2nd Exercise on Medical Image Processing

LU 183.630 - 2016SS

30. Mai 2016

Question to: Markus Krenn: `markus.krenn@meduniwien.ac.at`,
Markus Holzer: `markus.holzer@meduniwien.ac.at`

1 Submission guidelines

- One submission per team, including team member names and group number.
- Submission per mail to `markus.krenn@meduniwien.ac.at`
- Deadline: June 24th, 12pm.

Submission content:

- Executable code as `.zip`, filename: `Abgabe-PF-XX.zip`, where `XX` holds your group number.
- A script `RUN.m`, which computes all results and plots without user interaction.
- The code has to be documented including information on which exercise is solved in which function or file.
- A PDF report that contains explanation and interpretation of all exercises (approximately 5 - 8 pages).
- The report has to be short and precise, (do not include code).
- The report can be written in german or english.
- Questions regarding the exercise / feedback to your code or parts of your report: during laboratory sessions or per mail.

2 Submission

The second exercise aims at the implementation of a segmentation algorithm that uses the PCA model build within the first exercise of this course resulting in a simplified version of Particle Filters[1]. Relevant topics within this exercise:

- Feature extraction
- Classification and feature selection using Random Forests
- Implementation and optimization of a cost function for segmentation

2.1 Data and helper functions (including documentation) provided within the source code:

- The dataset `handdata.mat`: `images` contains images, `masks` contains contour masks of the objects, `landmarks` contain $n_{dim} \times n_{landmarks} \times n_{bones} \times n_{images}$ landmarks and `aligned` contains already aligned landmarks of each bone of the PCA.
- `computeHaarLike.m` computes Haar-like features of an image
- `optimize.m` optimizes a given cost function
- `optimizeDEMO.m` demonstrates the functionality of `optimize.m`
- `cache.m` serves as helper function to store and cache intermediate results under a unique name (see also `cacheDEMO.m`)

Other eventually required matlab functions: `gradient`, `meshgrid`, `TreeBagger`, `randperm`.

2.2 Exercises

Maximum number of points of subtasks are denoted in brackets (40 in total). Images 1-30 are part of the training data (PCA model and classification training), images 31-50 are part of the test data to evaluate your implementation.

1. **Shape-Modell (5 Punkte)** Extend your function `generateShape.m` so that it allows rotation, scaling and translation of shapes according to the parameters r, s, x, y (hint: rotation matrix). Next to the parameter vector b, the extended function should have 4 additional parameters: $p = (b, scaling, rotation, x-$

[1]see PDF `deBruijne2004MICCAI_ParticleFilters.pdf`

$translation, y-translation$). Similar to the first exercise, plot shapes for different values of scaling and rotation parameters.

2. **Feature extraction (7 Punkte)** Implement a function `computeFeatures(image)` that returns a $n_{features} \times n_{pixels}$ dimensional feature matrix of an image, including the following features:

   - Grey value of an image
   - Gradient in x- and y- direction
   - Magnitude of the gradient
   - Haar-like features[2] of the gray value image, using the function `computeHaarLike.m`.
   - Haar-like features of the gradient magnitudes
   - x- und y- coordinates of a pixel

   Illustrate the features of image 1 using `imagesc` (use only the first item of haar-like features). Once you've correctly implemented this function you can use `cache` to store computed features on the disk so that they do not have to be recomputed within each run of your code. Feel free to implement and evaluate other additional features!

3. **Classification & Feature-Selection (11 Punkte)** Use features of the training images to train a classifier that is able to classify edges of objects to be segmented. Use the Random Forest classifier[3] for this purpose which is implemented in the Matlab class `TreeBagger`.

   (a) Implement a function `train(images, masks)` that computes features for all images and trains a Random Forest features of all images and their masks as class labels. The function call to train a Random Forest should look like
   `rf = TreeBagger(32,features',labels','OOBVarImp','on');` Hint: To speed up the training process you should use all pixels of the bone contours but only a randomly sampled subset of the background pixels (equal amount of fore- and background samples).

   (b) Evaluate and interpret the impact of the number of trees using `oobError`.

   (c) Evaluate and interpret the importance of different features using `plot(rf.OOBPermutedVarDeltaError)`.

[2] `http://www.cognotics.com/opencv/servo_2007_series/part_2/sidebar.html`
[3] `http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm`

4. **Shape Particle Filters (17 Punkte)** In the next step we formulate a function that models costs of fitting a shape to a target image. I.e. we are looking for a point in a parameter space (described by shape parameters, rotation, scaling and translation) which describes an optimal fitted shape that segments the contours of a target object.

   (a) Create a function `train` that trains a classifier on all training images and creates a PCA shape model. Furthermore, implement a function `predict` that predicts a segmentation on the test images.

   (b) Create a cost function that takes a parameter vector p and the classification result of an image as input and returns a scalar value which describes how well the (from p) generated shapes fits the classification result. (The better the shape fits the classification result, the lower the returned value).

   (c) Optimize this function for all test images. We are using a stochastic optimization approach called Differential Evolution[4] for this purpose. This method is very simple, robust and converges fast. We provide an implementation of this approach in `optimize.m`. An example to create and use a cost function for an optimization process can be found in `optimizeDEMO.m`. You can also use the Matlab-Function `ga` which implements a generic algorithm.

   (d) Investigate the segmentation performance of your method (`boxplot` are pracitcal tools for illustration). Interpret the impact of all steps of the algorithm, describe investigated variants, the convergence criteria, etc.

[4] `http://en.wikipedia.org/wiki/Differential_evolution`