Computer Vision I
Assignment 2

Prof. Stefan Roth
Krishnakant Singh
Shweta Mahajan

TECHNISCHE UNIVERSITÄT DARMSTADT

22/11/2021

This assignment is due on December 6th, 2021 at 23:59.

Please refer to the previous assignments for general instructions and follow the handin process described there.

Problem 1 - Search and Recognition for Face Image Pyramids (15 points)

Image pyramids are a widely used concept in computer vision. One of their key applications is multi-scale detection and recognition – determining the presence and localising features of interest in the image in a scale-invariant manner. In this problem, we will implement a simple detection pipeline using template matching techniques.

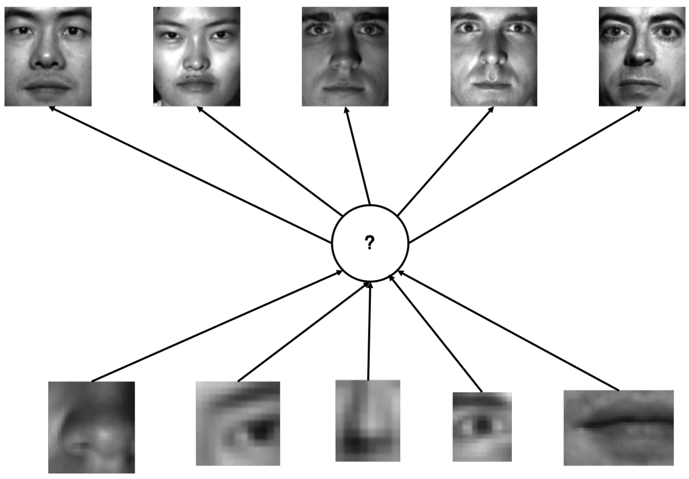


Figure 1: **Task overview.** Match the facial features (bottom) to the corresponding face images (top).

The overall task, illustrated in Fig. 1, is to match facial features we provide (*e.g.* nose, eye) with the corresponding face image. Note that these features are provided at different scales. Therefore, you will first implement a Gaussian pyramid to obtain a multi-scale representation of an image. You will then use a sliding window approach to match the facial features to images at multiple scales.

Tasks:

Load Data. Your first task is to implement the function `load_data`.

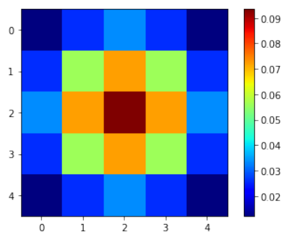
1. Load the data from the folder `data`. We placed the face images in folder `facial_images` and the facial features in folder `facial_features`. The face images and the facial features shoud be loaded as two lists of `numpy` arrays.

(1 point)

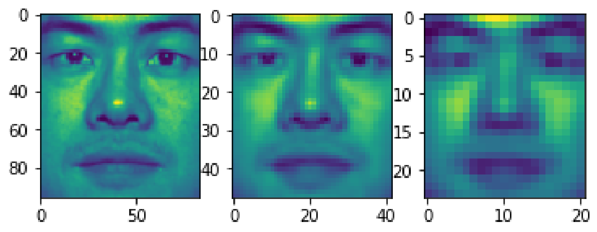
Make a Gaussian Pyramid. In this task, you will create an image pyramid with Gaussian smoothing.

1. First, using function `gaussian_kernel` to create a Gaussian filter kernel (Fig. 2a: size 5×5 and σ=1.4).

(2 points)



(a) 5×5 Gaussian kernel.



(b) Image pyramid with Gaussian smoothing.

Figure 2: Facial image pyramid.

2. Next, implement function `downsample_x2` that takes an image and downsamples it by a factor of 2. Make sure your implementation does not use bilinear or bicubic interpolation.

(2 points)

3. Finally, the image pyramid will be generated by the function `gaussian_pyramid`, which makes use of the downsampling and the Gaussian kernel you have just implemented. As shown in Fig. 2b, we will use a 3-level Gaussian pyramid. However, your implementation should be general enough to support other (reasonable) levels specified by the argument `nlevels`.

(3 points)

Search and Recognition. The next task is to implement a sliding window approach. The sliding window has the size of a facial feature. In addition to sliding through all image locations, we will also do so at different scales of the face image.

1. In the lecture, we have looked at two distance functions between feature vectors: the dot product and the sum of squared differences (SSD). Implement the distance of your choice in `template_distance`: it calculates and returns the distance between two vectors.

(2 points)

2. Implement the function `sliding_window`. The function initializes a window with the size of the facial feature and slides it across a face image with a stride of 1. Use `template_distance` to measure the distances between the features in the sliding window and the facial feature. Return the smallest distance among all locations.

(2 points)

3. Combine the two functions implemented above in `find_matching_with_scale`. First, construct the face image pyramid with function `gaussian_pyramid` and then use function `sliding_window` to find the minimum distance of the facial feature at a given image scale. Finally, for each feature, return the matched face image and the corresponding (minimum) distance as a list of three items: the feature itself, the corresponding image and the minimum distance.

(2 points)

4. Experiment with the two distance functions, the dot product and SSD, in the context of our detection algorithm. Specify in your code which method you think is a more reasonable choice and justify why with a commment in your code.

(1 point)

Problem 2 - PCA for Face Images (15 points)

You will be working with a training database of human face images and build a low-dimensional model of the face appearance using Principal Component Analysis (PCA). We provide function definitions you have to implement in `problem2.py` and adhere to the notation used in class in the task description below.

Tasks:

- Implement function `loadfaces` that loads N images of human faces in a given path into a `numpy` array of dimension $N \times M$ where $M = \text{height} \times \text{width}$, *i.e.* the number of pixels in the image. For visualisation later on, it will be useful to recover the original shape of the image which is lost due to such vectorisation. Return a tuple (`height`, `width`) as the second value to preserve this information.

  (1 point)

- Before we move on to implement PCA, please specify in your code which method is a more reasonable choice for your implementation, SVD or eigendecomposition, and justify why with a comment in your code.

  (2 points)

- Implement the PCA of the face images in `compute_pca` using the loaded data array. Function `compute_pca` returns all principal component vectors u_i and the corresponding variance λ_i.

  (3 points)

What do the principal components represent? To understand this better we can project individual face images on a few principal components and visualise the result. Concretely, we can represent an image as $x^n - \bar{x} \approx \sum_i^D a_i u_i$, where D is the number of components we select.

- Implement function `basis` that selects the *fewest* possible principal components corresponding to the percentile fraction $\eta \in (0, 1]$ of the total variance. That is, $D_c^* = \{\text{min } D | \sum_i^D \lambda_i \geq \eta \sum_i^M \lambda_i.\}$

  (2 points)

- Implement function `project` that projects a provided face image onto the bases we have computed in the previous step.

  (2 points)

You can now select a face image of your choice and visualise the part of the face image that falls into the above subspace by using the original image and its projected counterpart. Experiment with different percentiles, *e.g.* $\eta = 0.5$, 0.7, 0.9, and analyse the result.

- Using the class `NumberOfComponents` provided in your code, select the observations made by varying the number of basis vectors you use in your projection.

  (1 point)

We can now explore some useful applications of the basis representation we have obtained.

- *Image Search.* We can use the projection coefficients a_i as image descriptors and compare images by computing the distance between their compact vector representation in terms of a few principal components (*e.g.* corresponding to a sufficiently large percentile η). Implement `search_face` that first decomposes the provided face image into a few a_i's and then searches for the top-n most similar images based on a vector consisting of these coefficients. You should use L2 distance as the similarity metric. *Sanity check:* A function call with top-1 should always return the image itself.

  (2 points)

- *Face Interpolation.* Implement function `interpolate` that takes two face images and produces a given number of intermediate images. First, project each image on the provided basis vectors to obtain vectors with a_i's. Then, interpolate between the two representations at equal steps and project it back onto the principal components to obtain the corresponding image. *Hint:* You may find `np.linspace` useful for this task.

  (2 points)