Working with Images

Reading, Exploring and Analyzing, Feature Extraction

Yordan Darakchiev

Technical Trainer

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

sli.do

#DataScience

Table of Contents

- Image processing
  - Reading
  - Exploring
  - Manipulation
  - Convolution
  - Image Morphology

Image Processing

Understanding What People See

Loading and Inspecting Images

- There are many ways to **read an image**
  - One of the easiest is using scikit-image

```
from skimage.io import imread
tiger_image = imread("tiger.jpg")
```

  - Displaying the image

```
plt.imshow(tiger_image)
```

Loading and Inspecting Images

- The image is actually a **matrix of pixels**
  - Each pixel is an array of three values: **R, G, B** $\in [0; 255]$
  - Grayscale images only have **one value per pixel**
- Most image **processing algorithms** are easier to understand on grayscale images

```
red = tiger_image[:, :, 0]
green = tiger_image[:, :, 1]
blue = tiger_image[:, :, 2]
```

Image Histogram

- As usual, histograms tell us **how the values are distributed**
  - How many dark values, how many light values
  - Maximum brightness, peaks, etc.

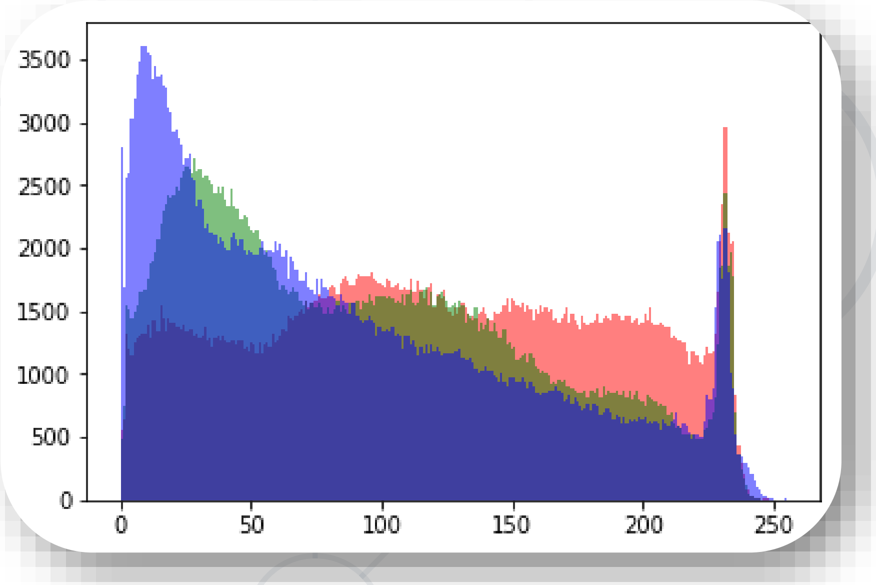


Image Histogram

- Histograms need to have a **single variable**
  - Take each channel separately, e.g., red
  - Convert the **2D matrix to 1D array**: `image.ravel()`
  - Show the histogram as usual
    - It's common to use 256 bins

```
plt.hist(red.ravel(), bins = 256, color = "red")
plt.show()
```

  - We can also plot **all channels on a single histogram**

Converting to Grayscale

- Sometimes working **per channel is not necessary**
  - We can combine **all three channels** and get a grayscale image
  - Simplest way: **get the mean of all values**

```
tiger_grayscale = np.mean(tiger_image, axis = 2)
```

Converting to Grayscale

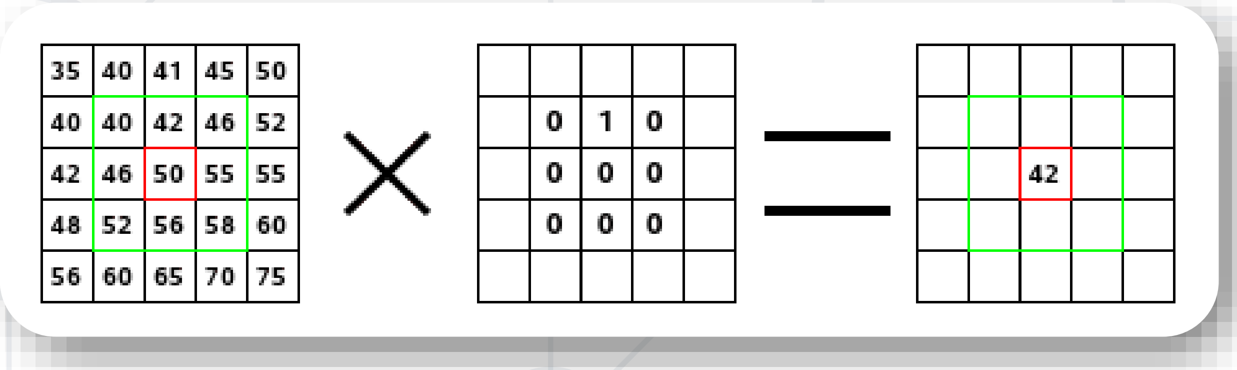
- Better way: **use coefficients for each channel**
  - The human eye **discerns colors differently**
  - Were more sensitive to green colors
  - Some formulas are given here

```
tiger_grayscale = 0.299 * red + 0.587 * green + 0.114 * blue
```

- Depending on the image, the differences **may or may not be easy to see**
- For art purposes, we can experiment with our own **coefficients for combining all channels**

Convolution

- **Convolution kernel (filter)**
  - A small, usually 3x3, matrix of numbers
- Convolution process
  - **Input**: image, kernel
  - **Output**: new image



35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

×

	0	1	0	
	0	0	0	
	0	0	0	

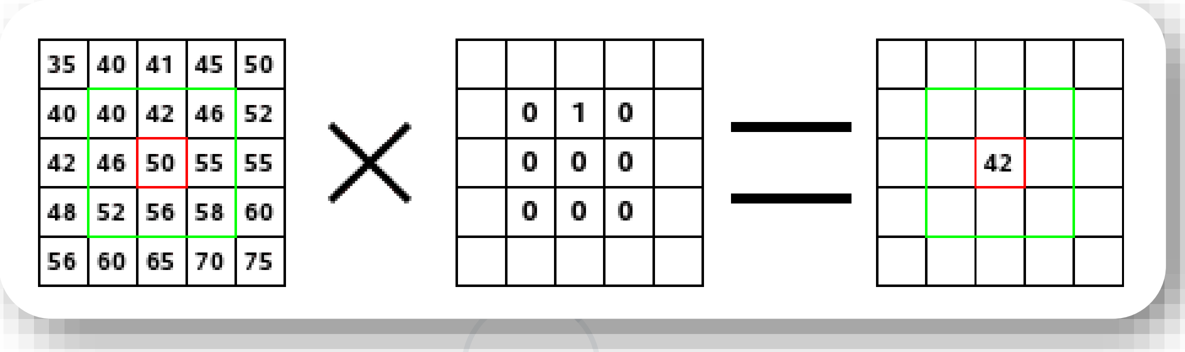
=

		42		

Convolution

Combining the image and a kernel:

- Apply the kernel **over each pixel**
- Multiply the values **element-wise** (Hadamard product)
- Sum **all values**
- Assign the **sum to the corresponding pixel** in the output image
  - Image corners are treated in different ways, not really important how



35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

×

	0	1	0	
	0	0	0	
	0	0	0	

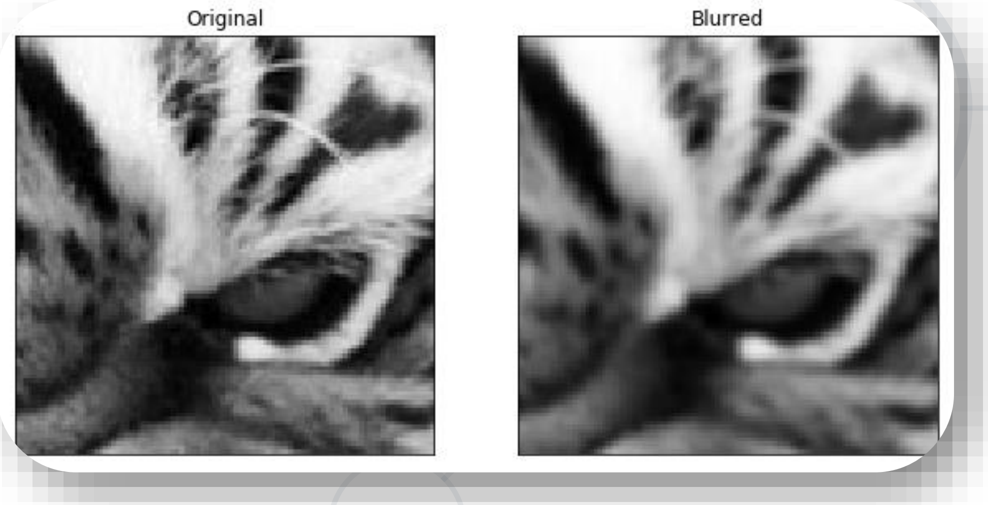
=

		42		

Convolution

- The choice of kernel depends what the **output image will represent**

```
from scipy.ndimage.filters import convolve
convolve(image, kernel)
```



Convolution

Example: **box blur**

```
box_blur_kernel = np.array([
  [1, 1, 1],
  [1, 1, 1],
  [1, 1, 1]
]) / 9

blurred = convolve(tiger_grayscale, box_blur_kernel)
plt.imshow(tiger_grayscale[150:250, 300:400], cmap = "gray")
plt.show()
plt.imshow(blurred[150:250, 300:400], cmap = "gray")
plt.show()
```

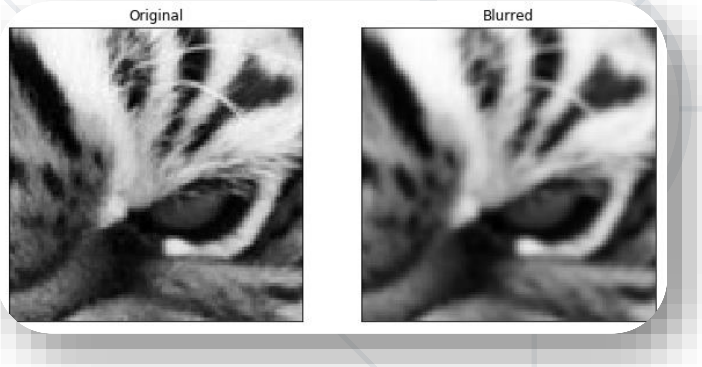
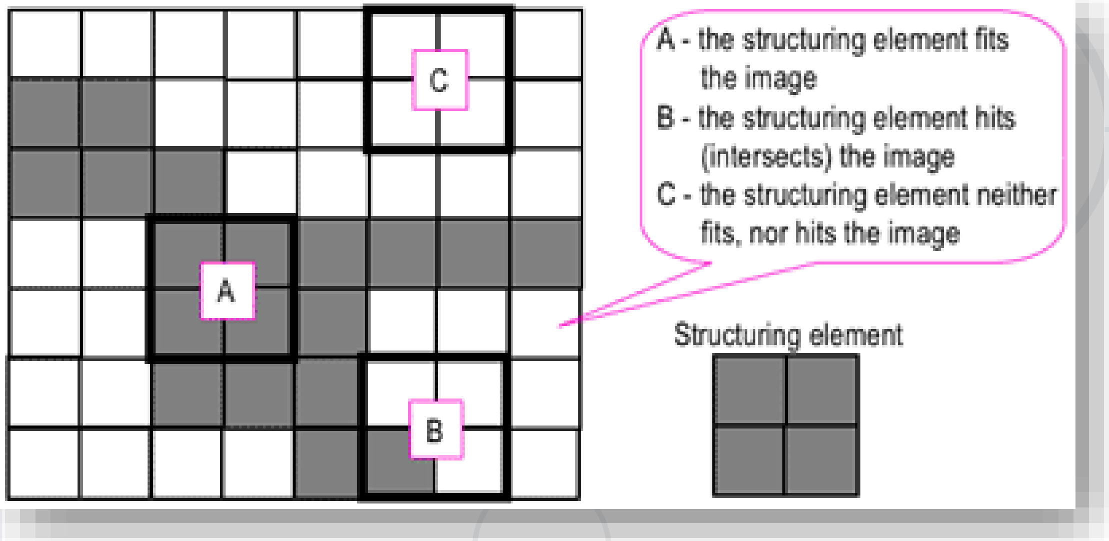


Image Morphology

- Four main operations (see this tutorial)
  - **Dilation**
  - **Erosion**
  - **Opening**
  - **Closing**
- A simple series of **algorithms for image transformation**
- Basic methodology
  - Choose a structuring element (e.g., 2x2 square or cross)
  - Move the element around the image
  - Apply an operation

Image Morphology

- **Input**: **binary image**
  - Pixel values **0** and **1, not [0; 255]**
  - This is called **thresholding**
- **Output**: **transformed image**

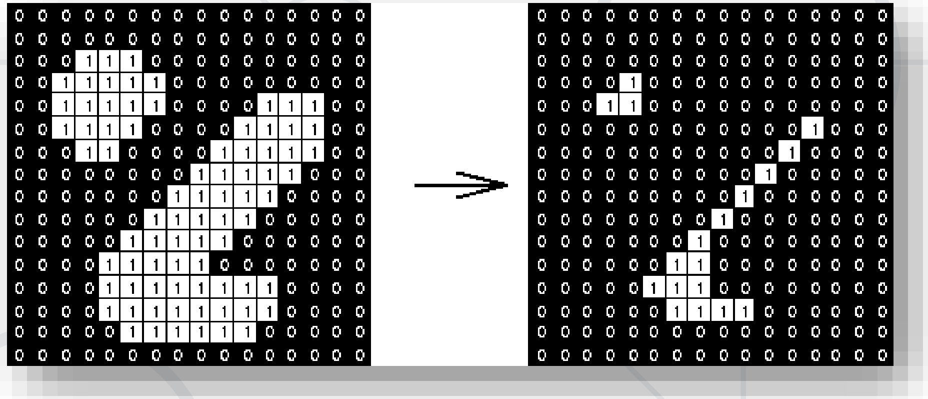


C
A
B
A - the structuring element fits the image
B - the structuring element hits (intersects) the image
C - the structuring element neither fits, nor hits the image
Structuring element

Image Morphology

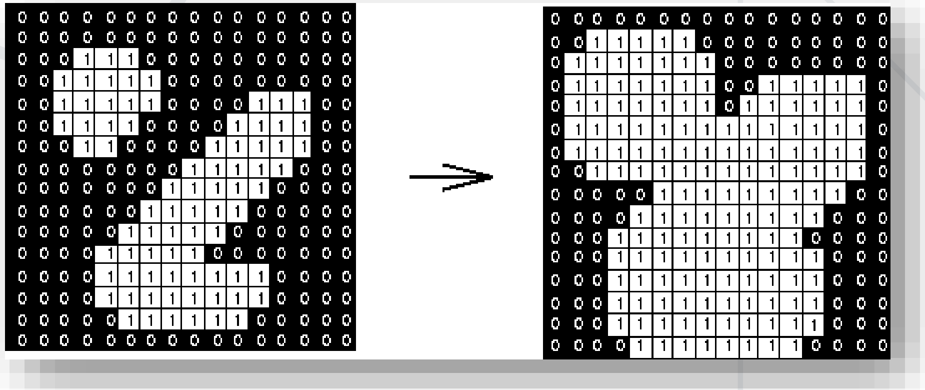
First get **all values** inside the structuring element

- **Erosion: replace all values with the min value**
  - Strips away a layer of pixels
  - Holes become larger
  - Small regions are eliminated



```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0        0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0        0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0        0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0        0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0   ->   0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0        0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0        0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0        0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0        0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0        0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0        0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

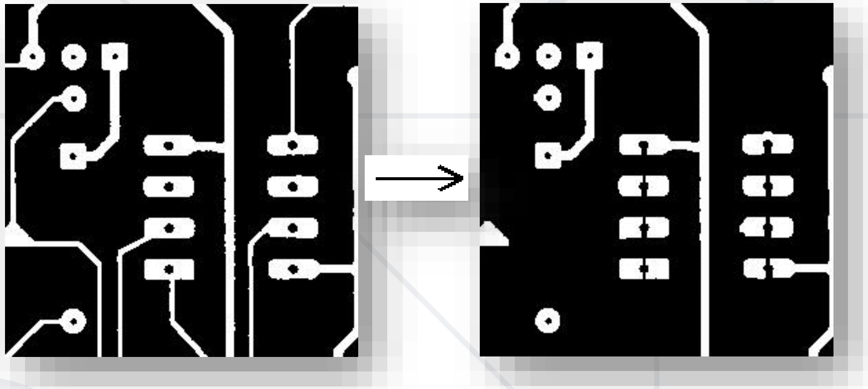
- **Dilation: replace all values with the max value**
  - Adds a layer of pixels
  - Gaps become smaller
  - Small gaps are filled in



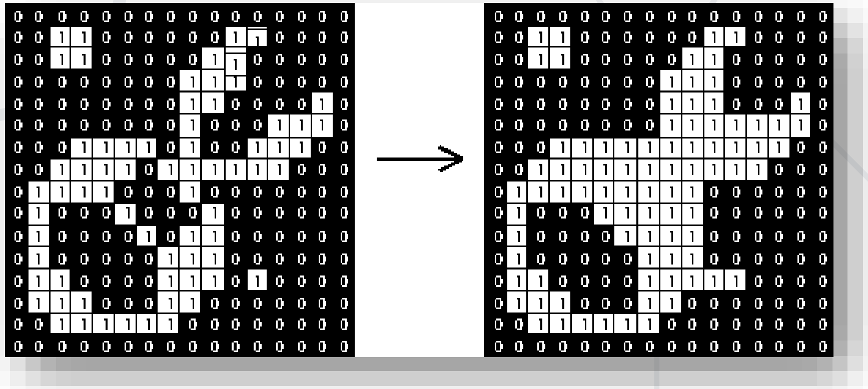
```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0        0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0        0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0        0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0
0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0        0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0
0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0        0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0        0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0   ->   0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0        0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0        0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0        0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0        0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0        0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0        0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0        0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0        0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
```

Image Morphology

- **Opening: erosion followed by dilation**
  - Pixels which survived erosion are restored to their original size
  - Opens up a gap between two objects connected by thin bridges



- **Closing: dilation followed by erosion**
  - Fills in holes in the regions while keeping the initial region sizes



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0
0	0	0	1	1	1	1	0	1	0	0	1	1	1	0	0
0	0	1	1	1	1	0	1	1	1	1	1	1	0	0	0
0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0
0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	0
0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
0	1	1	0	0	0	0	1	1	1	1	1	0	0	0	0
0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Other Operations on Images

- Matrix operations – **pixel-wise**
  - One image:
    - **Addition, Gain, Negative**
    - **Resampling, Cutting**
  - Geometric transformations – **perspective**, **warp**, etc.
  - Two (or more) images:
    - **Addition (multiple exposure)**
    - **Subtraction (difference)**
    - **Division (normalization)**
    - **Averaging**

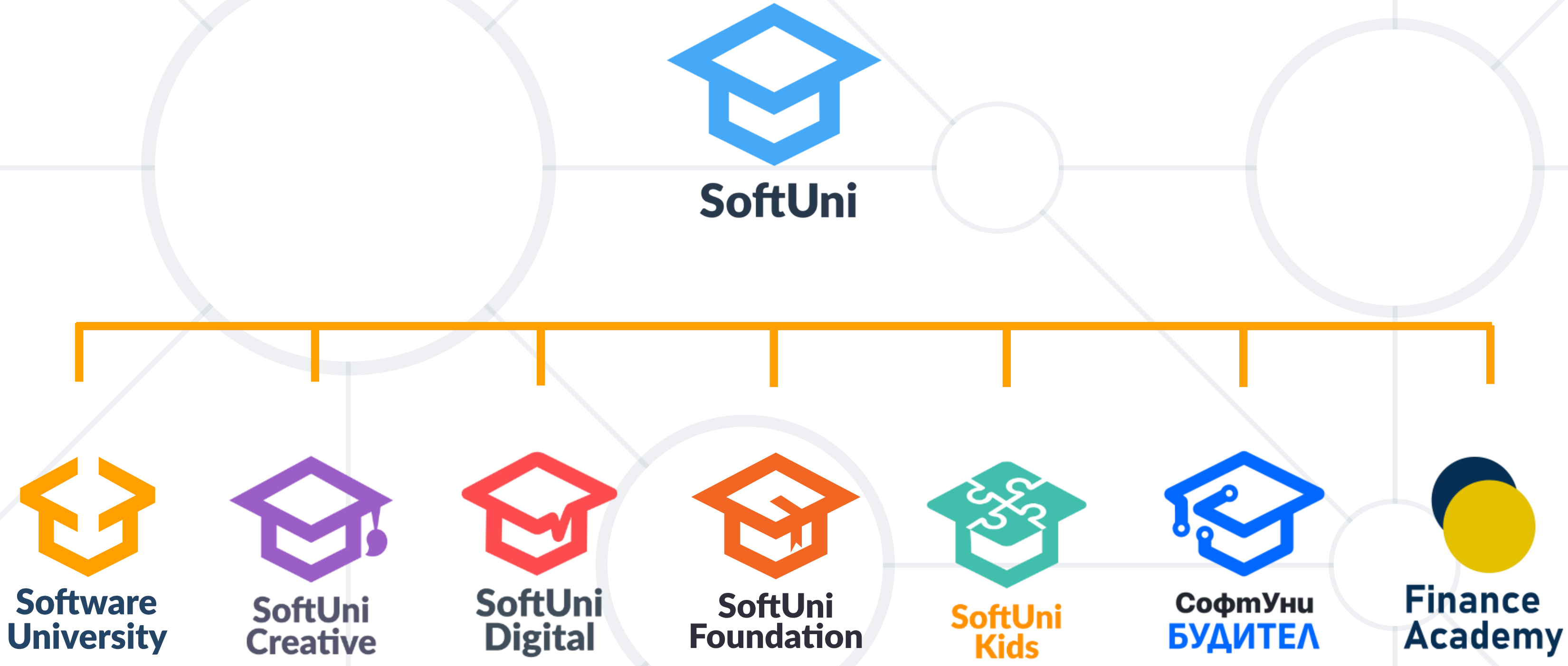
Other Operations on Images

- Thresholding **(usually 2 levels)**
- Fourier **transform**, **filtering** and **convolution**
- Contrast **enhancement**
- Histogram **equalization**
- Stacking (many **2D images** ⇒ **one 3D image**)
- Analysis:
  - **Measurements, Segmentation, Object extraction / Identification**
  - **Enhancements, Inpainting**

Summary

- Image processing
  - Reading
  - Exploring
  - Manipulation
  - Convolution
  - Image Morphology

Questions?



SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

СофтУни БУДИТЕЛ

Finance Academy

SoftUni Diamond Partners

SmartIT

createX

VIVACOM

INDEAVR
Serving the high achievers

Coca-Cola HBC
Bulgaria

Postbank
Решения за твоето утре

AMBITIONED

SOFTWARE
GROUP

SUPER
HOSTING
.BG

DRAFT
KINGS

PHAR
VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg