Introduction

Objectives

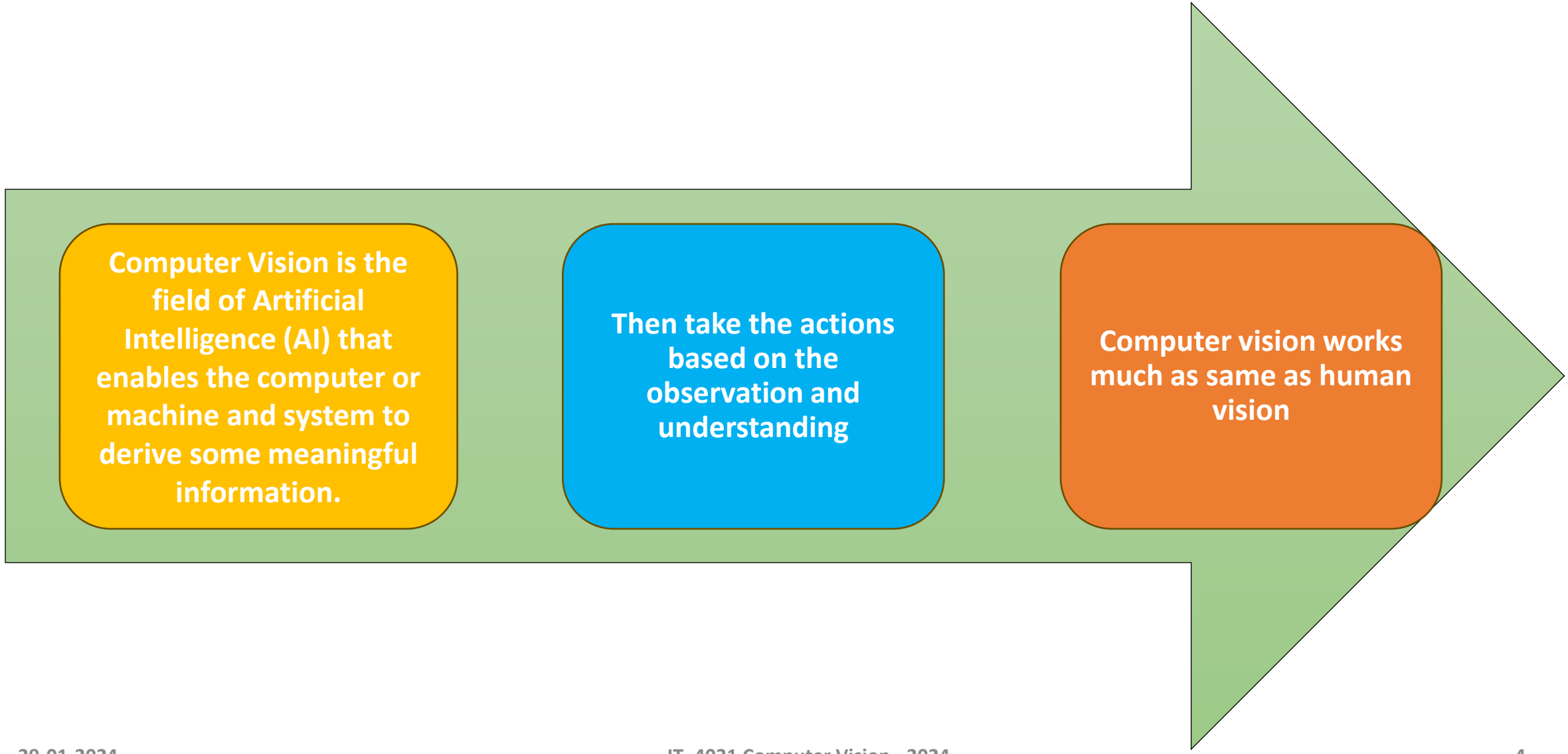
To learn and appreciate the following concepts:

- ✓Introduction to computer vision and its applications
- ✓Image and Image Formation
- ✓Image Filtering Techniques
- ✓Image Transformation and Color models

Session outcome

- At the end of session the student will be able to understand:
  - What is Computer vision and its applications
  - Fundamentals of image and image formation
  - Importance of image filtering techniques
  - Understanding image transformation and color models

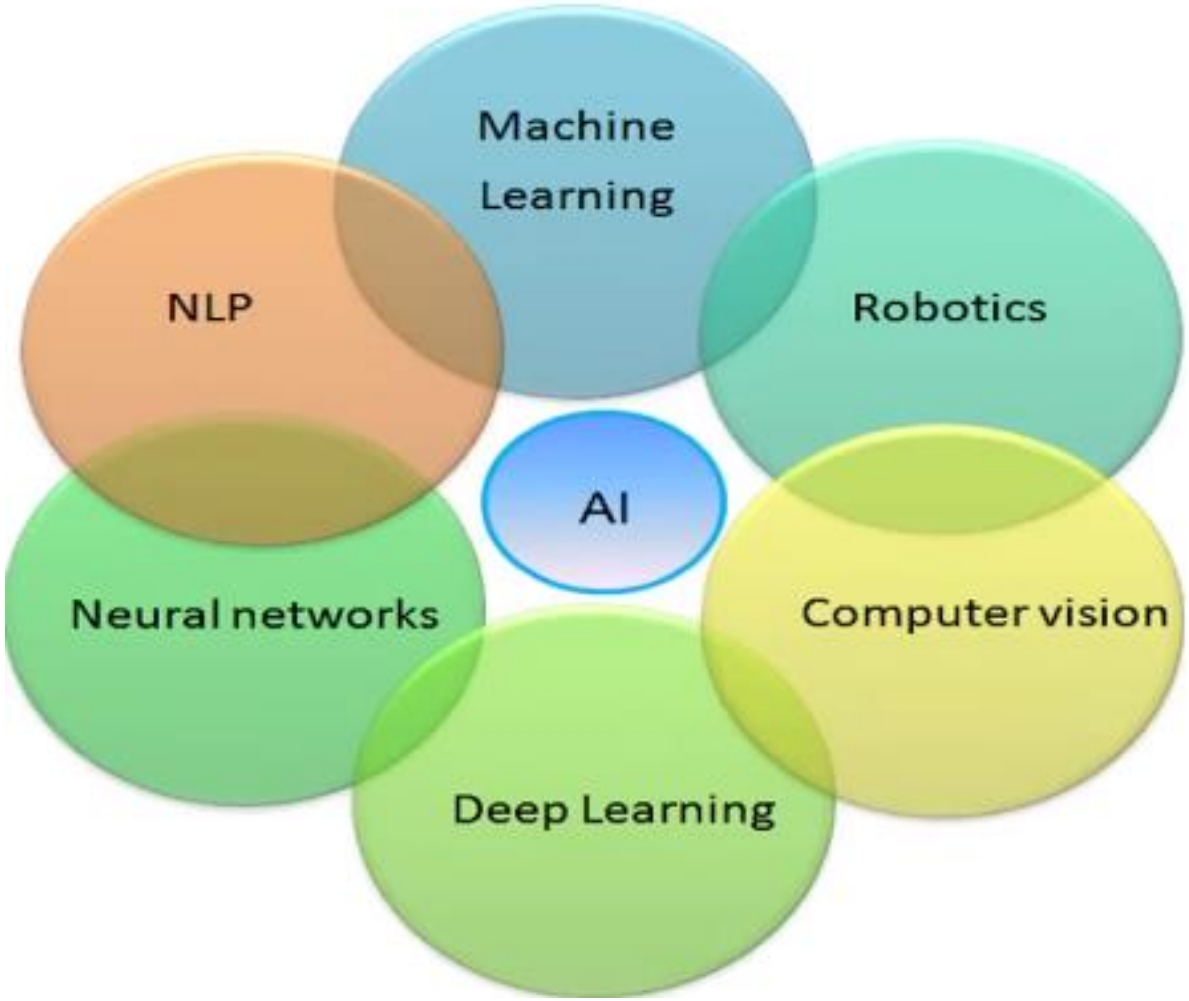
Introduction to Computer Vision



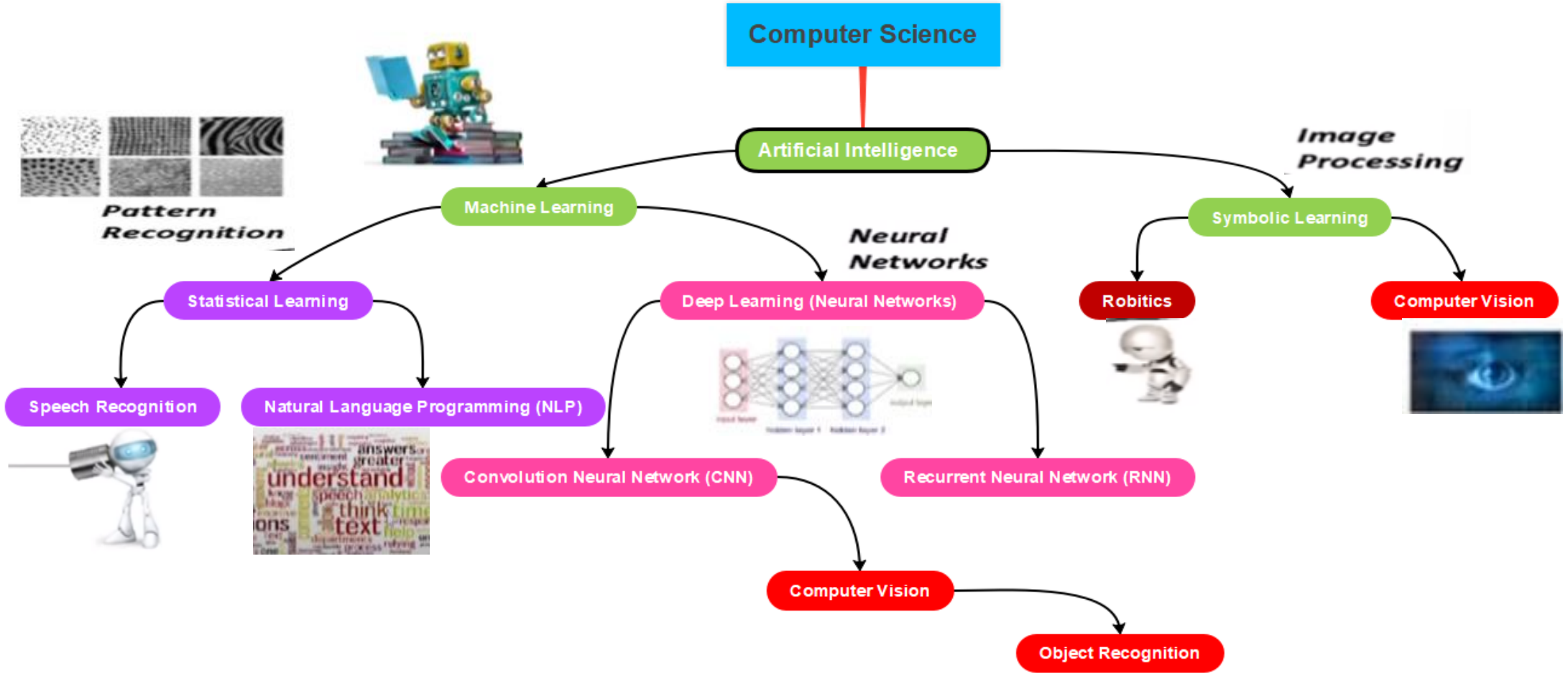
Introduction to Computer Vision

- **Computer vision is a subfield of AI (Artificial Intelligence), which enables machines to derive some meaningful information from any image, video, or other visual input and perform the required action on that information.**
- Computer vision is like eyes for an AI system, which means if AI enables the machine to think, computer vision enables the machines to see and observe the visual inputs.
- Computer vision technology is based on the concept of teaching computers to process an image or a visual input at pixels and derive meaningful information from it.

Introduction to Computer Vision – Artificial Intelligence



Introduction to Computer Vision – Artificial Intelligence

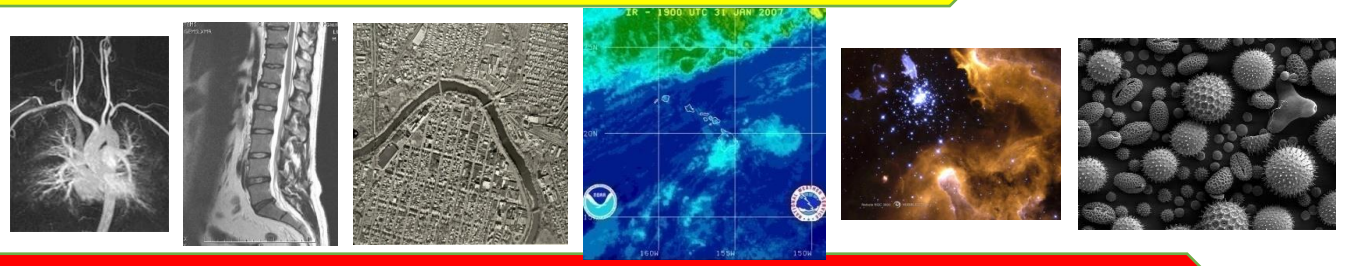


Why Computer Vision

- **Billions of images/videos captured per day**

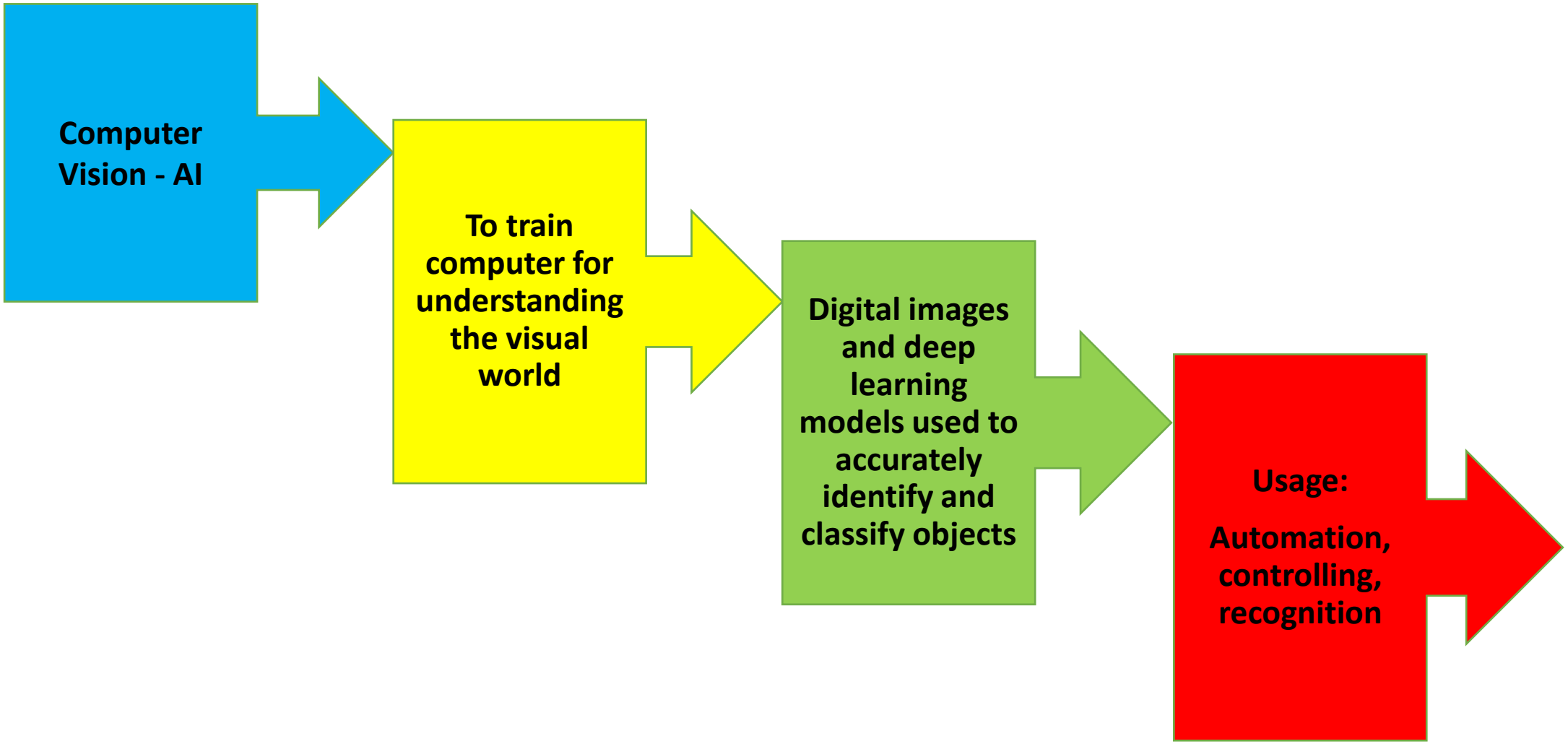
- Huge number of potential applications

- **Computer Vision becoming increasingly important in various industries including safety, productivity and accuracy.**

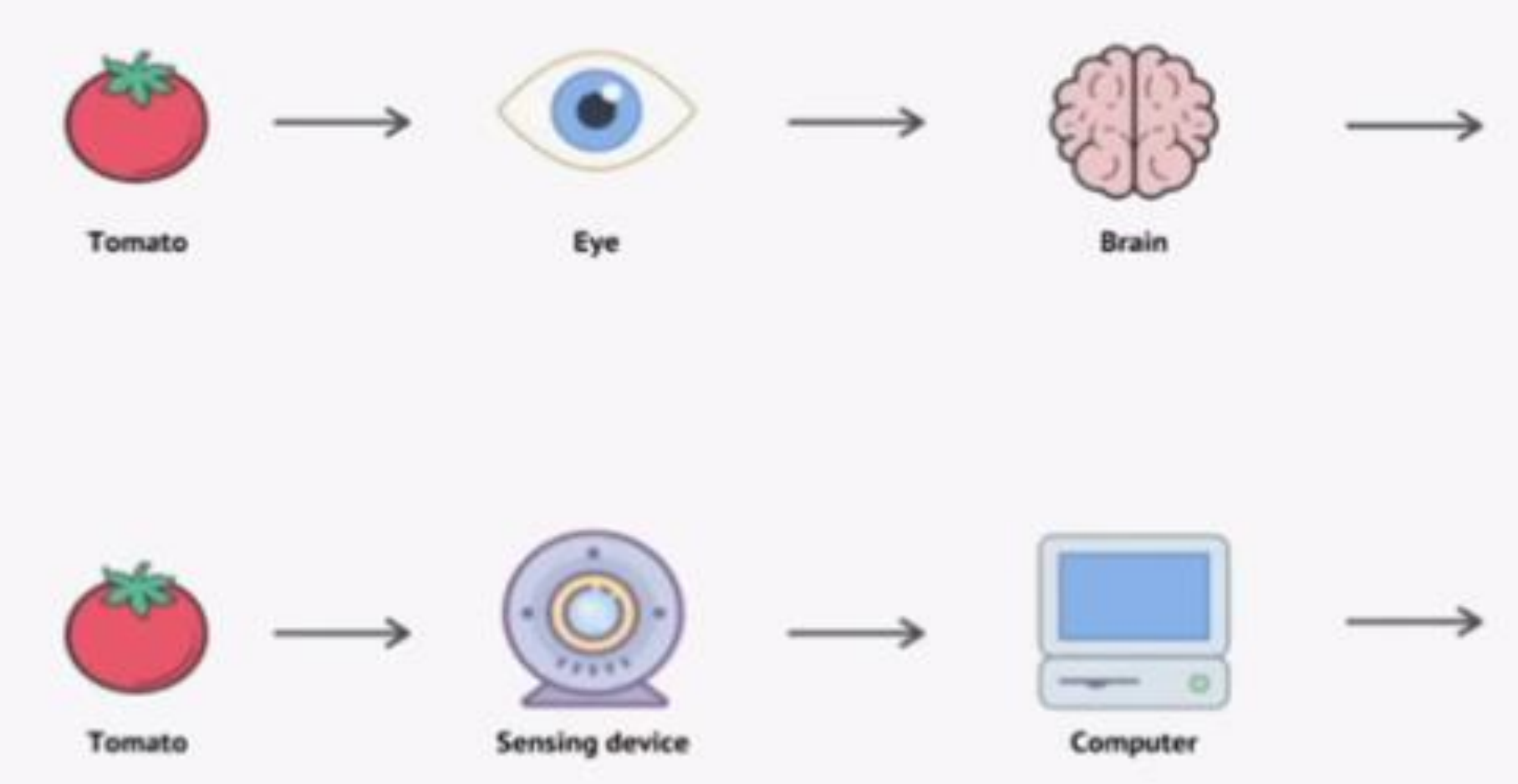


- **In the field of artificial intelligence technology, computer vision adds great importance to the AI, and without computer vision there is no future to AI.**

What is Computer Vision?



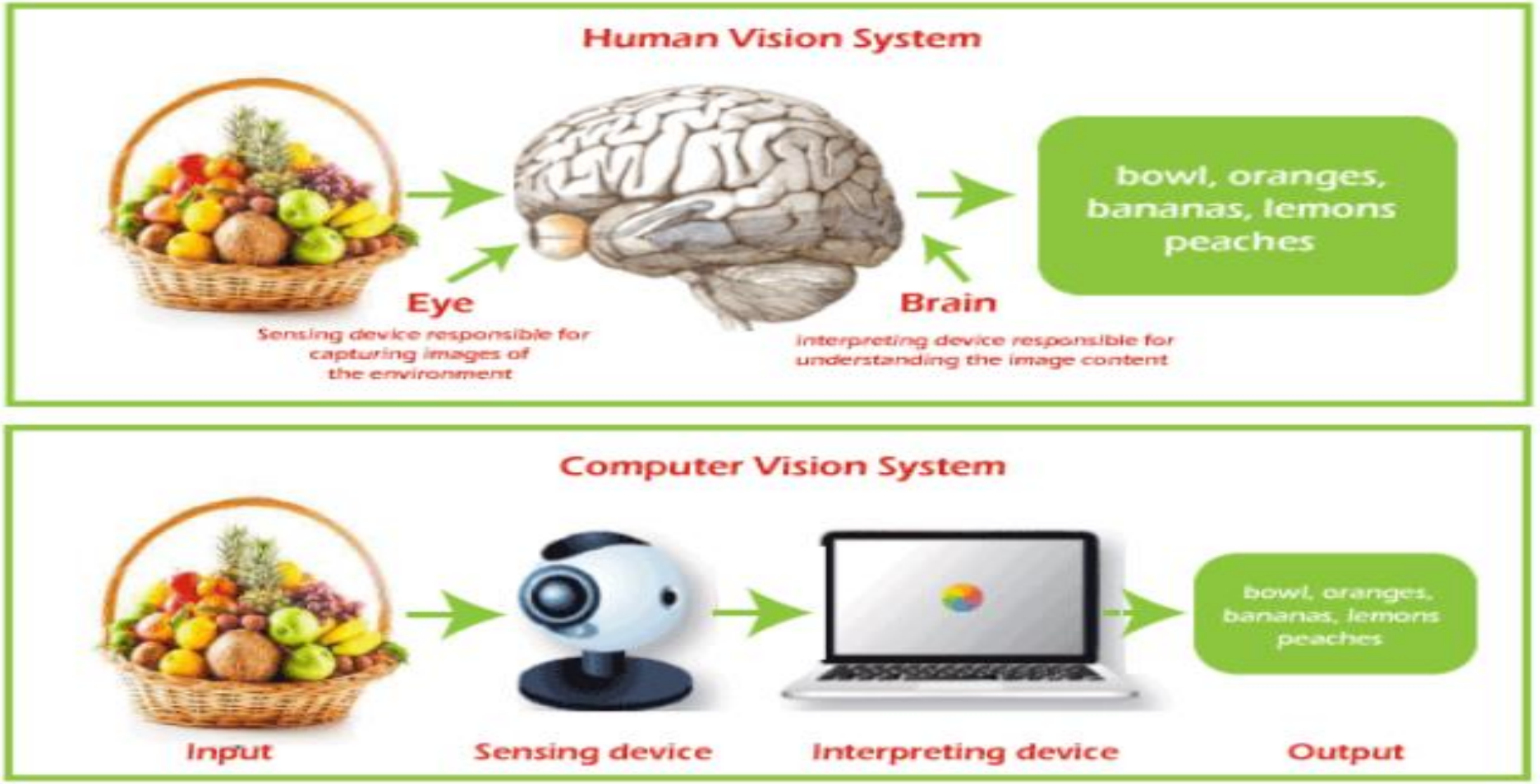
Human Vision Vs Computer Vision



Can computers match human perception?

- Yes and no (mainly no)
  - computers can be better at “easy” things
  - humans are better at “hard” things
- But huge progress
  - Accelerating in the last five years due to deep learning
  - What is considered “hard” keeps changing

How does computer vision works?

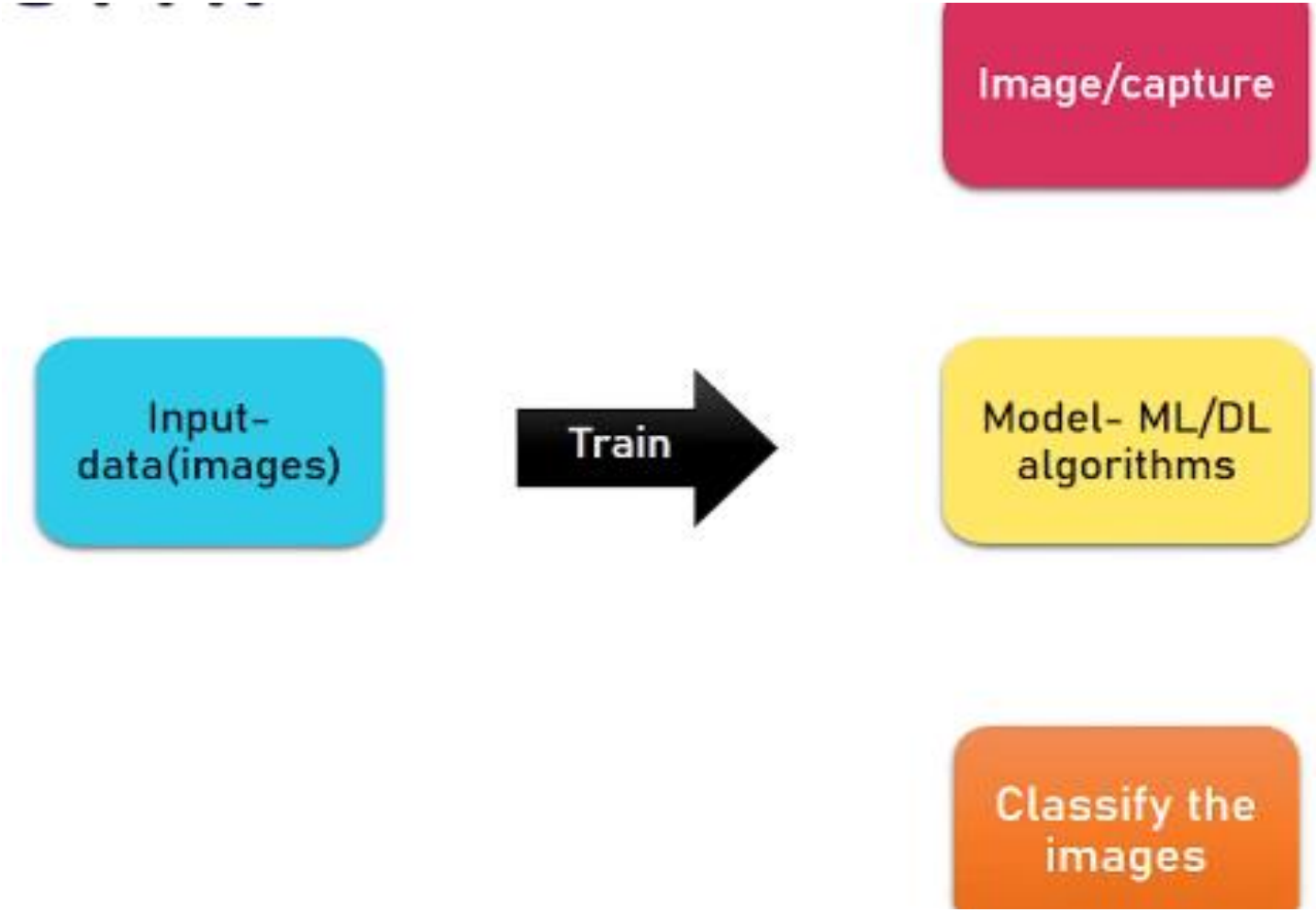


Computer vision working steps:

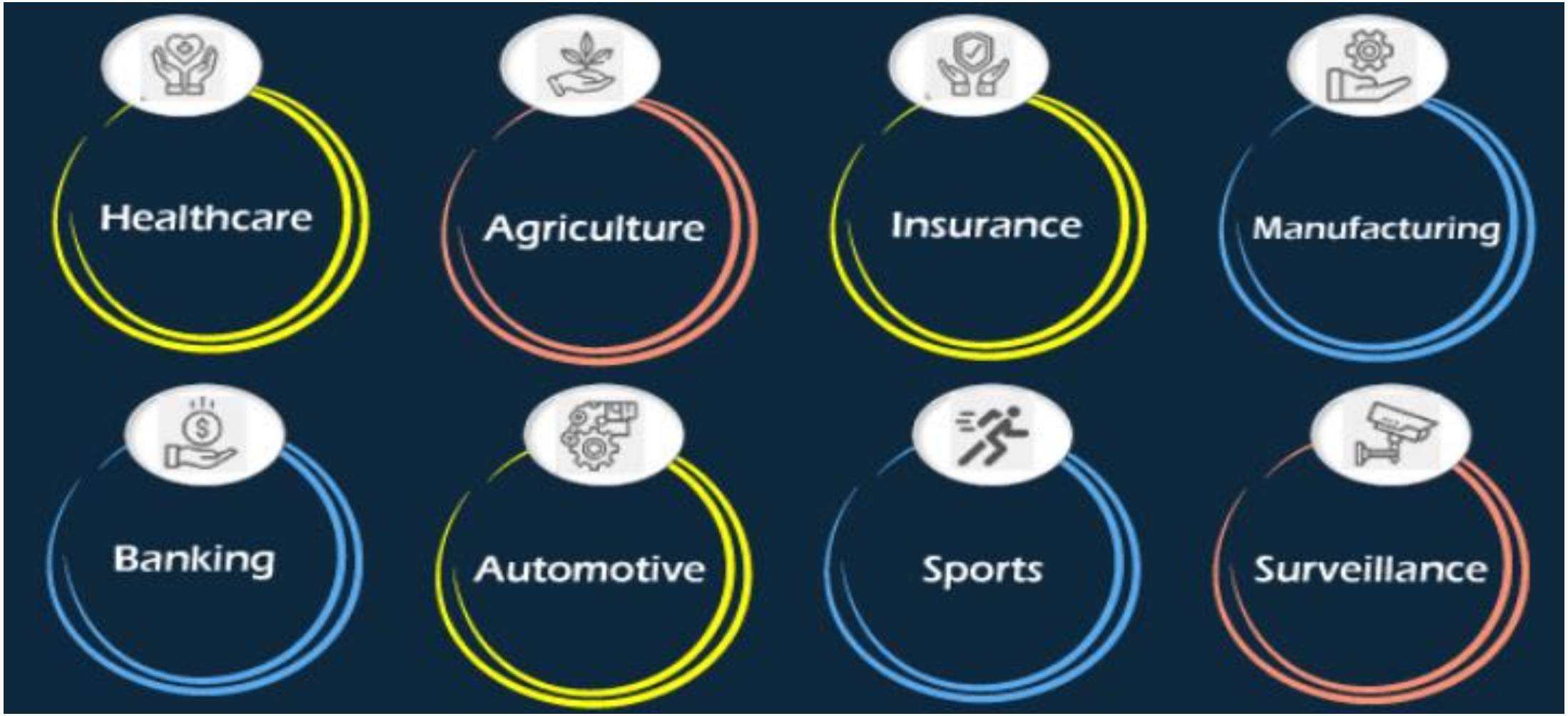
Computer Vision work much the same as, human computer vision have to trains machines to perform these function

It has to do train the machine with a help of camera and data, algorithms

Computer vision working process



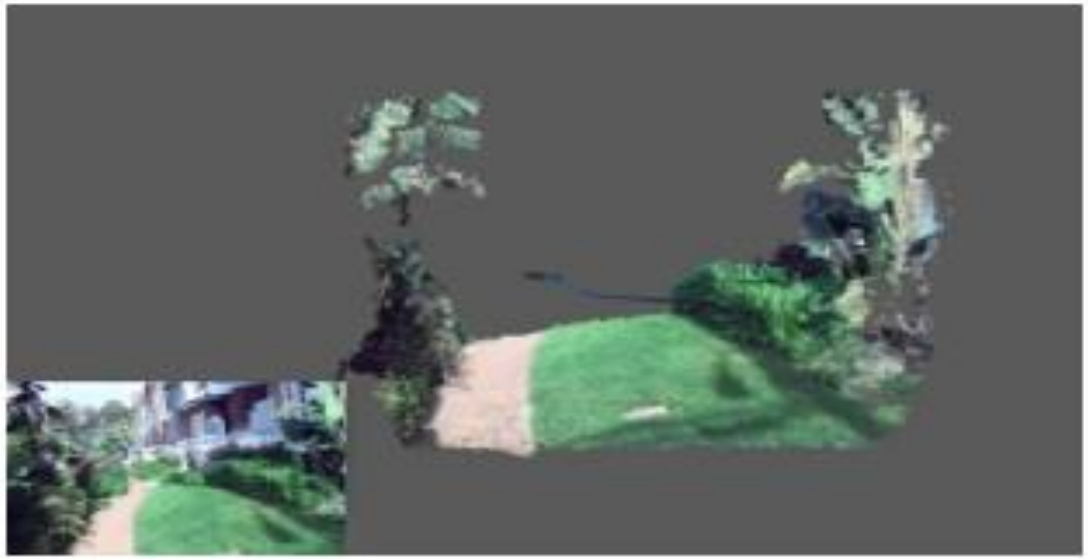
Applications of computer vision

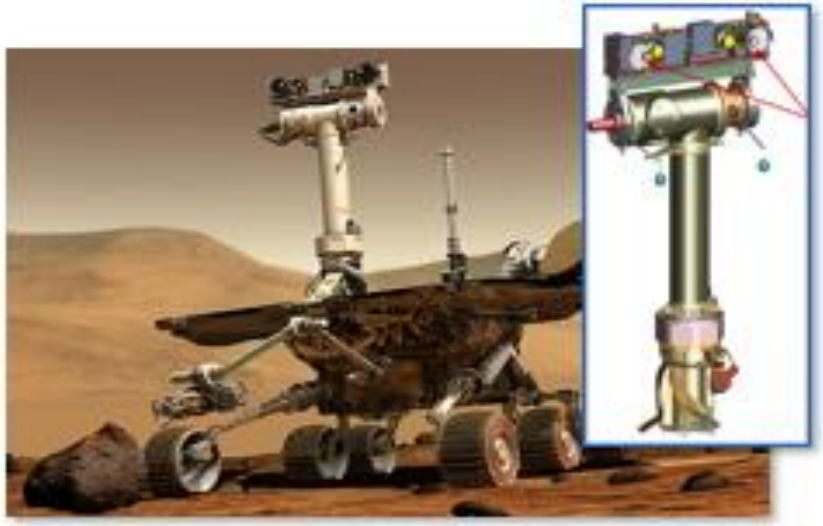


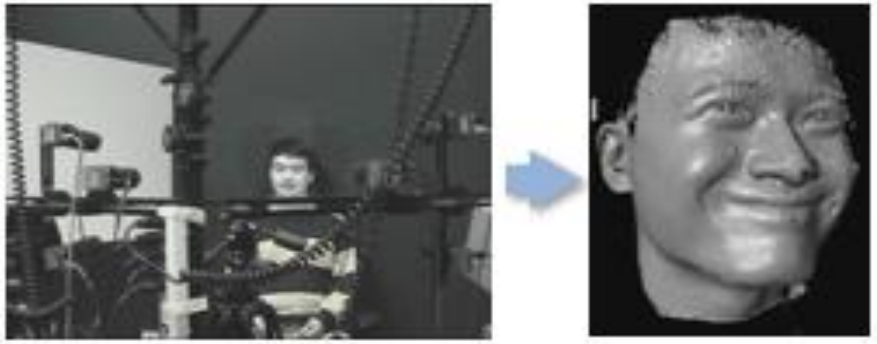
Goal of computer vision

- Compute the 3D shape of the world

ZED 2i Camera



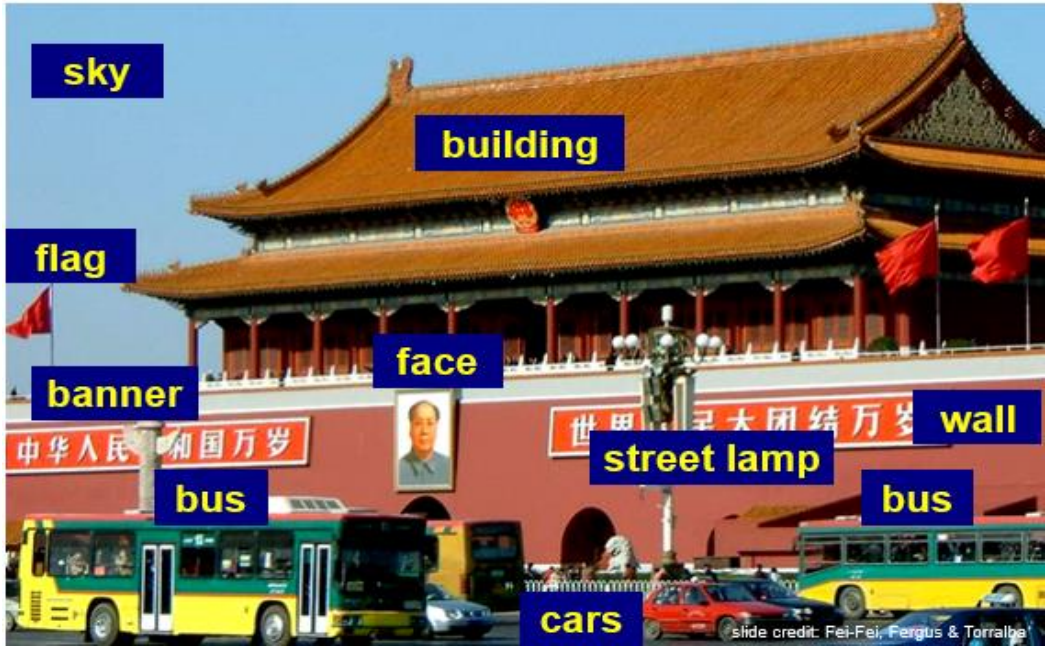




Goal of computer vision

- Recognize objects and people

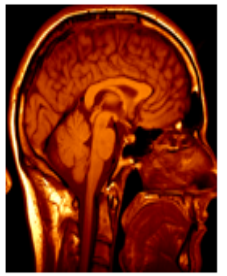
Terminator 2, 1991



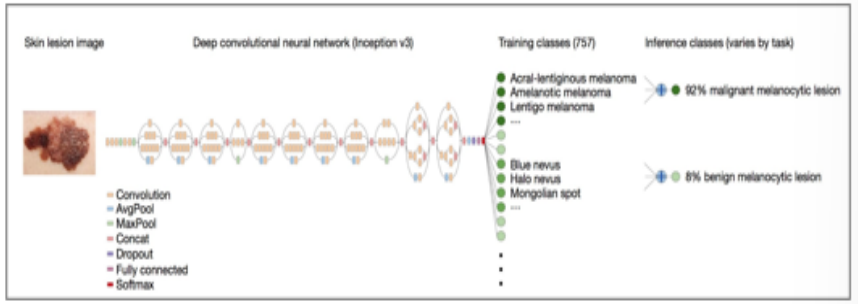
Goal of computer vision

- "Enhance" images

Medical imaging



3D imaging
(MRI, CT)



Skin cancer classification with deep learning
https://cs.stanford.edu/people/esteva/nature/

Goal of computer vision

- Forensics

Goal of computer vision

- Improve photos ("Computational Photography")

Super-resolution (source: 2d3)

Low-light photography
(credit: Hasinoff et al., SIGGRAPH ASIA 2016)

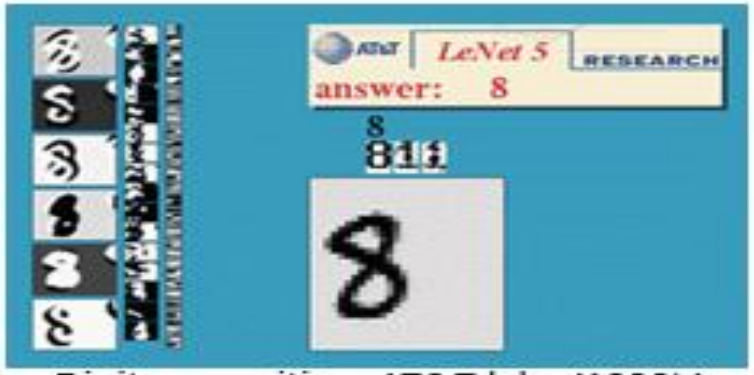
Depth of field on cell phone camera (source: Google Research Blog)

Removing objects
(Google Magic Eraser)

Goal of computer vision

Optical character recognition (OCR)

- If you have a scanner, it probably came with OCR software



Digit recognition, AT&T labs (1990's)
http://yann.lecun.com/exdb/lenet/

License plate readers
http://en.wikipedia.org/wiki/Automatic_number_plate_recognition

Automatic check processing

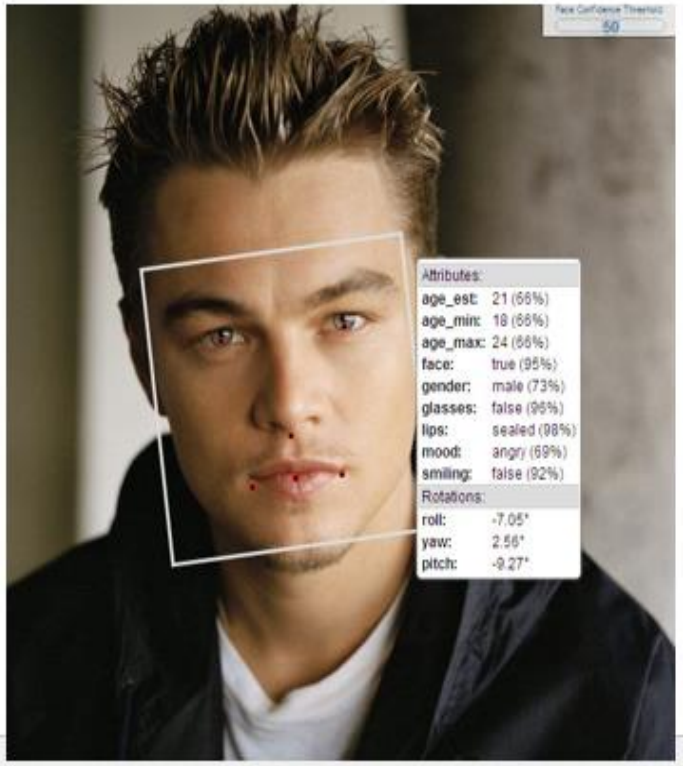
Sudoku grabber
http://sudokugrab.blogspot.com/

Goal of computer vision

Face detection

- Nearly all cameras detect faces in real time

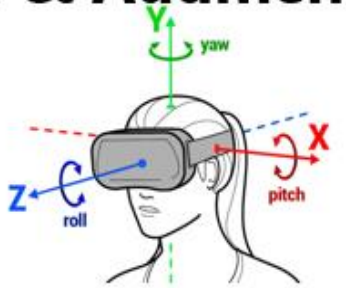
Face analysis and recognition



Goal of computer vision

Self-driving cars

Virtual & Augmented Reality



6DoF head tracking

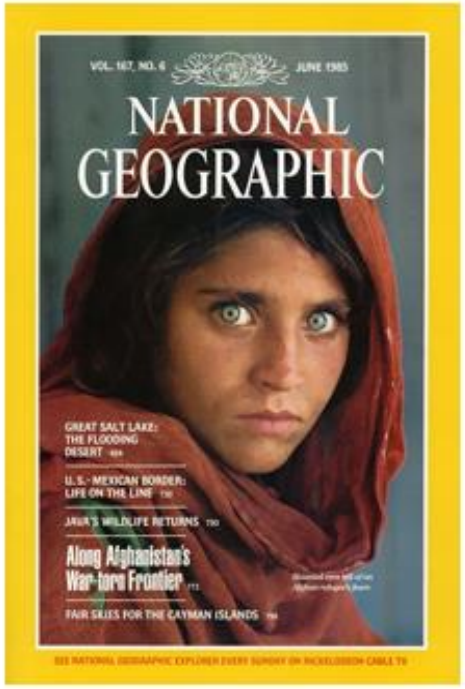
Hand & body tracking

3D scene understanding

3D-360 video capture

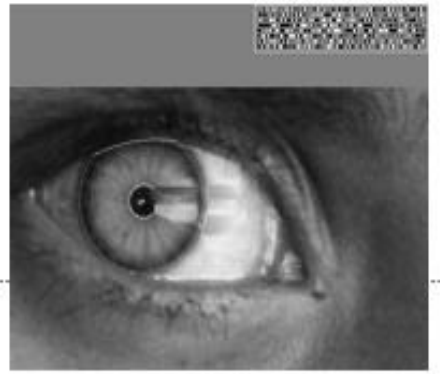
Goal of computer vision

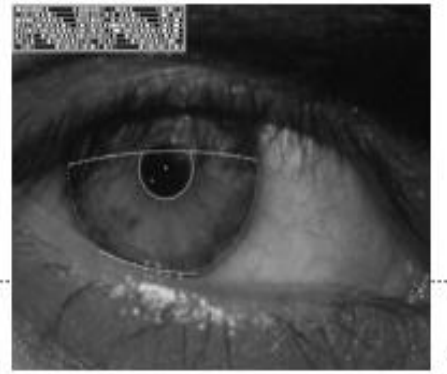
Vision-based biometrics



Who is she?

"How the Afghan Girl was Identified by Her Iris Patterns" Read the story





Source: S. Seitz

Goal of computer vision

Login without a password

Fingerprint scanners on many new smartphones and other devices

Face unlock on Apple iPhone X
See also http://www.sensiblevision.com/

Goal of computer vision

Robotics

NASA's Mars Curiosity Rover
https://en.wikipedia.org/wiki/Curiosity_(rover)

Amazon Picking Challenge
http://www.robocup2016.org/en/events/amazon-picking-challenge/

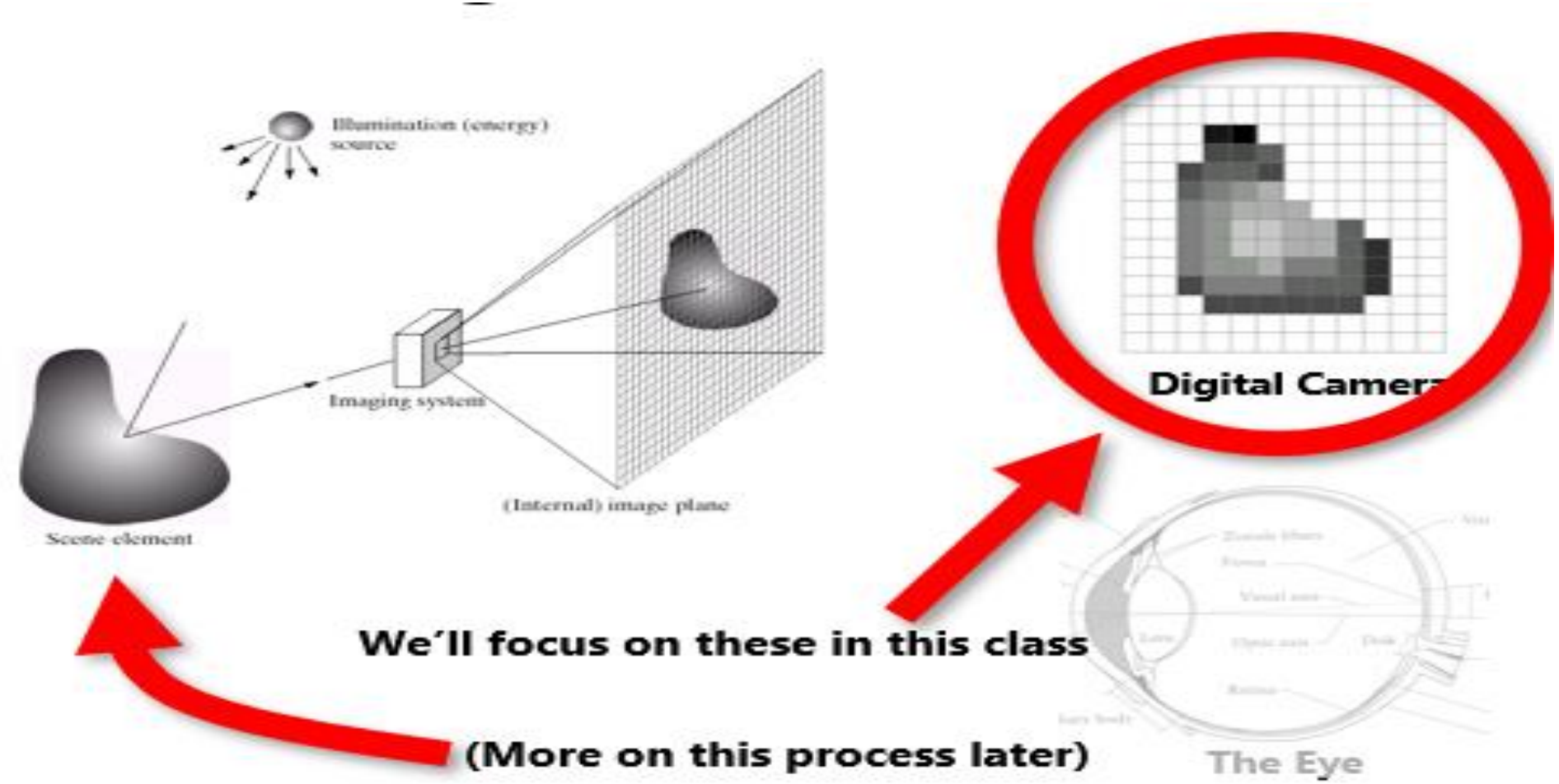
Amazon Prime Air

Amazon Scout

Current state of the art

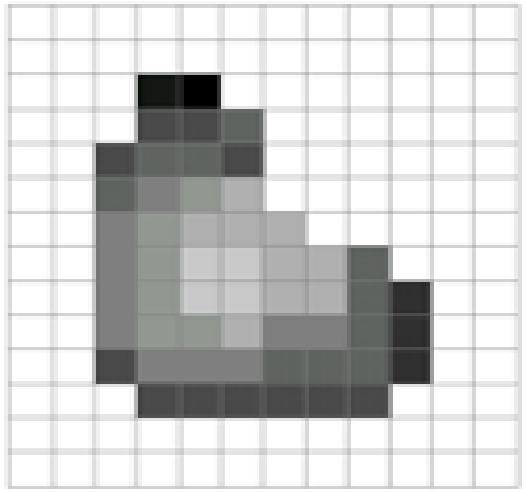
- You just saw many examples of current systems.
  - Many of these are less than 5 years old
- Computer vision is an active research area, and rapidly changing
  - Many new apps in the next 5 years
  - Deep learning powering many modern applications
- Many startups across a dizzying array of areas
  - Deep learning, robotics, autonomous vehicles, medical imaging, construction, inspection, VR/AR, …

What is an image?



What is an image?

- A grid (matrix) of intensity values



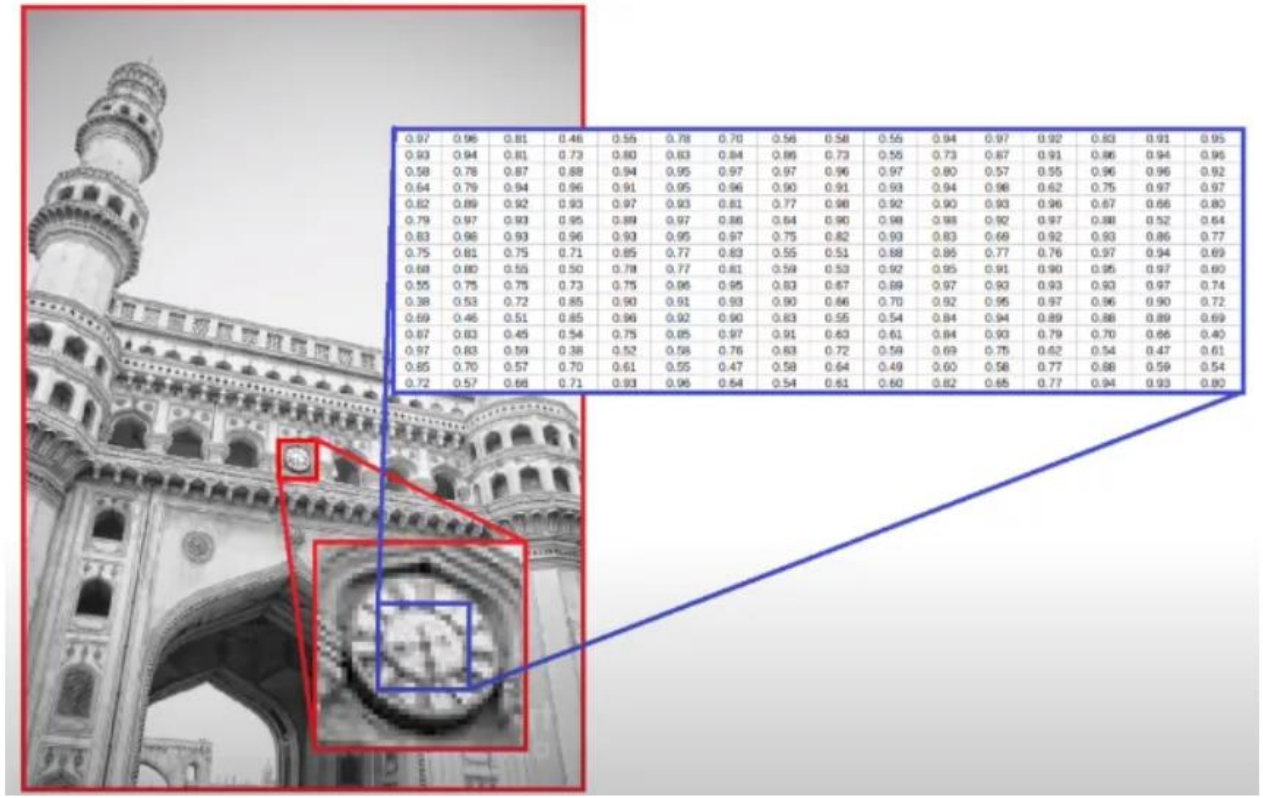
=

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

(common to use one byte per value: 0 = black, 255 = white)

What is an image?

- The simplest way to represent the image is in the form of a matrix.



we can see that a part of the image, i.e., the clock, has been represented as a matrix. A similar matrix will represent the rest of the image too.

What is an image?

- **An image is a visual representation of something, while a digital image is a binary representation of visual data.**

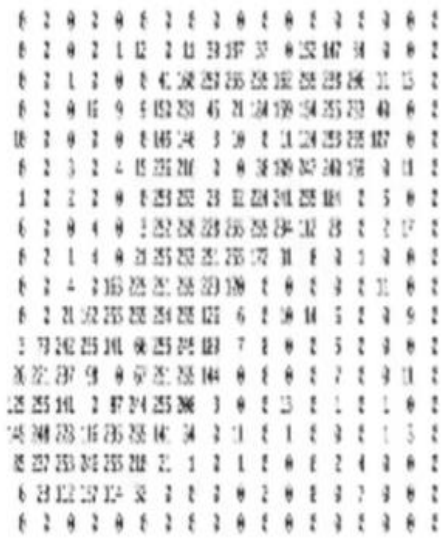
- It is commonly seen that people use up to a byte to represent every pixel of the image.

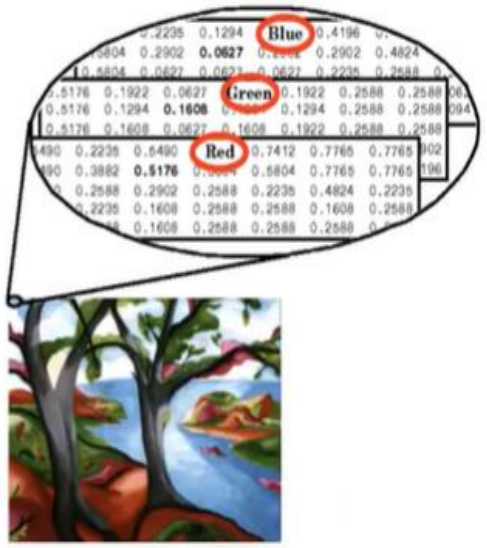
- This means that values between 0 to 255 represent the intensity for each pixel in the image where 0 is black and 255 is white.

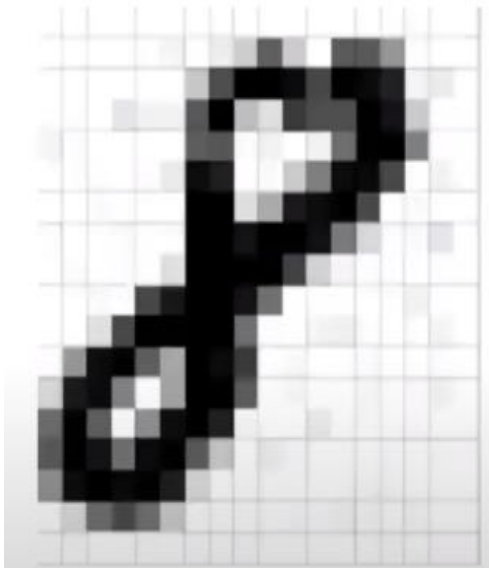
- For every color channel in the image, one such matrix is generated.

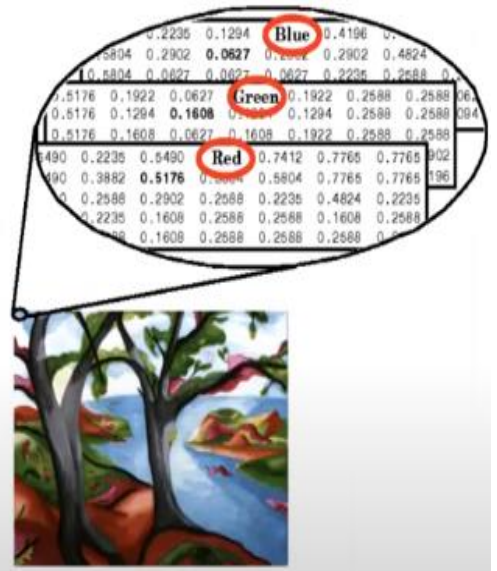
- In practice, it is also common to normalize the values between 0 and 1

What is an image?









Grayscale Images

- Pixels values range in gray levels from 0 (black) to 255 (white)

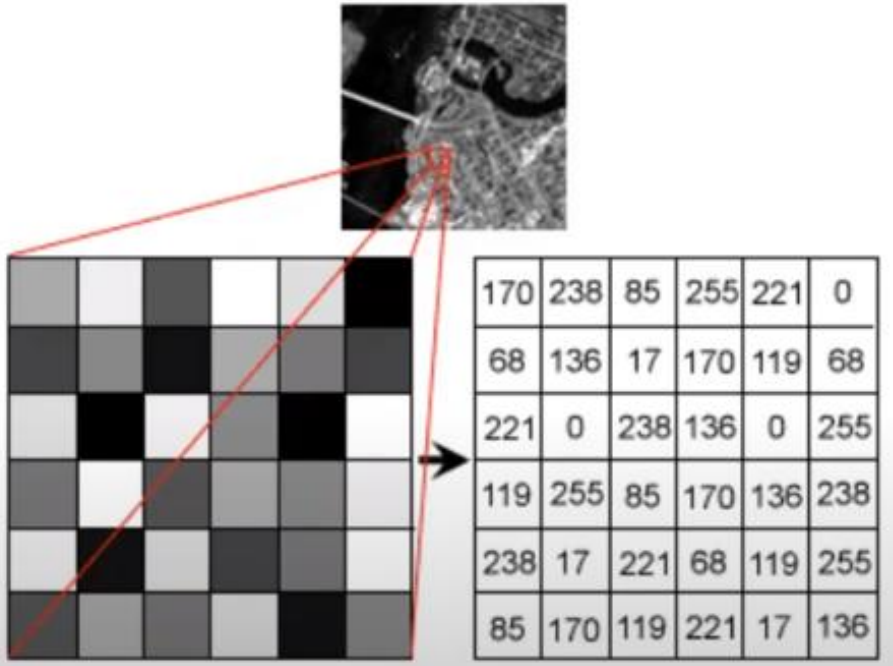
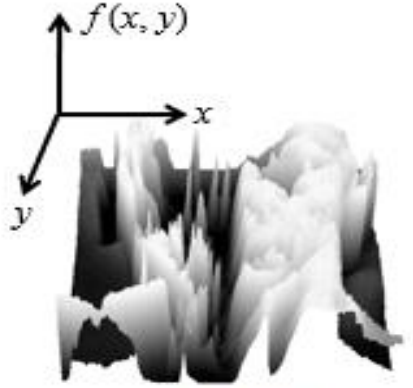


Image as a function

- An image can also be represented as a function. An image (grayscale) can be thought of as a function that takes in a pixel coordinate and gives the intensity at that pixel.
- It can be written as function $f: \mathbb{R}^2 \rightarrow \mathbb{R}$ that outputs the intensity at any input point (x,y). The value of intensity can be between 0 to 255 or 0 to 1 if values are normalized.

snoop

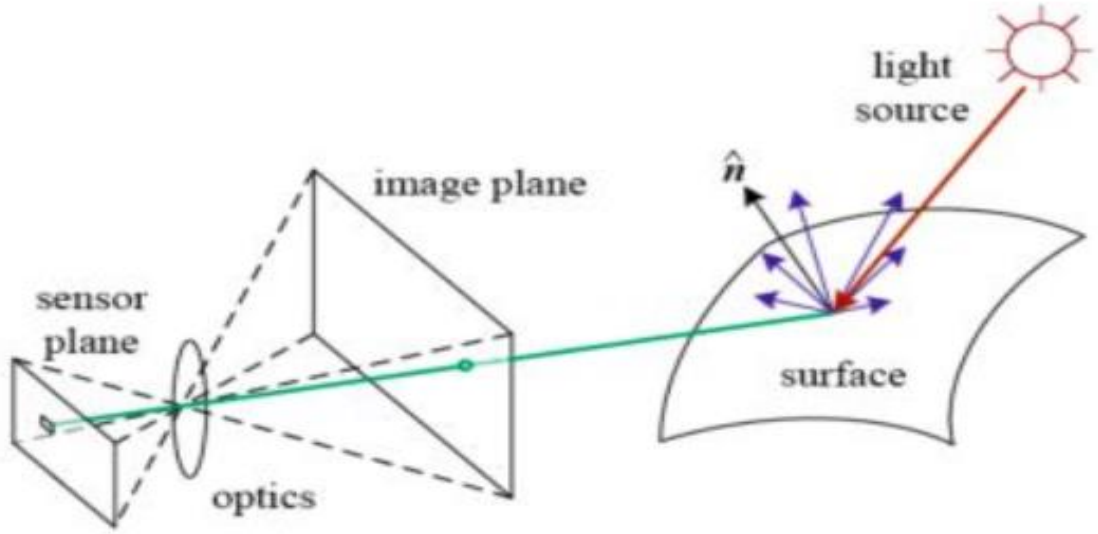


3D view

– A **digital** image is a discrete (**sampled**, **quantized**) version of this function

Image Formation

- An image will always depend on :
  - lighting conditions
  - scene geometry
  - surface properties
  - camera optics



- Using the imaging system, the photons that arrive at each cell are integrated and the digitized. An image appears as a grid of intensity values, corresponding to the value of each pixel.

0 = black, 255=white

An image can be compared to a function $f : R^2 \rightarrow R$ giving an intensity at each point (x, y).

Image Filtering

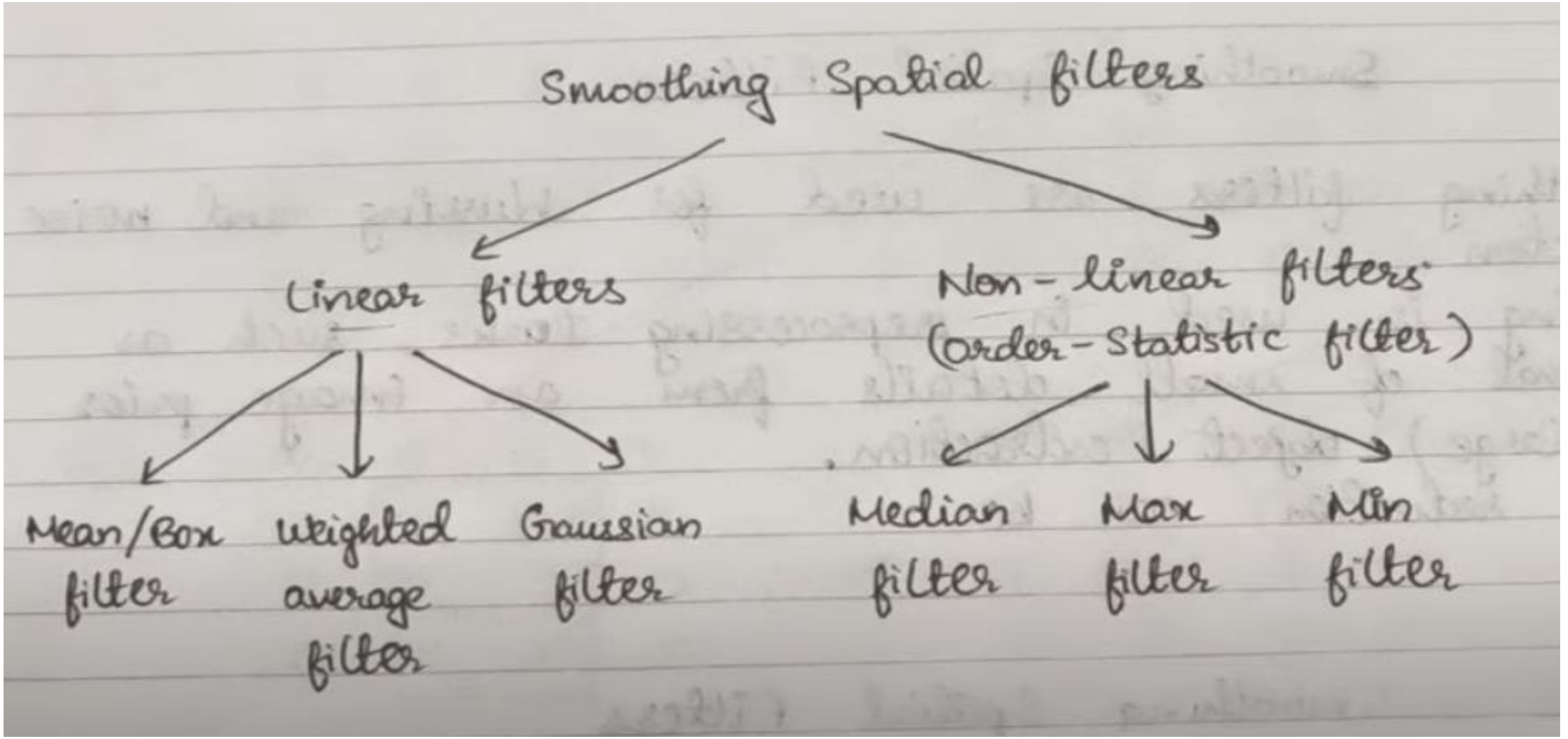


Image Filtering

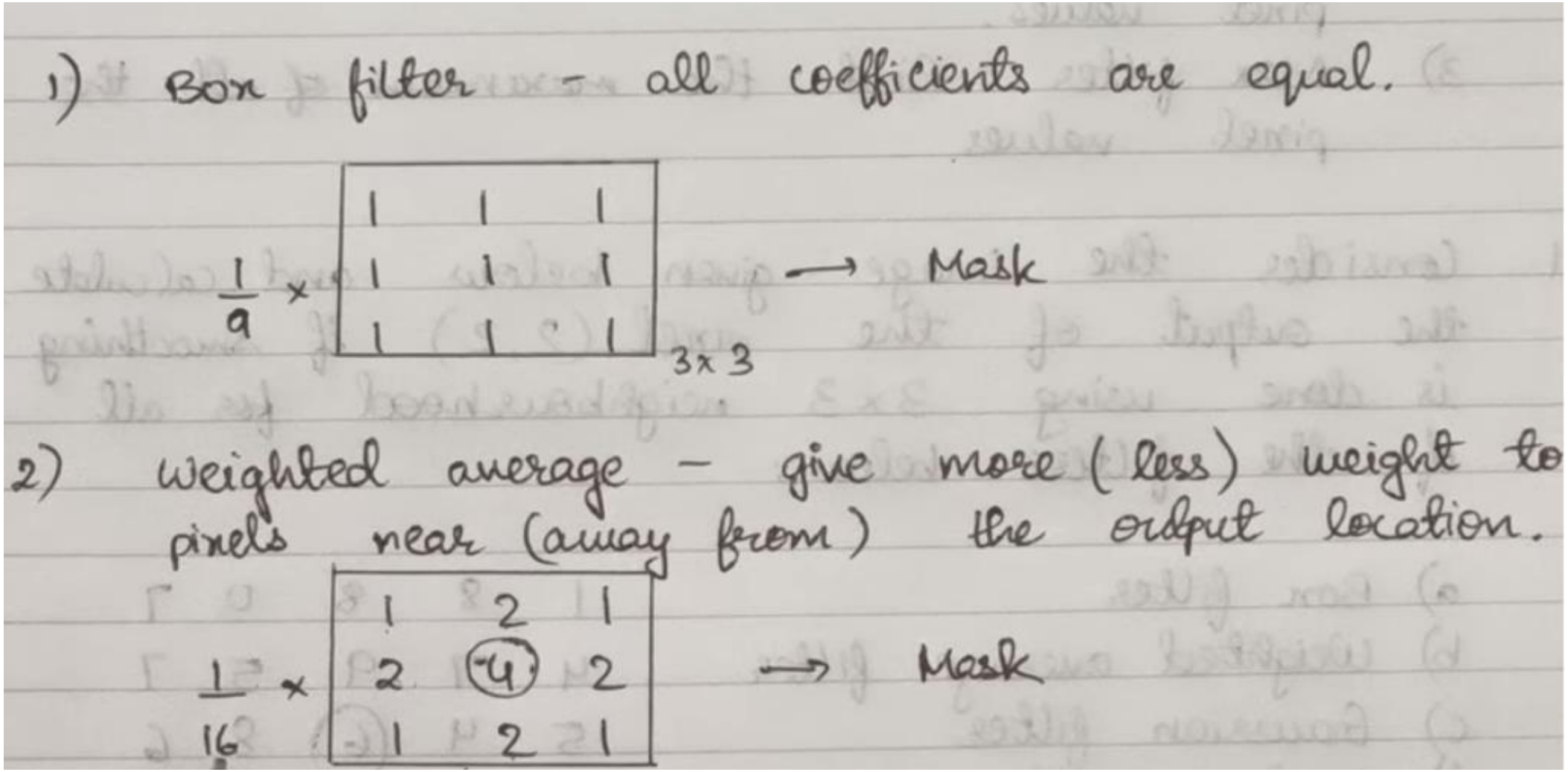


Image Filtering

3) Gaussian filter - the weights are samples of 2D Gaussian function:

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$ (2D Gaussian function)

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

→ Used to blur edges and reduce contrast.
→ Similar to median filter but is faster.

Image Filtering

- A filter can be seen as any kind of operator that can be applied to an image.

- Filtering is often used for :
  - image enhancement (denoise, resize…)
  - extract information (texture, edges)
  - detect patterns (template matching)

1. **Noise Reduction –**

   (a) Averaging, (b) Linear Filtering, (c) Gaussian Filtering

2. **Convolution Linear filters –**

   (a) Convolution Vs. Corelation, (b) Examples of Convolution

Image Filtering (Why?)

- To reduce noise.
- To fill into the missing information in the image.
- To find important features like edges – detect edges.
- To sharpen contrast.

- Image filters can be classified as linear or nonlinear.
- Linear filters are also know as convolution filters as they can be represented using a matrix multiplication.
- Thresholding and image equalization are examples of nonlinear operations, as is the median filter.

1. Noise Reduction

- A first way to reduce noise is to average out several noised images, but it typically requires a lot of images, which we generally do not have. We need to use smoothing filters to overcome this issue.
- Smoothing filters have several properties :
  - all values are positive
  - they all sum to 1
  - the amount of smoothing is proportional to the mask size
  - it removes high-frequency components

1. Noise Reduction – (a) Averaging/Mean Filtering

H

$\frac{1}{9} \times$

1	1	1
1	1	1
1	1	1

$*$

F

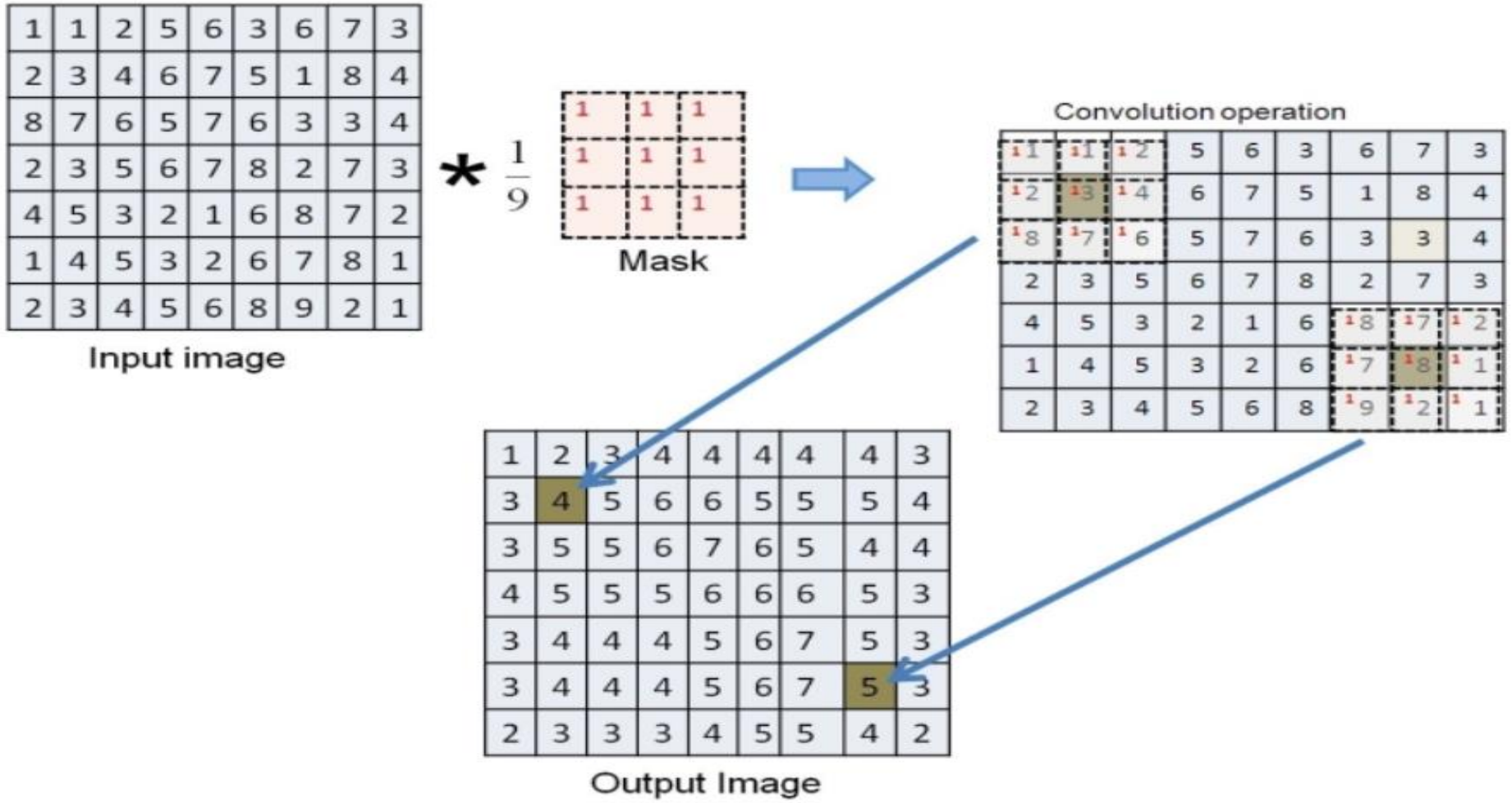
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$=$

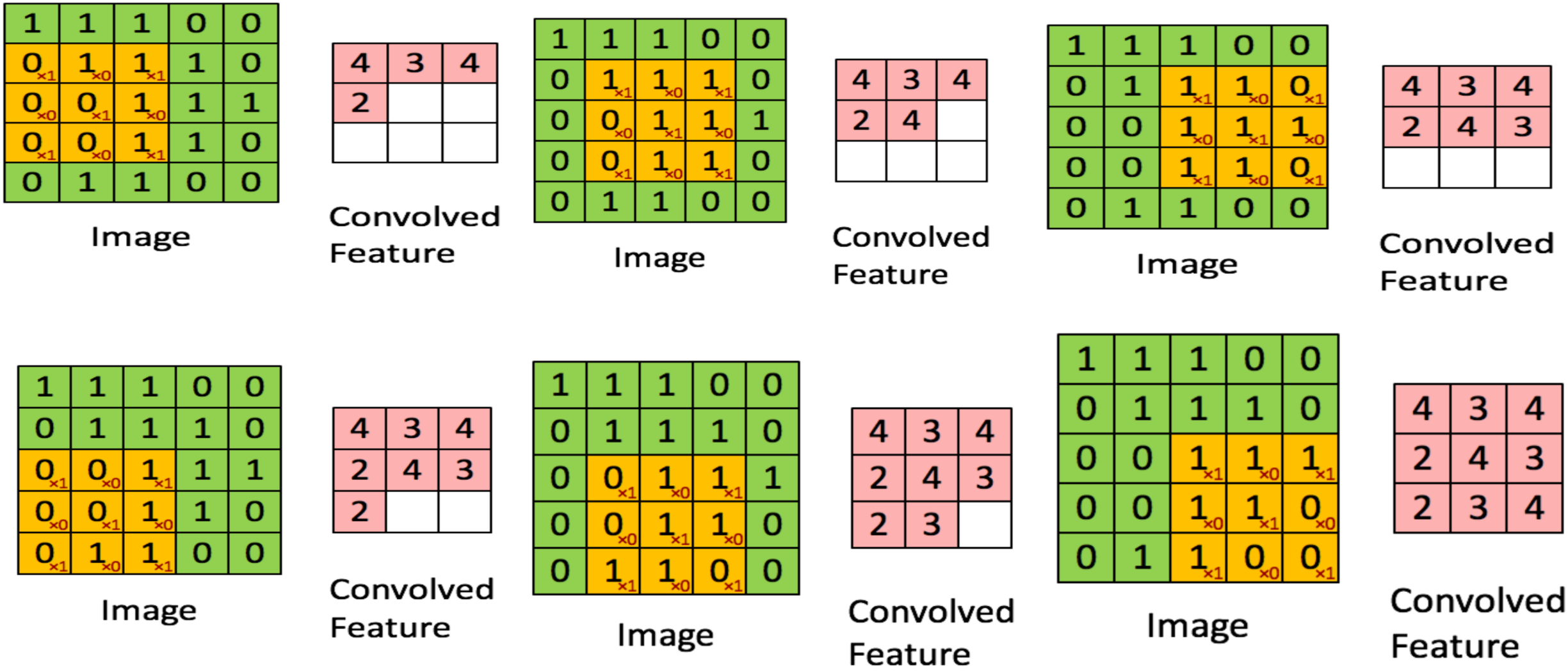
G

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

1. Noise Reduction – (a) Averaging/Mean Filtering



1. Noise Reduction – (a) Averaging



1. Noise Reduction – (a) Averaging/Mean Filtering

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10							

1. Noise Reduction – (a) Averaging/Mean Filtering

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20						

1. Noise Reduction – (a) Averaging/Mean Filtering

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20	30					

1. Noise Reduction – (a) Averaging/Mean Filtering

$F[x, y]$

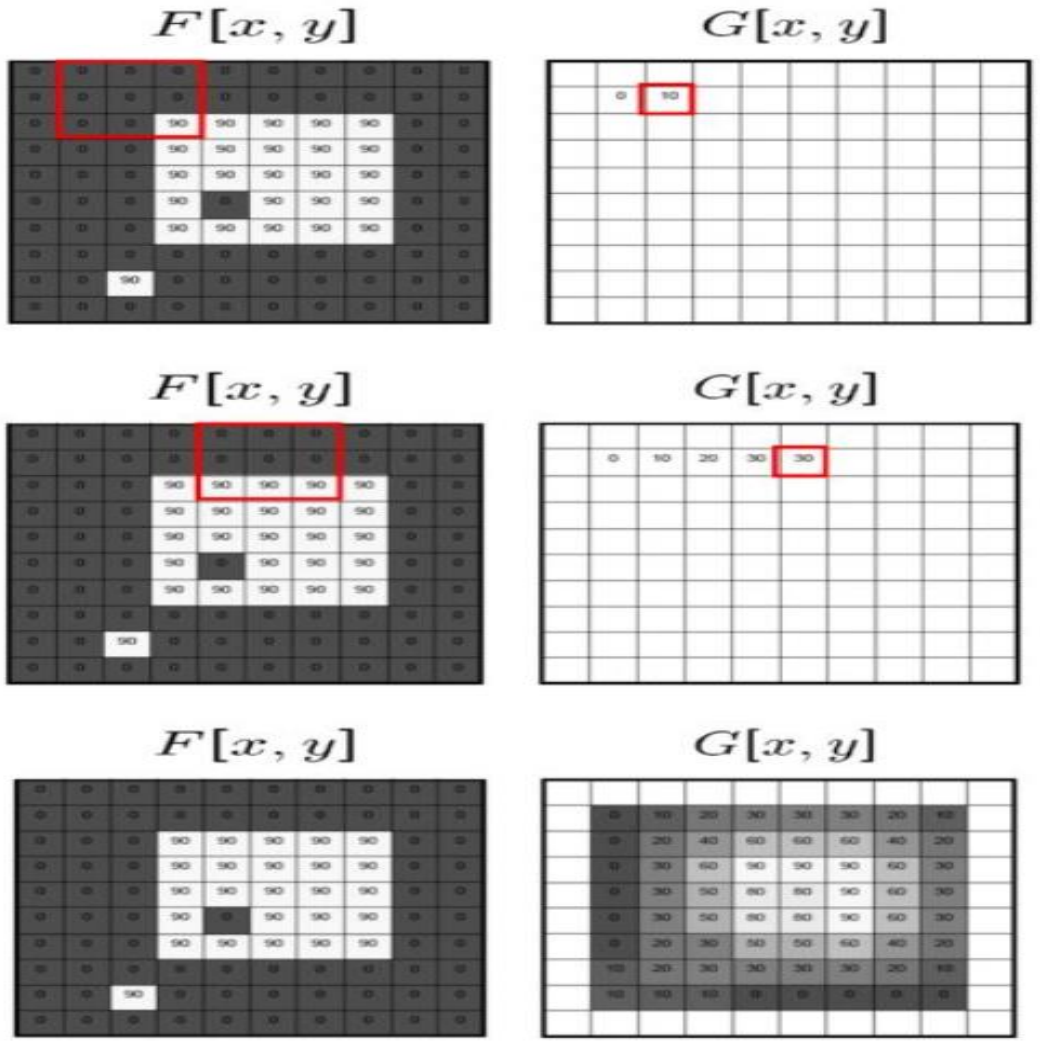
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20	30	30				

1. Noise Reduction – (a) Averaging/Mean Filtering

- On an image, the process leads to the following averaged image :



1. Noise Reduction – (a) Averaging/Mean Filtering

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

1. Noise Reduction – (a) Averaging

- The Averaging Filter technique takes the average of all the pixels under the kernel area and replaces the central element.
- The functions **cv2.blur(), cv2.filter2D()** and **cv2.boxFilter()** can be used to perform the averaging filter. All functions smooth an image using the kernel.

1. Noise Reduction – (b) Linear Filtering

*Intuition *: This involved a weighted combination of pixels in the small neighborhood of the pixel we want to estimate.

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

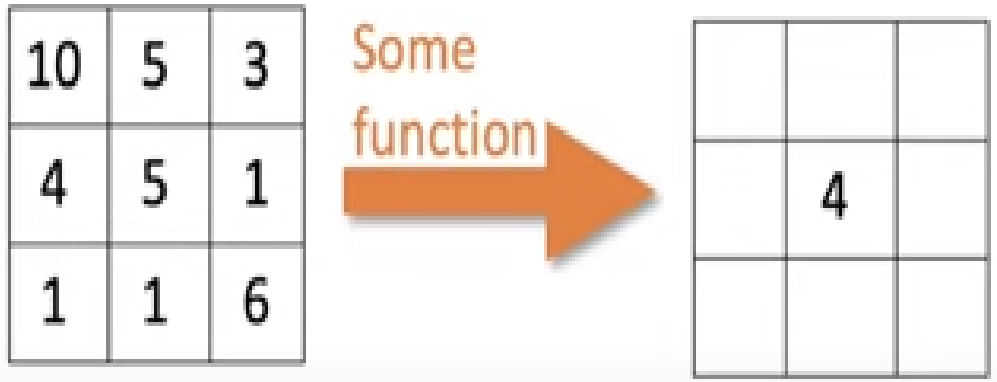
We call $h(k, I)$ a kernel, or mask, and the problem can be rewritten using the correlation/convolution operator : $G = H \otimes F$.

original

filtered

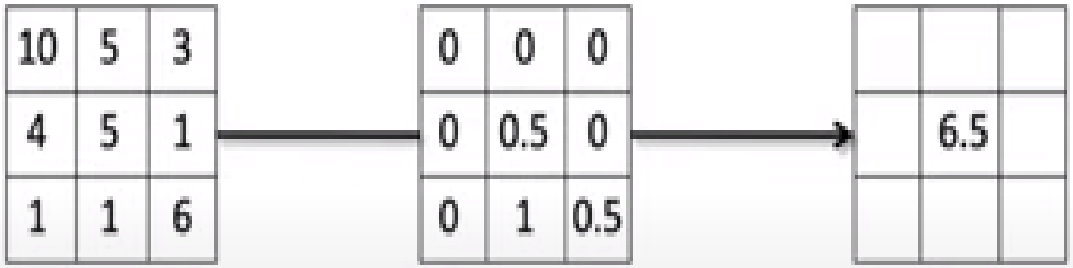
1. Noise Reduction – (b) Linear Filtering

- **Image Filter:** Modify image pixels based on some function of a local neighbourhood of each pixel



What's the function?

- **Linear Filter:** Replace each pixel by linear combination (a weighted sum) of neighbours
- Linear combination called kernel, mask or filter



1. Noise Reduction – (c) Gaussian Filtering

Gaussian filtering is used to blur images and remove noise and detail.

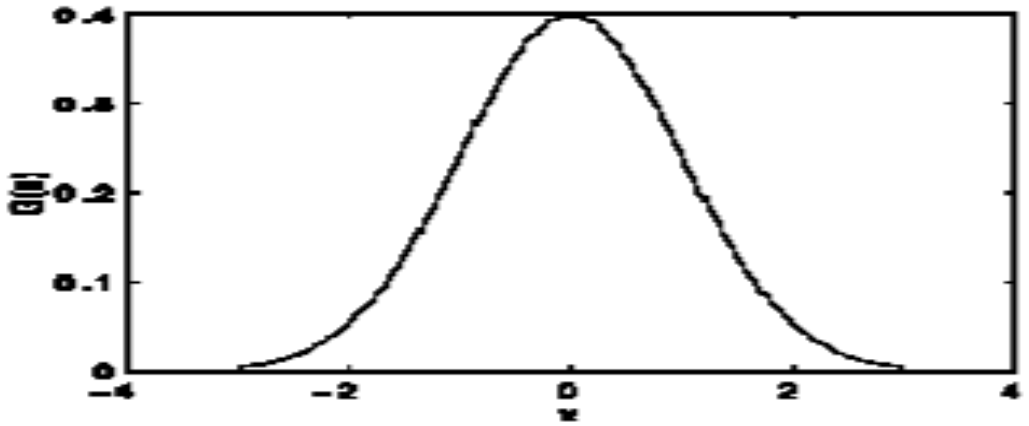
In one dimension, the Gaussian function is:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Where σ is the standard deviation of the distribution. The distribution is assumed to have a mean of 0.

Shown graphically, we see the familiar bell shaped Gaussian distribution.

Gaussian distribution with mean 0 and σ = 1



1. Noise Reduction – (c) Gaussian Filtering

Gaussian kernel coefficients are sampled from the 2D Gaussian function.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Where σ is the standard deviation of the distribution.

The distribution is assumed to have a mean of zero.

We need to discretize the continuous Gaussian functions to store it as discrete pixels.

An integer valued 5 by 5 convolution kernel approximating a Gaussian with a σ of 1 is shown to the right,

$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

1. Noise Reduction – (c) Gaussian Filtering

The Gaussian filter is a non-uniform low pass filter.

The kernel coefficients diminish with increasing distance from the kernel's centre.

Central pixels have a higher weighting than those on the periphery.

Larger values of σ produce a wider peak (greater blurring).

Kernel size must increase with increasing σ to maintain the Gaussian nature of the filter.

Gaussian kernel coefficients depend on the value of σ.

At the edge of the mask, coefficients must be close to 0.

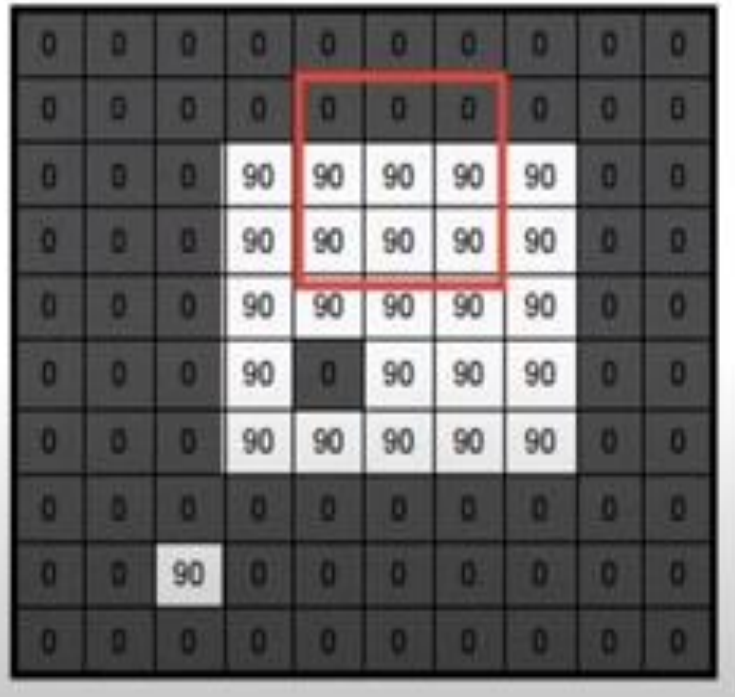
The kernel is rotationally symmetric with no directional bias.

Gaussian kernel is separable, which allows fast computation.

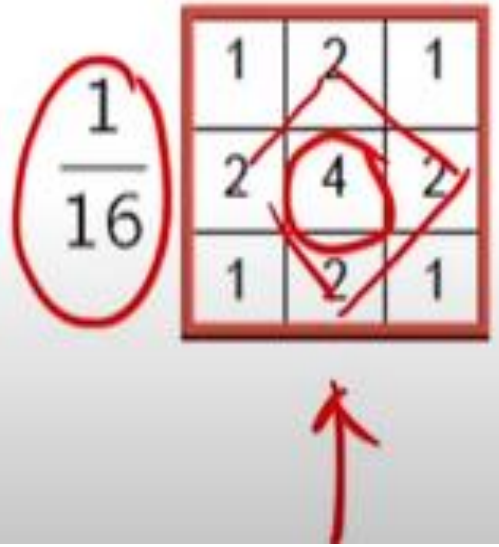
Gaussian filters might not preserve image brightness.

1. Noise Reduction – (c) Gaussian Filtering

$I(i,j)$

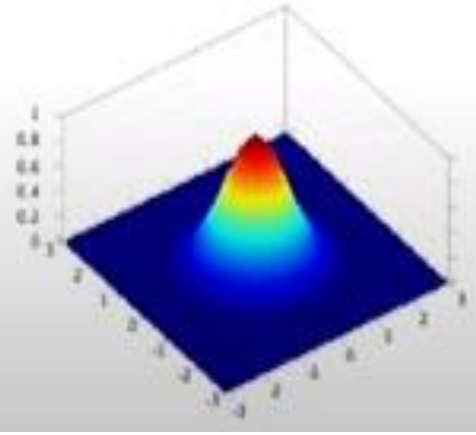


$\otimes \quad H(u,v)$



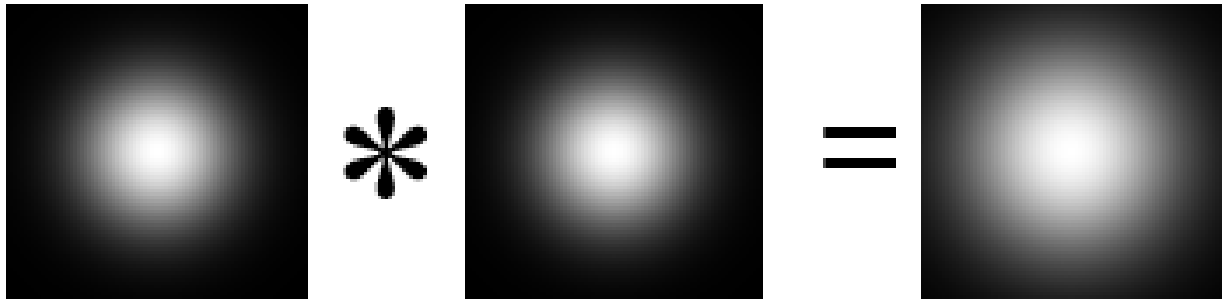
This kernel is an approximation of a 2D Gaussian function:

$$h(u,v) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{u^2+v^2}{\sigma^2}}$$



1. Noise Reduction – (c) Gaussian Filtering

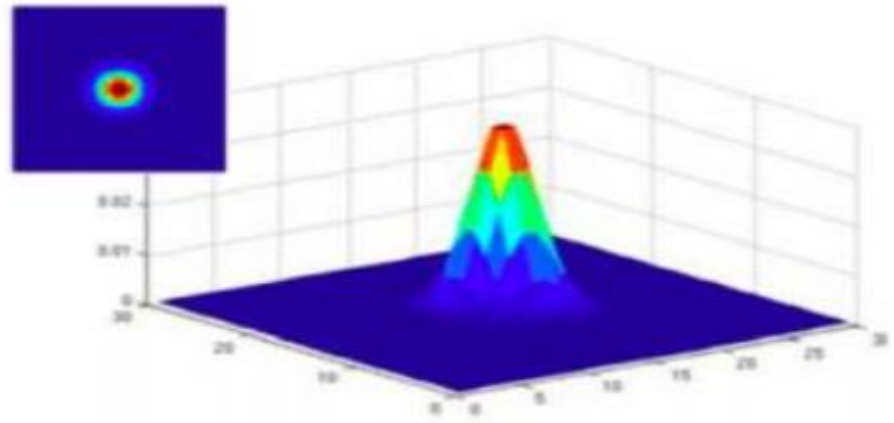
- Removes "high-frequency" components from the image (low-pass filter)
- Convolution with self is another Gaussian



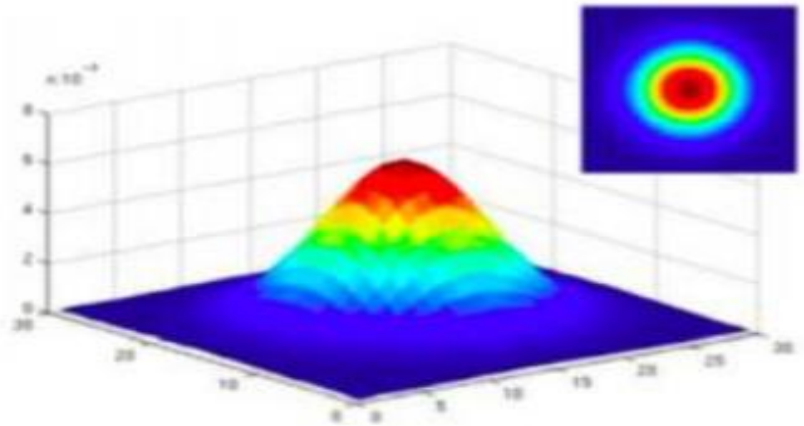
– Convolving twice with Gaussian kernel of width σ
= convolving once with kernel of width $\sigma\sqrt{2}$

1. Noise Reduction – (c) Gaussian Filtering

We can choose the size of the kernel or mask, and the variance, which determines the extent of smoothing.



$\sigma = 2$ with 30 x 30 kernel



$\sigma = 5$ with 30 x 30 kernel

Problems:

- Consider the image below and calculate the output of the pixel (3 * 3), if smoothing is done using 3 * 3 neighbourhood using all the filters below:
- (a) Box filter
- (b) Weighted Average Filter
- (c) Median Filter
- (d) Min filter
- (e) Max filter

1	8	8	0	7
4	7	9	5	7
5	4	6	8	6
4	2	0	1	5
0	1	0	2	3

Input image

Mask to find box filter

1/9	1	1	1
	1	1	1
	1	1	1

Mask to find weighted average filter

1/16	1	2	1
	2	4	2
	1	2	1

2. Convolution Filters– (a) Convolution Vs. Correlation

- **Correlation / Convolution**
  - **Correlation/convolution** is the **process** of moving a filter mask over the **image** and computing the sum of products at each location.
  - We use Correlation to check similarity between two images

- **difference between convolution** and **correlation** is that the **convolution process** rotates the matrix by 180 degrees.

2. Convolution Filters– (a) Convolution Vs. Correlation

Cross-correlation

Let F be the image, H be the kernel (of size $2k+1 \times 2k+1$), and G be the output image

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

This is called a **cross-correlation** operation:

$$G = H \otimes F$$

- Can think of as a "dot product" between local neighborhood and kernel for each pixel

2. Convolution Filters– (a) Convolution Vs. Correlation

Convolution

- Same as cross-correlation, except that the kernel is "flipped" (horizontally and vertically)

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

This is called a **convolution** operation:

$$G = H * F$$

- Convolution is **commutative** and **associative**

2. Convolution Filters– (a) Convolution Vs. Correlation

Convolution and Correlation are slightly different operations :

1. Correlation :

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

$$G = H \otimes F$$

1. Convolution :

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l)$$

$$G = H * F$$

2. Convolution Linear Filters– (b) Examples of Convolutions

a. Do nothing

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Filtered
(no change)

b. Shift the image

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

c. Sharpen an image

Original

$$* \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) =$$

Sharpening filter
(accentuates edges)

d. Find only the details of an image

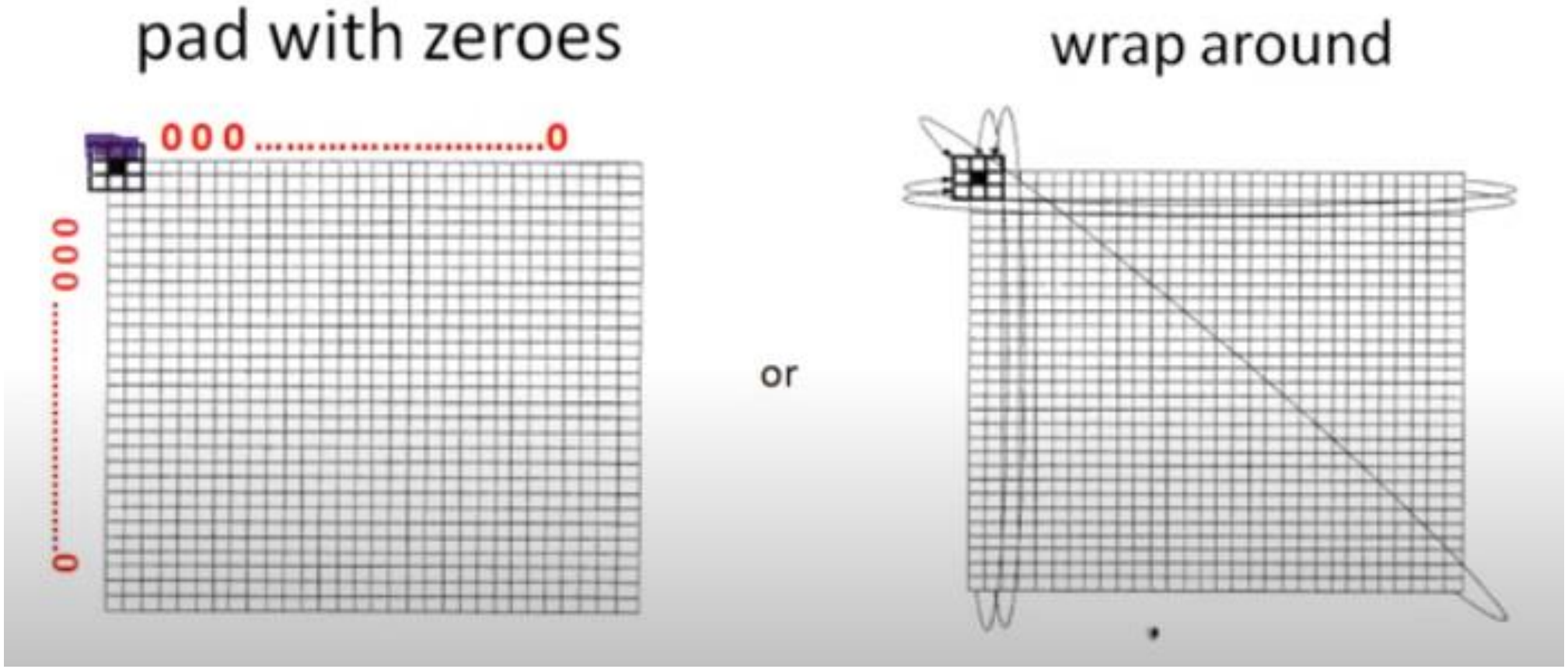
$-$

$=$

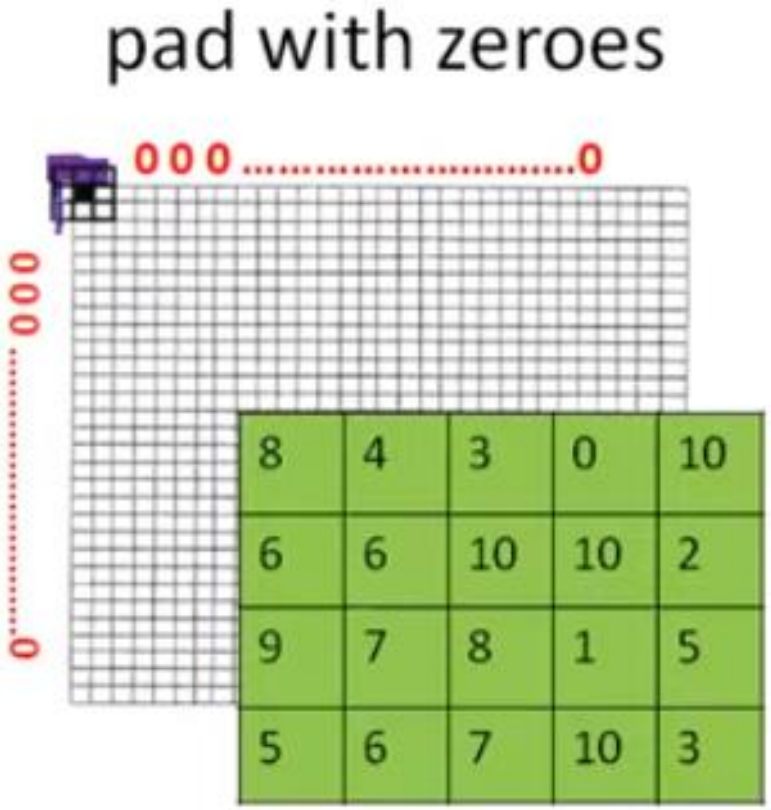
Filters - Thresholding

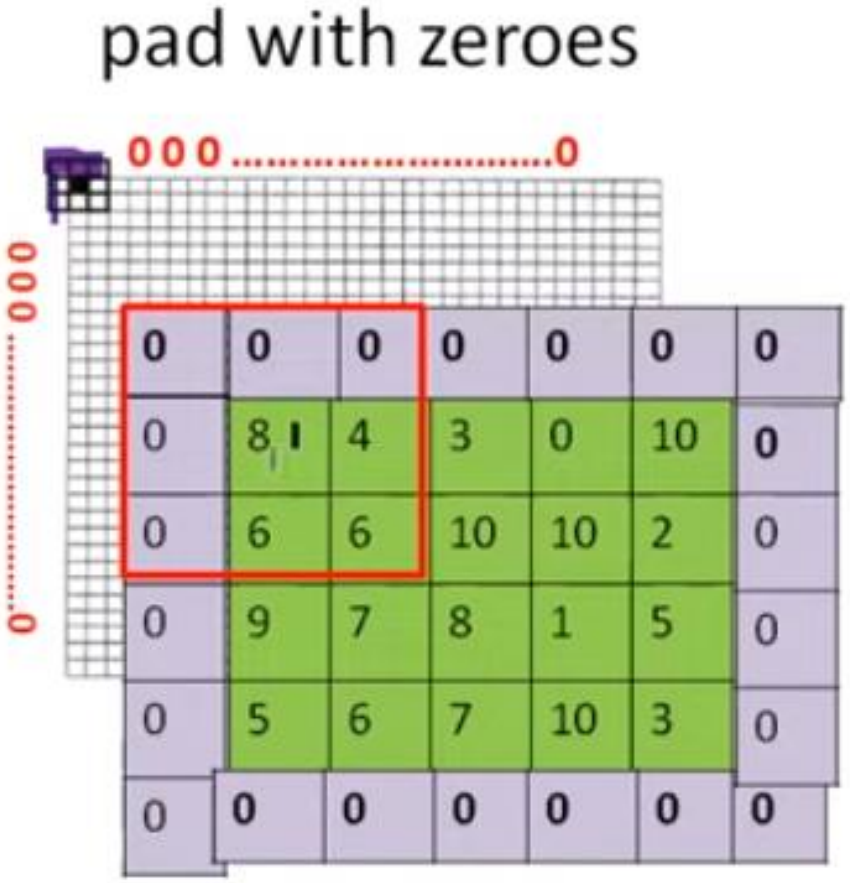
$$g(m,n) = \begin{cases} 255, & f(m,n) > A \\ 0 & otherwise \end{cases}$$

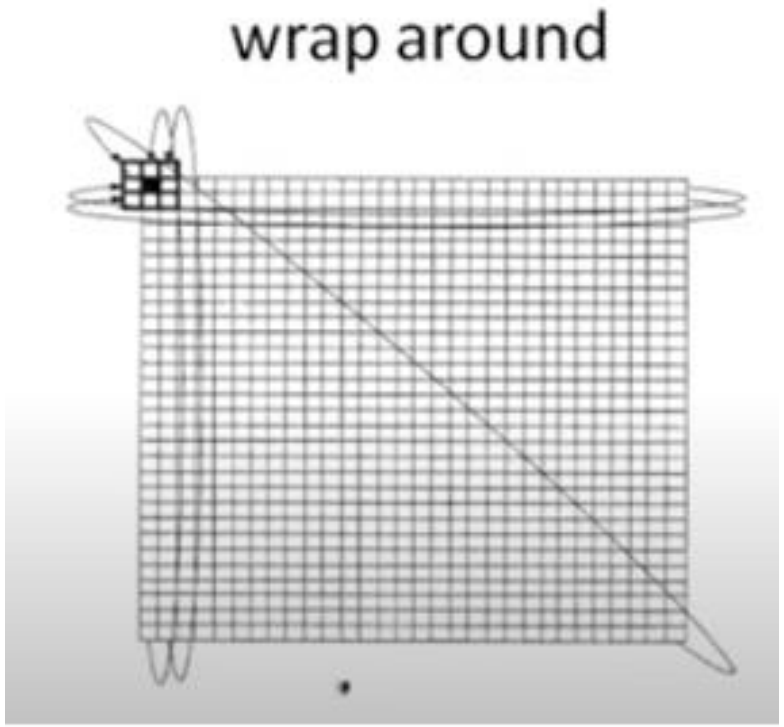
Handling Pixel close to boundaries



Padding the image with zeros







wrap around

5	6	7	10	3	5
8	4	3	0	10	
6	6	10	10	2	
9	7	8	1	5	
5	6	7	10	3	

wrap around

	5	6	7	10	3	5
10	8	4	3	0	10	
2	6	6	10	10	2	
5	9	7	8	1	5	
3	5	6	7	10	3	
10						

wrap around

3	5	6	7	10	3	5
10	8	4	3	0	10	
2	6	6	10	10	2	
5	9	7	8	1	5	
3	5	6	7	10	3	
10	8	4	3	0	10	8

wrap around

3	5	6	7	10	3	5
10	8	4	3	0	10	8
2	6	6	10	10	2	6
5	9	7	8	1	5	9
3	5	6	7	10	3	5
10	8	4	3	0	10	8

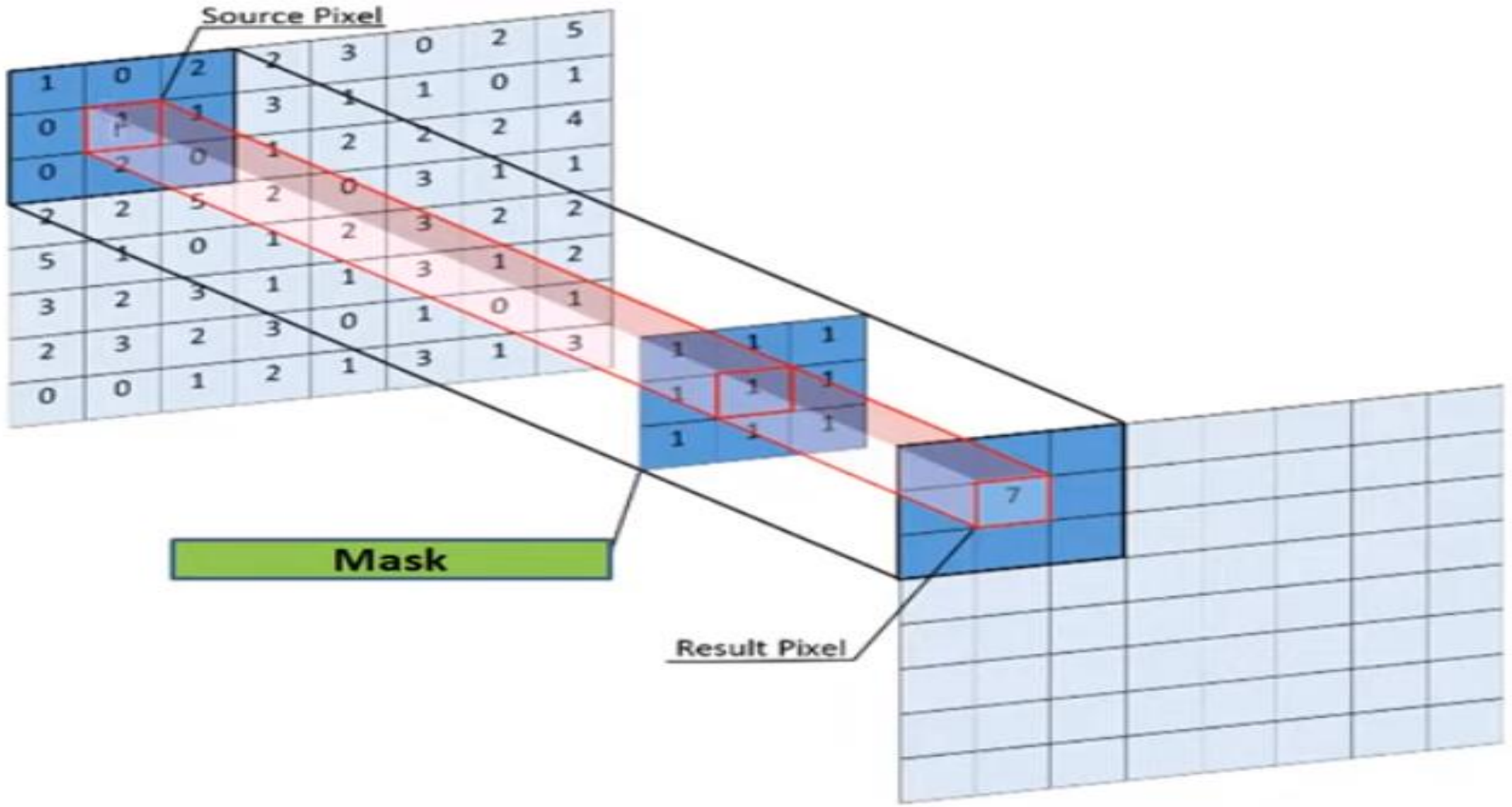
Example

F(x,y) =

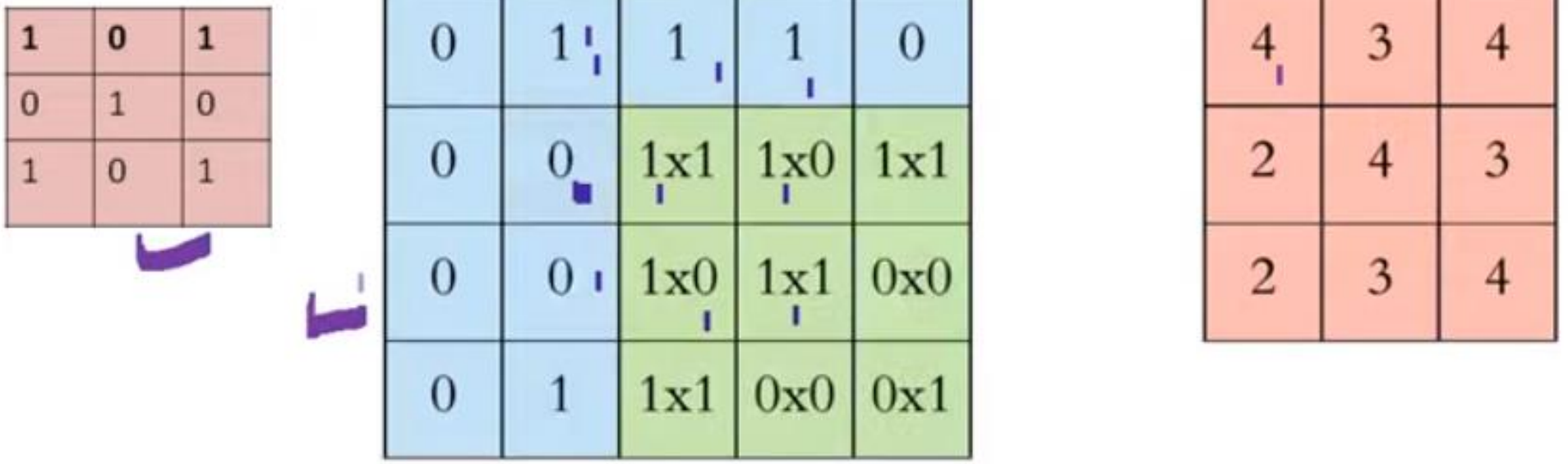
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

mask =

1	0	1
0	1	0
1	0	1


Source Pixel
Mask
Result Pixel

Solution without zero padding or wrap around



Solution using zero padding

8	4	3	0	1	2	3	6
6	6	0	0	2	5	7	8
9	7	8	1	5	6	0	3
5	6	7	1	3	8	6	0
3	8	4	6	5	4	3	8
6	0	7	0	7	9	8	8
0		3	0	6	6	4	1
3	8	6	7	1	4	0	4

8	**1**	**5**
7	1	3
4	6	5

Correlation =
8x0+1x0+5x0+7x0+1x8+3x4+4x0+6x6+5x6
= 86

Use of Correlation for template matching

0	0	0	0	0	0	0	0	0	0
0	8	4	3	0	1	2	3	6	0
0	6	6	0	0	2	5	7	8	0
0	9	7	8	1	5	6	0	3	0
0	5	6	7	1	3	8	6	0	0
0	3	8	4	6	5	4	3	8	0
0	6	0	7	0	7	9	8	8	0
0	0		3	0	6	6	4	1	0
0	3	8	6	7	1	4	0	4	0
0	0	0	0	0	0	0	0	0	0

Solution using zero padding

8	1	5
7	1	3
4	6	5

0	0	0	0	0	0	0	0	0	0
0	8	4	3	0	1	2	3	6	0
0	6	6	0	0	2	5	7	8	0
0	9	7	8	1	5	6	0	3	0
0	5	6	7	1	3	8	6	0	0
0	3	8	4	6	5	4	3	8	0
0	6	0	7	0	7	9	8	8	0
0	0		3	0	6	6	4	1	0
0	3	8	6	7	1	4	0	4	0
0	0	0	0	0	0	0	0	0	0

Correlation =
8x0+1x0+5x0+7x8+1x4+3x3+4x6+6x6+5x0
=129

8	1	5
7	1	3
4	6	5

Correlation = 7x1+1x3+4x6= **34**

0	0	0	0	0	0	0	0	0	0
0	8	4	3	0	1	2	3	6	0
0	6	6	0	0	2	5	7	8	0
0	9	7	8	1	5	6	0	3	0
0	5	6	7	1	3	8	6	0	0
0	3	8	4	6	5	4	3	8	0
0	6	0	7	0	7	9	8	8	0
0	0		3	0	6	6	4	1	0
0	3	8	6	7	1	4	0	4	0
0	0	0	0	0	0	0	0	0	0

Solution using zero padding

8	**1**	**5**
7	1	3
4	6	5

Correlation =
8x8+1x1+5x5+7x7+1x1+3x3+4x4+6x6+5x5
= 226

0	0	0	0	0	0	0	0	0	0
0	8	4	3	0	1	2	3	6	0
0	6	6	0	0	2	5	7	8	0
0	9	7	8	1	5	6	0	3	0
0	5	6	7	1	3	8	6	0	0
0	3	8	4	6	5	4	3	8	0
0	6	0	7	0	7	9	8	8	0
0	0		3	0	6	6	4	1	0
0	3	8	6	7	1	4	0	4	0
0	0	0	0	0	0	0	0	0	0

Solution using zero padding

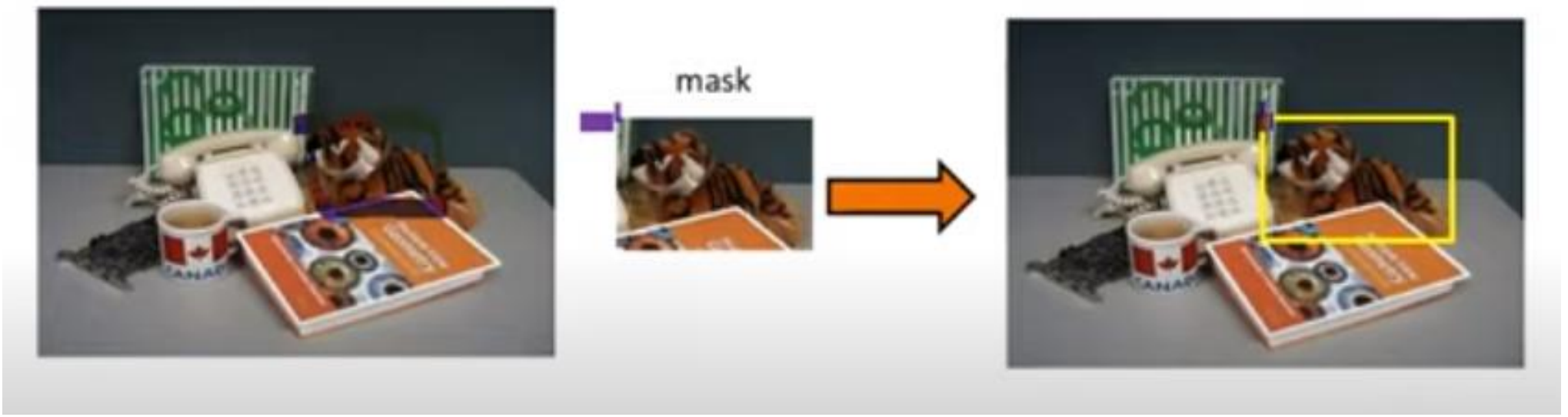
8	1	5
7	1	3
4	6	5

Correlation
={ 86,129,34,----------,226,----------37}

0	0	0	0	0	0	0	0	0	0
0	8	4	3	0	1	2	3	6	0
0	6	6	0	0	2	5	7	8	0
0	9	7	8	1	5	6	0	3	0
0	5	6	7	1	3	8	6	0	0
0	3	8	4	6	5	4	3	8	0
0	6	0	7	0	7	9	8	8	0
0	0		3	0	6	6	4	1	0
0	3	8	6	7	1	4	0	4	0
0	0	0	0	0	0	0	0	0	0

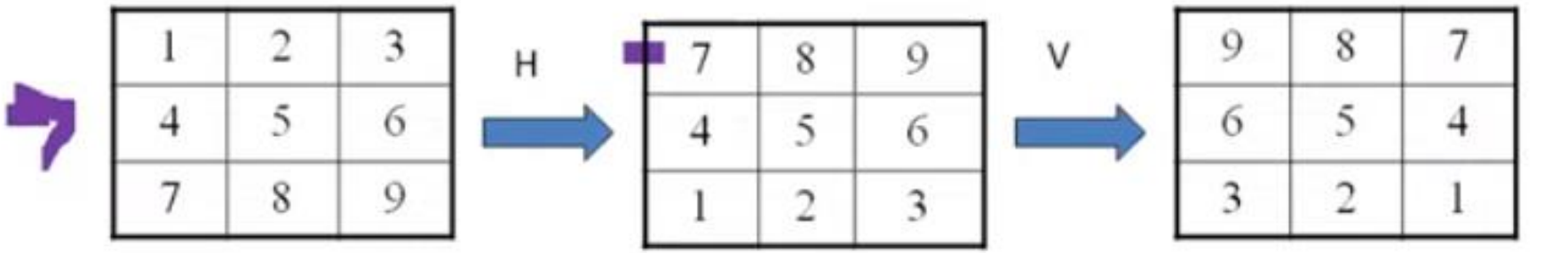
Correlation for an image

- Correlation measure the similarity between images or parts of images.



Convolution

- Same as correlation except that the mask is **flipped,** both horizontally and vertically.



For symmetric masks (i.e., h(i,j)=h(-i,-j)), convolution is equivalent to correlation!

Notation:

$h * f = f * h$

Image Transformations and color models

- As with any function, we can apply operators to an image

$g(x,y) = f(x,y) + 20$

$g(x,y) = f(-x,y)$

Image Transformations

- Image Transformation involves the transformation of image data in order to retrieve information from the image or preprocess the image for further usage.
- The image transformation are as followed:
  - **Image Translation**
  - **Reflection**
  - **Rotation**
  - **Scaling**
  - **Cropping**
  - **Shearing in x-axis**
  - **Shearing in y-axis**

Image Translation

- The most simple geometric transform is the translation along one of the image axis or all at once. An image, or the ensemble of pixels that are translated in the coordinate system, undergo the equations:

$$x' = x + x_T$$

$$y' = y + y_T$$

where x' and y' are the coordinates of a pixel P in the new image and x and y the coordinates of the original. The distance of which P is translated in every direction is denoted by x_T and y_T, respectively.

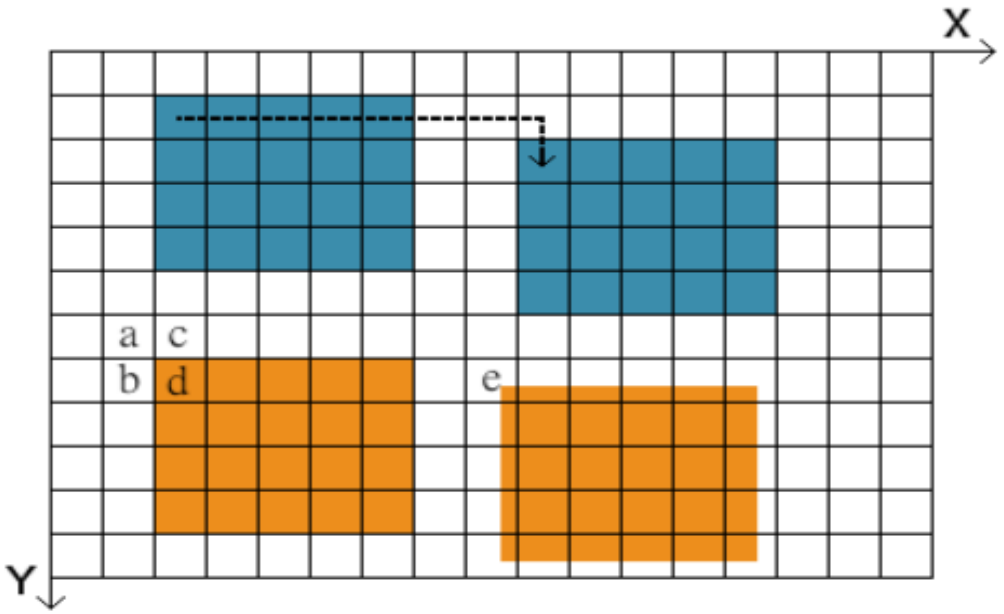
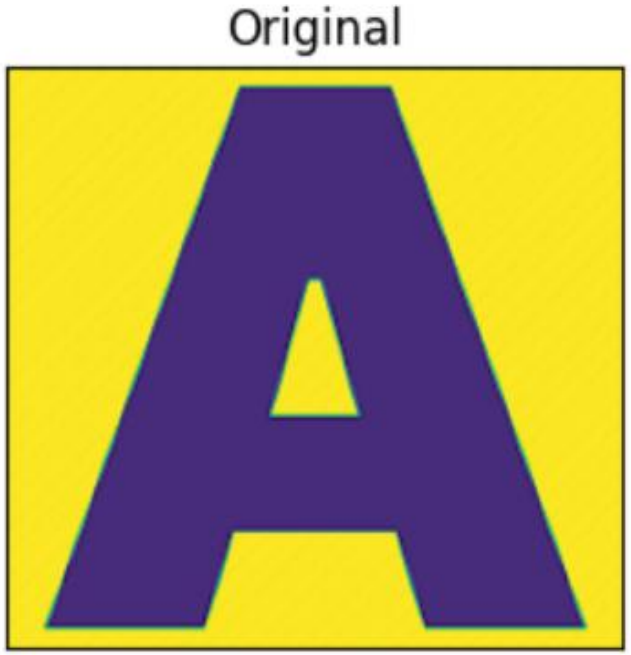
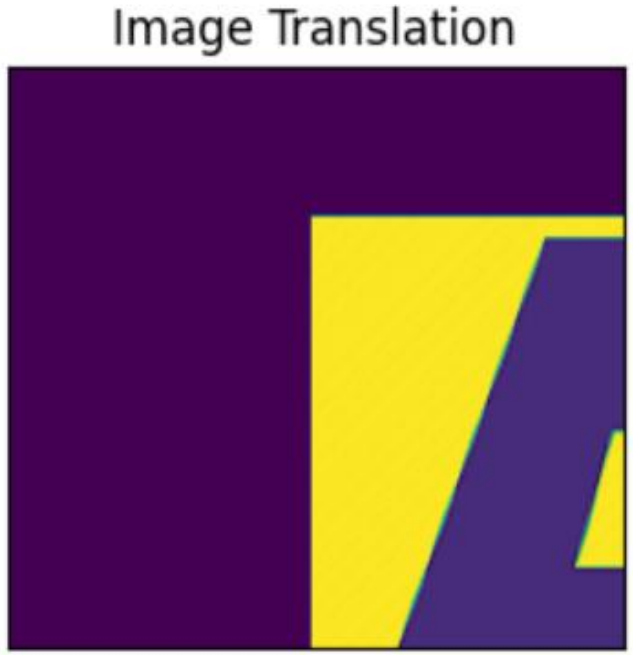


Image Translation

- In computer vision or image processing, image translation is the rectilinear shift of an image from one location to another, so the shifting of an object is called translation.
- In other words, translation is the shifting of an object's location.






```
dst = cv.warpAffine(img,M,(cols,rows))
```

Image Reflection

- The reflection/mirroring of an image is a simple operation that switches the position of pixels in an image on the left/right or top/bottom part, also called "flop" and "flip", respectively.
- Flipping reverses the X-axis, whereas flipping reverses consequently the Y-axis. Both operations together produce an image that is the same as the original image that has been rotated by 180°.

One has to take into account that the simple reversal of coordinates will place the new image outside of the old image. If the image needs, for example, to be mirrored along the Y-axis, the new image also needs to be translated by the height of the image:

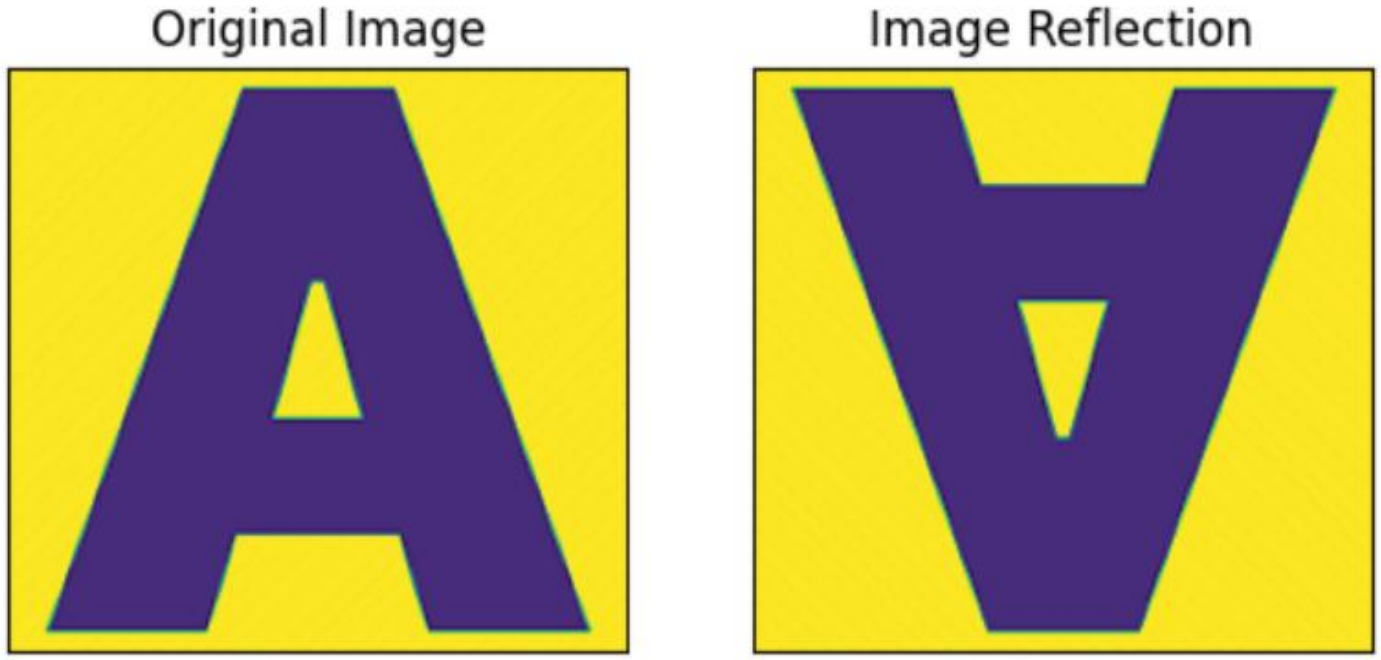
$$x' = x$$

$$y' = y_{max} - y$$

Since the operations only deal with integer coordinates and the boundaries of the new image (usually) equal the boundaries of the old image no interpolation is necessary.

Image Reflection

- Image reflection is used to flip the image vertically or horizontally.
- For reflection along the x-axis, we set the value of Sy to -1, Sx to 1, and vice-versa for the y-axis reflection.



To flip the image horizontally:

```
M = np.float32([[1, 0, 0], [0, -1, rows],[0,  0, 1]])
```

To flip the image vertically:

```
M = np.float32([[-1, 0, cols], [0, 1, 0], [0, 0, 1]])
```

Image Rotation

In order to match an image to another image or coordinate system with different orientation it needs to be rotated. With the rotation angle θ the equations for the new coordinates of a pixel are described by:

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

in a counterclockwise rotation. In an analogous formulation, the position of every pixel (x', y') in the new image can be written as a vector and the rotation matrix:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

In this simple case the point of rotation (x_0, y_0) is the point of origin of the coordinate system $(0,0)$.

In order to rotate a pixel at (x, y) around an arbitrary point (x_0, y_0) the equations become:

$$x' = x_0 + (x - x_0)\cos\theta + (y - y_0)\sin\theta$$

$$y' = y_0 - (x - x_0)\sin\theta + (y - y_0)\cos\theta$$

Image Rotation

The only pixel that does not move from its position is the pixel at (x_0, y_0) itself. Again, if the new image is being calculated form the old one, the equations that describe the rotation need to be solved for x and y, i.e. the inverse transform is needed. The values of all pixels of the rotated image need to be interpolated from the pixel values of the old image in most cases. Therefore, equations for x and y that access the values of all four (old) surrounding pixels at the position of the new pixel are required.

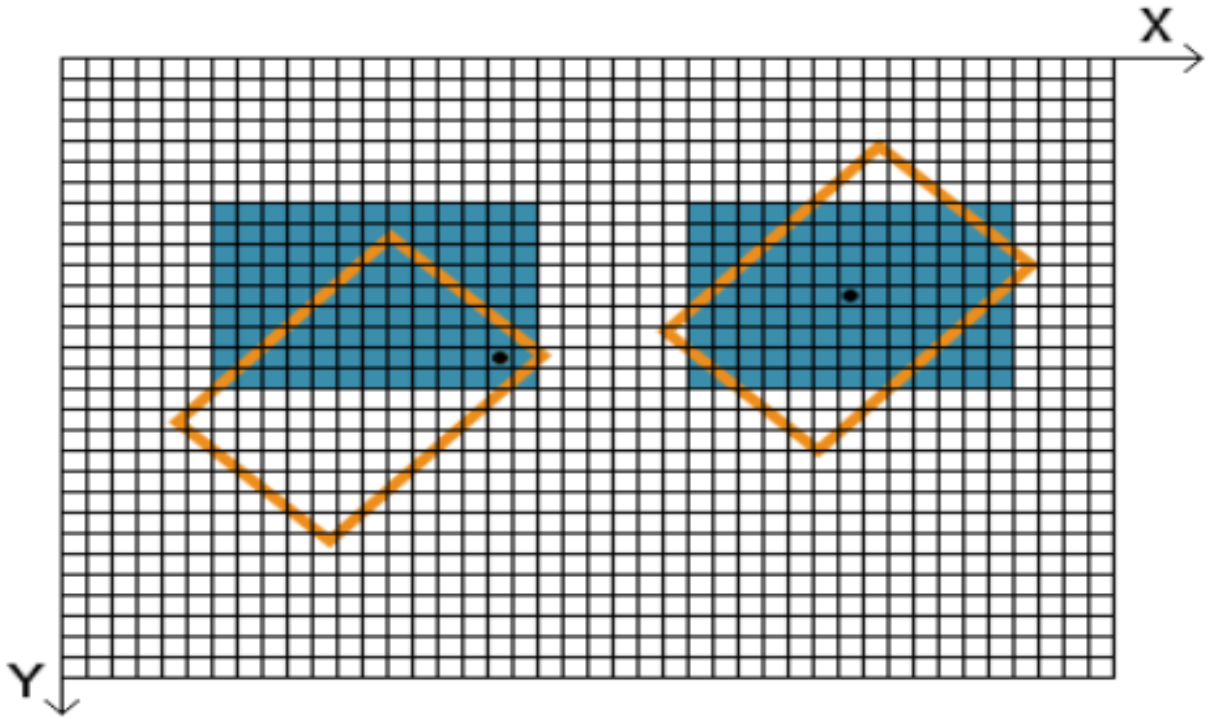


Image Rotation

- Image rotation is a common image processing routine with applications in matching, alignment, and other image-based algorithms, in image rotation the image is rotated by a definite angle.
- It is used extensively in data augmentation, especially when it comes to image classification.

Original Image

Image Rotation

We have used the get rotation matrix function to define the parameter required in the warpAffine function to tell the function to make a matrix that can give a required rotation angle(here it is 30 degrees) with shrinkage of the image by 40%.

Image Scaling

Changing the size of an image is also called scaling. This may be done for increasing or decreasing size, respectively. Assuming an equal image scaling of both axes the process can be expressed in its most basic form via:

$$x' = s\,x$$

$$y' = s\,y$$

with the scale factor s. The coordinate of every point is multiplied with the same factor.To scale points relative to a coordinate set (x_0, y_0), which may be for example $(0, 0)$ or the middle of the image, the equations become:

$$x' = x_0 + s(x - x_0)$$

$$y' = y_0 + s(y - y_0)\,.$$

Image Scaling

Again, for computing the new and scaled image we must use the equations that are solved for x and y in order to obtain all contributions of surrounding pixels in the old image that is mapped onto a new grid. Scaling an image up with an integer factor always results in the best image quality since no interpolation is necessary. Non-integer transformations require interpolation and result in a slightly smoothed image. The figure below shows on the right hand side an example of scaling the factor 2 and the point (x_0, y_0) in the top left corner of the image. The left hand side shows the scaling up with a factor close to 1. This results in visible image artifacts that come from the interpolation of four pixels of the old image in most cases whereas some pixel values are taken entirely from the old image. In the example below, the scaling also results in a smoothed out edge on two sides.

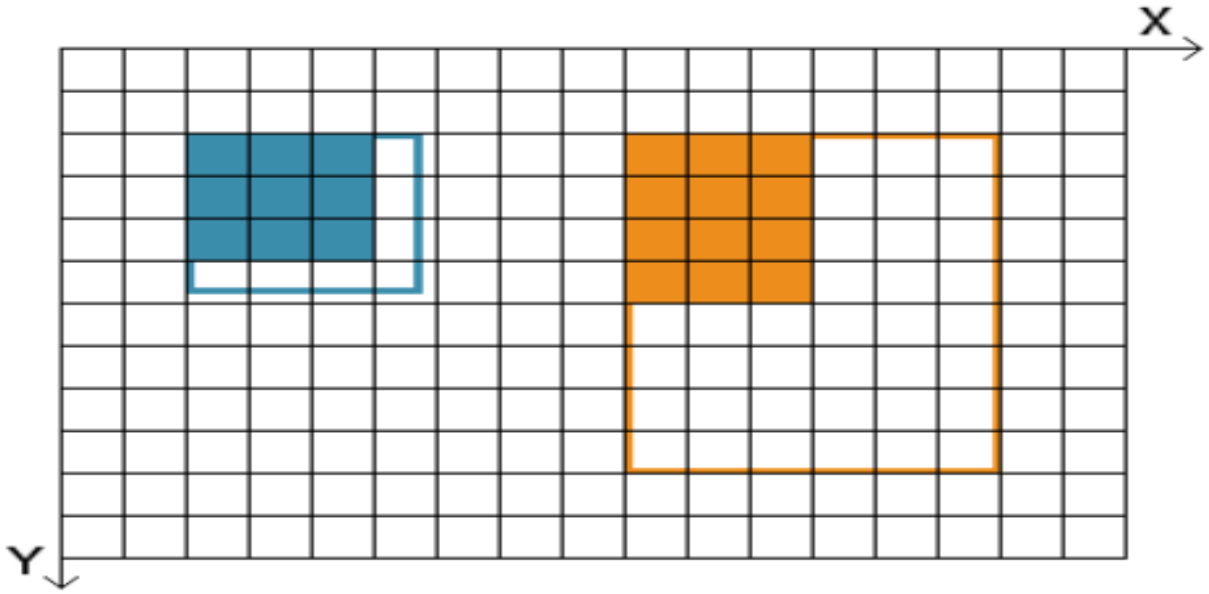
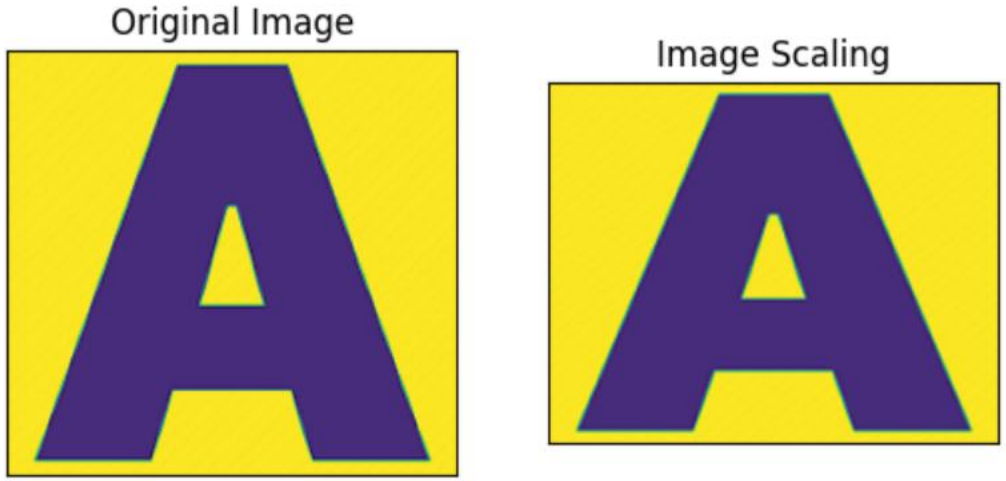


Image Scaling

- Image scaling is a process used to resize a digital image.
- We perform two things in the image scaling either we enlarge the image or we shrink the image, OpenCV has a built-in function **cv2.resize()** for image scaling.



Shrinking an image:

```
img_shrinked = cv2.resize(image, (350, 300),
                          interpolation =
cv2.INTER_AREA)
```

Note: Here 350 and 300 are the height and width of the shrunk image respectively

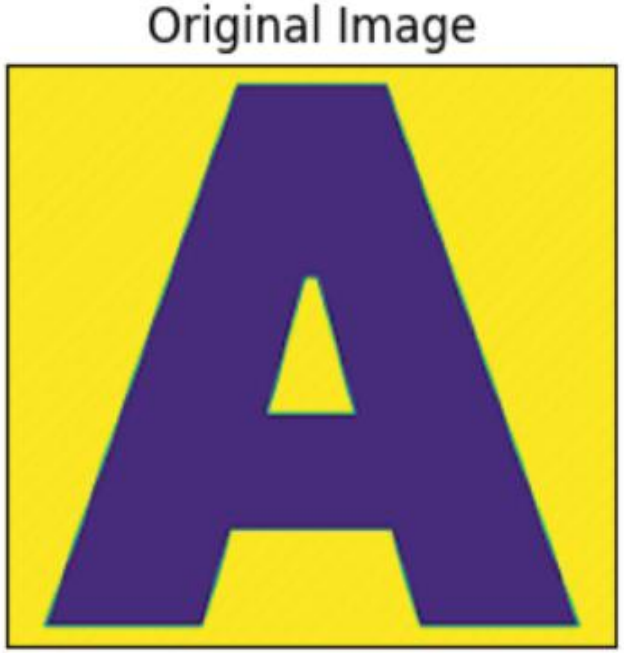
Enlarging Image:

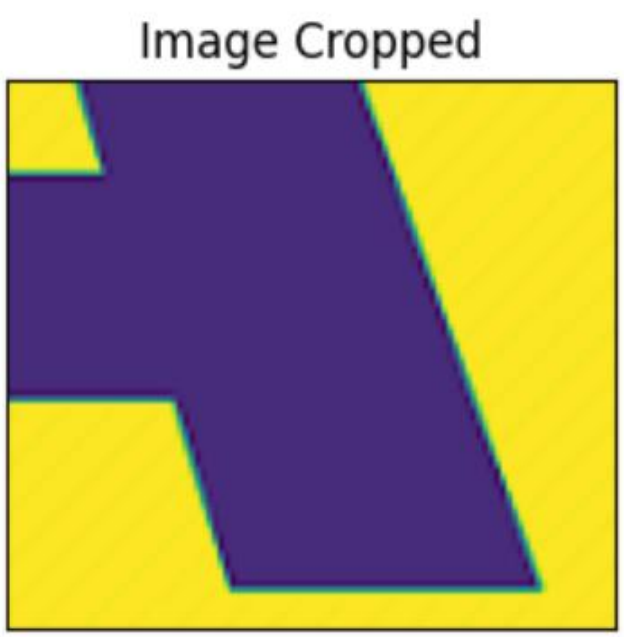
```
img_enlarged = cv2.resize(img_shrinked, None,
                          fx=1.5, fy=1.5,

interpolation=cv2.INTER_CUBIC)
```

Image Cropping

- Cropping is the removal of unwanted outer areas from an image.
- OpenCV loads the image as a NumPy array, we can crop the image simply by indexing the array, in our case, we choose to get 200 pixels from 100 to 300 on both axes.






```
cropped_img = img[100:300, 100:300]
```

Image Shearing in X-Axis

- While the shearing image is on the x-axis, the boundaries of the image that are parallel to the x-axis keep their location, and the edges parallel to the y-axis change their place depending on the shearing factor.

Original Image

Image Shearing in X-Axis

```
M = np.float32([[1, 0.5, 0], [0, 1, 0], [0, 0, 1]])
sheared_img = cv.warpPerspective(img, M,
                                 (int(cols*1.5),
                                 int(rows*1.5)))
```

Image Shearing in Y-Axis

- When shearing is done in the y-axis direction, the boundaries of the image that are parallel to the y-axis keep their location, and the edges parallel to the x-axis change their place depending on the shearing factor.

Original Image

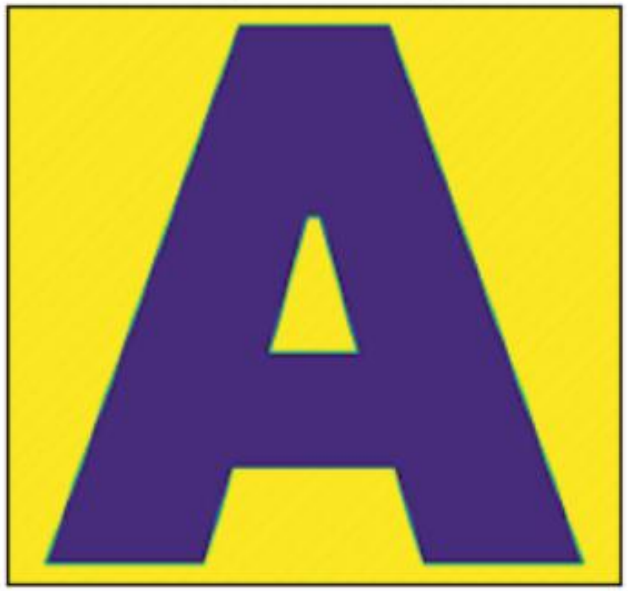
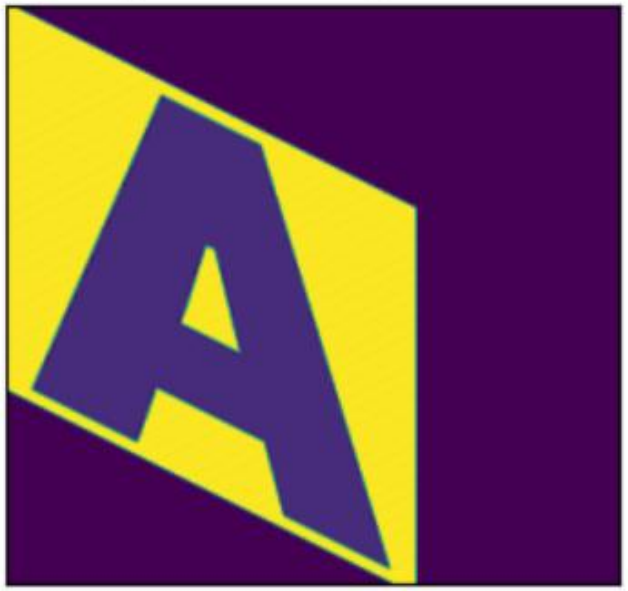


Image Shearing in Y-Axis



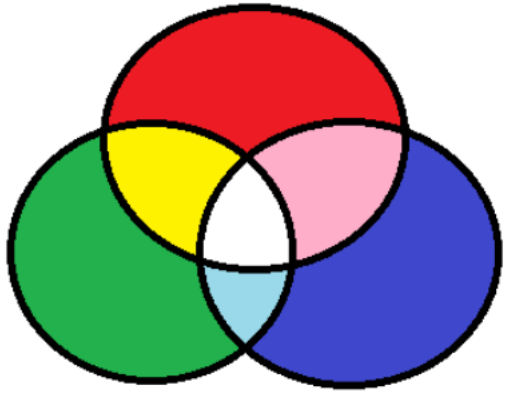
```
M = np.float32([[1, 0, 0], [0.5, 1, 0], [0, 0, 1]])
sheared_img = cv.warpPerspective(img, M,
                                 (int(cols*1.5),
                                 int(rows*1.5)))
```

Color Models

- **Color spaces** are a way to represent the color channels present in the image that gives the image that particular hue.
- There are several different color spaces and each has its own significance. Some of the popular color spaces are:
  - ***RGB* (Red, Green, Blue),**
  - ***HSV* (Hue, Saturation, Value),**
  - ***CMYK* (Cyan, Magenta, Yellow, Black),**

BGR color space:

- **BGR color space:** OpenCV's default color space is RGB. However, it actually stores color in the BGR format.
- It is an additive color model where the different intensities of Blue, Green and Red give different shades of color.



BGR color space:

- The R,G and B components are usually reported on a scale of 0 to 255, but can also be reported on a scale of 0 to 1 (distinguished as r,g and b here). Confusingly, these RGB values sometimes refer to **linear** units of light energy, or radiance, and sometimes to **nonlinear** units of perceived brightness (i.e., in equal perceptual steps):
- In Photoshop, both relative brightness (B) and the R,G and B components are reported in nonlinear (brightness) units.
- The conversion is:

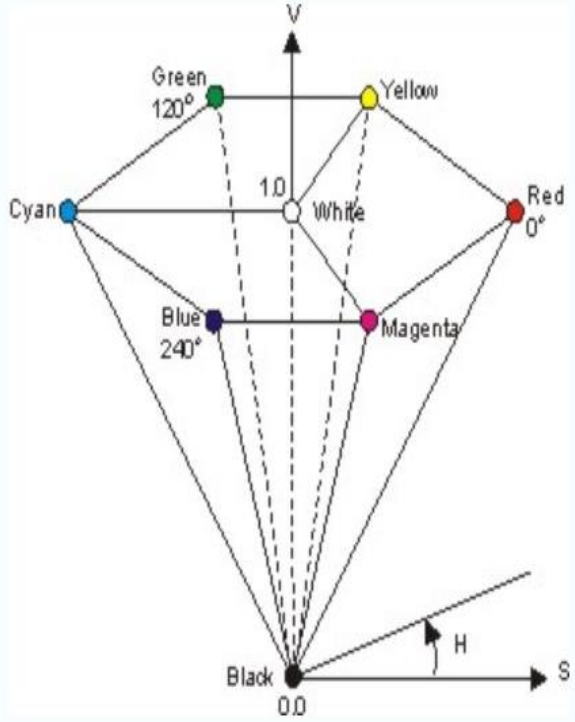
$$\text{(nonlinear) brightness} = \text{linear "brightness"}^{0.45}$$

- The formula has a generally similar effect to the more elaborate nonlinear conversion between CIE luminance (Y) and Lightness (L):

$$L = 116\,(Y/Y_n)^{1/3} - 16;\ 0.008856 < Y/Y_n$$

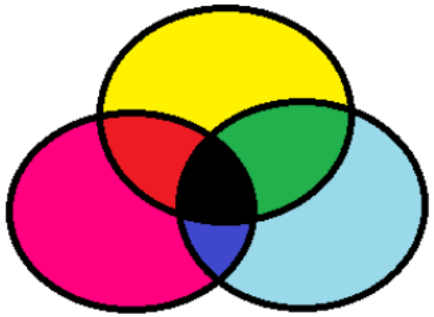
HSV color space:

- It stores color information in a cylindrical representation of RGB color points.
- It attempts to depict the colors as perceived by the human eye. Hue value varies from 0-179, Saturation value varies from 0-255 and Value varies varies from 0-255.
- It is mostly used for color segmentation purpose.



CMYK color space:

- The CMYK model works by partially or entirely masking colors on a lighter, usually white, background.
- The ink reduces the light that would otherwise be reflected. Such a model is called subtractive because inks “subtract” the colors red, green and blue from white light.
- White light minus red leaves cyan, white light minus green leaves magenta, and white light minus blue leaves yellow.



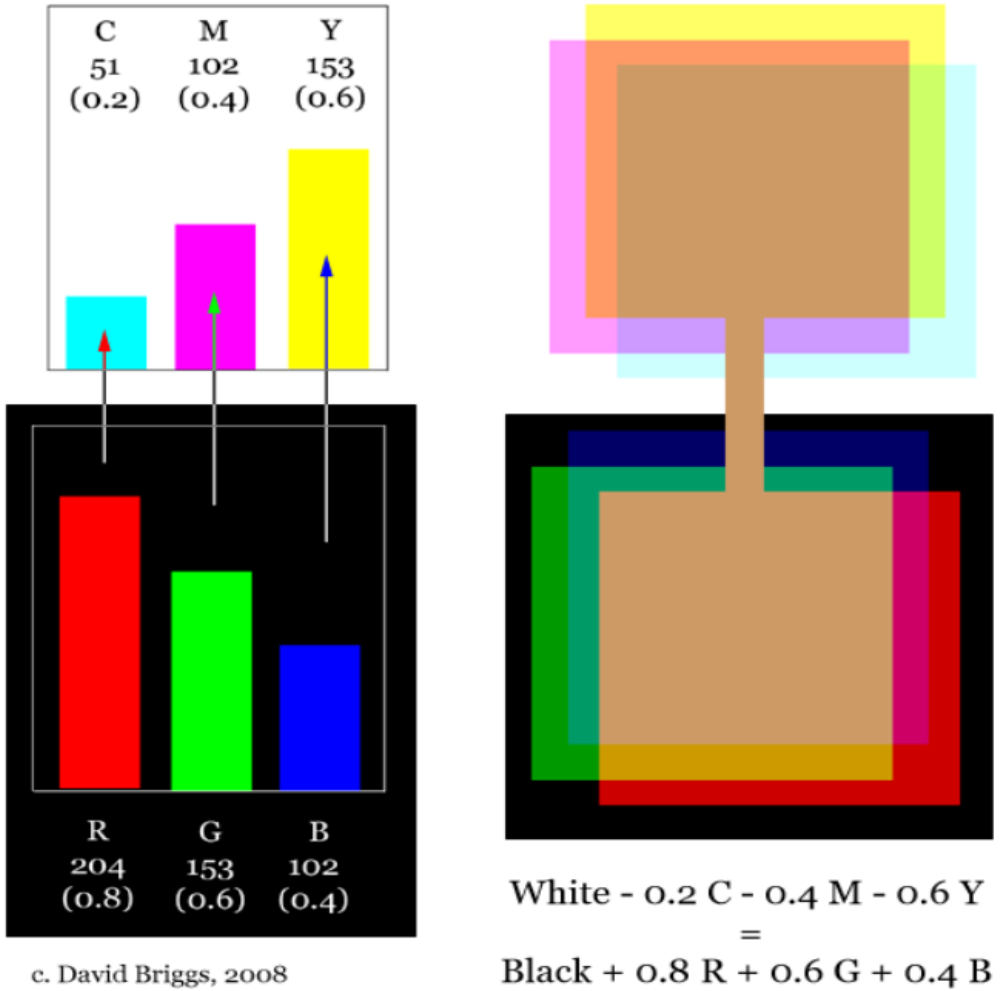
CMYK color space:

- The conversion is given by the formulae:

  C = 255 minus R (or 1 - r),

  M = 255 minus G (or 1 - g), and

  Y = 255 - B (or 1 minus b).

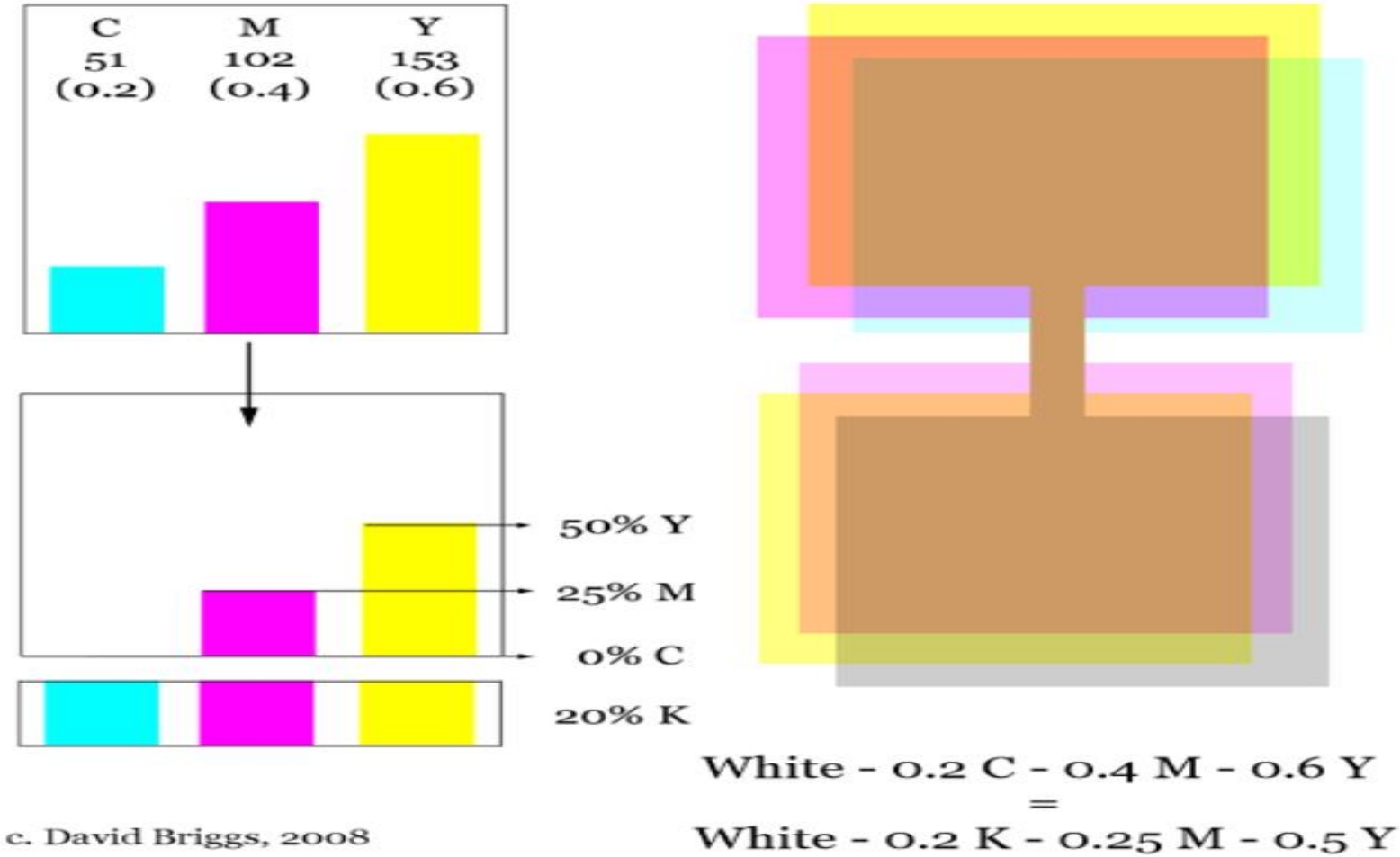


CMYK color space:

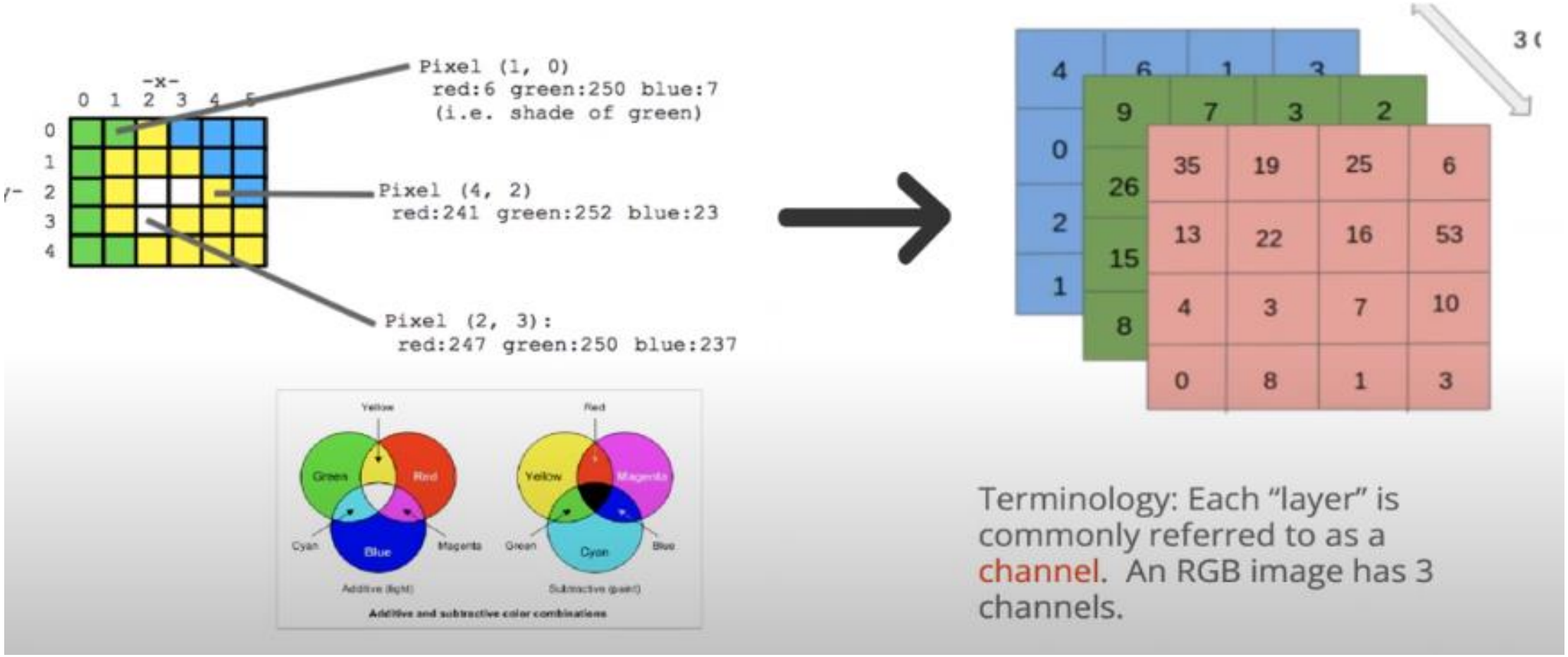
- The amount of the black component needed is conventionally specified by the letter K. CMYK values are typically reported as percentages. *Ideal* CMYK values can be calculated by simple formulae directly from CMY values, but these values are not accurate for color printing (Ford and Roberts, 1998):

$$\text{Black (K)} = \text{minimum of C,M,Y}$$
$$\text{Cyan}_{CMYK} = (C - K)/(1 - K)$$
$$\text{Magenta}_{CMYK} = (M - K)/(1 - K)$$
$$\text{Yellow}_{CMYK} = (Y - K)/(1 - K)$$

CMYK color space:



Color Space:


-x-
0 1 2 3 4 5
0
1
2
3
4
Pixel (1, 0)
red:6 green:250 blue:7
(i.e. shade of green)
Pixel (4, 2)
red:241 green:252 blue:23
Pixel (2, 3):
red:247 green:250 blue:237
4 6 1 3
9 7 3 2
0 35 19 25 6
26 13 22 16 53
2 15 4 3 7 10
1 8 0 8 1 3
Yellow
Green
Red
Cyan
Blue
Magenta
Red
Yellow
Magenta
Green
Cyan
Blue
Additive and subtractive color combinations
Terminology: Each "layer" is commonly referred to as a channel. An RGB image has 3 channels.