Convolution and Local Image Structure

Report for Lab Exercise 2 for the Computer Vision Course

M. Pfundstein (10452397)
T.M. Meijers (10647023)
University of Amsterdam
The Netherlands

November 24, 2014

Introduction

Written exercises

This report contains all the written answers to the theory questions in ***convolution and Local Image Structure****: Lab Exercise 2*

Programming exercises

For the programming exercises the choice was made to explain them fully by comments. This means that besides this subsection, no other part of this report will refer to the programming exercises. The code is commented well enough so that every step is clear, and the structure of the code is well organized. The main file (*main.m*) is organized into sections which correspond to the structure in the assignment. Every section refers to which helper functions were written to solve the exercise.

1 Convolution

1.1

$f = \{1, 2, 1\}$
$g = \{0, 0, 0, 0, 1, 1, 1, 1, 1\}$

Sample calculation of $(f * g)(-1)$.

$$\begin{aligned}(f * g)(-1) &= \sum_{j=-1}^{1} f(j)g(-1-j) \\ &= f(-1)g(-1-(-1)) + f(0)g(-1-0) + f(1)g(-1-1) \\ &= f(-1)g(0) + f(0)g(-1) + f(1)g(-2) \\ &= 1 \cdot 1 + 2 \cdot 0 + 1 \cdot 0 \\ &= 1\end{aligned}$$

If we do this procedure analog for $(f * g)(0)$ and $(f * g)(1)$, we get:

$f * g = \{1, 3, 4\}$

We decided to handle the edge cases by cutting them off. Hence a $1 \times n$ input image f, will produce a $1 \times n$ output image $(f * g)$.

1.2

$f = \{0, 0, 0, 0, 1, 1, 1, 1, 1\}$
$g = \{1, 2, 1\}$

Applying Convolution therefore gives us the result:

$f * g = \{0, 0, 0, 1, 3, 4, 4, 4, 3\}$

In this case we handled the edge cases as follows: When an index of g was out of bounds, we simply used 0 ((e.g. $g(-5) = 0$).

1.3

$f = \{0, 0, 0, 0, 1, 1, 1, 1, 1\}$
$g = \{-1, 1\}$

Applying Convolution gives us:

$f * g = \{0, 0, 0, -1, 0, 0, 0, 0, 1\}$.

We used the method as in 1.1.2 to handle border cases.

1.4

Proof of communatitivity of convolution:

$$\begin{aligned}(f*g)(i) &= \sum_{j=-\infty}^{\infty} f(i-j)g(j) \\ &\quad j \leftarrow i-j(substitution) \\ &= \sum_{j=-\infty}^{\infty} f(i-(i-j))g(i-j) \\ &= \sum_{j=-\infty}^{\infty} f(i+(-i)+j))g(i-j) \\ &= \sum_{j=-\infty}^{\infty} f(j)g(i-j) \\ &= \sum_{j=-\infty}^{\infty} g(i-j)f(j) \\ &= (g*f)(i)\end{aligned}$$

1.5

Proof of associativity of convolution:

$$\begin{aligned}((f*g)*h)(i) &= \sum_{j=-\infty}^{\infty} [(f*g)(j)]h(i-j) \\ &= \sum_{j=-\infty}^{\infty} [\sum_{k=-\infty}^{\infty} f(k)g(j-k)]h(i-j) \\ &= \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} f(k)g(j-k)h(i-j) \\ &= \sum_{k=-\infty}^{\infty} f(k) \sum_{j=-\infty}^{\infty} g(j-k)h(i-j) \\ &\quad j \leftarrow j+k(substution) \\ &= \sum_{k=-\infty}^{\infty} f(k) \sum_{j=-\infty}^{\infty} g(j+k-k)h(i-(j+k)) \\ &= \sum_{k=-\infty}^{\infty} f(k) \sum_{j=-\infty}^{\infty} g(j)h(i-j-k)) \\ &= \sum_{k=-\infty}^{\infty} f(k) \sum_{j=-\infty}^{\infty} g(j)h(i-k-j)) \\ &= \sum_{k=-\infty}^{\infty} f(k)(g*h)(i-k) \\ &= (f*(g*h))(i)\end{aligned}$$

1.6

Identity operation:

$g = \{\underline{1}\}$

1.7

Multiplying intensity by 3:

$g = \{\underline{3}\}$

1.8

Translating by $[-3 \;\; 1]^T$:

$g = \begin{Bmatrix} 0 & 0 & 0 & \underline{0} \\ 1 & 0 & 0 & 0 \end{Bmatrix}$

1.9

With a rotation, you have to perform a different operation based on which point you are rotating in the image. Convolution can only do the same operation independant of the point it is transforming. Convulation cannot rotate or mirror.

1.10

Taking the average in a 3 X 3 neighbourhood:

$g = \frac{1}{9} \begin{Bmatrix} 1 & 1 & 1 \\ 1 & \underline{1} & 1 \\ 1 & 1 & 1 \end{Bmatrix}$

1.11

Taking the median in a 3 X 3 neighborhood:

Not possible through convulsion, since the median has a different value for each different point. Hence one cannot find a specific convolution kernel for this problem.

1.12

Computing the minimum value in a 5 X 5 neighborhood:

Again, this can not be done and the reason wy is the same as with 1.1.11. The convolution kernel is specific for each point and not one kernel for an image.

1.13

Performing motion blur, as if the camera moved horizontally to the right over 5 pixels during recording:

$g = \frac{1}{6} \begin{Bmatrix} 1 & 1 & 1 & 1 & 1 & \underline{1} \end{Bmatrix}$

1.14

Gaussian with standard deviation of 3:

$\frac{1}{2 \cdot 3^2 \pi} \cdot e^{-\frac{x^2 y^2}{18}} = \frac{1}{18\pi} \cdot e^{-\frac{x^2 y^2}{18}}$

Hence if you have a $M \times N$ image f, then you can calculate the convutation matrix from this by plucking in the (x, y) indicies of the matrix into the equation above.

1.15

Unsharp masking of an image:

This is possible, one would take a certain blur convolution kernel (preferable guassian) and then subtract this from a convolution matrix with a 2 in its origin (two times the original pixel value in the origin), and 0's around it so it has the same dimension . This would result in a convolution matrix that applies unsharp masking.

1.16

Taking an approximation of the derivative in the x-direction:

So, $g = \frac{1}{2}\{1 \ \underline{0} \ -1\}$ is the right one since our x increases if we move to the right.

1.17

Taking the second derivative in the x-direction:

To obtain the convolution kernel we have to multiply the convolution kernel with itself:
$g = \frac{1}{2}\{1 \ \underline{0} \ -1\}$
$g * g = \frac{1}{4}\{1 \ 0 \ -\underline{2} \ 0 \ 1\}$

1.18

Zooming in on an image to 4 times the size, and performing bilinear interpolation to obtain the intermediate points:

This is not possible since this is again dependant on the point we are transforming by convolution. So with zooming we need to "stretch" in multiple (different) directions, this is not possible through a single convolution kernel.

1.19

Thresholding an image so that all pixels with an intensity lighter than 0.5 become 1, and all others 0:

It is impossible to give a single convolution kernel that does a transformation based on values of the point it is transforming. Convolution can't detect difference between different points in the same image.

Extra

We applied three filters (See Fig 1).

1.20

the Gaussian has smooth derivatives of arbitrary order
the Gaussian has smooth derivatives of arbitrary order
the Gaussian has smooth derivatives of arbitrary order
the Gaussian has smooth derivatives of arbitrary order
the Gaussian has smooth derivatives of arbitrary order
Now we're convinced. (Joke, see rest of answer).
As found on `http://www.gris.informatik.tu-darmstadt.de/~akuijper/course/TUD11/lecture3.`

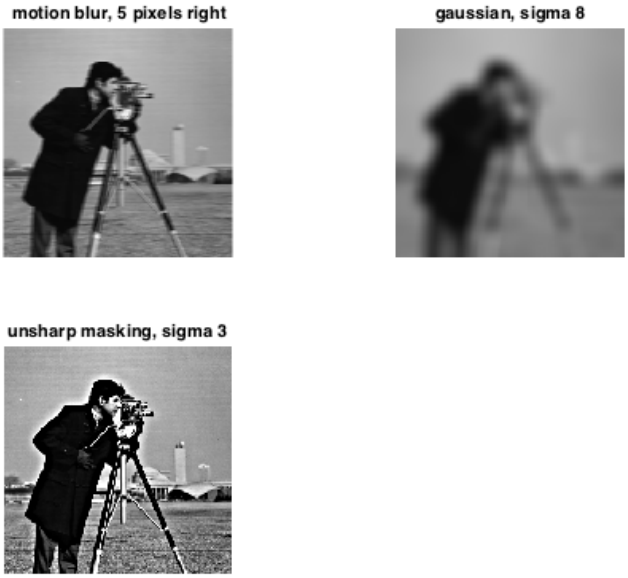

Figure 1: 3 filters applied

`pdf` on slide 6, the Gaussian itself is always part of its derivative. The n-th derivative of the gaussian is the gaussian itself multiplied with a polynomial of n-th order (see slide 7 of above link for how those polynomials look like). This pattern suggests indeed that every n-th order derivative of the gaussian function is smooth (differentiable).

1.21

The short answer: So that we can combine it with other convolutions. Convolutions are a general operation on images and taking the derivative should also be one as it is involved in many operations as it enables us to find certain properties in images.

1.22

If we write out the order of operations (taking the derivative of an image, then apply a gaussian filter), we get the following formula: $G * (\delta * f)$. Because of associativity we can rewrite this to $G * \delta * f$. Because convulution commutes we can again rewrite this to $\delta * G * f$. Applying associativity again we get $(\delta * G) * f$. This represents *better* our notion of how we apply a gaussian derivative kernel to an image f.

1.23

In two dimensions we can rewrite the formula to $G(x_1, x_2) = \frac{1}{2\pi\sigma^2} e^{\frac{-x_1^2 - x_2^2}{2\sigma^2}}$. We will now take two derivates. One with respect to x_1 and one with respect to x_2 to show that the identity holds.

Proof for x_1

$$\begin{aligned}
\frac{\delta G_\sigma(x_1, x_2)}{\delta x_1} &= \frac{1}{2\pi\sigma^2} \frac{\delta}{\delta x_1} e^{\frac{-x_1^2 - x_2^2}{2\sigma^2}} \\
&= \frac{1}{2\pi\sigma^2} e^{\frac{-x_1^2 - x_2^2}{2\sigma^2}} \cdot \frac{-x_1}{\sigma^2} \\
&= \frac{-x_1}{\sigma^2} G_\sigma(x_1, x_2)
\end{aligned}$$

Proof for x_2

$$\begin{aligned}\frac{\delta G_\sigma(x_1,x_2)}{\delta x_2} &= \frac{1}{2\pi\sigma^2}\frac{\delta}{\delta x_2}e^{\frac{-x_1^2-x_2^2}{2\sigma^2}} \\ &= \frac{1}{2\pi\sigma^2}e^{\frac{-x_1^2-x_2^2}{2\sigma^2}}\cdot\frac{-x_2}{\sigma^2} \\ &= \frac{-x_2}{\sigma^2}G_\sigma(x_1,x_2)\end{aligned}$$

If we now rewrite this back into the vectorized form we clearly get the desired identity.

1.24

To prove: $\frac{\delta^2 G_\sigma}{\delta x^2}(\vec{x}) = (\frac{\vec{x}^2}{\sigma^4} - \frac{1}{\sigma^2})G_\sigma(\vec{x})$
For first derivative see previous answer (1.2.4)

$$\begin{aligned}\frac{\delta^2 G_\sigma}{\delta x_1^2}(x_1,x_2) &= \frac{\delta}{\delta x_1}\frac{-x_1}{2\pi\sigma^4}\cdot e^{\frac{-x_1^2-x_2^2}{2\sigma^2}} \\ &= \frac{-1}{2\pi\sigma^4}\cdot e^{\frac{-x_1^2-x_2^2}{2\sigma^2}} + \frac{-x_1}{2\pi\sigma^4}\cdot e^{\frac{-x_1^2-x_2^2}{2\sigma^2}}\cdot\frac{2x_1}{2\sigma^2} \\ &= e^{\frac{-x_1^2-x_2^2}{2\sigma^2}}(-\frac{1}{2\pi\sigma^4} - \frac{x_1}{2\pi\sigma^4}\cdot-\frac{-x_1}{\sigma^2}) \\ &= e^{\frac{-x_1^2-x_2^2}{2\sigma^2}}(-\frac{1}{2\pi\sigma^4} + \frac{x_1^2}{2\pi\sigma^6}) \\ &= \frac{1}{2\pi\sigma^2}\cdot e^{\frac{-x_1^2-x_2^2}{2\sigma^2}}(-\frac{1}{\sigma^2} + \frac{x^2}{\sigma^4}) \\ &= (\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2})G_\sigma(x_1,x_2)\end{aligned}$$

This process is the same for all elements in $\vec{x}$, so for 2D this will give the same second order derivative with respect to x_2. Again we can rewrite this back into the vectorized form so we get the desired identity:

$$(\frac{\vec{x}}{\sigma^4} - \frac{1}{\sigma^2})G_\sigma(\vec{x})$$

Derive f_{xy}

$$\begin{aligned}\frac{\delta G_\sigma(x_1,x_2)}{\delta x_1 x_2} &= \frac{\delta}{\delta x_1}(\frac{-x_2}{\sigma^2}\frac{1}{2\pi\sigma^2}e^{\frac{-x^2-y^2}{2\sigma^2}}) \\ &= \frac{-x_2}{\sigma^2}\frac{1}{2\pi\sigma^2}\frac{\delta}{\delta x_1}e^{\frac{-x^2-y^2}{2\sigma^2}} \\ &= \frac{-x_2}{\sigma^2}\frac{1}{2\pi\sigma^2}e^{\frac{-x^2-y^2}{2\sigma^2}}\cdot\frac{-x_1}{\sigma^2} \\ &= \frac{x_1x_2}{\sigma^4}G_\sigma(x_1,x_2)\end{aligned}$$

1.25

...

1.26

Show that Gaussian kernel is separable. $G_{\sigma,x}$ denotes 1-D kernel in x direction. $G_{\sigma,y}$ denotes 1-D kernel in y direction. G_σ denotes standard 2-D gaussian.

$$\begin{aligned} G_{\sigma,x} &= \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{x^2}{2\sigma^2}) \\ G_{\sigma,y} &= \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{y^2}{2\sigma^2}) \\ G_{\sigma,y}G_{\sigma,x} &= \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{x^2}{2\sigma^2}) \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{y^2}{2\sigma^2}) \\ &= \frac{1}{2\pi\sigma^2} exp(\frac{-x^2-y^2}{2\sigma^2}) \\ &= G_\sigma \end{aligned}$$

1.27

From 1.26, we know that the Gaussian kernel is separable. One property from the Gaussian kernel is that the derivative always includes the Gaussian (See 1.20). We can conclude that:

$$\frac{\delta(\frac{\delta G_\sigma(x)}{\delta x})}{\delta y}(y) = \frac{\delta G_\sigma(x,y)}{\delta x \delta y}$$

This holds also for if $y = x$ or $x = y$.

2 Implementation of Gaussian Derivatives

This assignmenet is fully done in Matlab.

3 The Canny Edge Detector

3.1

Assume

$$f(x,y) = A\ \ sin(Vx) + B\ \ cos(Wy)$$

Calculate analytically the derivates $f_x, f_y, f_{xx}, f_{yy}, f_{xy}$. The derivates are:

1. f_x:

$$f_x(x,y) = VA\ \ cos(Vx)$$

2. f_y:

$$f_y(x,y) = -WB\ \ sin(Wy)$$

3. f_{xx}:

$$f_{xx}(x,y) = -V^2A\ \ sin(Vx)$$

2. f_{yy}:

$$f_{yy}(x,y) = -W^2B\ \ cos(Wy)$$

3. f_{xy}:

$$f_{xx}(x,y) = 0$$

3.2

See Matlab, Section 3.2 for the sampled images F_x and F_y.

3.3

See Matlab, Section 3.3

3.4

See Matlab, Section 3.4

3.5

See Matlab, Section 3.5

3.6

Analytically derive f_w and f_{ww}.
Consider gradient vector ∇f at arbitrary location $\vec{a}$.
The unit vector e_w is $\frac{1}{|\nabla f|}\nabla f = \frac{1}{|\nabla f|}\begin{bmatrix} f_x & f_y \end{bmatrix}^T$.
The unit vector e_w is defined so that $e_v \cdot e_w = 0$. Hence $e_v = \frac{1}{|\nabla f|}\begin{bmatrix} f_y & -f_x \end{bmatrix}^T$.
The span $\begin{bmatrix} e_v & e_w \end{bmatrix}$ describes our new coordinate system for the gradient gauge.
To rotate ∇f into the new coordinate system, we have to normalize it and apply the rotation matrix R that is defined as:

$$R = \begin{bmatrix} f_y & -f_x \\ f_x & f_y \end{bmatrix}$$

This leads to

$$\begin{aligned}
\begin{bmatrix} f_y & -f_x \\ f_x & f_y \end{bmatrix} \frac{1}{|\nabla f|} \nabla f &= \frac{1}{|\nabla f|} \begin{bmatrix} f_y & -f_x \\ f_x & f_y \end{bmatrix} \begin{bmatrix} f_x \\ f_y \end{bmatrix} \\
&= \frac{1}{|\nabla f|} \begin{bmatrix} fyfx - fxfy \\ fxfx + fyfy \end{bmatrix} \\
&= \frac{1}{\sqrt{(f_x)^2 + (f_y)^2}} \begin{bmatrix} 0 \\ (f_x)^2 + (f_y)^2 \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ \frac{(f_x)^2 + (f_y)^2}{\sqrt{(f_x)^2 + (f_y)^2}} \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ ((f_x)^2 + (f_y)^2) * ((f_x)^2 + (f_y)^2)^{-\frac{1}{2}} \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ ((f_x)^2 + (f_y)^2)^{\frac{1}{2}} \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ \sqrt{(f_x)^2 + (f_y)^2} \end{bmatrix} \\
&= \begin{bmatrix} f_v \\ f_w \end{bmatrix}
\end{aligned}$$

If we rotate an arbitrary vector $\vec{x}$ into the new coordinate system $\begin{bmatrix} e_v e_w \end{bmatrix}$ we get $\vec{v} = R\vec{x}$. Because R is an orthogornal matrix, we can now that $\vec{x} = R^T \vec{v}$.

It is now possible to rewrite xHx^T into $(R^T\vec{v})^T H (R^T\vec{v})$.

$$(R^T\vec{v})^T H (R^T\vec{v}) = \vec{v}^T (RHR^T)\vec{v} =$$

So our new Hessian will be RHR^T. (Because we are only interested in $f_w w$ for this assignment I will omit some calculations.

$$\begin{aligned}
\begin{bmatrix} f_y & -f_x \\ f_x & f_y \end{bmatrix} \begin{bmatrix} f_{xx} & f_{xy} \\ f_{xy} & f_{yy} \end{bmatrix} \begin{bmatrix} f_y & f_x \\ -f_x & f_y \end{bmatrix} &= \begin{bmatrix} f_y & -f_x \\ f_x & f_y \end{bmatrix} \begin{bmatrix} f_{xx}f_y - f_{xy}f_x & f_{xx}f_x + f_{xy}f_y \\ f_{xy}f_y - f_{yy}f_x & f_{xy}f_x + f_{yy}f_y \end{bmatrix} \\
&= \begin{bmatrix} .. & .. \\ .. & f_x(f_{xx}f_x + f_{xy}f_y) + f_y(f_{xy}f_x + f_{yy}f_y) \end{bmatrix} \\
&= \begin{bmatrix} .. & .. \\ .. & f_x^2 f_{xx} + 2f_{xx}f_x f_y + f_y^2 \end{bmatrix} \\
&= \begin{bmatrix} f_{vv} & f_{vw} \\ f_{vw} & f_{ww} \end{bmatrix}
\end{aligned}$$

3.7

See Matlab, Section 3.7

3.8

See Matlab, Section 3.8

3.9

See Matlab, Section 3.9