Image Analysis and Computer Vision

Giacomo Boracchi

giacomo.boracchi@unibocconi.it

February 12th 2024

UEM, Maputo

https://boracchi.faculty.polimi.it

Who I am

Giacomo Boracchi (giacomo.boracchi@polimi.it)

Mathematician (Università Statale degli Studi di Milano 2004),

PhD in Information Technology (DEIB, Politecnico di Milano 2008)

Associate Professor since 2019 at DEIB, Polimi (Computer Science)

My research interests are mathematical and statistical methods for:

- Image analysis and processing
- Machine Learning and in particular unsupervised learning, change and anomaly detection

… and the two combined

Teaching

Advanced courses taught:

- Artificial Neural Networks and Deep Learning (MSc, Polimi)
- Mathematical Models and Methods for Image Processing (MSc, Polimi)
- Advanced Deep Learning Models And Methods (PhD, Polimi)
- Online Learning and Monitoring (PhD, Polimi)
- Learning Sparse Representations for image and signal modeling (PhD, Polimi and TUNI)

- Computer Vision and Pattern Recognition (MSc in USI, Spring 2020)
- Image Analysis and Computer Vision (MSc, Polimi, Prof. Caglioti)

Agenda

- Course Logistics
- Course Outline
- Digital Images
- The elements of photometric image formation
- Practical examples of image manipulation
- Intensity Transformation
- Linear Filtering: Correlation and Convolution
- Nonlinear Filtering and Morphological Image Processing

Course Logistics and Exam Rules

Course Slides

Slides can be found on my website

https://boracchi.faculty.polimi.it/

and follow Tutorials and Talks

https://boracchi.faculty.polimi.it/seminars.html

Colab Folder

In this folder you will find, regularly updated notebooks

https://drive.google.com/drive/folders/1Oj99rb2kKo4KpLxca-uMe7uesy-8RZeD

Notebooks require you to "fill in" some codes or to extend codes we illustrate during lectures to new data/new challenges

Course Requirement and Tools

- Linear algebra
- Rudiments of probability and statistics.
- Basics of machine learning and model fitting (overfitting and underfitting concepts)
- Neural networks (multi-layer perceptron and backpropagation).
- Programming Skills (Python), and…

Course Requirement and Tools

- Linear algebra
- Rudiments of probability and statistics.
- Basics of machine learning and model fitting (overfitting and underfitting concepts)
- Neural networks (multi-layer perceptron and backpropagation).
- Programming Skills (Python), and…

- Willingness to learn

The Course Outline and the Broad Landscape of CVPR

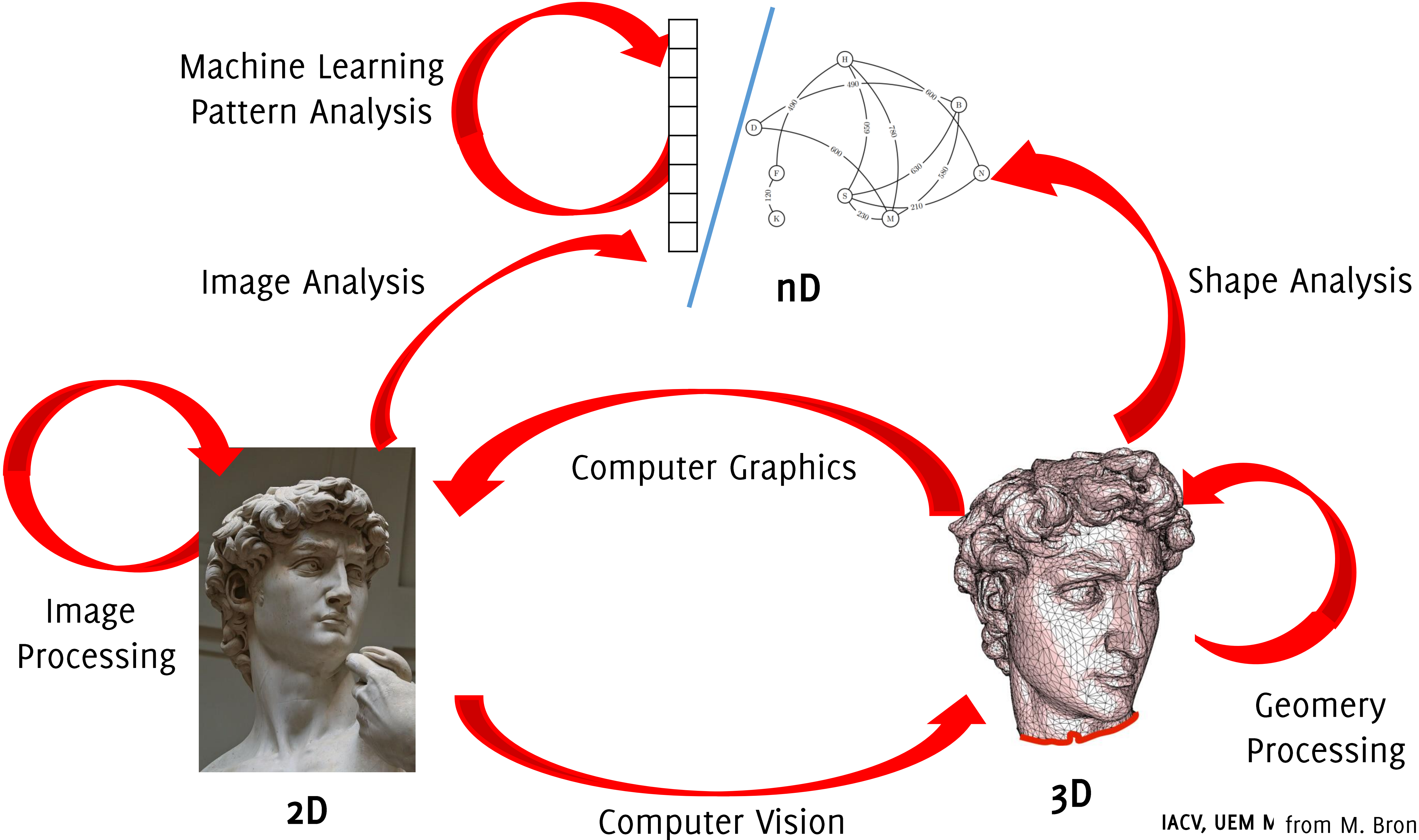
Computer Vision

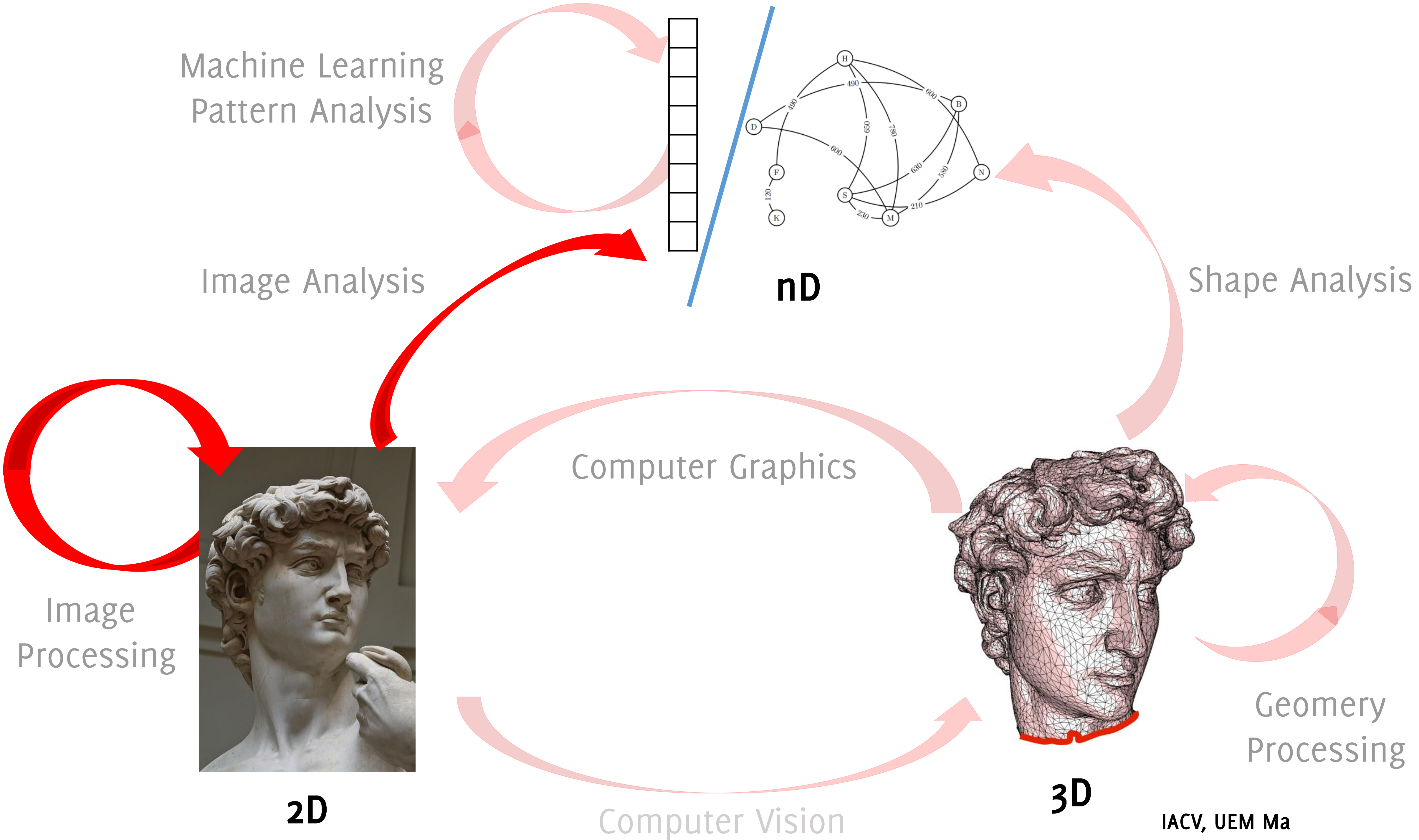
An interdisciplinary scientific field that deals with how computers can be made to **gain high-level understanding from digital images or videos**

Computer Vision

An interdisciplinary scientific field that deals with how computers can be made to **gain high-level understanding from digital images or videos**

… which has grown incredibly fast in the last years


Machine Learning
Pattern Analysis
H
D
B
F
N
S
K
M
490
490
600
650
780
600
630
580
120
230
210
Image Analysis
nD
Shape Analysis
Computer Graphics
Image
Processing
Geomery
Processing
2D
3D
Computer Vision



Machine Learning
Pattern Analysis
Image Analysis
nD
Shape Analysis
Computer Graphics
Image
Processing
2D
Computer Vision
3D
Geomery
Processing
IACV, UEM Ma

Image / Video Restoration

Image / Video Restoration

Restored 28.49 dB

Maggioni, M., Boracchi, G., Foi, A., & Egiazarian, K. (2012). Video denoising, deblocking, and enhancement through separable 4-D nonlocal spatiotemporal transforms. IEEE *TIP 2012*

oisy frame

Denoised

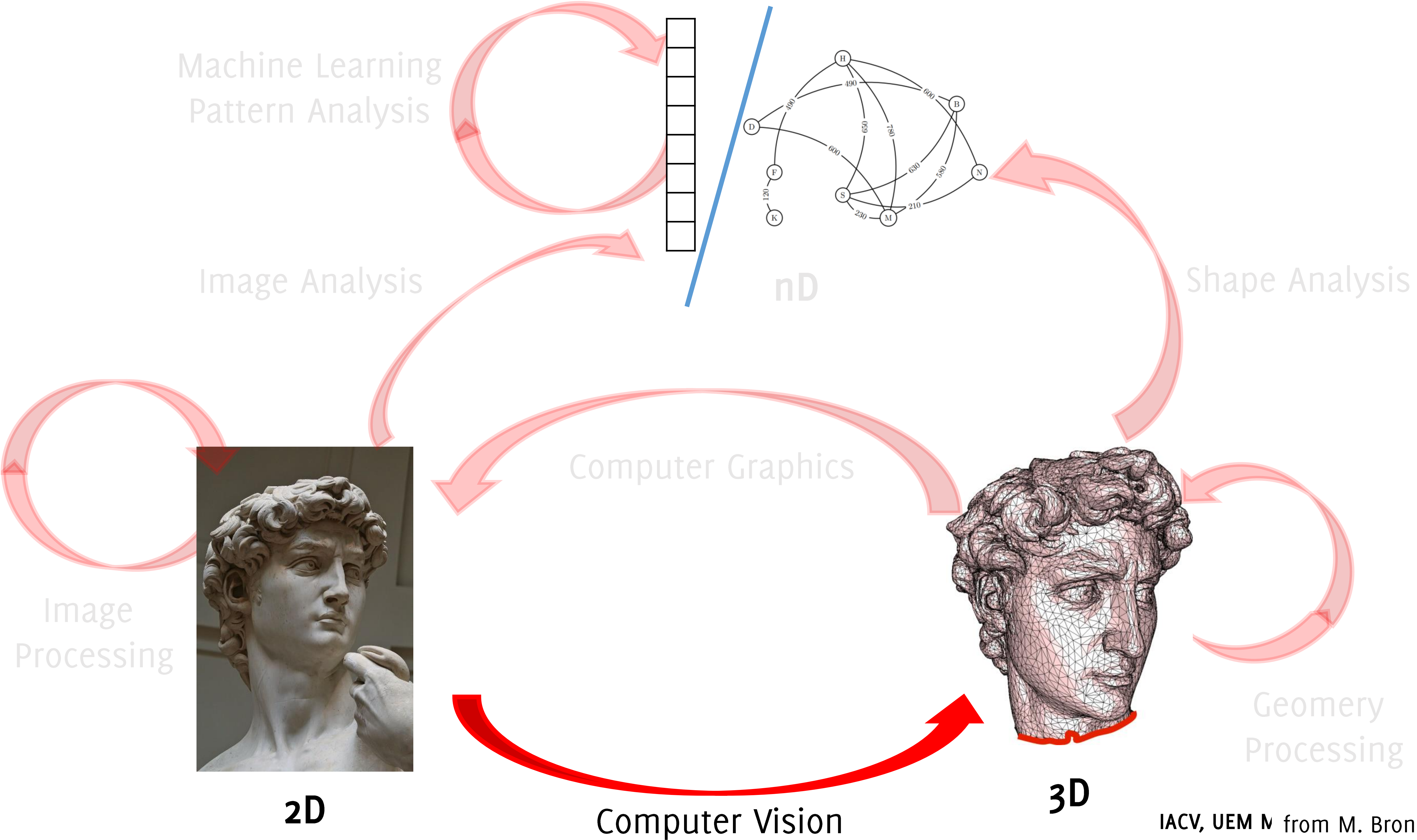
digitec
DIGITAL INNOVATION

Inpainting

Original damaged photo

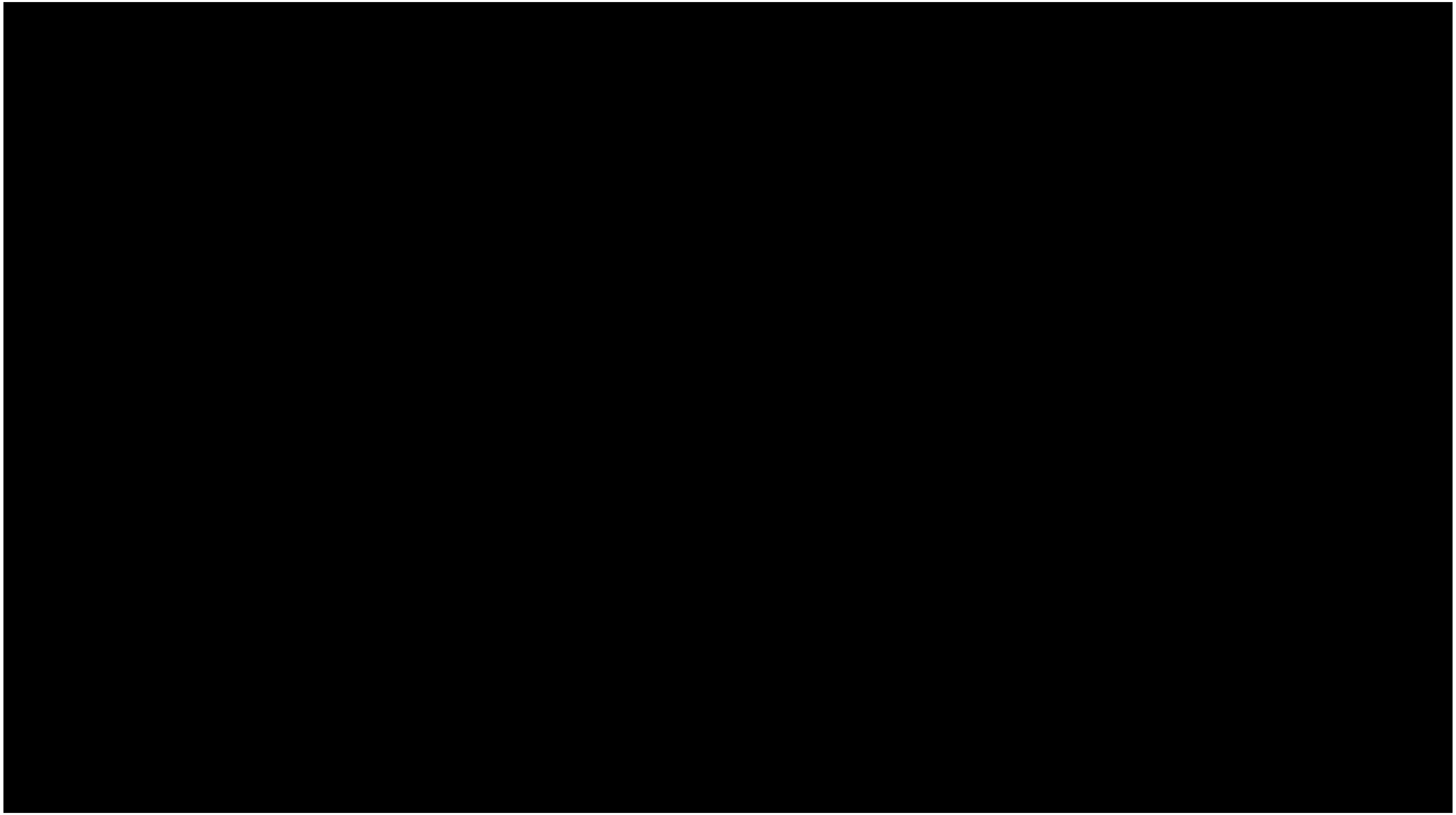
Restored photo

Bertalmio, M., Sapiro, G., Caselles, V., & Ballester, C. (2000, July). Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (pp. 417-424).



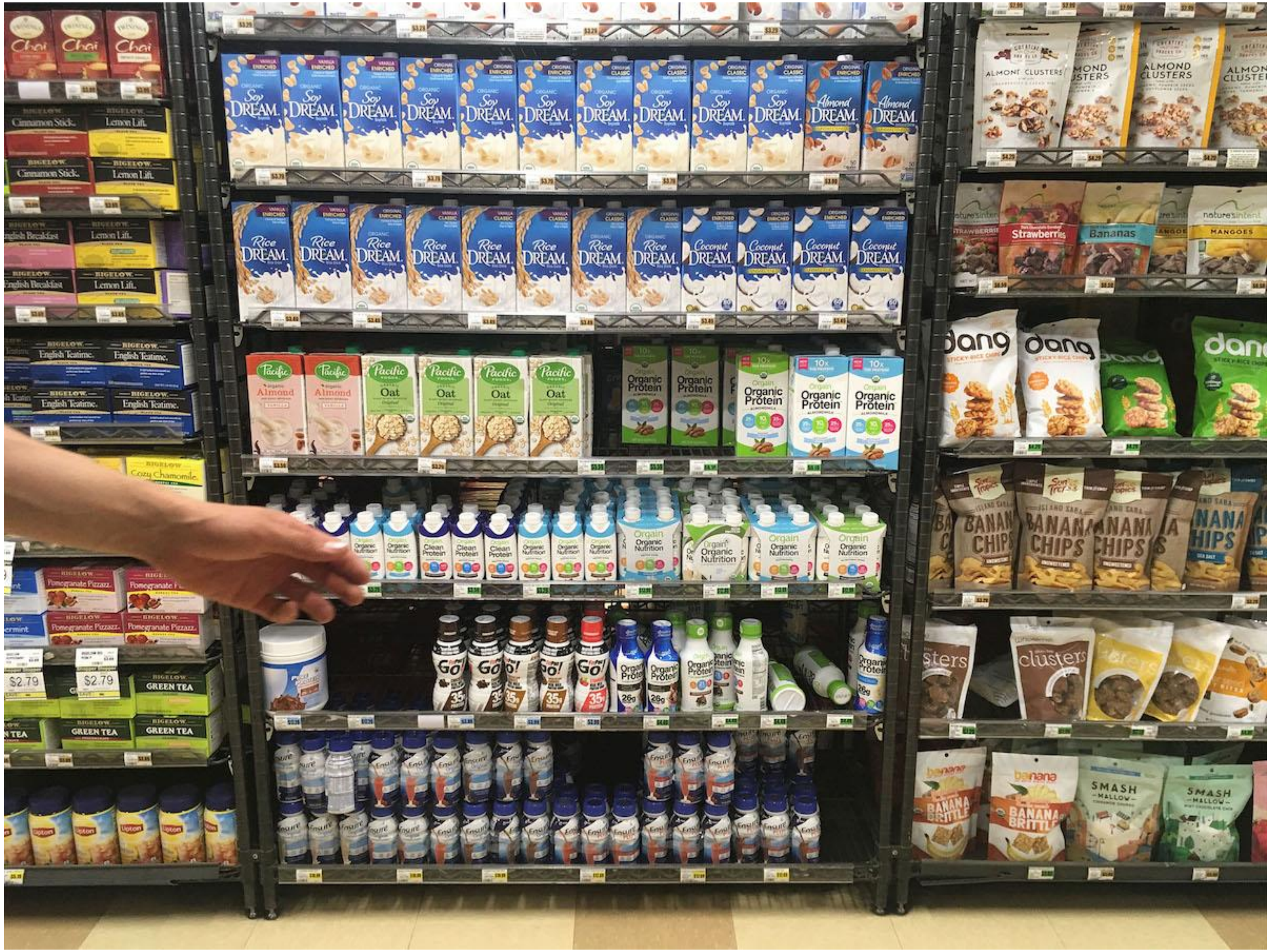
Machine Learning
Pattern Analysis
Image Analysis
nD
Shape Analysis
Image
Processing
Computer Graphics
Geomery
Processing
2D
3D
Computer Vision

3D Reconstruction



Autonomous Driving

Automatic Shelf Analysis

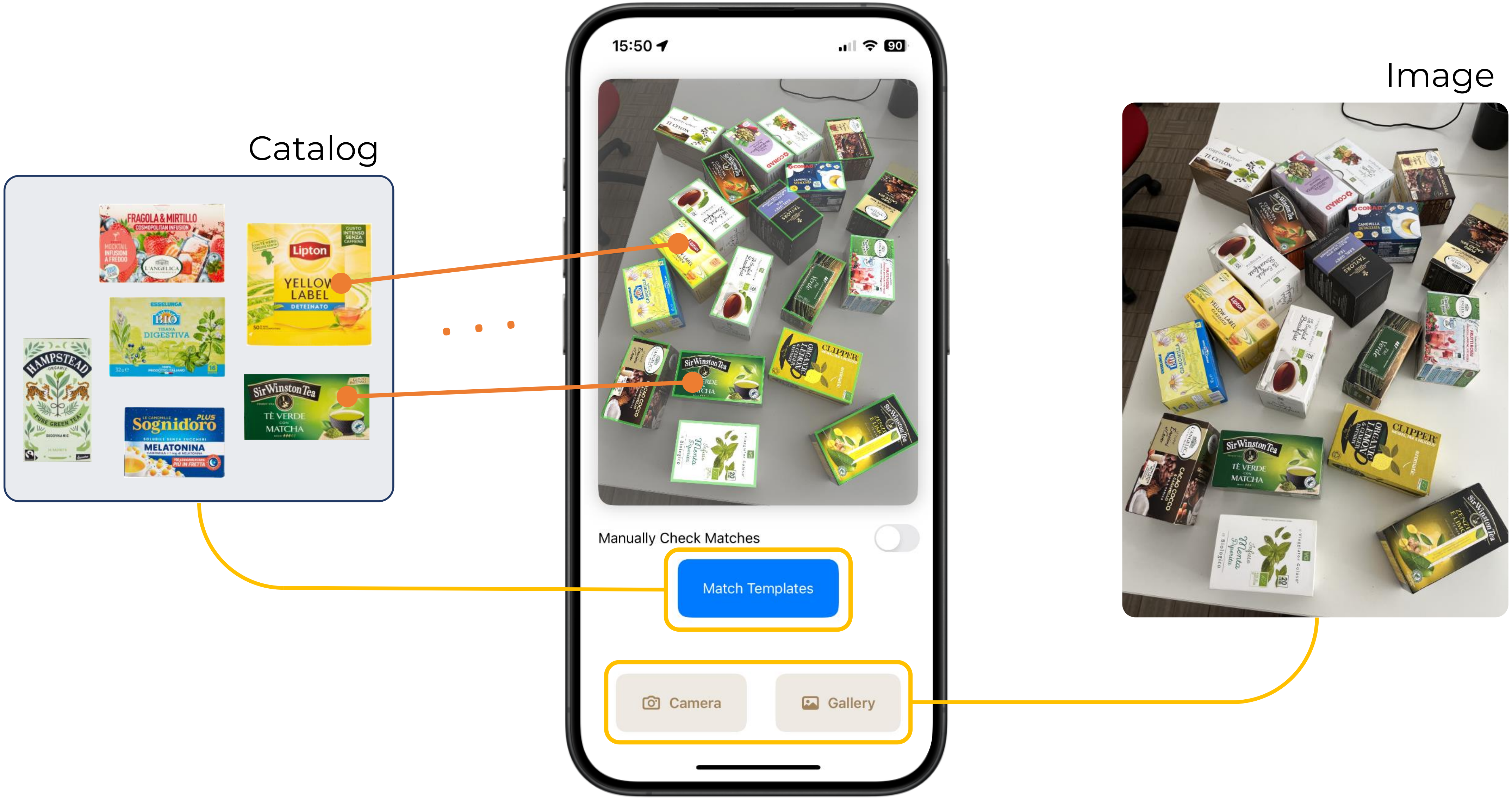


http:focal.systems

Automatic Shelf Analysis

http:focal.systems

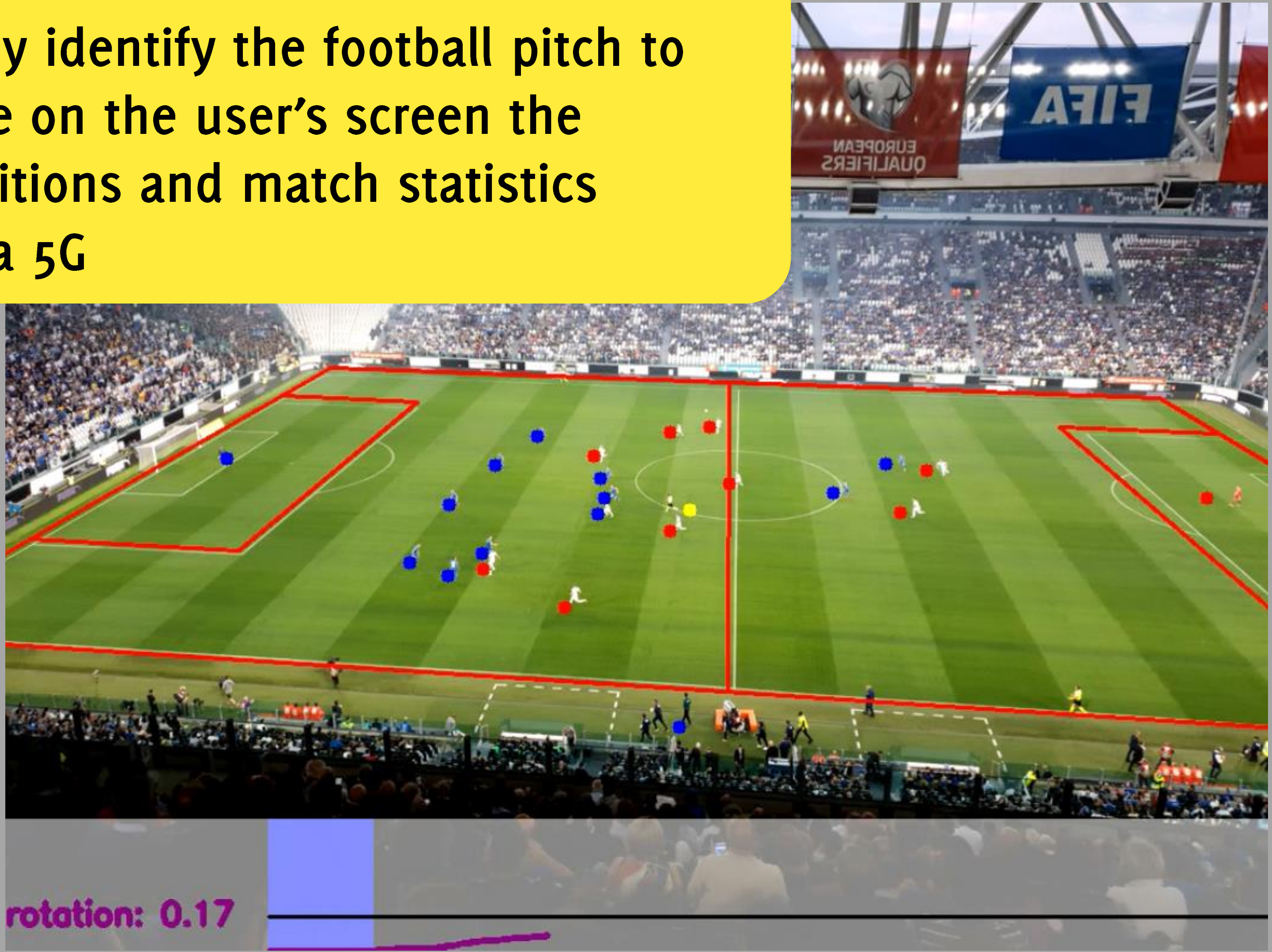
Template matching



math & sport

math&sport

Automatically identify the football pitch to superimpose on the user’s screen the players’ positions and match statistics streamed via 5G



Powered by

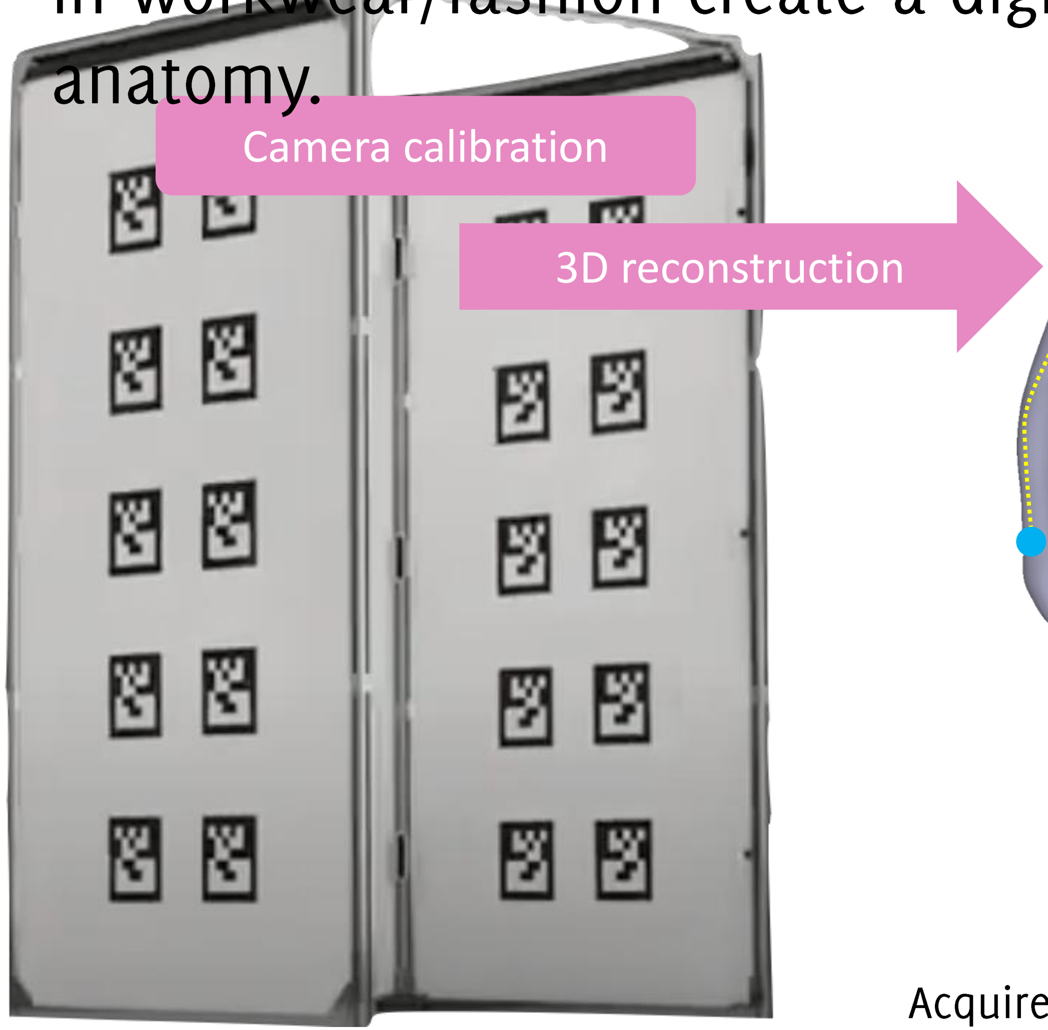
MOX OFF
MATHEMATICS FOR INNOVATION

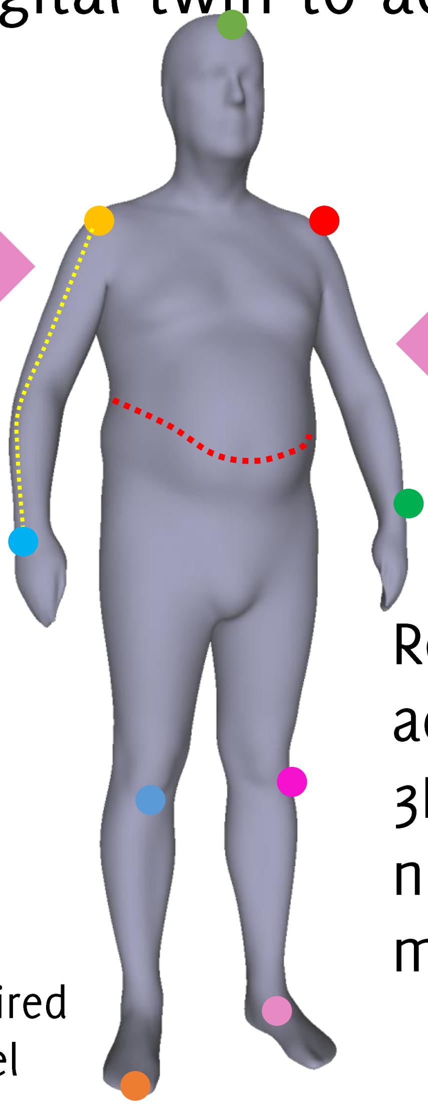
vodafone

Action for 5G,
il bando Vodafone
per le startup

3D body scanner for anthropometric measurements

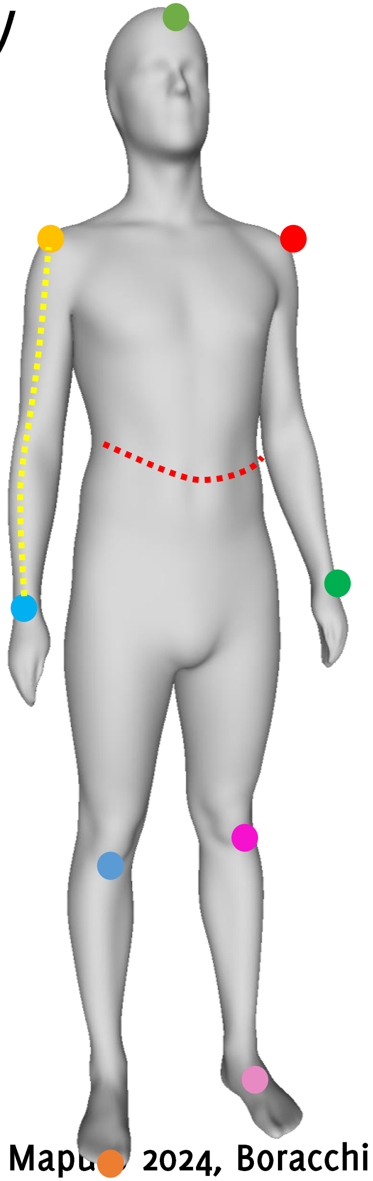
In workwear/fashion create a digital twin to accurately measure body anatomy.





Acquired model

Registration of the acquired model against a 3D template to identify notable points and take measurements

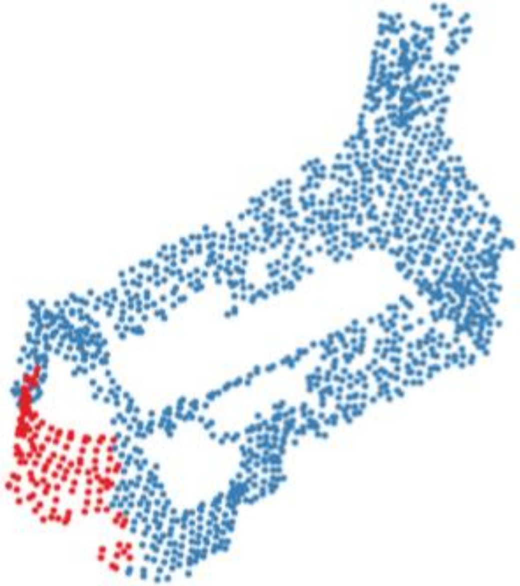


3D template

WE-WEAR the future of fashion

Model Fitting

Image of the scene

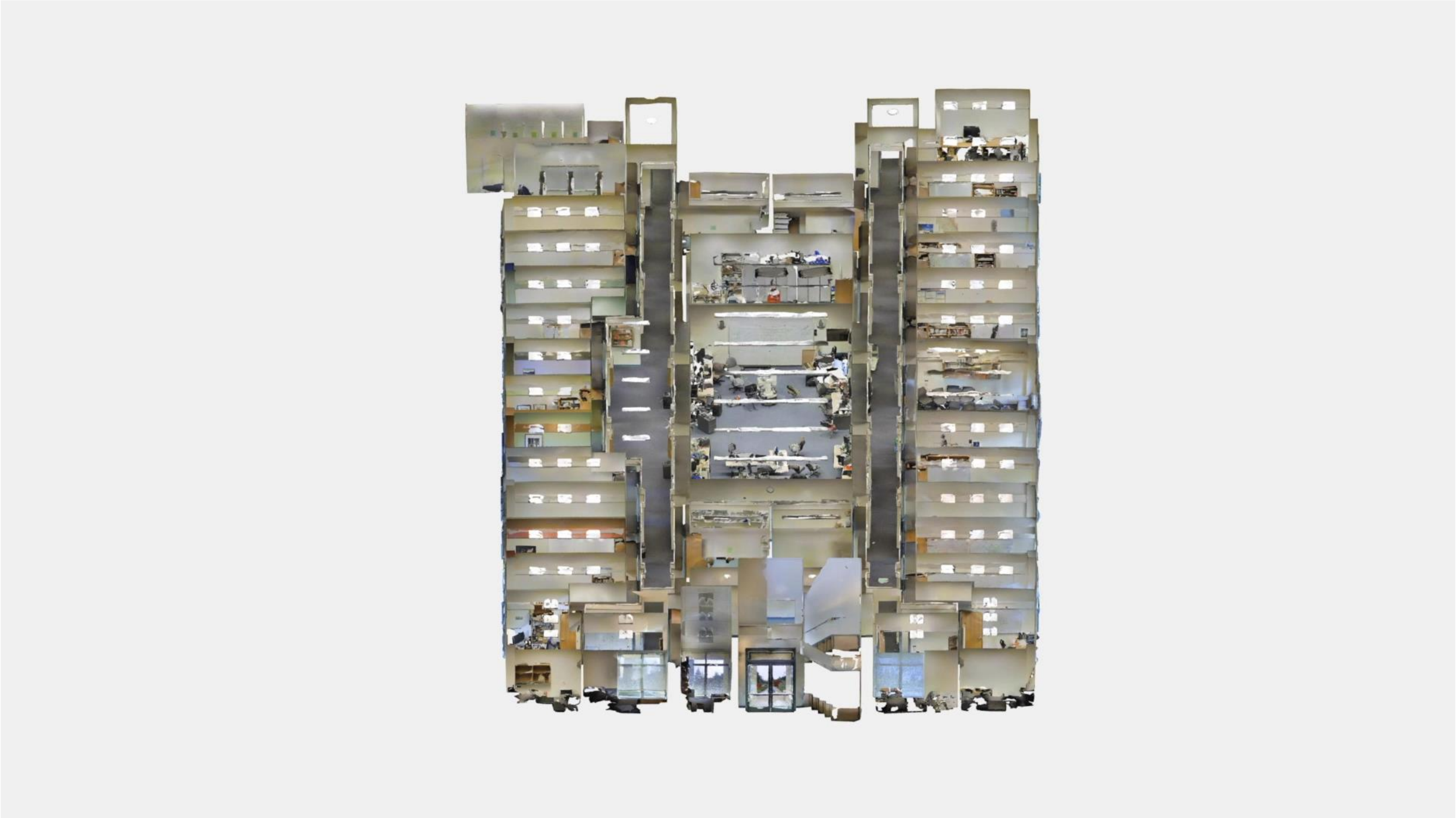


colour coded class

colour coded model

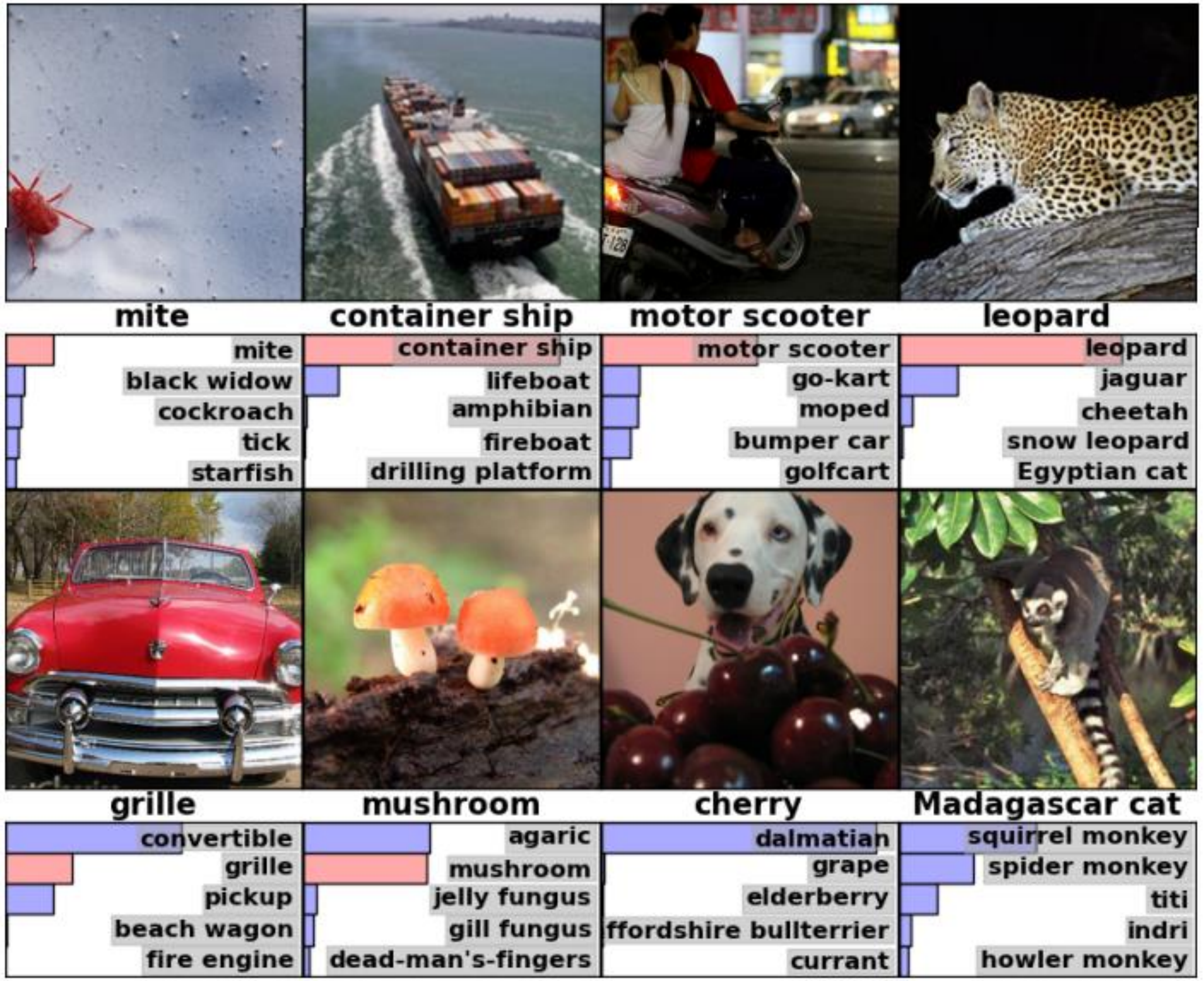
Magri, Luca, and Andrea Fusiello. "Fitting Multiple Heterogeneous Models by Multi-Class Cascaded T-Linkage." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

Scan2Bim



Let's look at some cool stuff you can do with Deep Neural Networks

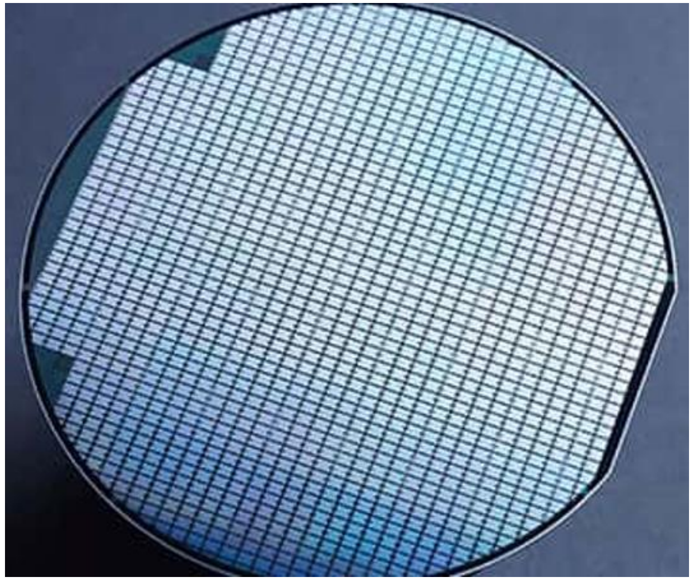
Image Classification on Imagenet



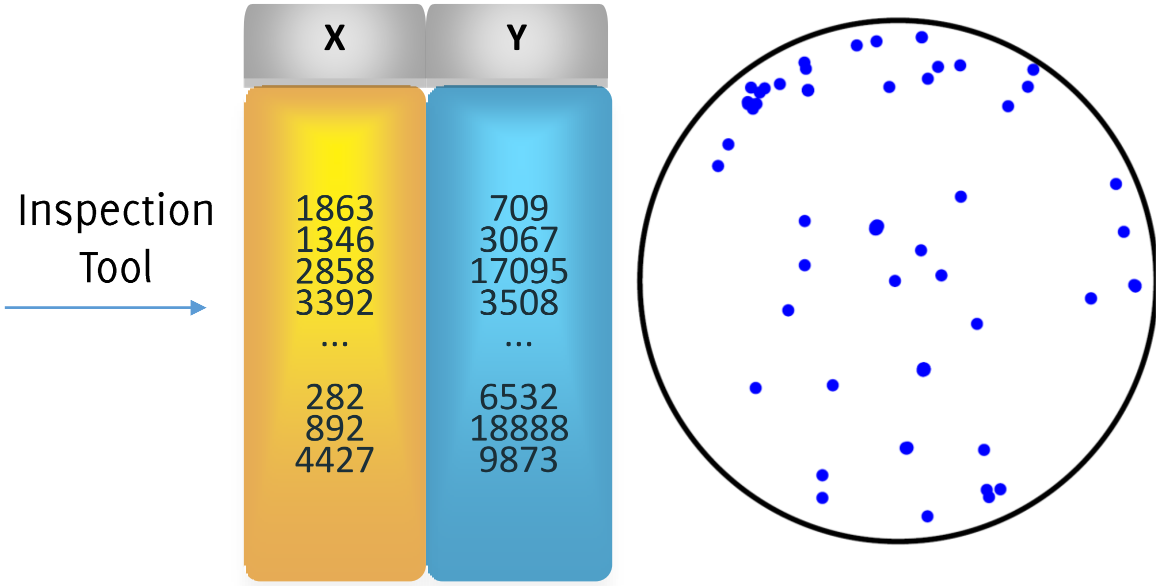
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012).

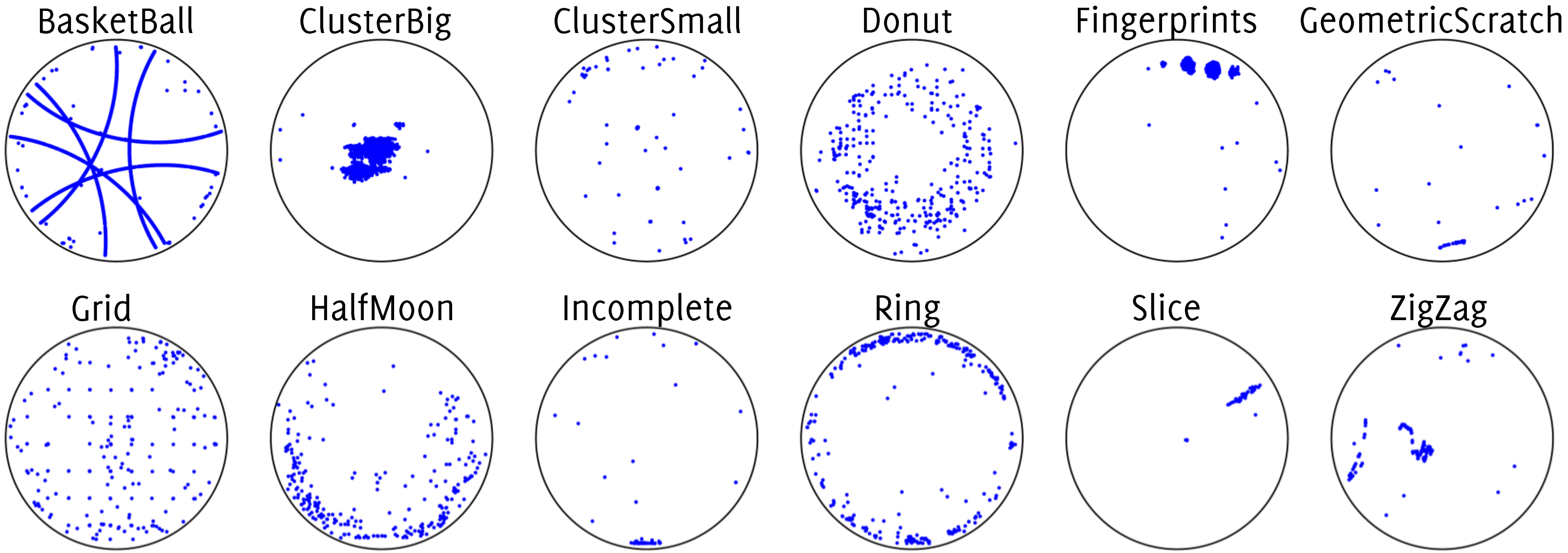
Image Classifcation

Wafer Defect Map (WDM)



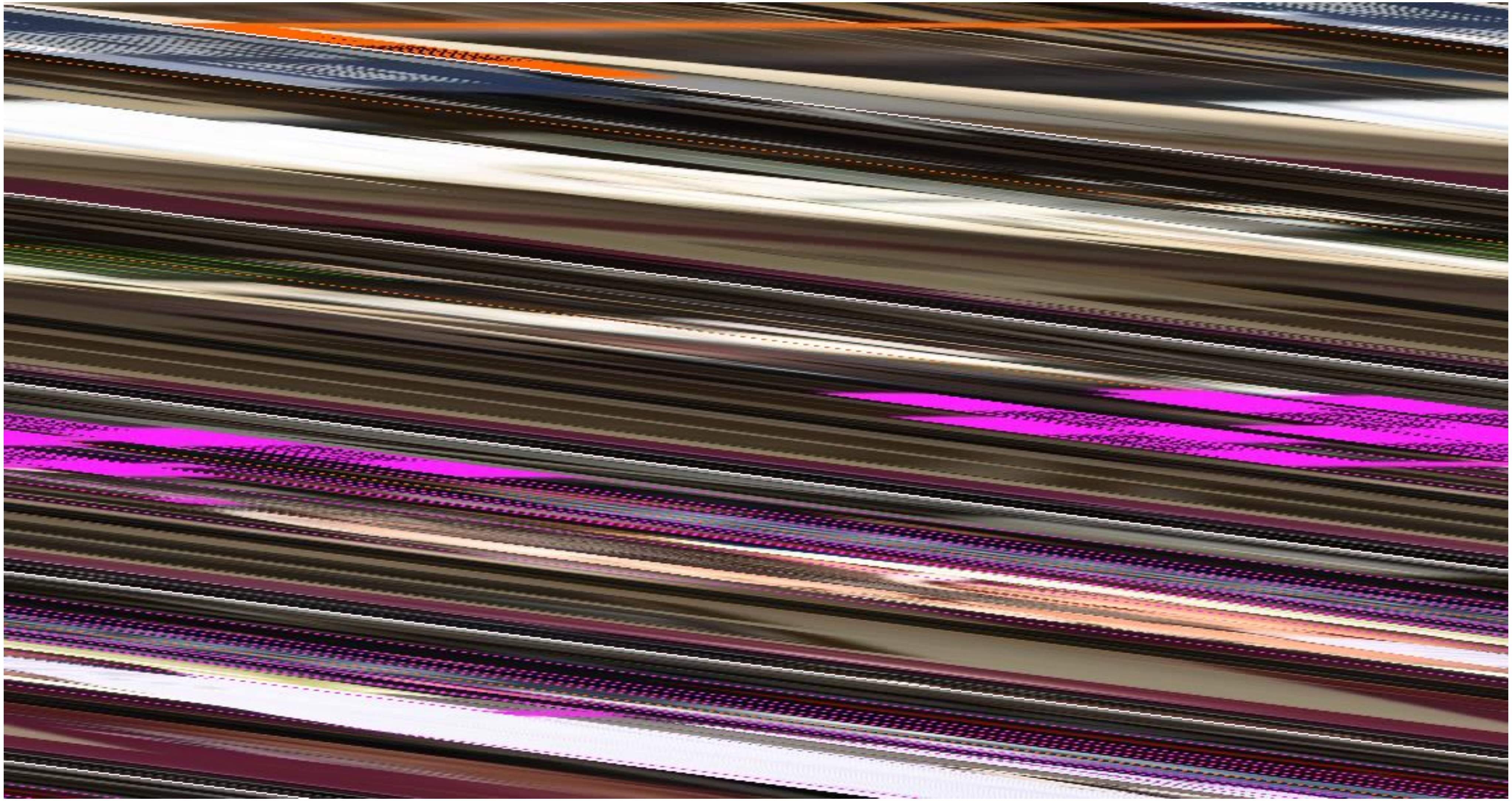
Silicon Wafer





Frittoli, Carrera, Rossi, Fragneto, Boracchi, *Deep Open-Set Recognition for Silicon Wafer Production Monitoring*, Pattern Recognition 2021

Object Detection



Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

Pose Estimation

Cao, Z., Simon, T., Wei, S. E., & Sheikh, Y. (2017). Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7291-7299).

Image Segmentation

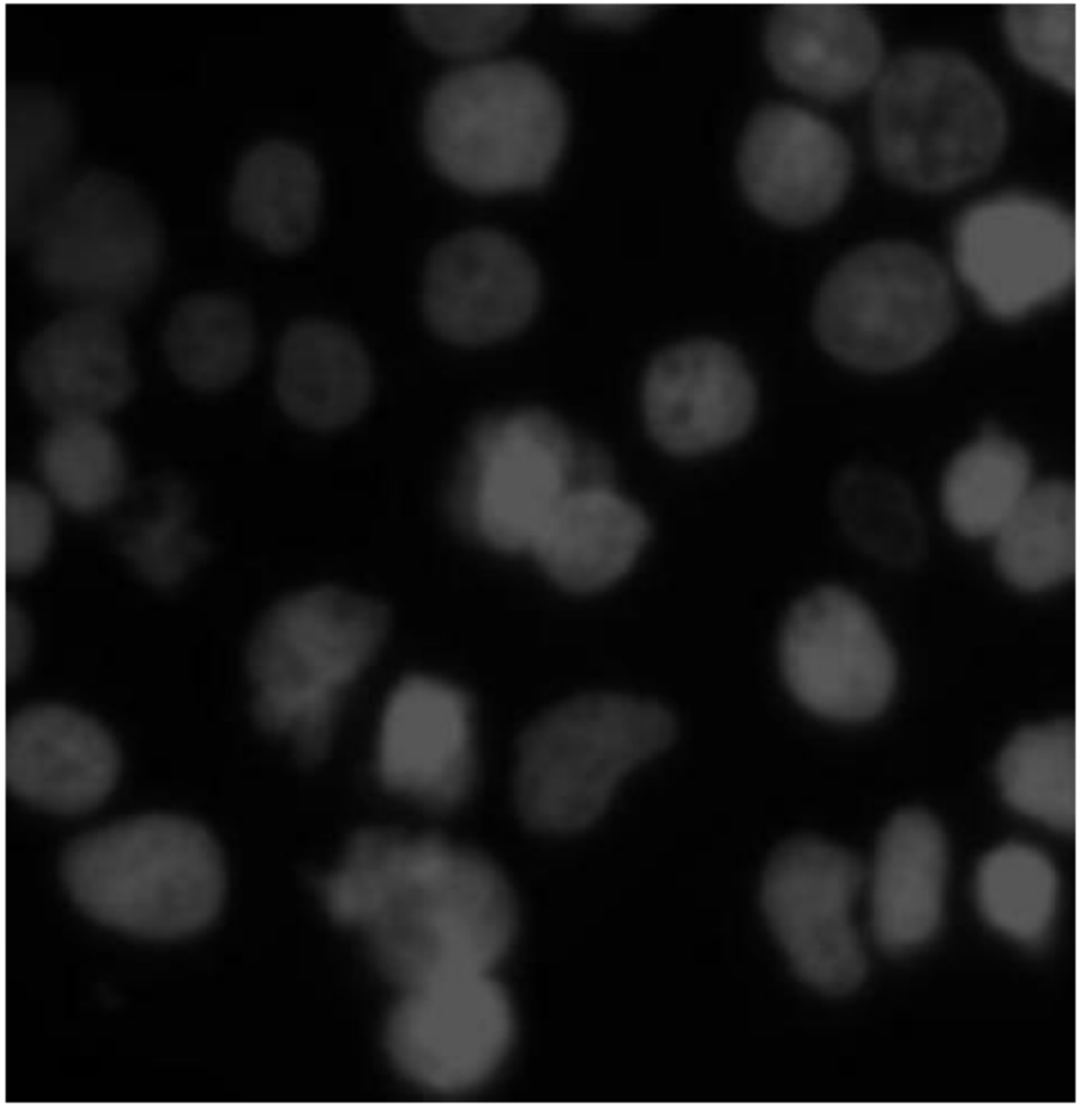
Zheng et al. “Conditional Random Fields as Recurrent Neural Networks”, ICCV 2015

Instance Segmentation

Instance Segmentation

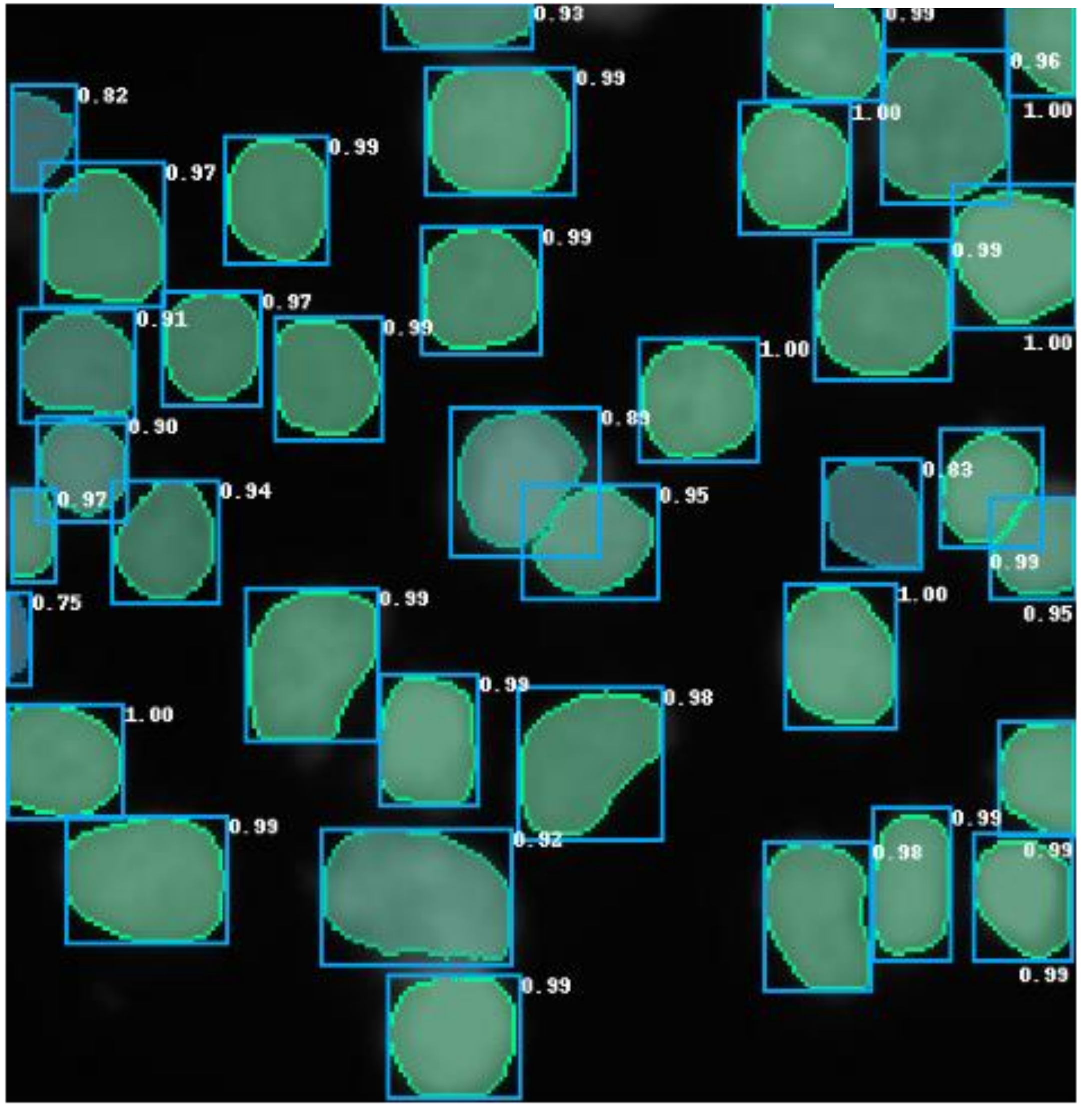
In collaboration with
ikonisys
FINDING THE CELLS THAT MATTER

Base image



Image

Prediction overlay with the base image



Prediction

Credits Roberto Basla

Image Searches

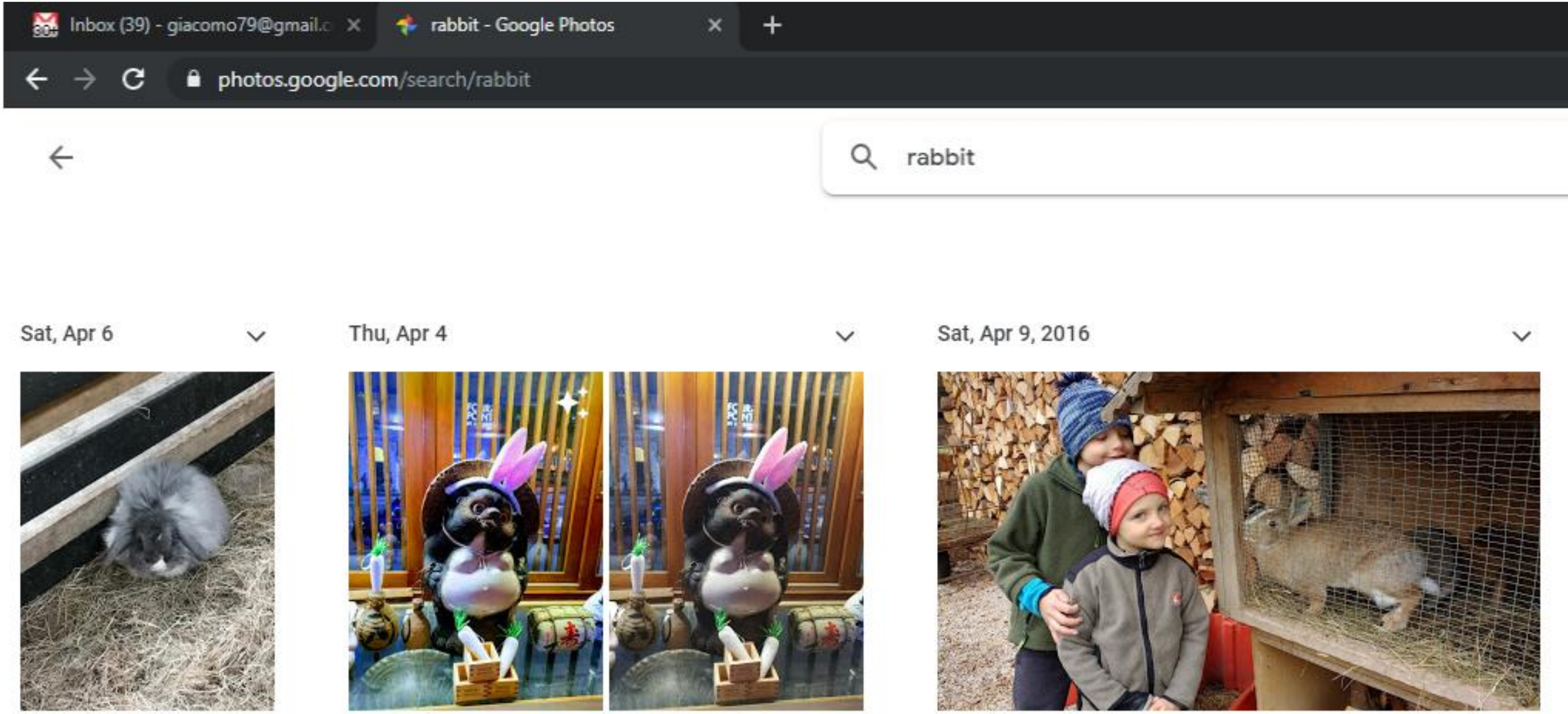
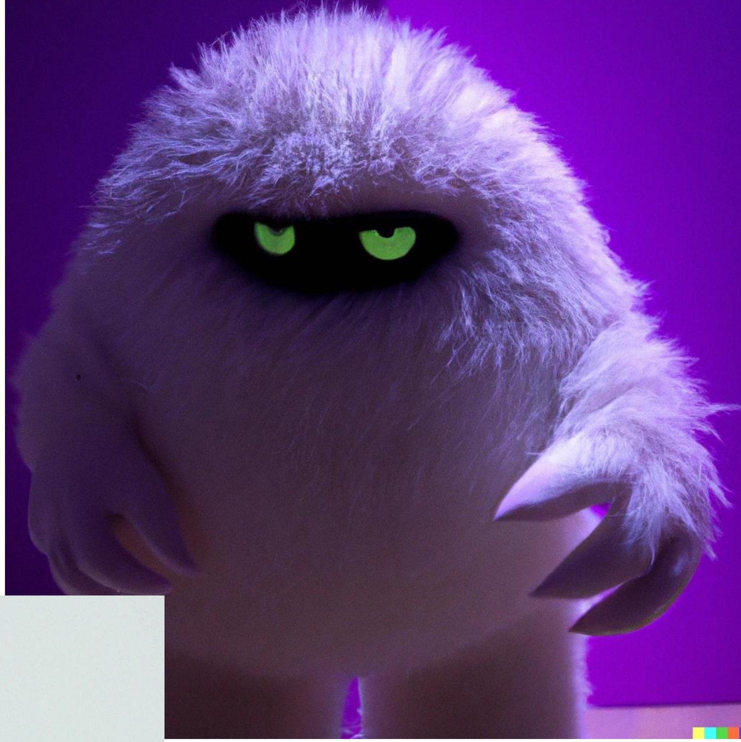


Image Generation

Tero Karras, Samuli Laine, Timo Aila «A Style-Based Generator Architecture for Generative Adversarial Networks" CVPR 2019

Image Generation

A photo of a white fur monster standing in a purple room



A van Gogh style painting of an American football player

A handpalm with a tree growing on top of it

A hand drawn sketch of a Porsche 911

https://openai.com/dall-e-2/

Course syllabus

- Basics of digital images, the image formation process.
- Basics of image transformations and image filtering
- Extracting Edges, Lines and Segments
- Extracting Salient Points and Matching Features

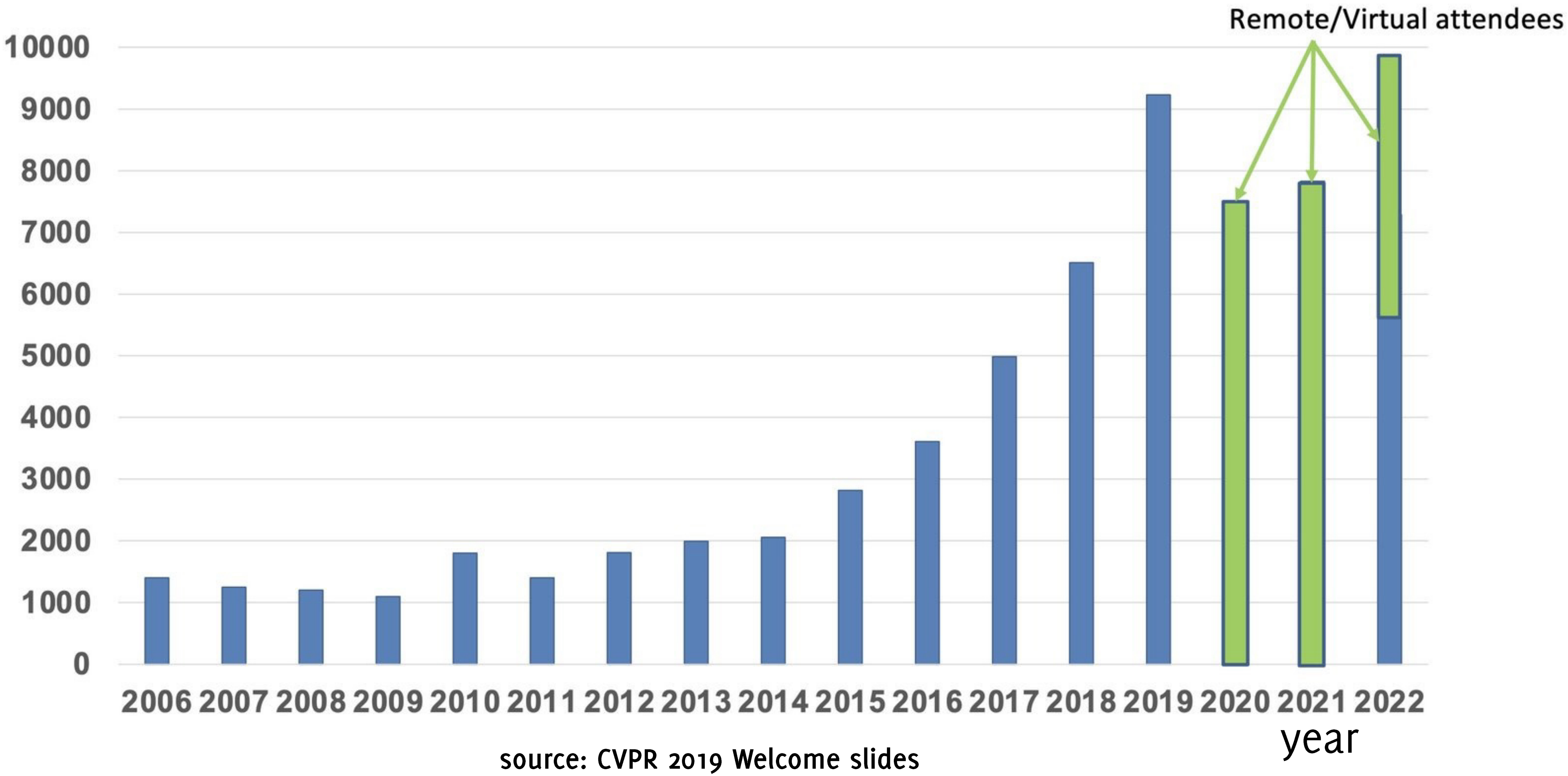
Image Processing

- The Image Classification Problem and the Deep Learning Revolution
- Convolutional Neural Networks
- Famous CNN architectures,
- Best practices in CNN training
- Advanced Visual Recognition Problems: Semantic Segmentation, Object Detection

Machine Learning

Is it worth to take this course?

CVPR Attendance Trend (as of June 20, 2022)



source: CVPR 2019 Welcome slides

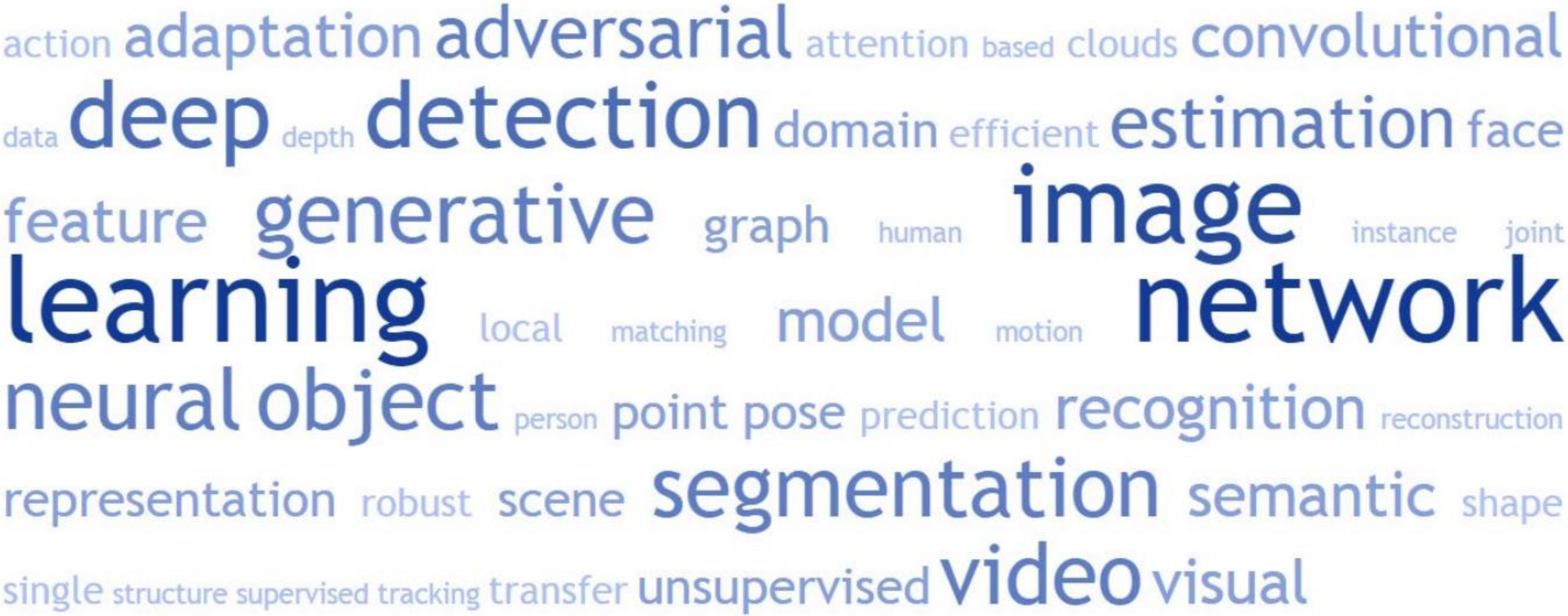
Lately, connection with ML

There has seen a dramatic change in CV:

- Once, most of techniques and algorithms **build upon a mathematical/statistical** description of **images**
- Nowadays, **machine-learning** methods are much more popular

1,294 papers in CVPR'19 (25.2% acceptance rate)

Acceptance rate is roughly even across topics



CVPR 2023 - top 10

1	3D from multi-view and sensors	1,090	246
2	Image and video synthesis and generation	889	185
3	Humans: Face, body, pose, gesture, movement	813	166
4	Transfer, meta, low-shot, continual, or long-tail learning	688	153
5	Recognition: Categorization, detection, retrieval	673	139
6	Vision, language, and reasoning	631	118
7	Low-level vision	553	126
8	Segmentation, grouping and shape analysis	524	113
9	Deep learning architectures and techniques	485	92
10	Multi-modal learning	450	89

All in all…

If you plan pursuing a research-oriented career:

- If you go towards ML and/or CV, consider these fields are increasingly contaminated
- In any case a strong background in Computer Vision and Pattern Recognition is definitively a plus considering the widespread use of imaging data and the need of automation
- This course is a good way to practice more fundamental theory aspects related to neural networks
- Companies are increasinly interested in Computer Vision and Pattern Recogntion appliction

Let's start:
Digital Images


Photo Credits: Andrea Sanfilippo

RGB Images

$I \in \mathbb{R}^{R \times C \times 3}$

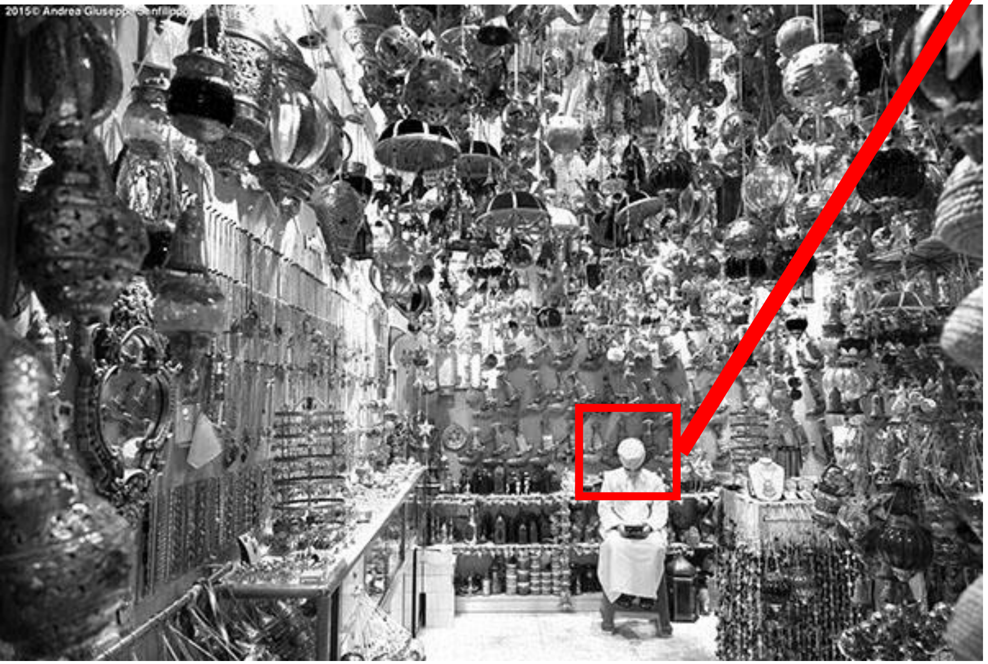
$R \in \mathbb{R}^{R \times C}$

$G \in \mathbb{R}^{R \times C}$

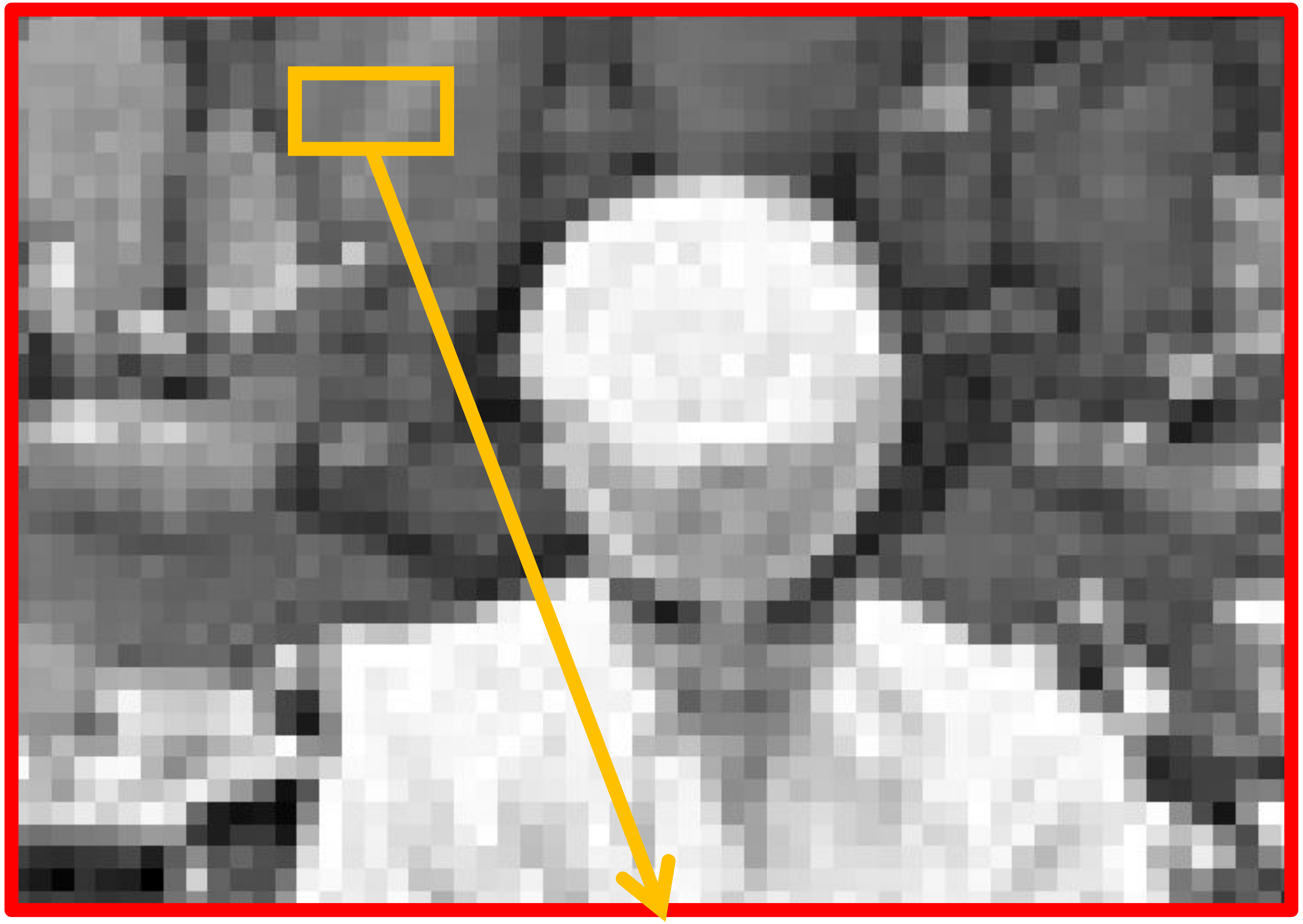
$B \in \mathbb{R}^{R \times C}$

RGB Images

Images are saved by encoding each color information in 8 bits. So images are rescaled and casted in [0,255]



$R \in \mathbb{R}^{R \times C}$



123	122	134	121	132
122	121	125	132	124
119	127	137	119	139

RGB Images

[0, 205, 155]

[15, 17, 19]

[230,234, 233]

[253, 5, 6]

This is an RGB triplet
R = 253; G = 2; B = 6

[106,124,138]

The Input of our Neural Network!

Three dimensional arrays $I \in \mathbb{R}^{R \times C \times 3}$

```
from skimage.io import imread

# Read the image
I = imread('bazar.jpg')

# Extract the color channels

R = I[:, :, 0]
G = I[:, :, 1]
B = I[:, :, 2]
```

```
% Matlab
R = I(:, :, 1)
G = I(:, :, 2)
B = I(:, :, 3)
```

When loaded in memory, image sizes are much larger than on the disk where images are typically compressed (e.g. in jpeg format)

Videos

Higher dimensional images

Videos are sequences of images (frames)

If a frame is

$$I \in \mathbb{R}^{R \times C \times 3}$$

a video of T frames is

$$V \in \mathbb{R}^{R \times C \times 3 \times T}$$

```
print(V.shape)
  (144, 180, 3, 30)
```

In this example: $R = 144, C = 180$, thus these 5 color frames contains: 388.800 values in [0,255], thus in principle, 388 KB

Dimension Increases very quickly

Without compression: 1Byte per color per pixel

1 frame in full HD: $R = 1080, C = 1920 \approx 6MB$

1 sec in full HD (24fps) $\approx 150MB$

Fortunately, visual data are very redundant, thus compressible

This has to be **taken into account when you design a Machine learning algorithm** for images or vides

- e.g. **during training** a neural network these information are **not compressed**!

Photometric Image Formation

Giacomo Boracchi

giacomo.boracchi@unibocconi.it

Image Analysis and Computer Vision

UEM, Maputo

https://boracchi.faculty.polimi.it

Colour Filter Arrays

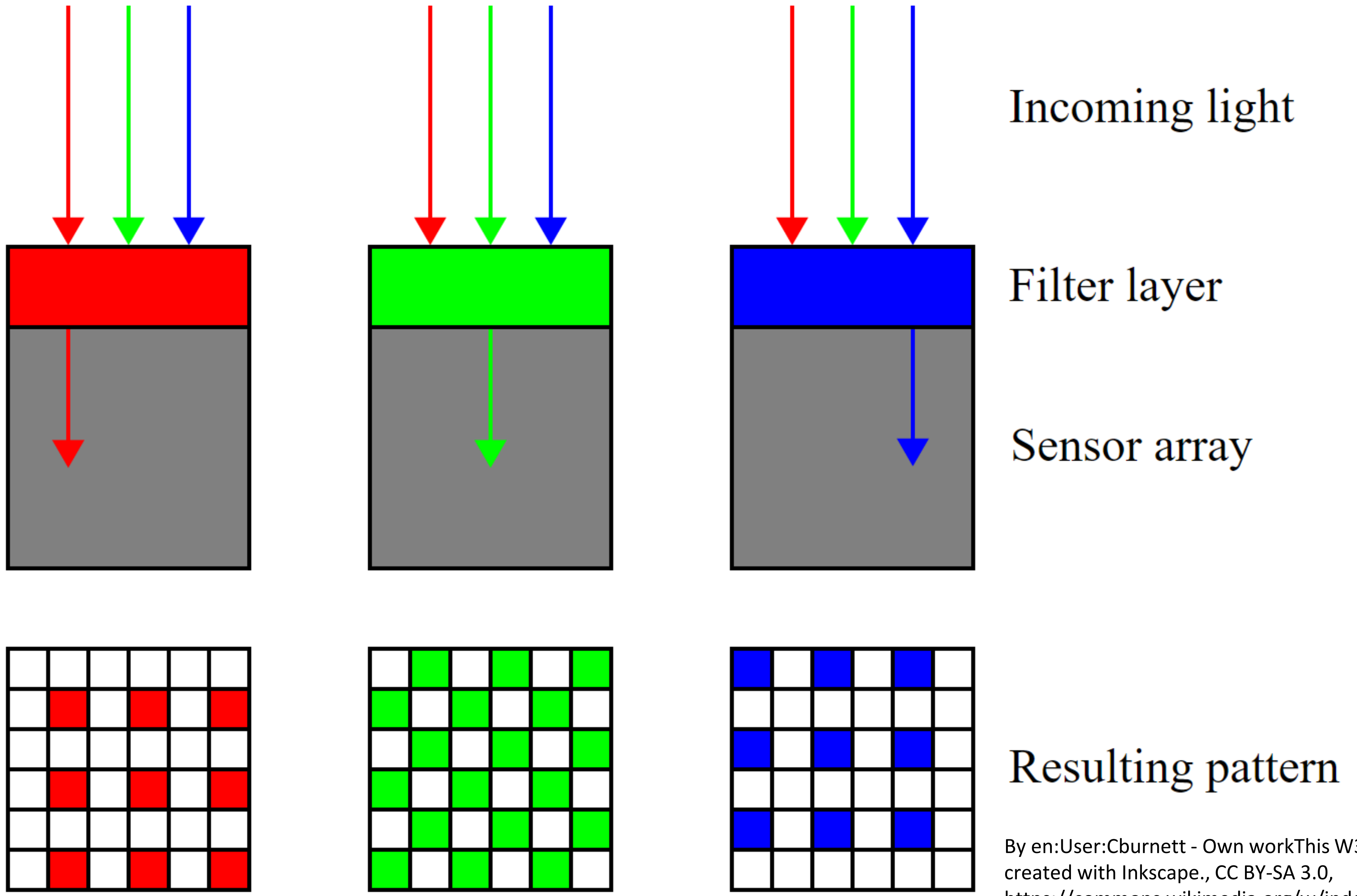
Typical **photosensors** detect light intensity with little or **no wavelength specificity**, and therefore cannot separate colour information.

Colour Filters Array (CFA) are used to filter the light by wavelength range.

Separate filtered intensities include information about the colour of light.

For example, the Bayer filter gives information about the intensity of light in red, green, and blue (RGB) wavelength regions

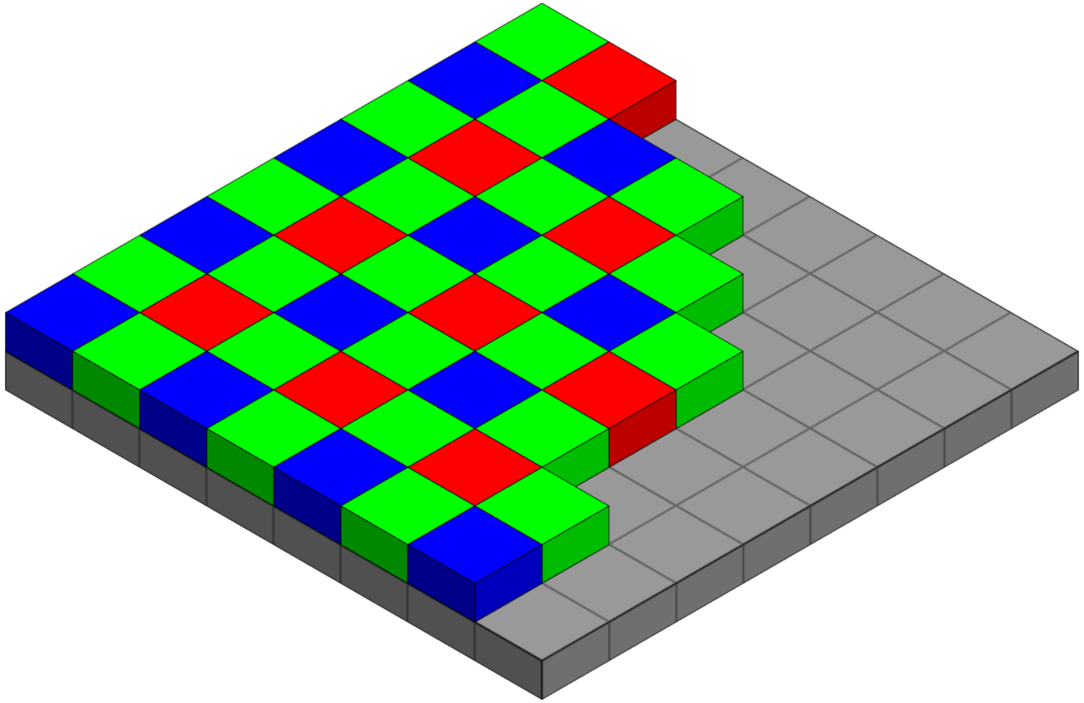
Colour Filter Arrays



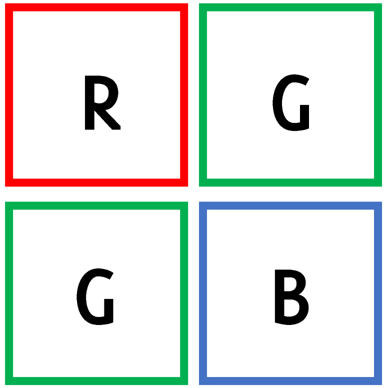
Bayer Pattern

For example, the Bayer filter (RGGB) gives information about the intensity of light in red, green, and blue wavelength regions.

- Green colour is sampled twice



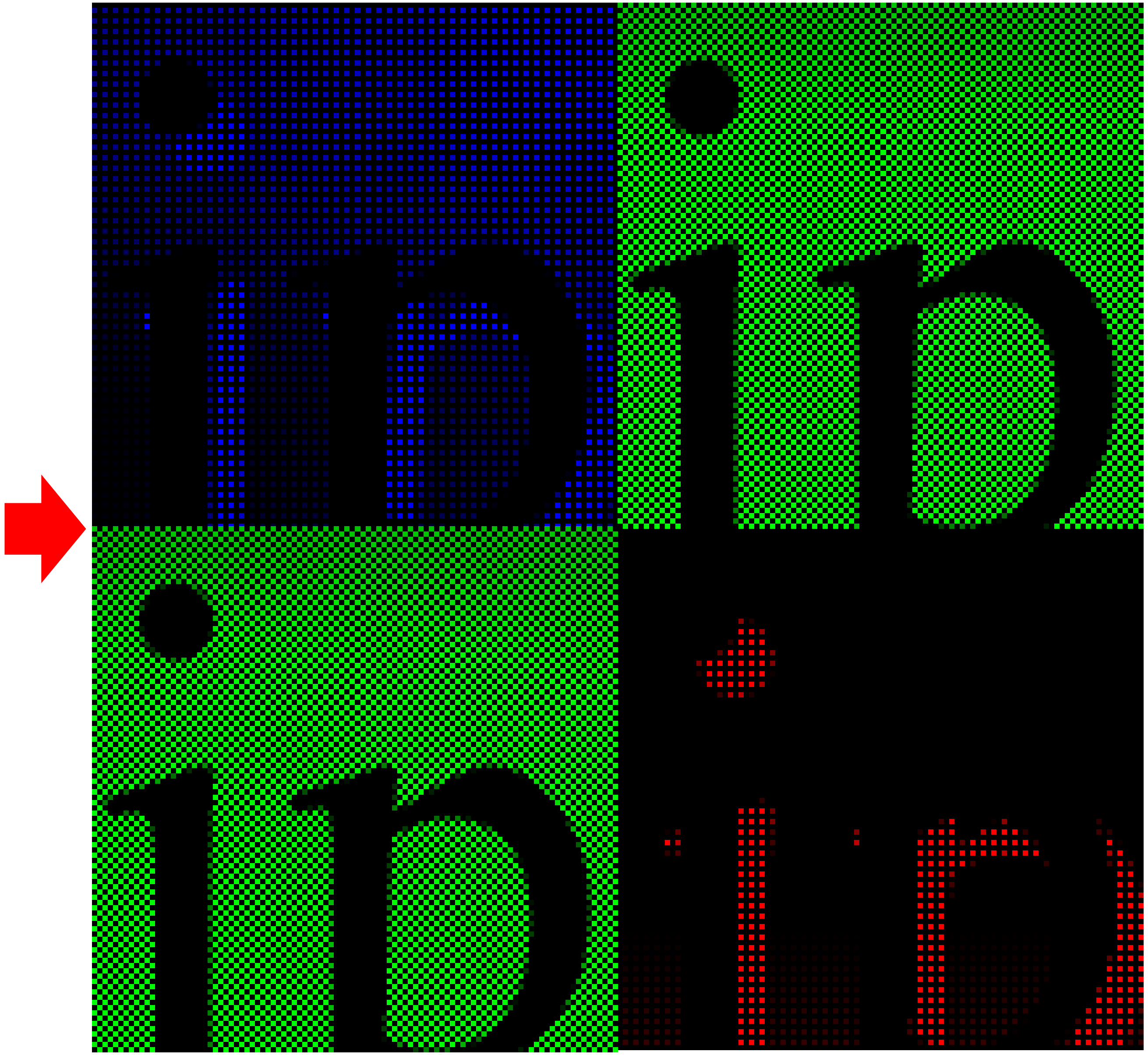
There are many different patterns, including RYYB which gives a better response in low-light conditions



The raw output of digital camera

Every pixel of the array is only sensitive to a single colour.

The raw output of digital camera

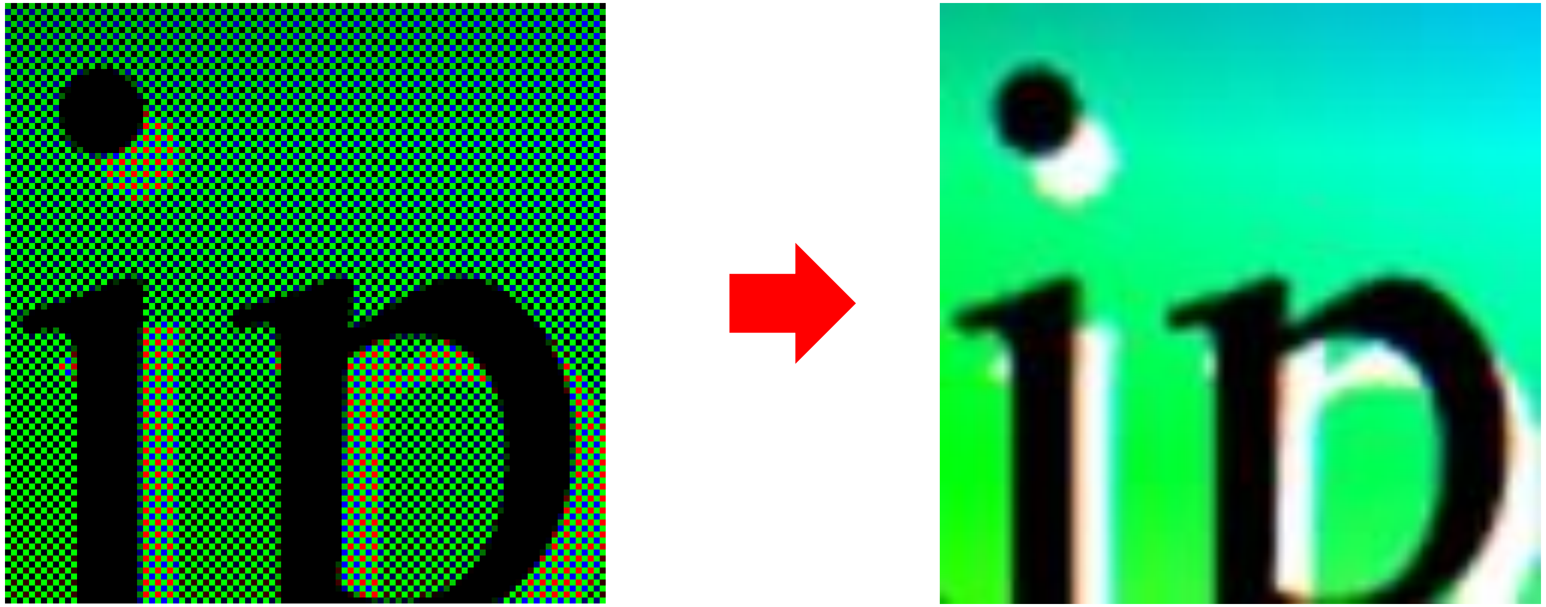


Demosaicing

Demosaicing, a.k.a. CFA interpolation or Colour Reconstruction

Algorithm to **reconstruct a full colour image** (3 colours per pixel) from the **incomplete colour output** from an image sensor (CFA).

This is a **multivariate regression problem**



Demosaicing

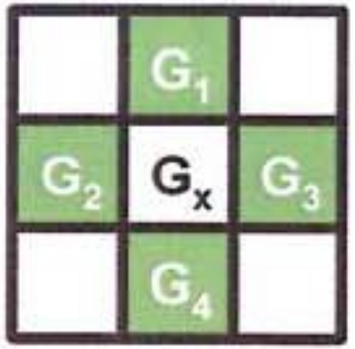
Issues:

- In **Bayer pattern each pixel is sensitive to a single colour**, while in the image each pixel portrays a mixture of 3 primary colours

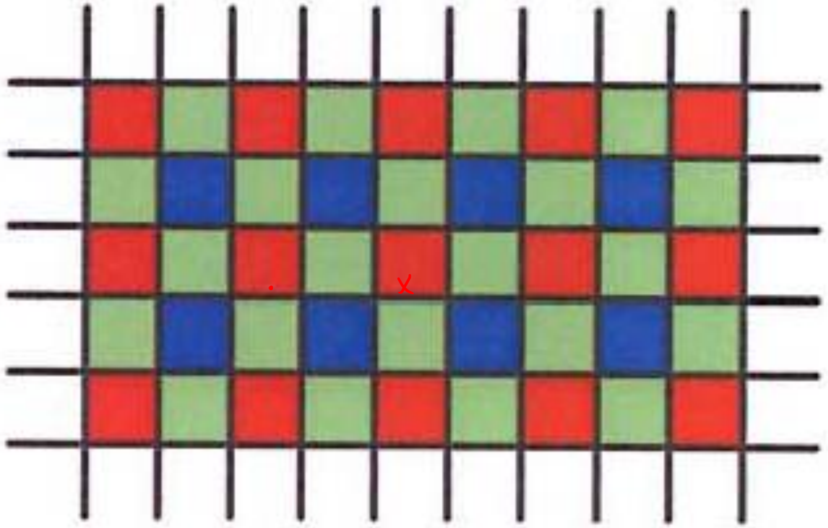
Desiderata:

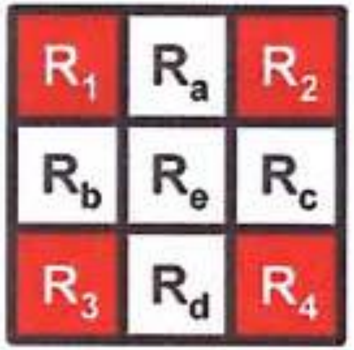
- Avoid colour artefacts
- Maximum preservation of the image resolution
- Low complexity or efficient in-camera hardware implementation
- Amenability to analysis for accurate noise reduction

Example of Demosaicing by bilinear interpolation

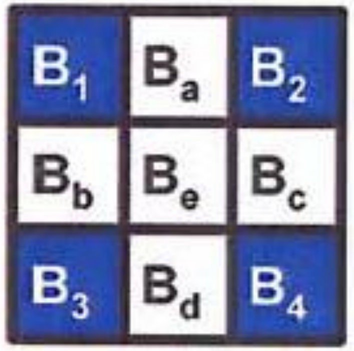


$$G_x = (G_1+G_2+G_3+G_4)/4$$





$$R_a = (R_1+R_2)/2$$
$$R_b = (R_1+R_3)/2$$
$$R_c = (R_2+R_4)/2$$
$$R_d = (R_3+R_4)/2$$
$$R_e = (R_1+R_2+R_3+R_4)/4$$



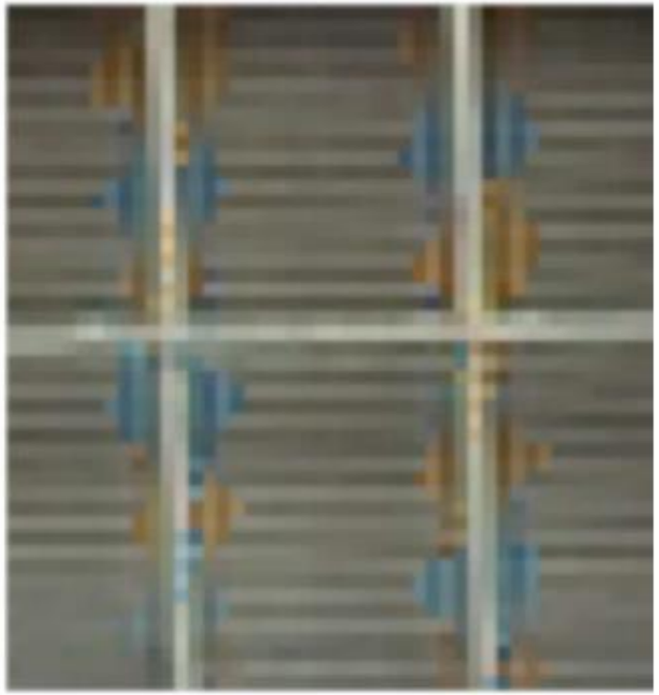
$$B_a = (B_1+B_2)/2$$
$$B_b = (B_1+B_3)/2$$
$$B_c = (B_2+B_4)/2$$
$$B_d = (B_3+B_4)/2$$
$$B_e = (B_1+B_2+B_3+B_4)/4$$

More sophisticated channel-wise interpolation include bicubic/spline interpolation, Lanczos resampling

Demosaicing

Color-independent algorithms typically present artifacts in regions containing edges and textures

Zipper effects *are unnatural changes of intensities over a number of neighboring pixels, manifesting as an "on-off" pattern in regions around edges*



False colors *are spurious colors which are not present in the original image scene [...] They appear as sudden hue changes due to inconsistency among the three color planes and usually around fine image details and edges*

Lanlan Chang and Yap-Peng Tan "*Hybrid color filter array demosaicking for effective artifact suppression*" JEI 2006

False Colors

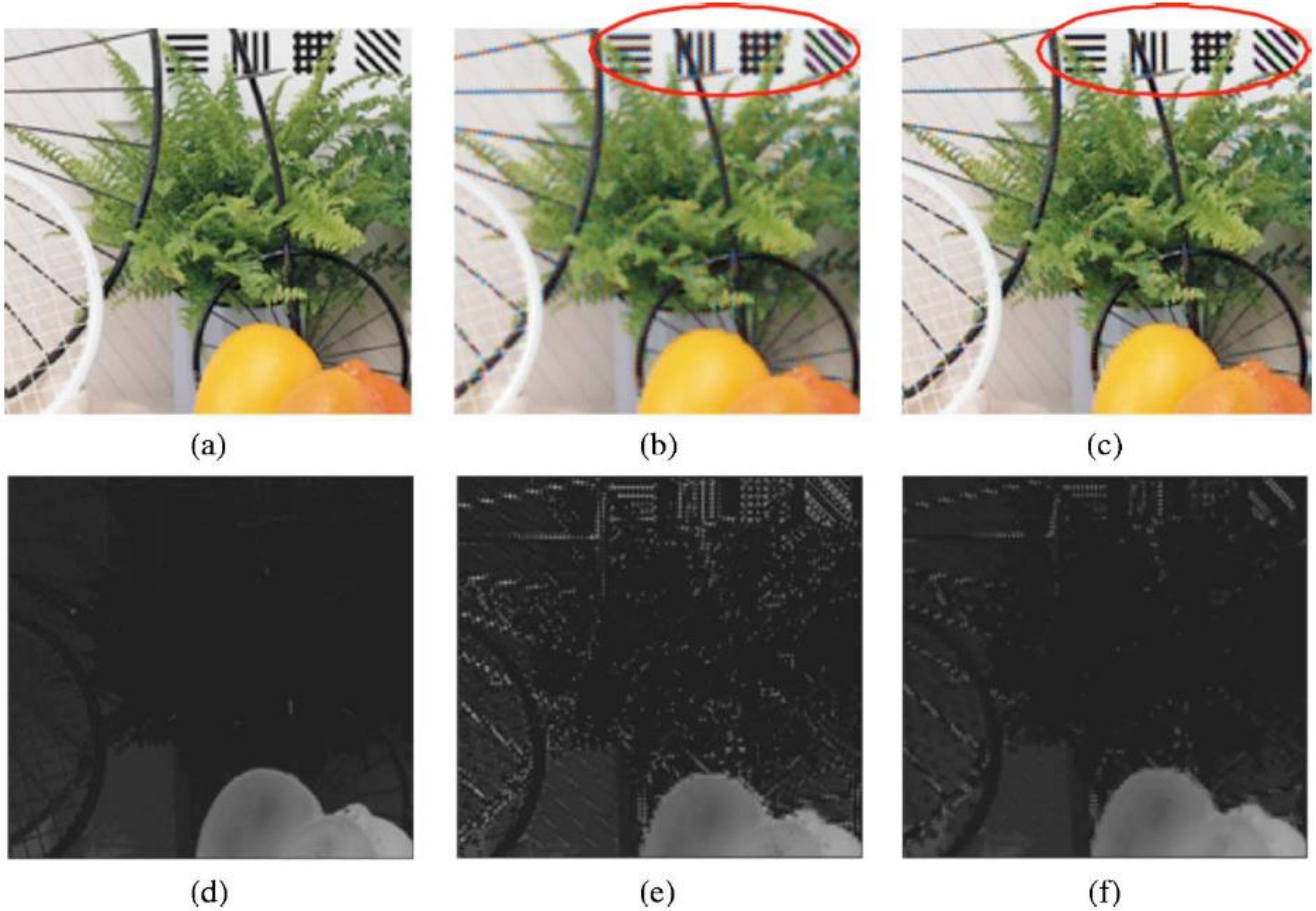


Fig. 10 Original image region (from image 27 in Fig. 15) and its demosaicked results obtained by (b) bilinear interpolation and (c) Freeman's method. The corresponding color difference planes (green minus red) are shown in (d), (e), and (f), respectively.

Lanlan Chang and Yap-Peng Tan *"Hybrid color filter array demosaicking for effective artifact suppression"* JEI 2006

Demosaicing

Examples of priors to be exploited to improve demosaicing quality

- Channel-wise similarity / consistency (colour differences, colour ratio)
- Spatial correlation, the structure of images
- Spectral correlation

Post-processing can be employed to suppress typical demosaicing artifacts

Basics of Image Manipulation

Giacomo Boracchi

giacomo.boracchi@unibocconi.it

Image Analysis and Computer Vision

UEM, Maputo

https://boracchi.faculty.polimi.it

When we work channel-wise…

Think of an image as a 2D, real-valued function

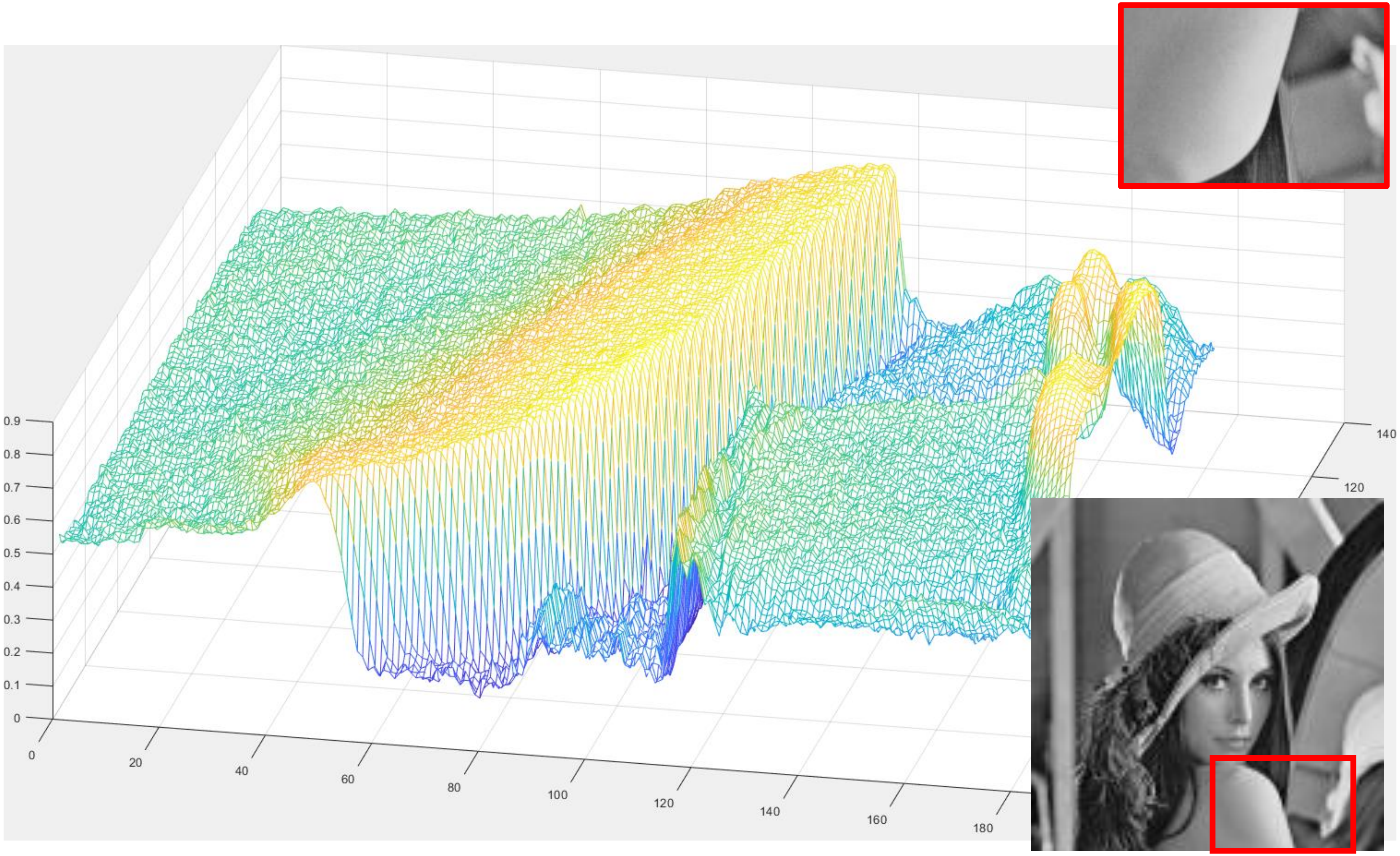


Image Manipulation

It is possible to operate on images as matrices.

For example, we can generate a matrix that can be displayed as an image

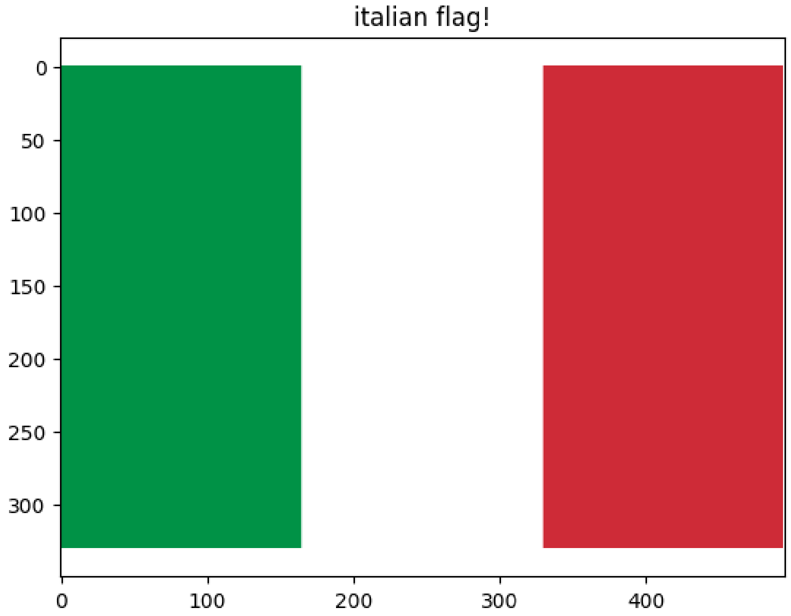


Image "generation"

Colors in Palette

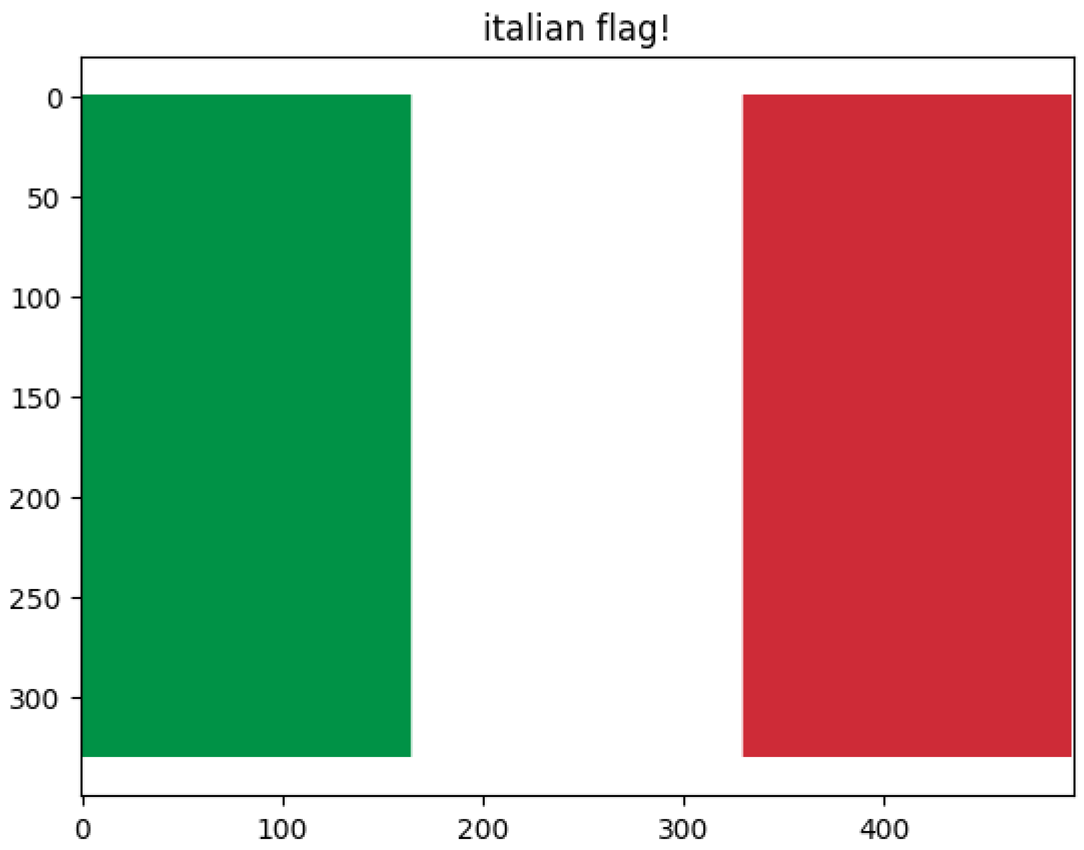
Color	Hex	RGB
	#f2f2f2	(242,242,242)
	#009246	(0,146,70)
	#ffffff	(255,255,255)
	#ce2b37	(206,43,55)
	#f2f2f2	(242,242,242)

Assemble 3D matrices that get displayed as RGB images

```
A = 255* np.ones([330, 495, 3]) # initialize white flag
A = A.astype("uint8") # convert to integer to ease the color

WIDTH = A.shape[1]//3 # integer division to get an index

# left vertical band (green)
A[:, 0 : WIDTH, 0] = 0 # R: there is no red in the green of the italian flag
A[:, 0 : WIDTH, 1] = 146 # G
A[:, 0 : WIDTH, 2] = 70 # B
# no need to modify the central band (white)
# right vertical band (red), it is possible to do a vector assignment
A[:, WIDTH : -1, :] = [206, 43, 55] # from end - WIDTH till end
```

Image “generation”

```
# visualize the matrix as an image
plt.imshow(A)
plt.axis("equal")
plt.title("italian flag!")
```



Try it yourself…

Here is some examples..

Everything done just by programming

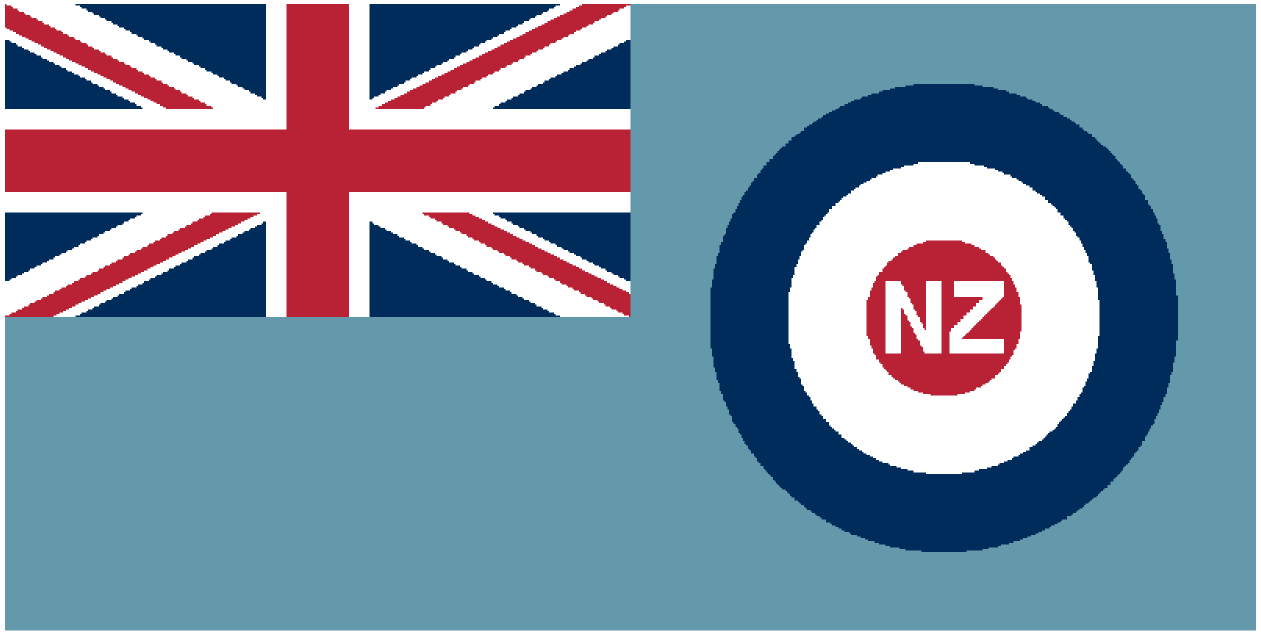


Image Superimposition

It is possible manipulate the content of an image as a matrix.

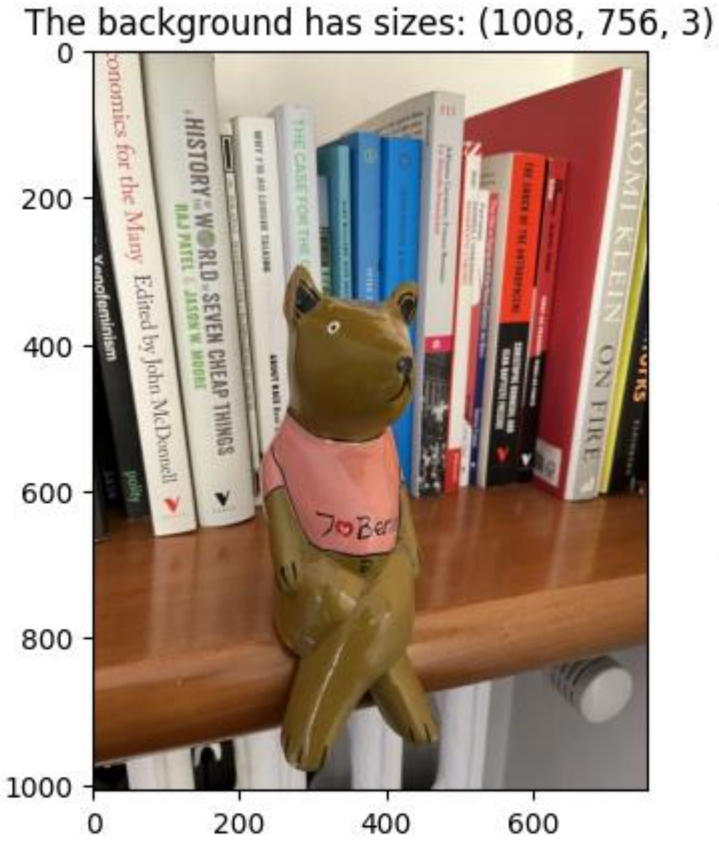
Load two images, background (B) and foreground (F) and superimpose F to B

Note that the two images

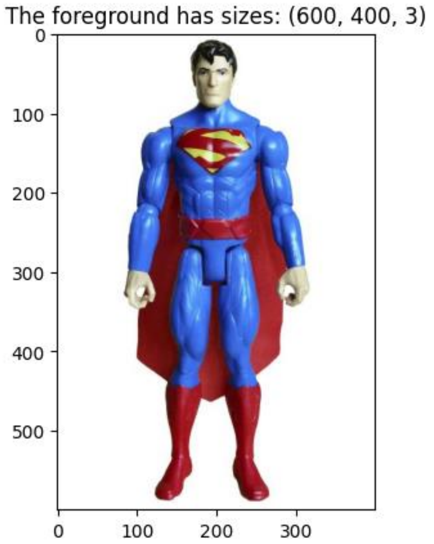
- have different sizes

- are (of course!) rectangular, but the superimposition implies that not all the pixels of F are copied in B, avoid the white ones!

B



F

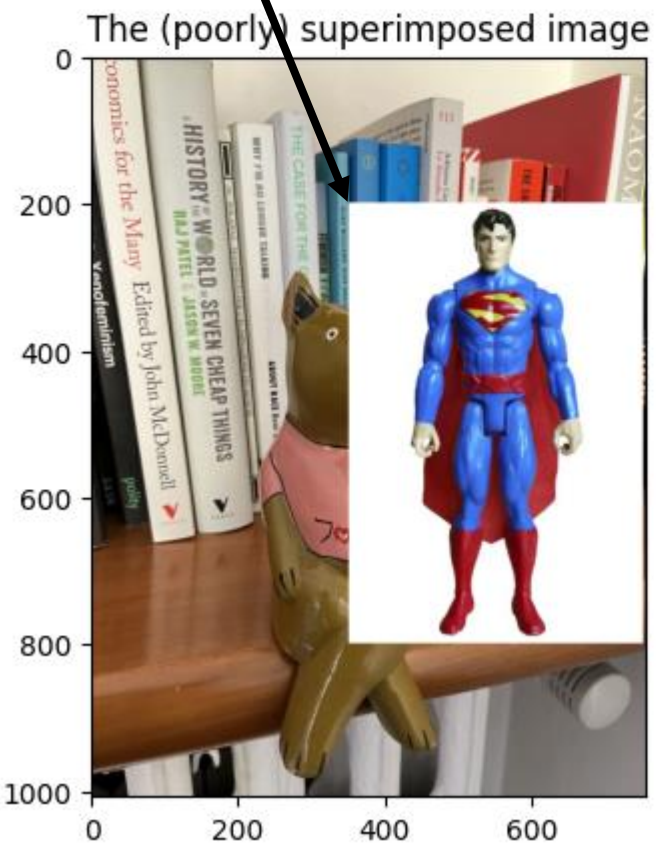


Trivial Superimposition

Replace all the pixels in a specific region of B by all the pixels in F

```
ul = [200, 350] # this is the location where
the upper left corner of F will be placed in B

F_size = F.shape

Res = B.copy() # otherise this is considered as
a reference and B will be modified

Res[ul[0] : ul[0] + F_size[0],
    ul[1] : ul[1] + F_size[1], : ] = F

plt.figure()

plt.imshow(Res)

plt.title("The (poorly) superimposed image")
```

```
ul = [200, 350]
```



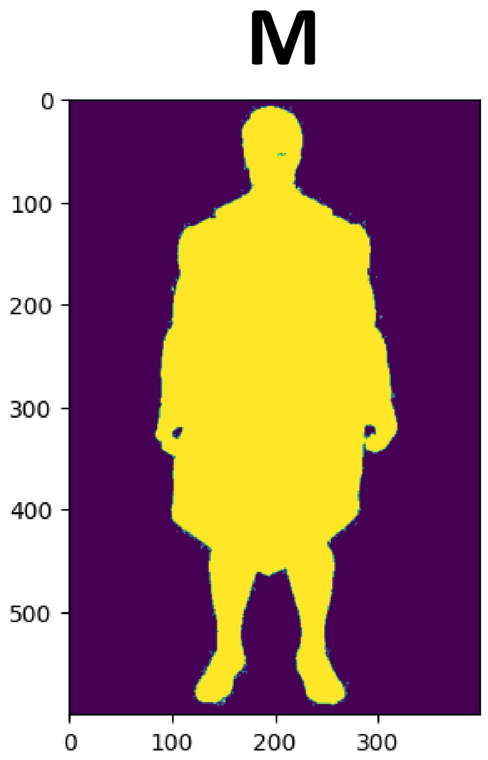
Masked Superimposition

Replace all the pixels in a specific region of B by pixels where F is not white (or reaches a high intensity)

We can identify pixels that are not white as pixels having at least one-color component lower than 240.

This can be done in two steps

- Define a 2D mask M of pixels of F to be replaced (M equals 1 where F is different from white)
- Replace within the selected image region the values of B with by $M * F + (1 - M) * B$

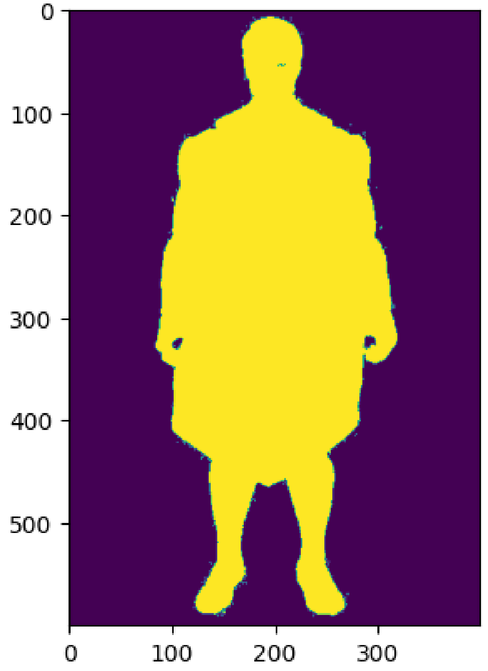


Masked Superimposition

Define a 2D mask M of pixels of F to be replaced (M equals 1 where F is different from white)

```
M3D = np.zeros([F_size[0], F_size[1]])
M3D = F.copy();
M3D[:,:,0]  = M3D[:,:,0]  > 240 # this are pixels that according to red, are background
M3D[:,:,1]  = M3D[:,:,1]  > 240 # this are pixels that according to green, are background
M3D[:,:,2]  = M3D[:,:,2]  > 240 # this are pixels that according to blue, are background

M = M3D[:,:,0] * M3D[:,:,1] * M3D[:,:,2] # these are pixels which are background for all the channels

M = 1 - M # the mask has to be the opposite, 1 where we need to keep F
```

M

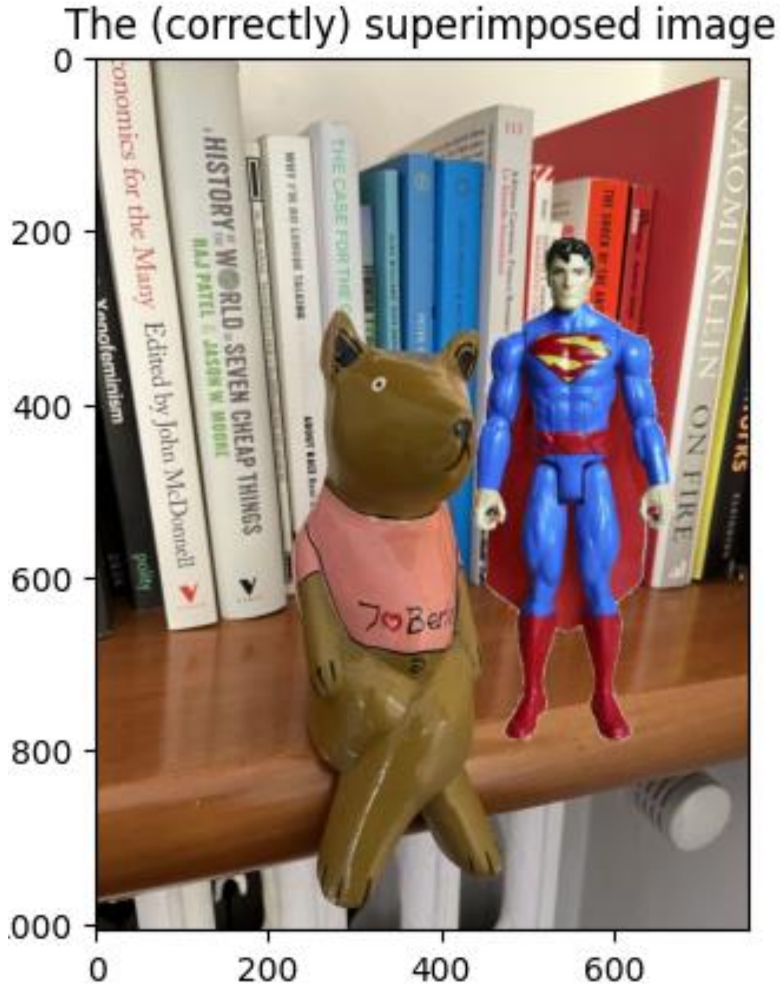


Masked Superimposition

Replace within the selected image region the values of B with by

$$M * F + (1 - M) * B$$

```
S = B.copy()
S[ul[0] : ul[0] + F_size[0], ul[1] : ul[1] + F_size[1], 0]
= M * F[:, :, 0] + (1 - M) * B[ul[0] : ul[0] + F_size[0],
ul[1] : ul[1] + F_size[1], 0]
S[ul[0] : ul[0] + F_size[0], ul[1] : ul[1] + F_size[1], 1]
= M * F[:, :, 1] + (1 - M) * B[ul[0] : ul[0] + F_size[0],
ul[1] : ul[1] + F_size[1], 1]
S[ul[0] : ul[0] + F_size[0], ul[1] : ul[1] + F_size[1], 2]
= M * F[:, :, 2] + (1 - M) * B[ul[0] : ul[0] + F_size[0],
ul[1] : ul[1] + F_size[1], 2]

plt.figure()plt.imshow(S)
plt.title("The (correctly) superimposed image")
```

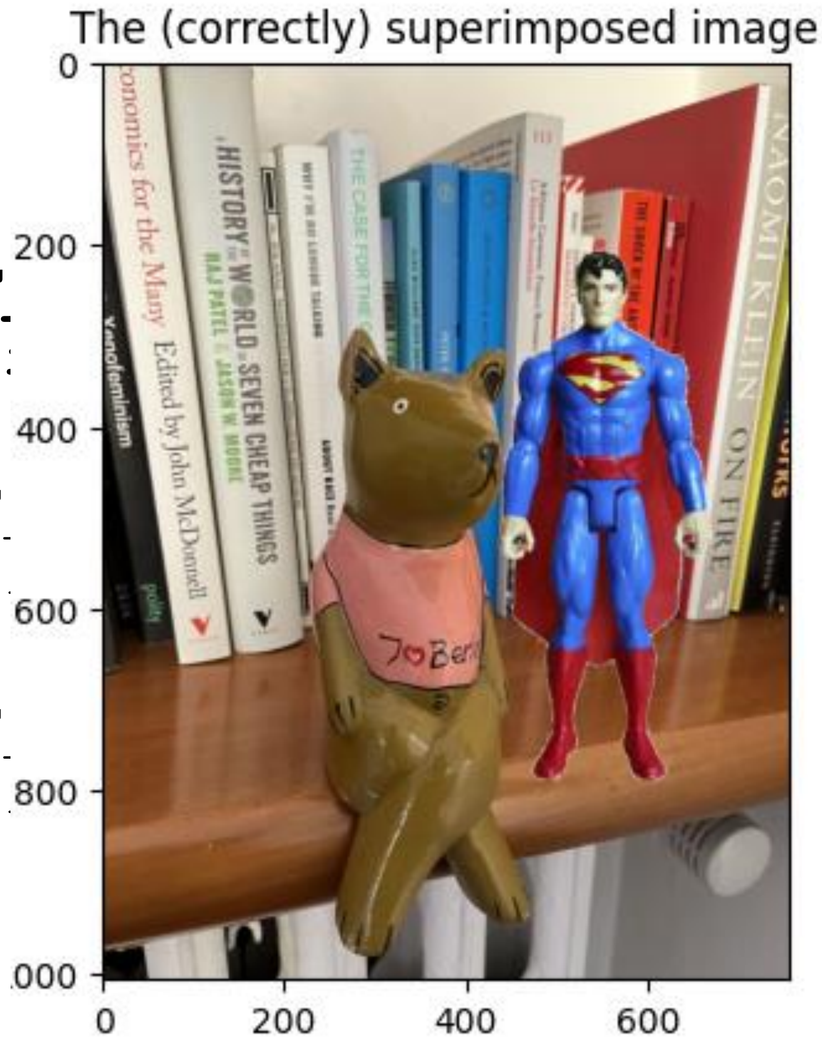


Alpha blending / Transparency

Replace within the selected image region the values of B with by

$$\alpha M * F + (1 - \alpha)(1 - M) * B$$

```
S = B.copy()
S[ul[0] : ul[0] + F_size[0], ul[1] : ul[1] + F_
* F[:, :, 0] + (1 - M) * B[ul[0] : ul[0] + F_s
ul[1] + F_size[1], 0]
S[ul[0] : ul[0] + F_size[0], ul[1] : ul[1] + F_
* F[:, :, 1] + (1 - M) * B[ul[0] : ul[0] + F_s
ul[1] + F_size[1], 1]
S[ul[0] : ul[0] + F_size[0], ul[1] : ul[1] + F_
* F[:, :, 2] + (1 - M) * B[ul[0] : ul[0] + F_s
ul[1] + F_size[1], 2]

plt.figure()plt.imshow(S)
plt.title("The (correctly) superimposed image")
```



TODO on Colab

Implement the green "screen trick"

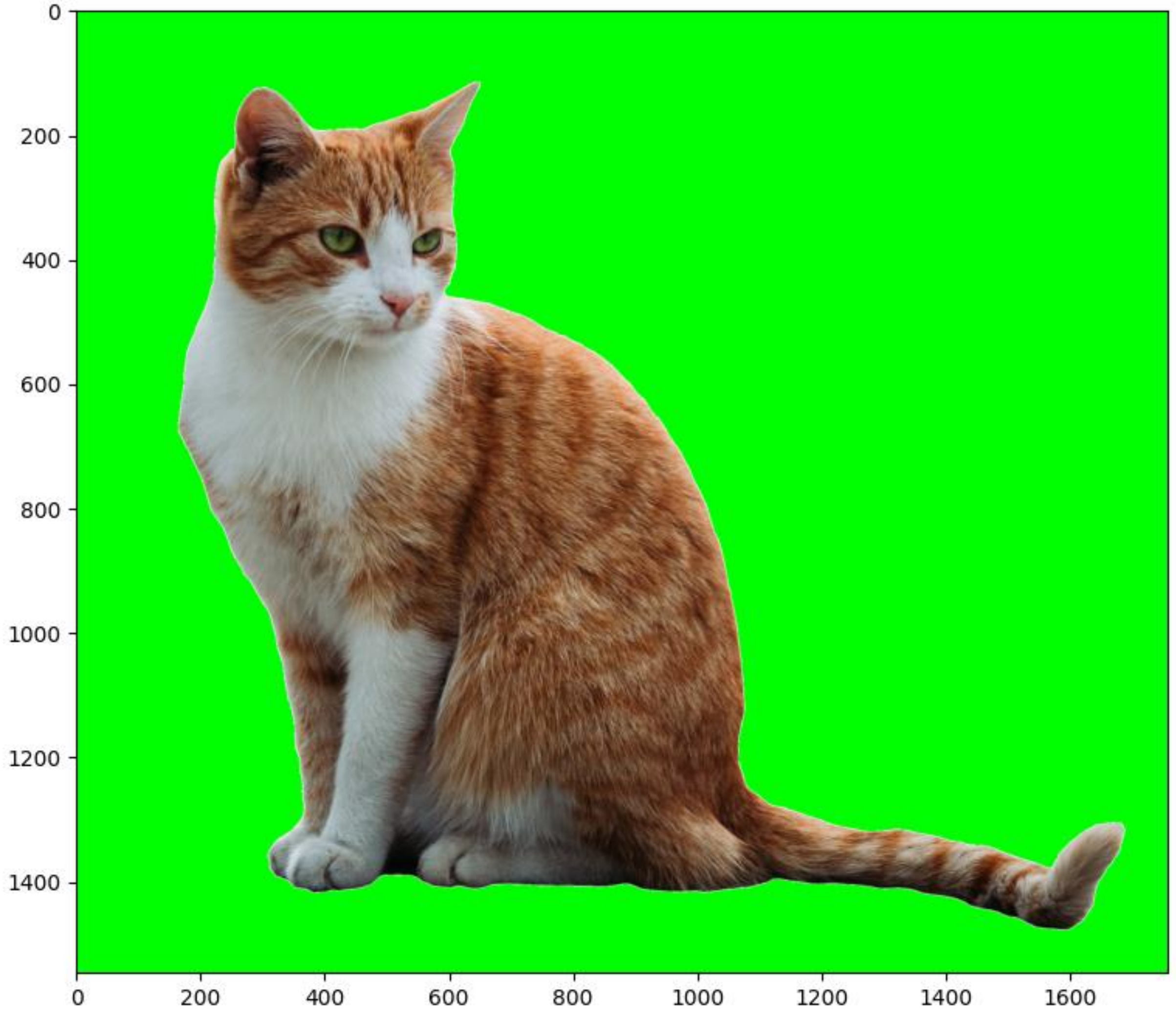


Image Transformations

Giacomo Boracchi

giacomo.boracchi@polimi.it

February 12, 2024

Agenda: Image transformation

- **Intensity transformations**
  - Gamma correction
  - Histogram equalization
  - Histogram matching
- **Local spatial transformations**
  - Correlation
  - Convolution

Intensity Transformations

Giacomo Boracchi

giacomo.boracchi@unibocconi.it

Image Analysis and Computer Vision

UEM, Maputo

https://boracchi.faculty.polimi.it

Intensity Transformations

In general, these can be written as

$$G(r, c) = T[I(r, c)]$$

Where

- I is the input image to be transformed
- G is the output
- T is a function, for instance
  - $T: \mathbb{R}^3 \rightarrow \mathbb{R}$ (e.g. colour to grayscale conversion)
  - $T: \mathbb{R}^3 \rightarrow \mathbb{R}^3$ (e.g. changing the colour encoding)
  - $T: \mathbb{R} \rightarrow \mathbb{R}$ (many channel-wise intensity transformation)

T operates independently on each single pixel.

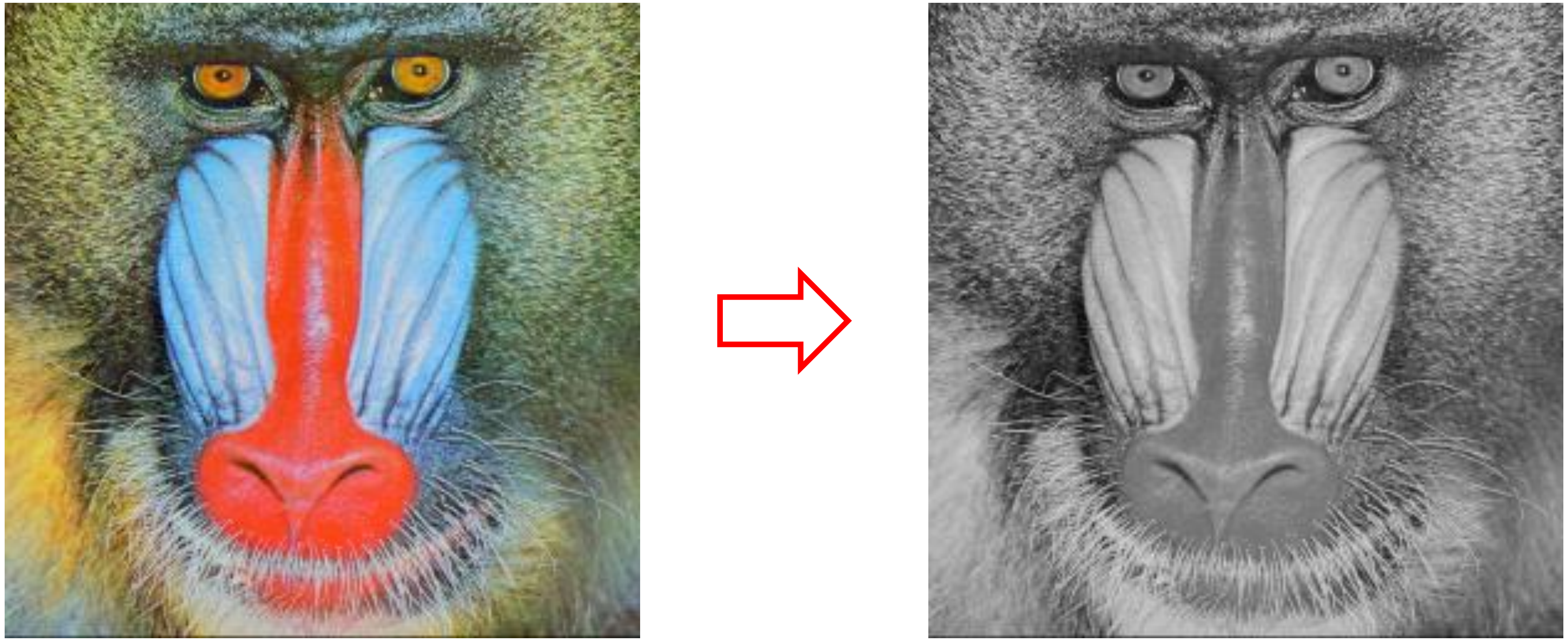
RGB → Grayscale Conversion

A linear transformation of pixel intensities $T\colon \mathbb{R}^3 \to \mathbb{R}$

$$Gray(r,c) = [0.299, 0.587, 0.114] * [R(r,c), G(r,c), B(r,c)]'$$

which corresponds to a linear combination of the 3 channels

$$Gray(r,c) = 0.299 * R(r,c) + 0.587 * G(r,c) + 0.114 * B(r,c)$$



YCbCr color space

Color space conversion $T: \mathbb{R}^3 \to \mathbb{R}^3$ to map RBG to YCbCr

- Y is the *luma* signal, similar to grayscale
- Cb and Cr are the *chroma* components

Human eye is less sensitive to color changes than luminance variations. Thus,

- Y can be stored / transmitted at high resolution
- *Cb* and *Cr* can be subsampled, compressed, or otherwise treated separately for improved system efficiency

(e.g., in JPEG compression the chromatic components are encoded at a coarser level than luminance)

RGB → YCbCr

There are many variants

$$\begin{bmatrix} Y' \\ P_B \\ P_R \end{bmatrix} = \begin{bmatrix} K_R & K_G & K_B \\ -\frac{1}{2}\cdot\frac{K_R}{1-K_B} & -\frac{1}{2}\cdot\frac{K_G}{1-K_B} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2}\cdot\frac{K_G}{1-K_R} & -\frac{1}{2}\cdot\frac{K_B}{1-K_R} \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

Where $'$ denotes the intensities are in the [0,1] range

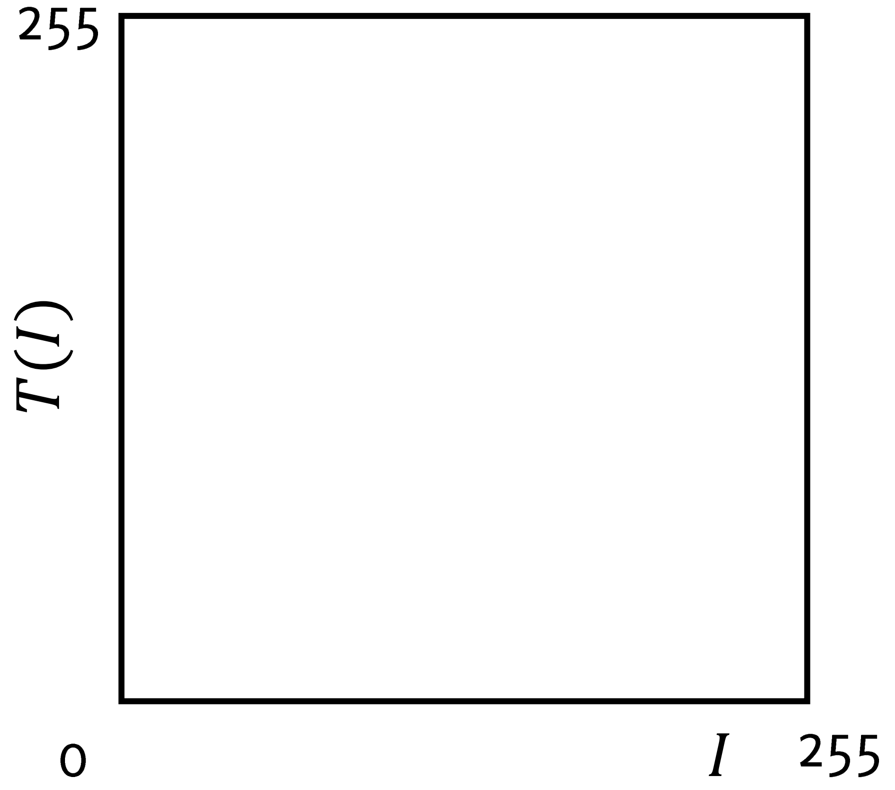
Here are color conversion in Python

```
img = cv2.cvtColor(src, bwsrc, cv::COLOR_RGB2GRAY)
img = cv2.cvtColor(frame, cv::COLOR_RGB2YCrCb)
img = cv2.cvtColor(frame, cv::COLOR_YCrCb2RGB)
```

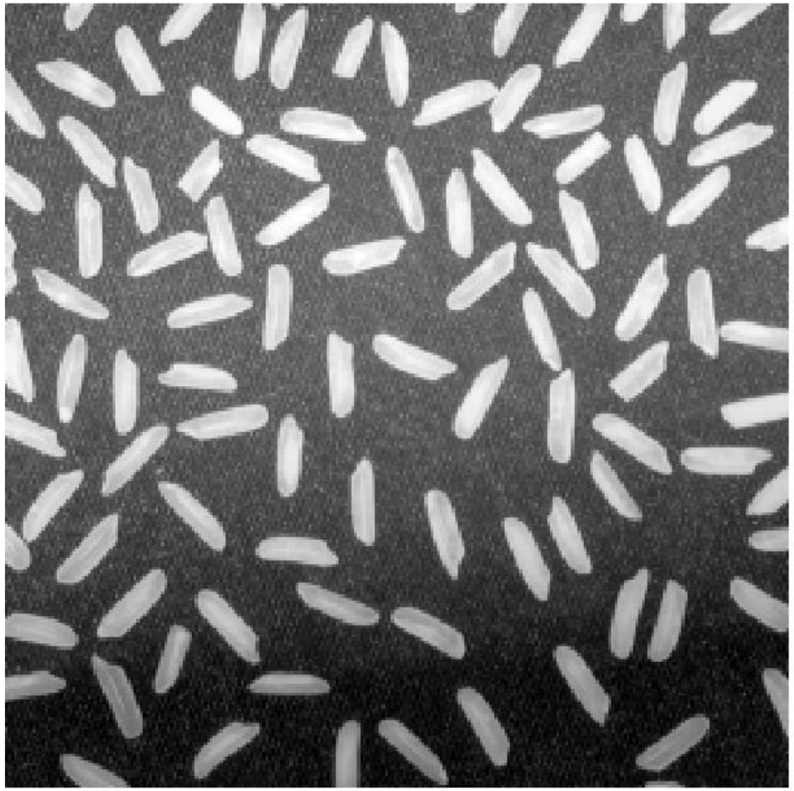
```
img = cv2.cvtColor(src, bwsrc, cv::COLOR_BGR2GRAY)
img = cv2.cvtColor(frame, cv::COLOR_BGR2YCrCb)
```

Negative Transformation

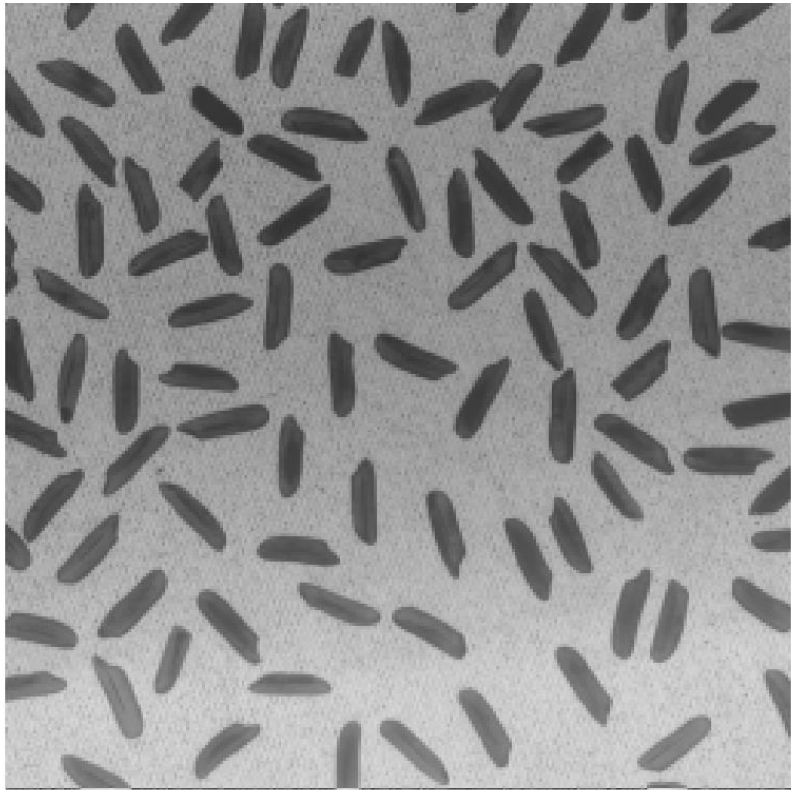
Simple transformation that maps black to white and white to black, and all the intensity levels in between as:



What Happened?



Input I



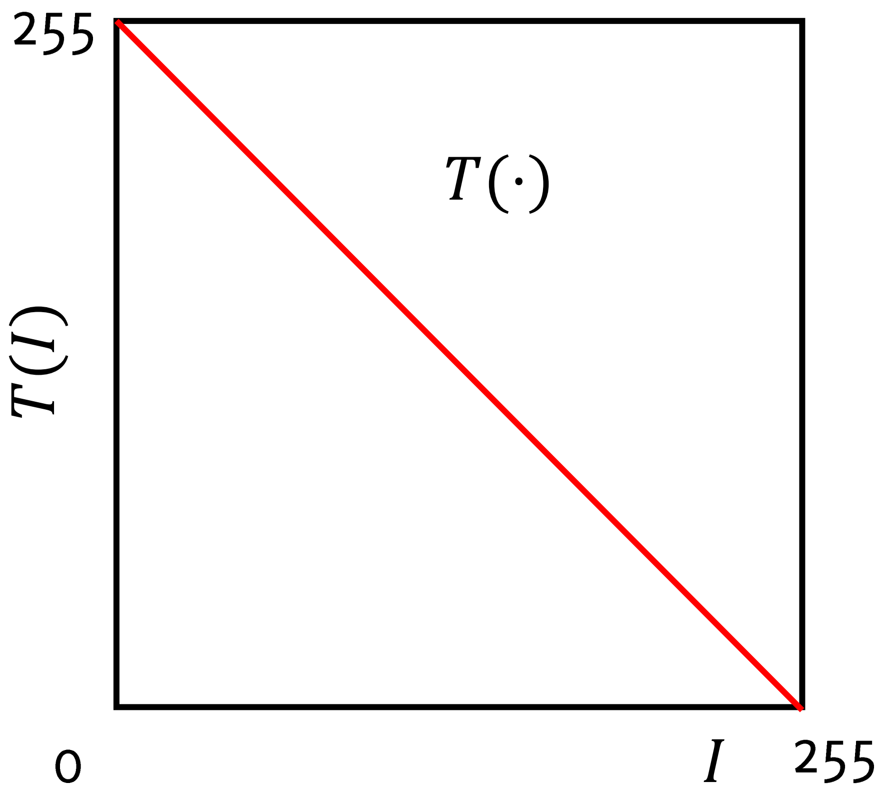
Output $G = T(I)$

Negative Transformation

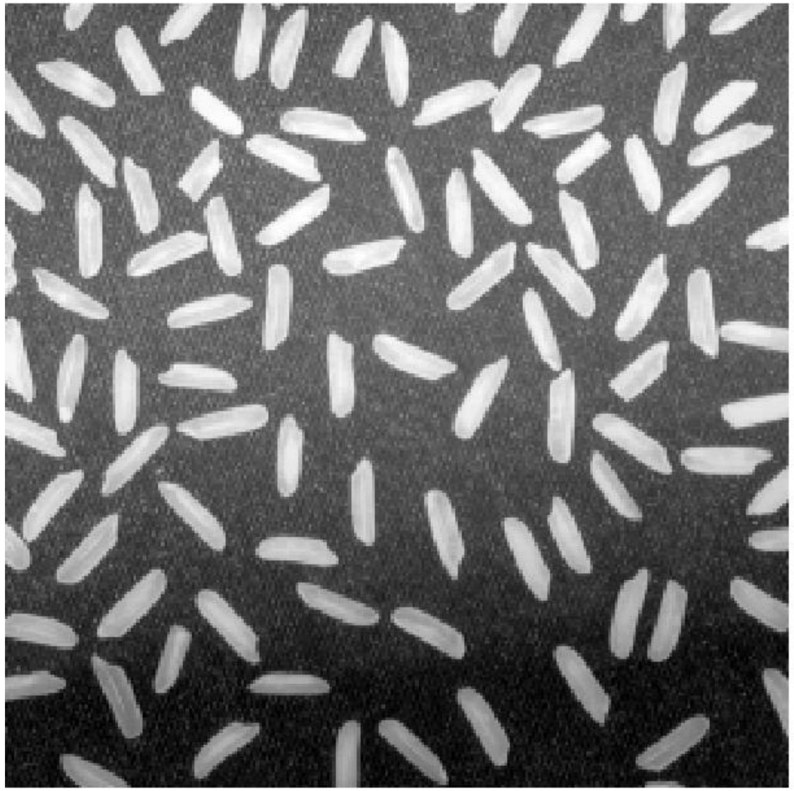
Simple transformation that maps black to white and white to black, and all the intensity levels in between as:

$$I(r,c) \rightarrow 255 - I(r,c)$$

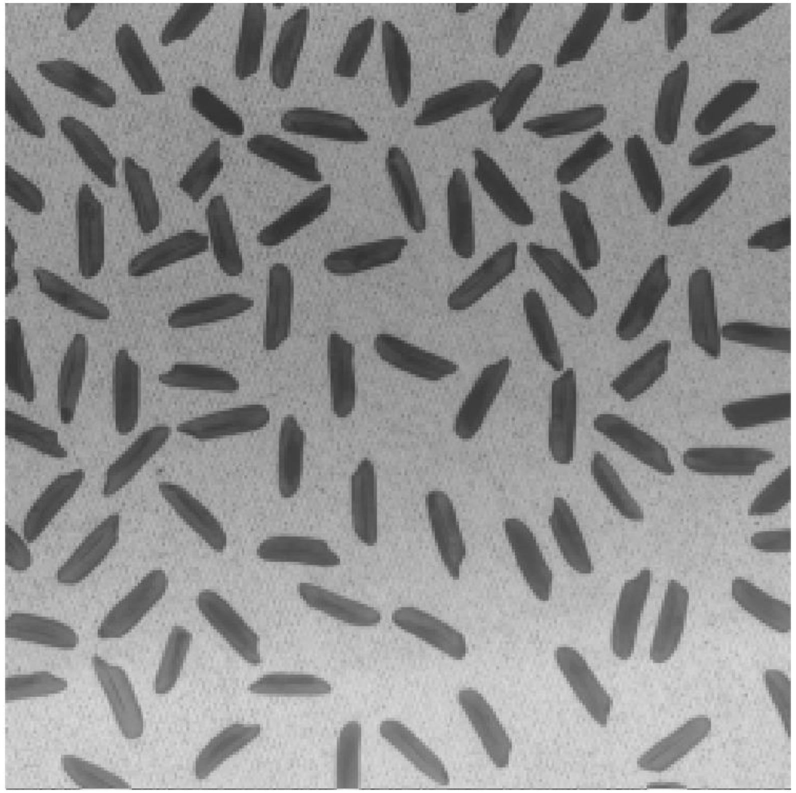
This is a linear transformation of intensities



What Happened?



Input I

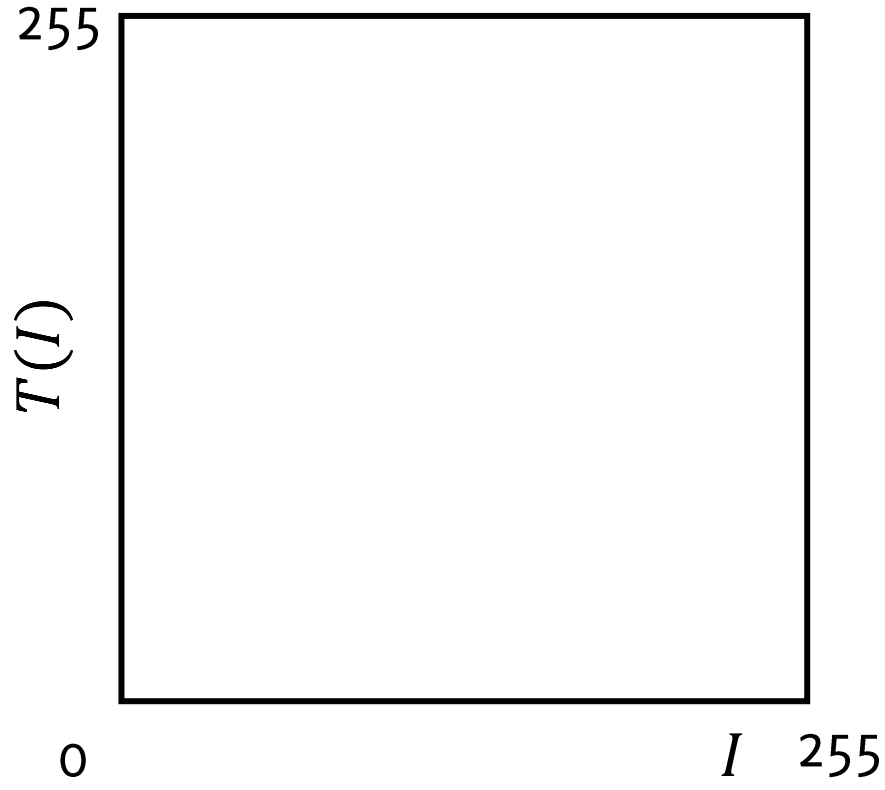


Output $G = T(I)$

Intensity Rescaling

In some cases images are conveniently mapped in the [0,255] range, covering such that

- $\min(T(I)) = 0$
- $\max(T(I)) = 255$



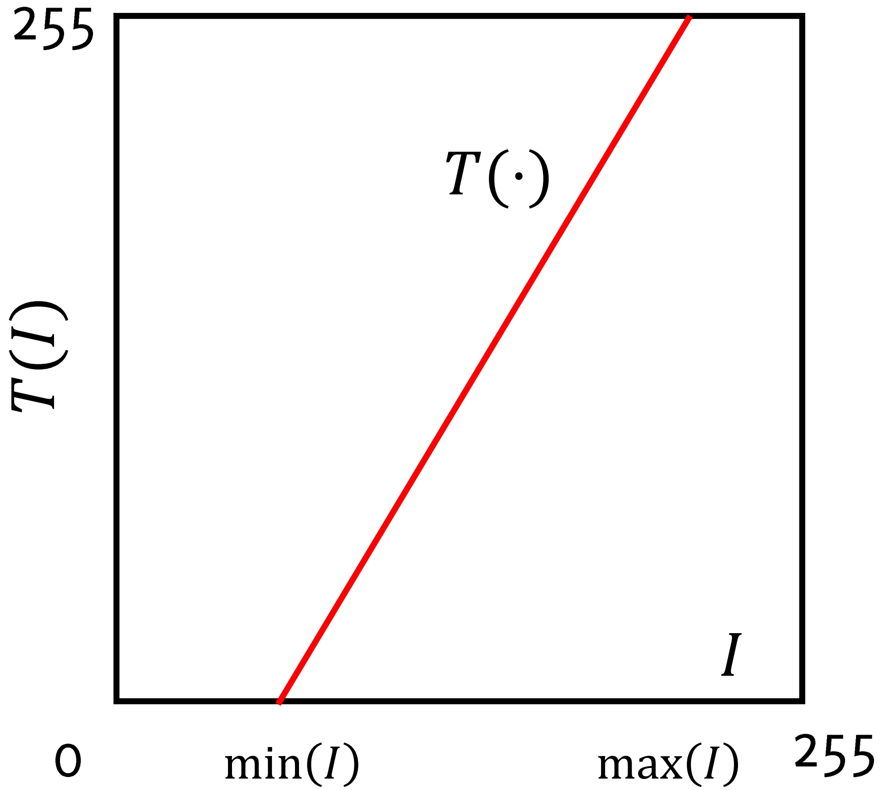
Intensity Rescaling

In some cases images are conveniently mapped in the [0,255] range, covering such that

- $\min\big(T(I)\big) = 0$
- $\max\big(T(I)\big) = 255$

$$I(r,c) \rightarrow 255 * \frac{I(r,c) - \min(I)}{\max(I) - \min(I)}$$

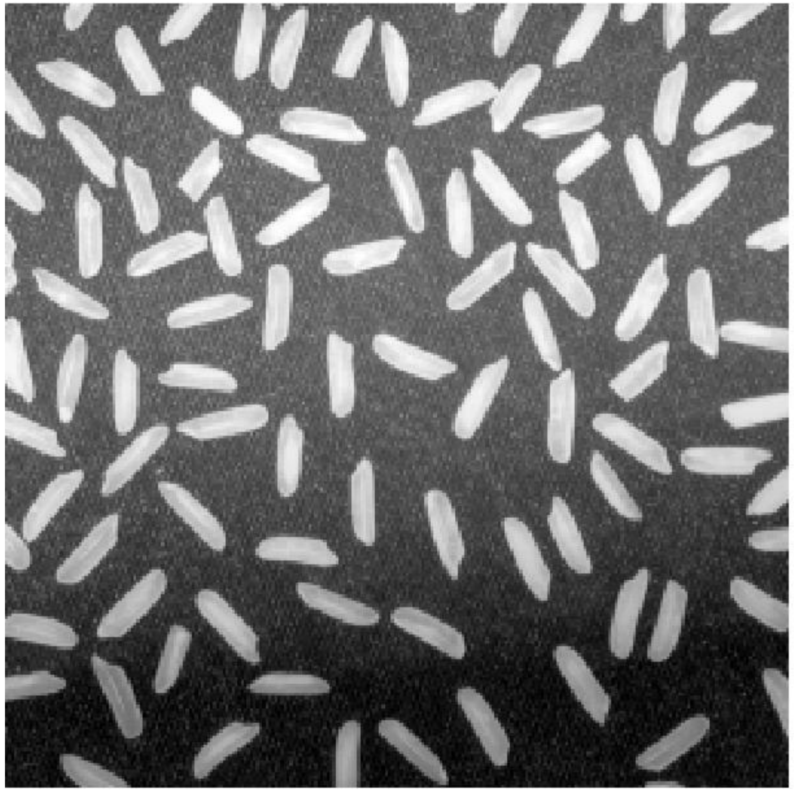
This is a linear transformation of intensities



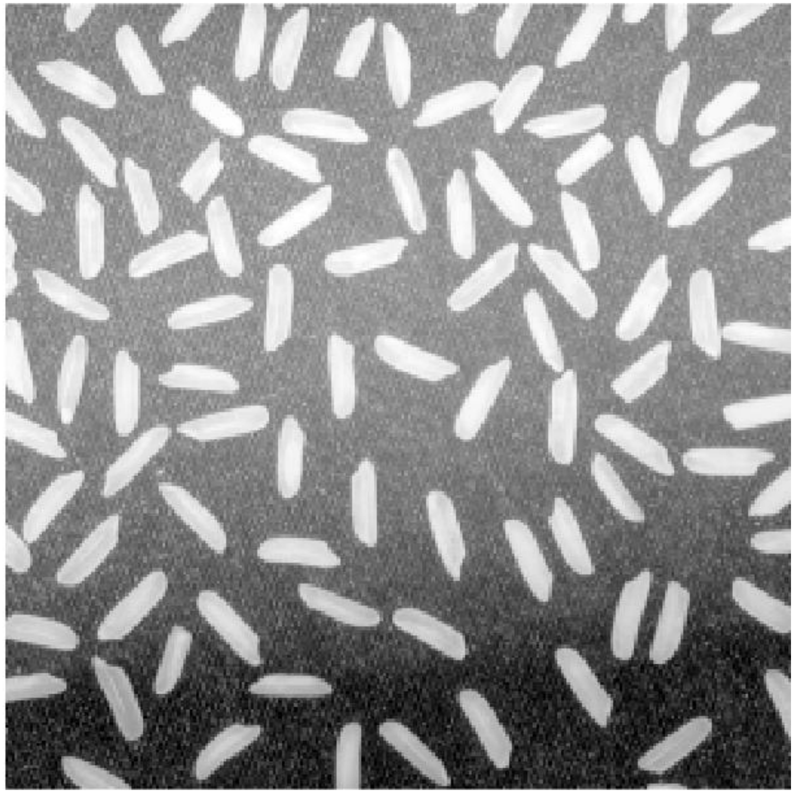
Intensity Rescaling

```
img_vec = img_underexposed.flatten(); # unroll the image in a vector
# get the min and the max
min_img = min(img_vec)
max_img = max(img_vec)
print('The range of the image is [', min_img,',',max_img,']')
# apply intensity rescaling
img_scaled = 255.0 * (img_underexposed-min_img)/(max_img-min_img)
img_scaled = img_scaled.astype('uint8')
```

What happened?

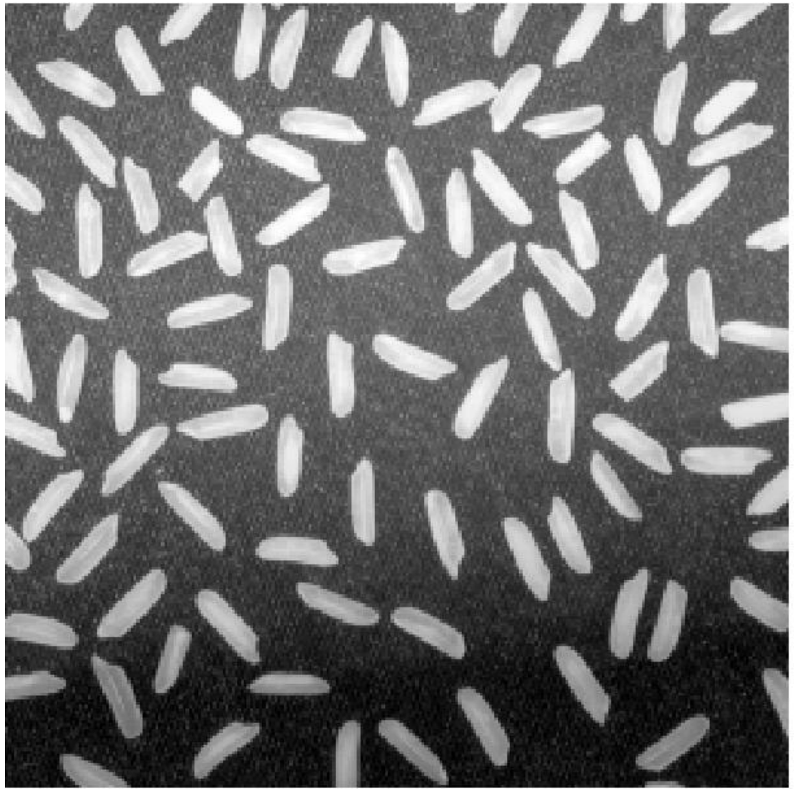


Input I

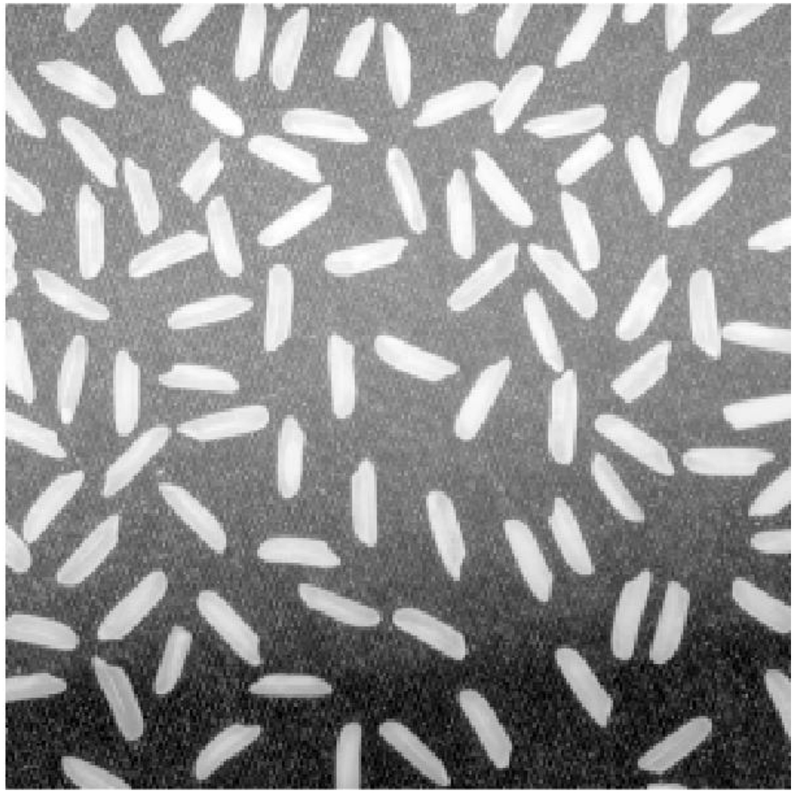


Output $G = T(I)$

Contrast increases in dark, decreases in bright



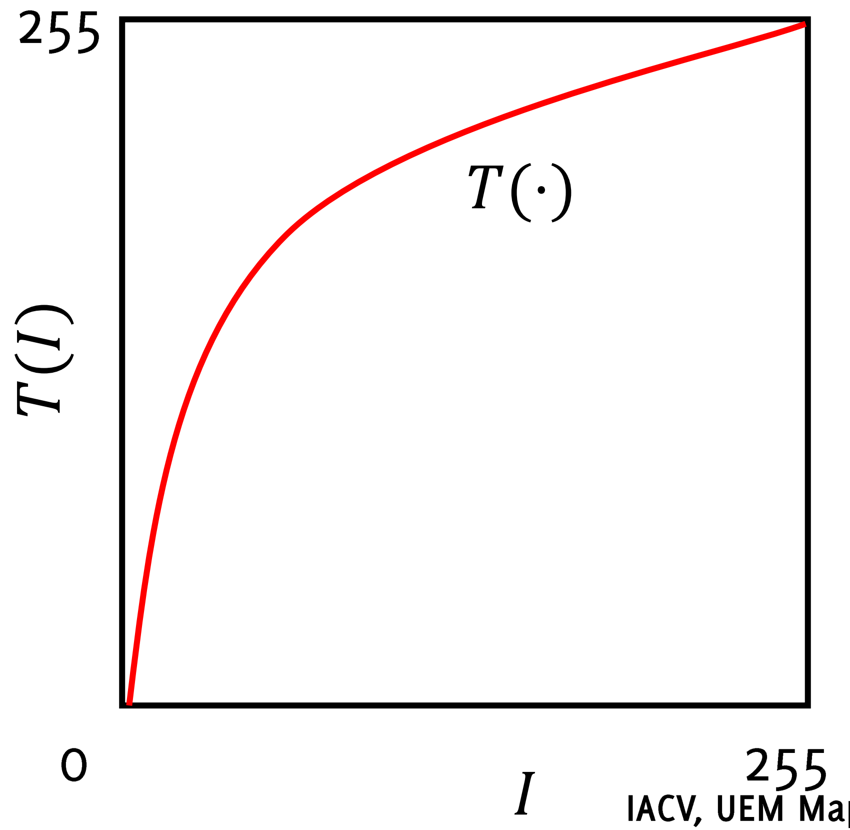
Input I



Output $G = T(I)$

Gray-level mapping

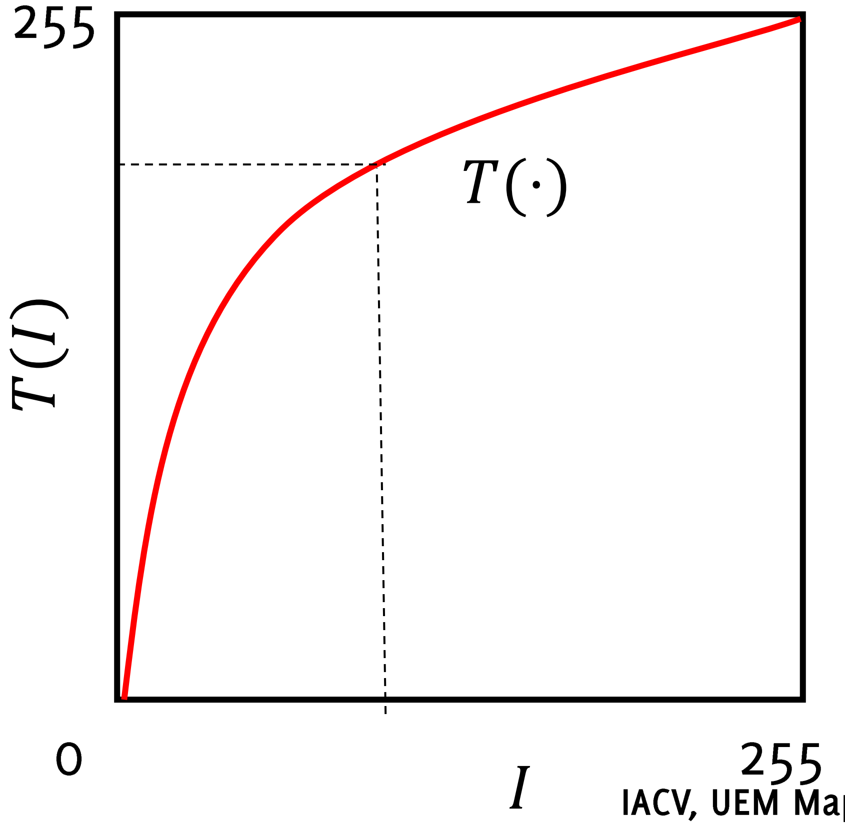
A transformation $T: \mathbb{R} \to \mathbb{R}$ that operates on gray-scale images or on each color-plane separately



Gray-level mapping

A transformation $T: \mathbb{R} \to \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

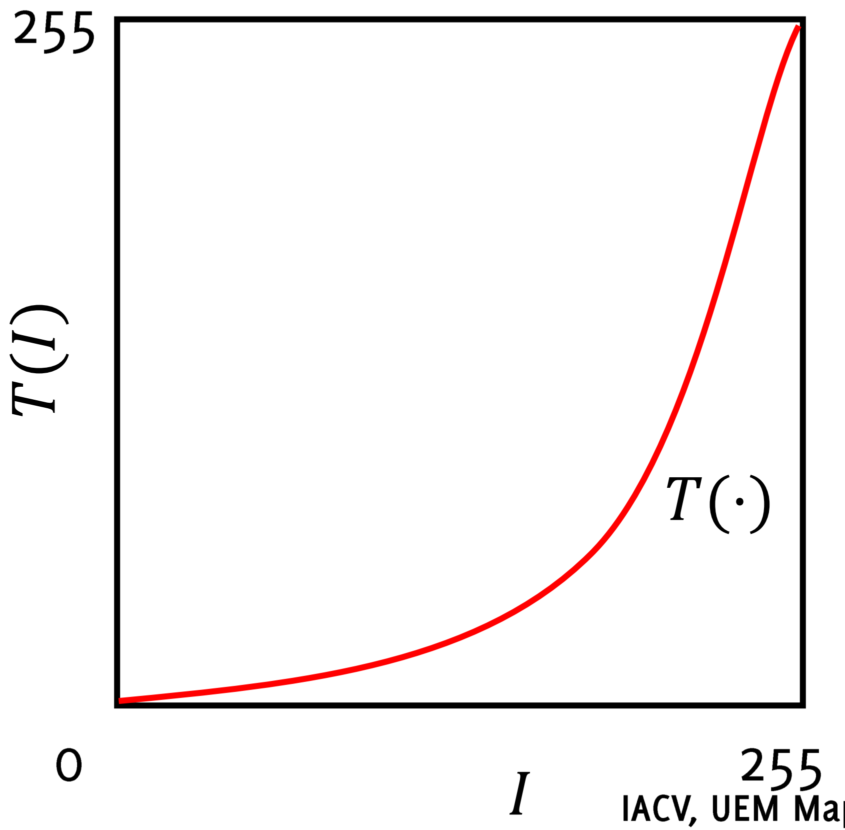
What does this T do?



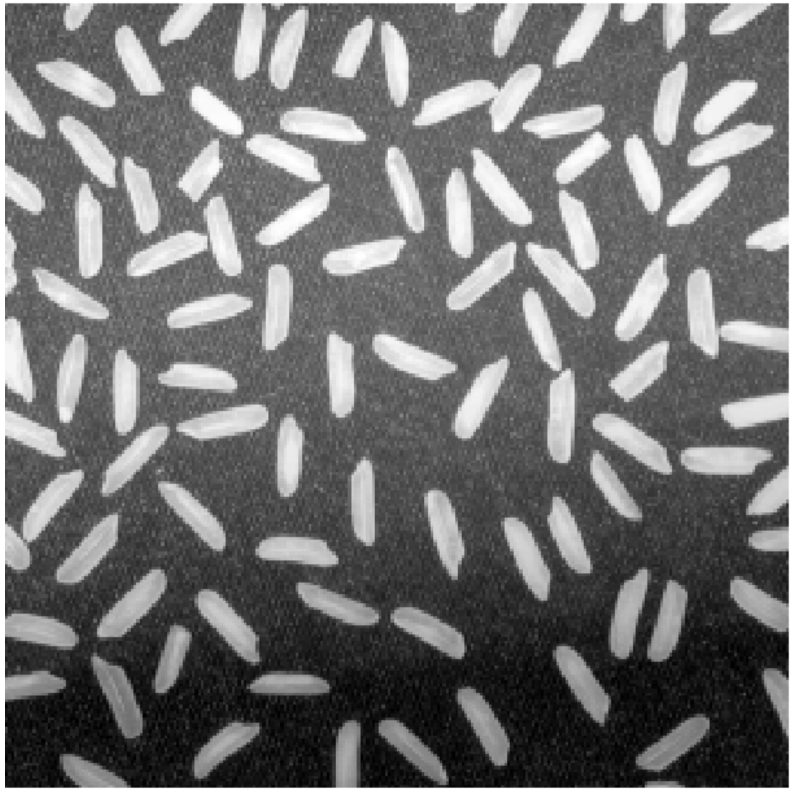
Gray-level mapping

A transformation $T: \mathbb{R} \to \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

What does this T do?



Contrast increases in bright, decreases in dark



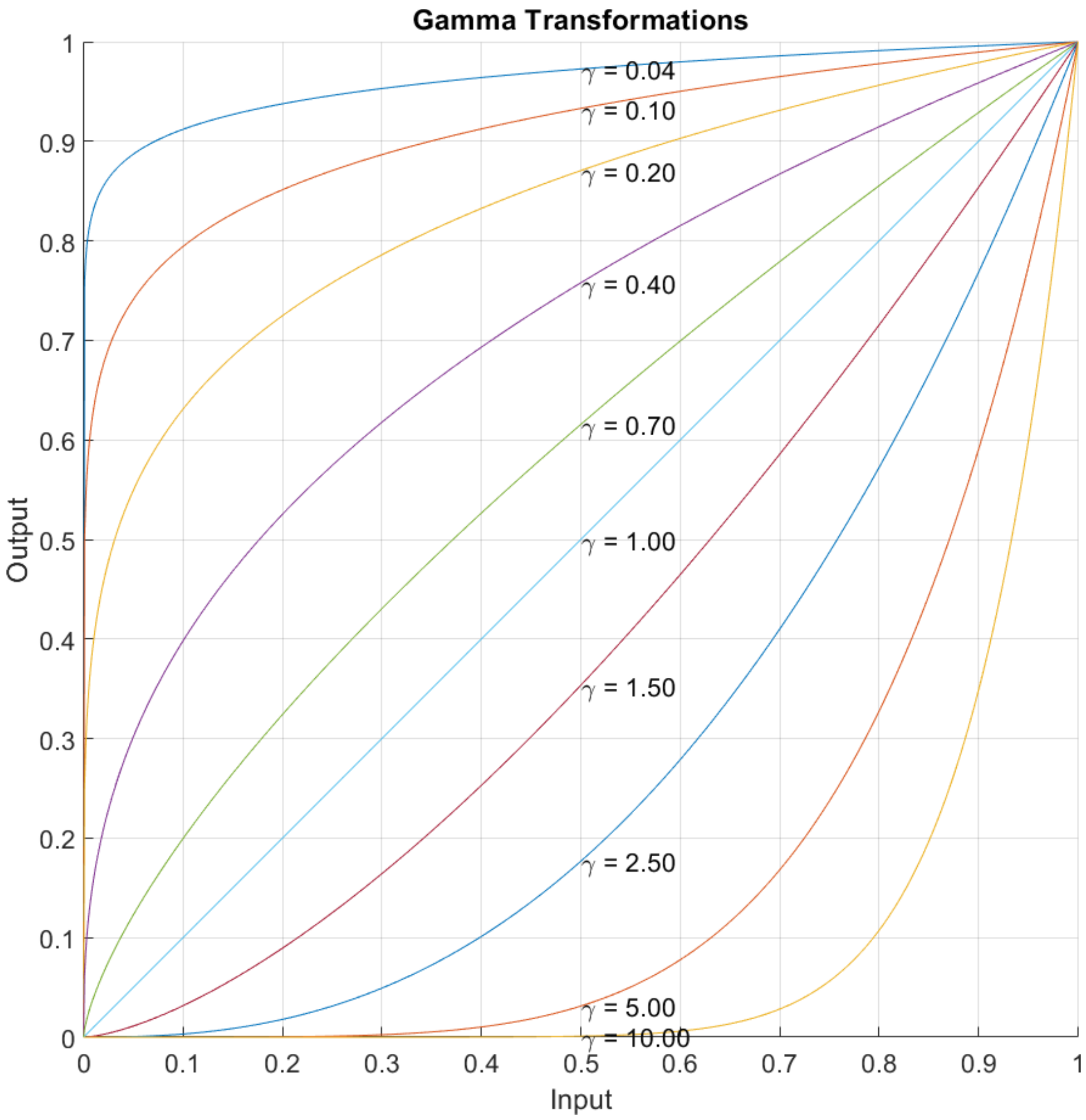
Input I

Output $G = T(I)$

Gamma Correction

Power-low transformation that can be written as

$$G(r, c) = I(r, c)^{\gamma}$$



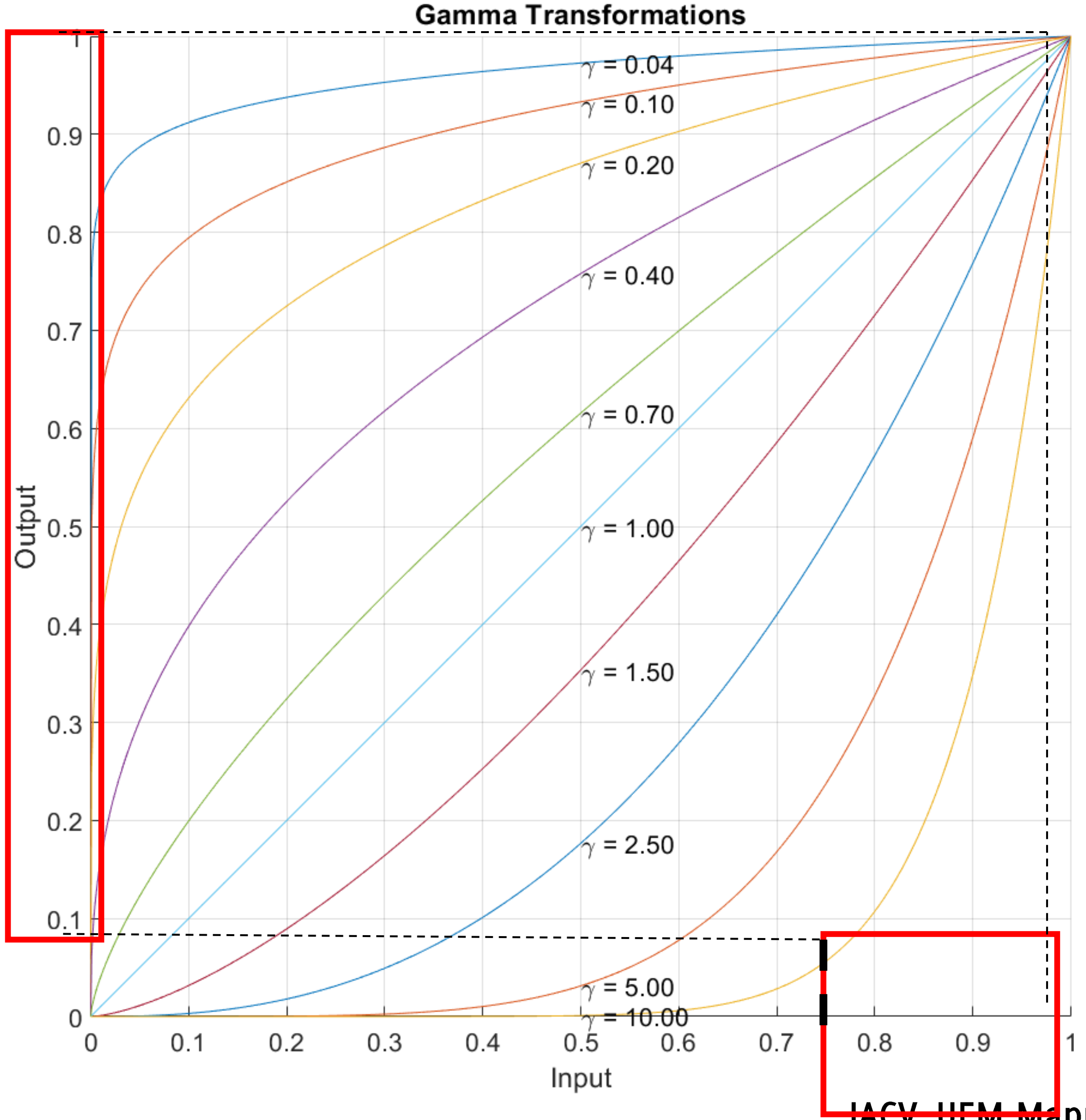
Gamma Correction

Power-low transformation that can be written as

$$G(r,c) = I(r,c)^{\gamma}$$

Contrast Enhancement:

- Low values of γ stretch the intensity range at high-values



Gamma Correction

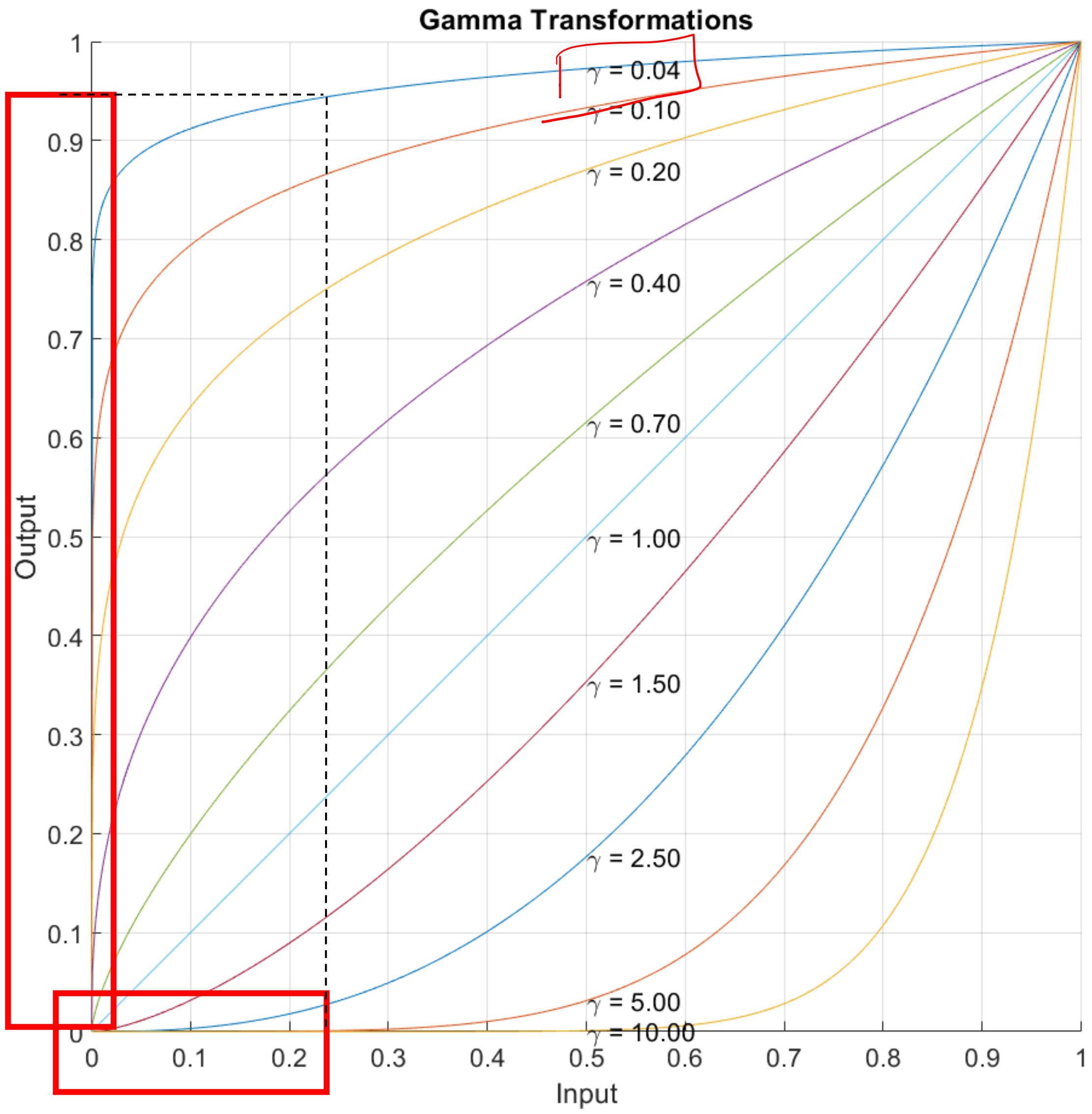
```
img_gamma = 255 * (img_gray/255) ** gamma
```

Power-low transformation that can be written as

$$G(r,c) = I(r,c)^{\gamma}$$

Contrast Enhancement:

- Low values of γ stretch the intensity range at high-values
- High values of γ stretch the intensity range at low values



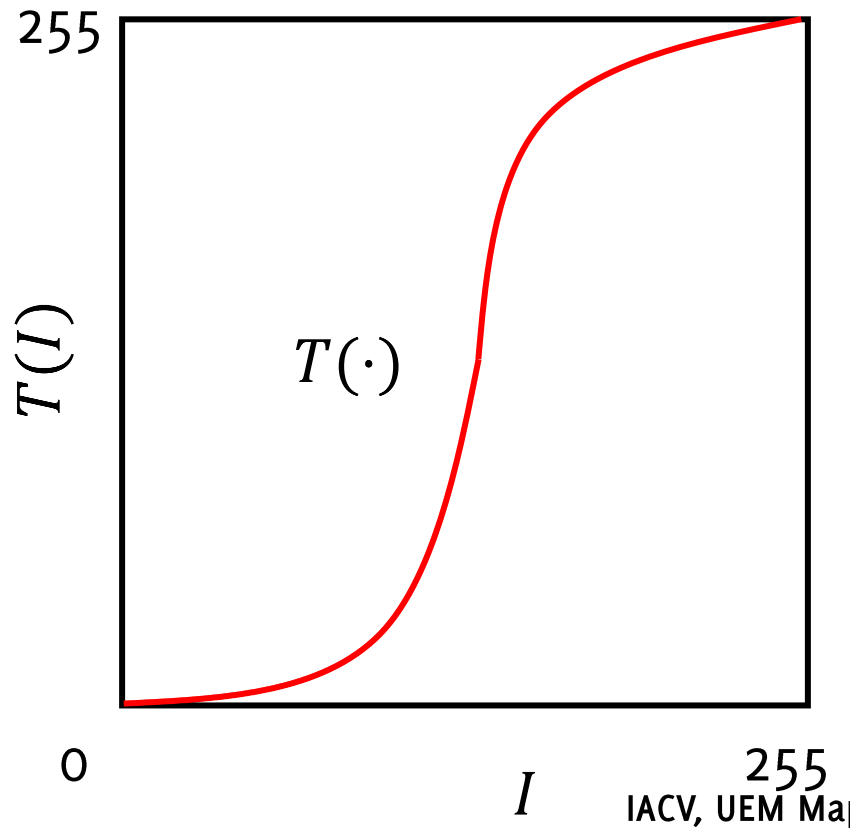
Gamma correction

```
gammas = [0.05, 0.1, 1, 2, 5]
num_gammas = len(gammas)

for gamma in gammas:
      img_gamma = 255 * (img_gray/255) ** gamma
      # display the result
      plt_idx = plt_idx + 1
      plt.subplot(1, num_gammas + 1, plt_idx)
      plt.imshow(img_gamma.astype('uint8'))
      plt.axis('off')
      plt.title('gamma %.2f' %gamma)
      plt.show()
```

Gray Level Mapping

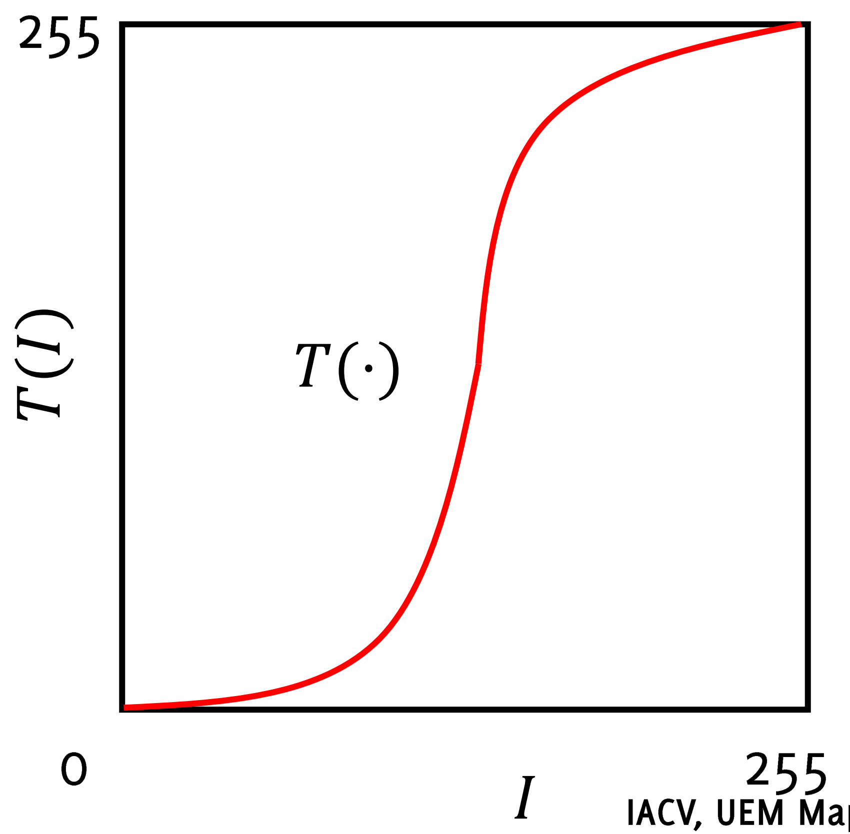
A transformation $T: \mathbb{R} \to \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

What does this T do?

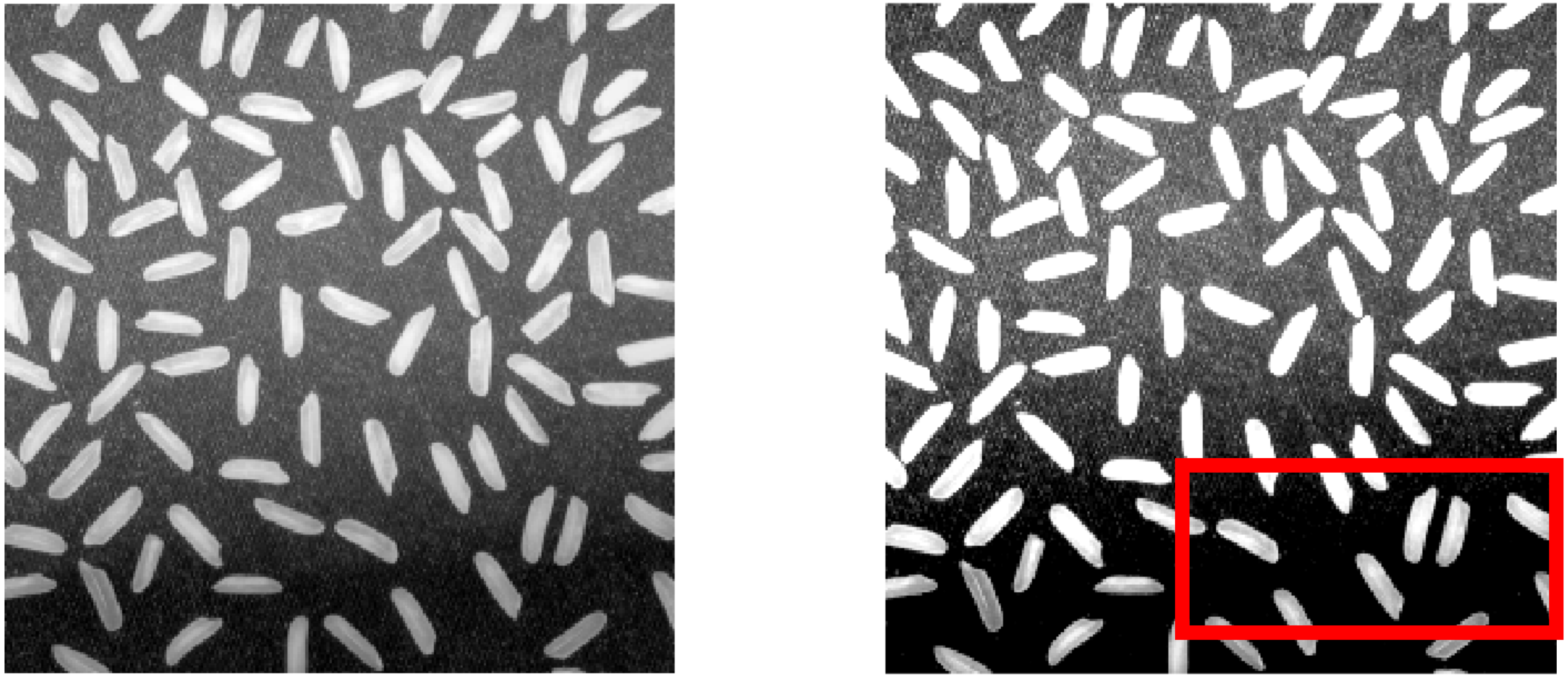


Contrast Stretching

A transformation $T: \mathbb{R} \to \mathbb{R}$ that operates on gray-scale images or on each color-plane separately



Contrast Stretching

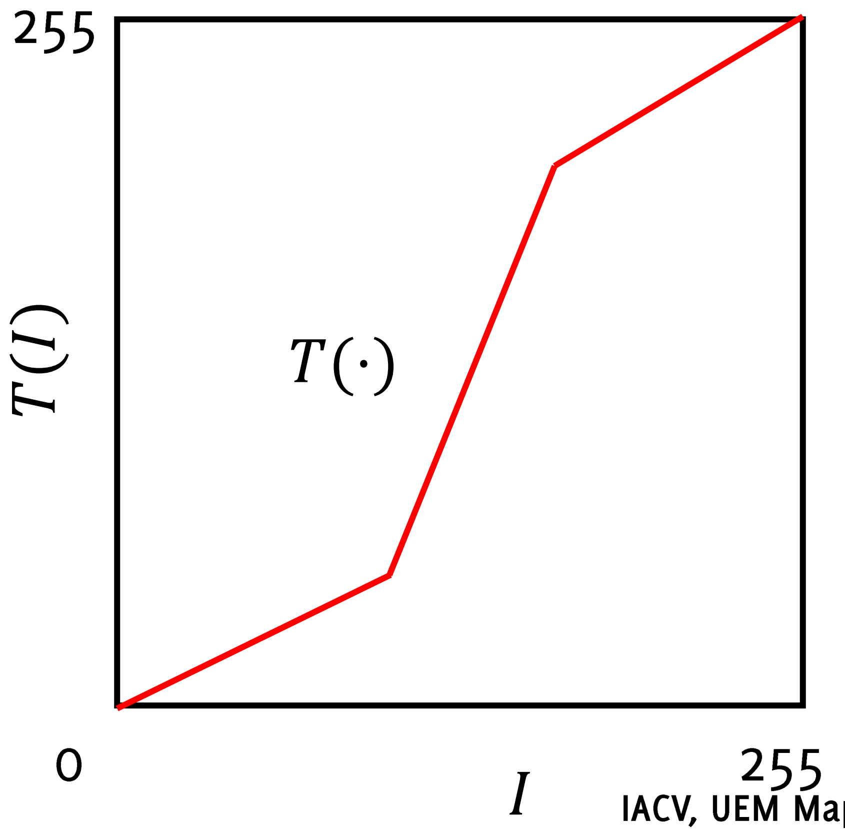


Contrast stretching: increases the constant at values in the middle of intensity range, decreases contrast at bright and dark regions.

It is implemented by piecewise or parametric transformations

Contrast Stretching

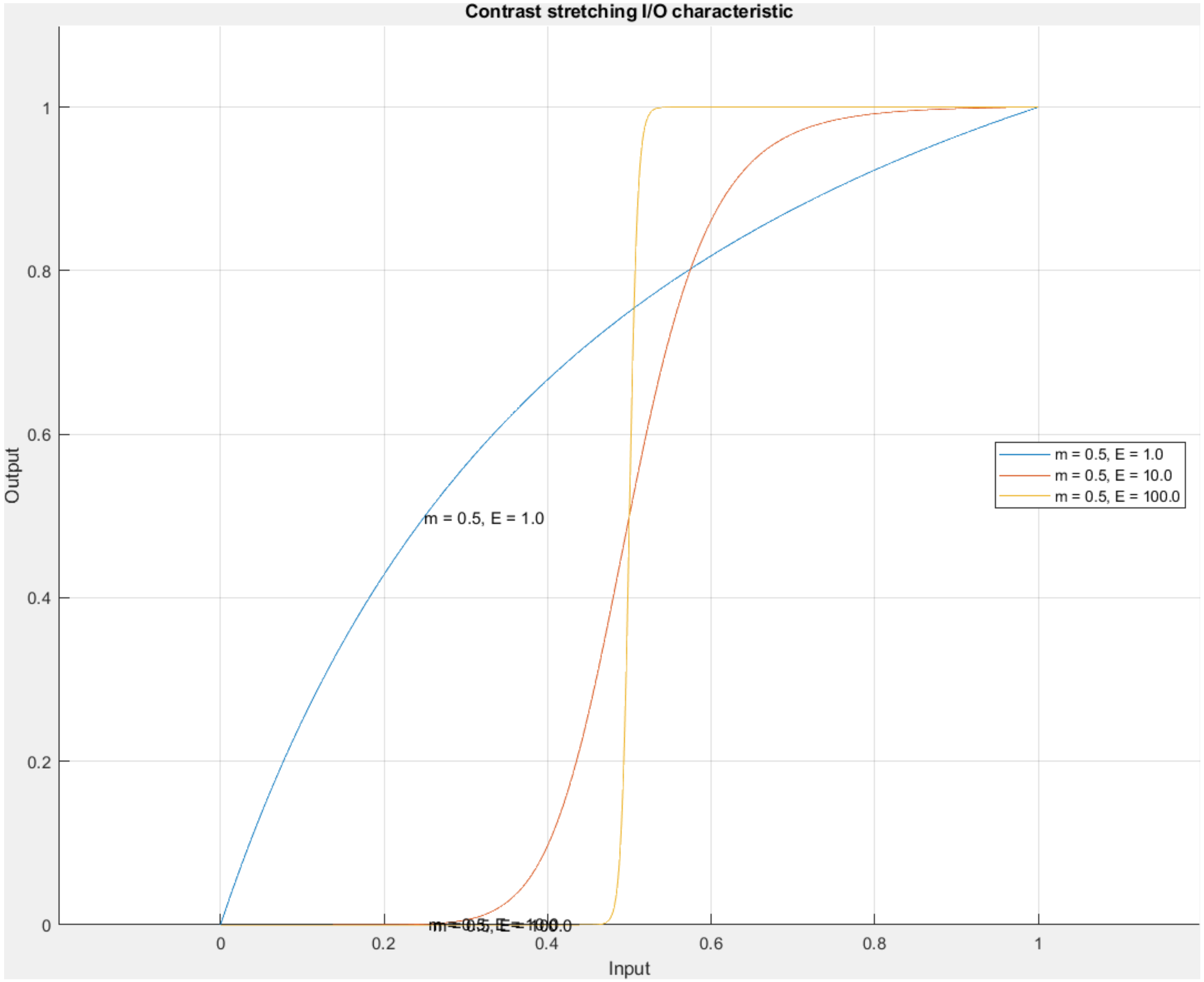
Can be defined by piecewise linear mapping...



Contrast Stretching

And there are also analytical expressions

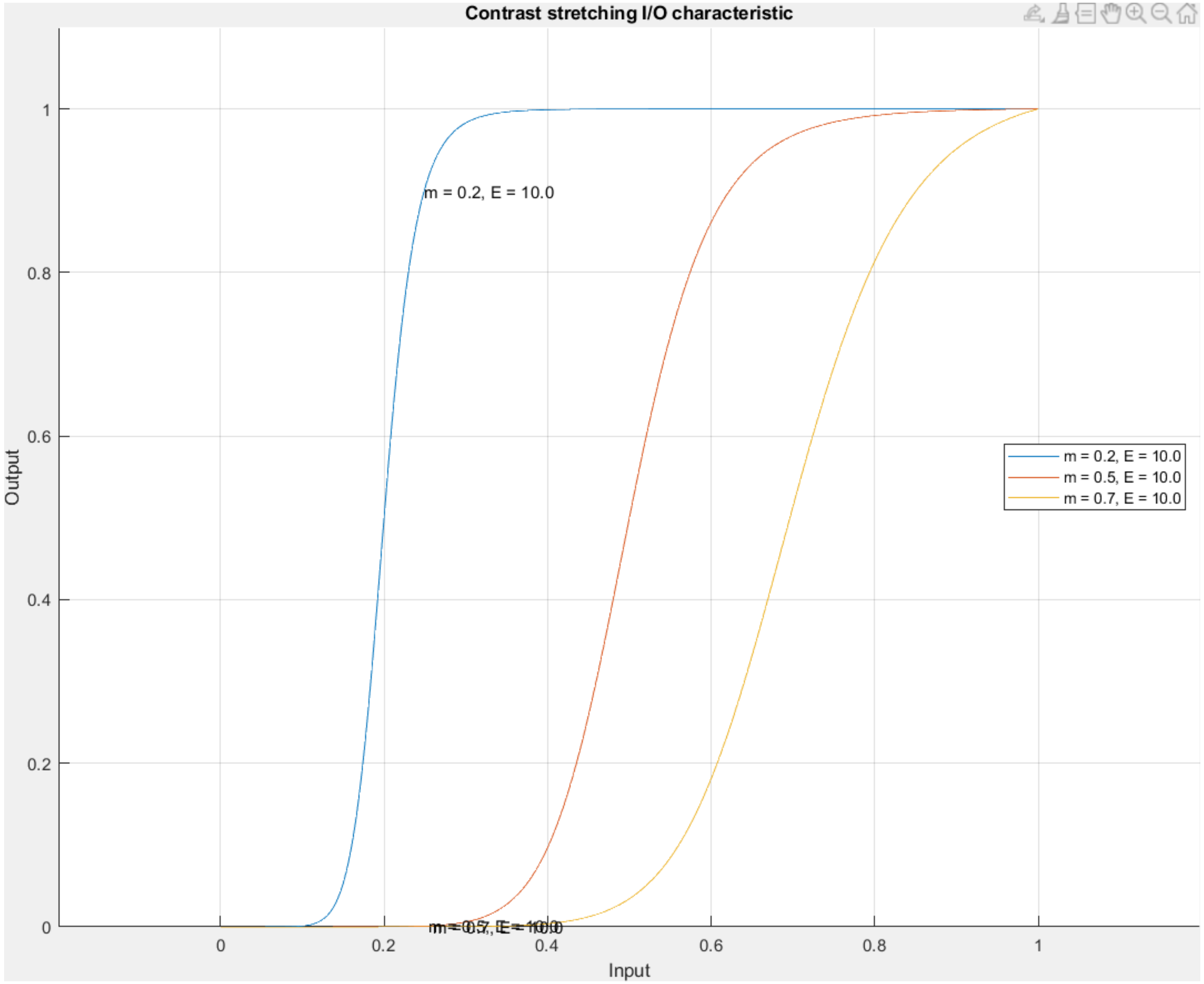
$$I(r,c) \rightarrow \frac{1+m^e}{\left(1+\left(\frac{m}{I(r,c)+\epsilon}\right)\right)^e}$$



Contrast Stretching

And there are also analytical expressions

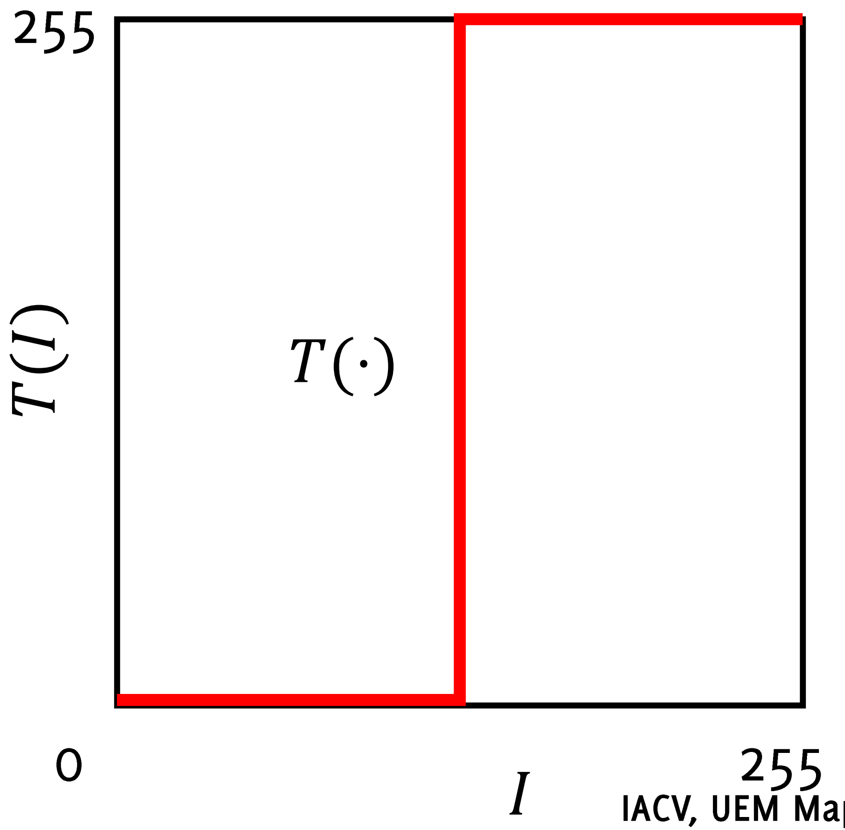
$$I(r,c) \rightarrow \frac{1+m^e}{\left(1+\left(\frac{m}{I(r,c)+\epsilon}\right)\right)^e}$$



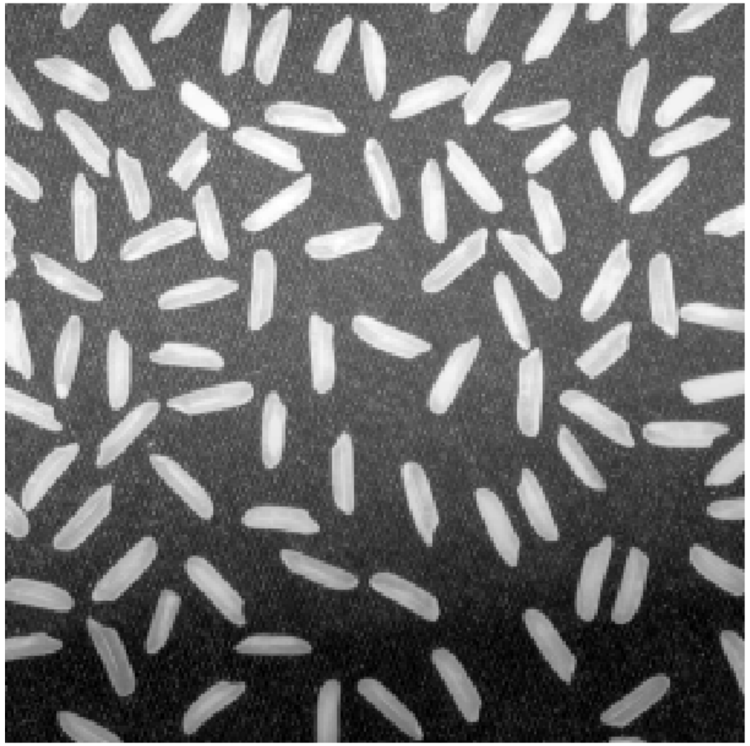
Gray-level mapping

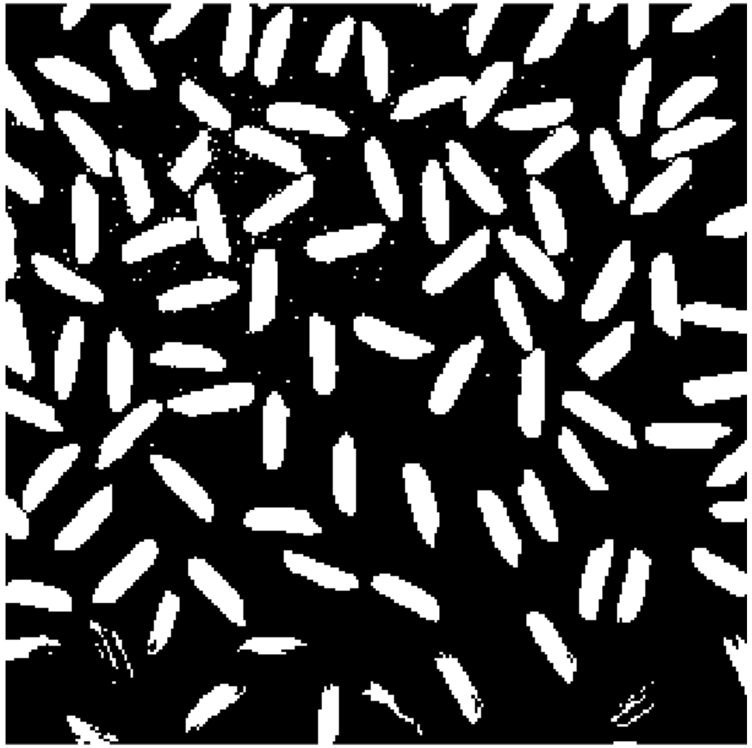
A transformation $T\colon \mathbb{R} \to \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

What does this T do?



Thresholding



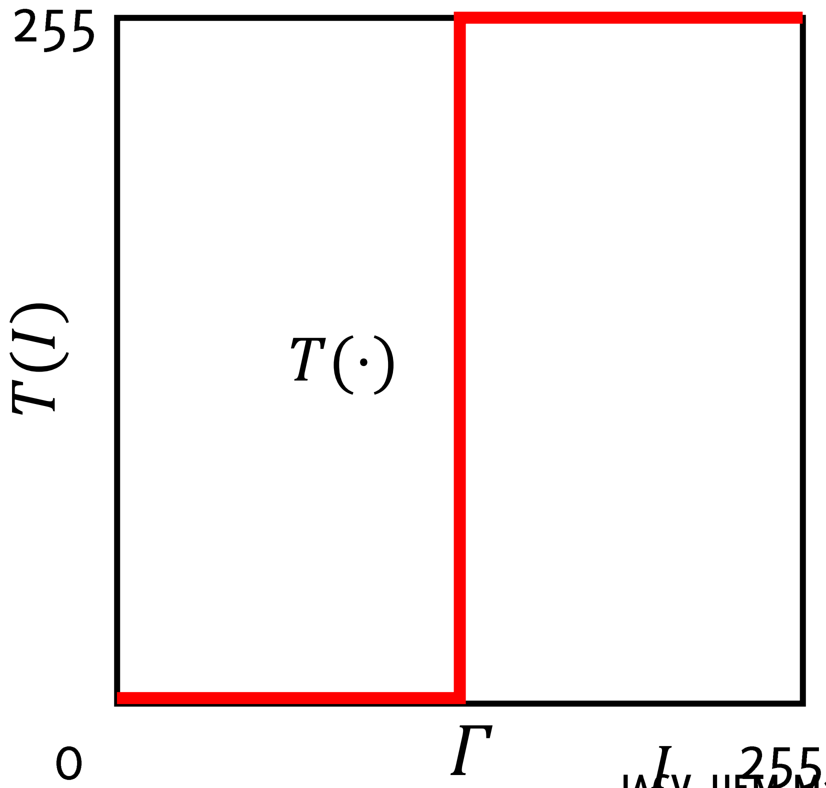


Thresholding binarizes images

Thresholding

A transformation $T: \mathbb{R} \rightarrow \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

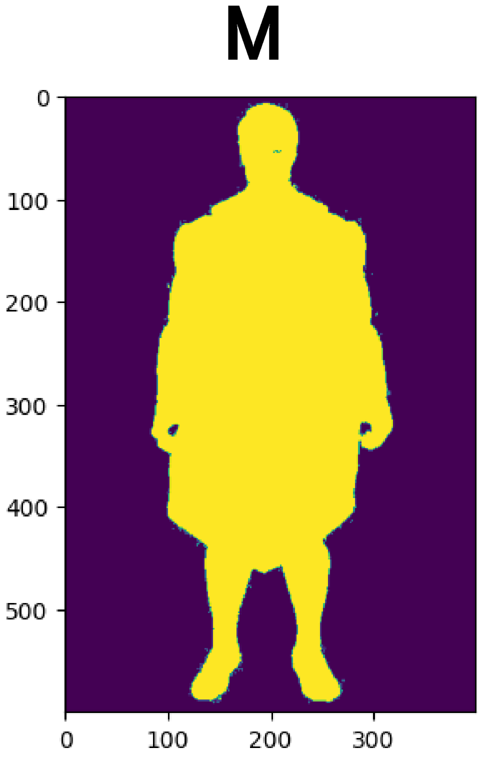
$$T\big(I(r,c)\big) = \begin{cases} 255, & \text{if } I(r,c) \geq \Gamma \\ 0, & \text{if } I(r,c) < \Gamma \end{cases}$$



This is exactly the same operation

But here, everything was done

- Within a single image region
- The operation was repeated along the three channels.

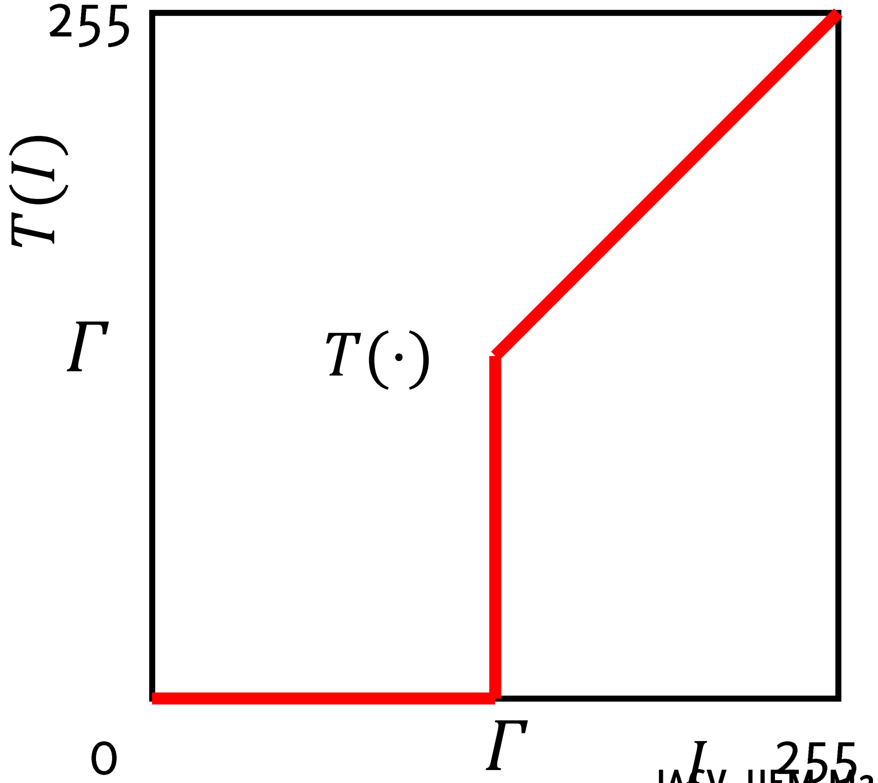



```
M3D = np.zeros([F_size[0], F_size[1]])
M3D = F.copy();
M3D[:,:,0]  = M3D[:,:,0]  > 240 # this are pixels that according to background
M3D[:,:,1]  = M3D[:,:,1]  > 240 # this are pixels that according to green, are background
M3D[:,:,2]  = M3D[:,:,2]  > 240 # this are pixels that according to blue, are background

M = M3D[:,:,0] * M3D[:,:,1] * M3D[:,:,2] # these are pixels which are background for all the channels
M = 1 - M # the mask has to be the opposite, 1 where we need to keep F
```

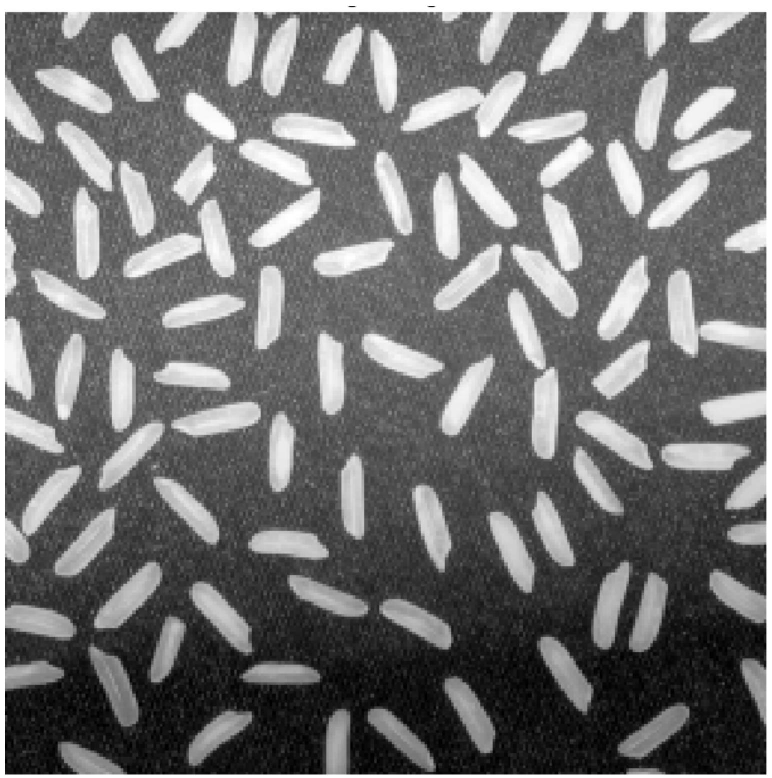
Thresholding

A transformation $T: \mathbb{R} \rightarrow \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

What does this T do?



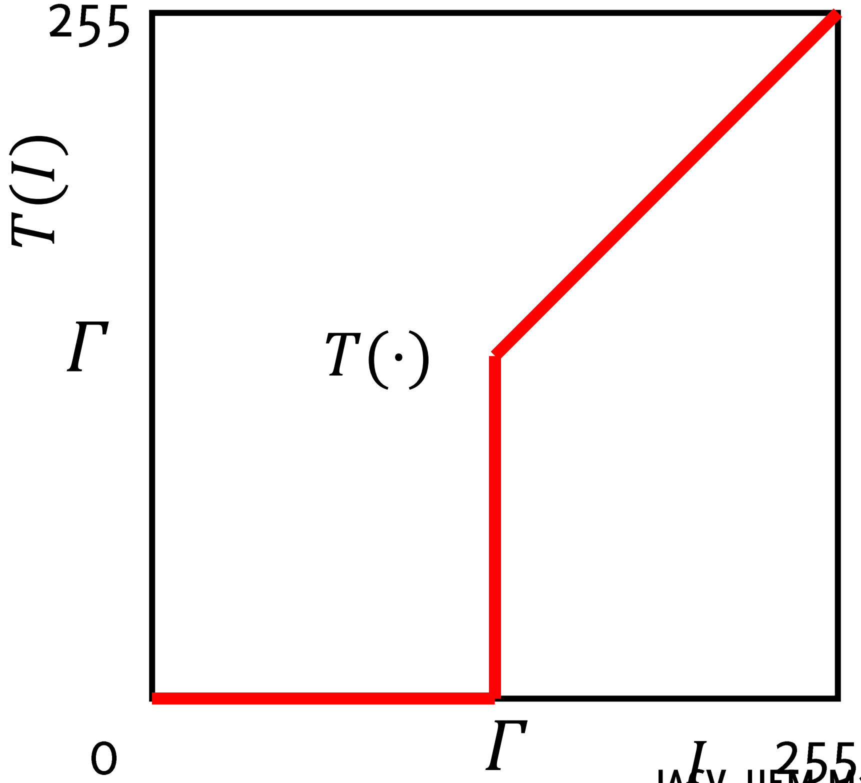
Thresholding



Thresholding

A transformation $T: \mathbb{R} \to \mathbb{R}$ that operates on gray-scale images or on each color-plane separately

$$T\big(I(r,c)\big) = \begin{cases} T\big(I(r,c)\big), & \text{if } I(r,c) \geq \Gamma \\ 0, & \text{if } I(r,c) < \Gamma \end{cases}$$



This simple operation is one of the most frequently used to add nonlinearities in CNN: the ReLU Layers

A Reference Book for Image Processing

"Digital Image Processing", 4th Edition Rafael C. Gonzalez, Richard E. Woods, Pearson 2017

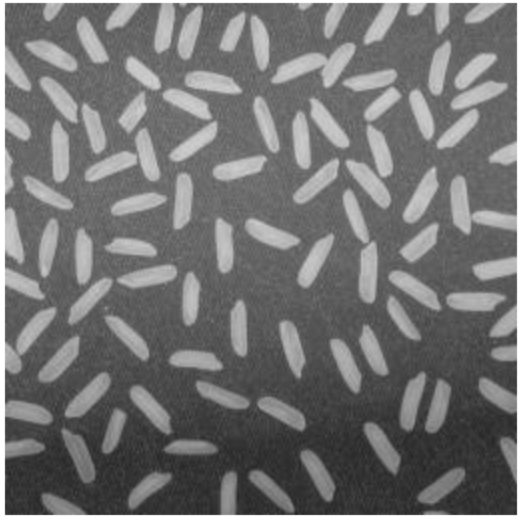
Histograms

How to define intensity transformations adaptively on the image

Image histograms

Histogram of pixel intensities can be used to define intensity transformation

Img I



Histogram $\{h_i\}$

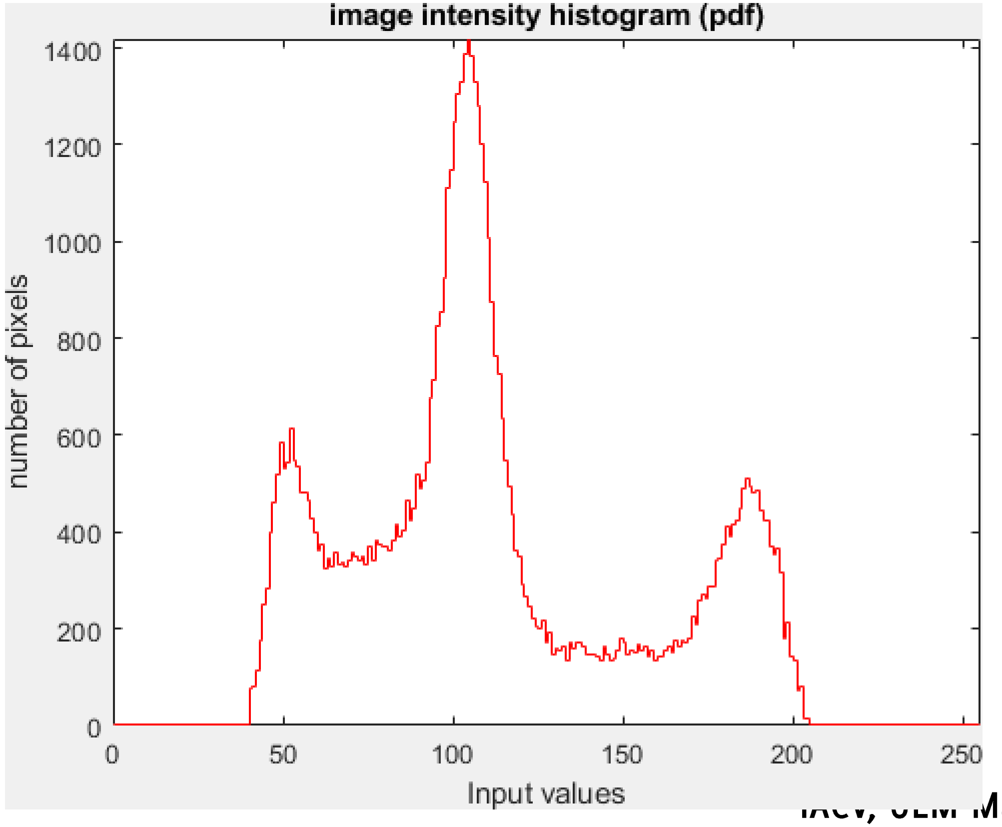


Image histograms

Histogram of pixel intensities can be used to define intensity transformation

Definition

The histogram $\{h_i\}$ associated to an image I is a vector of 256 bins, each corresponding to am intensity value $i = 0, \dots, 255$

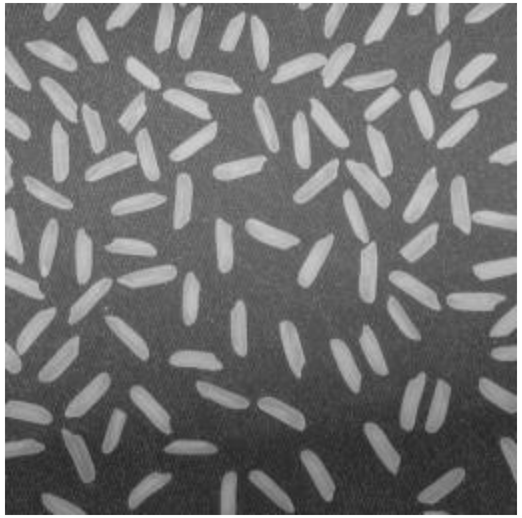
$$h_i = \#\{(r,c), \qquad \text{s.t. } I(r,c) = i\}$$

Where # denotes the cardinality of a set

```
[h, bins] = hist(I, bins)
```

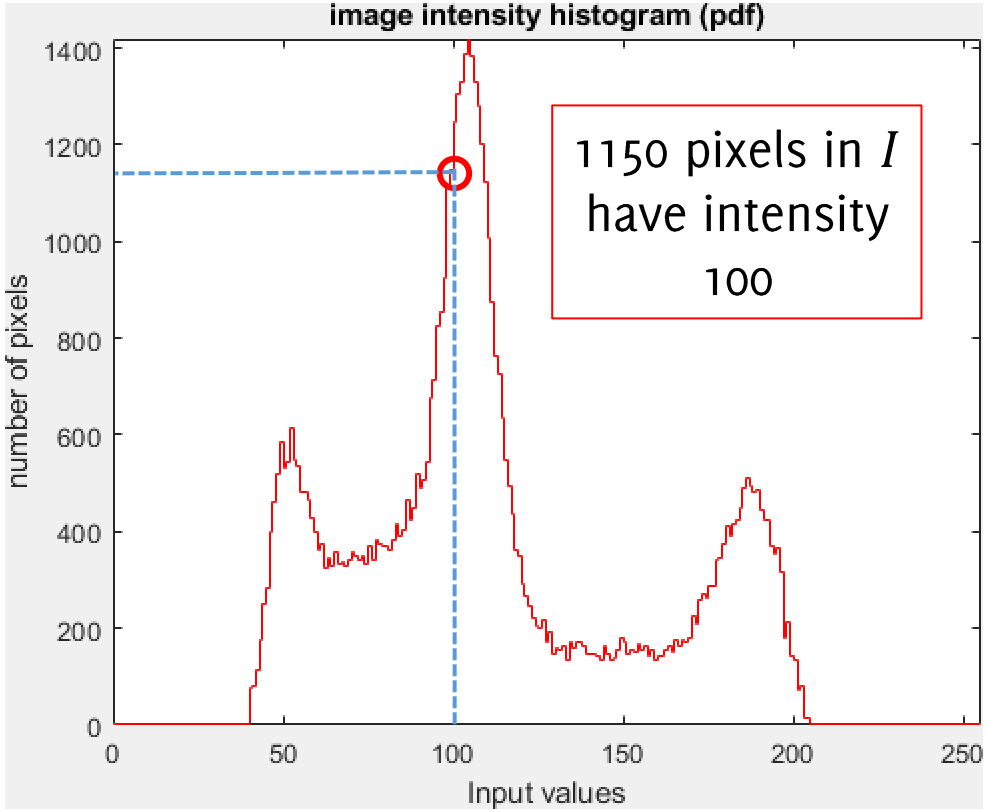
Image histograms

Histogram of pixel intensities can be used to define intensity transformations



img

$h_{100} = 1150$

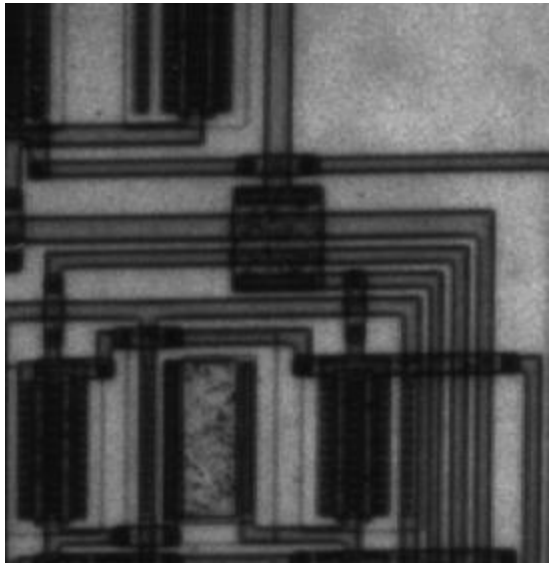


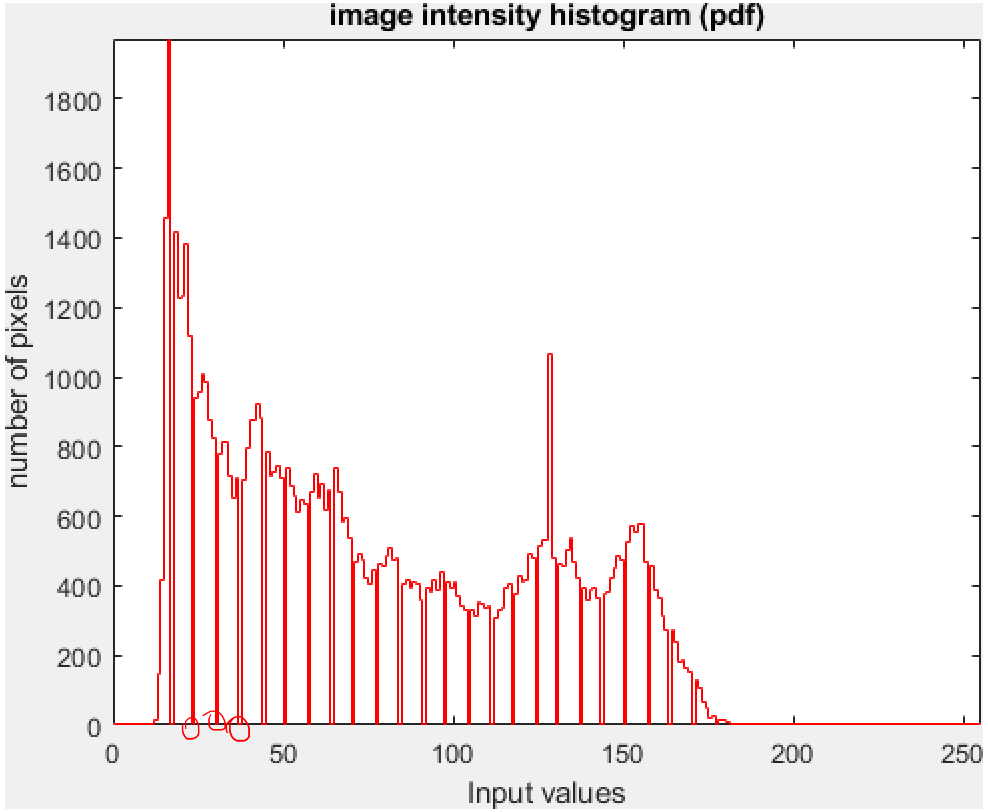
$i = 100$

Histogram

Image histograms

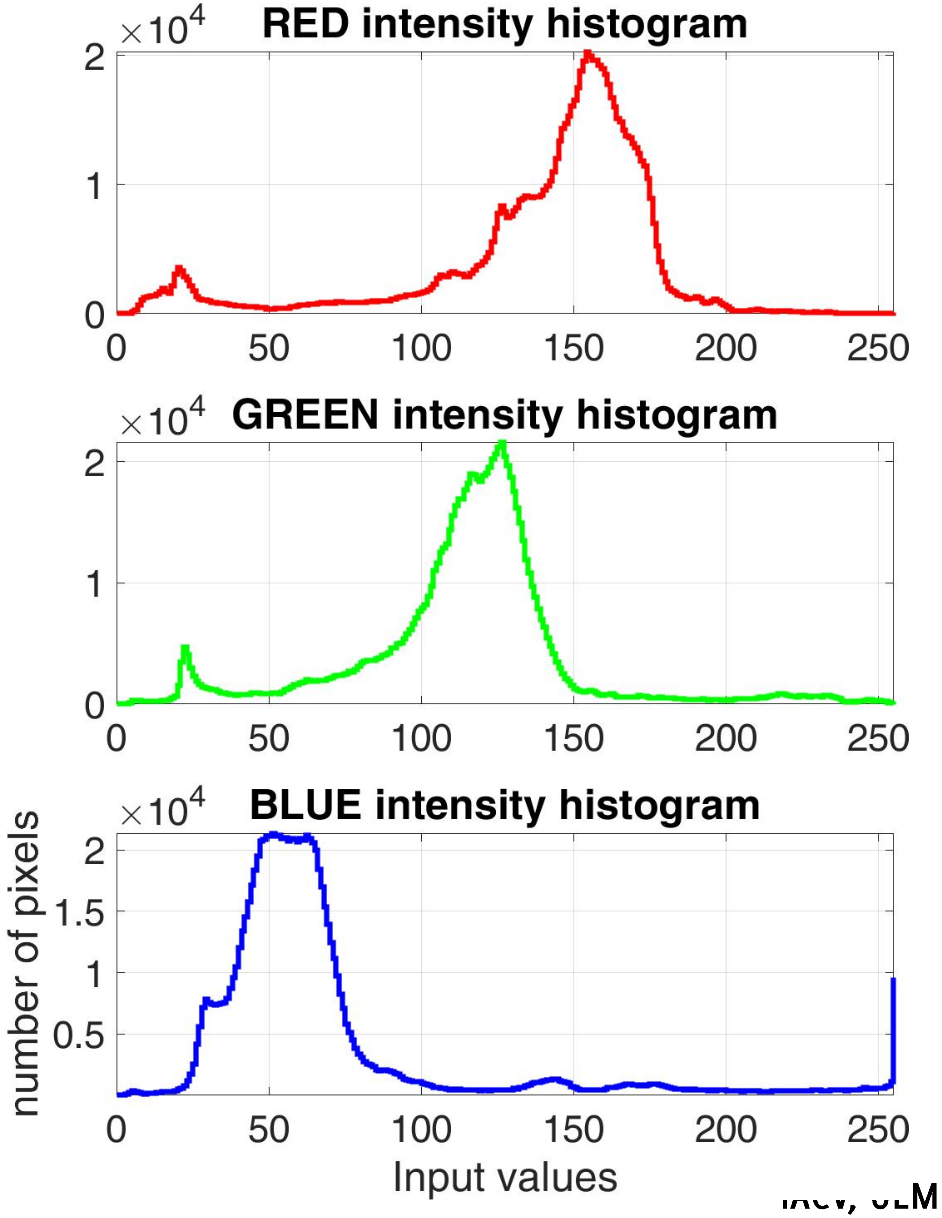
Remember that images assume integer values (uint8), thus there might be intensity values that do not occur in an image





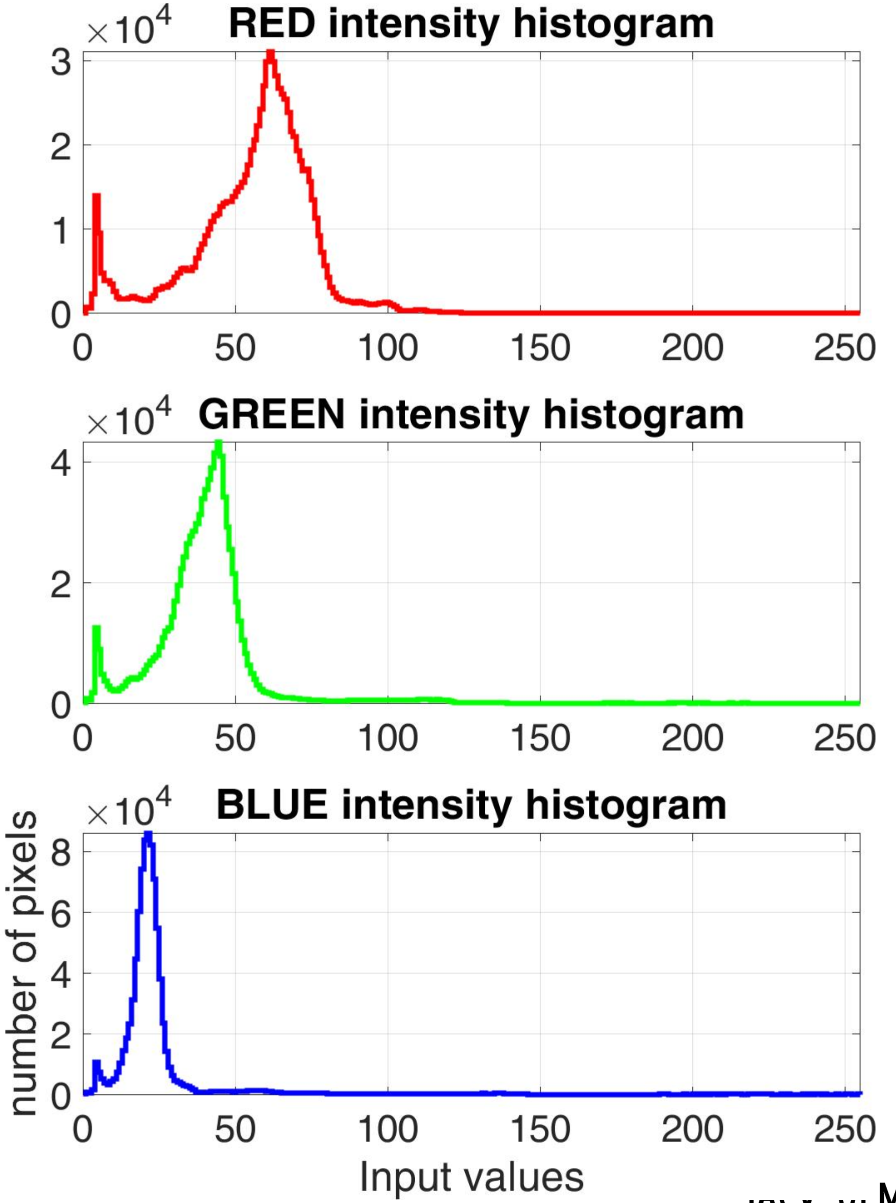
Histogram of a NORMAL image

NORMAL image

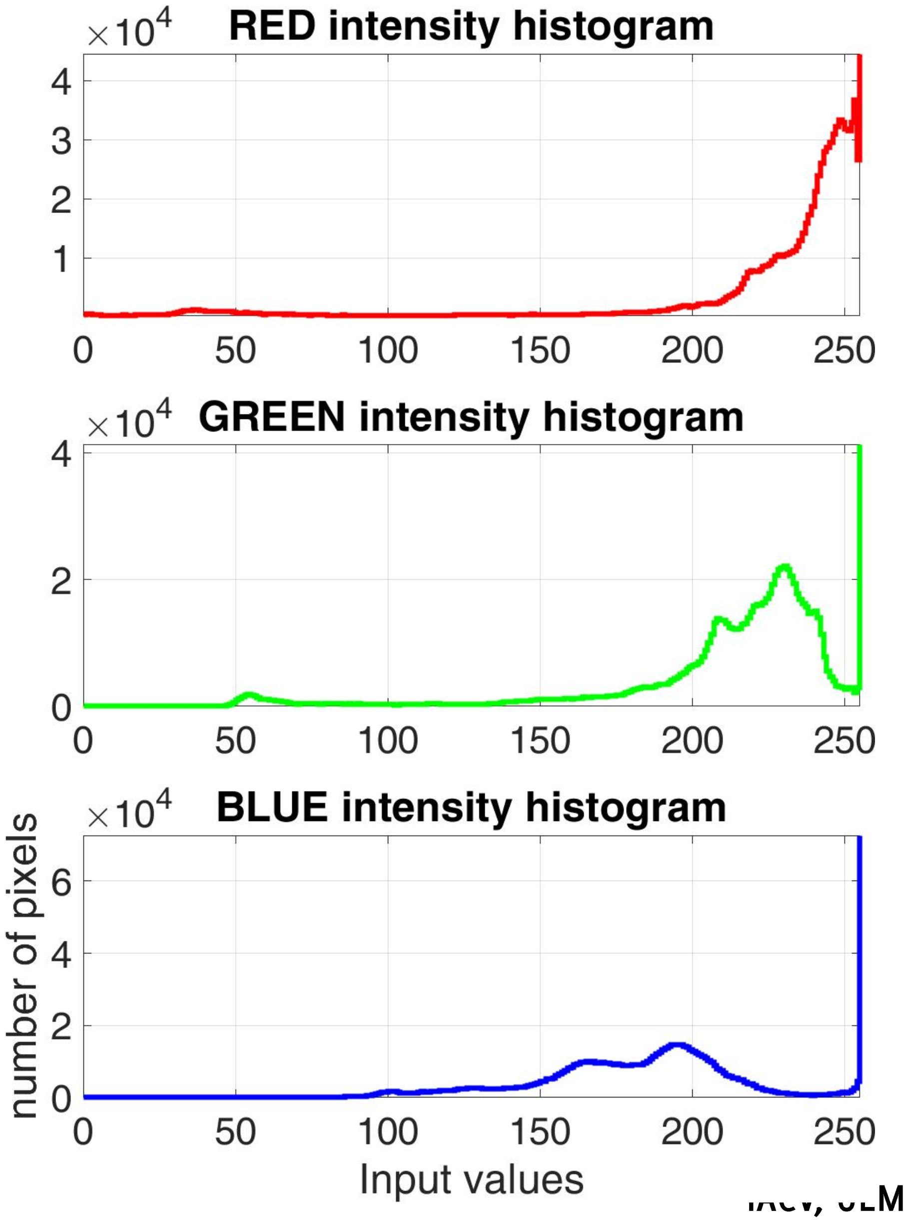


Histogram of an UNDEREXPOSED image

UNDEREXPOSED image



Histogram of an OVEREXPOSED image



There are many saturated pixels

Histogram Equalization

Contrast Enhancement by histogram equalization

Contrast enhancement transformation map the image intensity to the whole range [0,255]

Histogram equalization maps histogram bins. Let

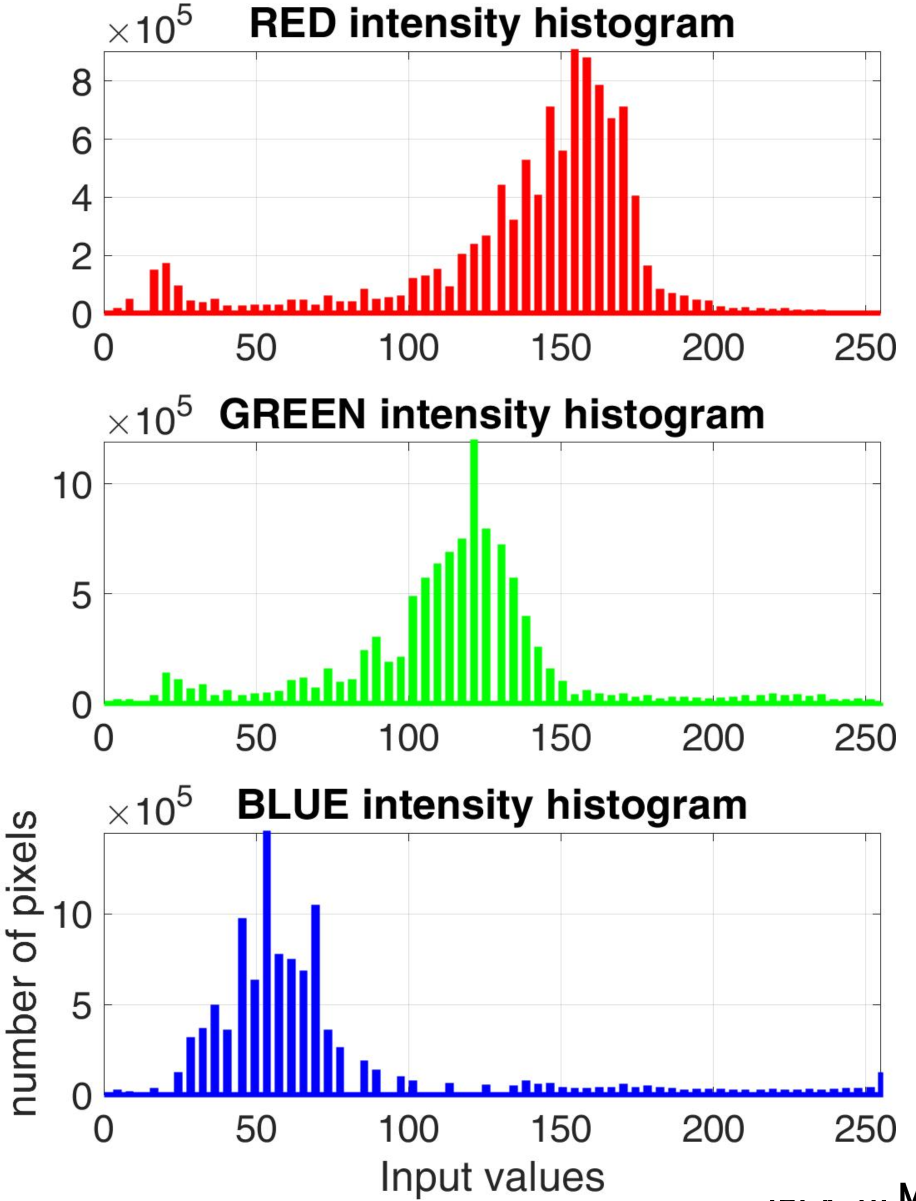
- $[0, L]$ be the intensity range of input image
- $\{h_j\}$ be the histogram of the input image and let $p_j = h_j/N$ be the proportion of pixels having intensity j in the input image

Histogram equalization is defined as

$$T(i) = \text{floor}\left((L-1) \sum_{j=0}^{i} p_j \right)$$

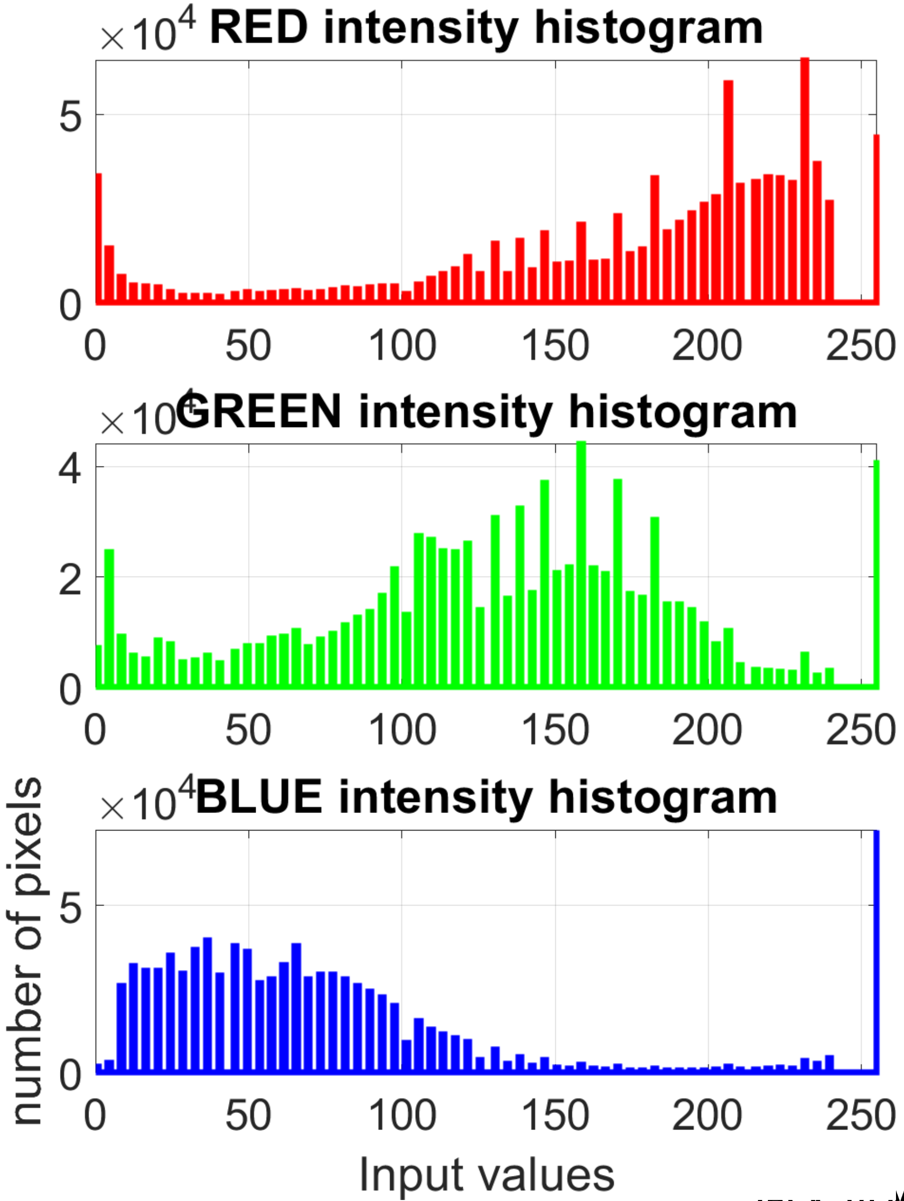
Histogram Equalization Results

UNDEREXPOSED TO NORMAL image

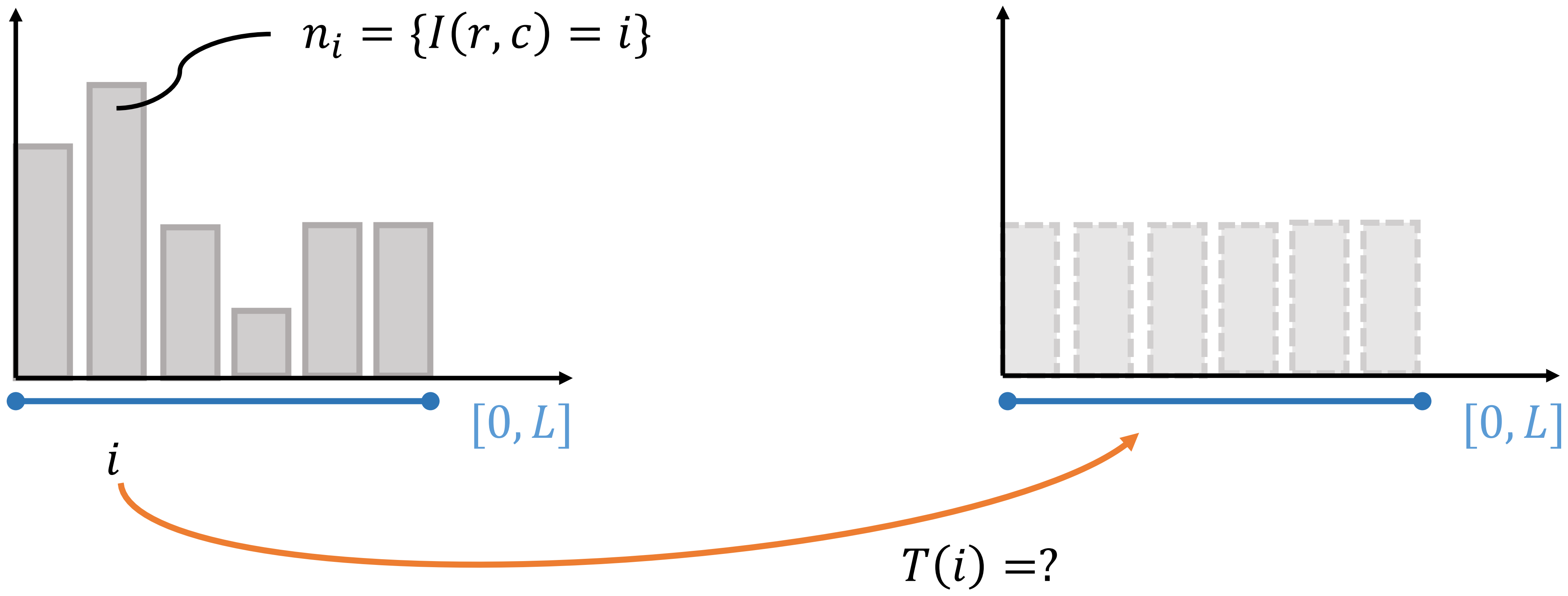


Histogram Equalization Results

OVEREXPOSED EQUALIZED image



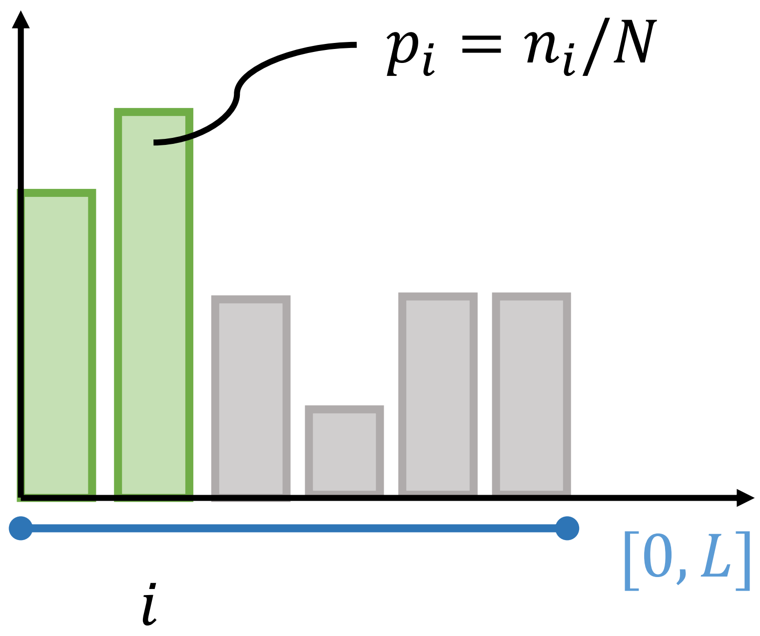
Histogram equalization: sketch of the idea

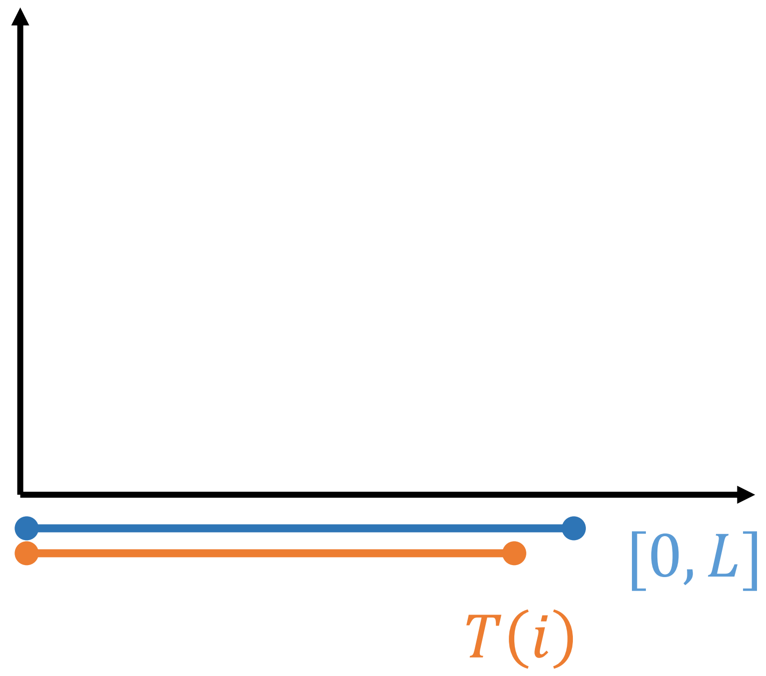


We would like to map the intensity values to get a flat histogram

Histogram equalization: sketch of the idea

Let's move to probability

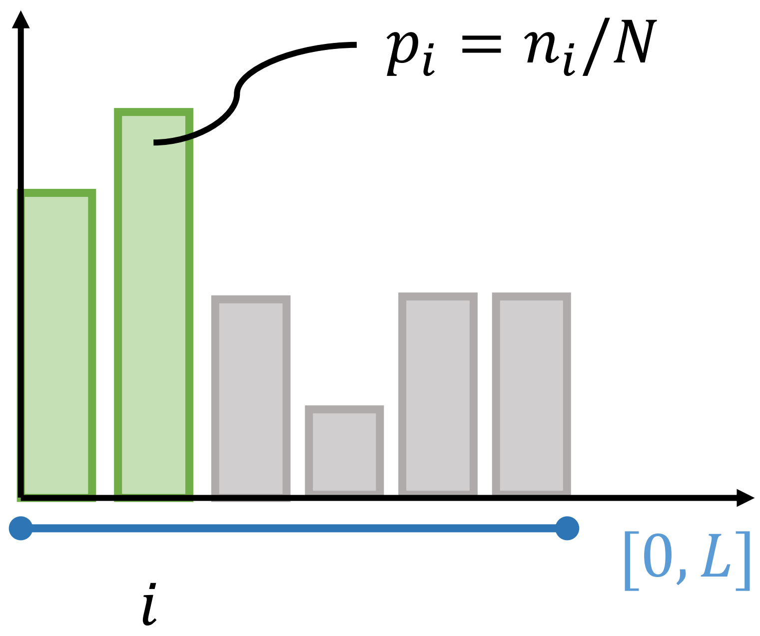


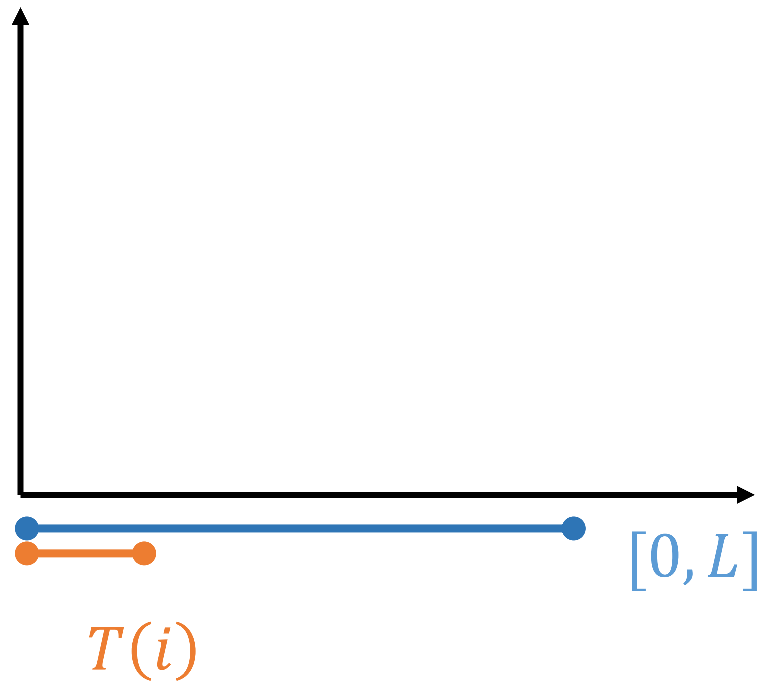


Since T is non decreasing, $T(i)$ is the portion of the available range to represent all the pixels lower or equal to i

Histogram equalization: sketch of the idea

Let's move to probability

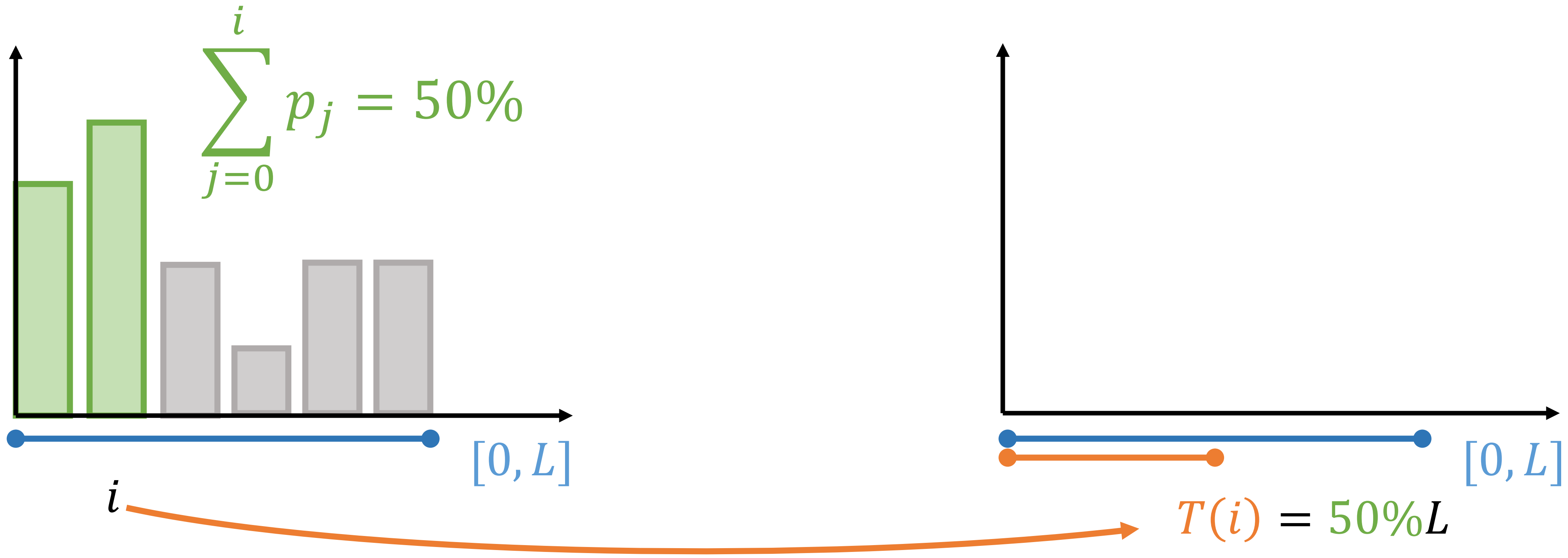




Since T is non decreasing, $T(i)$ is the portion of the available range to represent all the pixels lower or equal to i

Histogram equalization: sketch of the idea

Since we have 50% of pixels we should use 50% of the range

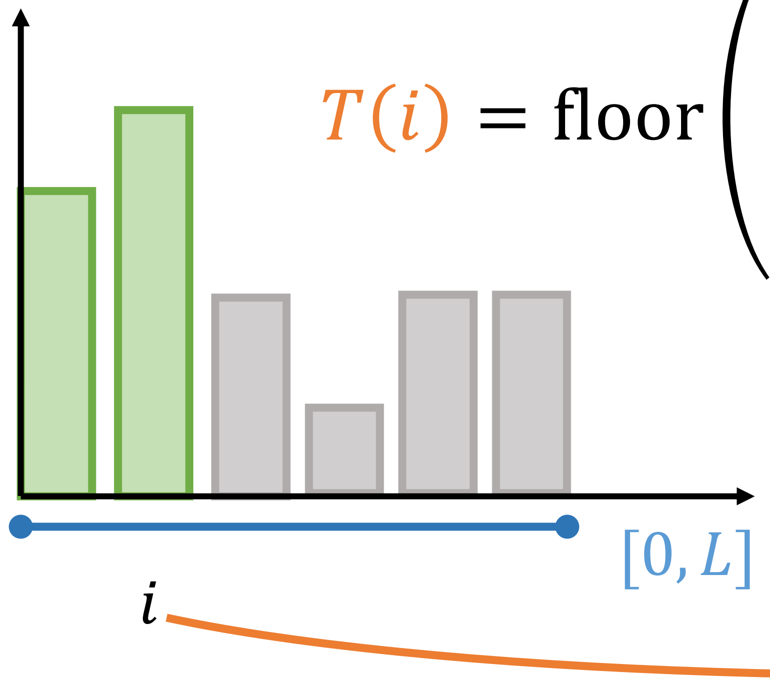


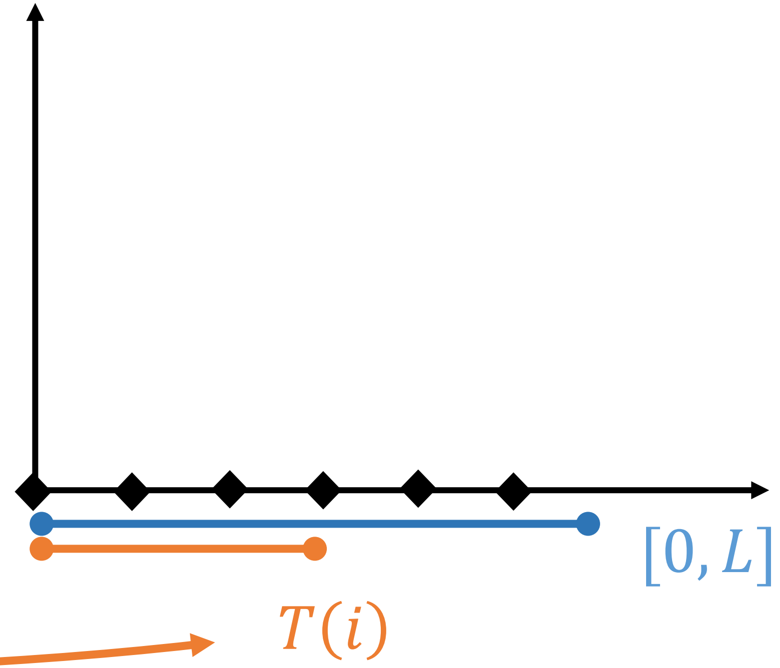
In general $T(i) = L \sum_{j \leq i} p_j$, but we must account that we are dealing with discrete values

Histogram equalization: sketch of the idea

We have to account that we are dealing with discrete values

$$T(i) = \text{floor}\left((L-1) \sum_{j=0}^{i} p_j \right)$$





Contrast Enhancement by histogram equalization

Histogram equalization maps any pdf (histogram of the input image) to a uniform pdf (output image)

$$T(i) = \text{floor}\left((L-1) \sum_{j=0}^{i} p_j \right)$$

Rationale: the **cumulative function** CDF of a random variable I maps the random variable to a uniform distribution, i.e.

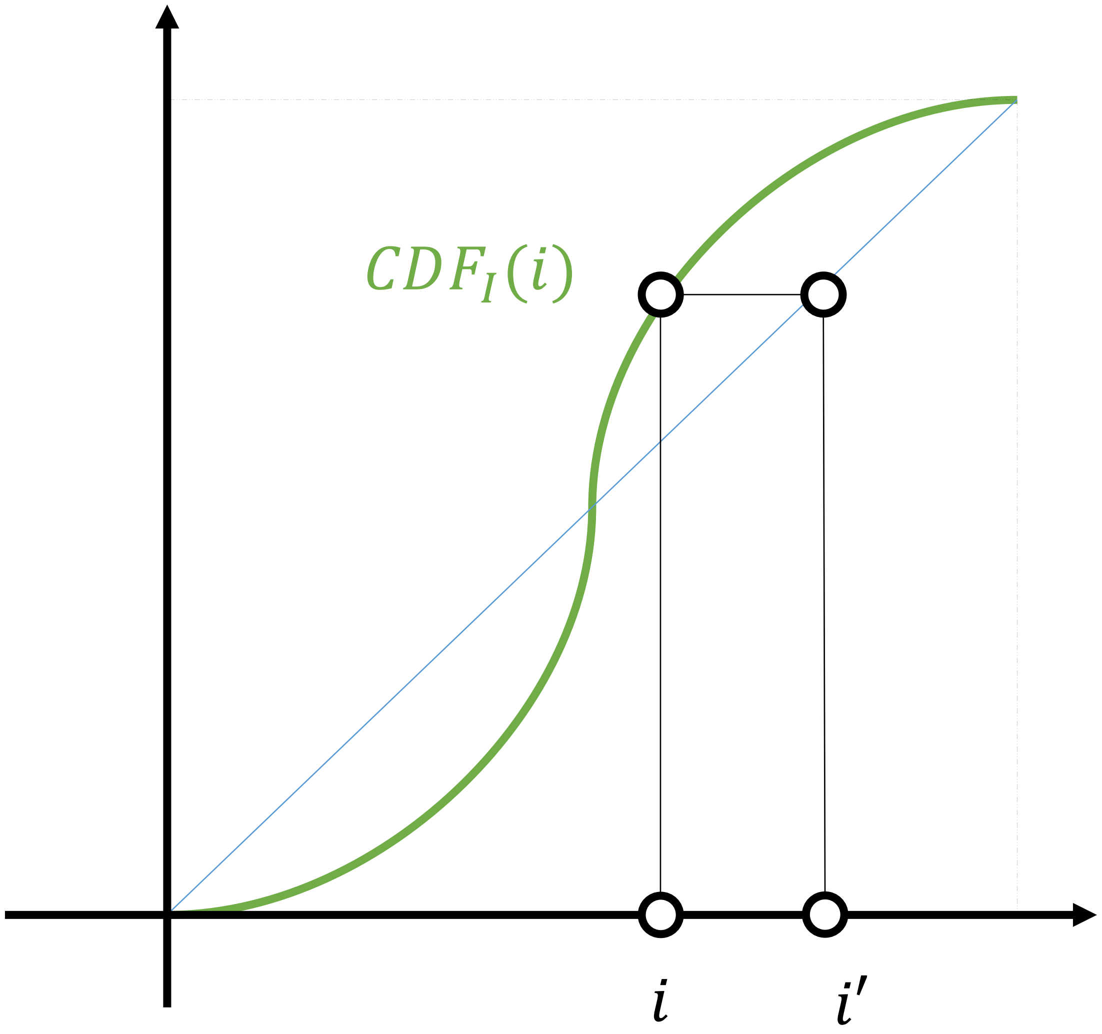
$$PDF(\,CDF(I)\,) \sim U(0,1)$$

The transformation then becomes the cumulative function itself (properly rescaled):

$$T(\cdot) = CDF(\cdot)$$

```
in opencv
cv.equalizeHist(img)
```

Contrast Enhancement by histogram equalization



Contrast Enhancement by histogram equalization

The **cumulative function** CDF of a random variable I maps the random variable to a uniform distribution, i.e.

$$PDF(\ CDF(I)\ ) \sim U(0,1)$$

Dim:

Let $Y = T(X) = CDF_X(X)$, then $CDF_Y(y) = y$.

Proof: $CDF(\bar{y}) = P(Y \leq \bar{y}) = P(T(X) \leq \bar{y}) = P(X \leq T^{-1}(\bar{y})) = T\left(T^{-1}(\bar{y})\right) = \bar{y}$

Implementing histogram equalization

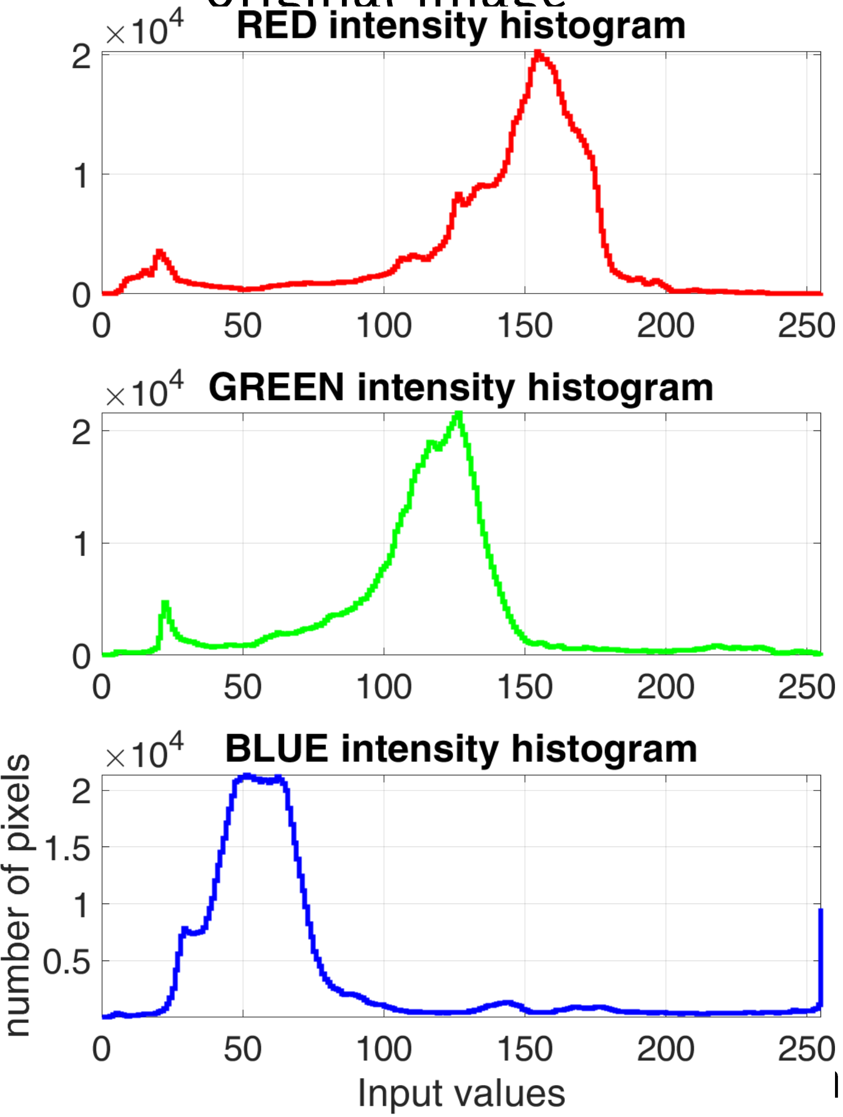
The main cicle looks like:

```
# loop through all the image pixels
for r in range(img.shape[0]):
        for c in range(img.shape[1]):
                intensity_value = img[r,c]
                # transform the pixel to a new intensity value
                new_intensity_value = np.floor(255 * np.sum(p[0:intensity_value]))
                img_T[r,c] = new_intensity_value
```

Where p is the vector collecting the pdf of the input image

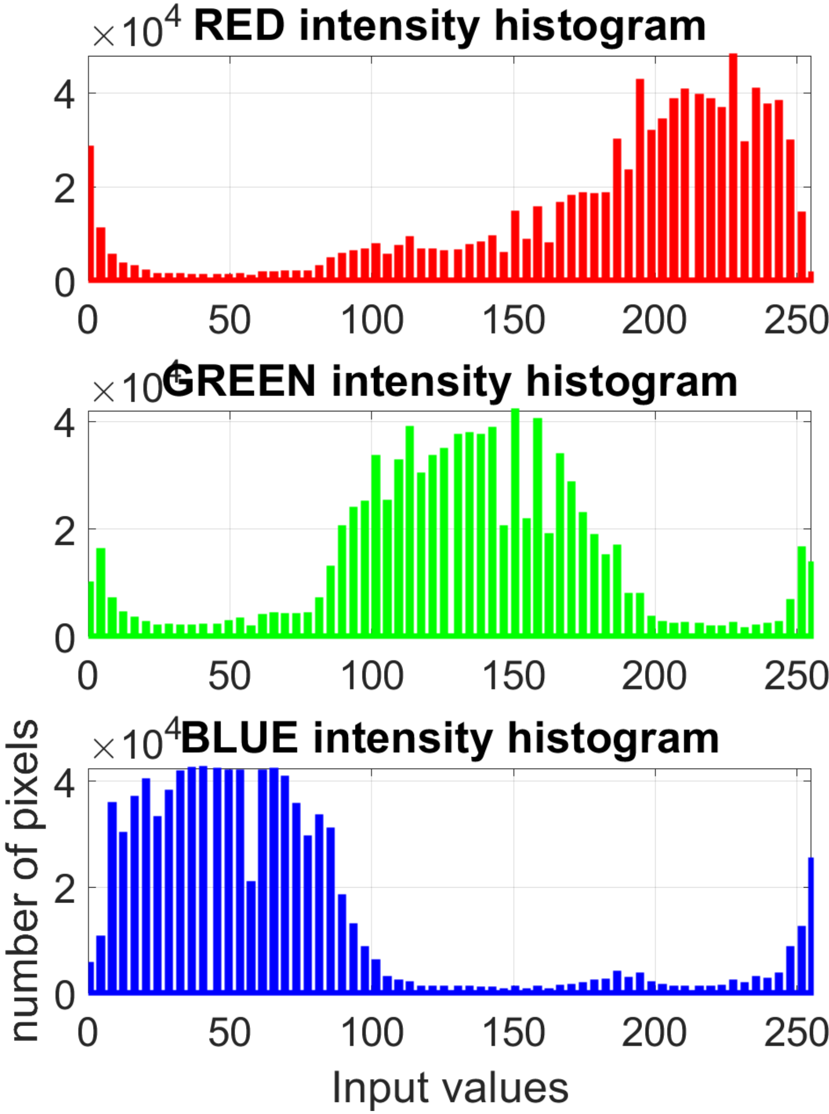
Histogram Equalization Results

Original image



NORMAL EQUALIZED image

Equalized image

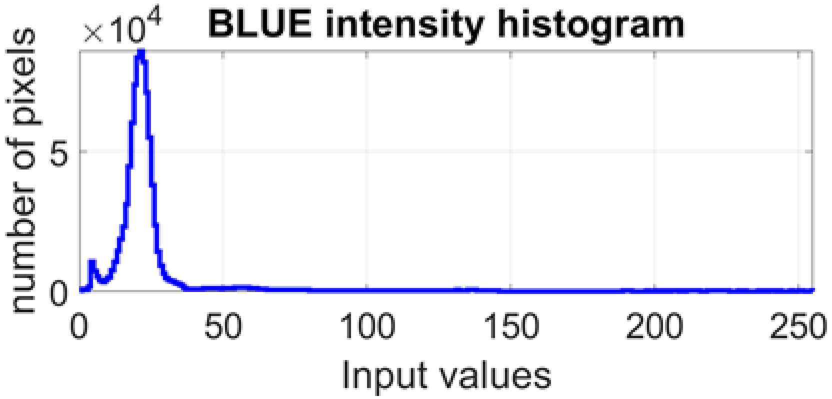


Rmk it is not possible to increase the number of different intensity that were in the input by this transformation

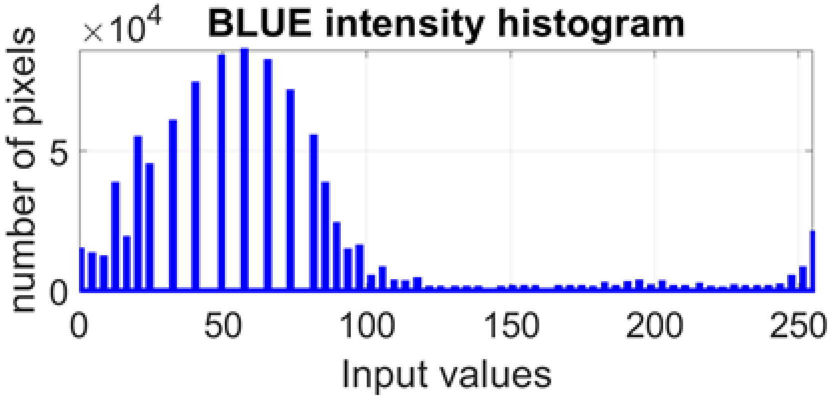
Equalization can not create new intensity values!

UNDEREXPOSED image

UNDEREXPOSED EQUALIZED image



Histogram equalization

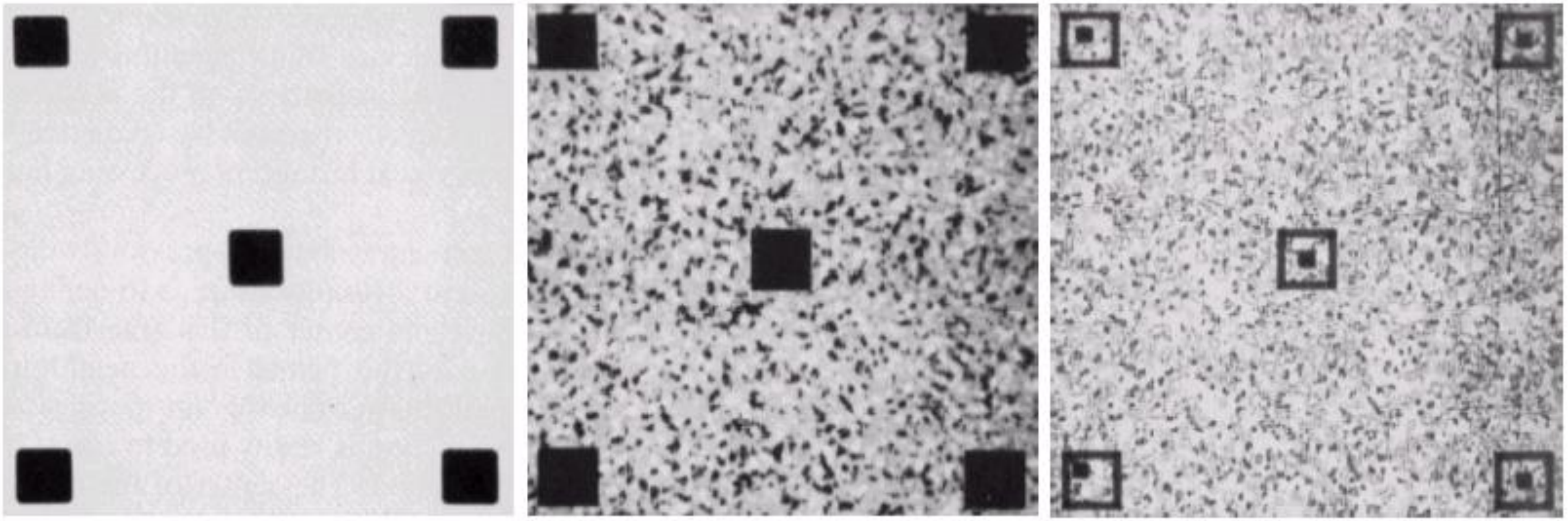


This does not look like uniform since there are only 50 different values in the input and cannot become more by that transformation

Local Histogram Equalization

For each pixel (r, c)

- compute the histogram in a $N \times N$ neighborhood of (r, c)
- compute the local equalization function $T_{r,c}$
- compute the output value in the pixel (r, c) as $T_{r,c}\big(I(r, c)\big)$



a b c

From GW **FIGURE 3.23** (a) Original image. (b) Result of global histogram equalization. (c) Result of local histogram equalization using a 7×7 neighborhood about each pixel.

Histogram Matching

Histogram Matching

Estimate the intensity transformation mapping an histogram to any target distribution.

For instance, given two images I_1 and I_2, estimate the transformation mapping the histogram of I_1 to the histogram of I_2

Histogram Matching

Idea: Estimate the transformation that makes their cumulative density functions to be the same

The transformation

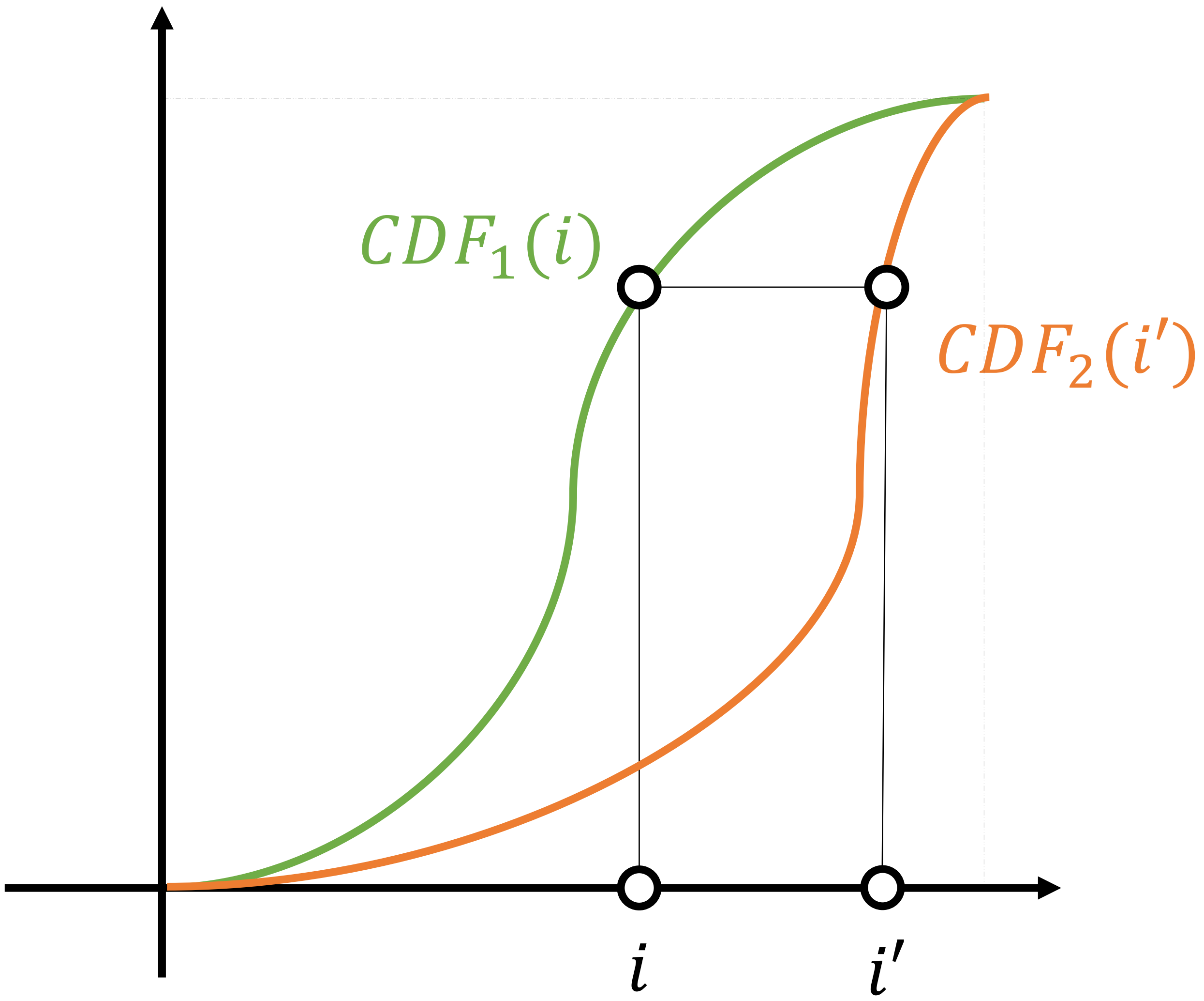
$$i' = T(i), \qquad \text{such that}$$
$$CDF_1(i) = F(i) = G(i') = CDF_2(i')$$

Solves this problem and can be easily computed since histograms are discrete

$$i'' = G^{-1}(F(i))$$

Being F, G the CDF of the two images that are discrete, positive and non-decreasing

Histogam Matching $T: i \mapsto i'$



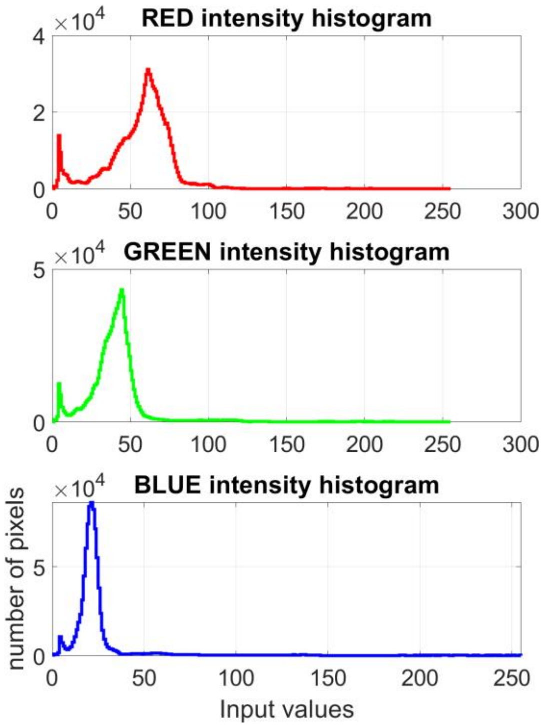

```
i_prime = np.argmin(np.abs(cdf_1[i] - cdf_2))
```

Histogram matching

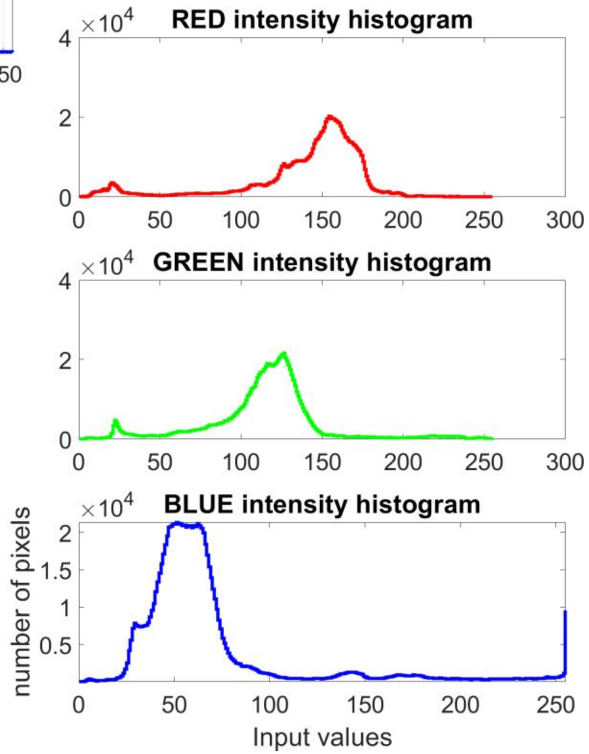
```
def histogram_match(source_image, target_image):
    # Compute histograms of the source and target images
    source_hist, _ = np.histogram(source_image, bins=256, range=(0, 256), density=True)
    target_hist, _ = np.histogram(target_image, bins=256, range=(0, 256), density=True)

    # Compute cumulative distribution functions (CDFs) of the histograms
    source_cdf = source_hist.cumsum()
    target_cdf = target_hist.cumsum()

    # Initialize an empty mapping for pixel values
    mapping = np.zeros(256, dtype=np.uint8)

    # Perform histogram matching
    for i in range(256):
        # Find the closest value in the target CDF for each value in the source CDF
        closest_value = np.argmin(np.abs(source_cdf[i] - target_cdf))

        # Map the source pixel value to the closest target pixel value
        mapping[i] = closest_value

    # Apply the mapping to the source image
    matched_image = mapping[source_image]

    return matched_image
```

Histogam Matching

UNDEREXPOSED image

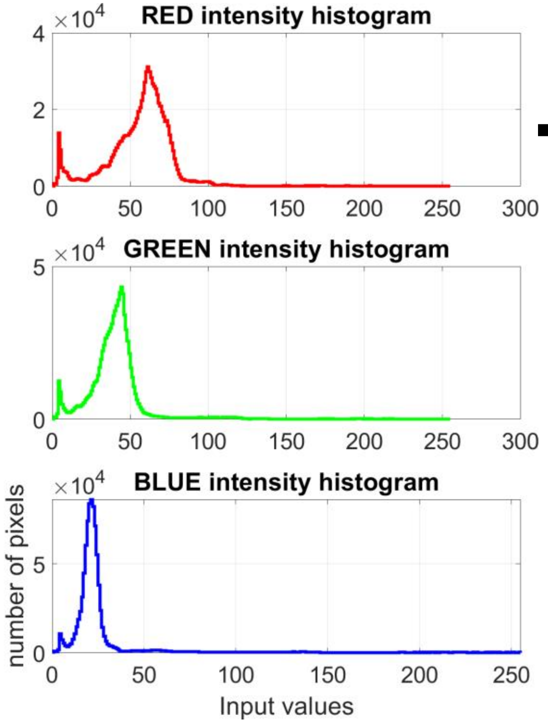


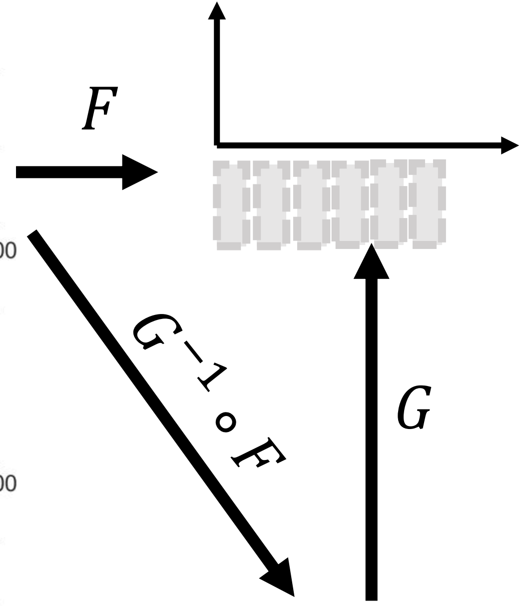
NORMAL image



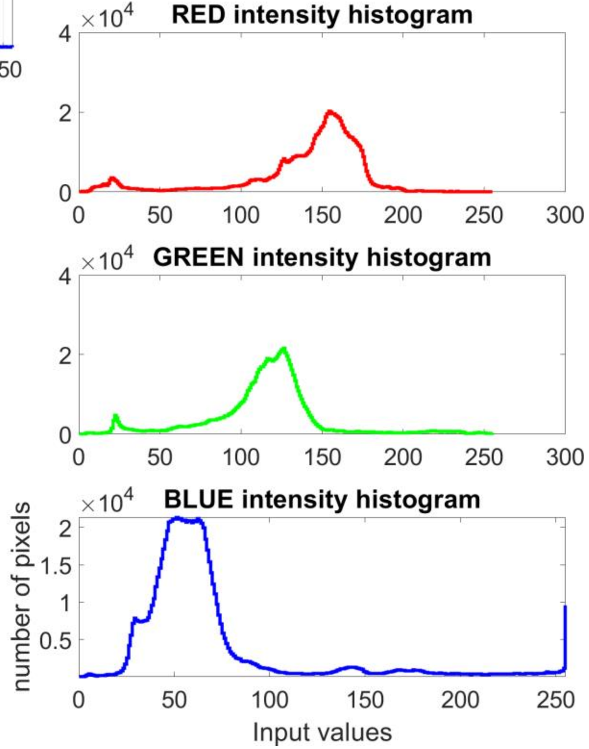
Histogam Matching

UNDEREXPOSED image



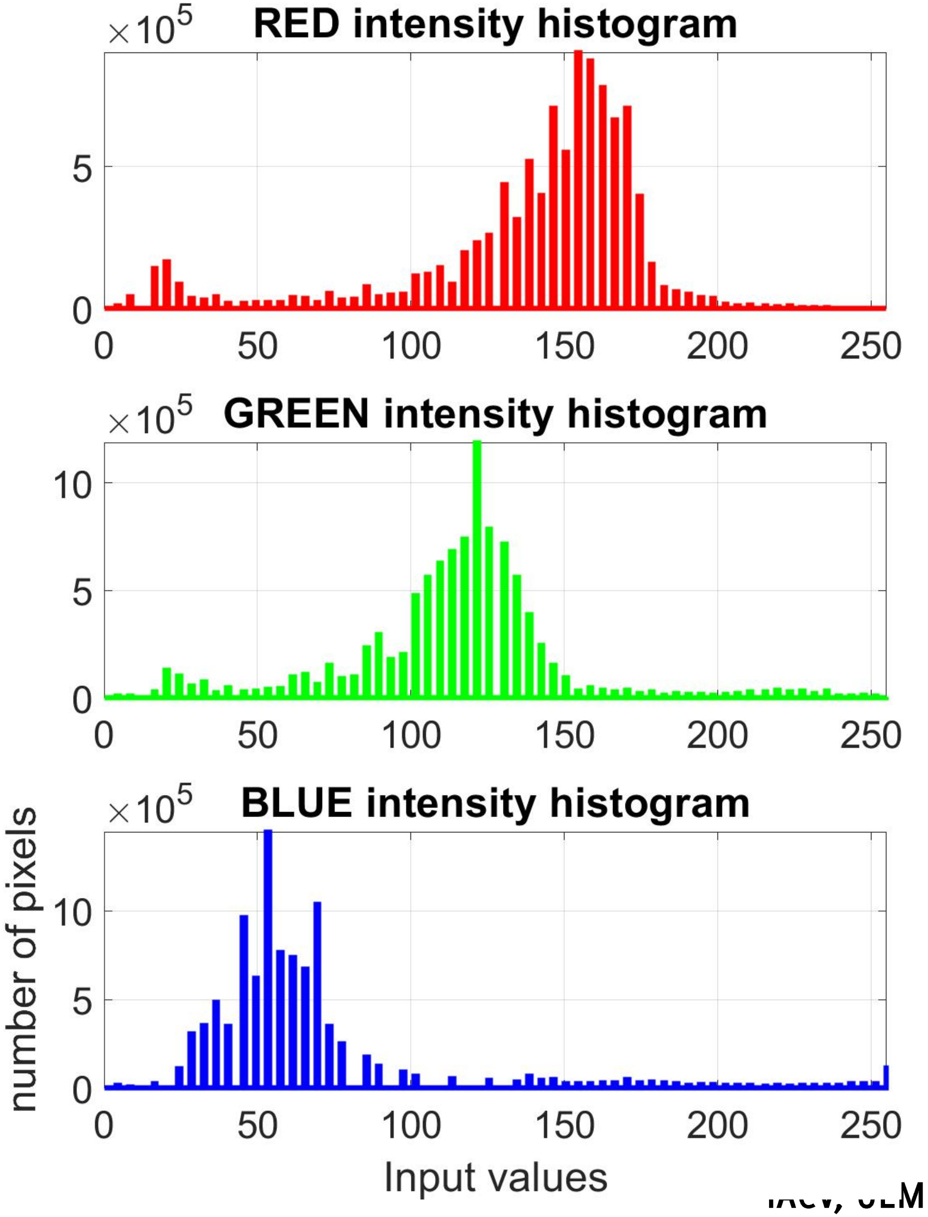


NORMAL image



Histogam Matching Results

UNDERPOSED TO NORMAL image

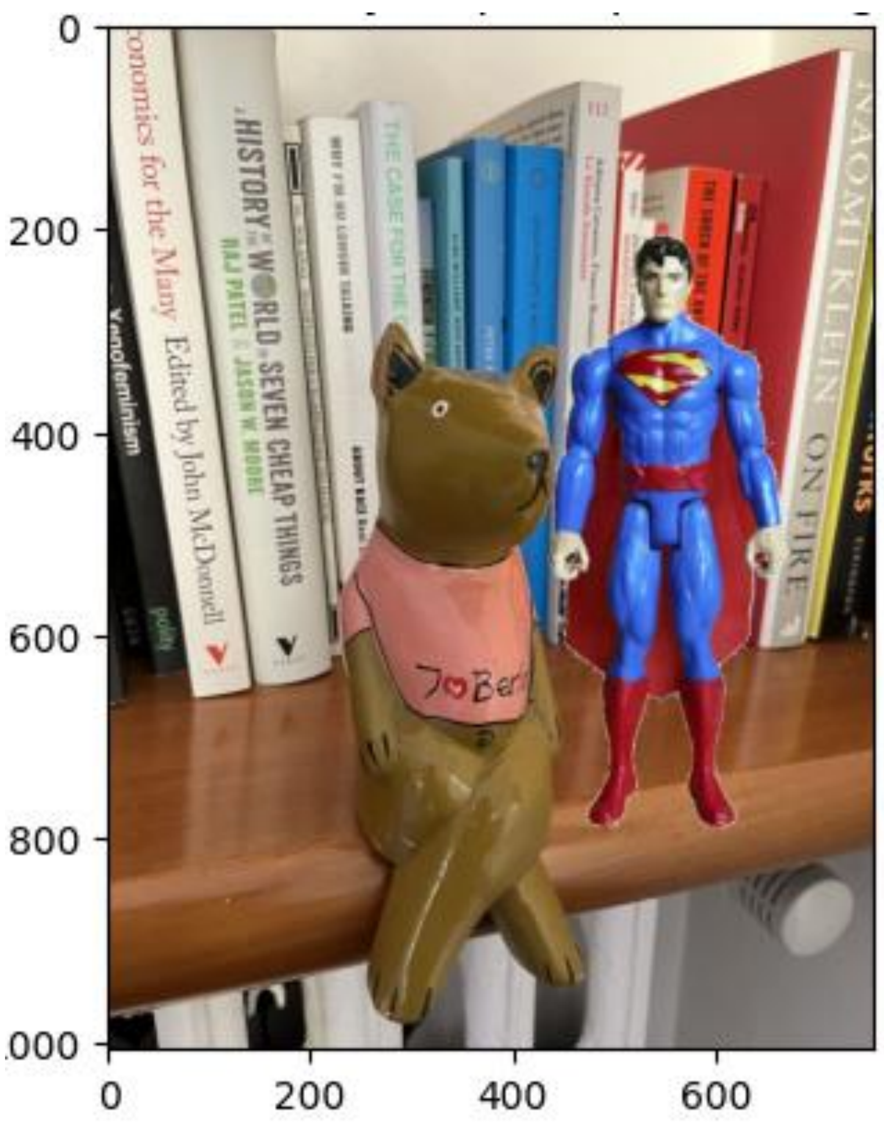


UNDEREXPOSED

NORMAL

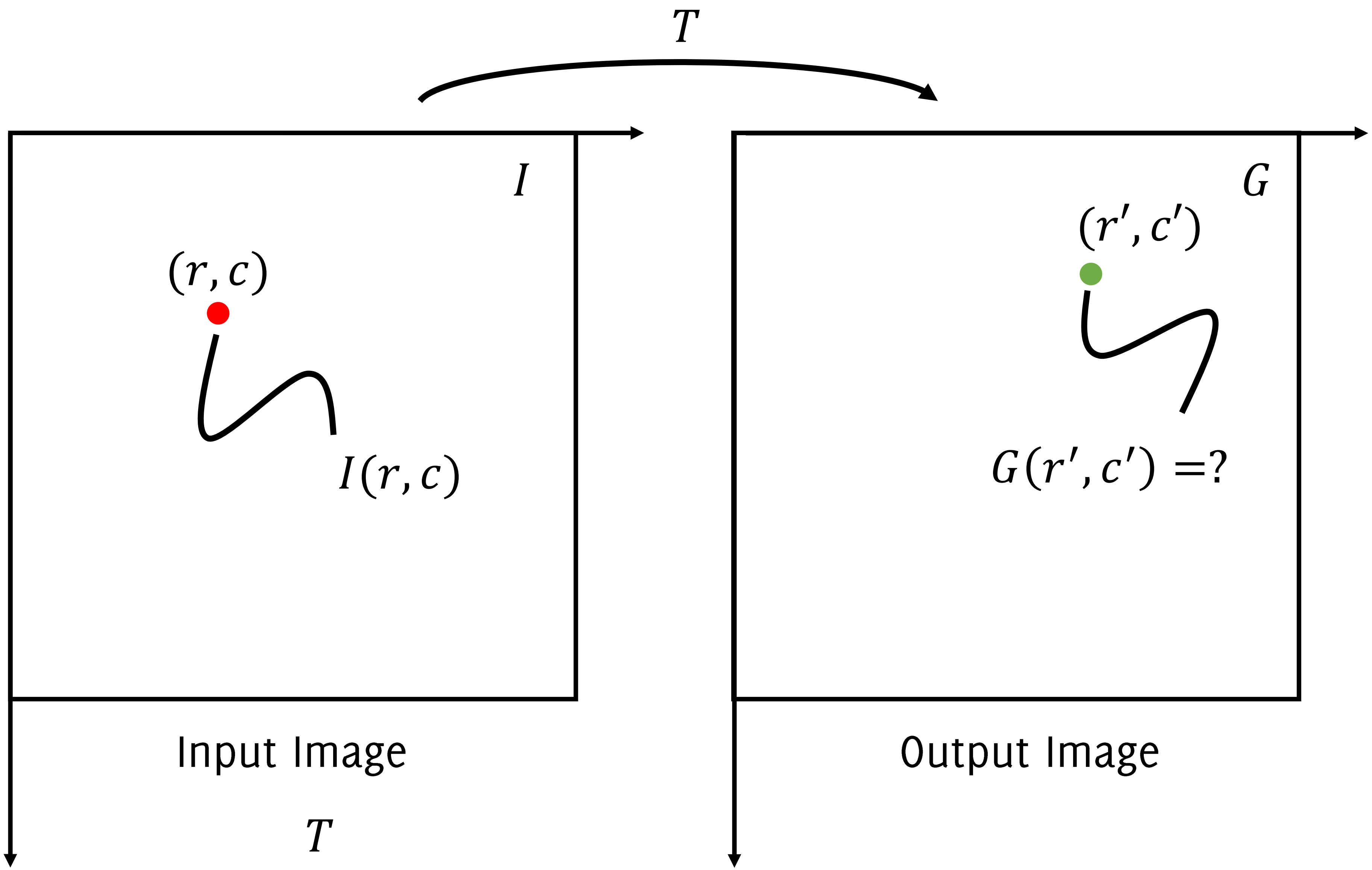
UNDEREXPOSED TO NORMAL

Can you improve the result? Do it yourself!



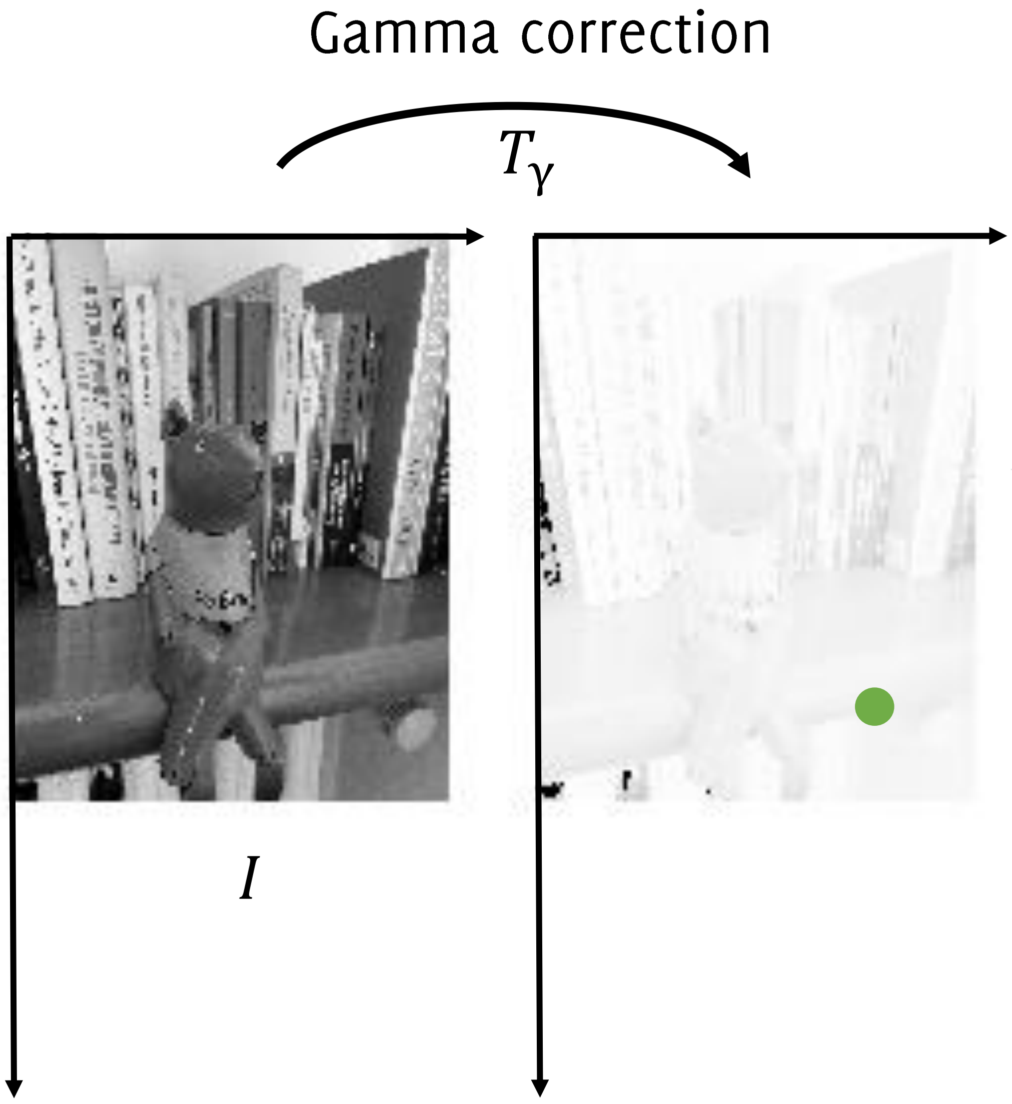
You can apply histogram matching to the “bear” and “superman” image to obtain better superimposition result!

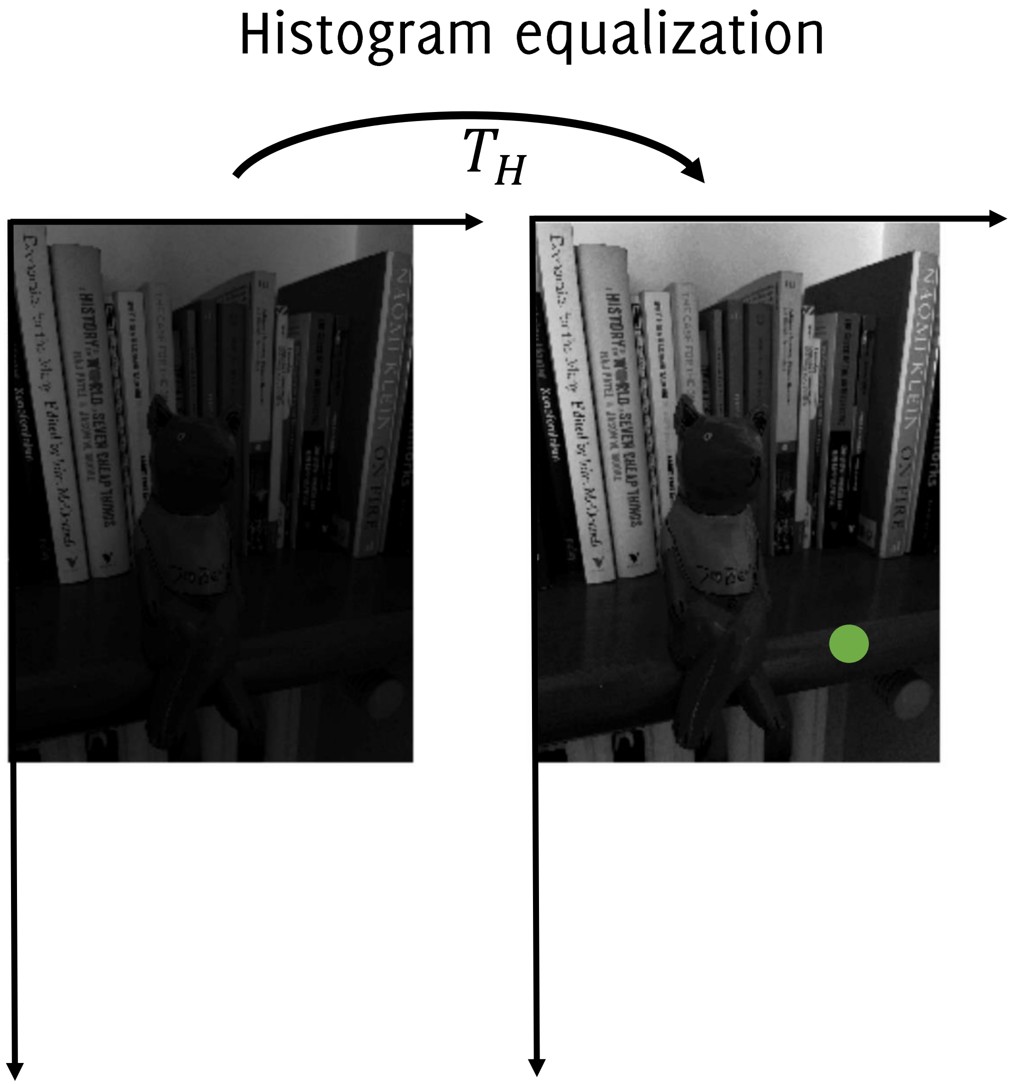
Recap: Image transformations



T

Recap: Image transformations - examples





Recap: Image transformations - examples

Gamma correction

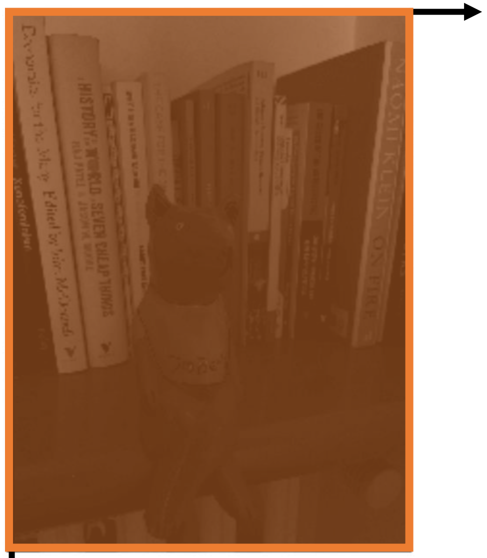
T_γ

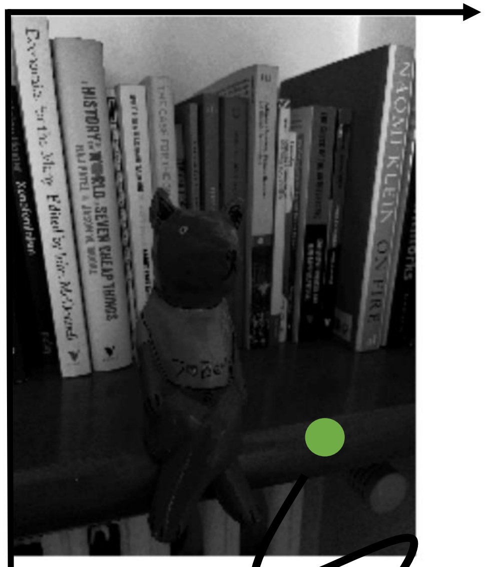
I

$G(r,c) = 255(I(r,c)/255)^{\gamma}$

Histogram equalization

T_H





Depends on all the values of I

Inputs

Both Gamma Correction and Histogram Equalization transform:

- Takes as input the intensity of a single pixel
- Returns the intensity of a single pixel

These are pixel-wise transfoms

The underlying functions are **parametric**

- Contrast enhancement depends on γ
- Histogram equalization depends on the image histogram (so it is adaptive to the image)

Now we will see transformations taking as input a set of intensities and returning a single intensity