**Digital Image Processing Lab**

**CEL-445**

Lab Journal: 5



Name: Omar

Class: BCE-7(A)

# Enrollment No: 01-132182-024

TASK1

**Task 1 Filter2D:**

The Filter2D operation convolves an image with the kernel. You can perform this operation on an image using the Filter2D() method of the imgproc class.

Using this function, we can create a convolution between the image and the given kernel for creating filters like smoothing and blurring, sharpening, and edge detection in an image. This function will simply convolute the 2d matrix with the image at pixel level and produce an output image. To understand this concept, we shall first skim through the concept of the kernel.

Kernel: A simple 2d matrix used in convolution or Convolution Matrix or a mask used to blur, sharpen and edge detect an image.

Working of the kernel: So, how this kernel works? Let’s see, we all know that images are represented as pixel values in OpenCV. These pixels are arranged as a matrix to form an image and as we know that a kernel is a simple 2d matrix with specific values in it based on the function of the kernel like if the kernel is used for blurring and sharpening the images are different.

Let us take an example, in this image take the first 3 rows and columns like a matrix and we have a kernel of 3 by 3 matrix. Each of the pixels in the image has a pixel value (i.e. pixel intensity) as shown. Now the convolution is done by multiplying the values of each pixel value with the kernel value in the respective places and adding all the values which were just weighted by the kernel by multiplying and forming one pixel (the center pixel which in this case it is [2,2]). And this method is repeated for the rest of the pixel value matrices in the image.

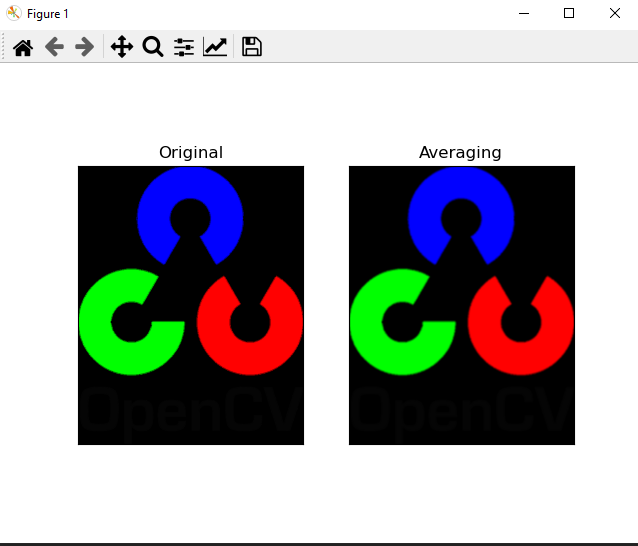
Diagram

Description automatically generated with low confidence

Some Common kernels are,

* Identity kernel
* Edge detection kernel
* Sharpening kernel
* Box blurring kernel
* Gaussian blurring kernel

import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
img = cv2.imread('img.png')  
  
kernel = np.ones((5,5),np.float32)/25  
dst = cv2.filter2D(img,-1,kernel)  
  
plt.subplot(121),plt.imshow(img),plt.title('Original')  
plt.xticks([]), plt.yticks([])  
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')  
plt.xticks([]), plt.yticks([])  
plt.show()

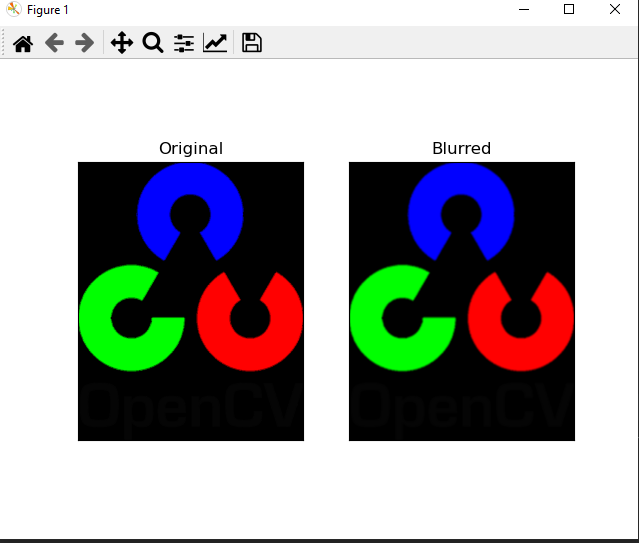


TASK2

**Task 2  Averaging**

This is done by convolving an image with a normalized box filter. It simply takes the average of all the pixels under the kernel area and replaces the central element. This is done by the function **cv2.blur**

import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
img = cv2.imread('img.png')  
  
blur = cv2.blur(img,(5,5))  
  
plt.subplot(121),plt.imshow(img),plt.title('Original')  
plt.xticks([]), plt.yticks([])  
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')  
plt.xticks([]), plt.yticks([])  
plt.show()

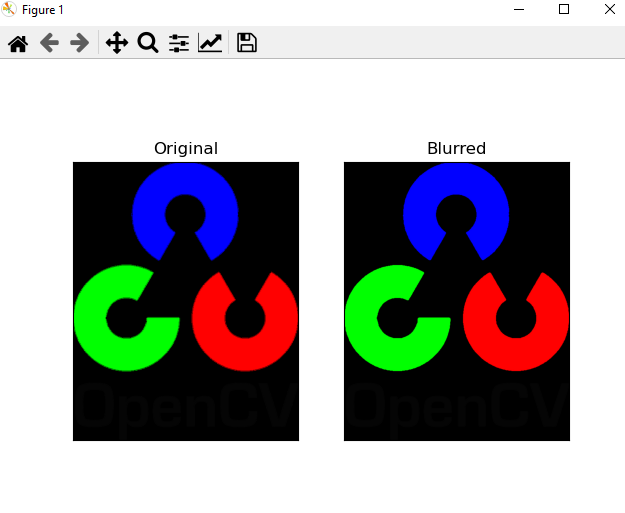


TASK3

**Task 3 Median Blurring**

Here, the function takes the median of all the pixels under the kernel area and the central element is replaced with this median value. This is highly effective against salt-and-pepper noise in an image. Interestingly, in the above filters, the central element is a newly calculated value which may be a pixel value in the image or a new value. But in median blurring, the central element is always replaced by some pixel value in the image. It reduces the noise effectively. Its kernel size should be a positive odd integer.

import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
img = cv2.imread('img.png')  
  
median = cv2.medianBlur(img,5)  
  
plt.subplot(121),plt.imshow(img),plt.title('Original')  
plt.xticks([]), plt.yticks([])  
plt.subplot(122),plt.imshow(median),plt.title('Blurred')  
plt.xticks([]), plt.yticks([])  
plt.show()



TASK4

**Task 4 Sobel Operator:**

This is obtained by multiplying the x, and y-derivative filters obtained above with some smoothing filter(1D) in the other direction. For example, a 3×3 Sobel-x and Sobel-y filter can be obtained as

Diagram

Description automatically generated

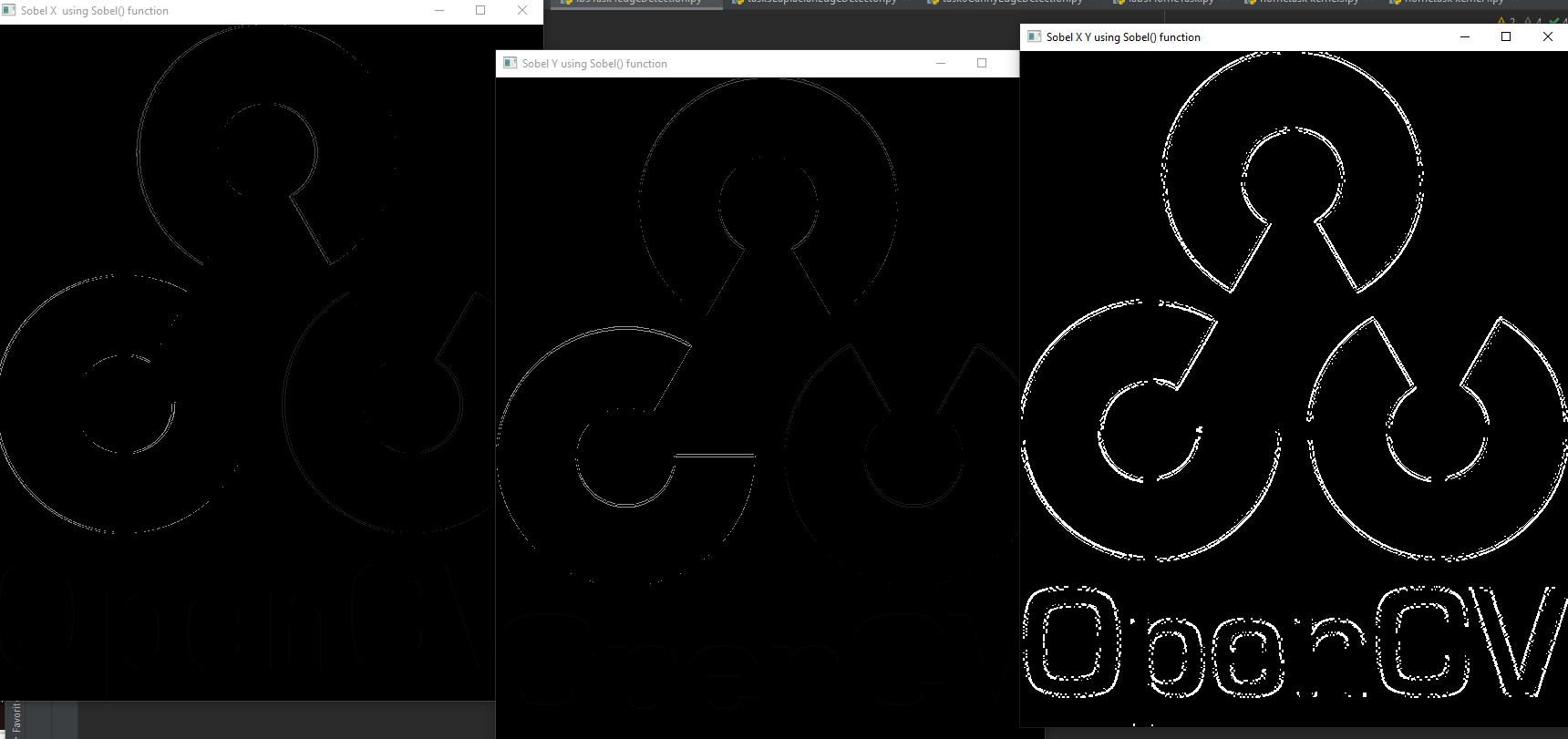
As we know that the Gaussian filter is used for blurring thus, the Sobel operator computes the gradient with smoothing. Thus this is less sensitive to noise. Because of separability property of the kernel, the Sobel operator is computationally efficient.

When we convolve these Sobel operators with the image, they estimate the gradients in the x, and y-directions(say Gx and Gy).

We can easily infer that the edge direction or the angle will be positive for the transition from dark to white and negative otherwise. Now, let’s see how to do this using OpenCV-Python

**Code dx=1, dy=1**

import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
#img = cv2.imread('img.png')  
img = cv2.imread('img.png',0)  
#Sobel Edge Detection  
sobelxy = cv2.Sobel(src=img, ddepth=cv2.CV\_64F, dx=1, dy=1, ksize=5)  
################################  
edgesx = cv2.Sobel(img, -1, dx=1, dy=0, ksize=1)  
edgesy = cv2.Sobel(img, -1, dx=0, dy=1, ksize=1)  
#Display Sobel Edge Detection Image:  
cv2.imshow('Sobel X Y using Sobel() function', sobelxy)  
#############################################  
cv2.imshow('Sobel X using Sobel() function', edgesx)  
cv2.imshow('Sobel Y using Sobel() function', edgesy)  
cv2.waitKey(0)



TASK5

**Task 5 Laplacian:**

To reduce the noise effect, image is first smoothed with a Gaussian filter and then we find the zero crossings using Laplacian. This two-step process is called the Laplacian of Gaussian (LoG) operation.

But this can also be performed in one step. Instead of first smoothing an image with a Gaussian kernel and then taking its Laplace, we can obtain the Laplacian of the Gaussian kernel and then convolve it with the image. This is shown below where f is the image and g is the Gaussian kernel.



Now, let’s see how to obtain LoG kernel. Mathematically, LoG can be written as

A picture containing text, clock, watch, gauge

Description automatically generated

The LoG kernel weights can be sampled from the above equation for a given standard deviation, just as we did in Gaussian Blurring. Just convolve the kernel with the image to obtain the desired result, as easy as that. Select the size of the Gaussian kernel carefully. If LoG is used with small Gaussian kernel, the result can be noisy. If you use a large Gaussian kernel, you may get poor edge localization.

import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
# loading image  
img0 = cv2.imread('img.png',0)  
  
# remove noise  
img = cv2.GaussianBlur(img0,(3,3),0)  
# convolute with proper kernels  
cv2.imshow('LAPLACIAN FUNCTION',img)  
cv2.waitKey(0)  
laplacian = cv2.Laplacian(img,cv2.CV\_64F)  
  
cv2.imshow('laplacian edge detection', laplacian)  
cv2.waitKey(0)



TASK6

**Task 6 Canny:**

The Canny edge detector is arguably the most well-known and the most used edge detector in all of computer vision and image processing. While the Canny edge detector is not exactly “trivial” to understand, we’ll break down the steps into bite-sized pieces so we can understand what is going on under the hood.Fortunately for us, since the Canny edge detector is so widely used in almost all computer vision applications, OpenCV has already implemented it for us in the cv2.Canny function.

import cv2  
  
img = cv2.imread('road.jpg', 0)  
  
image\_orignal = cv2.imread('road.jpg', cv2.IMREAD\_COLOR)  
image\_gray = cv2.cvtColor(image\_orignal, cv2.COLOR\_BGR2GRAY)  
filtered\_image = cv2.Canny(image\_gray, threshold1=20, threshold2=200)  
  
cv2.imshow('Orignal', image\_orignal)  
cv2.waitKey(0)  
cv2.imshow('Gray', image\_gray)  
cv2.waitKey(0)  
cv2.imshow('Filtered', filtered\_image)  
cv2.waitKey(0)



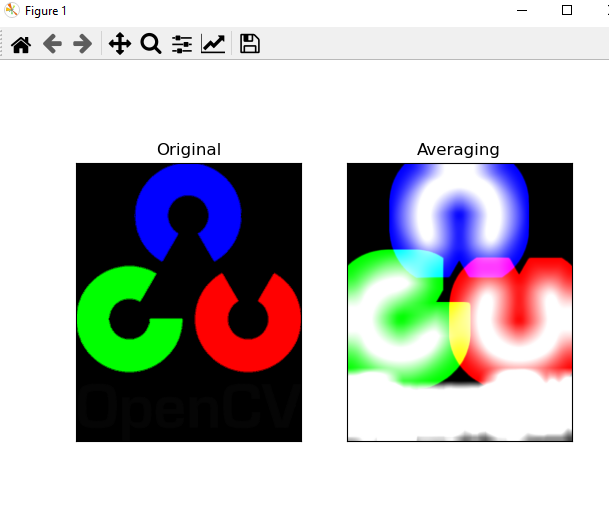
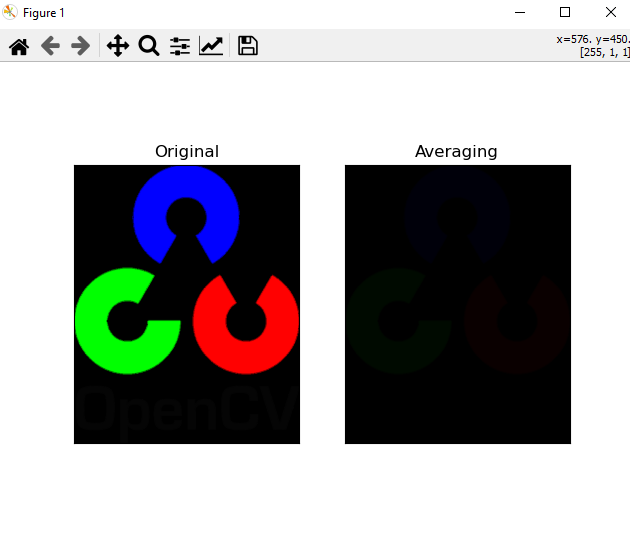
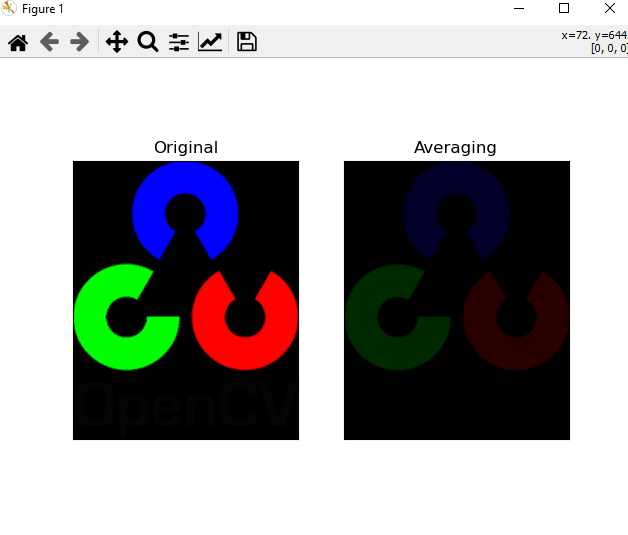
Home Task

2D convolution using 4 different kernels

#4 different Kernels  
import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
img = cv2.imread('img.png')  
kernel = np.ones((2,2),np.float32)/25  
dst = cv2.filter2D(img,-1,kernel)  
  
plt.subplot(121),plt.imshow(img),plt.title('Original')  
plt.xticks([]), plt.yticks([])  
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')  
plt.xticks([]), plt.yticks([])  
plt.show()

#4 different Kernels  
import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
img = cv2.imread('img.png')  
kernel = np.ones((1,1),np.float32)/25  
dst = cv2.filter2D(img,-1,kernel)  
  
plt.subplot(121),plt.imshow(img),plt.title('Original')  
plt.xticks([]), plt.yticks([])  
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')  
plt.xticks([]), plt.yticks([])  
plt.show()

#4 different Kernels  
import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
img = cv2.imread('img.png')  
kernel = np.ones((88,89),np.float32)/25  
dst = cv2.filter2D(img,-1,kernel)  
  
plt.subplot(121),plt.imshow(img),plt.title('Original')  
plt.xticks([]), plt.yticks([])  
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')  
plt.xticks([]), plt.yticks([])  
plt.show()



**Conclusion:**

In this lab we learned about Digital Image Processing and some of the functions that we can perform on an image file such edge detection and blurring an image file. This was a very interesting lab, and we learned a lot in this lab.