ZEWAIL CITY OF SCIENCE AND TECHNOLOGY

MINI-PROJECT 1 - CIE552

Youssef Mahmoud Mohamed 201901093
Mohamed Helmy ElMalkh 201900859

March 9, 2023

Instructions

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

Questions

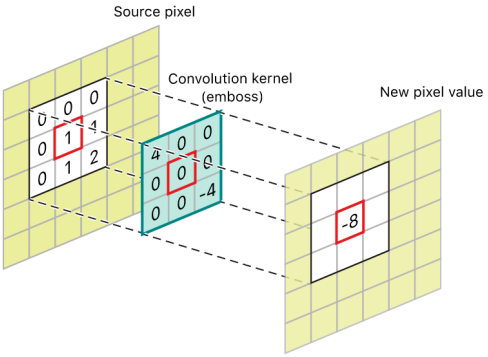
Q1: Explicitly describe image convolution: the input, the transformation, and the output. Why is it useful for computer vision?

A1:

- Input: image and kernel of the filter.
- Transformation: padding the image to get the same size at the end or the image will shrink.
- Output: the filtered image .

$$g(i,j) = \sum_{k}^{K} \sum_{l}^{L} I(i+k, j+l) * h(k,l)$$

- $I(x,y)$ is the image intensity at the pixel of corrdinates (x,y)
- $h(k,l)$ is the filter used which is called *kernel*
- K is the kernel length
- L is the kernel width
- i,j is the *anchor*, the pixel that we are calculating the correlation for



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: The main difference between convolution and correlation is the flip of the filter.So, in case of symmetric filters,there is no difference between them.

symmetric kernel , the same results

```
from scipy import ndimage
c = np.array([[2, 0, 1],
              [1, 0, 0],
              [0, 0, 0]])
k = np.array([[0, 1, 0],
              [0, 1, 0],
              [0, 1, 0],
              [0, 1, 0],
              [0, 1, 0]])
ndimage.convolve(c, k, mode='nearest')
```

```
array([[7, 0, 3],
       [5, 0, 2],
       [3, 0, 1]])
```

```
[134] ndimage.correlate(c,k,mode='nearest')
```

```
array([[7, 0, 3],
       [5, 0, 2],
       [3, 0, 1]])
```

non symmetric kernel by changing one number, different results

```
c = np.array([[2, 0, 1],
              [1, 0, 0],
              [0, 0, 0]])
k = np.array([[0, 1, 0],
              [0, 1, 0],
              [3, 1, 0],
              [0, 1, 0],
              [0, 1, 0]])
ndimage.convolve(c, k, mode='nearest')
```

```
array([[7, 3, 6],
       [5, 0, 2],
       [3, 0, 1]])
```

```
136] ndimage.correlate(c,k,mode='nearest')
```

```
array([[13,  6,  3],
       [ 8,  3,  2],
       [ 3,  0,  1]])
```

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:

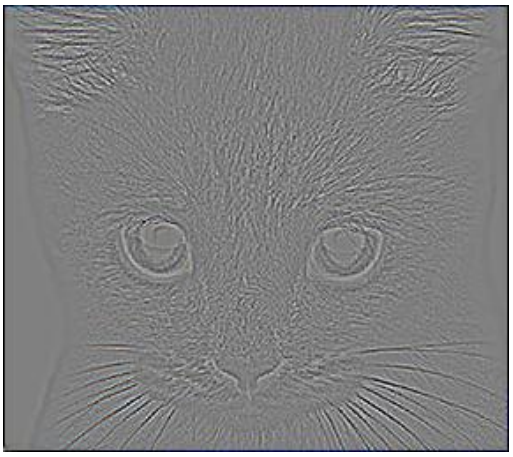
- High pass filter : neglect the low frequencies in the image and preserve the high frequencies which means the sudden changes like edges.
- Low pass filter: the opposite of high pass, it preserve the low frequencies and neglect the sudden transitions.So, it blur the image in most of the cases

Low pass filter

```
blur_filter
✓ 0.1s
array([[0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111]], dtype=float32)
```

High pass filter

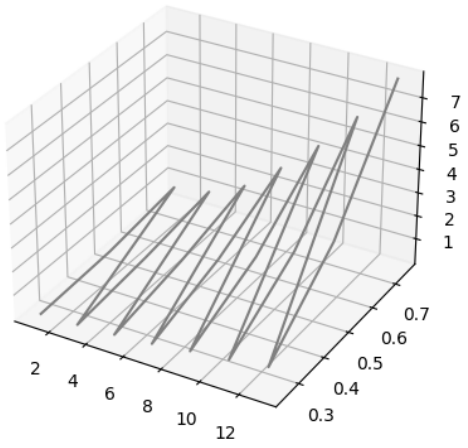
```
laplacian_image
✓ 0.0s
Output exceeds the size limit. Open the full output data in a text editor
array([[[0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.29647861],
        [0.        , 0.12393573, 0.386052  ],
        ...,
        [0.        , 0.        , 0.03825521],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ]],

       [[0.        , 0.        , 0.        ],
        [0.4797574 , 0.50020995, 0.51454234],
        [0.50246085, 0.53220905, 0.50570824],
        ...,
        [0.50083352, 0.49798542, 0.50494413],
        [0.48592097, 0.48383324, 0.46716426],
        [0.        , 0.        , 0.        ]],

       [[0.        , 0.        , 0.        ],
        [0.50114428, 0.50444573, 0.4922879 ],
        [0.5411984 , 0.53955593, 0.52112648],
```



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: Increasing the kernel size increases the time and scaling the image also increases the time . Yes, this is agrees with our expectations to convolution which is directly proportion to N*M which the kernel size and image dimension.

The time on the z axis - scale and kernel size on the x-y axes with correlation2d function



The time on the z axis - scale on the x-y axes using my filter