HW-4

Problem 01:

a. Original image:



Adding Salt-Peeper Noise:

#%%

#Problem: 01

#importing cv2

import cv2

# Using cv2.imread() method

img = cv2.imread('4\_1.bmp')

# Displaying the image using cv2.imshow()

cv2.imshow('Original Image', img)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

import numpy as np

image\_first\_band = img[:,:,0]

rows = image\_first\_band.shape[0]

coloumns= image\_first\_band.shape[1]

s\_p = np.random.randint(0, 256, size=(rows, coloumns))

# %%

for i in range(0, rows):

    for j in range(0, coloumns):

        if s\_p[i, j]==0 or s\_p[i, j]==255:

            image\_first\_band[i, j]= s\_p[i, j]

image\_final = cv2.merge((image\_first\_band, image\_first\_band, image\_first\_band))

cv2.imshow('salt\_peeper\_noise', image\_final)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

Noisy Image:



Low Pass Filter:

# %%

#LOW PASS FILTER

#padding

import math

new\_row= rows+2

new\_coloumn= coloumns+2

padded\_matrix = np.zeros((int(new\_row), int(new\_coloumn)))

s= -1

for i in range(1, new\_row-1):

    s= s+1

    t= 0

    for j in range(1, new\_coloumn-1):

        padded\_matrix[i, j]= image\_first\_band[s,t]

        t= t+1

p= 0

for q in range(1, new\_coloumn-1):

    padded\_matrix[p, q]= padded\_matrix[p+1, q ]

u= new\_row-1

for v in range(1, new\_coloumn-1):

    padded\_matrix[u, v]= padded\_matrix[u-1, v ]

padded\_matrix[0, 0]= padded\_matrix[0, 1]

padded\_matrix[new\_row-1, 0]= padded\_matrix[new\_row-1, 1]

padded\_matrix[0, new\_row-1] = padded\_matrix[0, new\_row-2]

padded\_matrix[new\_row-1, new\_row-1]= padded\_matrix[new\_row-1, new\_row-2]

b= 0

for a in range(1, new\_coloumn-1 ):

    padded\_matrix[a, b]= padded\_matrix[a, b+1]

d= new\_coloumn-1

for c in range(1, new\_coloumn-1 ):

    padded\_matrix[c, d]= padded\_matrix[c, d-1]

LPF\_matrix= np.zeros((int(rows), int(coloumns)))

for i in range(0, rows):

    for j in range(0, coloumns):

        LPF\_matrix[i, j]= math.floor((padded\_matrix[i, j]\*1+padded\_matrix[i, j+1]\*1+padded\_matrix[i, j+2]\*1+

                          padded\_matrix[i+1, j]\*1+padded\_matrix[i+1, j+1]\*1+padded\_matrix[i+1, j+2]\*1+

                          padded\_matrix[i+2, j]\*1+padded\_matrix[i+2, j+1]\*1+padded\_matrix[i+2, j+2]\*1)/9)

LPFFiltered= np.uint8(LPF\_matrix)

image\_final1 = cv2.merge((LPFFiltered, LPFFiltered, LPFFiltered))

cv2.imshow('LPF\_Filtered\_Image', image\_final1)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

LPF Filtered Image:



Median Filter:

#%%

#Median FILTER

import statistics

import math

new\_row= rows+2

new\_coloumn= coloumns+2

padded\_matrix = np.zeros((int(new\_row), int(new\_coloumn)))

s= -1

for i in range(1, new\_row-1):

    s= s+1

    t= 0

    for j in range(1, new\_coloumn-1):

        padded\_matrix[i, j]= image\_first\_band[s,t]

        t= t+1

p= 0

for q in range(1, new\_coloumn-1):

    padded\_matrix[p, q]= padded\_matrix[p+1, q ]

u= new\_row-1

for v in range(1, new\_coloumn-1):

    padded\_matrix[u, v]= padded\_matrix[u-1, v ]

padded\_matrix[0, 0]= padded\_matrix[0, 1]

padded\_matrix[new\_row-1, 0]= padded\_matrix[new\_row-1, 1]

padded\_matrix[0, new\_row-1] = padded\_matrix[0, new\_row-2]

padded\_matrix[new\_row-1, new\_row-1]= padded\_matrix[new\_row-1, new\_row-2]

b= 0

for a in range(1, new\_coloumn-1 ):

    padded\_matrix[a, b]= padded\_matrix[a, b+1]

d= new\_coloumn-1

for c in range(1, new\_coloumn-1 ):

    padded\_matrix[c, d]= padded\_matrix[c, d-1]

Median\_matrix= np.zeros((int(rows), int(coloumns)))

for i in range(0, rows):

    for j in range(0, coloumns):

        tupleA = (padded\_matrix[i, j], padded\_matrix[i, j+1], padded\_matrix[i, j+2],padded\_matrix[i+1, j], padded\_matrix[i+1, j+1], padded\_matrix[i+1, j+2], padded\_matrix[i+2, j], padded\_matrix[i+2, j+1], padded\_matrix[i+2, j+2])

        Median\_matrix[i, j]= statistics.median(tupleA)

MedianFiltered= np.uint8(Median\_matrix)

image\_final2 = cv2.merge((MedianFiltered, MedianFiltered, MedianFiltered))

cv2.imshow('Median\_Filtered\_Image', image\_final2)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

Median Filtered Image:



Comments:

Low pass filter removed the high frequency noise in some way but not completely. Median filter removed the salt and peeper noise better than LPF while avoiding blurring.

**Problem:2**

Script:

#%%

#Problem: 02

#importing cv2

import cv2

# Using cv2.imread() method

img = cv2.imread('4\_2.bmp')

# Displaying the image using cv2.imshow()

cv2.imshow('Original Image', img)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

# %%

img1= img[:, :, 0]

img2= img[:, :, 1]

img3= img[:, :, 2]

# %%

row = img1.shape[0]

coloumn= img1.shape[1]

# %%

#First\_Band

#Extracting a 3 by 3 patch around each pixel and forming matrix

List1 = []

for i in range(1, row-1):

    for j in range(1, coloumn-1):

        list1=[]

        list1.append(img1[i-1, j-1])

        list1.append(img1[i-1, j])

        list1.append(img1[i-1, j+1])

        list1.append(img1[i, j-1])

        list1.append(img1[i, j])

        list1.append(img1[i, j+1])

        list1.append(img1[i+1, j-1])

        list1.append(img1[i+1, j])

        list1.append(img1[i+1, j+1])

        List1.append(list1)

matrix1 = np.array(List1)

mask1= np.array([[-1], [-1], [-1], [-1],[9],[-1], [-1], [-1], [-1]]) #HPF MASK

C1=numpy.matmul(matrix1, mask1) #Convolution mask and matrix

C1A = C1.reshape(row-2, coloumn-2) #reshape to (height-2, width-2)

C1A= np.uint8(C1A)

# %%

#Second\_Band

#Extracting a 3 by 3 patch around each pixel and forming matrix

List2 = []

for i in range(1, row-1):

    for j in range(1, coloumn-1):

        list2=[]

        list2.append(img2[i-1, j-1])

        list2.append(img2[i-1, j])

        list2.append(img2[i-1, j+1])

        list2.append(img2[i, j-1])

        list2.append(img2[i, j])

        list2.append(img2[i, j+1])

        list2.append(img2[i+1, j-1])

        list2.append(img2[i+1, j])

        list2.append(img2[i+1, j+1])

        List2.append(list2)

matrix2 = np.array(List2)

mask2= np.array([[-1], [-1], [-1], [-1],[9],[-1], [-1], [-1], [-1]]) #HPF MASK

C2=numpy.matmul(matrix2, mask2) #Convolution mask and matrix

C1B = C2.reshape(row-2, coloumn-2) #reshape to (height-2, width-2)

C1C= np.uint8(C1B)

# %%

#Third\_Band

#Extracting a 3 by 3 patch around each pixel and forming matrix

List3 = []

for i in range(1, row-1):

    for j in range(1, coloumn-1):

        list3=[]

        list3.append(img3[i-1, j-1])

        list3.append(img3[i-1, j])

        list3.append(img3[i-1, j+1])

        list3.append(img3[i, j-1])

        list3.append(img3[i, j])

        list3.append(img3[i, j+1])

        list3.append(img3[i+1, j-1])

        list3.append(img3[i+1, j])

        list3.append(img3[i+1, j+1])

        List3.append(list3)

matrix3 = np.array(List3)

mask3= np.array([[-1], [-1], [-1], [-1],[9],[-1], [-1], [-1], [-1]]) #HPF MASK

C3=numpy.matmul(matrix3, mask3) #Convolution mask and matrix

C1D = C3.reshape(row-2, coloumn-2) #reshape to (height-2, width-2)

C1E= np.uint8(C1D)

# %%

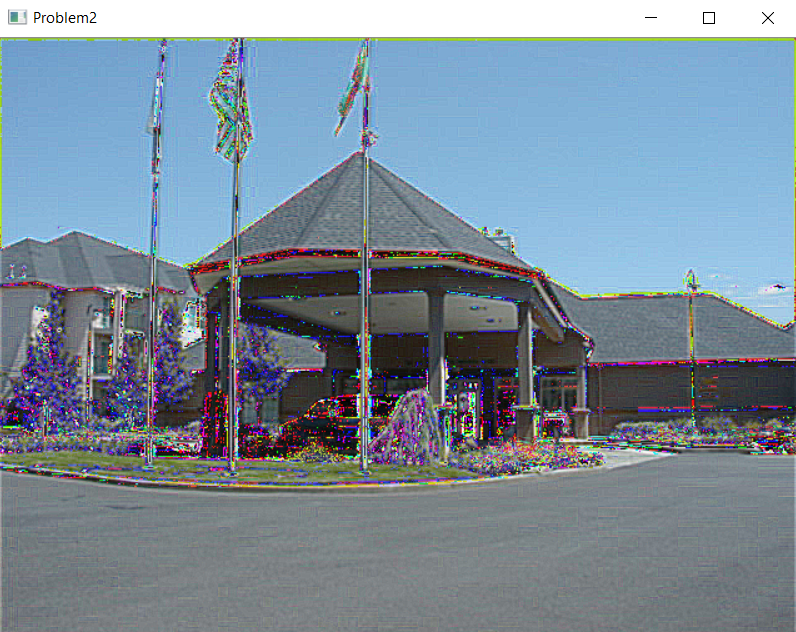
image\_final3 = cv2.merge((C1A, C1C, C1E))

cv2.imshow('Problem2', image\_final3)

#Maintain output window until user presses a key

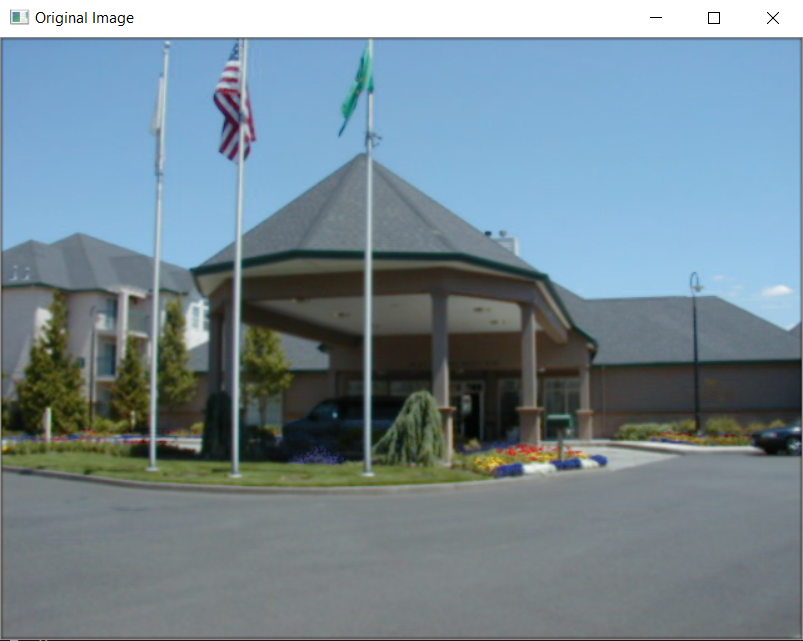
cv2.waitKey(0)

cv2.destroyAllWindows()



Problem3:

Original Image:



Code:

#%%

#Problem: 03

#importing cv2

import cv2

# Using cv2.imread() method

img = cv2.imread('4\_2.bmp')

# Displaying the image using cv2.imshow()

cv2.imshow('Original Image', img)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

import numpy as np

image\_first\_band = img[:,:,0]

rows = image\_first\_band.shape[0]

coloumns= image\_first\_band.shape[1]

#%%

#High PASS FILTER

#padding

import math

new\_row= rows+2

new\_coloumn= coloumns+2

padded\_matrix = np.zeros((int(new\_row), int(new\_coloumn)))

s= -1

for i in range(1, new\_row-1):

    s= s+1

    t= 0

    for j in range(1, new\_coloumn-1):

        padded\_matrix[i, j]= image\_first\_band[s,t]

        t= t+1

#%%

p= 0

for q in range(1, new\_coloumn-1):

    padded\_matrix[p, q]= padded\_matrix[p+1, q ]

u= new\_row-1

for v in range(1, new\_coloumn-1):

    padded\_matrix[u, v]= padded\_matrix[u-1, v ]

#%%

"""

padded\_matrix[0, 0]= padded\_matrix[0, 1]

padded\_matrix[new\_row-1, 0]= padded\_matrix[new\_row-1, 1]

padded\_matrix[0, new\_row-1] = padded\_matrix[0, new\_row-2]

padded\_matrix[new\_row-1, new\_row-1]= padded\_matrix[new\_row-1, new\_row-2]

"""

#%%

b= 0

for a in range(0, 482):

    padded\_matrix[a, b]= padded\_matrix[a, b+1]

d= 641

for c in range(0, 482):

    padded\_matrix[c, d]= padded\_matrix[c, d-1]

HPF\_matrix= np.zeros((int(rows), int(coloumns)))

for i in range(0, rows):

    for j in range(0, coloumns):

        HPF\_matrix[i, j]= math.floor((padded\_matrix[i, j]\*1+padded\_matrix[i, j+1]\*1+padded\_matrix[i, j+2]\*1+

                          padded\_matrix[i+1, j]\*1+padded\_matrix[i+1, j+1]\*(-8)+padded\_matrix[i+1, j+2]\*1+

                          padded\_matrix[i+2, j]\*1+padded\_matrix[i+2, j+1]\*1+padded\_matrix[i+2, j+2]\*1)/9)

HPFFiltered= np.uint8(HPF\_matrix)

image\_final4 = cv2.merge((HPFFiltered, HPFFiltered, HPFFiltered))

cv2.imshow('HPF\_Filtered\_Image', image\_final4)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

HPF filtered image:

