Computer Vision I, SS12 - Homework Assignment 2

June 5, 2013

Prof. Stefan Roth
Stephan Richter
Thorsten Franzel
Qi Gao

This assignment is due on Monday June 17th, 2013 at 23:55.

Please see the previous assignments for general instructions and follow the handin instructions from there.

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

Image pyramids are widely used in computer vision. In this problem you will build the Gaussian and Laplacian Pyramids for an image and solve a small image sharpening application. You will start with writing a few Matlab functions:

- Function `make_gauss_filter.m` with 2 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). Note: You should not use the Matlab built-in function `fspecial` to solve this problem.

- Function `downsample2.m` that takes an image and downsamples it by a factor of 2, i.e. resizes each dimension to half the size. To that end simply discard every other row and column. Note: You should not use the Matlab builtin `imresize` to solve this problem. Also, do not perform any Gaussian smoothing here; this will be done later.

- Function `upsample2.m` that takes an image and upsamples it 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 $\frac{1}{16}[1,4,6,4,1]$ (you should adapt it to 2D case) and multiply the results with 4. Note: You should not use the Matlab builtin `imresize` to solve this problem.

Then go on finishing the following tasks:

- Build a 6-level Gaussian pyramid of the image `a2p1.png`. For that implement the function `make_gaussianpyramid(...)`. To create the next level of Gaussian pyramid, filter the image with a Gaussian kernel ($\sigma = 1.5$ and size 5×5) and then downsample by a factor of 2. You have to use the above functions you wrote. When doing the filtering, use symmetric boundary conditions (*i.e.*, image outside the valid region is filled by mirror-reflecting the image across the borders) and make sure that the filtered image has the same size as the input. Display the resulting Gaussian pyramid in a single figure like Fig. 1.

- Based on the Gaussian pyramid, create the corresponding Laplacian pyramid. For that implement the function `make_laplacianpyramid(...)`. For building the Laplacian pyramid, you should also use the upsampling function from above. What is the difference between the top (coarsest) levels of Graussian and Laplacian pyramids? Make a comment in your code. Also, display the Laplacian pyramid in the same fashion as Fig. 1.

- Use the obtained Laplacian pyramid of `a2p1.png` to sharpen the image. In particular, amplify the high-frequency components contained in the finest 2 levels of the Laplacian pyramid by scaling them up by a factor. Afterwards, reassemble the pyramid back to obtain a sharpened full-resolution image and display it. Implement the reassembling in function `reconstruct_laplacianpyramid(...)`. 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 subband too much; try to avoid that. Briefly explain your findings.

Figure 1: Gaussian pyramid

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). Your tasks are:

- Implement the function `[data, imdims, nfaces] = loadfaces()`, which loads all face images from the `yale_faces/` subdirectory. The function returns 3 entities. `data` is the linearized and concatenated image data with size *MxN*, where *M* is is the number of face dimensions, and *N* is the number of faces. `imdims` contains the face dimensions, and `nfaces` contains the number of faces.
- Perform PCA on the loaded data. 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; in particular you want to use the "economy mode" (`[U,S,V] = SVD(X,0)` in Matlab).
- Show a plot of the cumulative variance of the principal components. For that make sure that the principal components are properly sorted in decreasing order of their variance. Also make a comment in your code to explain why the variance of each principal component is obtained in that way.
- Compute and print on screen how many bases account for 95% of the variance. Do the same for 80% of the variance.
- Display the mean face and the first ten principal components ("eigenfaces").
- Choose one face image, project it using the first 5 (15, 50, 150) principal components to get vectors of coefficients representing the face in this subspace. Then, compute the approximate versions of the face using the subspace. Show all the reconstructed face and its original in one figure (`subplot` might be useful here).

Problem 3 - Harris Detector (10 points)

Harris detector identifies corner-like structures by searching for points $p = (x, y)$ for which the structure tensor $\mathbf{C}$ around p has two large eigenvalues. The matrix $\mathbf{C}$ can be computed from the first derivatives at p, spatially weighted by a Gaussian filter kernel $G(\tilde{\sigma})$:

$$\mathbf{C}(\sigma, \tilde{\sigma}) = G(\tilde{\sigma}) * \begin{bmatrix} D_x^2(\sigma) & D_x D_y(\sigma) \\ D_x D_y(\sigma) & D_y^2(\sigma) \end{bmatrix} = \begin{bmatrix} G(\tilde{\sigma}) * D_x^2(\sigma) & G(\tilde{\sigma}) * D_x D_y(\sigma) \\ G(\tilde{\sigma}) * D_x D_y(\sigma) & G(\tilde{\sigma}) * D_y^2(\sigma) \end{bmatrix}, \tag{1}$$

where "$*$" is the convolution operator, and $D_x(\sigma), D_y(\sigma)$ denote the partial (horizontal and vertical) derivatives of the image that has been smoothed using a Gaussian smoothing filter with kernel width σ (Note the difference between σ and $\tilde{\sigma}$). The interest points are defined as those points whose Harris function values are larger than a certain threshold t

$$\det(\sigma^2 \mathbf{C}) - \alpha \cdot \left(\mathrm{trace}(\sigma^2 \mathbf{C})\right)^2 > t. \tag{2}$$

Note that we include an additional scale normalization factor σ^2 so that we can use the same threshold t independently of the value of σ. In practice, the parameters are usually set as: $\tilde{\sigma} = 1.6\sigma, \alpha = 0.06$.

In the following tasks, please use the Matlab built-in function `fspecial` to generate Gaussian smoothing filters with size 25×25. Compute the image derivatives with central differences, *i.e.*, $\frac{1}{2}[-1, 0, 1]$ and its transpose. For color images, you only need to detect the interest points in the grayscale space. (Function `rgb2gray` might be useful.)

Figure 2: Harris points.

Tasks:

- Load the image `a2p3.png`. Use $\sigma = 2.4$ to obtain the structure tensor $\mathbf{C}$. Compute the Harris function values for each pixel and display these values as an image. Please always use replicate boundary conditions.
- Then apply *non-maximum suppression* on the result to extract the local maxima, *i.e.*, points for which the value of Harris function given by Eq. (2) is the largest within their surrounding 5×5 windows.
- Find all local maxima with a Harris function value that is larger than the threshold $t = 2800$. Mark these interest points with "$\times$" in the original image (using `plot(·,·,'x')`), as shown in Fig. 2. (Please note that the results in this figure are ***not necessarily*** correct.)