Assignment 2: Recognition using histograms, convolution, and image filtering

Machine perception

2022/2023

General instructions: The assignment is composed of the compulsory tasks and optional tasks (denoted by ★). To approach the assignment defense, you are required to complete at least the compulsory parts. Successfully defending the compulsory part will give you a maximum 75 points out of 100. At the defense, the obligatory tasks **must** be implemented correctly and completely. If your implementation is incomplete or has major errors, you will not be able to successfully defend the assignment. You can choose arbitrarily among the optional tasks to reach the 100 points. The number of points for an optional task is given in the task description. It is possible to obtain more than 100 points on individual assignment. However, the maximum overall points obtained from the 6 assignments is 600 points.

Required formating and submission: Create a folder `assignment2` that you will use during this assignment. Unpack the content of the `assignment2.zip` that you can download from the course webpage to the folder. Save the solutions for the assignments as Python scripts to `assignment2` folder. In order to complete the assignment you have to present your solutions to the teaching assistant. Some assignments contain questions that require sketching, writing or manual calculation. Write these answers down and bring them to the assignment defense as well. The code must be submitted on the e-classroom **before** the defense.

COMMON ERRORS AND DEBUG IDEAS

- check your x and y coordinates
- check your data type: float, uint8
- check your data range: [0,255], [0,1]
- perform simple checks (synthetic data examples)

Introduction

This assignment contains three exercises. In the first one, you will learn about convolution, which is a basic operation in image processing. The second will deal with image filtering and removing noise. In the last one, you will familiarize yourself with several methods of histogram comparison on the domain of image retrieval.

Exercise 1: Convolution

A basic operation that is used in image processing is called convolution. For easier understanding, you will first implement the 1-D version of the operation. Convolution of a kernel $g(x)$ with a signal $I(x)$ is defined by the following equation:

$$I_g(x) = g(x) \star I(x) = \int_{-\infty}^{\infty} g(u)I(x-u)du, \tag{1}$$

or in the case of discrete signals (i.e. images):

$$I_g(i) = g(i) \star I(i) = \sum_{j=-\infty}^{\infty} g(j)I(i-j). \tag{2}$$

A nice visualization of convolution can be seen on Wikipedia[1]. In practice, the kernel is of finite size, thus the sum only runs from the leftmost element to the rightmost element of the kernel. Intuitively, the kernel is *flipped* and *overlaid* on the signal and the overlapping region is multiplied element-wise with the signal, then summed. This sum is the result at position i.

(a) Compute the convolution between the signal and kernel below $(k \star f)$ by hand.

$f =$

0	1	1	1	0	0.7	0.5	0.2	0	0	1	0

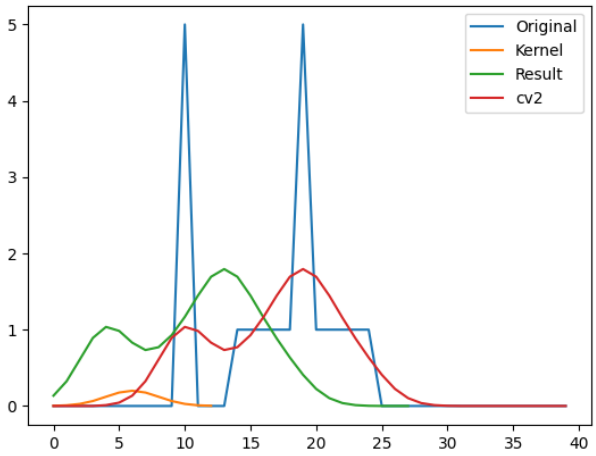
$k =$

0.5	1	0.3

(b) Implement the function `simple_convolution` that uses a 1-D signal I and a kernel k of size $2N + 1$. The function should return the convolution between the two. To simplify, you only need to calculate the convolution on signal elements from $i = N$ to $i = |I| - N - 1$. The first and last N elements of the signal will not be used (this is different in practice, where signal edges *must* be accounted for). Test your implementation by loading the signal (file `signal.txt`) and the kernel (file `kernal.txt`) using the function `read_data` from `a2_utils.py` and performing the operation. Display the signal, the kernel and the result on the same figure. You can compare your result with the result of function `cv2.filter2D`. Note that the shape should be generally identical, while the values at the edges of the results and the results' offset might be different since you will not be addressing the issue of the border pixels.

Question: Can you recognize the shape of the kernel? What is the sum of the elements in the kernel? How does the kernel affect the signal?

[1] http://en.wikipedia.org/wiki/Convolution

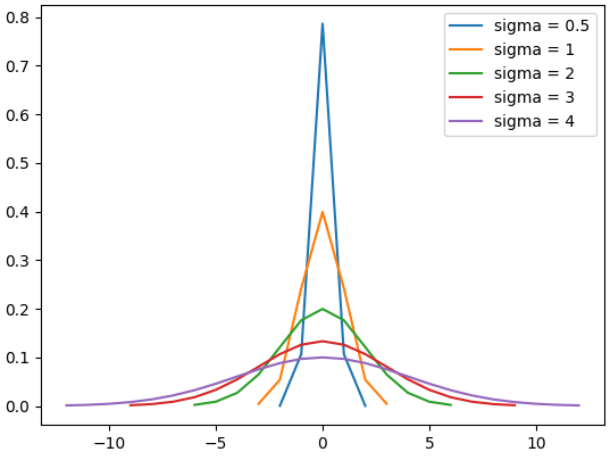


(c) ★ (5 points) Improve the function `simple_convolution` by also addressing the edges of the signal. To do this, implement one of the common methods for padding the signal, so the operation can be performed on all the signal elements. Take care that the input and the output signals are the same length.

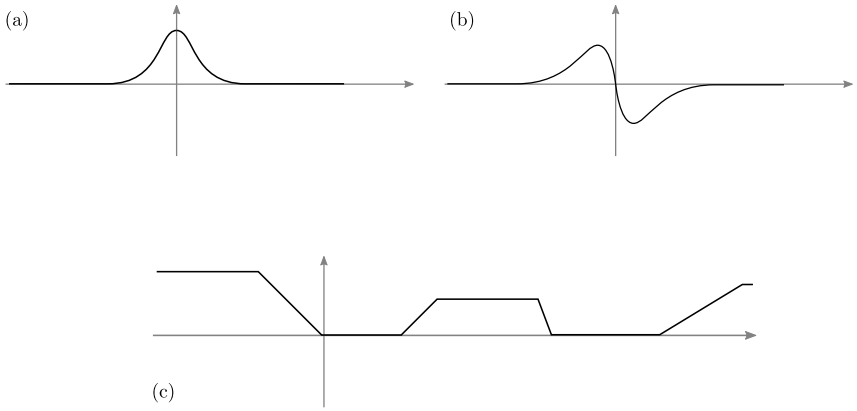
(d) Write a function `gauss(sigma)` that calculates a Gaussian kernel. Use the definition:

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{x^2}{2\sigma^2}). \tag{3}$$

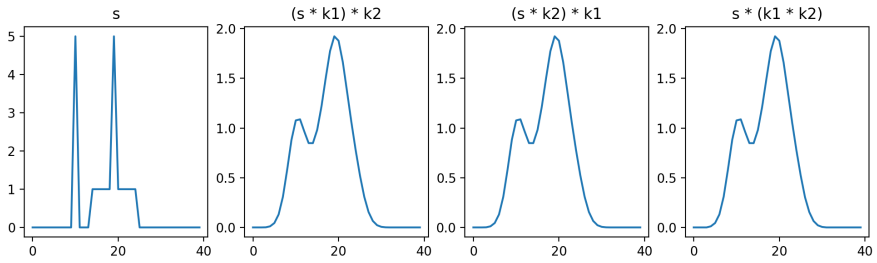
The input to the function should be parameter σ, which defines the shape of the kernel. Because the values beyond 3σ are very small, limit the kernel size to $2\lceil 3\sigma \rceil + 1$. Don't forget to normalize the kernel. Generate kernels for different values of $\sigma = 0.5, 1, 2, 3, 4$ and display them on the same figure (aligned).



Question: The figure below shows two kernels (a) and (b) as well as signal (c). Sketch (do not focus on exact proportions of your drawing, but rather on the understanding of what you are doing) the resulting convolved signal of the given input signal and each kernel.



(e) The main advantage of convolution in comparison to correlation is the associativity of operations. This allows us to pre-calculate multiple kernels that we want to use on an image. Test this property by loading the signal from `signal.txt` and then performing two consecutive convolutions on it. The first one will be with a Gaussian kernel k_1 with $\sigma = 2$ and the second one will be with kernel $k_2 = [0.1, 0.6, 0.4]$. Then, convolve the signal again, but switch the order of the operations. Finally, create a kernel $k_3 = k_1 * k_2$ and perform the convolution of the original signal with it. Display all the resulting signals and comment on the effect the different order of operations has on the signal. Use the function from c) or `cv2.filter2D()` to take care of the edges when convolving.



Exercise 2: Image filtering

This exercise will teach you how to use convolution on 2-D signals to achieve image filtering. Filtering can be used to smooth or sharpen images, or to remove certain types of noise from the image. You will also experiment with some filters that are not based on convolution.

(a) An important property of the Gaussian kernel is its separability in multiple dimensions. A Gaussian kernel in 2-D space can be written as:

$$\begin{aligned} G(x,y) &= \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\ &= \left[\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma^2}\right)\right]\left[\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{y^2}{2\sigma^2}\right)\right] \\ &= g(x)g(y), \end{aligned} \tag{4}$$

which is a product of two 1D Gaussian kernels, each one in its own dimension. If we again write a continuous convolution,

$$\begin{aligned} G(x,y) * I(x,y) &= \int_u \int_v g(u)g(v)I(x-u, y-v)dudv \\ &= \int_u g(u)(\int_v g(v)I(x-u, y-v)du)dv \\ &= g(x) * [g(y) * I(x,y)], \end{aligned} \tag{5}$$

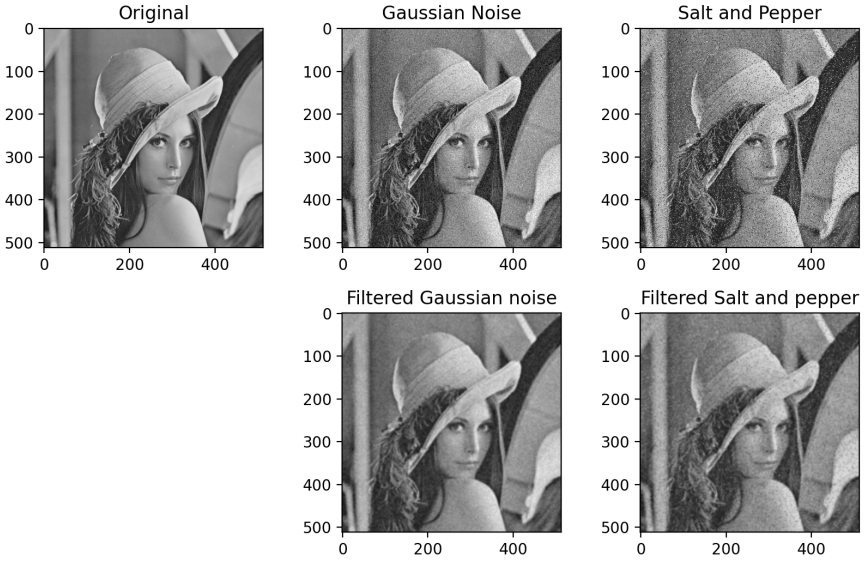
we can see that we get the same result if we filter a 2-D signal using a single 2-D Gaussian kernel or if we do it using two 1D Gaussian kernels that represent the separated components of the former 2-D Gaussian kernel. This technique can be used to translate a slow n-D filtering operation to a fast sequence of 1-D filtering operations.

Write a function `gaussfilter` that generates a Gaussian filter and applies it to a 2-D image. You can use the function `cv2.filter2D()` to perform the convolution using the desired kernel. Generate a 1-D Gaussian kernel and first use it to filter the image along the first dimension, then convolve the result using the same kernel, but transposed.

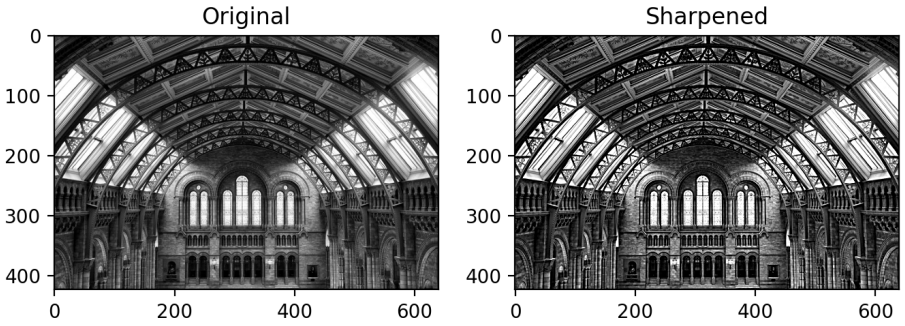
Hint: Numpy arrays have an attribute named `T`, which is used to access the transpose of the array, e.g. `k_transposed = k.T`.

Test the function by loading the image `lena.png` and converting it to grayscale. Then, corrupt the image with Gaussian noise (every pixel value is offset by a random number sampled from the Gaussian distribution) and separately with *salt-and-pepper* noise. You can use the functions `gauss_noise` and `sp_noise` that are included with the instructions (`a2_utils.py`). Use the function `gaussfilter` to try and remove noise from both images.

Question: Which noise is better removed using the Gaussian filter?



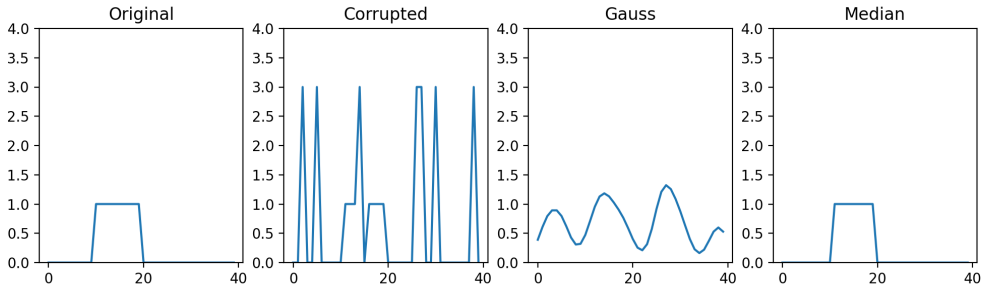
(b) Convolution can also be used for image sharpening. Look at its definition in the lecture slides and implement it. Test it on the image from file `museum.jpg`.



(c) Implement a nonlinear median filter. While a Gaussian filter locally computes a weighted average, the median filter sorts the signal values in the given filter window and uses the median value of the sorted sequence as the result. Implement a simple median filter as a function `simple_median` that takes the input signal I and the filter width w and returns the filtered signal.

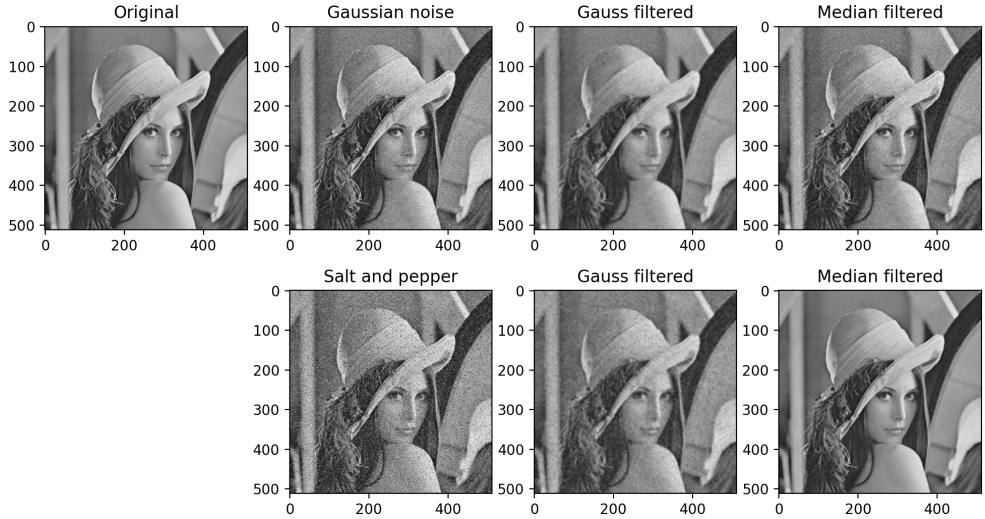
Create a 1-D signal corrupted with salt and pepper noise and filter it using `simple_median`. Display the result using different window sizes. Also try to remove the noise using the Gaussian filter. What does the result look like and why?

Question: Which filter performs better at this specific task? In comparison to Gaussian filter that can be applied multiple times in any order, does the order matter in case of median filter? What is the name of filters like this?



(d) ★ (5 points) Implement a 2-D version of the median filter. Test it on an image that was corrupted by Gaussian noise and on an image that was corrupted by *salt and pepper* noise. Compare the results with the Gaussian filter for multiple noise intensities and filter sizes.

Question: What is the computational complexity of the Gaussian filter operation? How about the median filter? What does it depend on? Describe the computational complexity using the $O(\cdot)$ notation (you can assume $n \log n$ complexity for sorting).



(e) ★ (15 points) Implement the hybrid image merging that was presented at the lectures. To do this, you will have to implement the Laplacian filter. Filter the images (one with the Gaussian and one with the Laplacian filter) and merge them together (regular or weighted average). You can use images `lincoln.jpg` and `obama.jpg`. *Hint:* To get good results, experiment with different kernel sizes for each operation and different weights when merging images.

Exercise 3: Global approach to image description

In the previous assignment, we used histograms created from a single channel images. Now we will extend this to multi-dimensional histograms and use histograms as global image descriptors in order to compare images.

(a) Firstly, you will implement the function `myhist3` that computes a 3-D histogram from a three channel image. The images you will use are **RGB**, but the function should also work on other color spaces. The resulting histogram is stored in a 3-D matrix. The size of the resulting histogram is determined by the parameter `n_bins`. The bin range calculation is exactly the same as in the previous assignment, except now you will get one index for each image channel. Iterate through the image pixels and increment the appropriate histogram cells. You can create an empty 3-D *numpy* array with `H = np.zeros((n_bins,n_bins,n_bins))`. Take care that you normalize the resulting histogram.

(b) In order to perform image comparison using histograms, we need to implement some distance measures. These are defined for two input histograms and return a single scalar value that represents the similarity (or distance) between the two histograms. Implement a function `compare_histograms` that accepts two histograms and a string that identifies the distance measure you wish to calculate. You can start with the L_2 metric.

The L_2 metric (commonly known as Euclidean distance) treats histograms with N bins as points in N-dimensional space. For histograms h_1 in h_2 (that of course must have the same number of bins) the L_2 distance is defined as:

$$L_2 = \Big[\sum\nolimits_{i=1:N} (h_1(i) - h_2(i))^2 \Big]^{\frac{1}{2}}. \tag{6}$$

Also implement the following measures that are more suitable for histogram comparison:

- Chi-square distance $\chi^2 = \frac{1}{2} \sum\limits_{i=1:N} \frac{(h_1(i)-h_2(i))^2}{h1(i)+h2(i)+\varepsilon_0}$, where ε_0 is a very small constant value (e.g. $1e-10$) that is used to avoid division by zero.
- Intersection $I = 1 - \sum_{i=1:N} \min(h_1(i), h_2(i))$
- Hellinger distance $H = \big(\frac{1}{2} \sum\limits_{i=1:N} (h_1(i)^{\frac{1}{2}} - h_2(i)^{\frac{1}{2}})^2\big)^{\frac{1}{2}}$

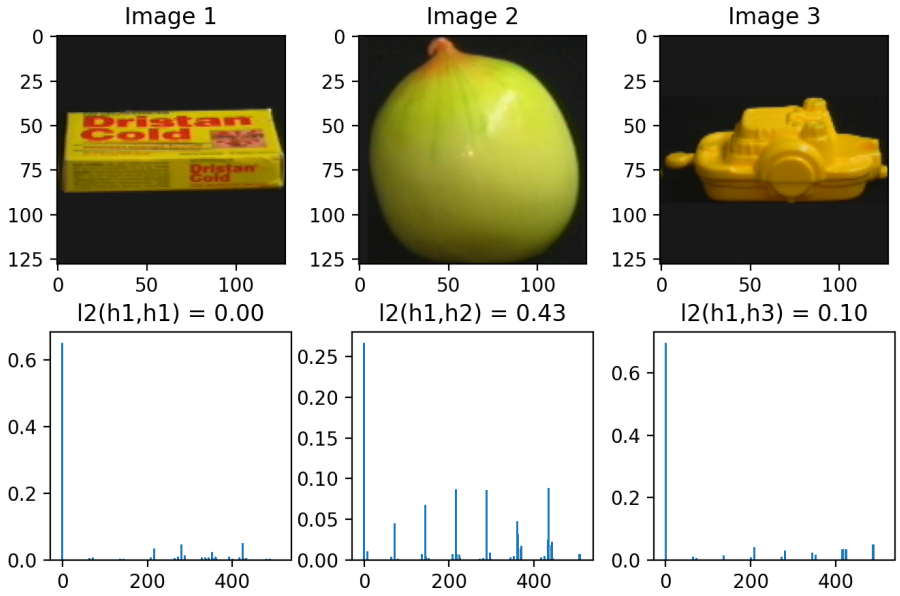
Try to avoid looping over histogram values and instead use vector operations on entire matrices at once.

(c) Test your function with the following images:

- `dataset/object_01_1.png`,
- `dataset/object_02_1.png`,
- `dataset/object_03_1.png`.

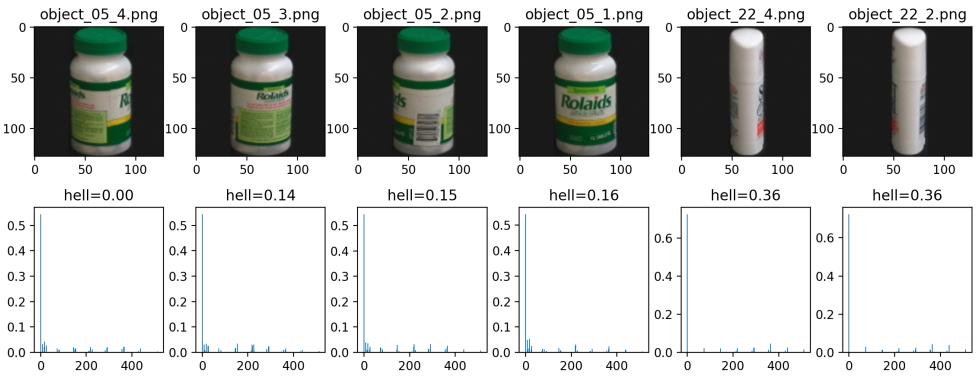
Compute a $8 \times 8 \times 8$-bin 3-D histogram for each image. Reshape each of them into a 1-D array. Using `plt.subplot()`, display all three images in the same window as well as their corresponding histograms. Compute the L_2 distance between histograms of object 1 and 2 as well as L_2 distance between histograms of objects 1 and 3.

Question: Which image (`object_02_1.png` or `object_03_1.png`) is more similar to image `object_01_1.png` considering the L_2 distance? How about the other three distances? We can see that all three histograms contain a strongly expressed component (one bin has a much higher value than the others). Which color does this bin represent?



(d) You will now implement a simple image retrieval system that will use histograms. Write a function that will accept the path to the image directory and the parameter `n_bins` and then calculate **RGB** histograms for all images in the directory as well as transform them to 1-D arrays. Store the histograms in an appropriate data structure.

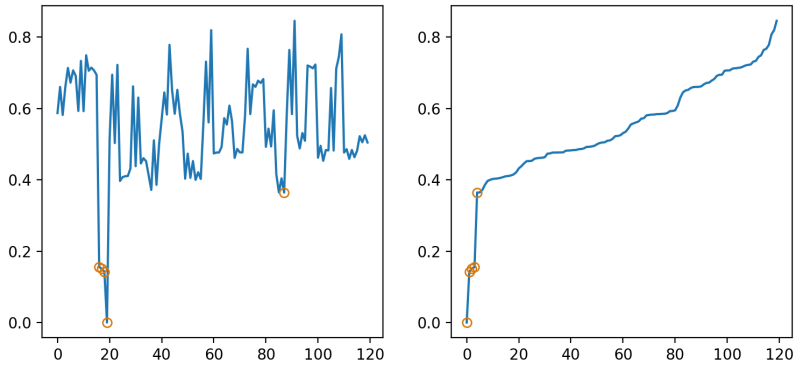
Select some image from the directory `dataset/` and compute the distance between its histogram and all the other histograms you calculated before. Sort the list according to the calculated similarity and display the reference image and the first five most similar images to it. Also display the corresponding histograms. Do this for all four distance measures that you implemented earlier.



Question: Which distance is in your opinion best suited for image retrieval? How does the retrieved sequence change if you use a different number of bins? Is the execution time affected by the number of bins?

(e) You can get a better sense of the differences in the distance values if you plot all of them at the same time. Use the function `plt.plot()` to display image indices

on the x axis and distances to the reference image on the y axis. Display both the unsorted and the sorted image sequence, and mark the most similar values using a circle (see *pyplot* documentation).



(f) ★ (10 points) This simple retrieval system is strongly influenced by dominant colors that carry no discriminative information. Analyze the presence of different colors by summing up all image histograms bin-wise and displaying the resulting histogram. Which bins dominate this histogram?

To address this issue, you will implement a simple frequency-based weighting technique, similar to the ones that are employed in document retrieval systems. Use the combined frequency histogram you calculated to determine the weight for each bin. The weights should be lower for bins that are strongly represented in the frequency histogram and vice versa. One way of computing this weight is to use the exponential function $w_i = e^{-\lambda F(i)}$, where $F(i)$ represents a frequency of the i-th bin and λ is a scaling constant that you have to set. There are other ways of computing weights that you can experiment with. Before calculating histogram similarity, you should multiply each histogram with the weights vector bin-wise (and normalize the result). Finally, you can compare the retrieval process for the weighted and the unweighted histograms. Report your observations. Did the weighting help with retrieving relevant results?