PROJECT 'UNSUPERVISED IMAGE SEGMENTATION'

INTRODUCTION

A subtask in many image analysis and computer vision task is image segmentation: the partitioning of the image plane into a number of meaningful, disjoint regions. The approach for image segmentation that is considered here is *pixel classification*. The goal is to assign a class label to each pixel of the image. Pixels with the same label form a region.

For simple problems, the objects of interest have a large contrast with the background of the scene. In these cases, comparing the grey level of a pixel against a threshold may suffice to determine whether the pixel belongs to the foreground (= object), or to the background. See Figure 1. However, most often such a strategy fails because the grey level of a single pixel is just not discriminating enough.

A more advanced technique is to bring in the spectral information of a pixel, and to use that as a measurement vector. For instance, in RGB-colour images each pixel comprises three measurements (red, green and blue). Pixel classification then boils down to a classification problem involving a 3-dimensional measurement space. Figure 2 provides an example.

In some applications, the segments are not defined by the colour, but rather by the textural information, i.e. the more or less repetitive spatial pattern that the grey levels (or colours) of the pixels in a region show. See Figure 3.

The goal of this project is to design and evaluate a pixel classification system for images in which both colour and texture are important clues for image segmentation. An example of such an image is depicted in Figure 4. Starting point of the design is a matlab script `ut_getgabor` that will be used to select a set of pixels from an input image, and to convert this selection to a set of measurement vectors containing textural and/or colour information. Using this set of measurement data an unsupervised training method must be designed and evaluated. (The examples in Figure 1 up to 3 are made using `ut_getgabor` as a pre-processor for obtaining the training set).

`ut_getgabor` can also be used to create a labelled data set. We will use such sets to evaluate the classifiers obtained by unsupervised learning.

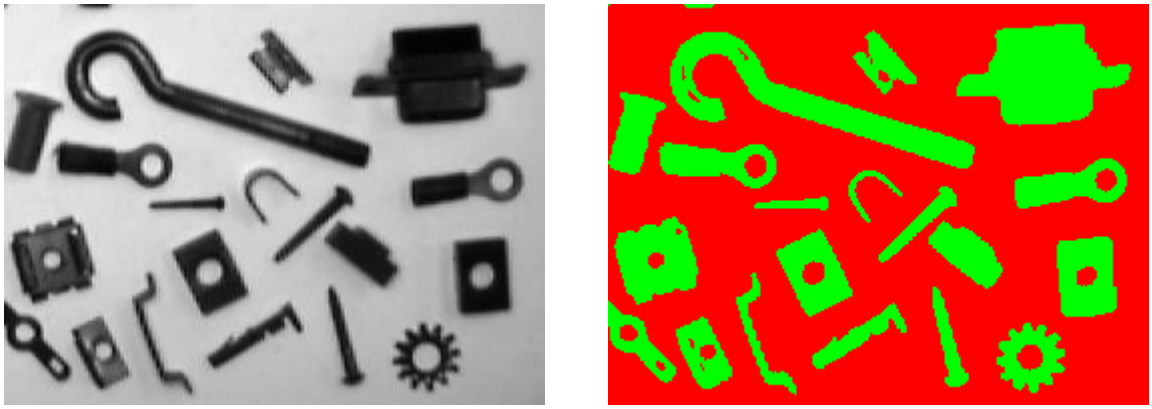

Figure 1. Pixel classification implemented by thresholding the grey levels

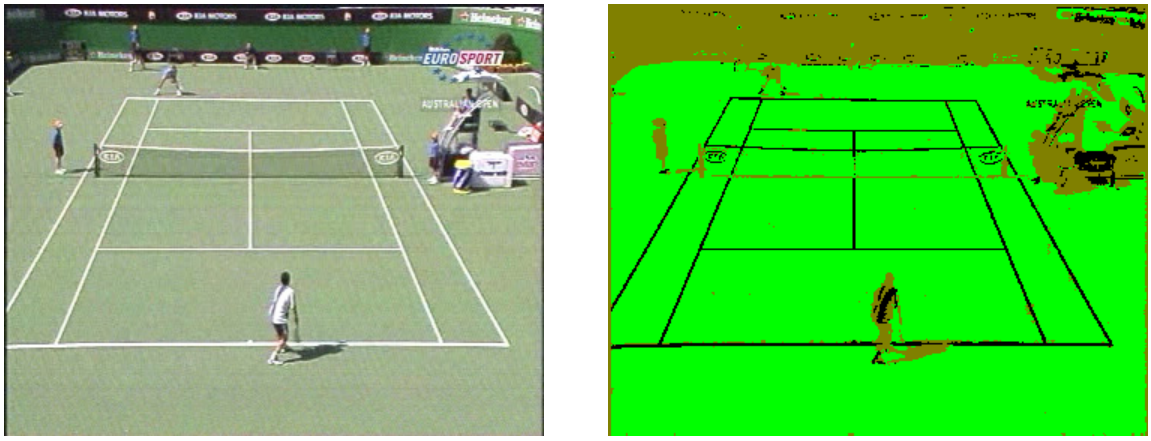

Figure 2. Pixel classification using RGB colour information

Literature:

- Chapter 2
- Chapter 3
- Chapter 7
- Section 9.1

Requirements:

- Matlab 6.5 or higher
- At least 512 Mbyte RAM
- PRTools Version CPESE
- Image processing toolbox
- Datafile: CPESE_PROJ_CLASS2.ZIP (includes images and `ut_getgabor.m`)

IMPLEMENTATION OF THE PIXEL CLASSIFICATOR

A short note on Gabor filters

The software `ut_getgabor` that is provided within this project uses log radial Gabor filtering to extract textural information from the neighbourhood of each image pixel. A Gabor filter is a bandpass filter with a Gaussian transfer function. In one dimension, the filter is defined by two parameters: the centre frequency u_c and the bandwidth B_u:

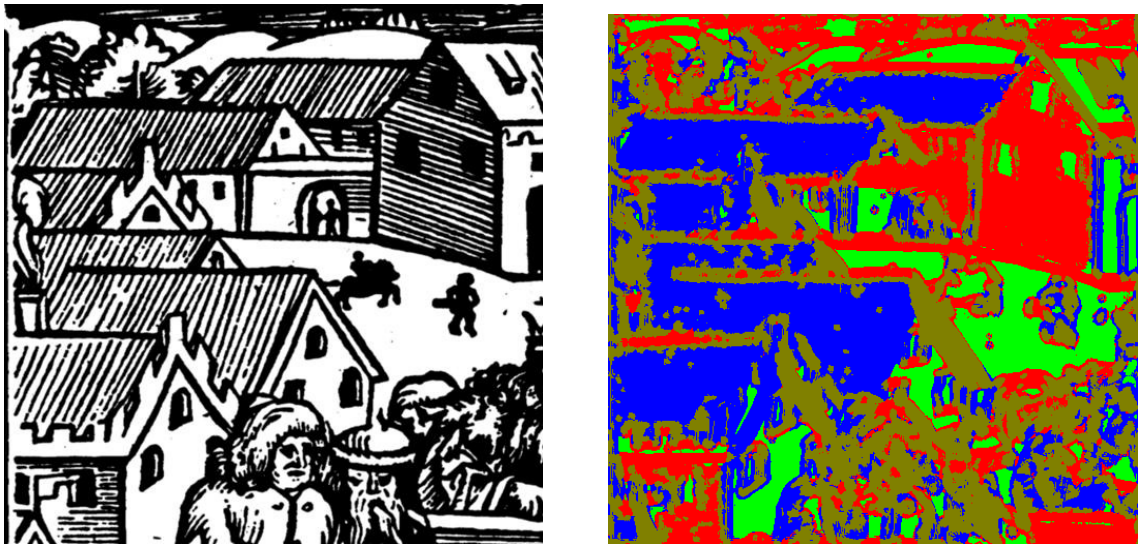

Figure 3. Pixel classification using textural information

Figure 4. Example of an image containing coloured textures.

$$H(u) = \exp\left(-\frac{(u-u_c)^2}{2B_u^2}\right) \tag{1}$$

where u is the frequency of harmonic function. There are two impulse responses associated with such a filter transfer, the so-called *inphase* response:

$$h_1(x) = 2B_u\sqrt{2\pi}\cos(2\pi u_c x)\exp(-2\pi^2 x^2 B_u^2) \tag{2}$$

and the *quadruture* response:

$$h_2(x) = 2B_u\sqrt{2\pi}\sin(2\pi u_c x)\exp(-2\pi^2 x^2 B_u^2) \tag{3}$$

Thus, actually u_c and B_u are associated with two Gabor filters. The two filter responses differ only in their phase. It is not difficult to show that:

$$\sqrt{h_1^2(x)+h_2^2(x)} = 2B_u\sqrt{2\pi}\exp(-2\pi^2 x^2 B_u^2) \tag{4}$$

In other words, the magnitude of $[h_1(x) \quad h_2(x)]$ rules out the carrier frequency u_c and only the envelope of each response is left. This trick can also be applied to the filtered images. Suppose that $f(x)*h(x)$ denotes convolution, then:

$$g(x) = \sqrt{\left(f(x)*h_1(x)\right)^2 + \left(f(x)*h_2(x)\right)^2} \tag{5}$$

is called the magnitude of the Gabor filtered signal.

In the 2-dimensional image domain, an harmonic wave function is characterized by two frequencies, one measured along the x-axis, and another along the y-axis. Alternatively, these two frequencies are described in polar coordinates: the radial frequency ρ (= number of periods per unit distance) and the orientation θ. A 2-dimensional radial Gabor filter is a filter whose transfer is expressed in these two coordinates:

$$H(\rho,\theta) = \exp\left(-\frac{(\rho-\rho_c)^2}{2B_\rho^2}\right)\exp\left(-\frac{(\theta-\theta_c)^2}{2B_\theta^2}\right) \tag{6}$$

For several reasons it is advantageous to define this filter on a logarithmic scale of the radial frequency. By doing so, the transfer function becomes:

$$H(\rho,\theta) = \exp\left(-\frac{\left(\log(\rho)-\log(\rho_c)\right)^2}{2\left(\log(\sigma)\right)^2}\right)\exp\left(-\frac{(\theta-\theta_c)^2}{2B_\theta^2}\right) \tag{7}$$

This is the transfer function used in `ut_getgabor`.

Creating the dataset

The script `ut_getgabor` opens the window as shown in Figure 5:

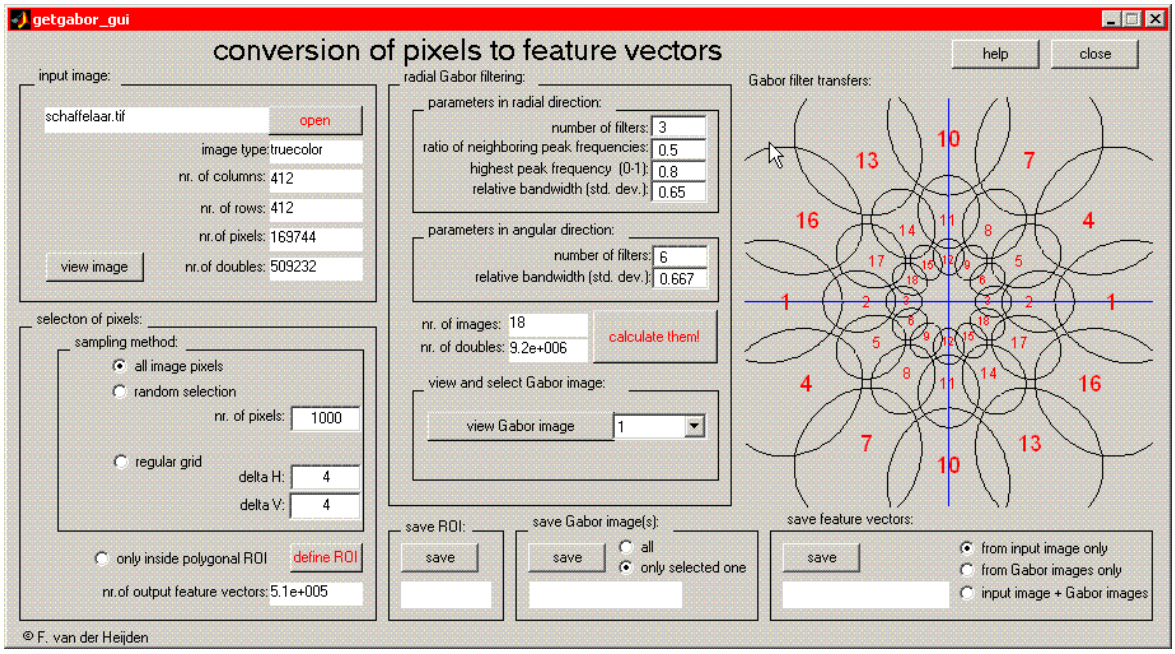

Figure 5. The graphical user interface of "ut_getgabor".

This gui (graphical user interface) permits us to specify an input image to which a number of Gabor filters can be applied. The parameters of these filters are specified in the panel 'radial Gabor filtering'. In the example above, there are 6x3=18 Gabor filters with central frequencies that form an orthogonal grid in polar coordinates (radial direction versus angular direction). The parameters are selected such that the full frequency domain is covered. For an extended explanation of the parameters: see the help. The output images, produced by the filters, are stacked so as to produce an image where each pixel is represented by an *M*-dimensional measurement vector. If, for instance, the input is RGB, and there are 18 Gabor filters, then each output pixel is an 3x18=54 dimensional measurement vector. See the Figure 6.

The number of pixels in the input image can be too much for a training set. For instance, if the input image has 500x500 RGB pixels, then the number of output measurements would be 500x500x3x18= 13.500.000 real numbers. Most training algorithms cannot handle so much data. The panel 'selection of pixels' offers provisions for selecting only a subset of the pixels. There are three mutual exclusive options:

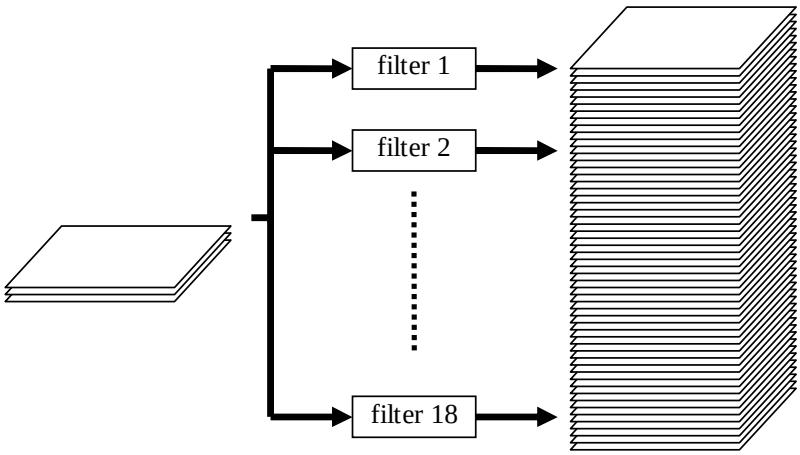

Figure 6. Building a measurement vector for each pixel by stacking the outputs of a number of texture filters.

- all image pixels are selected
- a random selection
- the image is subsampled along the rows and columns.

Apart from that, the user may also define a polygonal region of interest (ROI). In that case, only pixels inside the ROI are selected. This provision permits the manual creation of labelled datasets.

The output, i.e. the selected measurement vectors are saved to file in Matlab's '*.mat' format.

Importing and exporting datasets to and from PRTools

Typically, `ut_getgabor` is used to select a relative small data set of measurement vectors. This set is the one used for training. In case of unsupervised learning, there will be only one such a set. The set is stored in a file, say, `trainset.mat`[1]. Should the user want to apply his designed classifier to the whole image, then he has to press the "all image pixels" button, and store the vectors in a file called, say, `fullim.mat`. After closing `ut_getgabor` the procedure continues as follows:

```
load trainset;                % load the training set created in ut_getgabor
z=dataset(trainset);          % create a PRTools dataset
z=setlabtype(z,'soft');       % labels are set to soft
…
w = ……                        % create a mapping using PRTools facilities for unsupervised
…                               training

nrow = 412;                   % number of rows in the image
ncol = 412;                   % nr. of columns
load fullim;                  % load the set of all measurement vectors created in ut_getgabor
T=dataset(fullim);            % transfer all image vectors to a PRTtools dataset
x=T*w*classc;                 % apply a mapping and classify
labels=x*labeld;              % extract the labels from the classified pixels
imlabeled = reshape(labels,nrow,ncol);
                              % bring the labels in image format.
imshow(imlabeled,vga);        % show the labeled image (format: indexed; vga is a colourmap)
```

The line `x=T*w*classc;` can be very demanding with respect to memory usage. Therefore, if the image is too large, it may beneficial to break up the image into a number of blocks, to apply the operation to the different blocks individually and finally to merge the blocks. For instance, one can process the image row by row:

```
labels=zeros(length(fullim),1);
for i=1:nrow
    T=dataset(fullim(1+(i-1)*ncol:ncol*i,:));
    labels=T*w*classc*labeld;
    imlabeled(1+(i-1)*ncol:ncol*i,:) = labels;
end
imlabeled = reshape(imlabeled,nrow,ncol);
```

Design strategy

The PRTools cluster-finding algorithms that can be applied directly to this problem are:

- K-means
- EM-algorithm for mixture of Gaussians
- Mixture of probabilistic PCA's

Since the dimension of the measurement space is large, it might be advantageous to map this space

[1] In case of supervised learning (or evaluation) the procedure is repeated for each class with the ROI option on, and with manually defined ROI's, one for each class. This results in as many files as number of classes.

first to a lower dimensional space. Mappings for such include:

- PCA
- SOM's
- GTM's

These mappings do not provide a clustering of the data. Therefore, they should be followed with the cluster-finding algorithms listed above. Note that PCA is fast, and not difficult to train. However, it is a linear feature extractor and thus perhaps of limited use. SOM's and GTM's can establish nonlinear feature extraction, but they are difficult to train in high-dimensional data. For instance, the PRTools function `gtm` iteratively estimates the parameters of a nonlinear model of the data. But if the dimension of the data is too large, the iteration may not converge at all (in which case `gtm` returns an empty array). For these reasons it might be advisable to use PCA to obtain a first reduction of the dimension, and apply `som` and/or `gtm` next to further reduce the data to 1, 2 or 3 dimensions. (Hint: before applying PCA, normalize the data with respect to mean and variance).

The cluster-finding algorithms and mappings depend on parameters that should be selected. In order to find the best parameters we need a performance criterion. The result of pixel classification can be visualized easily (see Figure 1, 2 and 3). Such visualization permits a quick assessment of the performance of the classifier, but for parameter tuning a quantitative evaluation criterion (i.e. in terms of error rate or confusion matrix) is needed. However, for a quantitative criterion we need labelled evaluation data. These can be created manually by using the ROI option in `ut_getgabor`.

The design task

The objective is to find the best classifier for the image shown in Figure 4 (filename: schaffelaar.tif). The performance of the classifier is measured in terms of the error rate. Once a good design structure has been found, it is important to repeat the training procedure a couple of times to check the reproducibility of the procedure.

We assume that the computational complexity of the classifier is not a design issue by itself. But of course, the design choices should be such that the classifier can be trained within a reasonable amount of time and with a reasonable amount of computer memory.