

Learning from Images Image descriptors and applications (12 Points)

WiSe 2024/25

The aim of this exercise is to develop a deeper understanding of distinctive image features / descriptors, their implementation and application areas. Download the skeleton source code from Moodle and implement the necessary functions. Expand and modify the source code as you like. The source code contains many more comments.

Exercise 1: Simple Content-based Image Retrieval (4 Points)

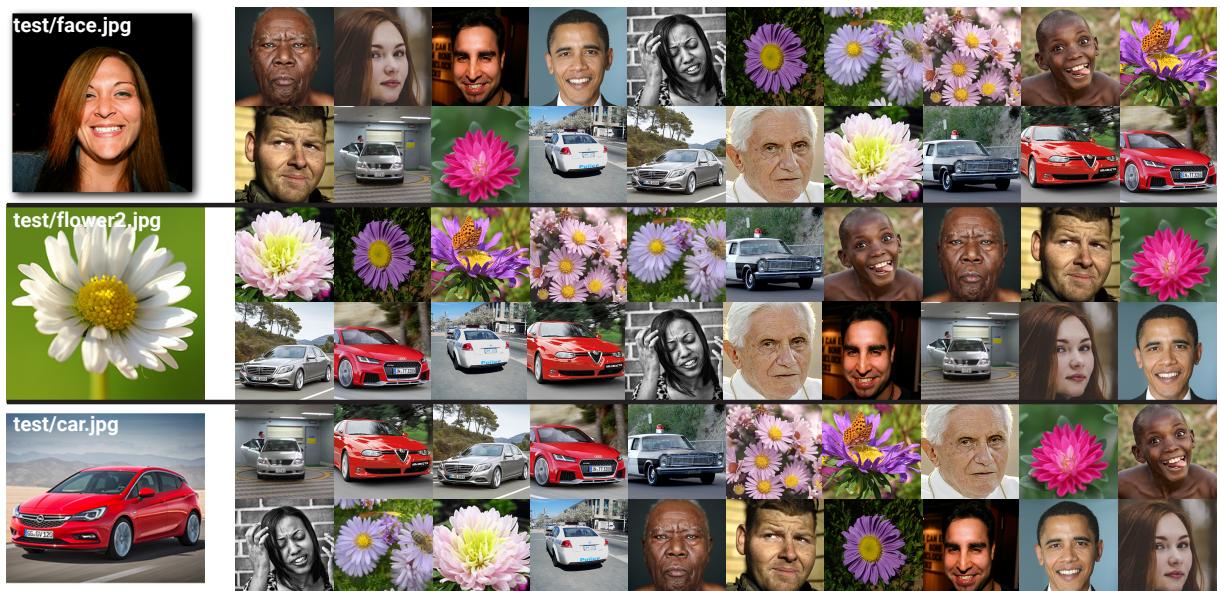


Figure 1: Retrieval results for test images from top left to bottom right.

The goal of a content-based image search is to find similar images related to an input image based only on image features (i.e. no keywords or tags). The search is also often based on the use of local feature vectors, e.g. SIFT. In the search, these vectors must be compared to find similarities. Different distance metrics can be used for this. In this very reduced example you should implement a search that compares the vectors based on the L2-norm (i.e. the Euclidean distance of the vectors). Proceed as follows and use *image_retrieval.py*:

- Load all images of the supplied 'database'. There are a total of 3 different picture classes with about 5-6 pictures.
- Create a set of keypoints that are distributed across the image in a uniform grid. Calculate the SIFT feature vectors at these locations.
- Use the keypoints to calculate the SIFT feature vectors at these locations. This creates exactly one descriptor for each image in the database. These should be cached in an array. Note: We do not use the SIFT keypoint detector here.

- d) Load a query image and create an image descriptor for the same keypoints. Compare the descriptor against the descriptors from the database. Use the L2 standard for this.
- e) Save the result in a *PriorityQueue*.
- f) Output the results of the query in the order of the shortest distance. The result should look like the faces shown in Figure 1.

Exercise 2: Write your own image descriptor (4 Points)

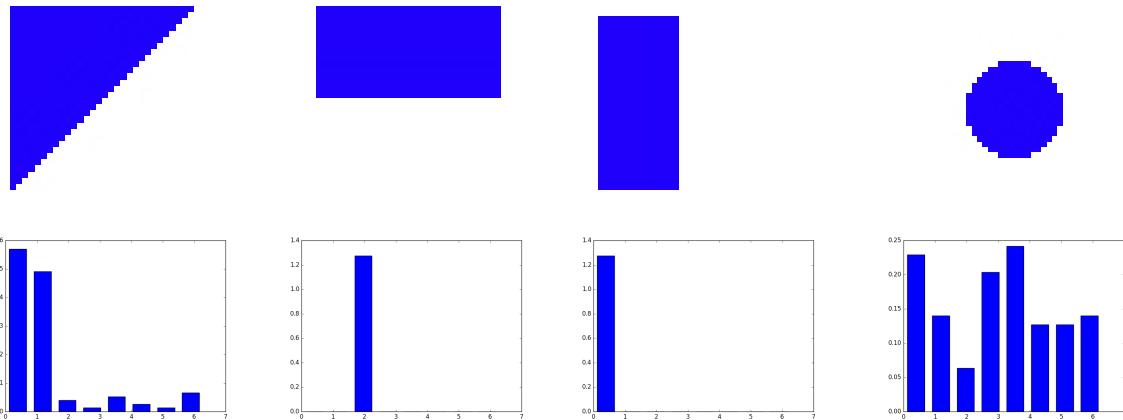


Figure 2: Histogram of oriented gradients test evaluation. Top row shows example images. Bottom row shows the histogram of gradients.

Implement your own local feature vector and test it with the supplied input images (`simple_hog.py`). The descriptor stores a histogram of gradient directions generated by a simple edge detection filter (e.g., sobel filter). In the specific case, the descriptor should only be extracted for a single keypoint (in the center of the image) and a given center size (e.g., 11). **Hint:** Take a look at the functions `cv2.Sobel`, `cv2.phase` and `cv2.magnitude`. The resulting histograms should look like in Figure 2 illustrated. The histogram contains 8 buckets. Think about a meaningful value range.

Exercise 3: Harris Corner Detector (4 Points)

As discussed in the lecture, the matrix M is needed to calculate the Harris-Corner detector

$$M = \sum_{x,y}^{3,3} w(x,y) \begin{bmatrix} G_{xx} & G_x G_y \\ G_y G_x & G_{yy} \end{bmatrix}$$

The eigenvalues of this matrix are different for edges, smooth regions and corners. For smooth regions in the picture both eigenvalues are very small. For edges, one eigenvalue is at least an order of magnitude larger than the other, and for corners both are of the same order of magnitude.

For the efficient computation of the Harris corners we don't need to calculate the Eigenvalues. It is enough to calculate the determinant (\det) and the trace Tr of the matrix:

$$H = \det(M) - k \cdot Tr(M)^2$$

As discussed in the lecture, the following points are necessary for you to implement the harris corner detector:

- a) Calculate the gradients of the image (x- and y-direction).
- b) Calculate the matrix M for each pixel in the image. **Hinweis:** For this you need to calculate G_{xx} , G_{yy} and G_{xy} over the entire image and sum over the values of a window size of 3×3 pixels. This step can be solved by using a convolution 2D filter. Please use a `cv2.filter2D` for this.

- c) Calculate the determinate of M by $\det(M) = G_{xx} \cdot G_{yy} - G_{xy} \cdot G_{xy}$.
- d) Calculate the trace of M by $Tr(M) = G_{xx} + G_{yy}$
- e) k is a parameter between 0.04 ... 0.06.
- f) The final image H contains now the Harris corners that should be filtered by a threshold.



Figure 3: Example result of the Harris corner detector using the parameters described in the given source code.

02_harris.py contains more information about the implementation. To test your algorithm, you can use the OpenCV implementation as specified in SourceCode and compare the results. The sum of the pixel differences between your and the OpenCV implementation should be 0.0 (there may be some floating point inaccuracies). Please try to implement the algorithm as efficient as possible. Your results should then look like illustrated in Figure 3.

Exercise 4: Readings (voluntary)

Please read the publications in the reading area (Moodle).

Abgabe: This is *Exercise 2/4*. The tasks are designed to be solvable in 3 weeks. **Information on due date and a link to upload your solution as a .zip file is given in the Moodle system.** Please submit only one single .zip file with the sources of your solution. **Please insert all necessary images so that each task is directly executable. Otherwise I reserve point deductions..** Submission after due date will be deducted with 5 points for each delayed week.