Lab 02

Basic Image Processing
Fall 2020

2D convolution in Practice

- In practice both the kernel and the image have finite size.

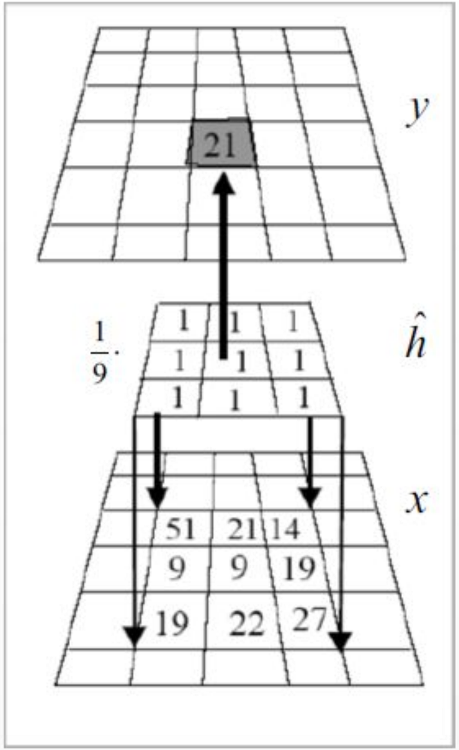
Let h and $\hat{h}$ be $(2r_1+1)\times(2r_2+1)$ sized kernels, where $\hat{h}$ is the $180°$ rotated version of h :

$$h=\begin{bmatrix} a_{-r_1,-r_2} & \cdots & a_{-r_1,r_2} \\ \vdots & \ddots & \vdots \\ a_{r_1,-r_2} & \cdots & a_{r_1,r_2} \end{bmatrix} \text{ and } \hat{h}=\begin{bmatrix} a_{r_1,r_2} & \cdots & a_{r_1,-r_2} \\ \vdots & \ddots & \vdots \\ a_{-r_1,r_2} & \cdots & a_{-r_1,-r_2} \end{bmatrix}$$

$$g(x,y)=\sum_{l=-r_1}^{r_1}\sum_{l=-r_2}^{r_2} f(k,l)\cdot h(x-k,y-l)=$$

$$=\sum_{k=-r_1}^{r_1}\sum_{l=-r_2}^{r_2} h(k,l)\cdot f(x-k,y-l)=$$

$$\boxed{=\sum_{k=-r_1}^{r_1}\sum_{l=-r_2}^{r_2} \hat{h}(k,l)\cdot f(x+k,y+l)}$$



2D convolution in practice

`I` =

.8	.9	.3	.4	.3	.8
.0	.2	.3	.4	.2	.1
.2	.8	.2	.1	.2	.3
.5	.3	.2	.1	.3	.2

`K_rot` =

1	0	1
0	-4	0
1	0	1

The inputs of the 2D convolution function are the image and the kernel.

`O` =

The output is an image which has the same size as the input.

2D convolution in practice

`I` =

.8	.9	.3	.4	.3	.8
.0	.2	.3	.4	.2	.1
.2	.8	.2	.1	.2	.3
.5	.3	.2	.1	.3	.2

`K_rot` =

1	0	1
0	-4	0
1	0	1

$$\sum \begin{pmatrix} .9 \times 1 + .3 \times 0 + .4 \times 1 \\ .2 \times 0 + .3 \times -4 + .4 \times 0 \\ .8 \times 1 + .2 \times 0 + .1 \times 1 \end{pmatrix} = 1.0$$

`O` =

		1.0			

In the output a pixel value is computed using the values in the corresponding neighborhood and the rotated kernel matrix.

The matrices are multiplied elementwise and the values are summed.

2D convolution in practice

`I` =

.8	.9	.3	.4	.3	.8
.0	.2	.3	.4	.2	.1
.2	.8	.2	.1	.2	.3
.5	.3	.2	.1	.3	.2

`K_rot` =

1	0	1
0	-4	0
1	0	1

With this method almost every pixel of the output can be calculated.

`O` =

		1.0			

2D convolution in practice

`I` =

	?	?	?		
.8	.9	.3	.4	.3	.8
.0	.2	.3	.4	.2	.1
.2	.8	.2	.1	.2	.3
.5	.3	.2	.1	.3	.2

`K_rot` =

1	0	1
0	-4	0
1	0	1

Using this method almost every pixel of the output can be calculated.

`O` =

		1.0			

The problem is that on the edges of the output the neighbor-hood includes non-existing pixels.

2D convolution in practice

`I` =

0	0	0	0	0	0	0	0
0	.8	.9	.3	.4	.3	.8	0
0	.0	.2	.3	.4	.2	.1	0
0	.2	.8	.2	.1	.2	.3	0
0	.5	.3	.2	.1	.3	.2	0
0	0	0	0	0	0	0	0

`K_rot` =

1	0	1
0	-4	0
1	0	1

Solution: extend the image; create a zero-padded version (add some rows and columns to the matrix to make its size 'OK').

`O` =

		1.0			

2D convolution in practice

`I` =

0	0	0	0	0	0	0	0
0	.8	.9	.3	.4	.3	.8	0
0	.0	.2	.3	.4	.2	.1	0
0	.2	.8	.2	.1	.2	.3	0
0	.5	.3	.2	.1	.3	.2	0
0	0	0	0	0	0	0	0

`K_rot` =

1	0	1
0	-4	0
1	0	1

With this ‘trick’ the non-existing pixels can be treated as zeros and the computation can be done just like in the previous case.

`O` =

		-0.6			
		1.0			

$$\sum \begin{pmatrix} 0 \times 1 + 0 \times 0 + 0 \times 1 \\ .9 \times 0 + .3 \times -4 + .4 \times 0 \\ .2 \times 1 + .3 \times 0 + .4 \times 1 \end{pmatrix} = -0.6$$

2D convolution in practice

`I` =

0	0	0	0	0	0	0	0
0	.8	.9	.3	.4	.3	.8	0
0	.0	.2	.3	.4	.2	.1	0
0	.2	.8	.2	.1	.2	.3	0
0	.5	.3	.2	.1	.3	.2	0
0	0	0	0	0	0	0	0

`K_rot` =

1	0	1
0	-4	0
1	0	1

`O` =

		-0.6			
		1.0			

2D convolution in practice

`I` =

0	0	0	0	0	0	0	0
0	.8	.9	.3	.4	.3	.8	0
0	.0	.2	.3	.4	.2	.1	0
0	.2	.8	.2	.1	.2	.3	0
0	.5	.3	.2	.1	.3	.2	0
0	0	0	0	0	0	0	0

`K_rot` =

1	0	1
0	-4	0
1	0	1

With the appropriate padding even the corner pixels can be computed.

`O` =

-3.0		-0.6			
		1.0			

$$\sum \begin{pmatrix} 0 \times 1 + 0 \times 0 + 0 \times 1 \\ 0 \times 0 + .8 \times -4 + .9 \times 0 \\ 0 \times 1 + .0 \times 0 + .2 \times 1 \end{pmatrix} = -3.0$$

Now please

download the ‘Lab 02’ code package

from the

moodle system

Exercise 1

Implement the function `myconv` in which:

- Extend your input image (`input_img`) with zero-valued boundary cells. Use `padarray()`.
- Rotate your kernel (`kernel`) with 180 degrees, (to ensure the right order of elements for element-wise multiplication – see the boxed formula on bottom of Slide 2). Use `rot90()`.
- Iterate through your extended image with two (nested) `for` loops, multiplying every portion of your extended image with the rotated kernel (even include the corner regions as shown in Slide 10).
- The resulting image (`output_img`) should have the same size as the input image (`input_img`).

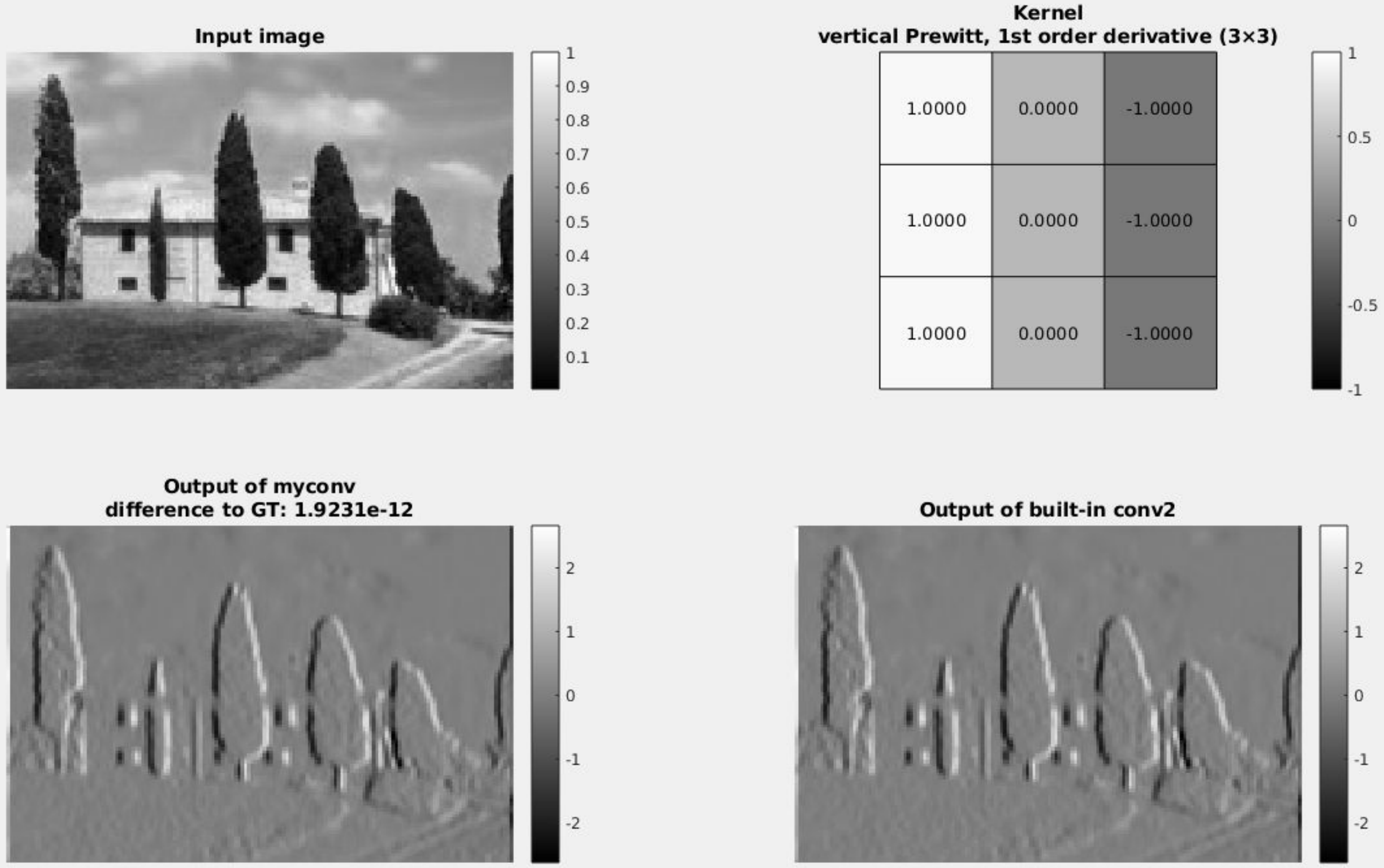
Exercise 1 – continued

You can assume that the input of the function is a double type grayscale image with values in the [0,1] range. You can also know that the **size of the kernel is 3 × 3.**

You should return the result of the convolution “as is”, without any scaling or type conversion.

Run `script1.m` to check your implementation, and please **examine the result.**

- Numerical check:
  - the calculated difference value should be smaller than 10^{-9}
  - the dynamics range of the convolved image is moved from [0, 1] to approx. [-2.5, 2.5]
- Visual check: the left side of the trees should be black, the right should be white.


Input image
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Kernel
vertical Prewitt, 1st order derivative (3×3)
1.0000 0.0000 -1.0000
1.0000 0.0000 -1.0000
1.0000 0.0000 -1.0000
1
0.5
0
-0.5
-1
Output of myconv
difference to GT: 1.9231e-12
2
1
0
-1
-2
Output of built-in conv2
2
1
0
-1
-2

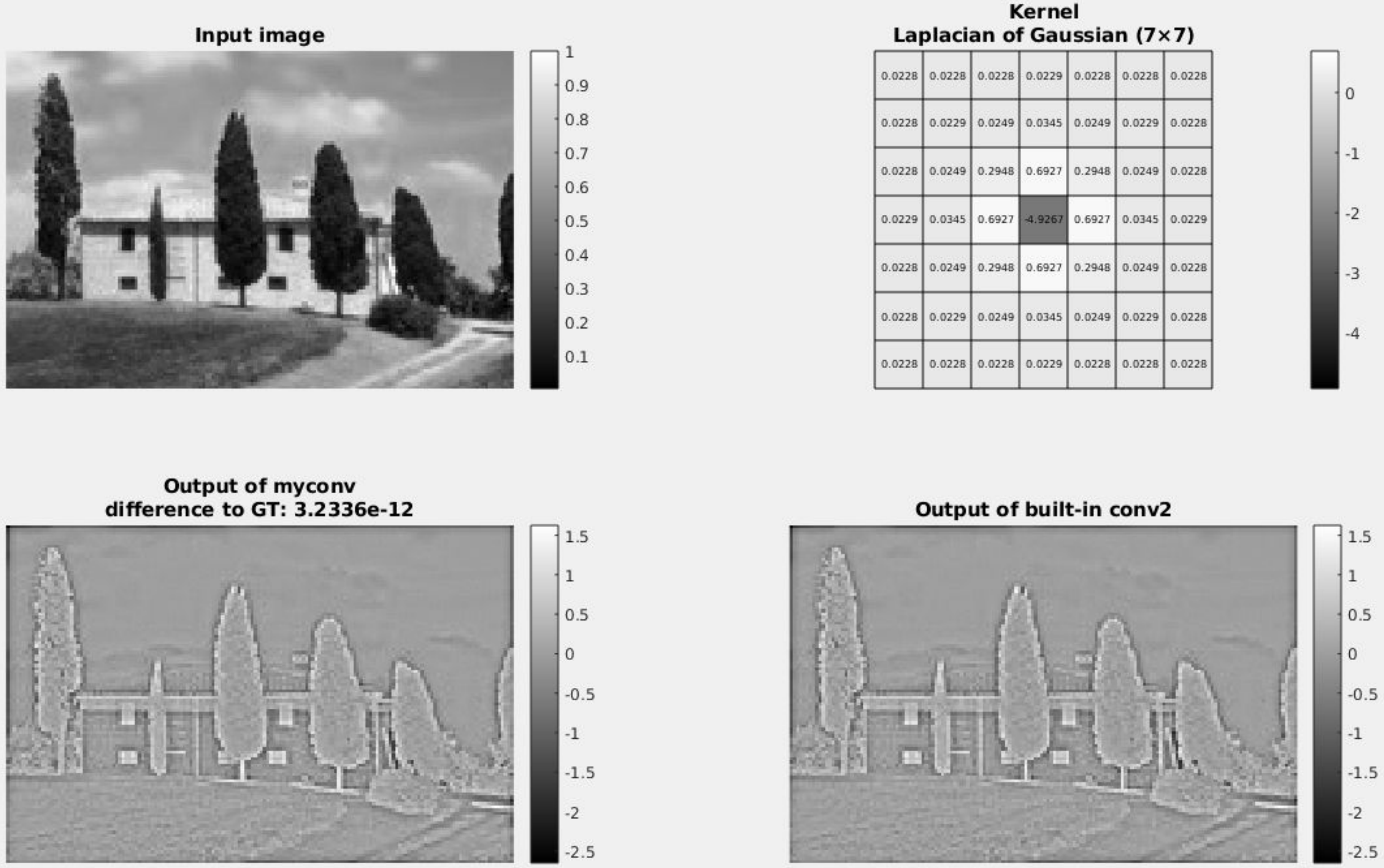
Exercise 2

Modify your function `myconv` in order to:

- Be able to compute with kernels of size $(2k+1) \times (2k+1)$ where $k = 1, 2, 3, ...$
  (it means: your padding should depend on the size of the incoming kernel)

- Furthermore, all of the previous conditions should be satisfied.

Run `script2.m` to check your implementation, and please **examine the result.**



Input image
Kernel
Laplacian of Gaussian (7×7)
0.0228 0.0228 0.0228 0.0229 0.0228 0.0228 0.0228
0.0228 0.0229 0.0249 0.0345 0.0249 0.0229 0.0228
0.0228 0.0249 0.2948 0.6927 0.2948 0.0249 0.0228
0.0229 0.0345 0.6927 -4.9267 0.6927 0.0345 0.0229
0.0228 0.0249 0.2948 0.6927 0.2948 0.0249 0.0228
0.0228 0.0229 0.0249 0.0345 0.0249 0.0229 0.0228
0.0228 0.0228 0.0228 0.0229 0.0228 0.0228 0.0228
Output of myconv
difference to GT: 3.2336e-12
Output of built-in conv2

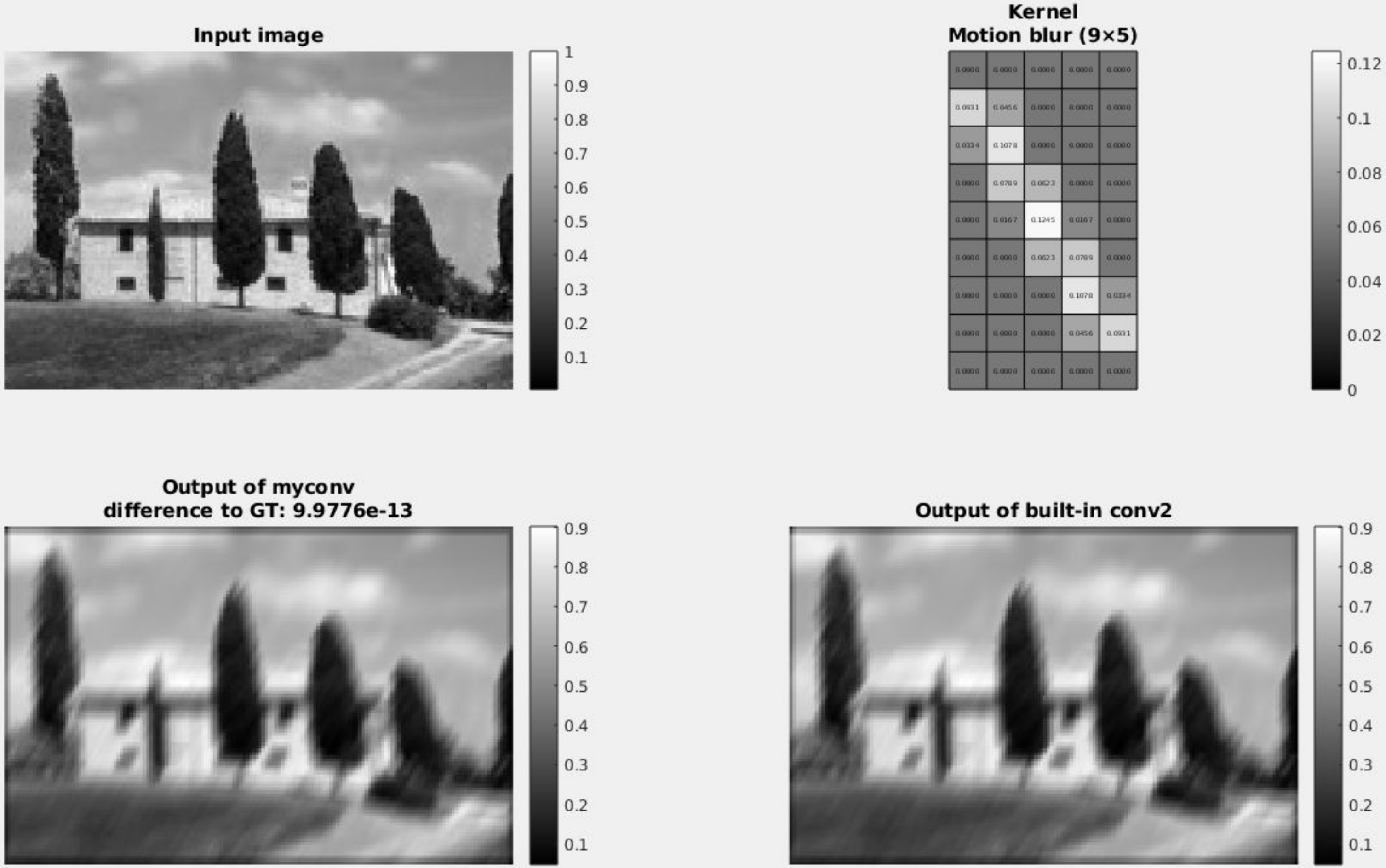
Exercise 3

Modify your function `myconv` in order to:

- Be able to compute with kernels of size $(2a+1) \times (2b+1)$ where
  $a = 1, 2, 3, \ldots$
  $b = 1, 2, 3, \ldots$ $\quad a \neq b$
  (it means: your padding should depend on the size of the incoming kernel in both dimensions as the kernel is not a square anymore)

- Furthermore, all of the previous conditions should be satisfied.

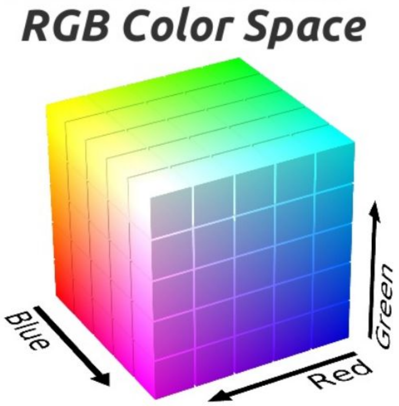
Run `script3.m` to check your implementation, and please **examine the result.**

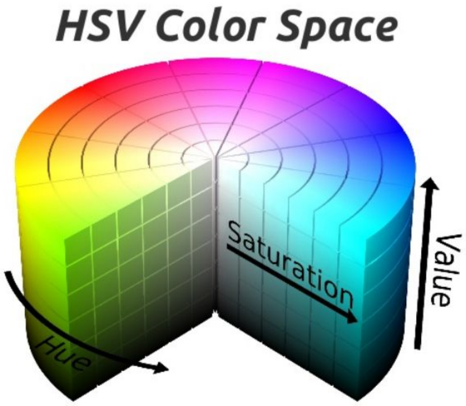


Input image
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Kernel
Motion blur (9×5)
0.0000 0.0000 0.0000 0.0000 0.0000
0.0931 0.0456 0.0000 0.0000 0.0000
0.0334 0.1078 0.0000 0.0000 0.0000
0.0000 0.0789 0.0623 0.0000 0.0000
0.0000 0.0167 0.1245 0.0167 0.0000
0.0000 0.0000 0.0623 0.0789 0.0000
0.0000 0.0000 0.0000 0.1078 0.0334
0.0000 0.0000 0.0000 0.0456 0.0931
0.0000 0.0000 0.0000 0.0000 0.0000
0.12
0.1
0.08
0.06
0.04
0.02
0
Output of myconv
difference to GT: 9.9776e-13
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Output of built-in conv2
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

Second part: Color spaces

You should remember…





`rgb2hsv()`	
RGB →	HSV
`uint8` ([0, 255]) `double` ([0, 1]) →	`double` ([0, 1])

`hsv2rgb()`	
HSV →	RGB
`double` ([0, 1]) →	`double` ([0, 1])

Image source: https://www.slideshare.net/michelalves/about-perception-and-hue-histograms-in-hsv-space

Thresholding, segmentation

- Thresholding / binarization with a single number:
  If the pixel intensity in the original image is higher than the threshold value then the pixel becomes white in the output image; otherwise it will be black.

```
R = squeeze(I(:, :, 1));
R_t = R > 128;
```

selecting the red-channel
bonus: if the singleton is the last dim, `squeeze` is not necessary

binarization at mid-gray

- Segmentation:
  The process of partitioning an image into multiple segments (sets of pixels, also known as image objects).

(images from: https://towardsdatascience.com/semantic-segmentation-of-150-classes-of-objects-with-5-lines-of-code-7f244fa96b6c)

Exercise 4

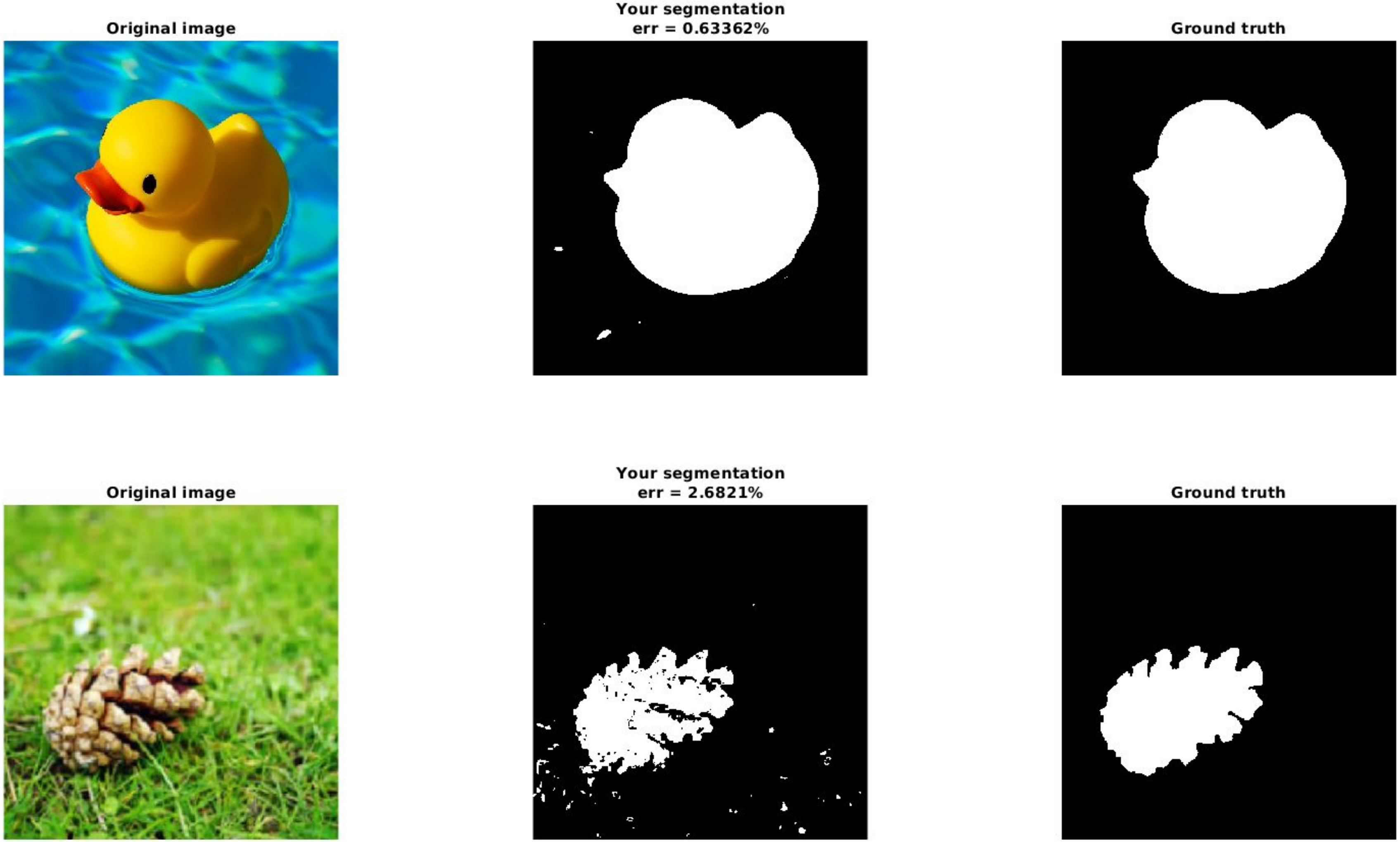
Implement the function `find_the_duck()` in which:

- Segment the input image based on color channels. You can use color space transformations and thresholding only.
- The function should return a logical matrix where true values indicate ‘duck’.
- It is required to have an **error value less than 1.5%**

Implement the function `find_the_pine()` in which:

- Segment the input image based on color channels. You can use color space transformations and thresholding only.
- The function should return a logical matrix where true values indicate ‘pine’.
- It is required to have an **error value less than 3.0%**

Run `script4.m` and examine the results.


Original image
Your segmentation
err = 0.63362%
Ground truth
Original image
Your segmentation
err = 2.6821%
Ground truth

THE END