Assignment 2 – CV1 WS18/19 12/11/2018

Prof. Stefan Roth
Junhwa Hur
Anne Wannenwetsch

This assignment is due on November 26th, 2018 at 13:00.

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

Problem 1 - Image Pyramids and Image Sharpening (15 points)

Image pyramids are widely used in computer vision. In this problem you will create a small sharpening application by using both Gaussian and Laplacian image pyramids. You will first write a few helper functions. Please note that you are not allowed to use the Julia built-in functions `restrict` and `gaussian2d` for this problem. **Implement the following functions by yourself**:

1. Function `makegaussianfilter` with two arguments that creates a Gaussian filter with specified size (number of rows and columns) and specified kernel width (standard deviation σ). You can find the formula in the slides. Make sure that the maximum is in the middle of the filter mask (for even and odd filter sizes), and also ensure that the filter is properly normalized (i.e. the filter coefficients should add to 1).

2. Function `makebinomialfilter` that creates a binomial filter with the specified filter size (number of rows and columns). To construct a binomial filter you initially compute the set of binomial coefficients, e.g. as in Pascal's triangle, and normalize afterwards. For instance, the weights w_k of a 1D binomial filter with $N+1$ weights can be constructed by formula

$$w_k = \widehat{w}_k / \sum_{l=0}^{N} \widehat{w}_l, \quad \widehat{w}_k = \binom{N}{k}, \qquad k = 0, 1, \ldots, N. \tag{1}$$

3. Function `downsample2` that takes an image and downsamples it by a factor of 2, i.e. resizes both dimensions to half the size. To that end simply discard every other row and column. Note: Do not perform any Gaussian smoothing here; this will be done later.

4. Function `upsample2` that takes an low resolution image as well as a filter size and upsamples the image by a factor of 2, i.e. resizes each dimension to double the size. In particular, insert one zero row between every low-resolution row and then insert one zero column between every "old" column. Filter the result with a binomial kernel of the given size and finally apply a scale factor of 4. Note: You should make use of your function `makebinomialfilter` and use symmetric boundary conditions for filtering (i.e. the image outside the valid region is filled by mirror-reflecting the image across the borders).

The outline for the sharpening application is given in the function `problem1` and should be completed with the necessary calls. Implement the following tasks by using the functions you wrote above:

5. Function `makegaussianpyramid` that builds a multi-level Gaussian pyramid of a gray-value input image (a 6-level Gaussian pyramid in this assignment, using the variable `nlevel`). To create a subsequent level of the Gaussian pyramid, filter the image with a Gaussian kernel ($\sigma = 1.4$ and size 5×5) and then downsample by a factor of 2. For filtering steps, apply symmetric boundary conditions and make sure that filtered images have the same size as corresponding inputs. Note: The result of this function should be an array of images with decreasing sizes.

Figure 1: Gaussian pyramid

6. Function `displaypyramid` that shows image pyramids in a single figure as depicted in Fig. 1 . Since we want to use the same function to display both Gaussian and Laplacian image pyramids, you should normalize the images of the pyramid levels such that they have values in $[0, 1]$.

7. Function `makelaplacianpyramid` that, based on the Gaussian pyramid, creates the corresponding Laplacian pyramid. For building the Laplacian pyramid, you should also use the upsampling function from above. What is the difference between the top (coarsest) level of Gaussian and Laplacian pyramids? Please answer that question briefly in `answers_problem1.txt`.

8. In our skeleton we load and sharpen the image `a2p1.png`. To make the sharpening work you have to implement the function `amplifyhighfreq2` that amplifies the high-frequency components contained in the finest two levels of the Laplacian pyramid by scaling them up by a factor. Afterwards, implement `reconstructlaplacianpyramid` that reassembles the pyramid back to obtain a sharpened full-resolution image. Try various amplification factors for both levels and choose those that lead to good or interesting results. You may find that the image noise gets amplified if you scale up the coefficients of the finest sub-band too much; try to avoid that. Briefly explain your findings in `answers_problem1.txt`. Finally, display the original image, its reconstruction and their difference in one Figure next to each other.

   Note: Julia arrays (and other non-primitive types) are assigned by reference. For example, look at the following expression `A = zeros(1); B=A; B[1] = 1; println(A[1]);`

Submission: Please include `problem1.jl` and a file `answersproblem1.txt` in your submission.

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). The outline is given in `problem2.jl` and should be completed with the necessary functions. Your tasks are:

1. All faces are located in the `yale_faces` directory. Implement the function `loadfaces` that loads all face images into a big data matrix and returns this matrix, the dimension of a single face and the number of faces.
2. Implement the PCA of all face images in `computepca`. Each face image has over 8000 pixels and there are many fewer training images than this. Consequently, you want to use SVD as we discussed in class. Make sure that the principal components are properly sorted in decreasing order. Also compute the mean face and a vector that contains the cumulative variance of the principal components. The function `svd` included in `LinearAlgebra` might be helpful for this task.
3. Show a plot of the cumulative variance of the principal components in `plotcumvar`. Briefly explain in `answersproblem2.txt` how you compute the variance of a single principal component and why the variance can be obtained in that way.
4. Implement the function `computecomponents` to compute how many bases account for 75% and 99% of the variance, respectively.
5. Display the mean face and the first ten Eigenfaces in a single figure by implementing `showeigenfaces`.
6. Choose a face from the data matrix by implementing `takeface`. Then implement the function `computereconstruction` that projects a face image to the low-dimensional space of the first n principal components and then computes a reconstruction of the face image from this projection. This function is used to reconstruct a face image using 5/15/50/150 components. Finally, implement `showreconstructedfaces` to show the 4 reconstructed faces as well as the original in a single figure.

Submission: Please include `problem2.jl` and a file `answersproblem2.txt` in your submission.