Gideon Mooijen, 10686290
Lucas van Berkel, 10747958

1 Convolution

1.1

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

	0	0	0	0	$\underline{1}$	1	1	1	1	
				1	$\underline{2}$	1				
	0	0	0	0	2	2	2	2	2	
0	0	0	0	1	1	1	1	1		
		0	0	0	0	1	1	1	1	1
0	0	0	0	1	$\underline{3}$	4	4	4	3	1

y = { 0,0,0,0,1,$\underline{3}$,4,4,4,3,1 }

This results in: $y = [0, 0, 0, 1, \underline{3}, 4, 4, 4, 3]$. Every empty cell is regarded a 0. The bordercases are disregarded when the center of the kernel is not 'in the frame', so they start when the underlined character is the first element in the second row. They end when the underlined character is the last element in the second row.

1.2

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

				1	$\underline{2}$	1				
	0	0	0	0	$\underline{1}$	1	1	1	1	
				1	2	1				
			0	0	0					
		0	0	0						
	0	0	0							
0	0	0								
					1	2	1			
						1	2	1		
							1	2	1	
								1	2	1
0	0	0	0	1	$\underline{3}$	4	4	4	3	1

y = { 0,0,0,0,1,$\underline{3}$,4,4,4,3,1 }

1.3

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

				$\underline{1}$	-1				
0	0	0	0	$\underline{1}$	1	1	1	1	
				1	-1				
			0	0					
		0	0						
	0	0							
0	0								
					1	-1			
						1	-1		
							1	-1	
								1	-1
0	0	0	0	1	$\underline{0}$	0	0	0	-1

y = { 0,0,0,0,1,0,0,0,0,-1 }

1.4

Prove the communativity of convolution.
Standard convolution formula:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)dy \tag{1}$$

$z = x - y$
So: $y = x - z$

$$\int_{-\infty}^{\infty} f(x-z)g(z)dz = (g * f)(x) \tag{2}$$

Convolution is commutative.

1.5

Prove the associativity of convolution.

$$\begin{aligned} ((f*g)*z)(u) &= \textstyle\int_{-\infty}^{\infty}(f*g)(x)z(u-x)dx \\ &= \textstyle\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(y)g(x-y)z(u-x)dydx \\ &= \textstyle\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(y)g(x)z(u-(x+y))dydx \\ &= \textstyle\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(y)g(x)z((u-y)-x)dydx \\ &= \textstyle\int_{-\infty}^{\infty} f(y)(g*z)(u-y)dx \\ &= (f*(g*z))(u) \end{aligned}$$

Convolution is associative

1.6

Identity operation is: $\begin{bmatrix} 0 & 0 & 0 \\ 0 & \underline{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}$

1.7

Intensity operation is: $\begin{bmatrix} 0 & 0 & 0 \\ 0 & \underline{3} & 0 \\ 0 & 0 & 0 \end{bmatrix}$

1.8

Translate operation over vector $(-3,1)^T$: $\begin{bmatrix} 0 & 0 & 0 & \underline{0} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$

1.9

For a linear transformation to be written as a convolution, the transformation must be invariant linear, which rotation is not.

1.10

Mean operation is: $\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & \underline{1} & 1\\ 1 & 1 & 1\end{bmatrix}$

1.11

Calculating the median of a set of pixels means using variables, which a convolution kernel is not capable off.

1.12

Same as calculating the median, calculating the minimum is not possible using convolution kernels.

1.13

5 pixels horizontal motion to the right operation: $\frac{1}{5}\begin{bmatrix}1 & 1 & 1 & 1 & \underline{1}\end{bmatrix}$

1.14

Optical blur operation: $\frac{1}{8}\begin{bmatrix}0 & 1 & 0\\ 1 & \underline{4} & 1\\ 0 & 1 & 0\end{bmatrix}$

1.15

Unsharp masking operation: $\begin{bmatrix}0 & 0 & 0\\ 0 & \underline{1} & 0\\ 0 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 0\\ 0 & \underline{1} & 0\\ 0 & 0 & 0\end{bmatrix} - \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & \underline{4} & 2\\ 1 & 2 & 1\end{bmatrix} = \frac{1}{8}\begin{bmatrix}0 & -1 & 0\\ -1 & \underline{12} & -1\\ 0 & -1 & 0\end{bmatrix}$

1.16

Approximate derivative operation: $\frac{1}{2}\begin{bmatrix}-1 & \underline{0} & 1\end{bmatrix}$

1.17

A single convolution kernel would not be able to approximate the second derivative, because it would mean that the first derivative is already calculated.

1.18

Not possible, variables would be saved, which is not possible with convolution kernels.

1.19

Not possible, it would mean that the kernel would perform differently when the value passes a certain threshold, so the kernel would check that value, which can not.

Bonusquestion:
View the delivered mat-lab scripts.

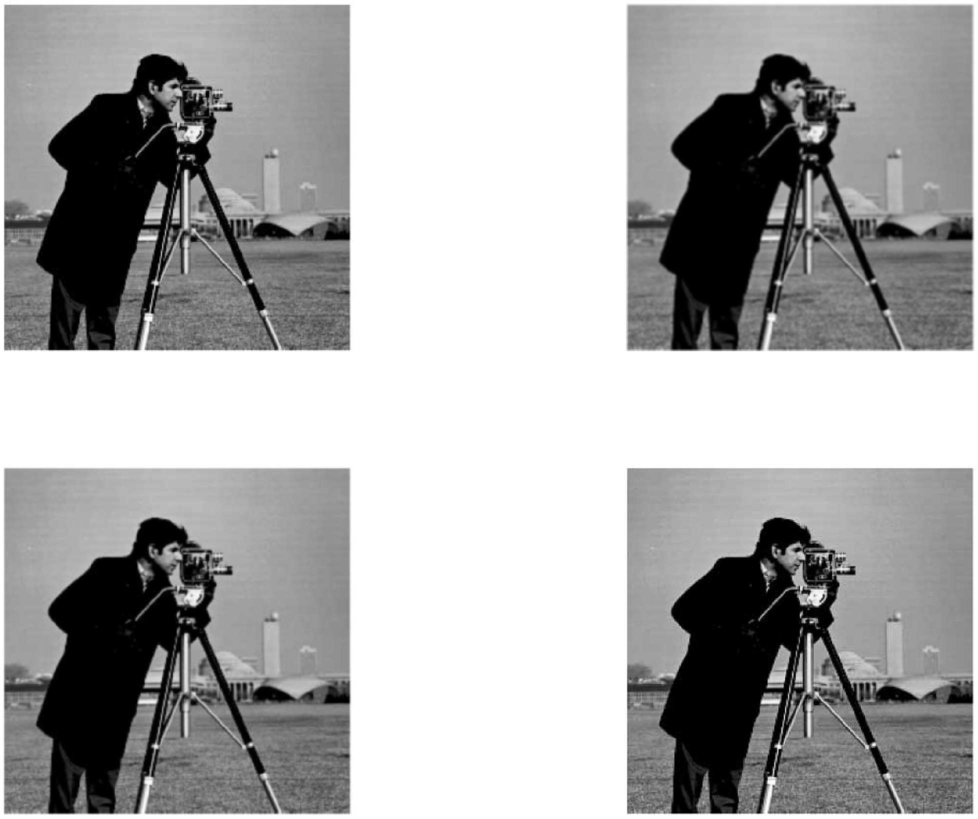

Figure 1: Original, mean, blur and unsharp mask

1.20

Since a Gaussian in 1D is like blurring a image in one direction, where every new pixelvalue is dependent of its neighbours. Because of that dependence, the derivative of the Gaussian would be smooth, without unexpected spikes, or any derivative of the Gaussian of an arbitrary order for that matter.

1.21

In image processing, the function over which the derivatives are calculated, are not perfect mathematical functions, but sets of pixels. To obtain the derivative of a certain pixel, we have to look at the surrounding pixels in order to calculate the derivative. With a convolution, the derivative of the pixel can be calculated with the surrounding pixels. That's why 'Taking a derivative' should be a convolution.

1.22

The derivative of the convolution of an image intensity function with a Gaussian kernel is represented by the following function: $\partial(f * G_\sigma)$. By replacing taking the derivative of the convolution by the differentiation kernel, the new calculation is one of two convolutions. Because the convolution is associative and commutative, the brackets may be swapped and the order changed, which results in the following formula:
$\partial(f * G_\sigma) = D * (f * G_\sigma) = f * (D * G_\sigma) = f * (\partial G_\sigma)$
The last result is indeed the convolution between the original image with the derivative of the Gaussian.

1.23

$G_\sigma(x) = \frac{1}{(\sqrt{2\pi}\sigma)^2} e^{-\frac{\|x\|^2}{2\sigma^2}}$
$u = -\frac{x^2}{2\sigma^2}$
$G_\sigma(x) = \frac{1}{(\sqrt{2\pi}d)^2} e^u$
$\partial G_\sigma(x) = \frac{1}{(\sqrt{2\pi}\sigma)^d} e^u \cdot u'$
$u' = -\frac{x}{\sigma^2}$
$\frac{\partial G_\sigma(x)}{\partial x} = \frac{1}{(\sqrt{2\pi}\sigma)^2} e^{-\frac{\|x\|^2}{2\sigma^2}} \cdot -\frac{x}{\sigma^2} = -\frac{x}{\sigma^2} \cdot G_\sigma(x)$

1.24

$x = -\frac{x}{\sigma^2} \cdot G_\sigma(x)$
$x_1 = -\frac{x}{\sigma^2}$
$x_2 = G_\sigma(x)$
$\partial x_1 = -\frac{1}{\sigma^2}$
$\partial x_2 = -\frac{x}{\sigma^2} G_\sigma(x)$
$\frac{\partial -\frac{x}{\sigma^2} \cdot G_\sigma(x)}{\partial x} = x_1 \cdot \partial x_2 + x_2 \partial x_1 = -\frac{x}{\sigma^2} \cdot -\frac{x}{\sigma^2} G_\sigma(x) + G_\sigma(x) \cdot -\frac{1}{\sigma^2} = (\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}) \cdot G_\sigma(x)$

1.25

$G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}$
With smoothing a image with a Gaussian, you convolve the image with the Gaussian. The second Gaussian is then convolved with the smoothed image. Since convolution is associative, the two Gaussian may be first convolved before convolving with the image. Which results in the following procedure.

$$\begin{aligned}
(G_1 * G_1)(x) &= \int_{-\infty}^{\infty} G_1(x-u) G_1(u) du \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(x-u)^2} e^{-\frac{1}{2}u^2} du \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-\frac{1}{2}((x-u)^2+u^2)} du \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(x^2-2xu+2u^2)} du \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-\frac{1}{2}((\sqrt{2}u-\frac{x}{\sqrt{2}})^2+(\frac{x}{\sqrt{2}})^2)} du \\
&= \frac{1}{2\pi} (\int_{-\infty}^{\infty} e^{-\frac{1}{2}(\sqrt{2}u-\frac{x}{\sqrt{2}})^2} du) e^{-\frac{1}{2}(\frac{x}{\sqrt{2}})^2} \\
&= \frac{1}{2\pi} (\int_{-\infty}^{\infty} e^{-\frac{1}{2}v^2} (dv\sqrt{2})) e^{-\frac{1}{2}(\frac{x}{\sqrt{2}})^2} \\
&= \frac{1}{2\pi} \frac{\sqrt{2\pi}}{\sqrt{2}} e^{-\frac{1}{2}(\frac{x}{\sqrt{2}})^2} \\
&= \frac{1}{\sqrt{4\pi}} e^{-\frac{1}{2}(\frac{x}{\sqrt{2}})^2} \\
&= G_{\sqrt{2}}(x)
\end{aligned}$$

Convolving two Gaussian with $\sigma = 1$ results in a Gaussian with $\sigma = \sqrt{2}$. So yes the Gaussian smoothing of a Gaussian smoothed image is a Gaussian smoothed image. The resulting smoothing scale is $G_{\sigma 1} * G_{\sigma 2} = G_{\sqrt{\sigma 1^2 + \sigma 2^2}}$

1.26

A matrix is seperable if the rank of the matrix is 1. Take an random Gaussian kernel of scale 1 for example:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Since the first and third row are identical and the second row is twice the first row, the rank will be 1. That is why a 2-dimensional Gaussian kernel is separable. The same principle applies to a Gaussian function.

1.27

Since a derivative of a Gaussian is a partial derivative, the direction perpendicular to partial derivative remains the same.

$$\begin{aligned}
\frac{\partial G_\sigma}{\partial x}(x) &= -\frac{x}{\sigma^2} G_\sigma(x) \\
&= [-\frac{x}{\sigma^2} G_\sigma(x)][G_\sigma(y)]
\end{aligned}$$

That is why every Gaussian derivative is still seperable.

2 Implementation of Gaussian derivatives

We used the calculations in the lecture notes to decide the size of the sampling grid. This is 2.5 times as large as sigma, must be rounded off, than we check if it is uneven and if so, we added one to obtain the appropriate sampling grid.
The sigma stands for standard deviation, and a lower sigma means less deviation. The expected sum of the sum of the Gaussian - `sum(sum(Gauss(sigma)))` - is 1. This is obtained by executing the Gaussian, calculating `sum(sum(gauss))` and thus, obtaining the normalizing factor. Then, the Gaussian is divided by this normalizing factor. Otherwise, `sum(sum(Gauss(sigma)))` will not be equal to 1, and the intensity of the image will not be maintained.
The standard deviation σ has the same physical unit as the input data. If we consider the average length (in cm), the standard deviation will be expressed in centimeters as well. As we are dealing with pixeldistances, the physical unit of σ is in pixeldistances as well.
The computational complexity is influenced by the scale. The runtimes are directly related to the size of the kernel. When the size of the kernel increases, more calculations needs to be done. Where n is the size of the kernel, n^2 calculations need to be done per pixel. In terms of Big-O notation, the runtime is in $O(n^2)$ of the size of the kernel. Because the size of the kernel is roughly $2.5 \cdot \sigma$, the runtime is in $O((2.5 \cdot \sigma)^2)$, which is $O(\sigma^2)$. If we split up the

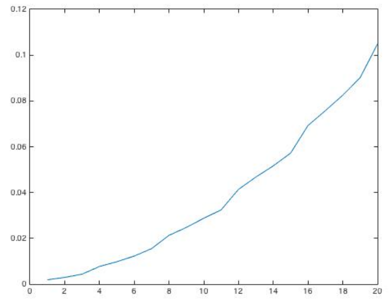

Figure 2: Runtime of convolution, in O(σ^2)

transformation, by convolving along the columns first, and along the rows afterwards, we obtain a straight line (not horizontal, but straight). This tells us it is directly related to the size of the kernel, so directly related to σ, the runtime is in O(n). We also checked the similarity between images resulting from two convolutions, and images

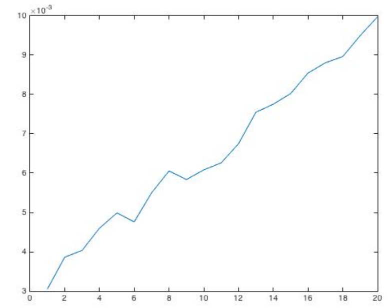

Figure 3: Runtime of separated convolutions, in O(σ)

with a single convolution of the approprate scale. We deducted that a Gaussian with $\sigma = 3$, followed by a Gaussian with $\sigma = 4$ must have the same outcome as a direct convolution with $\sigma = 5$. We used our code from the previous assignment to detect the different pixelvalues. The outcome is 2.8704 which is very small and due to the fact that Matlab tends to round off variables where these calculations should be even more precise. For example, the first and last few elements of the kernel (those that are not close to the center) are so close to zero that rounding off causes our code to be less precise. Because this value (2.8704) increases when we alter $\sigma = 5$, we can conclude that our code is indeed correct.

3 The Canny Edge Detector

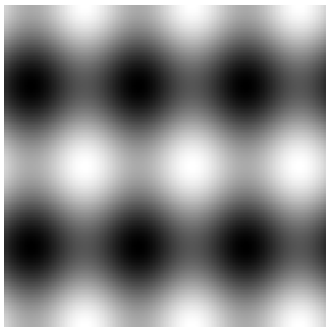

Figure 4: Display of original function

3.1

Given function f is:
$f(x, y) = A \cdot sin(Vx) + B \cdot cos(Wy)$
For analytically calculating the derivatives of this function, you must apply the chain rule. For instance the derivative of f over x is calculated in the following way:
$f(x, y) = A \cdot sin(Vx) + B \cdot cos(Wy)$
$u = Vx$
$f(x, y) = A \cdot sin(u) + B \cdot cos(Wy)$
$f_x(x, y) = A \cdot cos(u) \cdot u'$
$u' = V$
$f_x(x, y) = A \cdot V \cdot cos(Vx)$
The principle holds for the rest of the derivatives. This leads to the following derivatives.
$f_y(x, y) = -B \cdot W \cdot sin(Wy)$
$f_{xx} = -A \cdot V^2 \cdot sin(Vx)$
$f_{yy} = -B \cdot W^2 \cdot cos(Wy)$
$f_{xy} = A \cdot V \cdot cos(Vx) + B \cdot W \cdot sin(Wy)$

3.2

3.3

Note: when executing the Matlab code it might not output the same image as shown in Figure 7. The scaling property did not work in Matlab, and after editing some settings we finally managed to make the arrows visible. f_w can be found through the following equation:

$$f_w = \frac{1}{\sqrt{f_x^2 + f_y^2}} \left((f_x f_y) \cdot \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{pmatrix} \right) f = \frac{f_x f_x + f_y f_y}{\sqrt{f_x^2 + f_y^2}} = \sqrt{f_x^2 + f_y^2} \tag{3}$$

Figure 5: Display of derivatives of function over X and Y

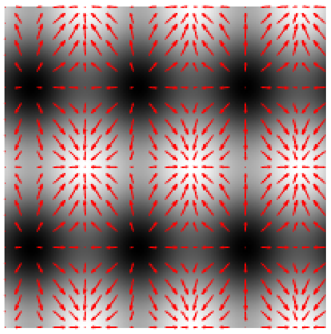

Figure 6: Display of gradient vectors over original function

4 How to run

To check all matlab code for part 2 (chronologically), the following files must be run:

1. Part 1: theoryQuestion1_20.m
2. Part 2: plotspeed.m
3. Part 2: seperable.m
4. Part 2: consecutive.m
5. Part 3: assignment3_5.m (stepwise, until after the rotation)
6. Part 3: cannyEdgeDetector.m

All other files can be opened for inspection, but do not require running (they are used in the files mentioned above). All theory questions are answered in this document and comments can be found in the code.

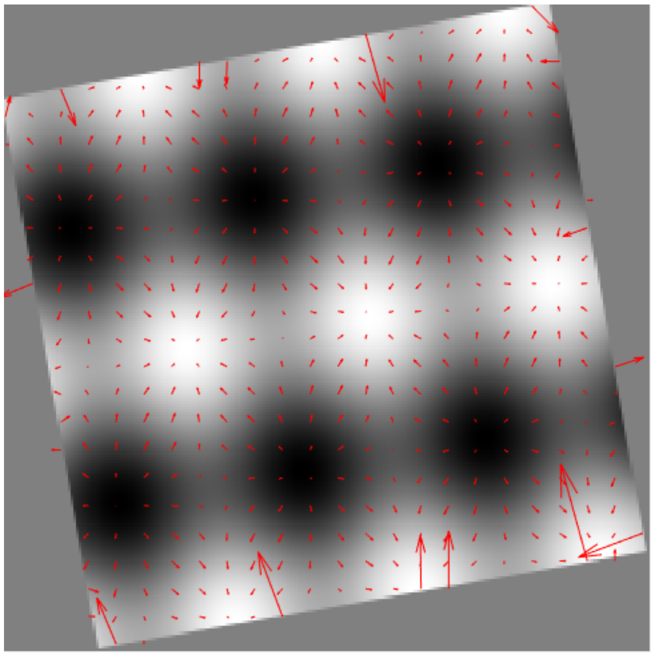
Figure 7: Display of gradient vectors after rotation