project 2:

eigenfaces

solution(s) due:

June 29, 2017 at **12:00** via email to **bauckhag@bit.uni-bonn.de**

problem specification:

task 2.1: computing eigenfaces

- download and unzip the file `cbcl-faces.zip`; you will find a collection of 2429 tiny face images (of size 19×19 pixels).
- randomly select 90% of these images and collect them into a set X_{train}; collect the remaining 10% into a set X_{test}
- read the *training images* into a data matrix $\boldsymbol{X}_{\text{train}}$ and center the data, i.e. subtract the mean $\boldsymbol{m}_{\text{train}}$
- compute the covariance matrix $\boldsymbol{C}$ as well as its eigenvectors $\boldsymbol{v}_i$ and eigenvalues λ_i
- plot the *spectrum* of $\boldsymbol{C}$, i.e. the set of eigenvalues in descending order; what do you see?
- determine the smallest eigenvalue λ_k such that $\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{d} \lambda_j} \geq 0.9$
- visualize the first k eigenvectors $\boldsymbol{v}_i \in \mathbb{R}^{361}$ as 19×19 images; what do you see?
- now, read the *test images* into a data matrix $\boldsymbol{X}_{\text{test}}$ and center the data w.r.t. the *training mean*, i.e. subtract $\boldsymbol{m}_{\text{train}}$
- randomly select 10 test images, compute their Euclidean distances to all training images, sort (in descending order) and plot the distances
- now, project the training and test data into the subspace spanned by the first k eigenvectors of $\boldsymbol{C}$

- consider the same 10 test images as above; in the lower dimensional space, compute their Euclidean distances to all the training images, sort the set of distances in descending order and plot them; compare your plots
- again, for the same 10 test images as above, use the Euclidean distance to determine their nearest neighbor among the training images; first, in the original space and then in the lower dimensional space; are the nearest neighbors you find identical in both cases?

general hints and remarks

- Send all your solutions (code, resulting images, slides) in a ZIP archive to bauckhag@bit.uni-bonn.de

  note: if you are implementing in C/C++, you may use whatever libraries appear useful. It is, however, suggested you use python and the Image and scipy modules for all your practical work. If you insist on using a language other than python, you have to figure out elementary image processing functions/toolboxes in these languages by yourself. **Implementations in MATLAB will be rejected.**
- Remember that you have to successfully complete all three practical projects (and the tasks therein) to be eligible to the written exam at the end of the semester. Your grades (and credits) for this course will be decided based on the exam only, but –once again– you have to succeed in the projects to get there.
- Not handing in a solution implies failing the course.
- Your project work needs to be *satisfactory* to count as a success. Your code and results will be checked and your presentation needs to be convincing.
- If your solutions meets the above requirements and you can demonstrate that they work in practice, it is a *satisfactory* solution.
- A *good* to *very good* solution requires additional efforts especially w.r.t. to elegance and readability of your code. If your code is neither

commented nor well structured, your solution is not good! A very good solution requires additional efforts towards the quality of your project presentation in the colloquium. Your presentation should be well timed, consistent, and convincing. Striving for very good solutions should always be your goal!