Features to be employed in urine particle analysis – a few suggestions

P. Perona - Caltech

26 May, 2006
Updated 19 January, 2007

1 Introduction

I am listing here a few ideas for features that might prove useful in classifying urine particles. The differential invariant features were suggested by Koenderink and van Doorn and used by both Schmid and Ranzato in his thesis.

I will use this notation:

$$
\begin{aligned}
I &= I(x,y) \text{ the image} \\
G_\sigma &= G_\sigma(x,y) \text{ A zero-mean 2-dimensional symmetric Gaussian with std } \sigma \\
I * J &= \int\int I(u,v)J(x-u,y-v)dudv \text{ the convolution of images I and J} \\
I_\sigma &= I * G_\sigma \text{ An image blurred with a Gaussian kernel of width } \sigma \\
L_x &= \frac{\partial}{\partial x} \text{ the operator that takes the x derivative of a function} \\
I_x &= L_x I = \frac{\partial}{\partial x} I \text{ The derivative of the image w.r. to } x \\
I_{xy} &= L_{xy} I = \frac{\partial^2}{\partial x \partial y} I \text{ The derivative of the image w.r. to } x, y \\
L_x^p &= (\frac{\partial}{\partial x}\cdot)^p \text{ The operator that takes the x derivative and then the p power i.e.} \\
L_x^p I &= (\frac{\partial}{\partial x} I)^p \text{ may also write for clarity } L_x^p(I)
\end{aligned}
$$

most of these quantities will be defined later in the document, so do not panic yet!

2 Taking derivatives in images: finite differences

Images are sampled on a square lattice: the value of $I(x,y)$ is only available for $x \in \{1,2,\ldots,N_x\}$ and $y \in \{1,2,\ldots,N_y\}$. Thus, it is impossible to compute the derivative

of an image, the derivative is defined for functions defined on a continuum. One may define an approximation of the derivative by means of finite differences:

$$\begin{aligned} \frac{\partial I}{\partial x}(x,y) &\approx \frac{I(x+1,y)-I(x-1,y)}{2\cdot 1} \\ \frac{\partial^2 I}{\partial^2 x}(x,y) &\approx \frac{I(x+1,y)-2I(x,y)+I(x-1,y)}{4} \end{aligned}$$

The 'central difference' above is better than other approximations of the first derivative read the article on the Wikipedia:

`http://en.wikipedia.org/wiki/Finite_differences`

The 2nd derivative is obtained by taking the central difference of the central difference approximating the first derivative. (Try to say this sentence fast three times in a row. Now try and understand the meaning of the sentence. Which one was more difficult?). So: you can generalize the first derivative to any level of differentiation by taking more and more levels of differences.

Excercise - *Write the discrete expression for* $\frac{\partial^2 I}{\partial x \partial y}(x,y)$

At the boundaries of the domain of definition of the image the central differences is not defined. What one can do is define the value of the image by 'mirror' boundary conditions: $I(x=0,y)=I(x=2,y)$. As a result, the first derivative is zero (but other derivatives may be different from zero).

From now on, for the sake of simplicity, we will use the same notation for continuous derivatives and corresponding finite differences: the domain of definition of the function being differentiated will tell which operator corresponds to a given notation.

3 The convolution is a useful operator

For functions f, g of one variable $x \in R$, the convolution $f * g$ is another function of the same variable, and it is defined as:

$$(f*g)(x) = \int_R f(x-\tau)g(\tau)d\tau$$

For more details on the convolution and its properties see the Wikipedia article

`http://en.wikipedia.org/wiki/Convolution`

The generalization of the convolution to discrete bounded domains, where f is defined for $x \in \{1,\ldots,N\}$ and g is defined for $x \in \{-M_1,\ldots,M_2\}$ (notice that the two domains are different, and in our applications typically $M << N$):

$$(f*g)(x) = \sum_{\tau=-M}^{M} f(x-\tau)g(\tau) \qquad x \in \{1+M_1,\ldots,N-M_2\}$$

3.1 Implementing derivatives using the convolution

Why are we talking about convolution here? Because one may write finite differences and other useful operators on images in terms of convolutions of the image I with small function which are typically called 'kernels' k. For example: consider the kernel $k = \frac{1}{2}[1, 0, -1]$ (this is shorthand to say $k(-1) = 1/2, k(0) = 0, k(1) = -1/2$) and convolve it with an image $I(x, y)$:

$$(I * k)(x, y) \;\; = \;\; \sum_{\tau=-1}^{1} I(x - \tau, y)k(\tau) = \frac{I(x + 1) - I(x - 1)}{2} \qquad x \in \{2, \ldots, N - 1\}$$

Thus, taking the 1st derivative corresponds to convolving the image with the kernel $k = \frac{1}{2}[-1, 0, 1]$.

Excercise – *Write the kernel corresponding to taking the 2nd derivative of the image along x and the one corresponding to taking the 1st derivative of the image along y.*

In `Matlab` there is a function called `conv2` for computing convolutions of images. Suppose that `ima` is the variable containing the image. Then here is how to compute the first derivative of an image in x and y:

```
k = [1 0 -1]/2;
I_x = conv2(ima,k,'same');
I_y = conv2(ima,k','same');
```

Notice the argument `'same'`, which is passed along to the function `conv2`: this tells `Matlab` to use mirror boundary conditions, so that the domain of definition of `I_x` and of `I_y` is the same as that of `ima`.

Excercise – *Also notice that the derivative in y was obtained by using the kernel* `k'`. *Explain why.*

Excercise – *Define the image as a white field (image equal to 1) with a black (image = 0) rectangle in the middle. Compute its first derivative in x and its first derivative in y and display the result. Use the function* `imagesc` *to display the results. OK, just this time I will be nice. Here is how to do it:*

```
I = zeros(100,100);
I(30:60,40:70)=1;
k = [1 0 -1]/2;
figure(1);
subplot(1,3,1); imagesc(I); axis equal; axis off; title('Image')
Ix = conv2(I,k,'same');
```

```
Iy = conv2(I,k','same');
subplot(1,3,2); imagesc(Ix); axis equal; axis off; title('x derivative of the image');
subplot(1,3,3); imagesc(Iy); axis equal; axis off; title('y derivative of the image');
```

Is this what you were expecting? Observe that the y derivative of the image does not look right: why? Hint: what is the coordinate system that Matlab uses for matrices? Which way is the y axis pointing?

3.2 Blurring the image

Excercise – *Write some* `Matlab` *code to blur the image of the white square in a black field defined above. You can do it using the kernel* $k = [1, 2, 1]/4$. *First blur along the x direction (convolution with k) and then blur along the y direction (convolution with k'). Look at the result. Blur again a few times. Look at the result again. This is called 'Gaussian' blur because if you blur repeatedly a delta function (a black image with a white dot in the middle) you obtain a Gaussian function. Try. The* σ *of the Gaussian increases with the number of times you blur with the kernel k defined above.*

Why is blurring a useful operation? Blurring will not alter the coarse structure of the image very much, but it will destroy the fine-grained details. Conversely: taking the difference between the original version of the image and a blurred version of the image will highlight the fine-grained details and eliminate the coarse structure. Thus: blurring is a useful tool to generate different features.

4 Differential invariants

Ranzato used the convolution with a Gaussian, differential operator and other nonlinear operators to define 108 features. (See Ranzato's thesis, Chapter 6, pages 39 and following, and see Ranzato's draft paper, pages 7 and following.)

Consider the first 9 differential operators that are rotation-invariant, i.e. they commute with a rotation of the image in the (x, y) plane.

$$
\begin{aligned}
K_0 &= \delta(x,y) \text{ (the identity operator)} \\
K_1 &= L_x^2 + L_y^2 \\
K_2 &= 2L_xL_{xy}L_y + L_x^2L_{xx} + L_y^2L_{yy} \\
K_3 &= L_{xx} + L_{yy} \\
qK_4 &= L_{xx}^2 + L_{yy}^2 + 2L_{xy}^2 \\
K_5 &= L_{xxx}L_y^3 - L_{yyy}L_x^3 + 4L_{xyy}L_x^2L_y - 4L_{xxy}L_xL_y^2 \\
K_6 &= L_{xxy}L_y^3 - L_{xyy}L_x^3 + L_{xxy}L_x^2L_y - L_{xyy}L_xL_y^2 \\
K_7 &= L_{xyy}L_y^3 - L_{xxy}L_x^3 + L_{xxx}L_yL_x^2 - L_{yyy}L_xL_y^2 + 2L_{xxy}L_xL_y^2 - 2L_{xyy}L_y^2L_y \\
K_8 &= L_{xxx}L_x^3 + L_{yyy}L_y^3 + 3L_{xxy}L_x^2L_y + 3L_{xyy}L_xL_y^2
\end{aligned}
$$

just to make sure that the notation is clear: e.g. $K_1 I = (LxI)^2 + (LyI)^2 = (I_x)^2 + (I_y)^2$.

Consider the following operators:

$$\begin{aligned} m(I) &= \text{mean background value of image } I \\ f_1(I) &= f^+(I - m(I)) \\ f_2(I) &= f^-(I - m(I)) \\ f_3(I) &= f_1(I) + f_2(I) + m \\ D(I) &= \{(x,y) \mid |I(x,y)| > 0\} \text{ (support of I)} \\ A(I) &= \frac{1}{|D(I)|} \int_{D(I)} I(x,y)dxdy \text{ (mean of I over its support)} \end{aligned}$$

where $f^+(I)$ is the positive part of I and $f^-(I)$ is the negative part.

Then the features $F_{\sigma,j,k}$ are defined as

$$F_{\sigma,j,k} = A(f_k(K_j(I * G_\sigma)))$$

Ranzato recommends using the four scales $\sigma = 0, 1, 2, 3$. There are nine invariants and three nonlinearities f_k, thus there are $4 \times 9 \times 3 = 108$ features in Ranzato's system. You should feel free to experiment with more scales and different nonlinearities as well.

Here is a possible implementation: For every image I compute in the order:

1. I_x by convolving I with the 1st derivative kernel $k_x = [1, 0, -1]$. I_y by convolving I with the kernel $k_y = k_x^T$. Compute all other derivatives by appropriate sequences of convolutions of I and I_y with k_x and k_y.

2. Blur those derivatives using repeated applications of the blurring kernel $k_1 = [\frac{1}{4}, \frac{1}{2}, \frac{1}{4}]$ and k_1^T. I.e. $I * G_1 =$ `conv2(conv2(I,k1,'same'),k1','same')` (i.e. be sure to convolve with the blurring kernel both in the x and y direction).

3. $m(I)$ by taking the median brightness of a 5-pixel-wide strip of pixels at the boundary of the image (maybe 10-pixel-wide as Ranzato did). Since the differential invariants, apart from K_0 have mean zero, it is OK to assume that $m(I)$ is zero for all but the first one, K_0.

4. Compute the positive, negative and absolute value functions f_k of each one of the 4×9 blurred affine-invariant images. This produces 108 images.

5. Compute the indices of the support of each one of the 108 images J e.g. using the instruction `support = find(abs(J) > 0)`. Compute the average of one such function by `F = sum(J(support)) / length(support))`.

5 Other ideas for features

Size of particle - One feature you could use is the size of the support (i.e. n. of pixels) of the part of the image that is different from the background.

Quantiles of invariants - You did well with the quantiles of the image grayscale values. You could try and compute quantiles of the images after applying blurring and the differential invariants. You may quickly end up with thousands of features this way. You will need to select the best ones.

6 Feature selection

You have developed a method based on Fisher linear discriminant analysis. There are other methods. See for example Francois Fleuret's method based on conditional mutual information:

```
Fast Binary Feature Selection with Conditional Mutual Information
Francois Fleuret
Journal of Machine Learning Research (JMLR), 2004, 5, 15311555
http://cvlab.epfl.ch/~fleuret/papers/fleuret-jmlr2004.pdf
```