Homework 2 Questions

Instructions

- 4 questions.
- Write code where appropriate.
- Feel free to include images or equations.
- **Please use only the space provided and keep the page breaks.** Please do not make new pages, nor remove pages. The document is a template to help grading.
- If you really need extra space, please use new pages at the end of the document and refer us to it in your answers.

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 filtering process.
To perform an image convolution you need to specify a filter (sometimes also known as kernel) f and the image I to perform the convolution on. The filter has a squared shape of odd size (e.q. 3×3) and has to be normalized (so the sum has to be between 0 and 1). Having both specified, the convolution can be expressed by the formula:

$$h[m,n] = \sum_{k,l} f[k,l] * I[m-k, n-l]$$

One special difference to correlation (which is an other form of filtering) is the minus in the last part of the formula. This indicates a convertion of x and y axes and could be resolved by flipping f by x-Axis and by y-Axis.
To apply the convolution on I, run the formula on every pixel of I specified by its coordinates m, n. This multiplies cell-wise every filter element with the neighbor of the provided image pixel. The sum of this resulting matrix is the output of the convolution and so the value of $h[m,n]$.
Usually your resulting image h has a smaller size than I, because of the filter size and the lack of neighbors on the outer cells of I (but this can be resolved by some padding techniques). After applying the convolution on every pixel of I, the resulting image h has different effect, depending on f.
Some example effect could be: blurring, sharpening or edge detection.

The image convolution is a useful method in computer vision to extract information from images, detect patterns or enhance images.

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

A2: The difference between convolution and correlation is the way of multiplying the filter with the pixel neighbors of the image.
Correlation just multiplies both matrices element-wise and convolution convertes the x- and y-Axis of the filter and multiplies both matrices afterwards.
This can also be seen in the different formulas:
$h_{corr}[m,n] = \sum_{k,l} f[k,l] * I[m+k, n+l]$ is the formula for the correlation (note the $+$ in the last part of the formula)
$h_{conv}[m,n] = \sum_{k,l} f[k,l] * I[m-k, n-l]$ is the formula for the convolution (note the $-$ in the last part of the formula). This indicates the convertion of the x- and y-Axis.
Both methods provide the same solution/effect if the filter is symmetric.
Using a non symmetric filter results in different behaviour between the methods. For example the filter:

$$\frac{1}{12}\begin{bmatrix} 2 & 2 & 1 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

has different results in h_{corr} and h_{conv}.

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 filters out ”high-frequency” components from the image. The image becomes more smooth and the resulting image appears to blur. This is done to reduce the noise that appears on every image. One example filter (and for sure the easiest one) is the filter calculated of all neighbor pixel:

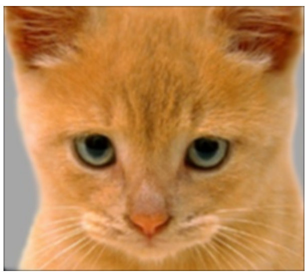
$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Low pass filters are created by assigning the neighbor pixels some positive values so that the resulting pixel contains the information of a wider range of pixels.
On the other side there is the high pass filter. It works the other way around and filters out the ”low-frequencies” of an image. It can be seen as the opposite of the low pass filter, because it focus on the detailed pixels and creates a more sharp image, compared to smoothing the image.
We can use high pass filtering to show fine details of a image or to just increase the sharpness. The following filter is one example of an high pass filter:

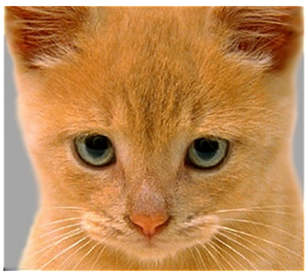
$$\frac{1}{4}\begin{bmatrix}0 & -1 & 0\\ -1 & 8 & -1\\ 0 & -1 & 0\end{bmatrix}$$

As you can see it focus on the central pixel. It has much more impact on the resulting value compared to the surrounding pixels. By having a value greater than 1 it leads to an sharpening. To sum up to 1, the other components of the filter have negative values assigned.

To visualize the difference of both filter techniques I add the resulting images of the introduced filters and the original image for reference.


(a) low pass filter

(b) original


(c) high pass filter

You can see that the left image appears more blurry compared to the original. If you look at the right image and focus on the hairs of the cat, you can see that they appear more sharp compared to the original one.

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 $cv2.filter2D()$ to produce a matrix of values. Use the $cv2.resize()$ function to vary the size of an image. Use an appropriate 3D charting function to plot your matrix of results, such as $plot\_surface()$ or $contour3D$.

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

See RISDance.jpg in the attached file.

A4: (My code is in hw2_testcase.py)
Using filter sizes from 3×3 to 15×15 and also changing the resolution of the image gives us the resulting plot.

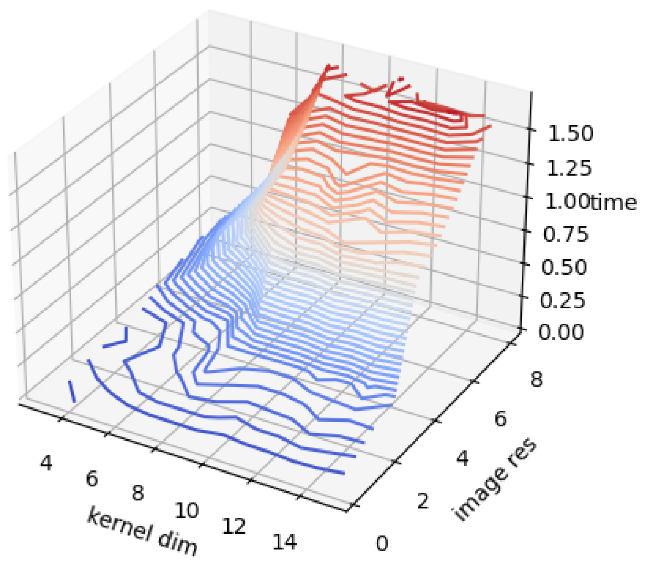


Figure 2: low pass filter

In general there is a time increase with an increase of the image resolution. One interesting thing is, that the computation time for kernels with size ≤ 6 is still low, independent of the image resolution. After a size of 6 there is a increase by kernel size and finally at around 8 the computation time doesn't change when further increasing the kernel size.

It matched my expectations that the time increases with the resolution.