Problem 1: Code

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Sat Sep 28 09:04:41 2019

@author: Sarang Joshi

"""

import numpy as np

import cv2

"""

Class to apply gaussian filters on an image

"""

class imageBlur:

# Class Attribute

input\_matrix = None

filepath = None

gaussian\_1D\_row = None

gaussian\_1D\_col = None

gaussian\_2D = None

"""

Initialize

"""

def \_\_init\_\_(self, filepath):

self.filepath = filepath

self.input\_matrix = cv2.imread(filepath,cv2.IMREAD\_GRAYSCALE)

self.gaussian\_1D\_row = self.get\_gaussian\_1D\_row()

self.gaussian\_1D\_col = self.get\_gaussian\_1D\_col()

self.gaussian\_2D = self.get\_gaussian\_2D()

"""

instance method to display the input image

"""

def display\_image(self, display\_matrix,captions = 'input\_image',):

if display\_matrix is None:

display\_matrix = self.input\_matrix

cv2.imshow(captions, display\_matrix)

cv2.waitKey(0)

"""

returns the matrix for the original image

"""

def get\_input\_matrix(self):

return self.input\_matrix

"""

returns a gaussian kernel 1\*11

"""

def get\_gaussian\_1D\_row(self):

return np.array([0.000003,0.000229,0.005977,0.060598,0.24173,0.382925,0.24173,0.060598, 0.005977, 0.000229, 0.000003])

"""

returns a gaussian kernel 11\*1

"""

def get\_gaussian\_1D\_col(self):

b = np.array([0.000003,0.000229,0.005977,0.060598,0.24173,0.382925,0.24173,0.060598, 0.005977, 0.000229, 0.000003])

return b.transpose()

"""

calculates the sum of the input kernel and the input matrix, same shape for 1D

"""

def sum\_neighbours\_1D(self, input\_kernel, input\_matrix):

kernel\_size = len(input\_kernel)

sum = 0

for i in range(kernel\_size):

sum+= input\_kernel[i] \* input\_matrix[i]

return sum

"""

applies a 1D gaussian kernel on an input matrix row major

"""

def apply\_1DGaussian\_row(self, inputMatrix, kernel\_1D):

# Set default values

if inputMatrix is None:

inputMatrix=self.input\_matrix

if kernel\_1D is None:

kernel\_1D = self.gaussian\_1D\_row

kernel\_rows = len(kernel\_1D)

filter\_window = int(kernel\_rows/2)

height = len(inputMatrix)

width = len(inputMatrix[0])

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

for i in range(1,len(inputMatrix)+1):

for j in range(filter\_window, len(inputMatrix[0])-filter\_window):

subMatrix\_1D = inputMatrix[i-1:i,j - filter\_window: j+filter\_window+1]

value = self.sum\_neighbours\_1D(kernel\_1D, subMatrix\_1D.reshape(len(subMatrix\_1D[0])))

outputMatrix[i-1][j] = value

return outputMatrix

"""

calculates a sum of neighbouring pixels by running a filter on a sub\_matrix

"""

def sum\_neighbours\_2D(self, sub\_matrix, sm\_row, sm\_col):

filter\_2D = self.gaussian\_2D

f\_row = len(filter\_2D)

f\_col = len(filter\_2D[0])

if (f\_row != sm\_row or f\_col != sm\_col):

print (exit)

else:

list\_filter = list (filter\_2D.reshape(f\_row\*f\_col))

list\_sm = list (sub\_matrix.reshape(sm\_row\*sm\_col))

sum = 0

for i in range(len(list\_filter)):

sum += list\_filter[i] \* list\_sm[i]

return sum

"""

applies a 2D gaussian filter on the input matrix and returns an output matrix

"""

def apply\_2D\_filter(self):

#testMatrix = np.arange(100).reshape(10,10)

height = len(self.input\_matrix)

width = len (self.input\_matrix[0])

#filter = np.array([(0.111108,0.111113,0.111108), (0.111113,0.111118,0.111113), (0.111108,0.111113,0.111108)])

filter\_2D = self.gaussian\_2D

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

ht\_filter = int(len(filter\_2D)/2)

wd\_filter = int(len(filter\_2D[0])/2)

# This works only for a 3\*# generalize in other cases

for i in range(ht\_filter,len(self.input\_matrix)-ht\_filter):

for j in range(wd\_filter,len(self.input\_matrix[0])-wd\_filter):

#handleBoundary()

sub\_matrix = self.extract\_sub\_matrix(self.input\_matrix, i,j,ht\_filter,wd\_filter)

avg\_sum = self.sum\_neighbours\_2D(sub\_matrix, len(sub\_matrix),len(sub\_matrix[0]))

output\_matrix[i][j] = avg\_sum

# do the main operation here

#val = filter.temp

#print (outputMatrix)

return (output\_matrix)

"""

extracts a sub matrix of br\_row and br\_column from the i, j coordinates in in an input matrix

"""

def extract\_sub\_matrix(self, input\_matrix, i, j, br\_row, br\_col):

return input\_matrix[i-br\_row:i+br\_row+1,j-br\_col:j+br\_col+1]

"""

2D gaussian mask - method to create a 2D gaussian filter

"""

def get\_gaussian\_2D(self, shape=(11,11),sigma=1.5):

m,n = [(ss-1.)/2. for ss in shape]

y,x = np.ogrid[-m:m+1,-n:n+1]

h = np.exp( -(x\*x + y\*y) / (2.\*sigma\*sigma) )

h[ h < np.finfo(h.dtype).eps\*h.max() ] = 0

sumh = h.sum()

if sumh != 0:

h /= sumh

return h

"""

Code to debug by counting common values in two matrices and common zero values

"""

def count\_common\_pixels(self, mat1, mat2):

common = 0

common\_0 = 0

for i in range(len(mat1)):

for j in range(len(mat1[0])):

if mat1[i][j] == mat2[i][j]:

common+=1

if mat1[i][j] == 0:

common\_0 +=1

print (common)

print (common\_0)

"""

applies a 1D gaussian kernel on an input matrix col major

"""

def apply\_1DGaussian\_col(self, inputMatrix, kernel\_1D):

# Set default values

if inputMatrix is None:

inputMatrix=self.input\_matrix

if kernel\_1D is None:

kernel\_1D = self.gaussian\_1D\_row

kernel\_rows = len(kernel\_1D)

filter\_window = int(kernel\_rows/2)

height = len(inputMatrix)

width = len(inputMatrix[0])

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

for i in range(1,len(inputMatrix[0])+1):

for j in range(filter\_window, len(inputMatrix)-filter\_window):

#print(str(j) +"::"+ str(i-1))

subMatrix\_1D = inputMatrix[j - filter\_window: j+filter\_window+1,i-1:i]

value = self.sum\_neighbours\_1D(kernel\_1D, subMatrix\_1D.reshape(len(subMatrix\_1D)))

outputMatrix[j][i-1] = value

return outputMatrix

def main():

filepath = '/home/wickedshaman/Fall\_2019/ECE\_5554/Computer\_Vision/hw1/elephant.png'

i1 = imageBlur(filepath)

input\_matrix = i1.get\_input\_matrix()

# show input image

cv2.imshow('Original image',input\_matrix)

cv2.waitKey(0)

#apply 1D filter row-wise and display image

kernel\_1D\_row = i1.get\_gaussian\_1D\_row()

output\_matrix\_row = i1.apply\_1DGaussian\_row(input\_matrix, kernel\_1D\_row)

#i1.display\_image(output\_matrix\_row, "output 1D\_gaussian\_row")

#apply 1D filter col-wise and display image

kernel\_1D\_col = i1.get\_gaussian\_1D\_col()

output\_matrix\_col = i1.apply\_1DGaussian\_col(input\_matrix, kernel\_1D\_col)

#i1.display\_image(output\_matrix\_col, "output 1D\_gaussian\_col")

# apply both column and row filters

output\_matrix\_two\_1D\_filters = i1.apply\_1DGaussian\_col(output\_matrix\_row, kernel\_1D\_col)

i1.display\_image(output\_matrix\_two\_1D\_filters, "Output image 2\*1D filters")

# apply a 2D filter

output\_matrix\_2D = i1.apply\_2D\_filter()

i1.display\_image(output\_matrix\_2D, "Output image 2D filter")

# calculate and display the difference matrix

i1.count\_common\_pixels (output\_matrix\_two\_1D\_filters,output\_matrix\_2D )

diff\_matrix = np.subtract(output\_matrix\_two\_1D\_filters, output\_matrix\_2D)

i1.display\_image(diff\_matrix, "Difference image")

cv2.destroyAllWindows()

# Print statistics for difference matrix from the two filters

print("\nStatistics for difference matrix")

print(np.mean(diff\_matrix))

print(np.median(diff\_matrix))

print(np.var(diff\_matrix))

print("\nStatistics for 2\*1D filters output matrix")

print(np.mean(output\_matrix\_two\_1D\_filters))

print(np.median(output\_matrix\_two\_1D\_filters))

print(np.var(output\_matrix\_two\_1D\_filters))

print("\nStatistics for 2D filter output matrix")

print(np.mean(output\_matrix\_2D))

print(np.median(output\_matrix\_2D))

print(np.var(output\_matrix\_2D))

if \_\_name\_\_== "\_\_main\_\_":

main()

**Console output for problem 1**

**1> Statistics for difference matrix**

Mean = 125.16785757575758

Median = 13.0

Variance = 15614.246438985767

**2> Statistics for 2\*1D filters output matrix**

Mean = 149.79788181818182

Median = 171.0

Variance = 4479.087096725426

**3> Statistics for 2D filter output matrix**

Mean = 150.29770909090908

Median = 172.0

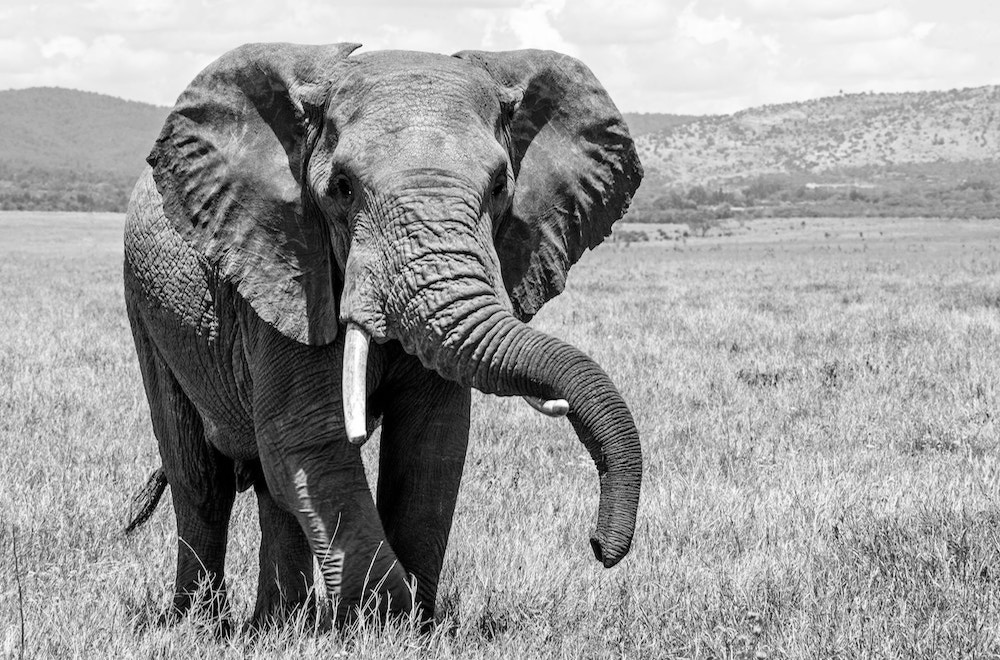
Variance = 4374.121984448705

**Discussion: As we analyze the mean , median and the variance for the output matrix after applying the filters and the difference matrix we observe the following**

1. **We find the median value for the difference to be 13. This is pretty good as ideally it should be as close to zero as possible.. The reason for the deviation from ideal value seems to be because of the filter functions used for generating the gaussian filters.**
2. **We also observed that the mean and the variance as well as the median generated for the output matrix by the 2D filter and the 1D filters is very similar.**

Images

1> Original image

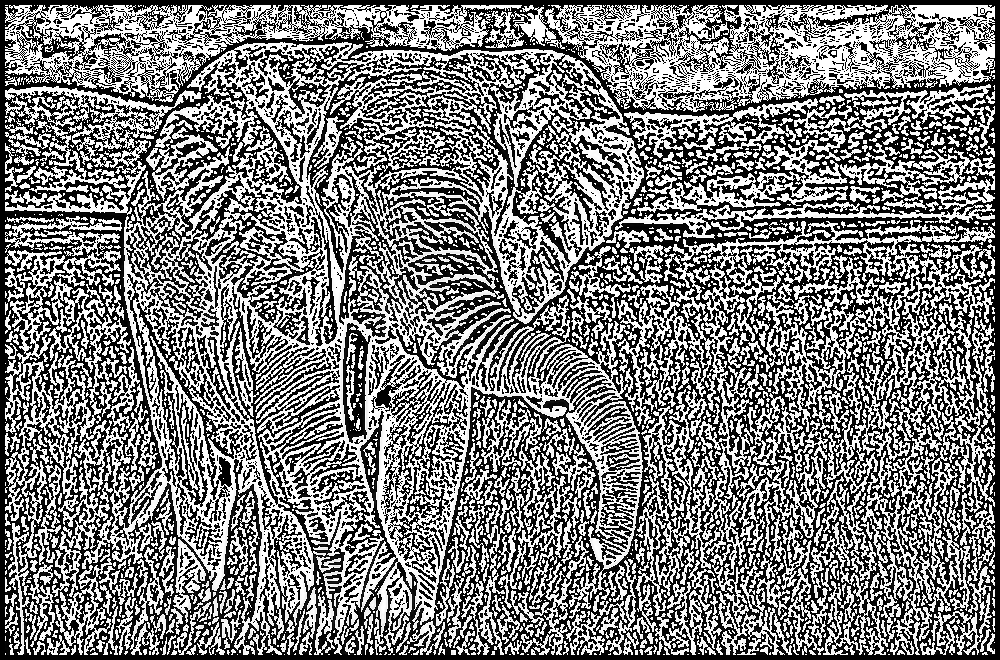


2> Image with 2D filter



3> Image with 1D filter





Lighthouse image

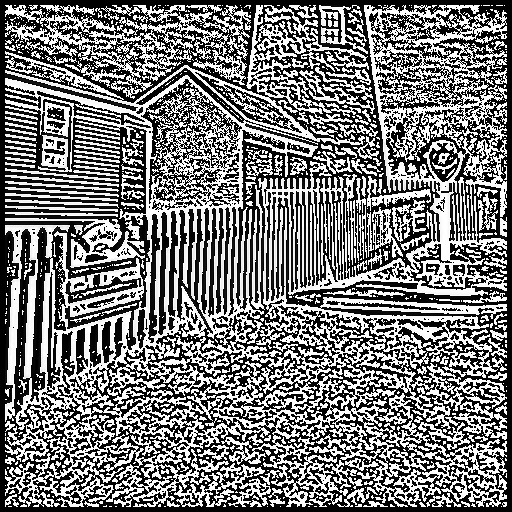
1-D Filter



2-D Filter



Difference

Statistics for difference matrix

120.51058959960938

10.0

15630.285066998564

Statistics for 2\*1D filters output matrix

110.16167068481445

106.0

2872.488659800365

Statistics for 2D filter output matrix

110.64229202270508

107.0

2761.823426686329

Problem 2: Code

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Sat Sep 28 19:16:17 2019

@author: Sarang Joshi

"""

import cv2

import numpy as np

from matplotlib import pyplot as plt

"""

def gauss(img):

midImage = cv2.GaussianBlur(img,(5,5),cv2.BORDER\_REPLICATE)

scale\_percent = 50

width = int(img.shape[1] \* scale\_percent / 100)

height = int(img.shape[0] \* scale\_percent / 100)

dim = (width, height)

# resize image

subImage= cv2.resize (midImage,dim, interpolation = cv2.INTER\_AREA)

return subImage

"""

def get\_image(path):

image = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

return image

def display\_image(input\_matrix, captions = "Original image"):

cv2.imshow(captions,input\_matrix)

"""

apply a gaussian filter and resize the image by a factor

"""

def gaussian\_blur(input\_image, resize\_factor):

blurred\_input\_image = cv2.GaussianBlur(input\_image,(5,5),cv2.BORDER\_REPLICATE)

columns = int(input\_image.shape[1] \* resize\_factor)

rows = int(input\_image.shape[0] \* resize\_factor)

image\_dimensions = (columns, rows)

resized\_image = cv2.resize (blurred\_input\_image, image\_dimensions, interpolation = cv2.INTER\_AREA)

return resized\_image

def laplace\_transform(input\_image):

blurred\_input\_image = cv2.GaussianBlur(input\_image,(5,5),cv2.BORDER\_REPLICATE)

laplace\_image=cv2.subtract(input\_image,blurred\_input\_image)

return laplace\_image

def fft(img):

dft = cv2.dft(np.float32(img),flags = cv2.DFT\_COMPLEX\_OUTPUT)

dft\_shift = np.fft.fftshift(dft)

magnitude\_spectrum = 20\*np.log(cv2.magnitude(dft\_shift[:,:,0],dft\_shift[:,:,1]))

return magnitude\_spectrum

def plot\_graphs(img\_magnitude, laplace\_magnitude, levels):

plt.subplot(121),plt.imshow(img\_magnitude, cmap = 'gray')

plt.title('Magnitude Gaussian{!s}'.format(levels)), plt.xticks([]), plt.yticks([])

plt.subplot(122),plt.imshow(laplace\_magnitude, cmap = 'gray')

plt.title('Magnitude Laplacian{!s}'.format(levels)), plt.xticks([]), plt.yticks([])

levels+=1

plt.show()

def write\_image(list\_of\_images, image\_name):

level = 0

for image in list\_of\_images:

image\_name = image\_name + '\_' + str(level) + ".JPEG"

cv2.imwrite(image\_name, image)

level+=1

def transform\_image(input\_image, iterations):

list\_gaussians = []

list\_laplacians = []

levels = 0

for i in range (iterations):

outImage = gaussian\_blur(input\_image, 0.5)

list\_gaussians.append(outImage)

lImage= laplace\_transform(input\_image)

list\_laplacians.append(lImage)

image\_magnitude=fft(input\_image)

laplace\_magnitude=fft(lImage)

input\_image = outImage

cv2.imshow('Image after applying gaussian ',outImage)

cv2.imshow('mage after applying laplace transform ',lImage)

plot\_graphs(image\_magnitude, laplace\_magnitude, levels)

levels+=1

cv2.waitKey(0) # waits until a key is pressed

write\_image(list\_gaussians, 'gaussian\_level')

write\_image(list\_laplacians, 'laplacian\_level')

def main():

# Load an image in grayscale

image\_path = '/home/wickedshaman/Fall\_2019/ECE\_5554/Computer\_Vision/hw1/elephant.png'

input\_image = get\_image(image\_path)

display\_image(input\_image)

# Apply gaussian blur, apply laplace transform and fft on image while resizing

transform\_image(input\_image, 5)

cv2.destroyAllWindows()

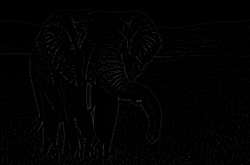
if \_\_name\_\_== "\_\_main\_\_":

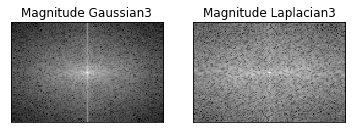
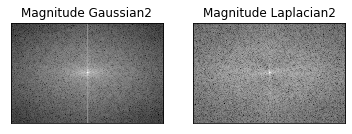
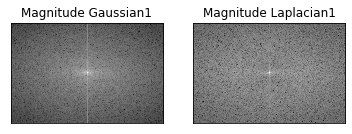
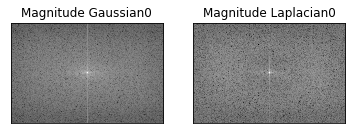
main()

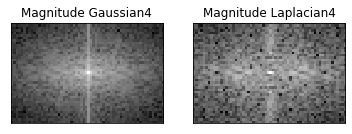
**Gaussian Image outputs**



**Laplacian images**

****

****

****