8(np.dot(rgb[...,:3], [0.2989, 0.5870,
0.1140]))
    else:
        return rgb
image = np.array(image_org)
grayscale_image = rgb2gray(image)

def calculate_median(array):
    sorted_array = np.sort(array)
    median = sorted_array[len(array)//2]
    return median

def level_A(z_min, z_med, z_max, z_xy, S_xy, S_max):
    if(z_min < z_med < z_max):
        return level_B(z_min, z_med, z_max, z_xy, S_xy, S_max)
    else:
        S_xy += 2
        if(S_xy <= S_max):
            return level_A(z_min, z_med, z_max, z_xy, S_xy,
S_max)
        else:
            return z_med
```

```
def level_B(z_min, z_med, z_max, z_xy, S_xy, S_max):
    if(z_min < z_xy < z_max):
        return z_xy
    else:
        return z_med

def amf(image, initial_window, max_window):
    xlength, ylength = image.shape
    z_min, z_med, z_max, z_xy = 0, 0, 0, 0
    S_max = max_window
    S_xy = initial_window
    output_image = image.copy()
    for row in range(S_xy, xlength-S_xy-1):
        for col in range(S_xy, ylength-S_xy-1):
            filter_window = image[row - S_xy : row + S_xy + 1,
col - S_xy : col + S_xy + 1] #filter window
            target = filter_window.reshape(-1)
            z_min = np.min(target)
            z_max = np.max(target)
            z_med = calculate_median(target)
            z_xy = image[row, col]
            new_intensity = level_A(z_min, z_med, z_max, z_xy,
S_xy, S_max)
            output_image[row, col] = new_intensity
    return output_image

output = amf(grayscale_image, 3, 11)
Image.fromarray(output)

Image.fromarray(grayscale_image)
```

PROGRAM:

```python
import numpy as np
import pandas as pd
from PIL import Image, ImageFilter

image_org = Image.open("originalpic.jpg")

def rgb2gray(rgb):
    if(len(rgb.shape) == 3):
        return np.uint8(np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140]))
    else:
        return rgb

image = np.array(image_org)
grayscale_image = rgb2gray(image)

def calculate_median(array):
    sorted_array = np.sort(array)
    median = sorted_array[len(array)//2]
    return median

def level_A(z_min, z_med, z_max, z_xy, S_xy, S_max):
    if(z_min < z_med < z_max):
        return level_B(z_min, z_med, z_max, z_xy, S_xy, S_max)
    else:
        S_xy += 2
        if(S_xy <= S_max):
            return level_A(z_min, z_med, z_max, z_xy, S_xy, S_max)
        else:
            return z_med

def level_B(z_min, z_med, z_max, z_xy, S_xy, S_max):
    if(z_min < z_xy < z_max):
        return z_xy
    else:
        return z_med

def amf(image, initial_window, max_window):
    xlength, ylength = image.shape

    z_min, z_med, z_max, z_xy = 0, 0, 0, 0
    S_max = max_window
    S_xy = initial_window

    output_image = image.copy()

    for row in range(S_xy, xlength-S_xy-1):
        for col in range(S_xy, ylength-S_xy-1):
            filter_window = image[row - S_xy : row + S_xy + 1, col - S_xy : col + S_xy + 1] #filter window
            target = filter_window.reshape(-1)
            z_min = np.min(target)
            z_max = np.max(target)
            z_med = calculate_median(target)
            z_xy = image[row, col]
            new_intensity = level_A(z_min, z_med, z_max, z_xy, S_xy, S_max)
            output_image[row, col] = new_intensity
    return output_image

output = amf(grayscale_image, 3, 11)
Image.fromarray(output)

Image.fromarray(grayscale_image)
```

OUTPUT:

ORIGINAL

AFTER ADAPTIVE FILTER

Wiener Filter:

The Wiener filter is a linear filter used for signal processing and image restoration. It is used to estimate an unknown signal or image from a noisy observation of the signal. It works by minimizing the mean square error between the estimated signal and the true signal. The filter assumes that the noise and signal are both stationary and have a known statistical correlation.

Mathematically, the Wiener filter is defined as a linear combination of the noisy observation and a noise reduction filter. The noise reduction filter is designed to minimize the mean square error between the estimated signal and the true signal. The filter coefficients are determined using the autocorrelation function of the signal and noise.

The Wiener filter is commonly used in image processing and speech processing applications. It is also used in signal processing applications such as radar, sonar, and communications. The filter can be implemented in real-time systems and is computationally efficient.

CODE:

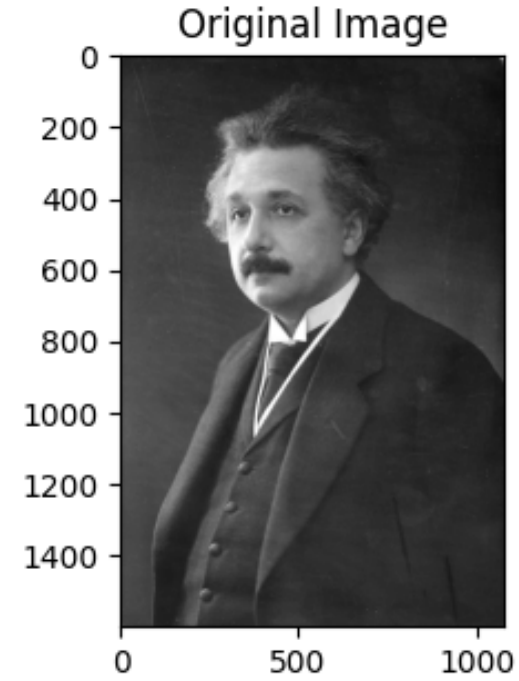
```
import os
import numpy as np
from numpy.fft import fft2, ifft2
from scipy.signal import gaussian, convolve2d
import matplotlib.pyplot as plt

def blur(img, kernel_size = 3):
     dummy = np.copy(img)
     h = np.eye(kernel_size) / kernel_size
     dummy = convolve2d(dummy, h, mode = 'valid')
     return dummy

def add_gaussian_noise(img, sigma):
     gauss = np.random.normal(0, sigma, np.shape(img))
     noisy_img = img + gauss
     noisy_img[noisy_img < 0] = 0
```

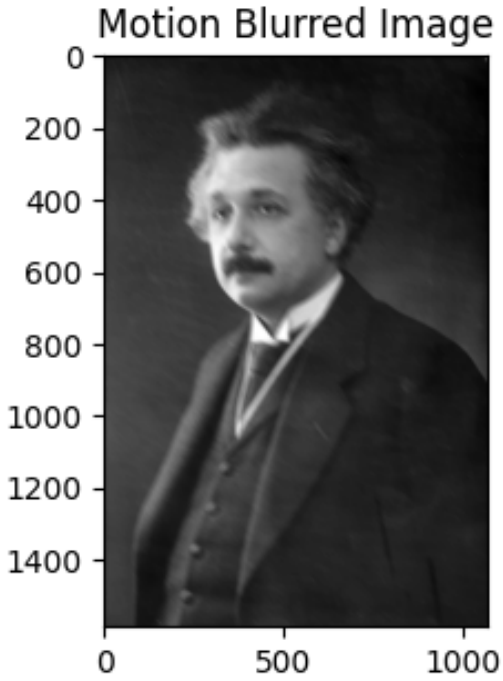
```
     noisy_img[noisy_img > 255] = 255
     return noisy_img

def wiener_filter(img, kernel, K):
     kernel /= np.sum(kernel)
     dummy = np.copy(img)
     dummy = fft2(dummy)
     kernel = fft2(kernel, s = img.shape)
     kernel = np.conj(kernel) / (np.abs(kernel) ** 2 + K)
     dummy = dummy * kernel
     dummy = np.abs(ifft2(dummy))
     return dummy

def gaussian_kernel(kernel_size = 3):
     h = gaussian(kernel_size, kernel_size /
3).reshape(kernel_size, 1)
     h = np.dot(h, h.transpose())
     h /= np.sum(h)
     return h

def rgb2gray(rgb):
     return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])


if __name__ == '__main__':
     file_name = os.path.join('AE.jpg')
     img = rgb2gray(plt.imread(file_name))
     blurred_img = blur(img, kernel_size = 15)
     noisy_img = add_gaussian_noise(blurred_img, sigma = 20)
     kernel = gaussian_kernel(3)
     filtered_img = wiener_filter(noisy_img, kernel, K = 10)
     display = [img, blurred_img, noisy_img, filtered_img]
     label = ['Original Image', 'Motion Blurred Image', 'Motion
Blurring + Gaussian Noise', 'Wiener Filter applied']

     fig = plt.figure(figsize=(12, 10))

     for i in range(len(display)):
          fig.add_subplot(2, 2, i+1)
          plt.imshow(display[i], cmap = 'gray')
          plt.title(label[i])

     plt.show()
```

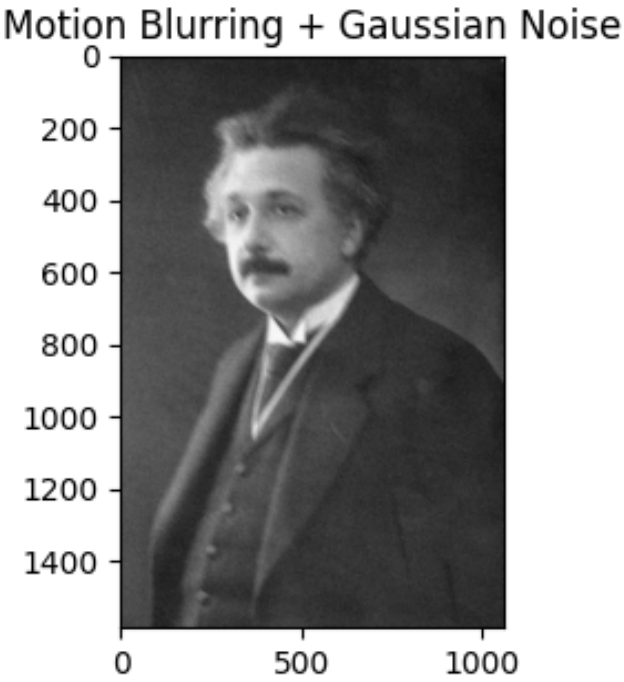
PROGRAM:

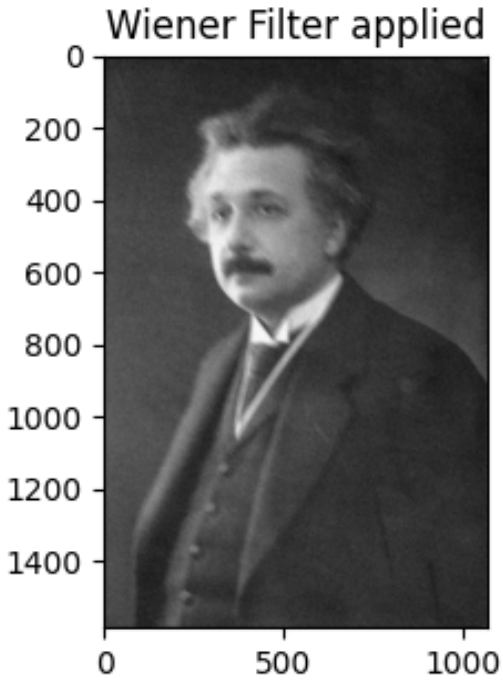
```python
import os
import numpy as np
from numpy.fft import fft2, ifft2
from scipy.signal import gaussian, convolve2d
import matplotlib.pyplot as plt

def blur(img, kernel_size = 3):
    dummy = np.copy(img)
    h = np.eye(kernel_size) / kernel_size
    dummy = convolve2d(dummy, h, mode = 'valid')
    return dummy

def add_gaussian_noise(img, sigma):
    gauss = np.random.normal(0, sigma, np.shape(img))
    noisy_img = img + gauss
    noisy_img[noisy_img < 0] = 0
    noisy_img[noisy_img > 255] = 255
    return noisy_img

def wiener_filter(img, kernel, K):
    kernel /= np.sum(kernel)
    dummy = np.copy(img)
    dummy = fft2(dummy)
    kernel = fft2(kernel, s = img.shape)
    kernel = np.conj(kernel) / (np.abs(kernel) ** 2 + K)
    dummy = dummy * kernel
    dummy = np.abs(ifft2(dummy))
    return dummy

def gaussian_kernel(kernel_size = 3):
    h = gaussian(kernel_size, kernel_size / 3).reshape(kernel_size, 1)
    h = np.dot(h, h.transpose())
    h /= np.sum(h)
    return h

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])


if __name__ == '__main__':
    # Load image and convert it to gray scale
    file_name = os.path.join('AE.jpg')
    img = rgb2gray(plt.imread(file_name))

    # Blur the image
    blurred_img = blur(img, kernel_size = 15)

    # Add Gaussian noise
    noisy_img = add_gaussian_noise(blurred_img, sigma = 20)

    # Apply Wiener Filter
    kernel = gaussian_kernel(3)
    filtered_img = wiener_filter(noisy_img, kernel, K = 10)

    # Display results
    display = [img, blurred_img, noisy_img, filtered_img]
    label = ['Original Image', 'Motion Blurred Image', 'Motion Blurring + Gaussian Noise', 'Wiener Filter applied']

    fig = plt.figure(figsize=(12, 10))

    for i in range(len(display)):
        fig.add_subplot(2, 2, i+1)
        plt.imshow(display[i], cmap = 'gray')
        plt.title(label[i])

    plt.show()
```

OUTPUT:


Original Image
0
200
400
600
800
1000
1200
1400
0
500
1000


Motion Blurred Image
0
200
400
600
800
1000
1200
1400
0
500
1000


Motion Blurring + Gaussian Noise
0
200
400
600
800
1000
1200
1400
0
500
1000


Wiener Filter applied
0
200
400
600
800
1000
1200
1400
0
500
1000

Canny Edge Detector:
The Canny edge detector is an edge detection algorithm that is widely used in computer vision and image processing. It works by detecting edges in an image based on the intensity gradient of the image. The algorithm has several stages:

Smoothing: The input image is convolved with a Gaussian filter to remove noise.

Gradient computation: The gradient of the image is calculated using a Sobel operator, which is a 3x3 filter that approximates the first derivative of the image.

Non-maximum suppression: The gradient magnitude is calculated and the edge points are identified as the local maxima along the gradient direction.

Double thresholding: The edge points are classified as strong, weak, or non-edges based on their gradient magnitude. A high threshold is used to classify strong edges, while a low threshold is used to classify weak edges. Non-edges are discarded.

Edge tracking by hysteresis: Weak edges that are connected to strong edges are promoted to strong edges, while weak edges that are not connected to strong edges are discarded.

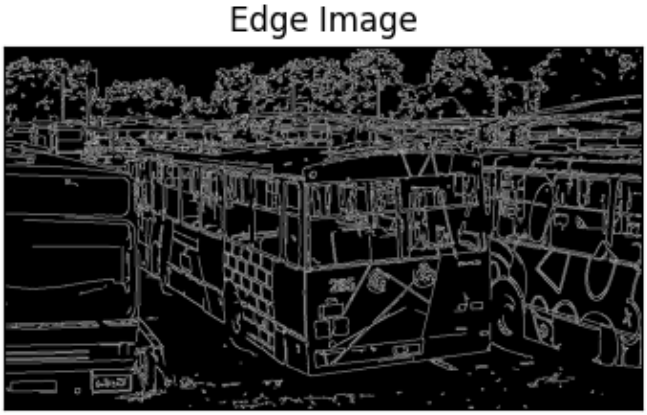
CODE:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('originalpic.jpg')

edges = cv2.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

PROGRAM:

```
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  img = cv2.imread('originalpic.jpg')
6
7  edges = cv2.Canny(img,100,200)
8
9  plt.subplot(121),plt.imshow(img,cmap = 'gray')
10 plt.title('Original Image'), plt.xticks([]), plt.yticks([])
11 plt.subplot(122),plt.imshow(edges,cmap = 'gray')
12 plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
13 plt.show()
```

OUTPUT:



RESULT:

We used image degradation techniques and noise filter techniques to initiate the restoration process and applied the required filters such as Adaptive Median Filter, Wiener Filter and Canny Edge Detection.

Hence, the image restoration process was successfully executed on the input image using Python code.