Lab 2 - theory

Beeldverwerken

April 25, 2018

Patrick Spaans (11268476)
Bram Otten (10992456)

Check out `lab2_Spaans_Otten.mlx` and the functions it references (which should all be in the main folder of the zip this PDF came in) for the MATLAB part of this assignment.

1 Convolution

Convolution Examples and Implementation

1.1

1D convolution $f * g = \{1\ \underline{2}\ 1\} * \{0\ 0\ 0\ 0\ \underline{1}\ 1\ 1\ 1\ 1\} = \{0\ 0\ 0\ 1\ 3\ 4\ 4\ 4\ 3\ 1\}$.

We handle border cases by considering pixel values outside of borders to be 0. (Here and everywhere else until we specify some other strategy.)

1.2

1D convolution $f * g = \{0\ 0\ 0\ 0\ \underline{1}\ 1\ 1\ 1\ 1\} * \{1\ \underline{2}\ 1\} = \{0\ 0\ 0\ 1\ 3\ 4\ 4\ 4\ 3\ 1\}$.

1.3

1D convolution $f * g = \{0\ 0\ 0\ 0\ \underline{1}\ 1\ 1\ 1\ 1\} * \{-1\ \underline{1}\} = \begin{Bmatrix}0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1\end{Bmatrix}$.

1.4

Proof that a convolutions are commutative:

The definition of the convolution is: $(f * g)(x) = \sum_{y\in E} f(x-y)g(y)$

If convolution is commutative, $(f * g)(x)$ should be equal to $(g * f)(x)$, which also means that $\sum_{y\in E} f(x-y)g(y)$ should be equal to $\sum_{y'\in E} f(x)g(x-y')$

By defining y as $x - y'$, and defining y' as $x - y$, we can prove that the two definitions are the same.

By substituting y' for y in $(f * g)(x)$, we transform $\sum_{y\in E} f(x-y)g(y)$ into $\sum_{y'\in E} f(y')g(x-y)$.

By substituting y for y' in $(g * x)(x)$, we transform $\sum_{y'\in E} f(y')g(x-y')$ into $\sum_{y\in E} f(x-y)g(y)$.

These substitutions prove that $(f * g)(x)$ is $(g * x)(x)$, and that $(g * x)(x)$ is $(f * g)(x)$, which proves that the convolution is commutative.

1.5

Proof that convolutions are associative:

The definition of the convolution is: $(f * g)(x) = \sum_{y\in E} f(x-y)g(y)$

If convolutions are associative, $(f * (g * h))(x)$ should be equal to $((f * g) * h)(x)$

$((f * g) * h)(x)$ is equal to $\sum_{z=0}^{E}(f * g)(z)h(x-z) = \sum_{z=0}^{E}\sum_{y=z}^{E} f(y)g(z-y)h(x-z)$

Now, $\sum_{z=0}^{E}$ is the sum from the values 0 to E, with $\sum_{y=z}^{E}$ being the sum from the values 0 to z, which is used for every value of z in E. We want to remove z from the equation, so the sums should be flipped. This leads to:

$\sum_{y=0}^{E}\sum_{z=y}^{E} f(y)g(z-y)h(x-z)$, which is exactly the same, since summations are associative.

Instead of letting z start at y and go on until E, we can let z start at 0 and go on until $E - y$, in order to keep everything else the same, we need to replace every current instance of z with $(z + y)$ to match this change, and keep the total the same.

$\sum_{y=0}^{E}\sum_{z=0}^{E-y} f(y)g(z)h(x-z-y)$

$\sum_{y=0}^{E}\sum_{z=0} g(z)h(x-z-y)$ is the a convolution itself, and can by reversing the definition be changed into $(g*h)(x-y)$, which leads to:

$\sum_{y=0}^{E} f(y)(g*h)(x-y)$, which itself is equal to $(f*(g*h))(x)$. We now have transformed $(f*(g*h))(x)$ into $((f*g)*h)(x)$

1.6

The convolution kernel for the identity operation is $g = \{\underline{1}\}$.

1.7

The convolution kernel for multiplying intensity by 3 is $g = \{\underline{3}\}$.

1.8

The convolution kernel for translating over $\begin{pmatrix} -3 \\ 1 \end{pmatrix}$ is $g = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \underline{0} \end{Bmatrix}$

1.9

Rotating an image can not be done by convolution.

1.10

The average in a 3×3 neigbourhood is given by $g = \frac{1}{9}\begin{Bmatrix} 1 & 1 & 1 \\ 1 & \underline{1} & 1 \\ 1 & 1 & 1 \end{Bmatrix}$.

1.11

The median of a (larger than 1×1) neigbourhood can not be found using a convolution, as it requires sorting the values in that neigbourhood.

1.12

The same goes for the minimum, we would need all values in the neighbourhood before we could find out the minimum value.

1.13

The convolution of a 5-pixel motion blur to the right is: $g = \frac{1}{5}\begin{Bmatrix} 1 & 1 & 1 & 1 & \underline{1} \end{Bmatrix}$

1.14

Gaussian blur with a 3-pixel standard deviation is convolution $g = \frac{1}{18\pi}\exp(-\left(\frac{x^2+y^2}{18}\right))$. Which is something like an underlined reference value and ever smaller values for ever larger distances from that reference value.

1.15

Unsharp masking of an image f is a normalized $f - \frac{1}{18\pi}\exp(-\left(\frac{x^2+y^2}{18}\right))$. (Subtracting a blur, in this example a Guassian with SD=3. Normalization restores intensity.)

1.16

The convolution of the derivative in the x direction is given by $g = \frac{1}{2}\begin{Bmatrix} 1 & \underline{0} & -1 \end{Bmatrix}$.

1.17

The convolution of the second derivative in the x direction is given by

$$g = \frac{1}{2}\begin{Bmatrix} 1 & 0 & -1 \end{Bmatrix} * \frac{1}{2}\begin{Bmatrix} 1 & 0 & -1 \end{Bmatrix} = \frac{1}{4}\begin{Bmatrix} 1 & 0 & -2 & 0 & 1 \end{Bmatrix}$$

1.18

Zooming in with factor 4 and interpolating can't be done with a convolution filter. We need a center to expand to a neighbourhood of 4×4 (that's the zoom), but there can't be such a center because 4 is an even number.

1.19

The convolution for thresholding an image to make values >.5 1 and others 0 is $g = \{\underline{> .5}\}$.

*

In the MATLAB script.

Gaussian Convolution

1.20

A function is smooth if it has derivatives everywhere for all orders. Our Gaussian has smooth derivatives because its derivatives are just itself multiplied by a combination of x's and σ's.

$$\begin{aligned}
G_\sigma(x) &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \\
\frac{\partial G_\sigma(x)}{\partial x} &= G_\sigma(x) \cdot \frac{\partial\left(-\frac{x^2}{2\sigma^2}\right)}{\partial x} \\
&= G_\sigma(x) \cdot -\frac{x}{\sigma^2} \\
\frac{\partial^2 G_\sigma(x)}{\partial x^2} &= G_\sigma(x) \cdot \frac{\partial^2\left(-\frac{x^2}{2\sigma^2}\right)}{\partial x^2} \\
&= G_\sigma(x) \cdot (\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2})
\end{aligned}$$

And so on for arbitrary orders.

1.21

Taking a derivative should be a convolution because its outcome should depend on pixels surrounding the center one being looked at. The derivative for an input should be positive if values go up in the direction of that derivative, and vice versa. Convolutions can be derived.

1.22

The derivatives of the convolution of an image intensity function with a Gaussian kernel (i.e., of a blurred image) can be calculated by convolving the original image with the derivative of the Gaussian, instead. The result is therefore often called the Gaussian derivative of the image.

This depends on associativity because we go from 'derivative * (blur * image)' to '(derivative * blur) * image.'

This depens on commutativity because go from 'derivative * blur' to 'blur * derivative'.

1.23

The derivation above (in section 1.20) ($\frac{\partial G_\sigma(x)}{\partial x} = G_\sigma(x) \cdot -\frac{x}{\sigma^2}$) is the is 1D derivative in the x direction. This derivative holds in 2D because we can take x and y (that's $\frac{\partial G_\sigma(y)}{\partial y} = G_\sigma(y) \cdot -\frac{y}{\sigma^2}$) derivatives seperately and convolve them with each other for the 2D xy derivative ($\frac{\partial G_\sigma(x,y)}{\partial(x,y)}$).

1.24

Look at 1.20 again for $\frac{\partial^2 G_\sigma(x)}{\partial x^2} = G_\sigma(x) \cdot (\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2})$. (Can take it for y too by just replacing x with y.)

1.25

By smoothing an image with G_{σ_1}, you get an Gaussian smoothed image, which is an image itself. Since it is just an normal image that we have created using Guassian Smoothing, we can smooth it again with G_{σ_2} without problems, and this will also be an Guassian smoothed image. The resulting smoothing scale will be $\sigma_1 + \sigma_2$.

1.26

2D Gaussian G_σ is an M $\times$ N matrix, which can be created by using matrix multiplication on two vectors, one vector being of size M*1, the other being of size 1×N. These vectors are both 1 Dimensional vectors, with the second vector being transposed, and they are the x-smoothing and y-smoothing vector respectively. This is confirmed by the formula of $G_\sigma(x, y)$, which is just $G_\sigma(x)_\sigma(y)$. The scale of x-smoothing is M, while the scale of y-smoothing is N.

1.27

Derivatives of the 2D Gaussian are seperable as well because:

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

2 Implementation of Gaussian Derivatives

There are some theoretical questions in this section too, these are in the MATLAB script. (Spoiler: $O(1)$, pixels.)

3 The Canny Edge Detector

Same goes for this section.