Technische Universität München
Fakultät für Informatik
Prof. N. Navab, Ph.D.

WS 2014
Exercise Sheet Praktikum Machine Learning
October 7th, 2014

Exercises in Praktikum Machine Learning

0. 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 functions that are computing a feature vector

  $$\mathbf{x}(i,j) = \left[x_1(i,j),\ldots,x_{N_{features}}(i,j)\right]$$

  must return a cell array $\mathcal{X}$ of size $1 \times N_{features}$. $\mathcal{X}\{k\}$ is an image such that $\left[\mathcal{X}\{k\}\right](i,j)$ is the value of the feature x_k at the location (i,j).
- The features you are computing must be extracted from a grayscale image of your choice. The value of a given feature over the image can be displayed using the function *imagesc*.
- 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 $\star$ between I and K is the image of size $n_{rows} \times n_{cols}$ defined as

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

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

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

  We recommend to perform cross-correlation products using the syntax

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

  which automatically pads the image I on the boundaries.

1. Bank of filters

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

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

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

a) Implement a function *standard_filters.m* taking as parameters:

- the image I
- a cell array $\mathcal{F}$ of strings
- the side a of the square used as kernel (used for 'mean' and 'std')
- a standard deviation σ (used for 'gaussian' and 'LoG')

that returns the cell array $\mathcal{X}$ giving the feature responses for the filters specified in the cell array $\mathcal{F}$. The user must be able to choose one or several strings among the following:

- 'straight derivatives' : gives the derivative along rows and columns at each pixel. As an example, the derivation along rows can be defined as:

$$\frac{\partial I}{\partial i}(i,j) = \frac{1}{2}(I(i+1,j) - I(i-1,j))$$

- 'diagonal derivatives' : gives the derivative along the two diagonals at each pixel.
- 'mean' : gives the mean of intensities over the patch of size a.
- 'std' : gives the standard deviation of intensities over the patch of size a. Consider applying the mean filter to the squared image.
- 'gaussian' : Gaussian kernel of standard deviation σ. You can choose $6\sigma+1$ as the size of the patch.
- 'LoG' : Laplacian-of-Gaussian kernel of standard deviation σ. You can choose $6\sigma+1$ as the size of the patch.
- 'all' : returns all the previous filters for the given a and σ.

Some predefined kernels can be found using *fspecial*. The cell array $\mathcal{X}$ contains at most 8 components (if 'all' is selected).

2. Features based on 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.

a) 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

$$\tilde{I}(i,1) = 0 \text{ for } i \in \{1,\dots,n+1\}$$
$$\tilde{I}(1,j) = 0 \text{ for } j \in \{1,\dots,p+1\}$$
$$\tilde{I}(i,j) = \sum_{\substack{i'<i \\ j'<j}} I(i',j') \text{ for } i>1 \text{ and } j>1$$

Implement a function *integral_image.m* taking as input parameter an image I and returning its integral image $\tilde{I}$. You can use for this the following identity (valid for $i > 1$ and $j > 1$)

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

b) Implement a function *mean_patch.m* taking as input arguments

- an integral image $\tilde{I}$
- the side of a patch a
- the coordinates (α, β) of this patch

that returns the mean of intensities of I over the patch of side a centered on (α, β). For this, you can notice that, in the configuration of the Figure 1, the sum of intensities of I over the gray rectangle is given by

$$\sum_{\substack{x_1 \le x \le x_2 \\ y_1 \le y \le y_2}} I(y,x) = \tilde{I}(y_2+1, x_2+1) - \tilde{I}(y_2+1, x_1) - \tilde{I}(y_1, x_2+1) + \tilde{I}(y_1, x_1)$$

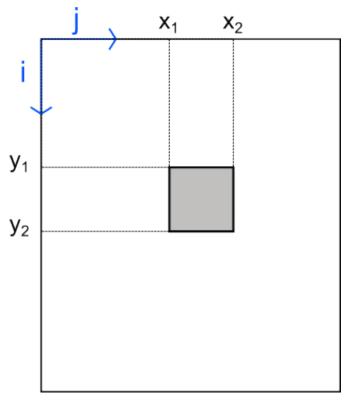

Figure 1: The integral image $\tilde{I}$ allows a very fast computation of the sum over the gray rectangle of the intensities of I

c) We take a patch of side a centered on the pixel (i, j) of interest and we consider its 8 neighbouring patches as described in Figure 2. By using *mean_patch.m* and a loop over the patch indexes, implement a function *mean_features.m* taking as input arguments

- an integral image $\tilde{I}$
- the side of the patches a

that returns the feature vector

$$\mathbf{x}(i,j) = [\mu_1(i,j), \mu_2(i,j), \ldots, \mu_9(i,j)]$$

as a cell array $\mathcal{X}$ of size 1×9 where $\mu_n(i,j)$ is the average of the intensities over the patch $P_n(i,j)$.

d) With a similar approach, implement a function *lbp.m* (for Local Binary Patterns) taking as input arguments

- an integral image $\tilde{I}$
- the side of the patches a

that extracts the binary feature vector

$$\mathbf{x}(i,j) = [x_1(i,j),\ldots,x_4(i,j),x_6(i,j),\ldots,x_9(i,j)]$$

as a cell array $\mathcal{X}$ of size 1×8, where $x_n(i,j)=1$ if $\mu_n(i,j)\geq\mu_5(i,j)$ and $x_n(i,j)=0$ if $\mu_n(i,j)<\mu_5(i,j)$.

e) We propose to compute now the same kind of features but on longer range.

(i) Implement a function *long_range_offset.m* taking as input arguments

- an integral image $\tilde{I}$
- a side of patches a
- an offset vector $\mathbf{w}=[u,v]$

that returns the feature vector $\mathbf{x}(i,j)=[x_1(i,j),x_2(i,j)]$ as a cell array $\mathcal{X}$ of size 1×2 where:

- $x_1(i,j)=\mu_w(i,j)-\mu_5(i,j)$, where $\mu_w(i,j)$ is the mean over the patch of side a centered on the pixel $(i+u,j+v)$
- $x_2(i,j)$ is the binarised version of $x_1(i,j)$: if $x_1(i,j)\geq 0$ then $x_2(i,j)=1$, else $x_2(i,j)=0$

(ii) Implement a similar function *long_range_two_offsets.m* taking as input arguments

- an integral image $\tilde{I}$
- a side of patches a
- a first offset vector $\mathbf{w}_1=[u_1,v_1]$
- a second offset vector $\mathbf{w}_2=[u_2,v_2]$

that returns the difference between the means of the patches centered on $(i+u_1,j+v_1)$ and on $(i+u_2,j+v_2)$, and its binary version. Again, the output is a cell array $\mathcal{X}$ of size 1×2.

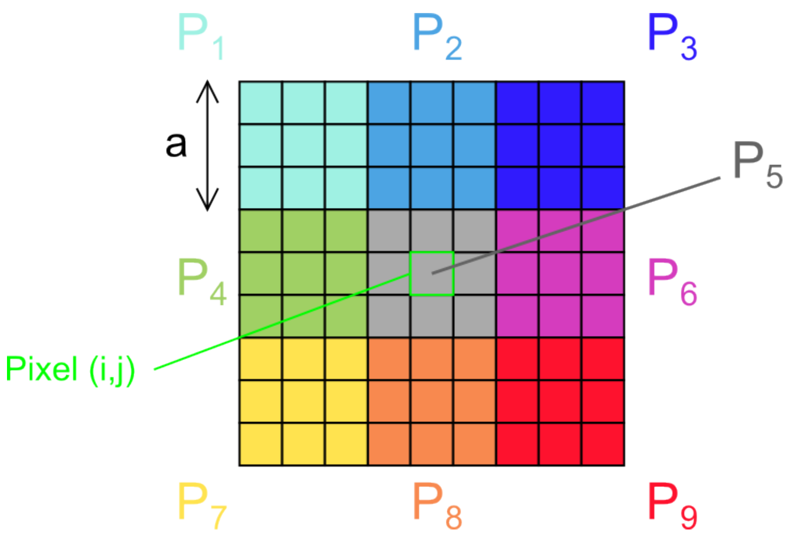

Figure 2: A patch of size $a=3$ centered on the pixel (i,j) of interest and its 8 neighbouring patches