Project 1 Questions

Instructions

- 4 questions.
- Write code where appropriate.
- Feel free to include images or equations.

Questions

Q1: Explicitly describe image convolution: the input, the transformation, and the output. Why is it useful for computer vision?

A1: Image convolution is a mathematical operation that includes multiplication and addition. Two matrices, one of which has to have odd dimensions(The kernel), are used as input in the operations most basic form. For each pixel i, convolution adds the pixel value with its neighbouring pixels, all of which are weighted by the kernel. The process is iteratively repeated for every pixel. As for the output, given two matrices of dimensions $M \times N$ and $K \times L$, the output is a matrix of the size $(M + K - 1) \times (N + L - 1)$.Convolution is an essential tool in Computer vision since it can be used to transform the matrix in a way that allows objects edges only to be seen and therefore help in the detection and identification of said objects(edge detection, sobel and emboss filters). Convolution can also be used to enhance images in several ways such as sharpening, noise removal, and blurring, which can be useful in specific applications such as computational photography and aiding with scene and color detection in mobile phones.

Q2: What is the difference between convolution and correlation? Construct a scenario which produces a different output between both operations.

Please use scipy.ndimage.convolve and scipy.ndimage.correlate to experiment!

A2: Convolution and Correlation preform the same multiplication and addition operations on matrices, with the only difference being that the kernel is flipped in convolution before addition and multiplication. In correlation no flipping occurs. This means that convolving one image with another using *scipy.ndimage.convolve* and correlating the two images using *scipy.ndimage.correlate* with one being already flipped should result in the same output. Usually convolution and correlation also produce the same output when the kernel is symmetric, as flipping a symmetric kernel results in no change. A good scenario where convolution and correlation produce different results is when an image is correlated and convolved with a right sobel kernel. A typical right sobel matrix is as follows:

1	0	-1
2	0	-2
1	0	-1

If the kernel above was used in both correlation and convolution the convolution result would be equivalent to a left sobel correlation.

Correlation

Convolution

Q3: What is the difference between a high pass filter and a low pass filter in how they are constructed, and what they do to the image? Please provide example kernels and output images.

A3:

A low pass filter is constructed by creating a 2-dimensional Gaussian matrix or by creating a mean filter through dividing a matrix of ones by the number of elements inside said matrix. This causes the value for each pixel to be influenced by its neighbours in decreasing amount as the neighbour gets more distant, or equally by all neighbours respectively. The output of applying the mean or Gaussian filter to an image is a blurry copy of the original.

A high pass filter is constructed by creating a 2-dimensional Laplacian matrix or creating an edge detection matrix in which all elements but the core of the kernel are negatives, while the core is positive and scaled accordingly. This causes the value for each output pixel to only be bright enough when sudden changes in brightness occur between each pixel and its neighbours. The output of applying a high pass filter is usually an image where only the edges are visible and it is usually used in edge detection.

Low pass filter

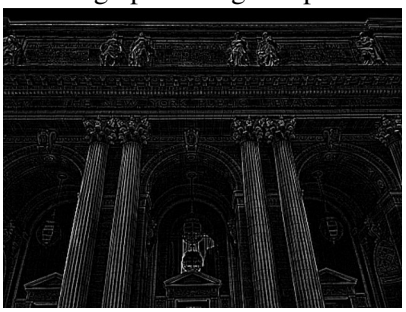
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

High pass filter

-1	-1	-1
-1	8	-1
-1	-1	-1

Low pass image output

High pass image output



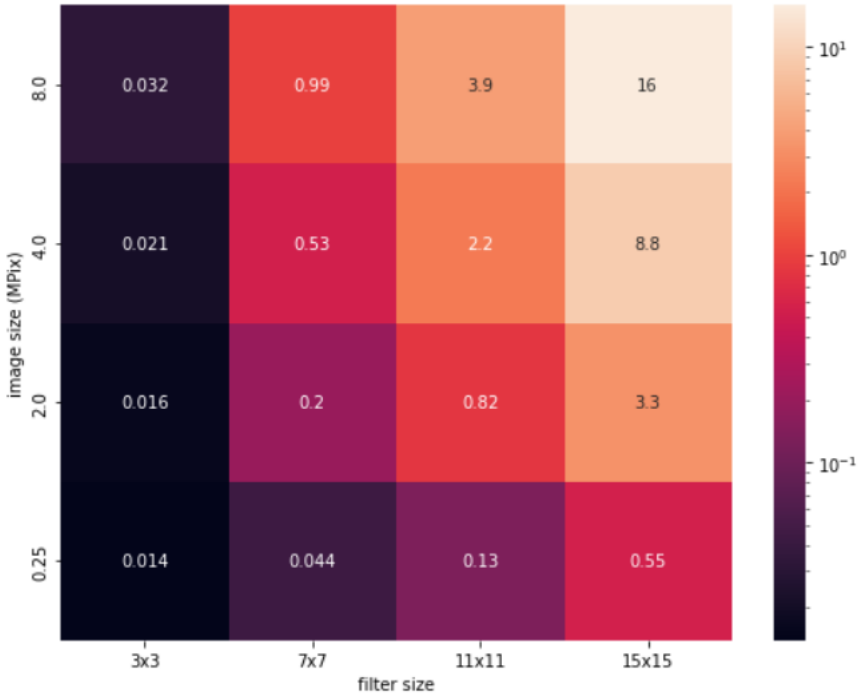
Q4: How does computation time vary with filter sizes from 3×3 to 15×15 (for all odd and square sizes), and with image sizes from 0.25 MPix to 8 MPix (choose your own intervals)? Measure both using *scipy.ndimage.convolve* or *scipy.ndimage.correlate* to produce a matrix of values. Use the *skimage.transform* module to vary the size of an image. Use an appropriate charting function to plot your matrix of results, such as *Axes3D.scatter* or *Axes3D.plot_surface*.

Do the results match your expectation given the number of multiply and add operations in convolution?

Image: RISDance.jpg (in the project directory).

A4: for each iteration in the convolution process, for a kernel with size $N \times N$, with N being an odd value ranging from 3 to 15, $N \times N$ multiplications are made and $(N \times N - 1)$ additions follow. This repeats for each pixel in the other image, which means it repeats K times with K ranging from 0.25×10^6 to 8×10^6

Matrix of Results:



Code :

```
from scipy.signal import convolve2d, correlate2d
import numpy as np
import cv2
from matplotlib import pyplot as plt
from scipy import ndimage
from skimage.transform import rescale
import time
import seaborn as sn
from matplotlib.colors import LogNorm, Normalize


img = cv2.imread('RISDance.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)) ;

imgsize = (img.shape[0]*img.shape[1])/(10**6)
sizes=np.array([0.25 ,2, 4, 8])
ratio=sizes/imgsize

img0= rescale(img, ratio[0], mode='reflect', multichannel=True)
img1= rescale(img, ratio[1], mode='reflect', multichannel=True)
img2= rescale(img, ratio[2], mode='reflect', multichannel=True)
img3= rescale(img, ratio[3], mode='reflect', multichannel=True)

def create_mean_filter(ksize):
    assert ksize % 2 != 0
    mean_filter = np.ones((ksize, ksize, 3), dtype=np.uint8) * (1/(ksize**
    return mean_filter

kernel0 = create_mean_filter(3)
kernel1 = create_mean_filter(7)
kernel2 = create_mean_filter(11)
kernel3 = create_mean_filter(15)

imglist= [img0, img1, img2, img3]
kernellist=[kernel0, kernel1, kernel2, kernel3 ]

exec_time = np.ndarray((4,4))
for i in range(4):
    for j in range (4):
        imgtmp,kerneltmp=imglist[j],kernellist[i]
        start_time = time.time()
        output = ndimage.convolve(imgtmp,kerneltmp, mode ='constant')
        endtime = time.time() - start_time
        exec_time[i,j]=endtime
```

```
plt.figure(figsize=(9,7));
hm = sn.heatmap(data=exec_time, annot=True,norm=LogNorm())
hm.invert_yaxis()
hm.set_xticklabels(['3x3','7x7','11x11','15x15'])
hm.set_xlabel('filter size')
hm.set_yticklabels(sizes)
hm.set_ylabel('image size (MPix)')
plt.show();
```