Lab 03

Basic Image Processing Algorithms
Fall 2022

Part 1
Convolution

2D convolution in Practice

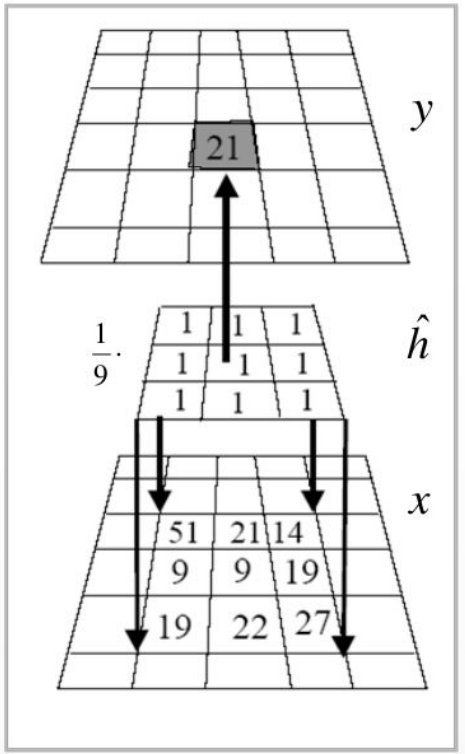
- Let h and $\hat{h}$ be $(2r_1 + 1) \times (2r_2 + 1)$ sized kernels where $\hat{h}$ is the rotated version of h with $180°$

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

$$g(x,y) = (f * h)(x,y) = (h * f)\,(x,y) =$$

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

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



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

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 03’ 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 3). 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.**

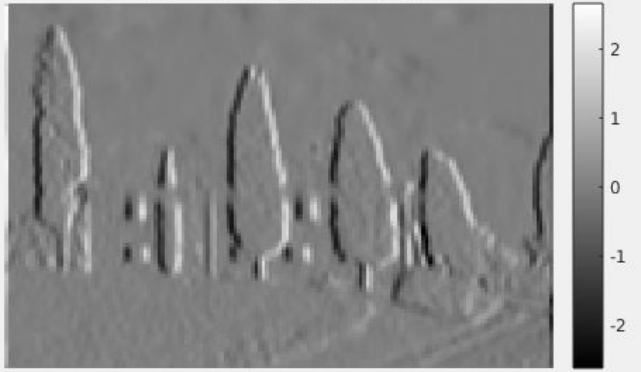
- Numerical check:
  - the calculated difference value should be smaller than 10^{-9}
  - the dynamics range of the convolved image is moved from [0, 1] to approx. [-2.5, 2.5]
- Visual check: the left side of the trees should be black, the right should be white.

Input image

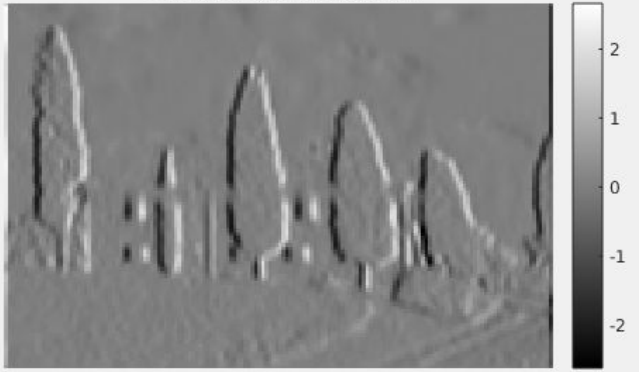
Kernel
vertical Prewitt, 1st order derivative (3×3)

1.0000	0.0000	-1.0000
1.0000	0.0000	-1.0000
1.0000	0.0000	-1.0000

Output of myconv
difference to GT: 1.9231e-12



Output of built-in conv2



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:

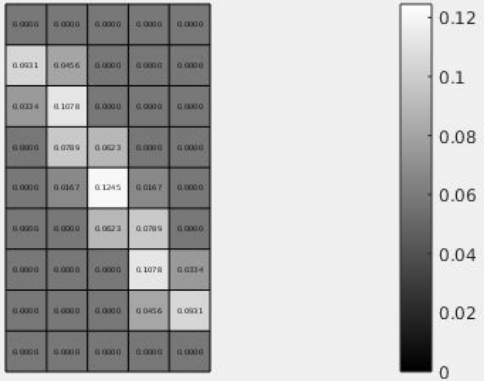
- Be able to compute with kernels of size $(2a+1) \times (2b+1)$ where
  $a = 1, 2, 3, \ldots$
  $b = 1, 2, 3, \ldots$ $a \neq b$
  (it means: your padding should depend on the size of the incoming kernel in both dimensions as the kernel is not a square anymore)

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

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

Input image

Kernel
Motion blur (9×5)



Output of myconv
difference to GT: 9.9776e-13

Output of built-in conv2

Part 2
Image enhancement

The Histogram of an Image

- **Histogram**:

  $h(k)$ = the number of pixels on the image with value k.

Original Image*

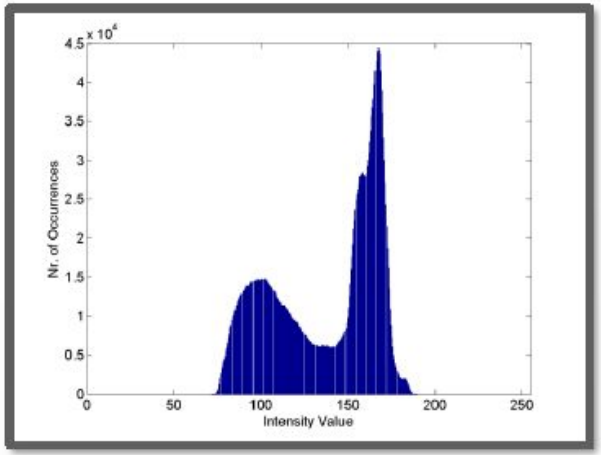


Image Histogram

- The histogram normalized with the total number of pixels gives us the ***probability density function*** of the intensity values.

* Modified version of Riverscape with Ferry by Salomon van Ruysdael (1639)

Histogram Transformations

- **Histogram Stretching**:
  - Based on the histogram we can see that the image does not use the whole range of possible intensities:
    - Minimum intensity level: 72
    - Maximum intensity level: 190
  - With the following transformation we can stretch the intensity values so they use the whole available range:

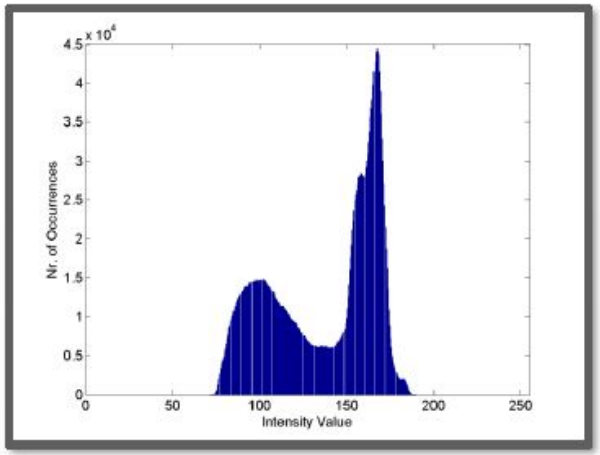


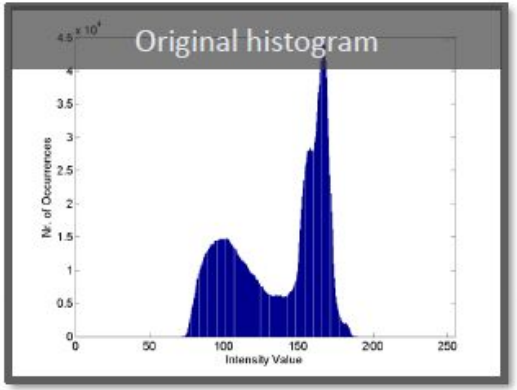
Image Histogram

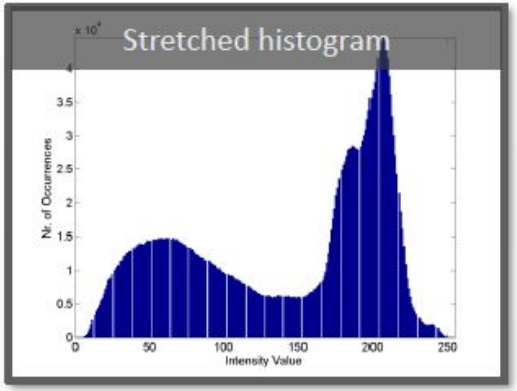
$$y(n_1, n_2) = \frac{255}{x_{\max} - x_{\min}} \cdot (x(n_1, n_2) - x_{\min})$$

$$x_{\max} = \max_{n_1, n_2}(x(n_1, n_2)) \qquad x_{\min} = \min_{n_1, n_2}(x(n_1, n_2))$$

Histogram Transformations

- **Histogram Stretching:**





Point-wise Intensity Transformation

- **Log transformation:** $y(n_1, n_2) = c \cdot \log\left(x(n_1, n_2) + 1\right)$
  - Expands low and compresses high pixel value range

Original Image*

Log Image

Log Image
after histogram stretching

* Abbaye du Thoronet by Lucien Hervé (1951)

Exercise 4

Implement the function `calc_hist_vector` in which:

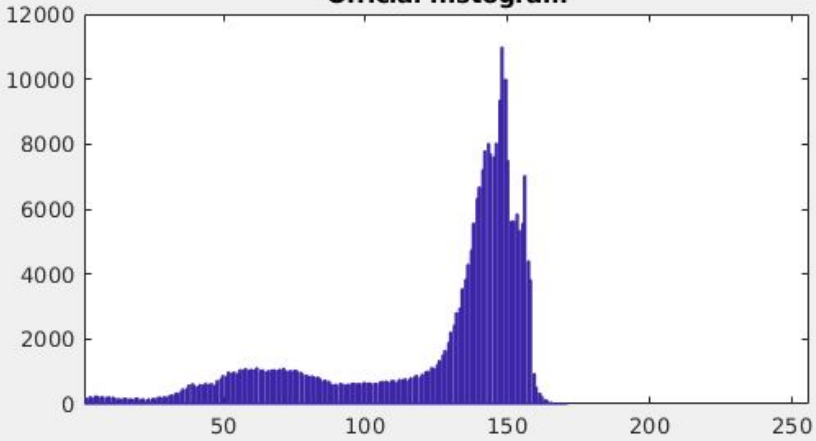
- Create the empty `hist_vector` as an accumulator vector, the number of elements should be the number of possible pixel intensities (256).
- Iterate through your input image (`input_img`) with two (nested) `for` loops, registering the intensity-values of every pixel in your accumulator vector:
  `hist_vector(idx) = hist_vector(idx) + 1;`
  (**Be careful!** Image intensity ∈ [0, 255], Matlab vector index ∈ [1, 256])

The sum of your `hist_vector` should give the total number of pixels present in your image.

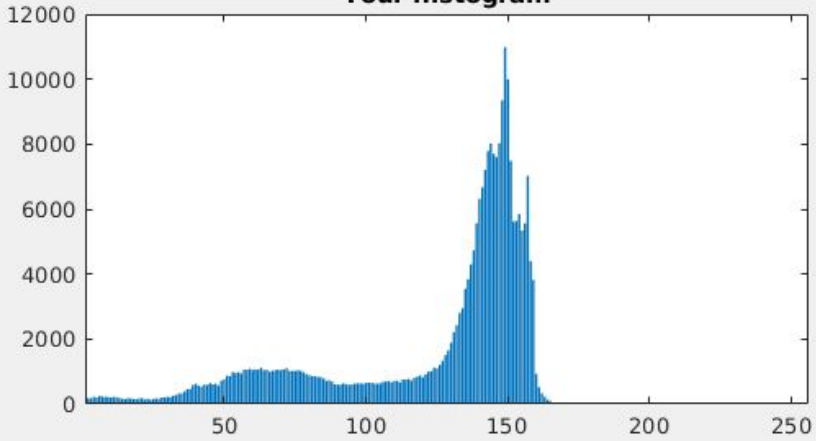
Run `script4.m` which will plot your returned vector as a bar chart.

Grayscale input

Official histogram



Your histogram



Exercise 5

Implement the function `stretch_lin` in which:

- Find the minimum and maximum intensity values of your input image (`input_img`).
- Stretch its dynamic range with the formula given on Slide 21.

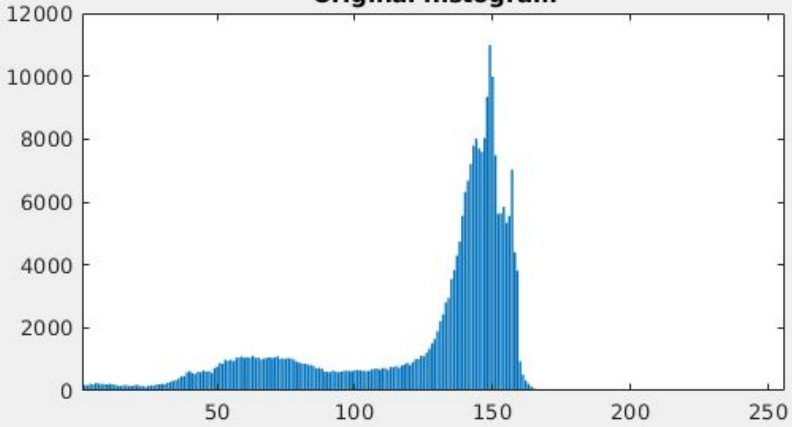
Your resulting image should contain rounded values in the range [0, 255] with type `uint8`.

Run `script5.m` to check your implementation.

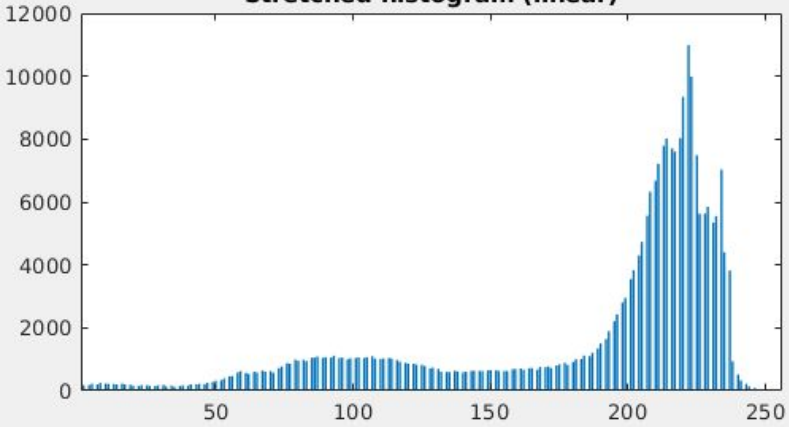
Original image

Stretched image (linear)

Original histogram



Stretched histogram (linear)



Exercise 6

Implement the function `stretch_log` in which:

- Apply the point-wise log transformation at every pixel (as given on Slide 23).
- Find the minimum and maximum intensity values of your transformed image.
- Stretch its dynamic range with the formula given on Slide 21.

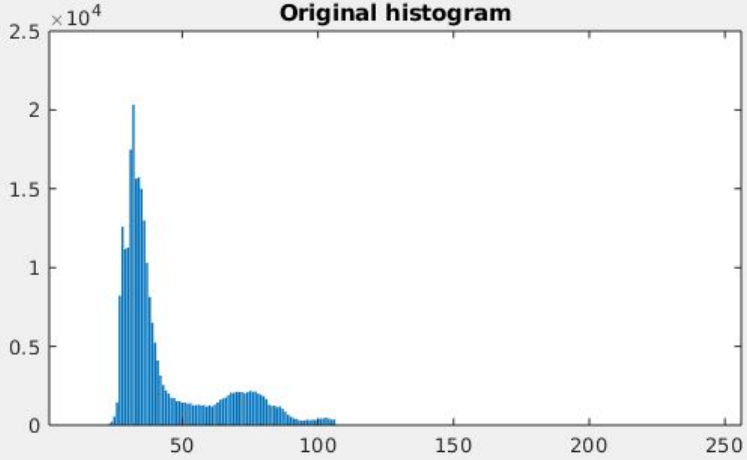
Your resulting image should contain rounded values in the range [0, 255] with type `uint8`.

Run `script6.m` to check your implementation.

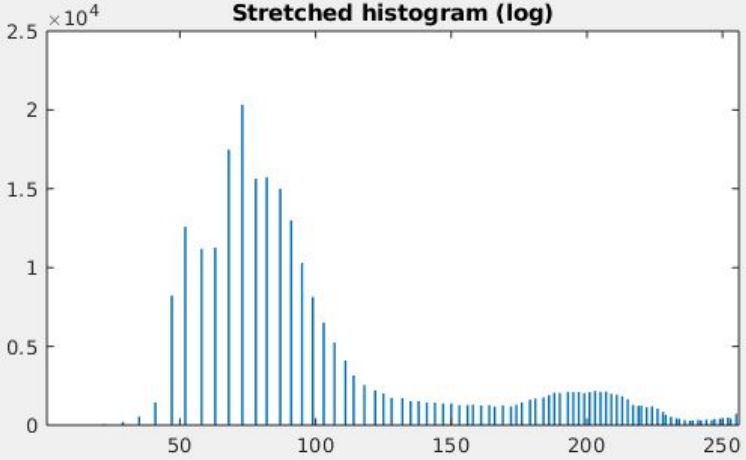
Original image

Stretched image (log)

Original histogram



Stretched histogram (log)



THE END