Technische Universität München
Fakultät für Informatik
Prof. Nassir Navab

WS 15/16
Exercise Sheet 2
October 21, 2015

Exercises in Machine learning in Medical Imaging

Exercise 1 General information

- The goal of this exercise sheet is to implement in MATLAB the extraction of standard features at each pixel from a given image.
- The features you are computing must be extracted from a grayscale image of your choice.
- The size a of patches or kernels must always be an odd number $a = 2\rho+1$. The centre of the patch/kernel is then clearly defined as the pixel $(\rho+1, \rho+1)$.
- Some questions are asked to be solved using a cross-correlation product, generally between an image I of size $n_{rows} \times n_{cols}$ and a kernel K of size $a \times a$ with $a = 2\rho+1$. As a reminder, the cross-correlation product $\otimes$ between I and K is the image of size $n_{rows} \times n_{cols}$ defined as

$$[I \otimes K](i,j) = \sum_{-\rho \leq \alpha, \beta \leq \rho} I(i+\alpha, j+\beta) K(\alpha, \beta)$$

  Up to a symmetry applied to the kernel, it is equivalent to a convolution product $\otimes$, defined as

$$[I \otimes K](i,j) = \sum_{-\rho \leq \alpha, \beta \leq \rho} I(i-\alpha, j-\beta) K(\alpha, \beta)$$

  We recommend to perform cross-correlation products using the syntax

$$imfilter(I, K, 'replicate', 'same')$$

  which automatically pads the image I on the boundaries.
- Some questions are asked to be solved using integral images which are obtained by creating a matrix holding in each pixel x the sum of all the image pixel values above that pixel. Please refer to the following for a more formal definition of integral images.
- In this exercise you will implement three classes, namely *BoxFeature.m*, *FilterFeature.m* and *IntImage.m*, plus a main file *main.m*.

Exercise 2 Integral Images

Integral images are an elegant and fast way to compute the sum (or the mean) of intensities over a rectangle. This section is dedicated to the computation of features based on this principle.

From an image I of size $n \times p$,we can define the integral image $\tilde{I}$ of size $(n+1) \times (p+1)$ by:

$$\tilde{I}(i,1) = 0 \; for \, i = \{1...n+1\}$$

$$\tilde{I}(1,j) = 0 \; for \, j = \{1...p+1\}$$

$$\tilde{I}(i,j) = \sum_{i'<i,\, j'<j} I(i',j') \; for \, i > 1;\; j > 1\}$$

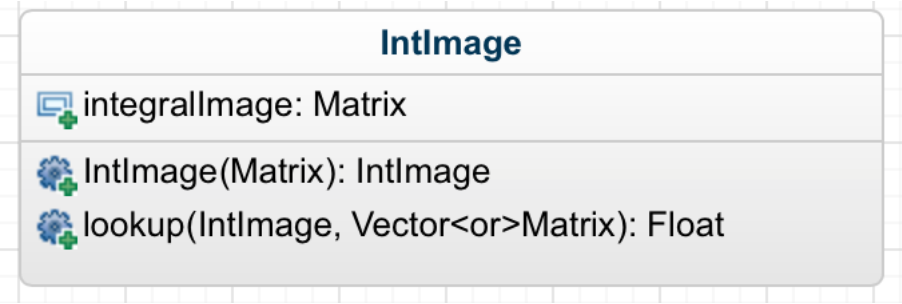

Figure 1: Class diagram IntImage.

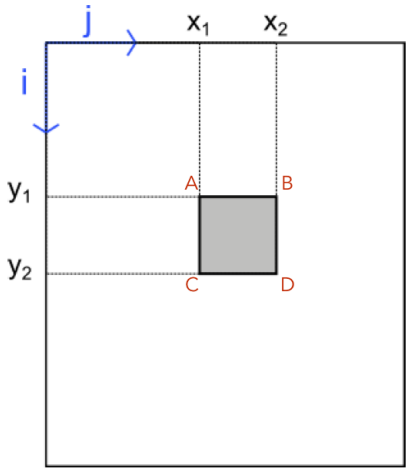

Figure 2: The sum of the values in the grey area is $v = \tilde{I}(D) - \tilde{I}(B) + \tilde{I}(A) - \tilde{I}(C)$

a) Implement the integral image computation in the constructor of the class *IntImage* (Figure 1), taking as input parameter an image I and returning an object of type *IntImage*. To improve efficiency you can use the following identity, valid for $i > 1$ and $j > 1$:

$$\tilde{I}(i,j) = I(i-1,j-1) + \tilde{I}(i-1,j) + \tilde{I}(i,j-1) - \tilde{I}(i-1,j-1)$$

Please refer to the comments in the file *IntImage.m* for further information.

b) Implement the method *value=lookup(obj,x)*, where $x = [row, col, h, w]$, that returns the sum of the intensities of I over the patch originating at (row, col) and having size h and w. For this, you can notice that, in the configuration of the Figure 2, the sum of intensities of I over the grey area is given by $v = \tilde{I}(D) - \tilde{I}(B) + \tilde{I}(A) - \tilde{I}(C)$ where A, B, C and D are the coordinates of the vertices of the rectangular region.

Please handle the case in which the user supplies a box $x = [row, col, h, w]$ that is not contained, or just partially contained into the image. In case the box is not contained in the image, the value returned by the method is zero. If the box is only partially contained into the image, it is resized appropriately. Please refer to the comments in the file *IntImage.m* for further information.

Exercise 3 Filter Features

This section is dedicated to the extraction of features that can be seen as filters over the image. The value of such a feature $x_k(i,j)$ is a linear combination of the intensities within the neighbourhood of (i,j). More formally, we can write:

$$x_k(i,j) = \sum_{-\rho \leq \alpha, \beta \leq \rho} I(i+\alpha, j+\beta) K(\alpha, \beta)$$

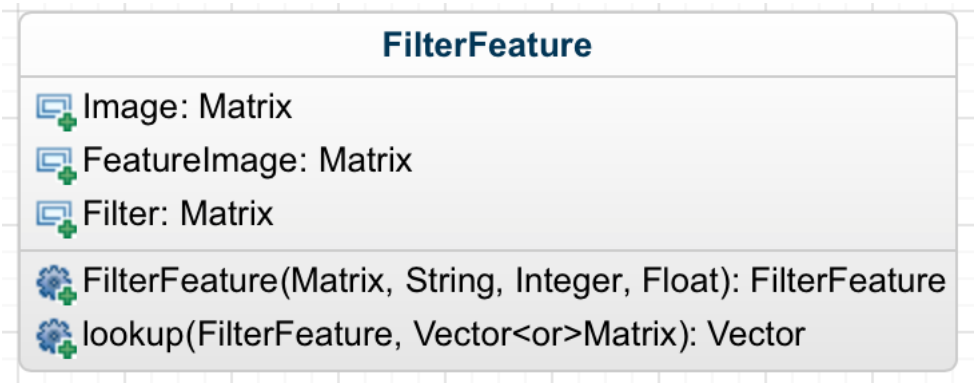

Figure 3: Class diagram FilterFeature.

where K, called the *cross-correlation* kernel, defines the coefficients of this linear combination.

a) Implement the constructor of the class *FilterFeature* which takes four arguments:

- *img*: the image you want to extract features from.
- *filterName*: a string expressing the type of the filter (see below).
- *siz*: which is the size of the filter. It is only one scalar expressing the size of the side of the filter which is always square. Not all the filter types require this parameter.
- *sigma*: the variance parameter used for some of the filters. Not all the filter types require this parameter.

The user must be able to choose the filter type by specifying one of the strings among the following:

- *gaussian*: Gaussian kernel of standard deviation *sigma*. You can choose $6sigma+1$ as the size of the patch. Use gaussian filter and Matlab fspecial(...).
- *LoG*: Laplacian-of-Gaussian kernel of standard deviation *sigma*. You can choose $6sigma + 1$ as the size of the patch. Use LoG filter and Matlab fspecial(...).
- *mean*: gives the mean of intensities over the patch of size *siz*.
- *horizontalDerivative*: gives the derivative along columns at each pixel. Use prewitt filter and Matlab fspecial(...).
- *verticalDerivative*: gives the derivative along rows at each pixel. Use prewitt filter and Matlab fspecial(...).
- *diagonalDerivative*: gives the derivative along the two diagonals at each pixel. the filter must be specified by the user and should be

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

- *std*: gives the standard deviation of intensities over the patch of size *siz*. Consider applying the mean filter to the squared image. The filtering has to be done in the constructor using the function *imfilter*. For the filter type *std*, you will most probably need to implement yourself the computation of the *filteredImage* by looping over the dimensions of the image.

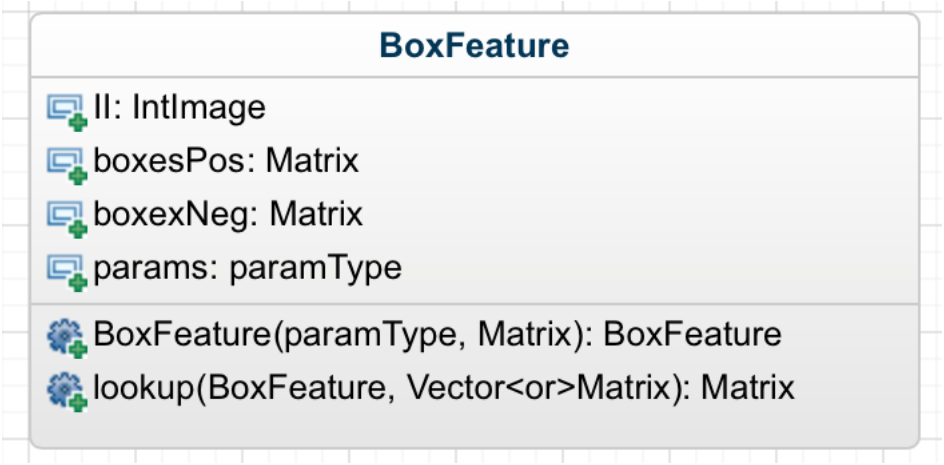

Figure 4: Class diagram BoxFeature.

b) Implement the function *lookup* of the class *FilterFeature* which takes the object itself and one argument x namely one or multiple coordinates (one coordinate per row), in the format $[x, y]$ or $[x, y, w, h]$ (where w and h are never used in this exercise, but we still want to be able to do our computations even if they are supplied by the user). When the user supplies the coordinates, the method should return the value of the *filteredImage* at that coordinate. In case the user requests features at coordinates that do not belong to the image, you have to return 0.

Exercise 4 Box Features

In this section you will implement the class BoxFeature. This class computes AVERAGES inside square regions of the image and performs operations between these averages. The operations are, differences with central box in case of *LBP* and *longRangeOffset* or differences between couples of boxes in case of *longRangeDoubleOffset*. In practice the user specifies, through the parameters, the type of features that need to be extracted, the size of the regions, and the offsets that represent the positions of the regions w.r.t. the pixel the features are extracted from. Using the member function *lookup* and passing a matrix of coordinates with as many row as sampling points and two (or four) columns one obtains a matrix *value* having as many rows as sampling points and as many columns as boxes.

a) Implement the constructor of the class *BoxFeature* which takes two arguments: the image to be processed and a structure called *params* containing user specified parameters. The form of *params* is:

- *params.type*: the type of box filter that you want to instantiate. it can be one of *LBP*, *longRangeOffset*, *longRangeDoubleOffset*.
- *params.sizes*: one scalar in case of *LBP*. A row vector with as many components as boxes when *longRangeOffset* or *longRangeDoubleOffset* is used.
- *params.offsets1*: significative only when *longRangeOffset* or *longRangeDoubleOffset* is used. It is a matrix having as many rows as desired boxes, and two columns. In each row you place a displacement $[d_x, d_y]$ that tells what is the displacement of the *central pixel* of box and the pixel where you want to extract the feature. In case of *longRangeOffset* the first box is the "positive" one, while the others are the "negative" ones. (the mean values from the negative ones get subtracted from the value of the positive one). In case of *longRangeDoubleOffset* all of these boxes are "positive" ones.

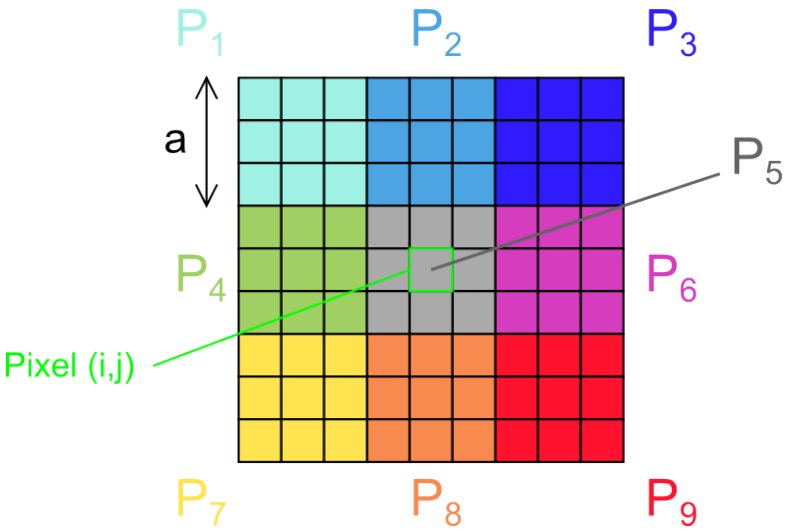

Figure 5: Example of boxes for LBP. Here the boxes have params.size=3. Central box in grey.

- *params.offsets2*: significative only when *longRangeDoubleOffset* is used. It is a matrix having as many rows as desired boxes, and two columns. This vector should have the same number of rows as *params.offsets1*. In each row you place a displacement $[d_x, d_y]$ that tells what is the displacement of the *central pixel* of box and the pixel where you want to extract the feature. These boxes are the "negative" ones. (the mean values from the negative ones get subtracted from the value of the positives ones). The constructor fills the variables *II*, *boxesPos*, *boxesNeg* using the information contained in parameter.

b) implement the *lookup* method which takes two argument, an instance of class *BoxFeature* and a matrix of coordinates with as many row as sampling points and two (or four) columns.

- In case of *LBP* features: the averages from 8 boxes around the central one are subtracted from the average of the central box itself (Figure 5) producing 8 values as feature vector (value has 8 columns). The coordinates of the 8 boxes should be stored row-wise in the matrix boxesNeg by the class constructor, while the coordinates of the central box should be in boxesPos.
- In case of *longRangeOffset* features: the user specifies N offsets corresponding to N boxes. The averages from the last $N-1$ boxes are subtracted from the average of the first box. The feature vector has $N-1$ values. The coordinates of the last $N-1$ boxes should be stored row-wise in the matrix boxesNeg by the class constructor, while the coordinates of the first box should be in boxesPos.
- In case of *longRangeDoubleOffset* features: the user specifies $N+N$ offsets(1,2) corresponding to $N+N$ boxes. The averages from the N boxes stored in params.offsets1 are subtracted from the average of the N boxes stored in params.offsets2. The feature vector has N values. The coordinates of the N boxes specified in params.offsets1 should be stored row-wise in the matrix boxesPos by the class constructor, while the coordinates of the N boxes specified in params.offsets2 should be in boxesNeg.

Exercise 5 **Main function**

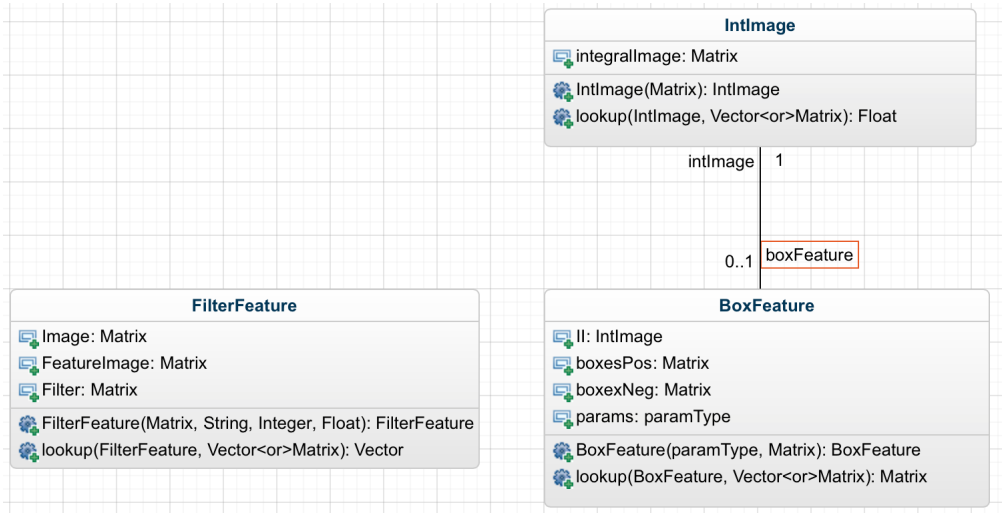

Figure 6: Global class diagram (tentative).

Please refer to the draft implementation that you find in the file *main.m* and create instances of the classes you have been implementing in this exercise having different parameters. Feel free to experiment with your own implementation in the body of this function.