Image and Video Processing Lab
Lab 3:Image Filtering in Spatial Domain

Aruna Shaju Kollannur (SC21M111)

Sub: Image and Video Processing Lab
Department: Avionics(DSP)
Date of Lab sheet: September 28, 2021
Date of Submission: October 4, 2021

Question 1: Implement a program for image convolution and correlation using a square convolution mask of any odd size.(3, 5, etc.).

Aim

To perform Convolution and Correlation on image using given mask

Discussion

The Convolution and Correlation of an image is is the process of transforming an image by applying a kernel over each pixel and its local neighbors across the entire image. They are used to modify the spatial frequency characteristics of an image. The difference between convolution and correlation is that in convolution, the kernel mask is mirrored.

Algorithm

```
Step 1: Start
Step 2: Read the image and convert in into gray-scale
Step 3: Obtain the Correlation mask.
Step 4: Mirror the Correlation mask to obtain the Convolution mask.
Step 5: Create Matrix with zeroes with size of image to hold correlation and convolution results.
Step 6: Pad the image to be convoluted/correlated with zeroes according to kernel mask size.
Step 7: Run a nested loop ( both convolution and correlation)
        Step 7.1: For each pixel in image, keeping it at centre, extract the pixel intensities
        of neighboring pixels falling under the Kernel window.
        Step 7.2: Multiply the extracted pixel intensities with respective kernel masks.
        Step 7.3: Take the sum of all elements and assign it to the convolution/correlation
        copy image at that pixel position.
Step 8: Adjust the convoluted and correlated pixel intensities to lie between [0,255].
Step 9: Display the convoluted and correlated images.
```

Program Code

```
from PIL import Image, ImageOps
import numpy as np

Cameraman_Image = Image.open("Cameraman.png")
Cameraman_gray =  ImageOps.grayscale(Cameraman_Image)
```

```
Image_copy = Cameraman_gray.copy()

height, width = Cameraman_gray.size

# Correlation mask used (size 3x3)

Corr_Mask = [[1,-1,-1],
             [1,2,-1],
             [1,1,1]]


# 180 degree flipped Convolution mask
Flip_Conv_Mask =  np.flipud(np.fliplr(Corr_Mask))

# Padding size
size = len(Flip_Conv_Mask)
pad = int((size-1)/2)

# Padding image for convolution
Image_copy_array = np.asarray(Image_copy)

#Converting image to array
Pad_image = np.pad(np.array(Image_copy_array), ((pad,pad), (pad, pad)), 'constant')

# Creating a matrix with zeros to store convolution and correlation results

height_p = height + 2*pad
width_p = width + 2*pad
Conv_Image = np.full((height_p, width_p),0)
Corr_Image = np.full((height_p, width_p),0)

# Convolution and Correlation

for i in range(pad,height_p-pad):
    for j in range(pad,width_p- pad):
        submatrix = Pad_image[i-pad:i+pad+1,j-pad:j+pad+1]

        submatrix_mask_multipy_conv = np.multiply(submatrix , Flip_Conv_Mask)
        submatrix_mask_multipy_corr = np.multiply(submatrix , Corr_Mask)

        sum_elements_conv = list(map(sum, submatrix_mask_multipy_conv))
        Conv_Image[i,j] = sum(sum_elements_conv)

        sum_elements_corr = list(map(sum, submatrix_mask_multipy_corr))
        Corr_Image[i,j] = sum(sum_elements_corr)

# Extract Convoulted image from padded zero Convoulted Image

Conv_Image_cropped = Conv_Image[pad:height_p -pad,pad:width_p-pad]
Corr_Image_cropped = Corr_Image[pad:height_p -pad,pad:width_p-pad]

# Saturate all pixel values greater than 255 to 255
Conv_Image_cropped = np.clip(Conv_Image_cropped, 0, 255)
Conv_Image_cropped = Conv_Image_cropped.astype('uint8')

Corr_Image_cropped = np.clip(Corr_Image_cropped, 0, 255)
Corr_Image_cropped = Corr_Image_cropped.astype('uint8')
```

```
# Converting array containing convolution results to image
Convoluted_Image = Image.fromarray(Conv_Image_cropped)
Correlated_Image = Image.fromarray(Corr_Image_cropped)

Convoluted_Image.show()
Convoluted_Image.save('Cameraman_Convoluted.png')

Correlated_Image.show()
Correlated_Image.save('Cameraman_Correlated.png')
```

Result

Figure 1: Cameraman : (a)Original (b)Convoluted Image (c)Correlated Image

Inference

Convolution/Correlation on an image can be used to change the spatial domain characteristics of the image according to the kernel mask used.

Question 2: Add different types of noises to the input image using inbuilt functions in python.

Aim

Adding different types of noises to the input image using inbuilt functions

Discussion

Different types of noises may exist in a image. For e.g. Gaussian noise (generally caused due to natural sources), salt and pepper noise (impulse noises generally added during data transmissions),Speckle noise, Poisson noise etc.

Algorithm

```
Step 1: Start
Step 2: Read the image and convert it into gray scale
Step 3: Add different types noises to image using inbuil functions.
Step 4: Display noise added images.
```

Program Code

```
import cv2
import numpy as np
from skimage.util import random_noise


# Load the image
Image_Cameraman = cv2.imread("Cameraman.png")

#Using skimage module

# 1)Add salt-and-pepper noise to the image.
Image_Cameraman_saltpepper = random_noise(Image_Cameraman, mode='s&p',amount=0.01)

Image_Cameraman_saltpepper = np.array(255*Image_Cameraman_saltpepper, dtype = 'uint8')
cv2.imshow('Image_Cameraman_saltpepper',Image_Cameraman_saltpepper)
cv2.imwrite('Cameraman_Noise_Salt_and_Pepper.jpeg',Image_Cameraman_saltpepper)

cv2.waitKey(0)


# 2) Add speckle noise to the image.
Image_Cameraman_speckle = random_noise(Image_Cameraman, mode='speckle',mean= 0.5)

Image_Cameraman_speckle = np.array(255*Image_Cameraman_speckle, dtype = 'uint8')
cv2.imshow('Image_Cameraman_speckle',Image_Cameraman_speckle)
cv2.imwrite('Cameraman_Noise_Speckle.jpeg',Image_Cameraman_speckle)

cv2.waitKey(0)


#3) Add gaussian noise to the image.
Image_Cameraman_gaussian = random_noise(Image_Cameraman, mode='gaussian')

Image_Cameraman_gaussian = np.array(255*Image_Cameraman_gaussian, dtype = 'uint8')
cv2.imshow('Image_Cameraman_gaussian',Image_Cameraman_gaussian)
cv2.imwrite('Cameraman_Noise_Gaussian.jpeg',Image_Cameraman_gaussian)

cv2.waitKey(0)

# 4) Add poisson noise to the image.
Image_Cameraman_poisson = random_noise(Image_Cameraman, mode='poisson')

Image_Cameraman_poisson = np.array(255*Image_Cameraman_poisson, dtype = 'uint8')
cv2.imshow('Image_Cameraman_poisson',Image_Cameraman_poisson)
cv2.imwrite('Cameraman_Noise_Poisson.jpeg',Image_Cameraman_poisson)
cv2.waitKey(0)
```

Result

Figure 2: Cameraman : (a)original

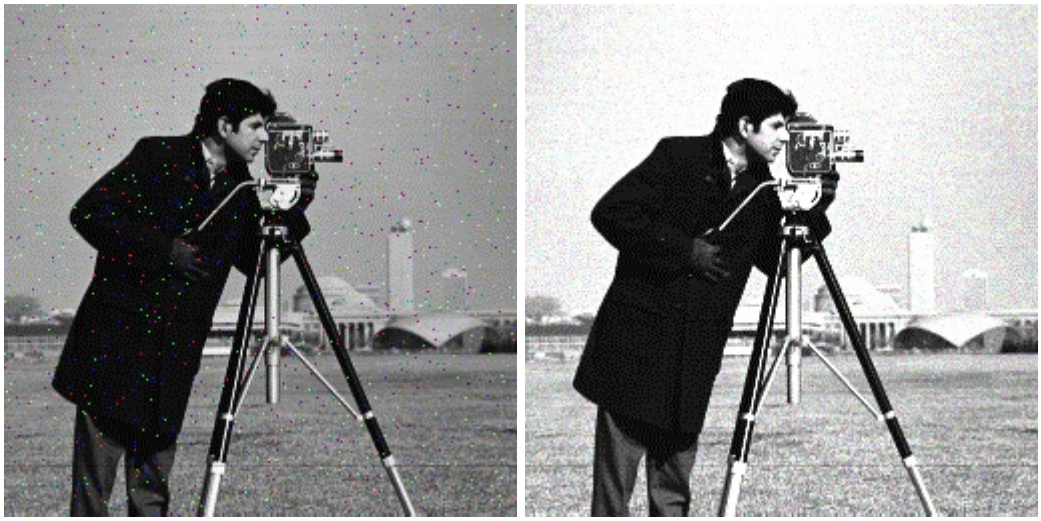

Figure 3: Cameraman : (a)Salt and Pepper Noise (b)Speckle Noise

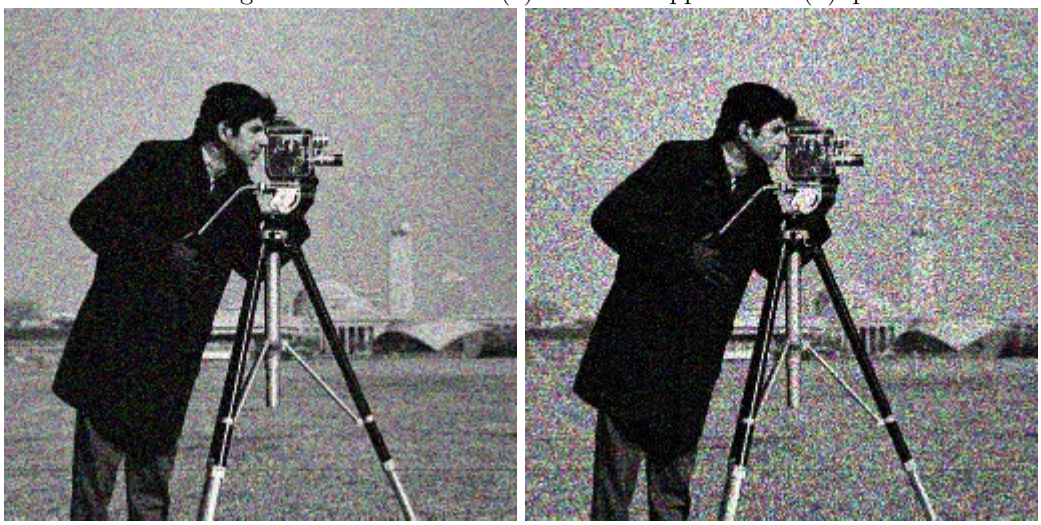

Figure 4: Cameraman : (a)Gaussian Noise (b)Poisson Noise

Inference

Different types of noises can be introduced into an image using inbuilt functions in python.
of an image.

Question 3: From the outputs obtained from question 2 i.e., noisy images, perform average filtering over the images and state which type of noise is best removed using average filter. Use 3 $\times$ 3, 5 $\times$ 5, 11 $\times$ 11 masks

Aim

Applying average filtering over noisy images using 3x3, 5x5 and 11x11 masks

Discussion

Averaging filter can be used as a blurring filter. It is used in reducing the details in an image. The kernel mask is a square matrix(generally odd) with all its coefficients being equal. It can be used in image denoising.

Algorithm

Step 1: Start
Step 2: Read the noisy images and convert them into gray-scale .
Step 3: Create average mask of required order with all coefficients being equal to (1/(order)^2).
Step 4: Convolve the noisy image with averaging filter mask.
Step 5: Repeat the above for all noisy images each for masks of order 3,5 and 11 and observe the results.
Step 6: Display all the Average filter applied images

Program Code

```
from PIL import Image, ImageOps
import numpy as np

def Avg_Filter(Image_Noisy, Factor):

    # Odd mask created with order Factor*Factor

    Mask = np.full((Factor, Factor),(1/(Factor**2)))

    size = len(Mask)
    paddy = int((size-1)/2)
    height, width = Image_Noisy.size

    Image_copy_array = np.asarray(Image_Noisy)

    #Converting image to array
    Pad_image = np.pad(np.array(Image_copy_array), ((paddy,paddy), (paddy, paddy)), 'constant')

    # Creating a matrix with zeros to store convolution and correlation results
```

```
    height_p = height + 2*paddy
    width_p = width + 2*paddy

    Corr_Image = np.full((height_p, width_p),0)

    # Correlation

    for i in range(paddy,height_p-paddy):
        for j in range(paddy,width_p- paddy):
            submatrix = Pad_image[i-paddy:i+paddy+1,j-paddy:j+paddy+1]


            submatrix_mask_multipy_corr = np.multiply(submatrix , Mask)


            sum_elements_corr = list(map(sum, submatrix_mask_multipy_corr))
            Corr_Image[i,j] = sum(sum_elements_corr)

    Corr_Image_cropped = Corr_Image[paddy:height_p -paddy,paddy:width_p-paddy]

    Corr_Image_cropped = np.clip(Corr_Image_cropped, 0, 255)
    Corr_Image_cropped = Corr_Image_cropped.astype('uint8')

    Correlated_Image = Image.fromarray(Corr_Image_cropped)
    return Correlated_Image


Noisy_Salt_and_Pepper = Image.open("Cameraman_Noise_Salt_and_Pepper.jpeg")
Noisy_Salt_and_Pepper =  ImageOps.grayscale(Noisy_Salt_and_Pepper)

Noisy_Speckle = Image.open("Cameraman_Noise_Speckle.jpeg")
Noisy_Speckle = ImageOps.grayscale(Noisy_Speckle)

Noisy_Gaussian = Image.open("Cameraman_Noise_Gaussian.jpeg")
Noisy_Gaussian =  ImageOps.grayscale(Noisy_Gaussian)

Noisy_Poisson = Image.open("Cameraman_Noise_Poisson.jpeg")
Noisy_Poisson = ImageOps.grayscale(Noisy_Poisson)

#Passing Noisy images through Averaging Filter

Noisy_Salt_and_Pepper_Avg_3 = Avg_Filter(Noisy_Salt_and_Pepper, 3)
Noisy_Salt_and_Pepper_Avg_3.show()
Noisy_Salt_and_Pepper_Avg_3.save('Noisy_Salt_and_Pepper_Avg_3.png')

Noisy_Salt_and_Pepper_Avg_5 = Avg_Filter(Noisy_Salt_and_Pepper, 5)
Noisy_Salt_and_Pepper_Avg_5.show()
Noisy_Salt_and_Pepper_Avg_5.save('Noisy_Salt_and_Pepper_Avg_5.png')

Noisy_Salt_and_Pepper_Avg_11 = Avg_Filter(Noisy_Salt_and_Pepper, 11)
Noisy_Salt_and_Pepper_Avg_11.show()
Noisy_Salt_and_Pepper_Avg_11.save('Noisy_Salt_and_Pepper_Avg_11.png')

Noisy_Speckle_Avg_3 = Avg_Filter(Noisy_Speckle, 3)
Noisy_Speckle_Avg_3.show()
Noisy_Speckle_Avg_3.save('Noisy_Speckle_Avg_3.png')

Noisy_Speckle_Avg_5 = Avg_Filter(Noisy_Speckle, 5)
```

```
Noisy_Speckle_Avg_5.show()
Noisy_Speckle_Avg_5.save('Noisy_Speckle_Avg_5.png')

Noisy_Speckle_Avg_11 = Avg_Filter(Noisy_Speckle, 11)
Noisy_Speckle_Avg_11.show()
Noisy_Speckle_Avg_11.save('Noisy_Speckle_Avg_11.png')


Noisy_Gaussian_Avg_3 = Avg_Filter(Noisy_Gaussian, 3)
Noisy_Gaussian_Avg_3.show()
Noisy_Gaussian_Avg_3.save('Noisy_Gaussian_Avg_3.png')

Noisy_Gaussian_Avg_5 = Avg_Filter(Noisy_Gaussian, 5)
Noisy_Gaussian_Avg_5.show()
Noisy_Gaussian_Avg_5.save('Noisy_Gaussian_Avg_5.png')

Noisy_Gaussian_Avg_11 = Avg_Filter(Noisy_Gaussian, 11)
Noisy_Gaussian_Avg_11.show()
Noisy_Gaussian_Avg_11.save('Noisy_Gaussian_Avg_11.png')


Noisy_Poisson_Avg_3 = Avg_Filter(Noisy_Poisson, 3)
Noisy_Poisson_Avg_3.show()
Noisy_Poisson_Avg_3.save('Noisy_Poisson_Avg_3.png')

Noisy_Poisson_Avg_5 = Avg_Filter(Noisy_Poisson, 5)
Noisy_Poisson_Avg_5.show()
Noisy_Poisson_Avg_5.save('Noisy_Poisson_Avg_5.png')

Noisy_Poisson_Avg_11 = Avg_Filter(Noisy_Poisson, 11)
Noisy_Poisson_Avg_11.show()
Noisy_Poisson_Avg_11.save('Noisy_Poisson_Avg_11.png')
```

Result

Figure 5: Cameraman : Salt and Pepper Noise

Figure 6: Salt and Pepper Noise : (a)Average Filter(3x3) (b)Average Filter(5x5)(c)Average Filter(11x11)

Figure 7: Cameraman : Speckle Noise

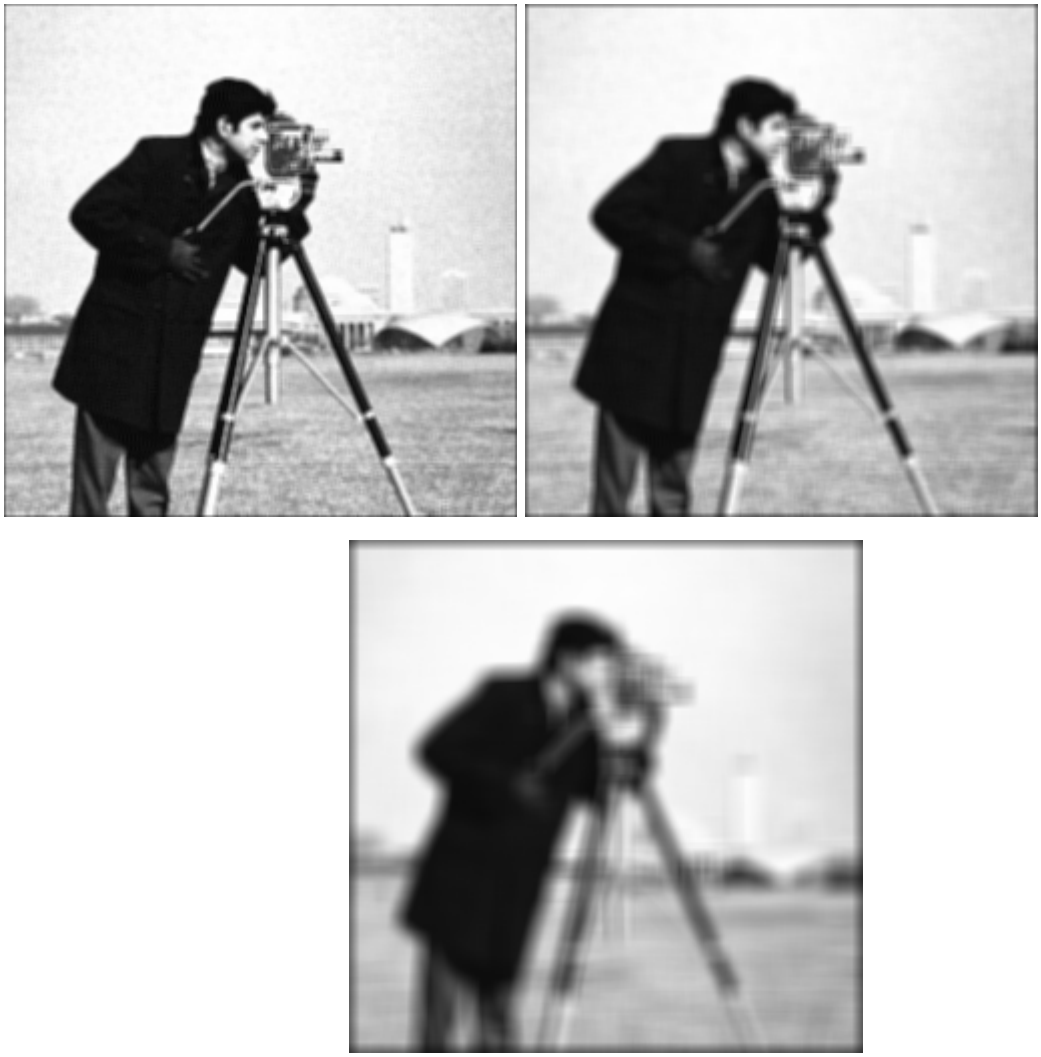

Figure 8: Speckle Noise : (a)Average Filter(3x3) (b)Average Filter(5x5)(c)Average Filter(11x11)

Figure 9: Cameraman : Gaussian Noise

Figure 10: Gaussian Noise : (a)Average Filter(3x3) (b)Average Filter(5x5)(c)Average Filter(11x11)

Figure 11: Cameraman : Poisson Noise

Figure 12: Poisson Noise : (a)Average Filter(3x3) (b)Average Filter(5x5)(c)Average Filter(11x11)

Inference

`Averaging filter works as a blurring filter which helps removing sharp impulse noises.`

Comparing the results of average filter with different kinds on noises, it gives the best result in removing salt and pepper noise.

Question 4: From the outputs obtained from question 2 i.e., noisy images, perform median filtering over the images and state which type of noise is best removed using average filter. Use 3 × 3, 5 × 5, 11 × 11 masks

Aim

Performing Median filtering on noisy images with mask size 3x3, 5x5 and 11x11.

Discussion

Median filter is a non-linear filter.It is very effective at removing noise while preserving edges. The median filter works by moving through the image pixel by pixel, replacing each value with the median value of neighbouring pixels.

Algorithm

```
Step 1: Start
Step 2: Read the noisy images and convert them into gray-scale .
Step 3: Create a copy of the noisy image for Median filtering.
Step 3: Run a nested loop
        Step 3.1: For each pixel in image, keeping it at centre, extract the pixel intensities
        of neighboring pixels falling under the Kernel window.
        Step 3.2: Take the median of extracted elements and assign it to the
        copy image at that pixel position.
Step 4: Adjust the convoluted and correlated pixel intensities to lie between [0,255].
Step 5: Repeat the same for all noisy images for Median filter sizes 3,5 and 11.
Step 6: Display the median filtered images and compare.
```

Program Code

```
from PIL import Image, ImageOps
import numpy as np

def Median_Filter(Image_Noisy, Factor):

    Mask = np.full((Factor, Factor),1)

    size = len(Mask)
    paddy = int((size-1)/2)
    height, width = Image_Noisy.size

    Image_copy_array = np.asarray(Image_Noisy)

    #Converting image to array
    Pad_image = np.pad(np.array(Image_copy_array), ((paddy,paddy), (paddy, paddy)), 'constant')

    # Creating a matrix with zeros to store convolution and correlation results
```

```
    height_p = height + 2*paddy
    width_p = width + 2*paddy

    Corr_Image = np.full((height_p, width_p),0)

    # Correlation

    for i in range(paddy,height_p-paddy):
        for j in range(paddy,width_p- paddy):
            submatrix = Pad_image[i-paddy:i+paddy+1,j-paddy:j+paddy+1]

            submatrix_mask_multipy_corr = np.multiply(submatrix , Mask)

            elements_list =[]
            for z in submatrix_mask_multipy_corr:
                elements_list.append(z)

            #Finding the Median value of pixel values in current window

            Corr_Image[i,j] = np.median(elements_list)

    Corr_Image_cropped = Corr_Image[paddy:height_p -paddy,paddy:width_p-paddy]
    Corr_Image_cropped = np.clip(Corr_Image_cropped, 0, 255)
    Corr_Image_cropped = Corr_Image_cropped.astype('uint8')

    Correlated_Image = Image.fromarray(Corr_Image_cropped)
    return Correlated_Image


Noisy_Salt_and_Pepper = Image.open("Cameraman_Noise_Salt_and_Pepper.jpeg")
Noisy_Salt_and_Pepper =  ImageOps.grayscale(Noisy_Salt_and_Pepper)

Noisy_Speckle = Image.open("Cameraman_Noise_Speckle.jpeg")
Noisy_Speckle = ImageOps.grayscale(Noisy_Speckle)

Noisy_Gaussian = Image.open("Cameraman_Noise_Gaussian.jpeg")
Noisy_Gaussian =  ImageOps.grayscale(Noisy_Gaussian)

Noisy_Poisson = Image.open("Cameraman_Noise_Poisson.jpeg")
Noisy_Poisson = ImageOps.grayscale(Noisy_Poisson)

#Passing Noisy images through Median Filter

Noisy_Salt_and_Pepper_Median_3 = Median_Filter(Noisy_Salt_and_Pepper, 3)
Noisy_Salt_and_Pepper_Median_3.show()
Noisy_Salt_and_Pepper_Median_3.save('Noisy_Salt_and_Pepper_Median_3.png')

Noisy_Salt_and_Pepper_Median_5 = Median_Filter(Noisy_Salt_and_Pepper, 5)
Noisy_Salt_and_Pepper_Median_5.show()
Noisy_Salt_and_Pepper_Median_5.save('Noisy_Salt_and_Pepper_Median_5.png')

Noisy_Salt_and_Pepper_Median_11 = Median_Filter(Noisy_Salt_and_Pepper, 11)
Noisy_Salt_and_Pepper_Median_11.show()
Noisy_Salt_and_Pepper_Median_11.save('Noisy_Salt_and_Pepper_Median_11.png')


Noisy_Speckle_Median_3 = Median_Filter(Noisy_Speckle, 3)
Noisy_Speckle_Median_3.show()
```

```
Noisy_Speckle_Median_3.save('Noisy_Speckle_Median_3.png')

Noisy_Speckle_Median_5 = Median_Filter(Noisy_Speckle, 5)
Noisy_Speckle_Median_5.show()
Noisy_Speckle_Median_5.save('Noisy_Speckle_Median_5.png')

Noisy_Speckle_Median_11 = Median_Filter(Noisy_Speckle, 11)
Noisy_Speckle_Median_11.show()
Noisy_Speckle_Median_11.save('Noisy_Speckle_Median_11.png')


Noisy_Gaussian_Median_3 = Median_Filter(Noisy_Gaussian, 3)
Noisy_Gaussian_Median_3.show()
Noisy_Gaussian_Median_3.save('Noisy_Gaussian_Median_3.png')

Noisy_Gaussian_Median_5 = Median_Filter(Noisy_Gaussian, 5)
Noisy_Gaussian_Median_5.show()
Noisy_Gaussian_Median_5.save('Noisy_Gaussian_Median_5.png')

Noisy_Gaussian_Median_11 = Median_Filter(Noisy_Gaussian, 11)
Noisy_Gaussian_Median_11.show()
Noisy_Gaussian_Median_11.save('Noisy_Gaussian_Median_11.png')


Noisy_Poisson_Median_3 = Median_Filter(Noisy_Poisson, 3)
Noisy_Poisson_Median_3.show()
Noisy_Poisson_Median_3.save('Noisy_Poisson_Median_3.png')

Noisy_Poisson_Median_5 = Median_Filter(Noisy_Poisson, 5)
Noisy_Poisson_Median_5.show()
Noisy_Poisson_Median_5.save('Noisy_Poisson_Median_5.png')

Noisy_Poisson_Median_11 = Median_Filter(Noisy_Poisson, 11)
Noisy_Poisson_Median_11.show()
Noisy_Poisson_Median_11.save('Noisy_Poisson_Median_11.png')
```

Result

Figure 13: Cameraman : Salt and Pepper Noise

Figure 14: Salt and Pepper Noise : (a)Median Filter(3x3) (b)Median Filter(5x5)(c)Median Filter(11x11)

Figure 15: Cameraman : Speckle Noise

Figure 16: Speckle Noise : (a)Median Filter(3x3) (b)Median Filter(5x5)(c)Median Filter(11x11)

Figure 17: Cameraman : Gaussian Noise

Figure 18: Gaussian Noise : (a)Median Filter(3x3) (b)Median Filter(5x5)(c)Median Filter(11x11)

Figure 19: Cameraman : Poisson Noise

Figure 20: Poisson Noise : (a)Median Filter(3x3) (b)Median Filter(5x5)(c)Median Filter(11x11)

Inference

Comparing the results of median filter with different kinds on noises, it gives the best

result in removing salt and pepper noise.

Question 5: Edge Detection Perform edge detection for the given images with and without using inbuilt functions and compare the results.

Aim

Performing edge detection using Sobel, Prewitt and Roberts mask with and without inbuilt functions.

Discussion

Edge detection is used to enhance the line structures and edges of an image. The edge detection filters used act as a high pass filter.

Algorithm

```
Step 1: Start
Step 2: Read the image and convert in into gray-scale
Step 3: Obtain the edge detection mask to be used.
Step 4: Create Matrix with zeroes with size of image to hold edge detection results.
Step 5: Pad the image to be correlated with zeroes according to kernel mask size.
Step 6: Run a nested loop
        Step 6.1: For each pixel in image, keeping it at centre, extract the pixel intensities
        of neighboring pixels falling under the Kernel window.
        Step 6.2: Multiply the extracted pixel intensities with respective kernel masks.
        Step 6.3: Take the sum of all elements and assign it to the copy image at that pixel
        position.
Step 7: Adjust the correlated ( edge detected ) pixel intensities to lie between [0,255].
Step 8: Repeat the above for Sobel , Prewitt and Roberts masks.
Step 9: Display the edge detected images.
```

Program Code

```
''' For Sobel and Prewitt Masks '''

from PIL import Image, ImageOps
import numpy as np
import cv2


def Edge_Filter(Image_Noisy, Mask):


    size = len(Mask)
    paddy = int((size-1)/2)
    height, width = Image_Noisy.size

    Image_copy_array = np.asarray(Image_Noisy)

    #Converting image to array
    Pad_image = np.pad(np.array(Image_copy_array), ((paddy,paddy), (paddy, paddy)), 'constant')
```

```
    # Creating a matrix with zeros to store convolution and correlation results

    height_p = height + 2*paddy
    width_p = width + 2*paddy

    Corr_Image = np.full((height_p, width_p),0)

    # Correlation

    for i in range(paddy,height_p-paddy):
        for j in range(paddy,width_p- paddy):
            submatrix = Pad_image[i-paddy:i+paddy+1,j-paddy:j+paddy+1]


            submatrix_mask_multipy_corr = np.multiply(submatrix , Mask)


            sum_elements_corr = list(map(sum, submatrix_mask_multipy_corr))
            Corr_Image[i,j] = sum(sum_elements_corr)

    Corr_Image_cropped = Corr_Image[paddy:height_p -paddy,paddy:width_p-paddy]
    Corr_Image_cropped = np.clip(Corr_Image_cropped, 0, 255)
    Corr_Image_cropped = Corr_Image_cropped.astype('uint8')


    Correlated_Image = Image.fromarray(Corr_Image_cropped)

    return Correlated_Image


Image_Cameraman = cv2.imread("Cameraman.png")
Image_Cameraman = cv2.cvtColor(Image_Cameraman, cv2.COLOR_BGR2GRAY)

Cameraman_Image_1 = Image.open("Cameraman.png")
Cameraman_gray =  ImageOps.grayscale(Cameraman_Image_1)

# Sobel Masks

Sobel_x =   [[-1,0,1],
             [-2,0,2],
             [-1,0,1]]

Sobel_y =   [[1,2,1],
             [0,0,0],
             [-1,-2,-1]]

Sobel_x_inbuilt = cv2.Sobel(Image_Cameraman,cv2.CV_8U,1,0,ksize=3)
cv2.imshow('Sobel_x_inbuilt',Sobel_x_inbuilt)
cv2.imwrite('Sobel_x_inbuilt.png',Sobel_x_inbuilt)
cv2.waitKey(0)

Sobel_x_custom_function = Edge_Filter(Cameraman_gray, Sobel_x)
Sobel_x_custom_function.show()
Sobel_x_custom_function.save('Sobel_x_custom_function.png')

Sobel_y_inbuilt = cv2.Sobel(Image_Cameraman,cv2.CV_8U,0,1,ksize=3)
cv2.imshow('Sobel_y_inbuilt',Sobel_y_inbuilt)
cv2.imwrite('Sobel_y_inbuilt.png',Sobel_y_inbuilt)
```

```
cv2.waitKey(0)

Sobel_y_custom_function = Edge_Filter(Cameraman_gray, Sobel_y)
Sobel_y_custom_function.show()
Sobel_y_custom_function.save('Sobel_y_custom_function.png')

#Prewitt Masks

Prewitt_y = [[1,1,1],
             [0,0,0],
             [-1,-1,-1]]

Prewitt_x = [[1,0,-1],
             [1,0,-1],
             [1,0,-1]]

Prewitt_x_inbuilt = cv2.filter2D(Image_Cameraman, -1, np.array(Prewitt_x))
cv2.imshow('Prewitt_x_inbuilt',Prewitt_x_inbuilt)
cv2.imwrite('Prewitt_x_inbuilt.png',Prewitt_x_inbuilt)
cv2.waitKey(0)

Prewitt_x_custom_function = Edge_Filter(Cameraman_gray, Prewitt_x)
Prewitt_x_custom_function.show()
Prewitt_x_custom_function.save('Prewitt_x_custom_function.png')


Prewitt_y_inbuilt = cv2.filter2D(Image_Cameraman, -1, np.array(Prewitt_y))
cv2.imshow('Prewitt_y_inbuilt',Prewitt_y_inbuilt)
cv2.imwrite('Prewitt_y_inbuilt.png',Prewitt_y_inbuilt)
cv2.waitKey(0)

Prewitt_y_custom_function = Edge_Filter(Cameraman_gray, Prewitt_y)
Prewitt_y_custom_function.show()
Prewitt_y_custom_function.save('Prewitt_y_custom_function.png')

''' For Roberts Mask'''
from PIL import Image, ImageOps
import numpy as np
import cv2
from skimage import filters

def Roberts_Filter(Image_Noisy, Mask):


    paddy = int(len(Mask)/2)
    height, width = Image_Noisy.size

    Image_copy_array = np.asarray(Image_Noisy)

    #Converting image to array
    Pad_image = np.pad(np.array(Image_copy_array), ((0,paddy), (0, paddy)), 'constant')

    # Creating a matrix with zeros to store convolution and correlation results

    height_p = height + paddy
    width_p = width + paddy

    Corr_Image = np.full((height_p, width_p),0)
```

```
    # Correlation  of Image with Even Order Mask

    for i in range(0,height_p-paddy):
        for j in range(0,width_p- paddy):
            submatrix = Pad_image[i:i+paddy+1,j:j+paddy+1]


            submatrix_mask_multipy_corr = np.multiply(submatrix , Mask)


            sum_elements_corr = list(map(sum, submatrix_mask_multipy_corr))
            Corr_Image[i,j] = sum(sum_elements_corr)

    Corr_Image_cropped = Corr_Image[0:height_p -paddy,0:width_p-paddy]
    Corr_Image_cropped = np.clip(Corr_Image_cropped, 0, 255)
    Corr_Image_cropped = Corr_Image_cropped.astype('uint8')


    Correlated_Image = Image.fromarray(Corr_Image_cropped)

    return Correlated_Image


Image_Cameraman = cv2.imread("Cameraman.png")
Image_Cameraman = cv2.cvtColor(Image_Cameraman, cv2.COLOR_BGR2GRAY)

Cameraman_Image_1 = Image.open("Cameraman.png")
Cameraman_gray =  ImageOps.grayscale(Cameraman_Image_1)


#Roberts Mask

Roberts_x = [[1,0],
             [0,-1]]

Roberts_y = [[0,1],
             [-1,-0]]

Roberts_x_inbuilt = filters.roberts_neg_diag(Image_Cameraman)
cv2.imshow('Roberts_x_inbuilt',Roberts_x_inbuilt)
cv2.imwrite('Roberts_x_inbuilt.png',Roberts_x_inbuilt)
cv2.waitKey(0)

Roberts_x_custom_function = Roberts_Filter(Cameraman_gray, Roberts_x)
Roberts_x_custom_function.show()
Roberts_x_custom_function.save('Roberts_x_custom_function.png')


Roberts_y_inbuilt = filters.roberts_pos_diag(Image_Cameraman)
cv2.imshow('Roberts_y_inbuilt',Roberts_y_inbuilt)
cv2.imwrite('Roberts_y_inbuilt.png',Roberts_y_inbuilt)
cv2.waitKey(0)

Roberts_y_custom_function = Roberts_Filter(Cameraman_gray, Roberts_y)
Roberts_y_custom_function.show()
Roberts_y_custom_function.save('Roberts_y_custom_function.png')
```

Result

Figure 21: Cameraman : (a)Original

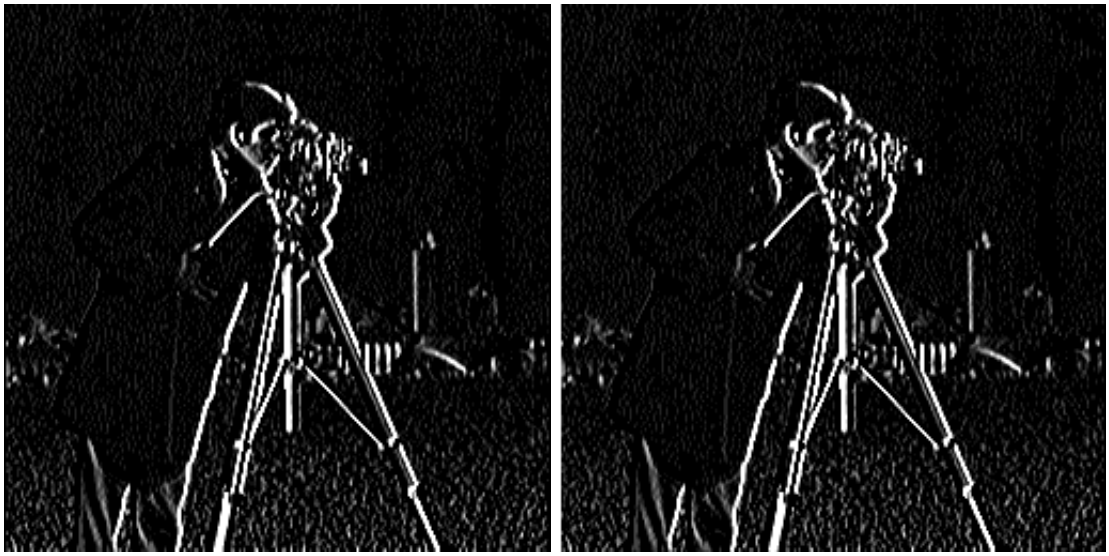

Figure 22: Sobel-x : (a)With Inbuilt Function(b)Without Inbuilt Function

Figure 23: Sobel-y : (a)With Inbuilt Function(b)Without Inbuilt Function

Figure 24: Prewitt-x : (a)With Inbuilt Function(b)Without Inbuilt Function

Figure 25: Prewitt-y : (a)With Inbuilt Function(b)Without Inbuilt Function

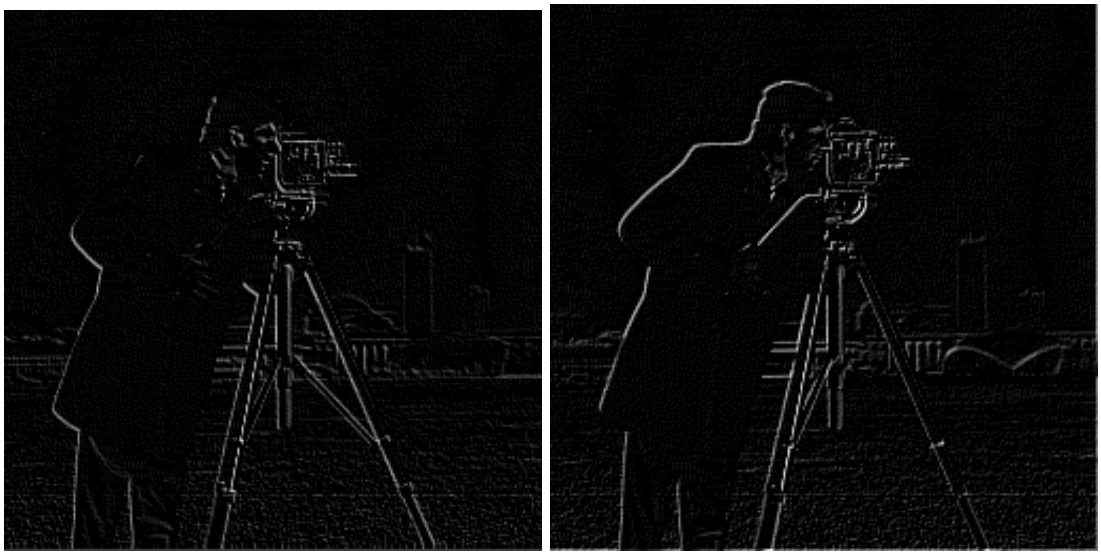

Figure 26: Roberts x : (a)With Inbuilt Function(b)Without Inbuilt Function

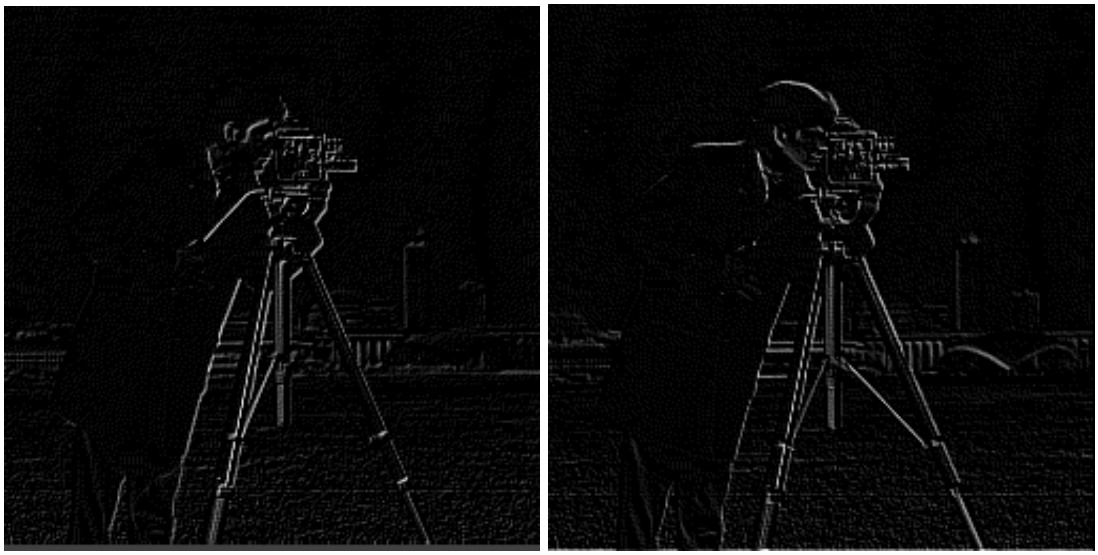

Figure 27: Roberts y : (a)With Inbuilt Function(b)Without Inbuilt Function

Inference

Edge detection enhances the edges - generally areas where there is sharp change in pixel intensity, thus working as a high pass filter.

Question 6: Implement the unsharp masking algorithm

Aim

To implement unsharp masking algorithm to sharpen image.

Discussion

Unsharp masking algorithm implements sharpening of image by using blurring(low pass) filters. It is implemented by :

1. Blur the original image 2. Subtract the blurred image from the original 3. Add the previous result to the original.

Algorithm

```
Step 1: Start
Step 2: Read the image and convert in into gray-scale
Step 3: Blur the image using Averaging filter.
Step 4: Subtract the blurred image from the original image.
Step 5: Add the subtracted image to the original image.
Step 6: Adjust the pixel intensities to lie between [0,255].
Step 7: Display the final output and images used in the unsharp mask filtering algorithm.
```

Program Code

```
from PIL import Image, ImageOps
import numpy as np
```

```
Cameraman_Image = Image.open("Cameraman.png")
Cameraman_gray =  ImageOps.grayscale(Cameraman_Image)

Factor = 3
Blur_Mask = np.full((Factor, Factor),(1/(Factor**2)))


height, width = Cameraman_gray.size
size = len(Blur_Mask)
pad = int((size-1)/2)

Image_copy_array = np.asarray(Cameraman_gray)

#Converting image to array
Pad_image = np.pad(np.array(Image_copy_array), ((pad,pad), (pad, pad)), 'constant')

# Creating a matrix with zeros to store convolution and correlation results

height_p = height + 2*pad
width_p = width + 2*pad

Corr_Image = np.full((height_p, width_p),0)

# Correlation

for i in range(pad,height-pad):
    for j in range(pad,width- pad):
        submatrix = Pad_image[i-pad:i+pad+1,j-pad:j+pad+1]


        submatrix_mask_multipy_corr = np.multiply(submatrix , Blur_Mask)



        sum_elements_corr = list(map(sum, submatrix_mask_multipy_corr))
        Corr_Image[i,j] = sum(sum_elements_corr)

# Blurred Image
Corr_Image_cropped = Corr_Image[pad:height_p -pad,pad:width_p-pad]

# Blurred Image subtracted from original image
OG_subtract_blur = Image_copy_array - Corr_Image_cropped

# Unsharp mask added to original image

OG_plus_Edge_Mask = Image_copy_array + OG_subtract_blur

Corr_Image_cropped = np.clip(Corr_Image_cropped, 0, 255)
Corr_Image_cropped = Corr_Image_cropped.astype('uint8')
Blurred_Image = Image.fromarray(Corr_Image_cropped)

OG_subtract_blur = np.clip(OG_subtract_blur, 0, 255)
OG_subtract_blur = OG_subtract_blur.astype('uint8')
Blur_Subtracted_Image = Image.fromarray(OG_subtract_blur)
```

```
OG_plus_Edge_Mask = np.clip(OG_plus_Edge_Mask, 0, 255)
OG_plus_Edge_Mask = OG_plus_Edge_Mask.astype('uint8')
Unsharp_Image = Image.fromarray(OG_plus_Edge_Mask)


Blurred_Image.show()
Blurred_Image.save('Blurred_Image.png')

Blur_Subtracted_Image.show()
Blur_Subtracted_Image.save('Blur_Subtracted_Image.png')

Unsharp_Image.show()
Unsharp_Image.save('Unsharp_Image.png')
```

Result

Figure 28: Cameraman : (a)Original (b)Blurred Image

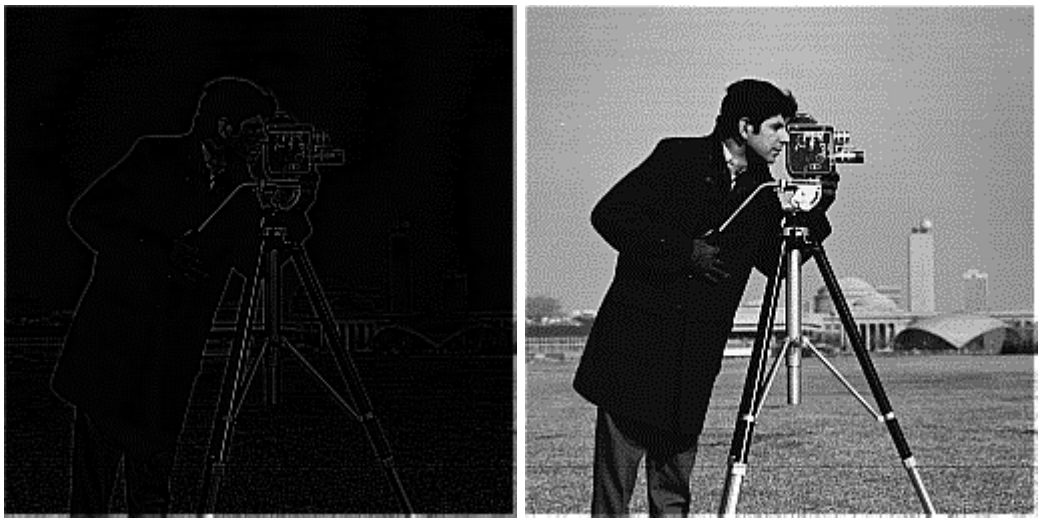

Figure 29: (c)Original - Blurred Image (a-b) (d)Sharpened Image(c+a)

Inference

Using Unsharp Masking Algorithm, images can be sharpened using a low pass (blurring) filter.