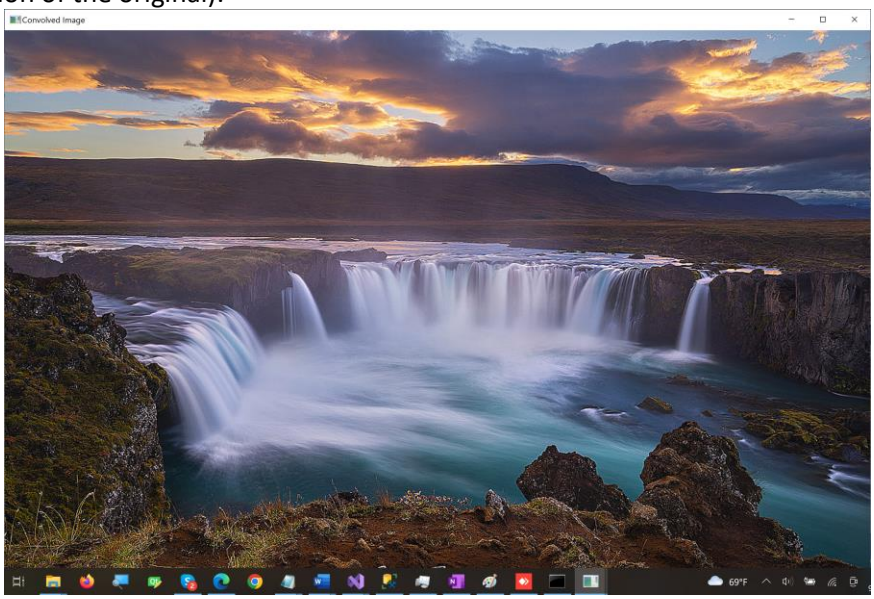
Computer Vision

Assignment #2 - Convolution Filters

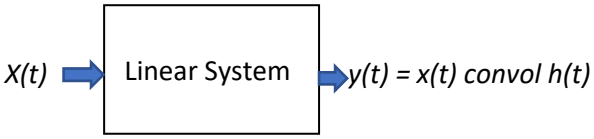
In this assignment, you will be deriving and implementing (in Python) some of the important convolution filters. A convolution filter works by doing a convolution of the kernel (a small matrix) with a given image (a large matrix). The resulting image gets enhanced in certain aspects depending upon the kernel being applied. For example, if we do the convolution of the following image with a kernel of:

$$kernel = \begin{bmatrix} 0 & -1 & -0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Image after doing the convolution with the kernel (if you look carefully, you will see that the following is sharpened version of the original):



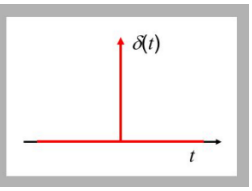
Convolution – Fundamental Concepts: Convolution is a fundamental principle followed by linear systems (e.g., our vision system) as:



where h(t) is the output of the system when input is an impulse *δ(t).* Thus, if we know the impulse response of a system, then we can determine the output of a system by simply doing the convolution of the input with the impulse response in the time domain.

The impulse function *δ(t)* is also known as the Dirac delta function. It has a magnitude of infinity and width of 0 and is positioned at t=0, as:

$$\delta(t) \equiv \begin{cases} \infty \text{ if } t = 0 \\ 0 \text{ if } t \neq 0 \end{cases}$$



The delta function has the following important properties.

$$\delta(x) = \begin{cases} +\infty & x = 0 \\ 0 & x \neq 0 \end{cases}$$

$$\int_{-\infty}^{\infty} \delta(x)dx = 1$$

$$\int_{-\infty}^{\infty} f(x)\delta(x-\xi)dx = f(\xi)$$

The convolution in time domain is mathematically defined as (* is often meant to be the convolution):

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$

which is flipping one of the two signals involved in the convolution, sliding it, and accumulating the result by integrating the product of the two signals.

An important property of convolution is that convolution in time domain is equivalent to multiplication in frequency domain. In any system when the signal or the input changes very quickly, it generates many frequencies in the frequency domain. For example, the Fourier transform (which represents frequency content) of an impulse contains all frequencies (as it involves extreme quick change in time domain).

Fourier Transform of delta function contains all frequencies:

If we do convolution of the delta function with any other function, we get the same function back, i.e., *f(x) * δ(t) = f(x)*

Convolution can be defined in two or higher dimensions.

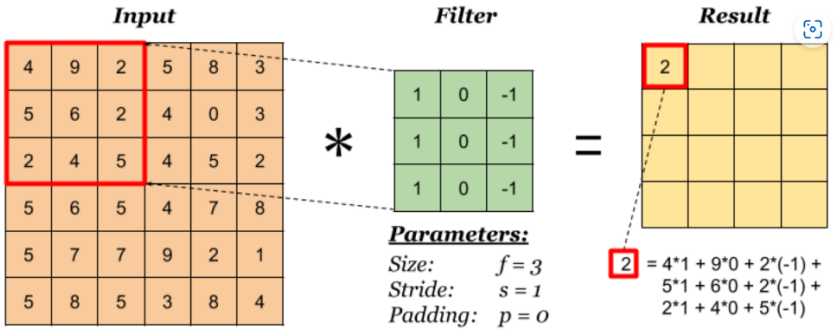
Discrete Convolution:
Images are defined in terms of discrete values i.e., pixels that take on values from 0 to 255. Discrete convolution in one dimension is defined as (integral becomes summation):

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

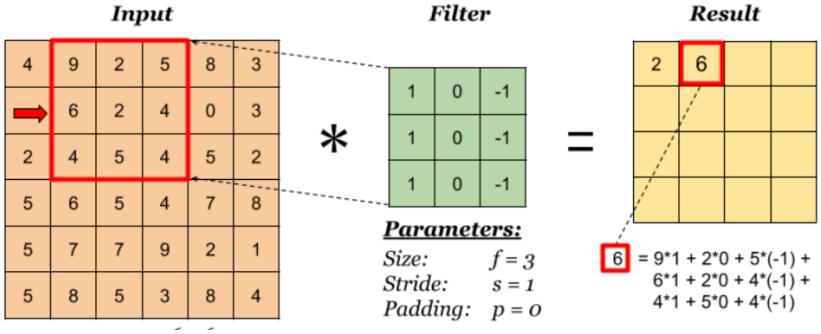
Since images are two dimensional, the convolution in two dimensions is defined as:

$$y(n_1, n_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} h(k_1, k_2) x(n_1 - k_1, n_2 - k_2)$$

which simply means rotate by 180 degrees one of the functions in the convolution, then accumulate the result as you slide and multiply the corresponding values. Here is a pictorial example of 2-d convolution (picture is from : https://indoml.com/2018/03/07)



Then we slide the kernel one pixel to the right to produce the second output as.



Designing Convolution Filters: Depending upon the values and the size of a kernel, the output image after convolution by the kernel will be enhanced accordingly.

Identity Kernel:
Identity kernel does not modify the image. Thus it is the equivalent of the delta function in the image or pixel space as convolution with a delta function does not change the function (or image in this case).
A 3x3 identity kernel appears as:

$$i = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Averaging (Low Pass) Filter Kernel:

The sum of values in a kernel should equal to 1. Thus a simple averaging filter which acts as low pass filter in the frequency domain appears as:

$$l = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} /9$$

High Pass Filter Kernel:

A high pass filter kernel can be easily derived by subtracting the all pass kernel (identity) from the low pass filter (average), i.e.,

$$h = i - l = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} /9$$

$$h = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} /9$$

Sharpening Filter Kernel:

Sharpening of an image can be obtained by adding a high pass filtered image with the original image. We can use $\propto$ as a weighting factor to control the amount of high pass fitered image to be added.

$$s = (1 - \propto) original\ image + \propto high\ pass\ filtered\ image$$

$$s = (1 - \alpha) \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \alpha \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} /9$$

$$s = \begin{bmatrix} -\alpha & -\alpha & -\alpha \\ -\alpha & 9 - \alpha & -\alpha \\ -\alpha & -\alpha & -\alpha \end{bmatrix} /9$$

Derivative Filter Kernels:

Derivatives tell us how fast the changes are occurring. Since images are two dimensional, derivative can be taken in the x or the y direction. To come up with a derivative filter kernel, we can use the property of Taylor series expansion which in simple terms tells us the value of a function at a nearby point in terms of the derivatives at the starting point as:

$$P(x \pm \Delta x, y \pm \Delta y) = P(x, y) \pm \Delta x \frac{dP}{dx} \pm \Delta y \frac{dP}{dy} + 0.5(\Delta x)^2 \frac{d^2P}{dx^2} + 0.5(\Delta y)^2 \frac{d^2P}{dy^2} + \dots$$

where $P(x, y)$ is the value of a pixel in the image at $x\ and\ y$ coordinates. $P(x \pm \Delta x, y \pm \Delta y)$ is some pixel in the local neighborhood of the center pixel $P(x, y)$ and $(\Delta x, \Delta y)$ are the integer offsets of the neighborhood pixel from the center pixel. In a 3×3 neighborhood, the offsets are ± 1. In a 5×5 neighborhood, the closest pixel will have offsets of ± 1 and the outer pixels in the neighborhood will have offsets of ± 2.

The gradient filtered image can be obtained as:

Gradient (Magnitude) Filtered Image $\sqrt{\left(\frac{dP}{dx}\right)^2 + \left(\frac{dP}{dy}\right)^2}$

To compute the Gradient filtered image, we use only the first order terms, so that the Taylor series expansion becomes:

$$P(x \pm \Delta x, y \pm \Delta y) = P(x, y) \pm \Delta x \frac{dP}{dx} \pm \Delta y \frac{dP}{dy}$$

The offsets for each pixel on the neighborhood can be expressed in table or matrix form as:

$$(\pm\Delta x, \pm\Delta y) = \begin{bmatrix} (-1,+1) & (0,+1) & (+1,+1) \\ (-1,0) & (0,0) & (+1,0) \\ (-1,-1) & (0,1) & (+1,-1) \end{bmatrix}$$

So, we can now express the value for each of neighborhood pixels in terms of the center pixel as:

$$P(x-1, y+1) = P(x, y) - \frac{dP}{dx} + \frac{dP}{dy} \quad (1)$$

$$P(x-1, y) = P(x, y) - \frac{dP}{dx} \quad (2)$$

$$P(x-1, y-1) = P(x, y) - \frac{dP}{dx} - \frac{dP}{dy} \quad (3)$$

$$P(x, y+1) = P(x, y) + \frac{dP}{dy} \quad (4)$$

$$P(x, y) = P(x, y) \quad (5)$$

$$P(x, y-1) = P(x, y) - \frac{dP}{dy} \quad (6)$$

$$P(x+1, y+1) = P(x, y) + \frac{dP}{dx} + \frac{dP}{dy} \quad (7)$$

$$P(x+1, y) = P(x, y) + \frac{dP}{dx} \quad (8)$$

$$P(x+1, y-1) = P(x, y) + \frac{dP}{dx} - \frac{dP}{dy} \quad (9)$$

If wee add the first three equations which represent the left column in a 3x3 neighborhood, we get:

$$\text{sum of left column of 3 pixels} = 3P(x, y) - 3\frac{dP}{dx} \quad (10)$$

Similarly, if we add equations 7-9, which represent the right column in a 3x3 pixel neighborhood, we get:

$$\text{sum of right column of 3 pixels} = 3P(x, y) + 3\frac{dP}{dx} \quad (11)$$

If we subtract equation 10 from equation 11, we obtain:

$$\text{sum of right column of 3 pixels} - \text{sum of left column of 3 pixels} = 6\frac{dP}{dx} \quad (12)$$

Equation 12 can be used to deduce the first derivative kernel in the x direction as:

$$\frac{dP}{dx} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} /6$$

A similar development can be used to determine the first derivative in the y direction as:

$$\frac{dP}{dy} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} /6$$

Thus the overall gradient filtered Image can e obtained as: $\sqrt{\left(\frac{dP}{dx}\right)^2 + \left(\frac{dP}{dy}\right)^2}$

Note that the equation 12 indicates to subtract the sum of pixels in the right column from the sum of pixels in the left column, so the following kernels are still valid for $\frac{dP}{dx}\ and\ \frac{dP}{dy}$ and referred to as the Sobel filter. Note also that multiplying a kernel by -1 usually does not cause a change in the output behavior.

$$\frac{dP}{dx} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$\frac{dP}{dy} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Note that if we wanted to have a 5x5 kernel, then the two derivative kernels become:

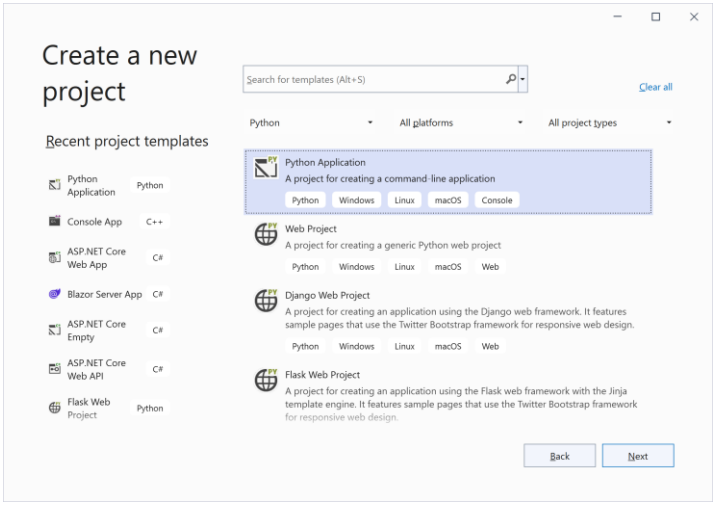
$$\frac{d}{dx} = \begin{bmatrix} -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \end{bmatrix}$$

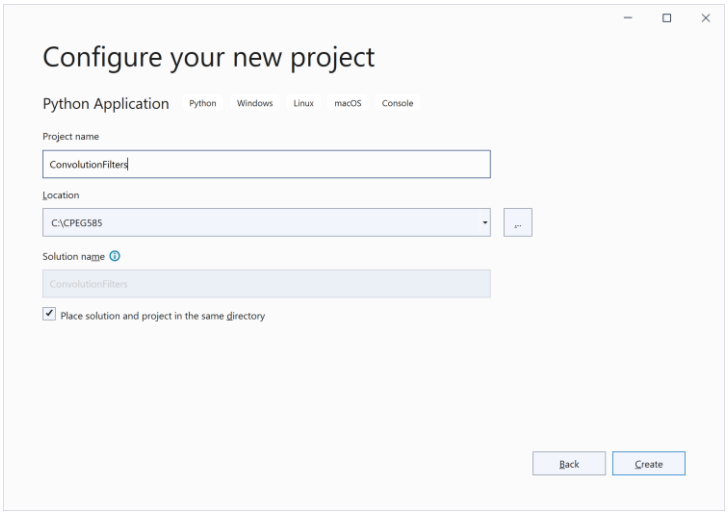
$$\frac{d}{dy} = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 & -1 \\ -2 & -2 & -2 & -2 & -2 \end{bmatrix}$$

Problem #1: Develop a Python application where a given convolution kernel can be specified and tested on an image.

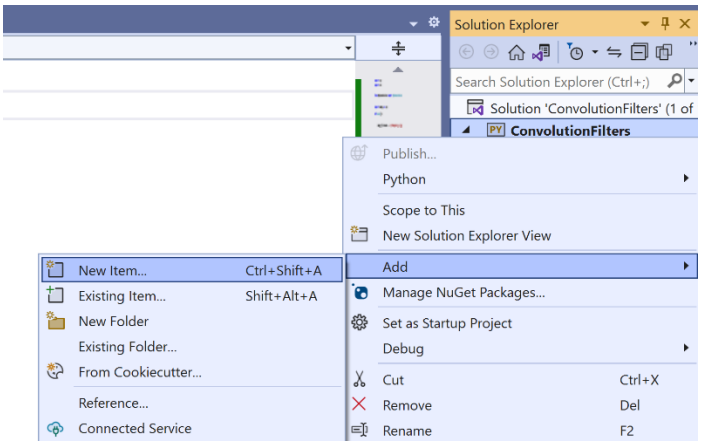
Partial Solution:

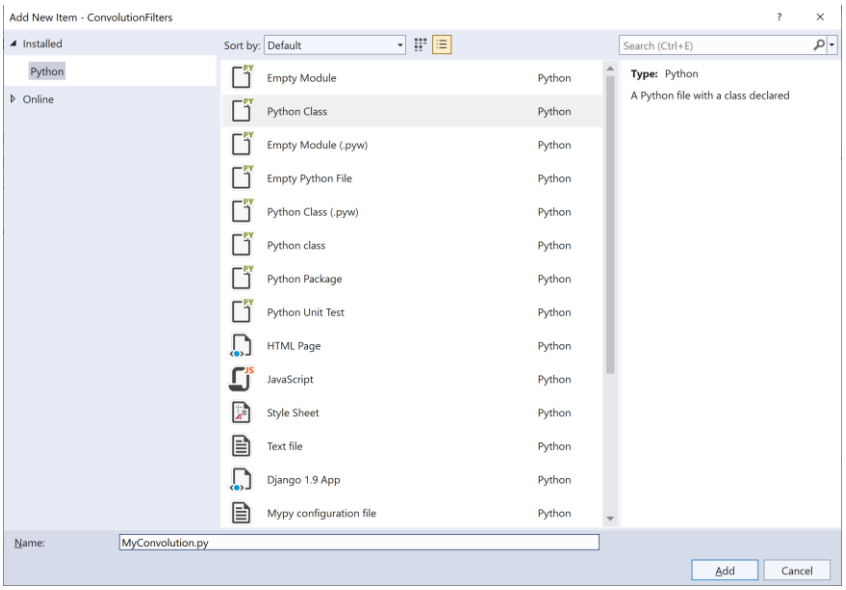
Using Visual Studio, create a new project of type Python application called ConvolutionFilters as shown below.





By right clicking on the project name, choose “Add New Item” and add a class called MyConvolution.py as shown below.





Then type the following code in MyConvolution.py.

```
import numpy as np

class MyConvolution(object):

    def convolve(self, img: np.array, kernel: np.array) -> np.array:
        # kernel is assumed to be square
        output_size = (img.shape[0]-kernel.shape[0]+1, img.shape[1]-
kernel.shape[0]+1)

        output_img = np.zeros((output_size[0],
output_size[1],img.shape[2]),dtype=img.dtype)
        kernel_size = kernel.shape[0]  # kernel size

        for i in range(output_size[0]):
             for j in range(output_size[1]):
                 for k in range(img.shape[2]):  # RGB
                    mat = img[i:i+kernel_size, j:j+kernel_size,k] # img values
at current kernel location
                    mat = mat.reshape((kernel_size,kernel_size))
                    # do element-wise multiplication and add the result
                    output_img[i, j, k] = np.clip(np.sum(np.multiply(mat,
kernel)),0,255)

        return output_img
```

As you can see from the above code, the class has a function called convolve which takes an image and a kernel as inputs, performs the convolution and returns the processed image.

Add a main method to the ConvolutionFilters.py by typing main and hitting the tab key. Then type the rest of the code in the ConvolutionFilters.py as:

```
import sys
import cv2
from MyConvolution import MyConvolution
import numpy as np

def main():
    image_filename = 'd:/temp/orig1.jpg'
    img = cv2.imread(image_filename)
    if img is None:
        print('Could not open or find the image: ', image_filename)
        exit(0)

    kernel = np.array([
    [0, -1, 0],
    [-1, 5, -1],
    [0, -1, 0]
    ])
    myconv = MyConvolution()
    conv_img = myconv.convolve(img,kernel)
    cv2.imshow('Original Image', img)
    cv2.imshow('Convolved Image', conv_img)
    cv2.waitKey()

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Problem #2: Test the program for the following kernels:

a) 3x3 kernel based averaging filter

b) 3x3 kernel based high pass filter

c) 3x3 kernel based sharpening filter with α of 0.5 and α of 0.3.

d) 3x3 kernel based Sobel filter

e) 5x5 kernel based average filter

f) 5x5 kernel based Sobel filter