Sistemi Intelligenti Avanzati
Corso di Laurea in Informatica, A.A. 2023-2024
Università degli Studi di Milano

Robotic Vision

Methods, Limits, Solutions

Michele Antonazzi
Dipartimento di Informatica
michele.antonazzi@unimi.it

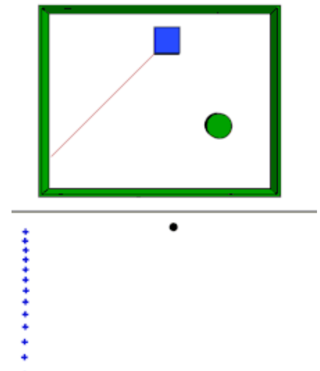
Outline

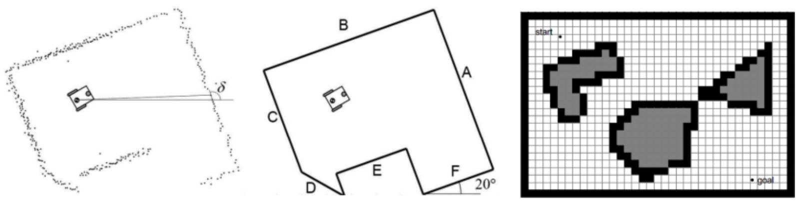
- **<u>What is Robotic Vision?</u>**
- Feature based methods
- A feature based method for Door Detection
- Deep Learning in Computer Vision (CNN)
- Deep Learning in RV for Door Detection
- A CNN for image classification: practical example

Perception in Mobile Robotics

Perception: interpretation of sensed data in a meaningful way

- A robot perceive the environment using exteroceptive sensors
- The most used are Range Finders and Cameras
- Data from cameras are difficult to be interpret and need an intensive **feature extraction**





From [Siegwart, Introduction to Autonomous Mobile Robots]

Feature Extraction in Images

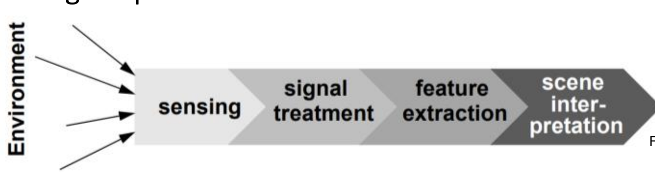
Raw data:

- Every bit of information is used
- Data has a low expressive power
- We can use raw sensors data for solving low-level tasks (e.g., obstacle avoidance).

But with images?

Images need an intensive **feature extraction**:

- **Low-level features (geometric primitives):** abstraction of raw data which deletes poor or useless information
- **High-level features (objects):** maximum abstraction of raw data, providing a lower volume of information with a high expressiveness



From [Siegwart, Introduction to Autonomous Mobile Robots]

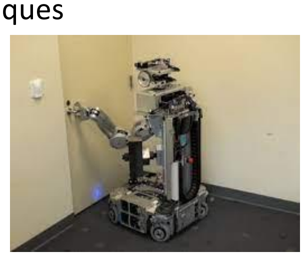
Robotic Vision

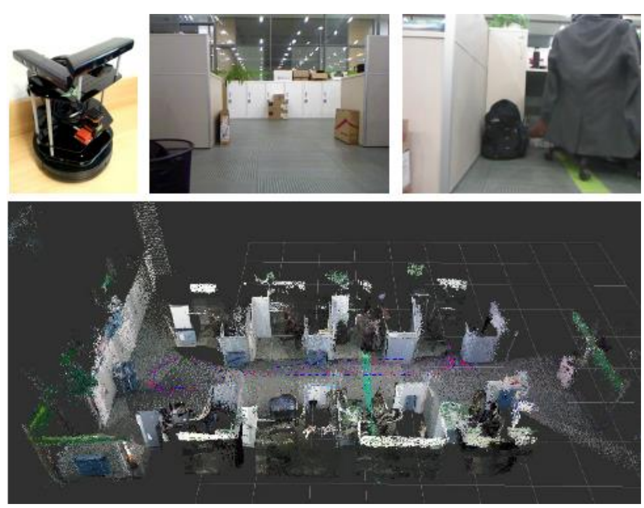
- Feature extraction through **Computer Vision** techniques
- **Robotic Vision** is the application of Computer Vision techniques in Mobile robotics for solving high level tasks
- While *Computer Vision* translates images into **information**, *Robotic Vision* translates images into **actions:**
  - A robot is an *active* agent
  - A robot often operates in *uncontrolled* and *unpredictable conditions* (the real world)
  - The actions executed by a robot are based on *incomplete* and *uncertain knowledge*
  - Actions can have potentially catastrophic results

Computer Vision in Mobile Robotics

Computer Vision in Mobile Robotics is useful to enable mobile robot to solve high level tasks:

- Object Grasping
- Object finding
- **Visual SLAM:** Simultaneous Localization and Mapping using visual features captured with CV techniques



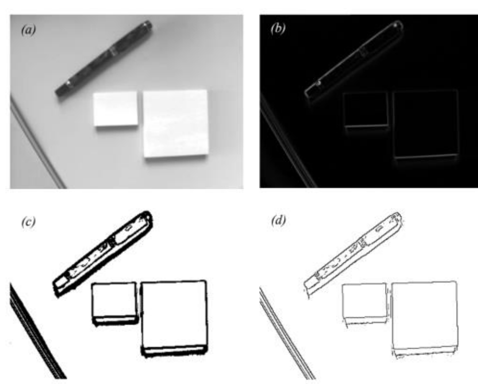


Outline

- What is Robotic Vision?
- **Feature based methods**
- A feature based method for Door Detection
- Deep Learning in Computer Vision (CNN)
- Deep Learning in RV for Door Detection
- A CNN for image classification: practical example

Feature-based Methods for Object Detection

Feature-based methods:
- The *low-level features* (edges, corners, points, …) are extracted through **image processing** techniques
- The *high-level features* are obtained by combining the low-level features in well-engineered **geometric models** that describe the object of interest



From [Siegwart, Introduction to Autonomous Mobile Robots]

Image Processing for Feature Extraction

Image processing is a form of signal processing:

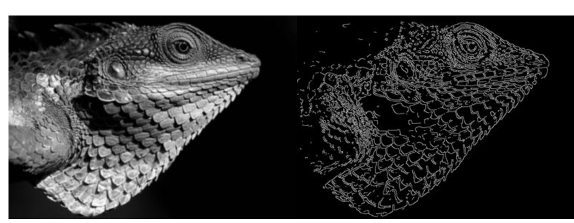
- The input is an image and the output is another image or a series of features associated to the image
- It treats images as discrete two-dimensional signals $I(x, y)$, where:
  - (x, y) are the spatial coordinates and
  - The value of I at any point (pixel) is the *intensity* or *gray level*

Image Filtering

Image filtering:

- Filtering refers to the process of accepting or rejecting certain frequencies
- **Lowpass filters** attenuate signals
  - Goal: blur images or remove noise
- **Highpass filters** cut off the frequencies lower than the cutoff frequency
  - Goal: feature extraction (edge, corner, …)

Lowpass filter



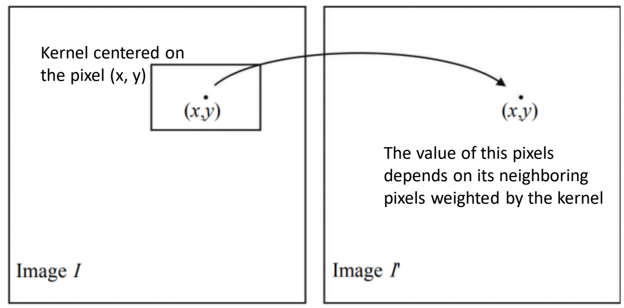
Highpass filter

Filtering in Images: Convolution

Convolution:

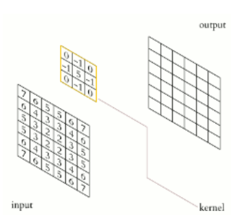
- applies spatial filtering to images
- modifies the intensity of each pixel based on its neighborhood weighted by a **kernel**

$$I'(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) I(x+s, y+t)$$

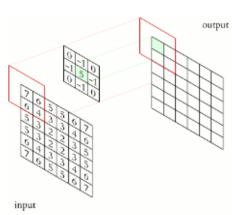


- I and I': input and output image
- w: the kernel (a small matrix containing the weights associated to the pixels)
- The kernel (odd) dimensions

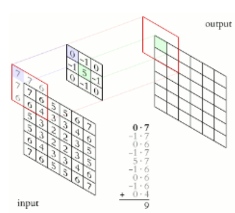
Convolution Explained



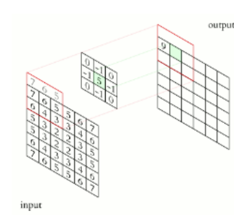
1



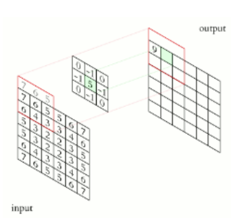
2



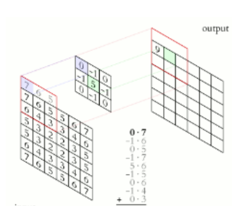
3



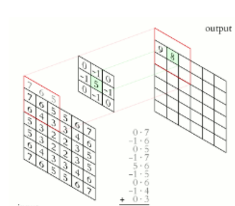
4



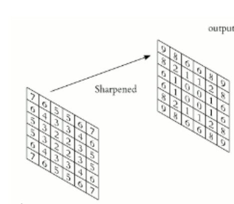
5



6



7



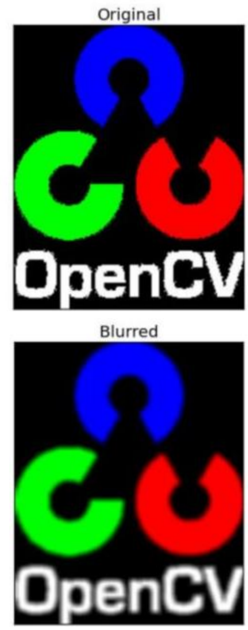
8

Smoothing Filters: Average filter

- Smoothing filters are uses for **blurring** and **noise reduction**
- **Average filter:** it simply yields the standard average of the pixels in the mask

$$\boldsymbol{w} = \frac{\mathbf{1}}{\mathbf{9}}\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{bmatrix}$$

  - **NB:** the pixels are first summed and then divided by 9: this is computationally more efficient that multiply every pixel by $\frac{1}{9}$



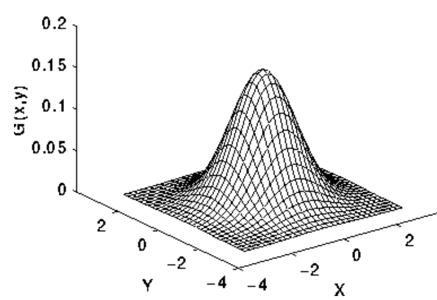
Smoothing Filters: Gaussian filter

The idea is to build a kernel by approximating an isotropic 2D Gaussian

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

Kernel parameters:

- d: kernel's dimensions
- σ: the standard deviation



Smoothing Filters: Gaussian filter

$$\begin{bmatrix} G_\sigma(-1,-1) & G_\sigma(-1,0) & G_\sigma(-1,1) \\ G_\sigma(0,-1) & G_\sigma(0,0) & G_\sigma(0,1) \\ G_\sigma(1,-1) & G_\sigma(1,0) & G_\sigma(1,1) \end{bmatrix}$$

The algorithm to build a 2D Gaussian kernel G_σ:

1. Choose the parameters d and σ (e.g., $d = 3, \sigma = 0.85$)
2. Sample $G_\sigma(x, y)$ centered to the kernel
3. Scaling all values with respect to the low value (i.e., the lowest values are set to 1)
4. Round the coefficients to the nearest integer
5. Calculate the normalization coefficient by summing the coefficients

$$\begin{bmatrix} 0.0551 & 0.1102 & 0.0551 \\ 0.1102 & 0.2202 & 0.1102 \\ 0.0551 & 0.1102 & 0.0551 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 1.99 & 1.0 \\ 1.99 & 3.99 & 1.99 \\ 1.0 & 1.99 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$w = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Smoothing Filters: Gaussian filter

$$\begin{bmatrix} G_\sigma(-1,-1) & G_\sigma(-1,0) & G_\sigma(-1,1) \\ G_\sigma(0,-1) & G_\sigma(0,0) & G_\sigma(0,1) \\ G_\sigma(1,-1) & G_\sigma(1,0) & G_\sigma(1,1) \end{bmatrix}$$

The algorithm to build a 2D Gaussian kernel

1. Choose the parameters d and σ (e.g., $d = 3, \sigma = 0.85$)
2. Sample $G_\sigma(x, y)$ centered to the kernel
3. Scaling all values with respect to the low value (i.e., the lowest values are set to 1)
4. Round the coefficients to the nearest integer
5. Calculate the normalization coefficient by summing the coefficients

$$\begin{bmatrix} 0.0551 & 0.1102 & 0.0551 \\ 0.1102 & 0.2202 & 0.1102 \\ 0.0551 & 0.1102 & 0.0551 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 1.99 & 1.0 \\ 1.99 & 3.99 & 1.99 \\ 1.0 & 1.99 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$w = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Smoothing Filters: Gaussian filter

$$\begin{bmatrix} G_\sigma(-1,-1) & G_\sigma(-1,0) & G_\sigma(-1,1) \\ G_\sigma(0,-1) & G_\sigma(0,0) & G_\sigma(0,1) \\ G_\sigma(1,-1) & G_\sigma(1,0) & G_\sigma(1,1) \end{bmatrix}$$

The algorithm to build a 2D Gaussian kernel

1. Choose the parameters d and σ (e.g., $d = 3, \sigma = 0.85$)
2. Sample $G_\sigma(x, y)$ centered to the kernel
3. Scaling all values with respect to the low value (i.e., the lowest values are set to 1)
4. Round the coefficients to the nearest integer
5. Calculate the normalization coefficient by summing the coefficients

$$\begin{bmatrix} 0.0551 & 0.1102 & 0.0551 \\ 0.1102 & 0.2202 & 0.1102 \\ 0.0551 & 0.1102 & 0.0551 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 1.99 & 1.0 \\ 1.99 & 3.99 & 1.99 \\ 1.0 & 1.99 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$w = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Smoothing Filters: Gaussian filter

$$\begin{bmatrix} G_\sigma(-1,-1) & G_\sigma(-1,0) & G_\sigma(-1,1) \\ G_\sigma(0,-1) & G_\sigma(0,0) & G_\sigma(0,1) \\ G_\sigma(1,-1) & G_\sigma(1,0) & G_\sigma(1,1) \end{bmatrix}$$

The algorithm to build a 2D Gaussian kernel

1. Choose the parameters d and σ (e.g., $d = 3, \sigma = 0.85$)
2. Sample $G_\sigma(x, y)$ centered to the kernel
3. Scaling all values with respect to the low value (i.e., the lowest values are set to 1)
4. Round the coefficients to the nearest integer
5. Calculate the normalization coefficient by summing the coefficients

$$\begin{bmatrix} 0.0551 & 0.1102 & 0.0551 \\ 0.1102 & 0.2202 & 0.1102 \\ 0.0551 & 0.1102 & 0.0551 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 1.99 & 1.0 \\ 1.99 & 3.99 & 1.99 \\ 1.0 & 1.99 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$w = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Smoothing Filters: Gaussian filter

The algorithm to build a 2D Gaussian kernel
1. Choose the parameters d and σ (e.g., $d = 3, \sigma = 0.85$)
2. Sample $G_\sigma(x, y)$ centered to the kernel
3. Scaling all values with respect to the low value (i.e., the lowest values are set to 1)
4. Round the coefficients to the nearest integer
5. Calculate the normalization coefficient by summing the coefficients

$$\begin{bmatrix} G_\sigma(-1,-1) & G_\sigma(-1,0) & G_\sigma(-1,1) \\ G_\sigma(0,-1) & G_\sigma(0,0) & G_\sigma(0,1) \\ G_\sigma(1,-1) & G_\sigma(1,0) & G_\sigma(1,1) \end{bmatrix}$$

$$\begin{bmatrix} 0.0551 & 0.1102 & 0.0551 \\ 0.1102 & 0.2202 & 0.1102 \\ 0.0551 & 0.1102 & 0.0551 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 1.99 & 1.0 \\ 1.99 & 3.99 & 1.99 \\ 1.0 & 1.99 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$w = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Smoothing Filters: Gaussian filter

The algorithm to build a 2D Gaussian kernel
1. Choose the parameters d and σ (e.g., $d = 3, \sigma = 0.85$)
2. Sample $G_\sigma(x, y)$ centered to the kernel
3. Scaling all values with respect to the low value (i.e., the lowest values are set to 1)
4. Round the coefficients to the nearest integer
5. Calculate the normalization coefficient by summing the coefficients

$$\begin{bmatrix} G_\sigma(-1,-1) & G_\sigma(-1,0) & G_\sigma(-1,1) \\ G_\sigma(0,-1) & G_\sigma(0,0) & G_\sigma(0,1) \\ G_\sigma(1,-1) & G_\sigma(1,0) & G_\sigma(1,1) \end{bmatrix}$$

$$\begin{bmatrix} 0.0551 & 0.1102 & 0.0551 \\ 0.1102 & 0.2202 & 0.1102 \\ 0.0551 & 0.1102 & 0.0551 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 1.99 & 1.0 \\ 1.99 & 3.99 & 1.99 \\ 1.0 & 1.99 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$w = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Smoothing Filters: Gaussian filter

$$\begin{bmatrix} G_\sigma(-1,-1) & G_\sigma(-1,0) & G_\sigma(-1,1) \\ G_\sigma(0,-1) & G_\sigma(0,0) & G_\sigma(0,1) \\ G_\sigma(1,-1) & G_\sigma(1,0) & G_\sigma(1,1) \end{bmatrix}$$

$$\begin{bmatrix} 0.0551 & 0.1102 & 0.0551 \\ 0.1102 & 0.2202 & 0.1102 \\ 0.0551 & 0.1102 & 0.0551 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 1.99 & 1.0 \\ 1.99 & 3.99 & 1.99 \\ 1.0 & 1.99 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$w = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

The algorithm to build a 2D Gaussian kernel

1. Choose the parameters d and σ (e.g., $d = 3, \sigma = 0.85$)
2. Sample $G_\sigma(x, y)$ centered to the kernel
3. Scaling all values with respect to the low value (i.e., the lowest values are set to 1)
4. Round the coefficients to the nearest integer
5. Calculate the normalization coefficient by summing the coefficients

Smoothing Filters: Gaussian filter

$$\begin{bmatrix} G_\sigma(-1,-1) & G_\sigma(-1,0) & G_\sigma(-1,1) \\ G_\sigma(0,-1) & G_\sigma(0,0) & G_\sigma(0,1) \\ G_\sigma(1,-1) & G_\sigma(1,0) & G_\sigma(1,1) \end{bmatrix}$$

$$\begin{bmatrix} 0.0551 & 0.1102 & 0.0551 \\ 0.1102 & 0.2202 & 0.1102 \\ 0.0551 & 0.1102 & 0.0551 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 1.99 & 1.0 \\ 1.99 & 3.99 & 1.99 \\ 1.0 & 1.99 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$w = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

The algorithm to build a 2D Gaussian kernel

1. Choose the parameters d and σ (e.g., $d = 3, \sigma = 0.85$)
2. Sample $G_\sigma(x, y)$ centered to the kernel
3. Scaling all values with respect to the low value (i.e., the lowest values are set to 1)
4. Round the coefficients to the nearest integer
5. Calculate the normalization coefficient by summing the coefficients

Smoothing Filters: Gaussian filter

$$\begin{bmatrix} G_\sigma(-1,-1) & G_\sigma(-1,0) & G_\sigma(-1,1) \\ G_\sigma(0,-1) & G_\sigma(0,0) & G_\sigma(0,1) \\ G_\sigma(1,-1) & G_\sigma(1,0) & G_\sigma(1,1) \end{bmatrix}$$

The algorithm to build a 2D Gaussian kernel

1. Choose the parameters d and σ (e.g., $d = 3, \sigma = 0.85$)
2. Sample $G_\sigma(x, y)$ centered to the kernel
3. Scaling all values with respect to the low value (i.e., the lowest values are set to 1)
4. Round the coefficients to the nearest integer
5. Calculate the normalization coefficient by summing the coefficients

$$\begin{bmatrix} 0.0551 & 0.1102 & 0.0551 \\ 0.1102 & 0.2202 & 0.1102 \\ 0.0551 & 0.1102 & 0.0551 \end{bmatrix}$$

$$\begin{bmatrix} 1.0 & 1.99 & 1.0 \\ 1.99 & 3.99 & 1.99 \\ 1.0 & 1.99 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$w = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Smoothing Filters: Gaussian filter

$$\begin{bmatrix} G_\sigma(-1,-1) & G_\sigma(-1,0) & G_\sigma(-1,1) \\ G_\sigma(0,-1) & G_\sigma(0,0) & G_\sigma(0,1) \\ G_\sigma(1,-1) & G_\sigma(1,0) & G_\sigma(1,1) \end{bmatrix}$$

The algorithm to build a 2D Gaussian kernel

1. Choose the parameters d and σ (e.g., $d = 3, \sigma = 0.85$)
2. Sample $G_\sigma(x, y)$ centered to the kernel
3. Scaling all values with respect to the low value (i.e., the lowest values are set to 1)
4. Round the coefficients to the nearest integer
5. Calculate the normalization coefficient by summing the coefficients

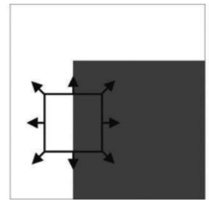
$$\begin{bmatrix} 0.0551 & 0.1102 & 0.0551 \\ 0.1102 & 0.2202 & 0.1102 \\ 0.0551 & 0.1102 & 0.0551 \end{bmatrix}$$

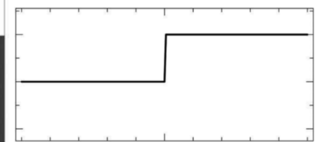
$$\begin{bmatrix} 1.0 & 1.99 & 1.0 \\ 1.99 & 3.99 & 1.99 \\ 1.0 & 1.99 & 1.0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

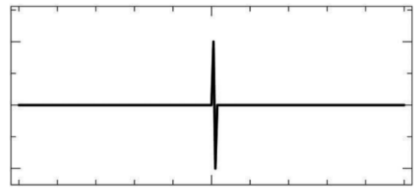
$$w = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Edge Detection

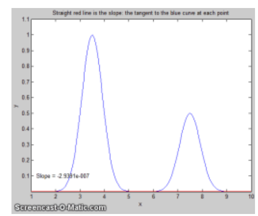




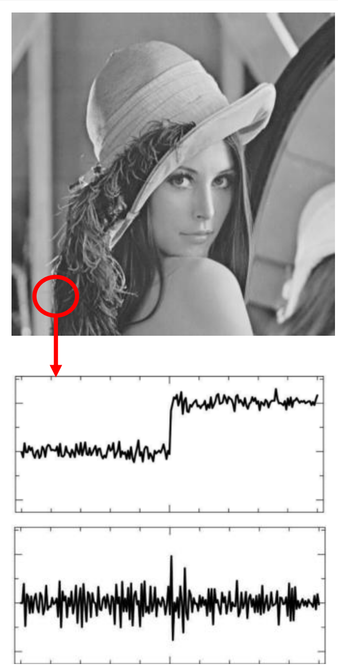
What are edges?
- **For us:** outlines of the shapes in an image
- **Signal domain:** significant change in signal intensity (brightness change)



To find edges, we can simply differentiate the signal:
- Edge = a large transition in signal intensity
- We can compute the **signal first derivative = the rate of change**



Edge Detection: a Challenging Task

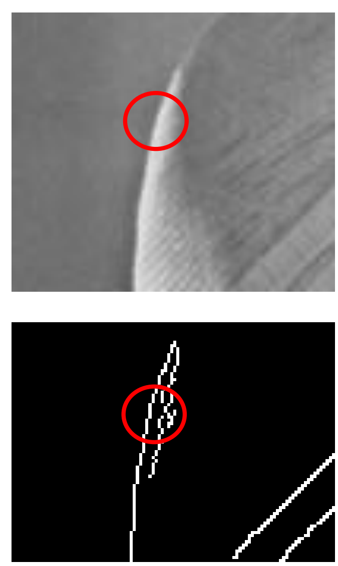


1. The signal of an image is noisy and simply computes derivatives is not enough to detect edges
2. How can we distinguish between noisy patterns and real shape contours in images?
3. Typically, the signal change smoothly in proximity of an edge, so how to precisely locate an edge?

Edge Detection: a Challenging Task

1. The signal of an image is noisy and simply computes derivatives is not enough to detect edges
2. How can we distinguish between noisy patterns and real shape contours in images?
3. Typically, the signal change smoothly in proximity of an edge, so how to precisely locate an edge?

Edge Detection: a Challenging Task



1. The signal of an image is noisy and simply computes derivatives is not enough to detect edges
2. How can we distinguish between noisy patterns and real shape contours in images?
3. Typically, the signal change smoothly in proximity of an edge, so how to precisely locate an edge?

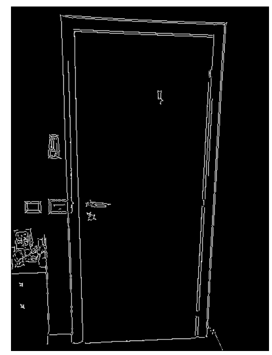
Canny Edge Detector

The Canny edge Detector is a multi-phase algorithm to detect edges in images
It treats edges detection as a signal-processing problem with 3 specific goal:

- Minimizing the edges generated by the image noise
- Achieving the highest precision on the location of edges
- Minimizing the edge responses associated to a single edge

Steps:

1. Noise reduction
2. Gradient calculation
3. Non-maximum suppression
4. Double threshold
5. Edge tracking



Canny Edge Detector: Noise Reduction

Starting for an image I, the noise reduction is performed by applying a Gaussian filter G_σ to blur the image

$$I_G = G_\sigma * I$$

I

I_G

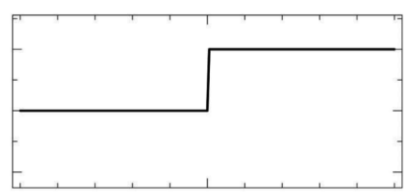
Canny Edge Detector: Gradient Calculation

- In images (considered as discrete bi-dimensional signals), we can **approximate derivatives** through **convolution** with kernels that **highlights the signal change in both x and y directions**
- This can be done with the **Sobel kernels:**

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ -1 & 2 & 1 \end{bmatrix}$$

Canny Edge Detector: Gradient Calculation

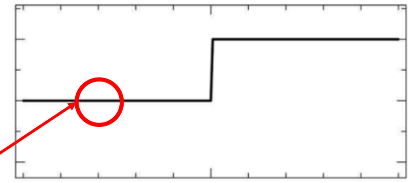


- Suppose to have a mono-dimensional discrete signal and a Sobel filter

$$S_x = \begin{bmatrix} -2 & 0 & 2 \end{bmatrix}$$

- If we convolve the signal with S_x in a flat region, we obtain a **very small value** (near to 0)
- If we convolve the signal with S_x in a region where it grows, we obtain a **high positive value**
- If we convolve the signal with S_x in a region where it falls, we obtain a high **negative value**

Canny Edge Detector: Gradient Calculation

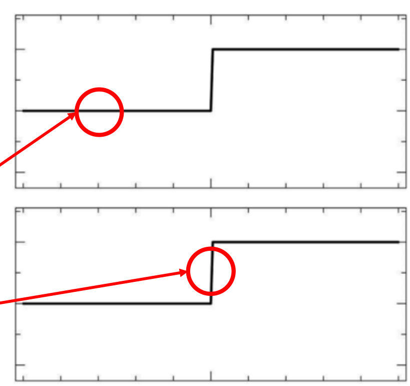


- Suppose to have a mono-dimensional discrete signal and a Sobel filter

$$S_x = [-2 \quad 0 \quad 2]$$

- If we convolve the signal with S_x in a flat region, we obtain a **very small value** (near to 0)
- If we convolve the signal with S_x in a region where it grows, we obtain a **high positive value**
- If we convolve the signal with S_x in a region where it falls, we obtain a **high negative value**

Canny Edge Detector: Gradient Calculation



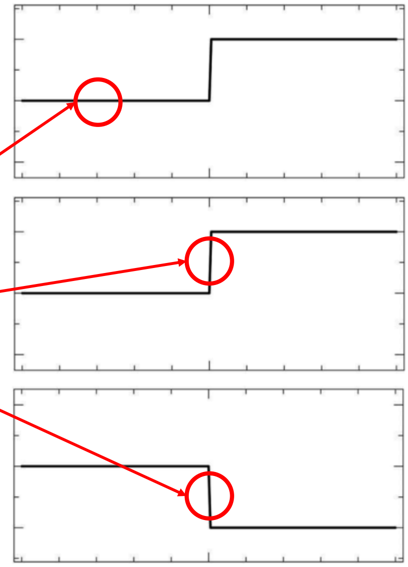
- Suppose to have a mono-dimensional discrete signal and a Sobel filter

$$S_x = [-2 \quad 0 \quad 2]$$

- If we convolve the signal with S_x in a flat region, we obtain a **very small value** (near to 0)
- If we convolve the signal with S_x in a region where it grows, we obtain a **high positive value**
- If we convolve the signal with S_x in a region where it falls, we obtain a **high negative value**

Canny Edge Detector: Gradient Calculation

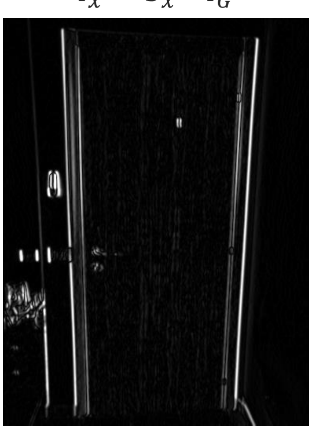
- Suppose to have a mono-dimensional discrete signal and a Sobel filter
  $$S_x = [-2 \quad 0 \quad 2]$$
- If we convolve the signal with S_x in a flat region, we obtain a **very small value** (near to 0)
- If we convolve the signal with S_x in a region where it grows, we obtain a **high positive value**
- If we convolve the signal with S_x in a region where it falls, we obtain a **high negative value**



Canny Edge Detector: Gradient Calculation

By applying the Sobel kernels (S_x and S_y) to our image I_G, we obtain:

$I_x = S_x * I_G$

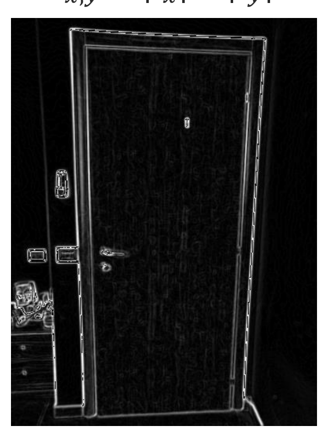


$|I_x|$

$I_y = S_y * I_G$

$|I_y|$

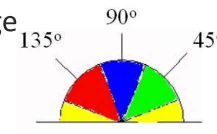
$I_{x,y} = |I_x| + |I_y|$

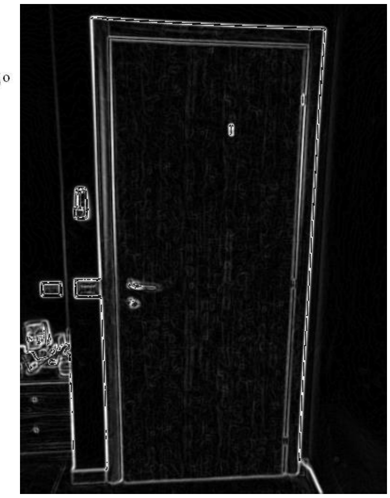


Canny Edge Detector: Non-Maximum Suppression

Suppress the non-maximum gradients in the edge directions -> thin edges

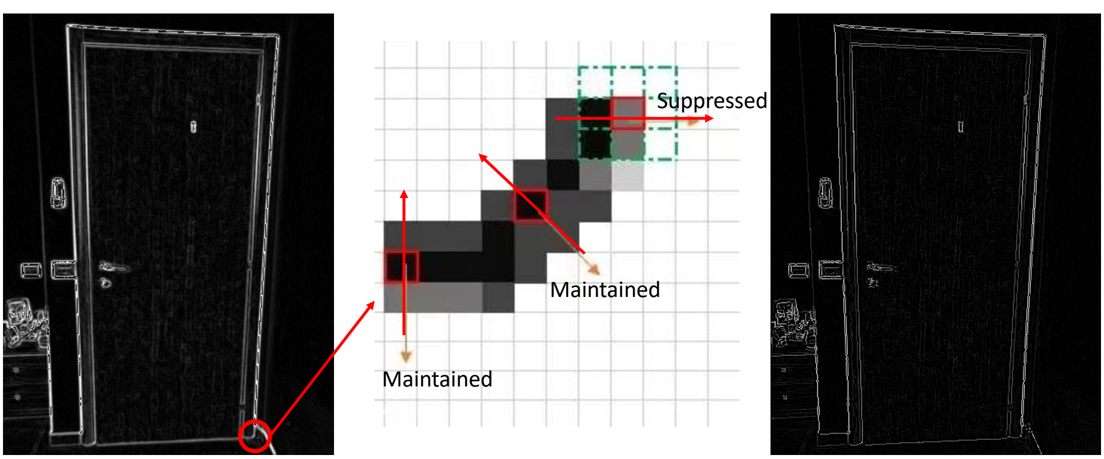
- **The gradient magnitude:** $G = \sqrt{I_x^2 + I_y^2}$
- **The direction of the edge:** $\theta = \text{atan2}(I_y, I_x)$
  Rounded to 0°, 45°, 90°, and 135°





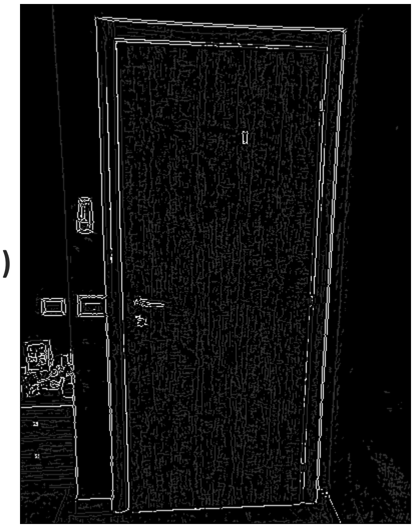
Then, every gradient is compared to the others in the direction θ to be:
- **Maintained:** if it is the maximum
- **Suppressed:** otherwise

Canny Edge Detector: Non-Maximum Suppression



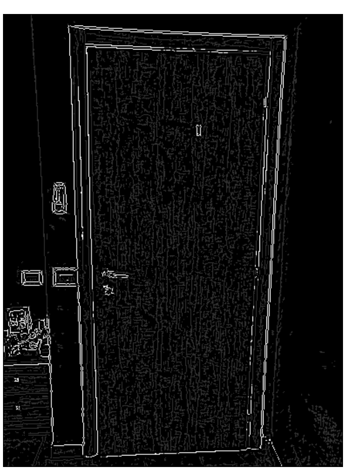
Canny Edge Detector: Double Threshold

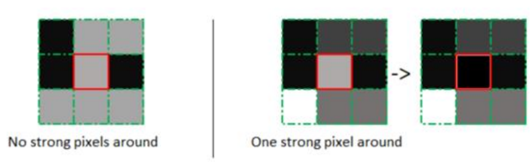
Group the gradients in 3 categories:
- **Strong:** high magnitude $\mathrm{G(x,y)} \geq T_h$ **(assumed to be edges)**
- **Non relevant:** low magnitude value $\mathrm{G(x,y)} \leq T_l$ **(suppressed)**
- **Weak:** in the middle (filtered in the last phase)

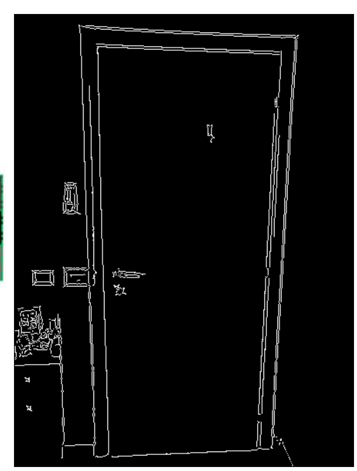


Canny Edge Detector: Edge Tracking

The last phase consists of **transforming weak pixels** into **strong ones**, if and only if at least one in the surrounding is strong

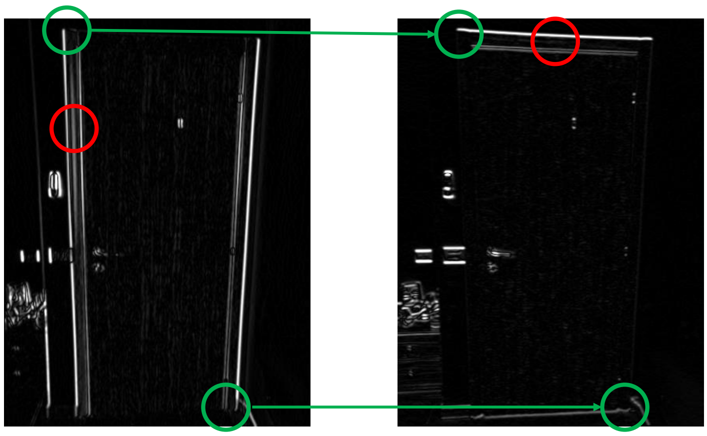


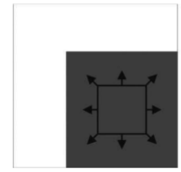




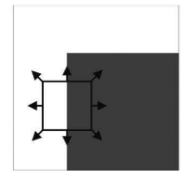
Corner Detector

Considering the gradients found in the Canny algorithm, the corners are located in those pixels where the gradients are far from zero in both x and y directions

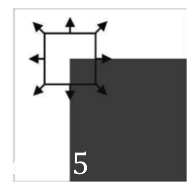




(a) Flat region



(b) Edge



(c) Corner

Harris Corner Detector

The Harris algorithm to detect corners works in the same way as Canny:

1. Noise reduction
2. Gradient calculation
3. Harris response calculation
4. Non-maximum suppression

I

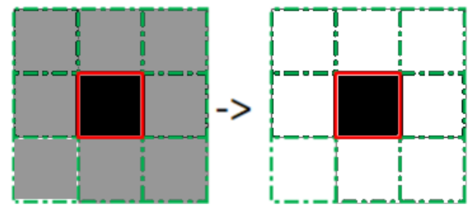
I_g

$|I_x|$

$|I_y|$

Harris Corner Detector

Non-maximum suppression



Calculate the Harris response:

- Given I_x and I_y, we build a 2x2 matrix associated

$$M = \begin{bmatrix} I_{x^2} & I_{xy} \\ I_{xy} & I_{y^2} \end{bmatrix}$$

- Given M, we calculate a value C

$$C = \det(M) - k * trace^2(M)$$

where

$$\det(M) = (I_{x^2} I_{y^2}) - (I_{xy} I_{xy})$$
$$trace^2(M) = (I_{x^2} + I_{y^2})^2$$
$$k \text{ is a constant empirically determined } \cong [0.04, 0.06]$$

The last step consists in extracting the real corners using non-maximum suppression

Harris Corner Detector

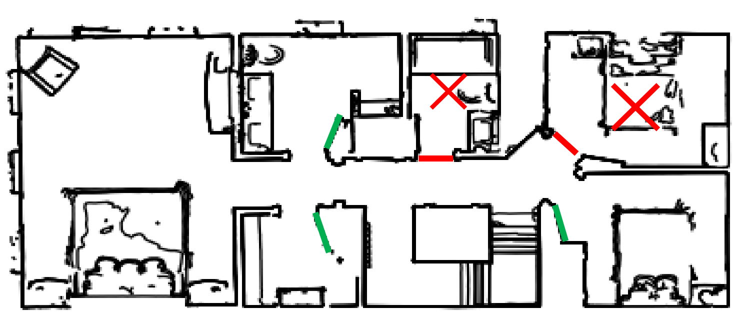
Outline

- What is Robotic Vision?
- Feature based methods
- **<u>A feature based method for Door Detection</u>**
- Deep Learning in Computer Vision (CNN)
- Deep Learning in RV for Door Detection
- A CNN for image classification: practical example

Door Detection in Mobile Robotics

Doors represent high-level features of an environment that can help robots to perform better its task

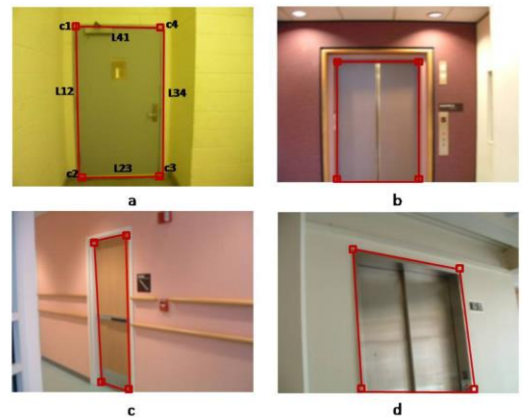
- **Doors represent dynamic obstacles**, that change the topology of the environment in which a robot operates
- Information about doors (such as location and status) can help robots to better perform their main tasks:
  - Mapping
  - Planning
  - Navigation



A Feature-based Door Detection Method

In [1], a feature-based method to detect doors is presented. To perform door detection, this method:

1. Extracts corners and edges from images
2. Aggregates these features to build the geometric model of a door, which is composed by 4 corners connected by 4 edges



[1] Yang, Tian, "*Robust Door Detection in Unfamiliar Environments by Combining Edge and Corner Features*", 2010

Feature Extraction

From [Yang, Tian, "*Robust Door Detection in Unfamiliar Environments by Combining Edge and Corner Features*", 2010]

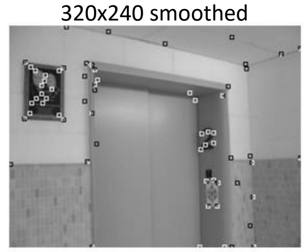
Feature extraction steps:

1. The image is scaled to be smaller (320x240) to reduce the number of features
2. The images is then smoothed with a Gaussian filter for denoising purposes
3. The image is elaborated with the algorithms of Harris and Canny to detect corners and edges
4. The contour of the image are considered edges and endpoints of open contours are also considered as corner, in order to detect partially occluded doors

640x480 non-smoothed

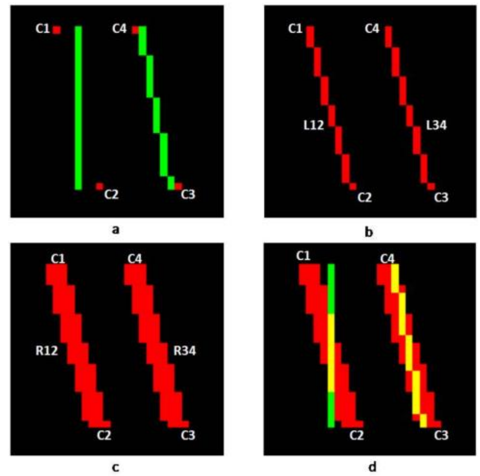
640x480 smoothed

320x240 smoothed



Geometric Model: Combining Edges and Corners

- Verify if the imaginary lines between corners match with a real edges found with the Canny algorithm
- For each corner group, each line is processed as follow:
  - **Fig. a** represents 4 candidate corners and 2 edges found woth Canny
  - **Fig. b** shows the imaginary lines that connect the corners
  - Each line is augmented with a mask (**Fig. c**)
  - If and only if the real edge is included in the mask, the line is considered valid (**Fig. d**)
- A 4 corner group with valid edges is considered a door

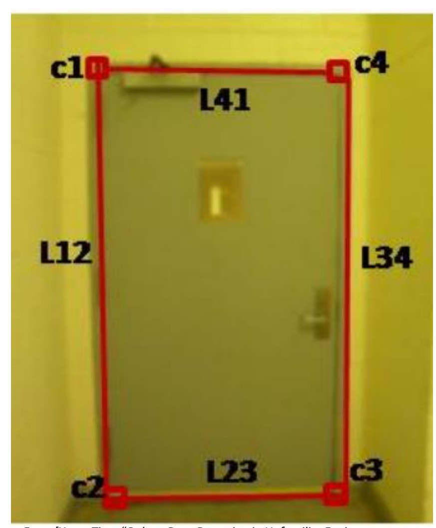


From [Yang, Tian, "*Robust Door Detection in Unfamiliar Environments by Combining Edge and Corner Features*", 2010]

Geometric Model: Candidates Filtering

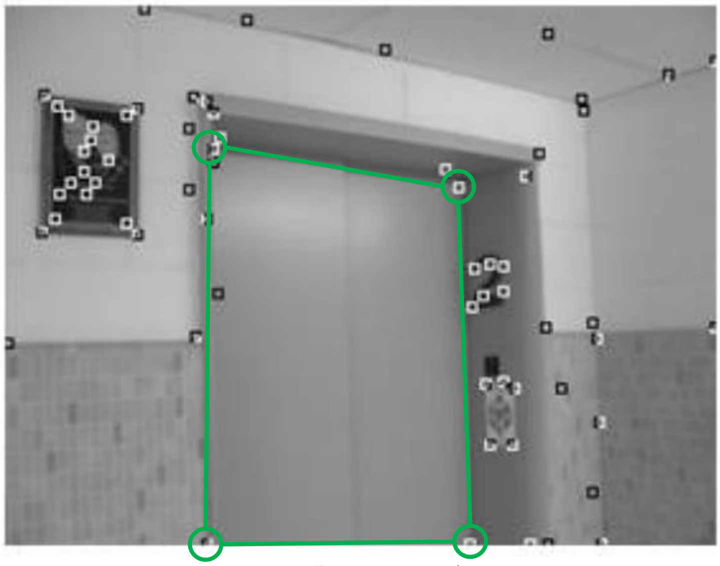
- The geometric model of a door consists of 4 corners $[C_1, C_2, C_3, C_4]$ connected by imaginary four lines $[L_{12}, L_{23}, L_{34}, L_{41}]$
- To reduce the total amount of the 4-corner groups, they are filtered according their relative geometric relationships:
  - The width and height of the lines with respect the image's dimensions
  - The horizontal lines $[L_{23}, L_{41}]$ should be parallel to the horizontal axis of the image
  - The vertical lines $[L_{12}, L_{34}]$ should be almost perpendicular with the horizontal axis of the image
  - The vertical lines $[L_{12}, L_{34}]$ should be parallel

All these constraints are determined by carefully tuned hyperparameters

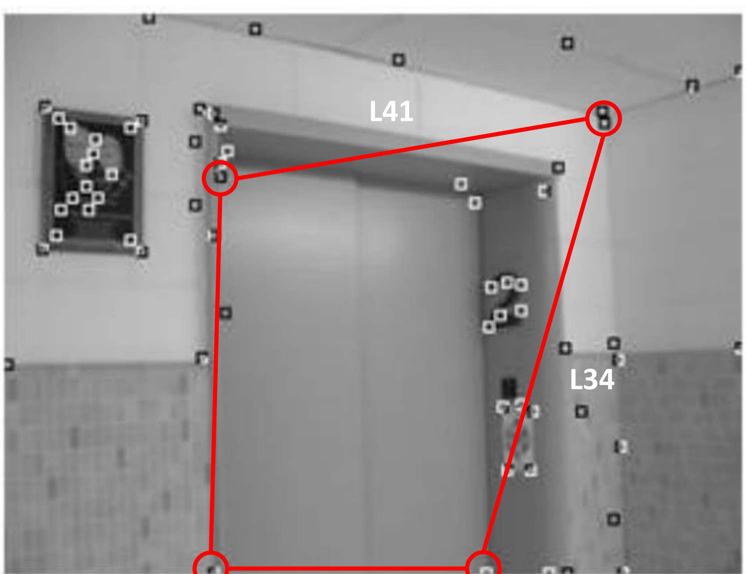


From [Yang, Tian, "*Robust Door Detection in Unfamiliar Environments by Combining Edge and Corner Features*", 2010]

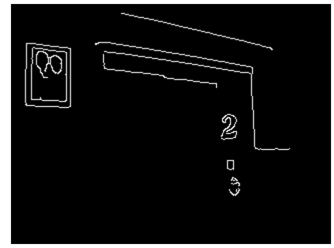
Geometric Model: Door-Corner Candidates Filtering

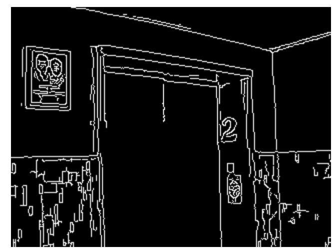


Geometric Model: Door-Corner Candidates Filtering



Limitations of Feature-based Methods

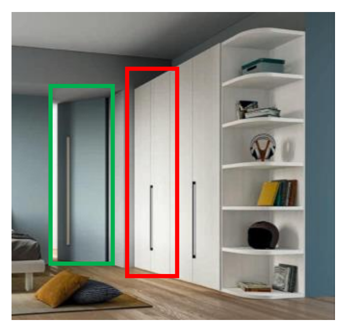




Feature extraction methods (Gaussian filtering, Canny, Harris, …)

- A robot operates in unpredictable conditions
- These techniques are strongly parametrized:
  - The characteristics of the camera (resolution, dimension of the images, noise, calibration, etc)
  - The illumination conditions (not static)

Limitations of Feature-based Methods



- The second limit regards the aggregation of the features in geometric models
  - Feature extraction failures
  - The aggregation of features could be very difficult to model: a door have a relatively simple geometric shape, but a face?
  - Multiple objects share the same shape
  - Variable geometric shape (different viewpoints)

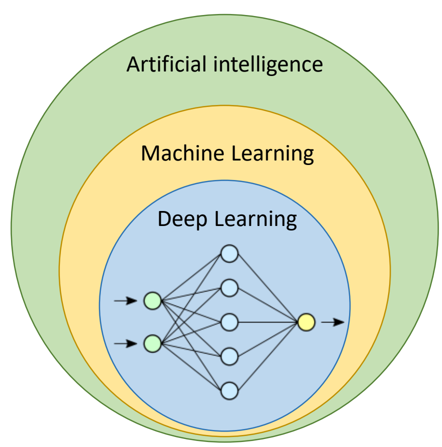
Outline

- What is Robotic Vision?
- Feature based methods
- A feature based method for Door Detection
- **<u>Deep Learning in Computer Vision (CNN)</u>**
- Deep Learning in RV for Door Detection
- A CNN for image classification: practical example

Computer Vision with Deep Learning

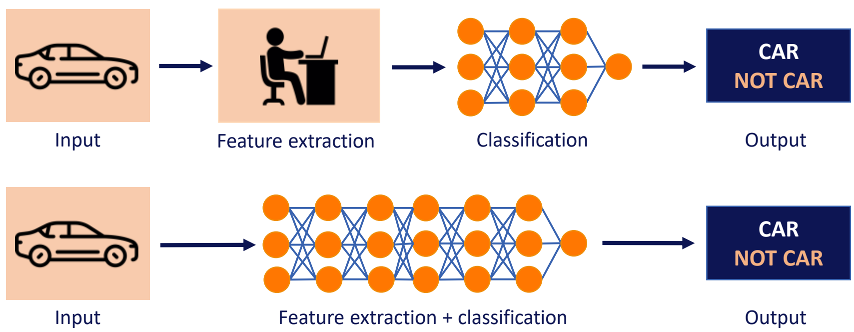
A modern approach in CV is to use **Deep Learning:**
- Based on the use of **Deep Neural Network**

Why Deep Learning?
- Neural networks compute non-linear functions $f: R^d \rightarrow R^n$



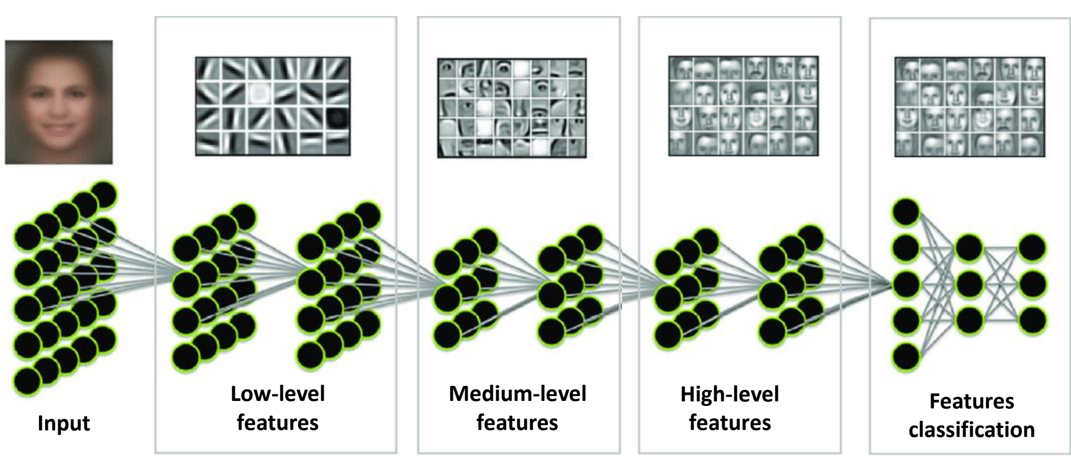
Computer Vision with Deep Learning

Deep Learning is a completely **end-to-end** approach:
- **In CV**, raw images do not represent strong features (require feature extraction)
- **Deep Neural networks** autonomously learns how to extract useful features to solve a specific task (or a dataset), eliminating humans in the loop



Feature Extraction

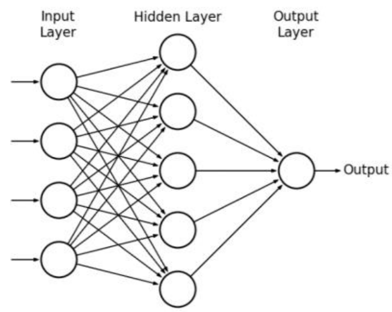
Hierarchical feature extraction (from low to high level).



Deep Learning In CV

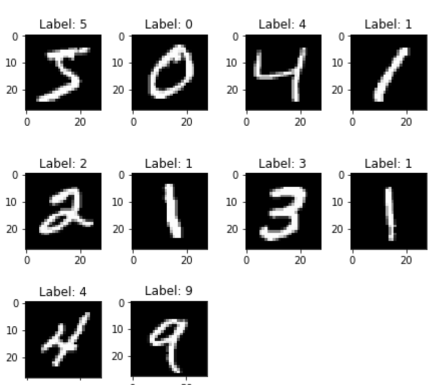
- **Naïve approach:** classify linearized images with MLP
- MLP are not suitable for treating images because they:
  - Are not suitable for larger inputs such as images
  - Pixels are not relevant features
  - Do not consider the **spatiality of images**

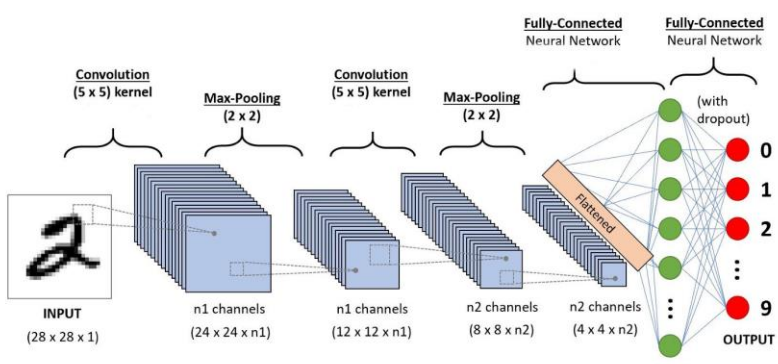
?



Convolutional Neural Networks

Convolutional Neural Network (CNN), composed of:
- **Convolutional Layer**
- **Pooling Layer**
- **MLP**



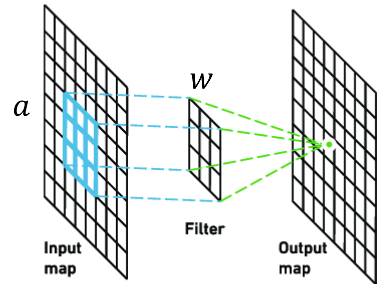


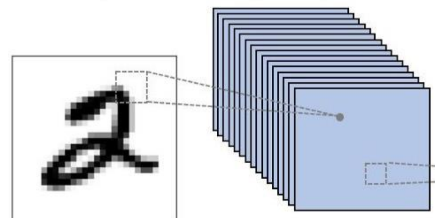
Convolutional Layer

- **Parameters**: a set of learnable kernels (or filters)
- Each **neuron** computes a single convolution

$$\sigma(a \cdot w) = \sigma\left(\sum_{i=0}^{n} a_i * w_i\right)$$

  - σ is the activation function
  - a is the sub-portion of the input
  - w is the kernel
- The **outputs** are 2-dimensional feature maps, one for each kernel, that are stacked together





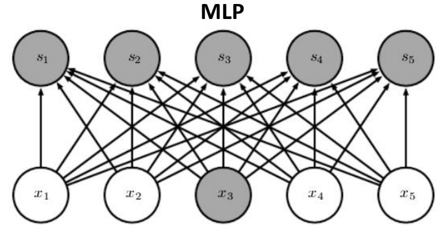
Stacked feature maps

Benefits of Convolutions – Sparse Interactions

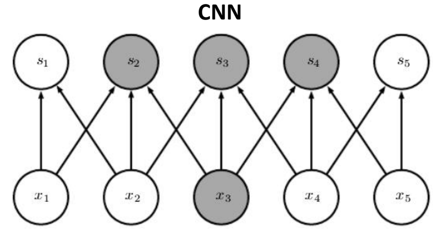
MLP: Each neurons in a layer interacts with all the neurons of the next layer

CNN: sparse interactions
- In images, meaningful features are small and well-localized
- This is useful reduces the **memory** to store the model and the **number of operations** to compute the output

MLP



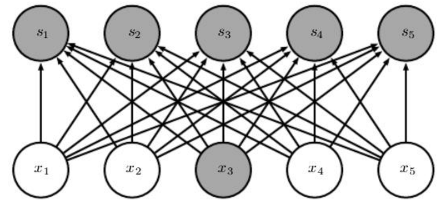
CNN



From [“Deep Learning” Goodfellow, Bengio, Courville]

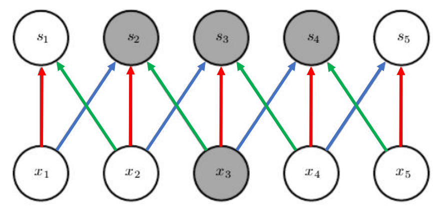
Benefits of Convolutions – Parameter Sharing

MLP



MLP: Each parameter is used exactly once and never revisited

CNN



$w = [w_1, w_2, w_3]$

CNN: Weights are tied between neurons:

- Unique set of shared parameters
- This is useful for find the **same features** all over the input

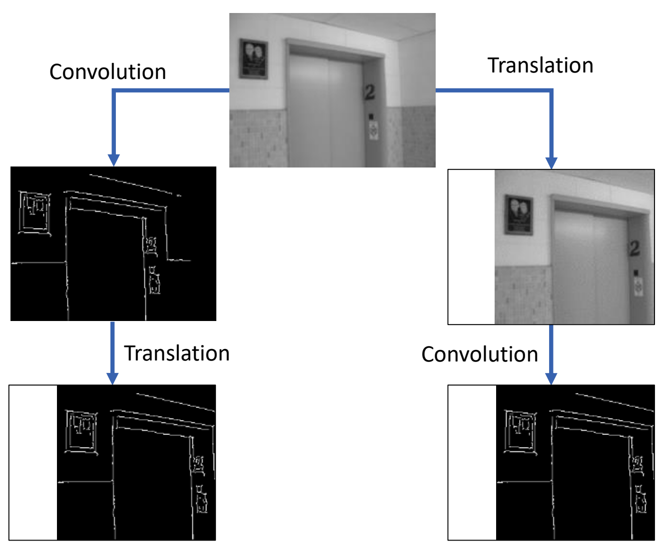
Benefits of Convolutions – Equivariance to Translation

A convolutional layer is **equivatiant to translation**

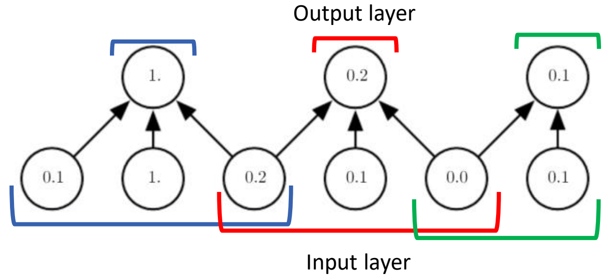
- f and g are equivariant when:

$$f(g(x)) = g(f(x))$$

- $g: I \rightarrow I'$ translation
- f (extract features) is equivariant to g
- This means that, given a feature in I, if we translate the image obtaining I', that feature will move by the same amount



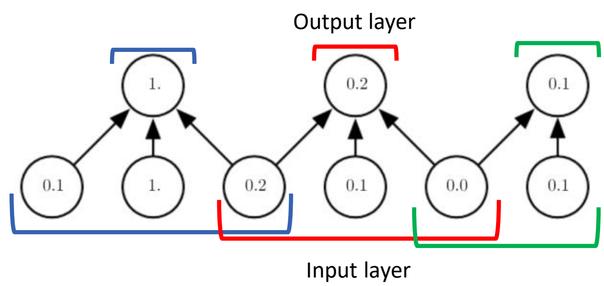
Pooling Layer

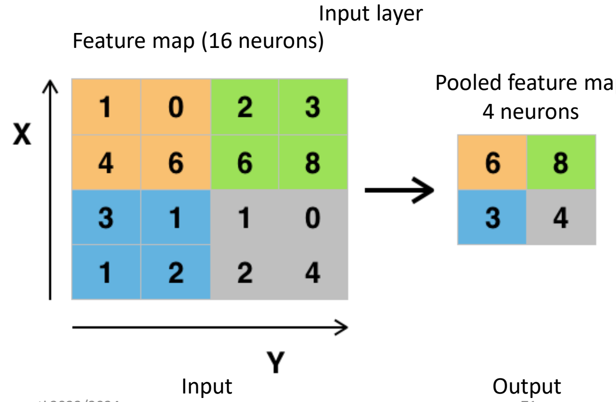


A pooling layer = non-linear down-sampling**:**
- Reduces dimensionality
- Multiple neuron to single neuron
- Divide the input in (overlapped) equally sized windows
- Each window is associated with a neuron:
  - **Max pooling:** highest value
  - **Average pooling:** average
  - **L_2 norm:** L_2 norm of the window (sum of the squared values)

Pooling Layer

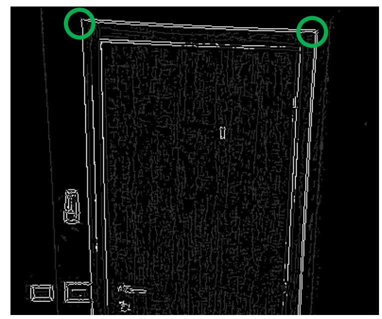
A pooling layer = non-linear down-sampling**:**
- Reduces dimensionality
- Multiple neuron to single neuron
- Divide the input in (overlapped) equally sized windows
- Each window is associated with a neuron:
  - **Max pooling:** highest value
  - **Average pooling:** average
  - **L_2 norm:** L_2 norm of the window (sum of the squared values)





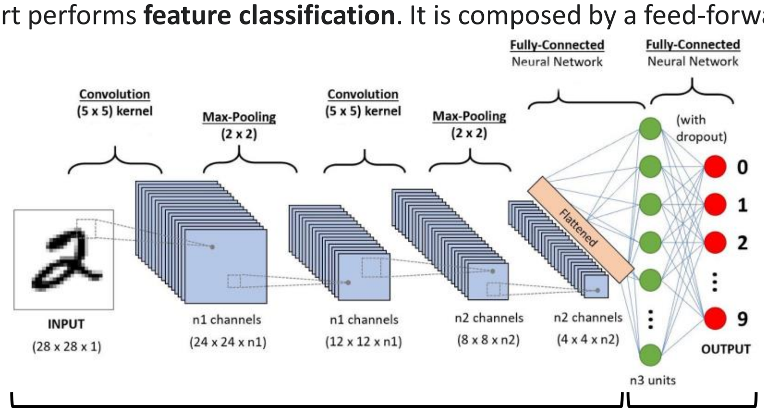
Pooling Layer Motivations

During feature extraction, intuitively:
1. Not all feature are important (suppression in Sobel fintering)
2. What matters is the rough location wrt to other features rather that the exact position



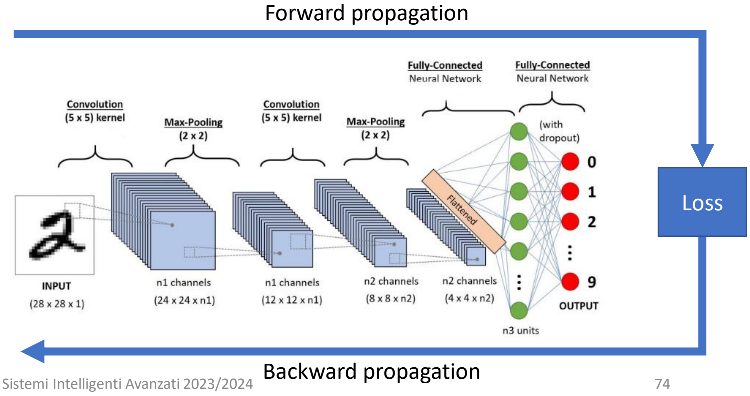
A Convolution Neural Network

A CNN is composed of two parts that perform:
1. the first part performs **feature extraction**. It is composed by consecutive convolutional and pooling layers
2. The second part performs **feature classification**. It is composed by a feed-forward network (MLP)



The Training of a CNN

Training:
- A **loss function** $l(y, \hat{y})$
- A **forward propagation step** to evaluate the loss of the CNN
- A **backward propagation step** with gradient descent to adjust the learnable parameters:
  - The kernels of the convolutional layers
  - The weights of the MLP



Outline

- What is Robotic Vision?
- Feature based methods
- A feature based method for Door Detection
- Deep Learning in Computer Vision (CNN)
- **Deep Learning in RV for Door Detection**
- A CNN for image classification: practical example

Door Status Detection with Deep Learning

Images from the robot POV

Images from an existing dataset

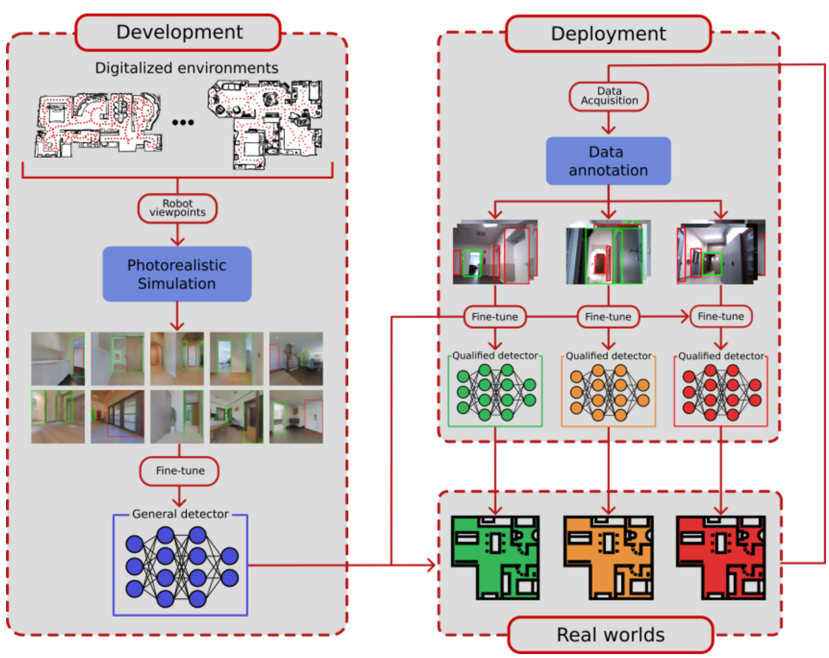
Inside our laboratory, we are working on these challenges considering the task of **Door Status Detection** [*]. In particular, we focus on the following issues:

1. Existing visual datasets do not represent the challenging point of view of mobile robots
2. When a robot is deployed in an unknown environment, the performance of an end-to-end model degrade

[*] Antonazzi, Basilico, Luperto, Borghese "Enhancing Door-Status Detection for Autonomous Mobile Robots during Environment-Specific Operational Use"

Method

1. **Deployment**: build a **general detector** (GD) trained in simulation
2. **Deployment**: adapt the GD for the target environment of the robot, obtaining a **qualified detector** (QD)



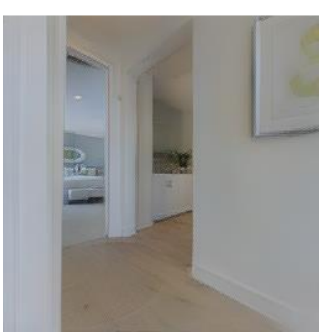
Dataset Collection

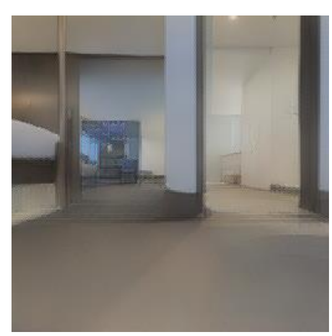
Dataset characteristics:
- Huge amount of images
- From different environment
- From the robot

How to acquire a dataset?
- **Real world:** best solution but impractical (time consuming, data annotations, …)
- **Simulation using game engines:** allow to capture a wide amount of annotated image but
  - How to create realistic environment?
  - The images are not photorealistic
- **Photorealistic simulation:** frameworks that virtualize environments scanned from the real world

Dataset Collection

Dataset collected using **Gibson Env:**
- About 5k images
- From 10 different environments
- **Used for training the general detector**

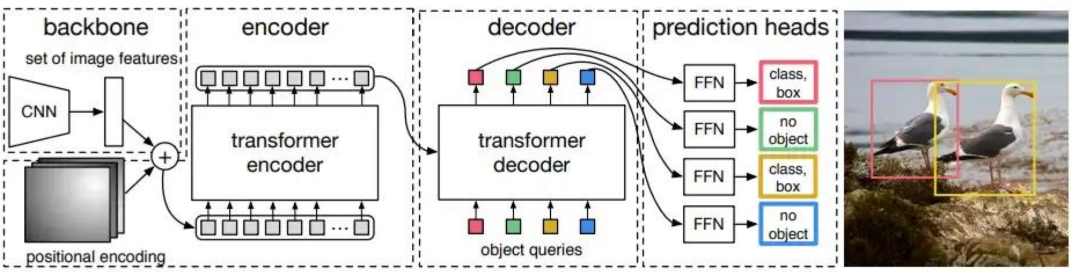




General Detector

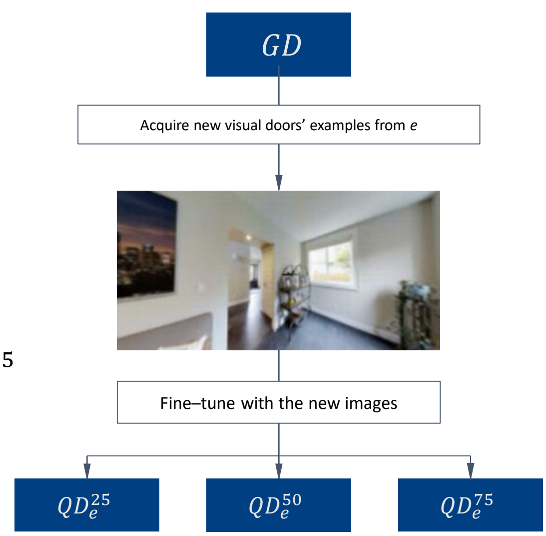
We build the GD by **fine-tune** DETR (Detection Transformer). Its architecture is composed of:

- **CNN:** a deep convolutional networks used for feature extraction
- **Transformer:** a modern deep architecture (initially developed for natural language process) which is able to find complex relationships between a sequence of items (in this case, the features of an image)
- **MLP:** which classifies the output of the transformer to find the bounding boxes and their labels



Qualified Detector

- We fine-tune the GD with new data from the target environment of the robot, obtaining a **Qualified Detector**
- During the robot deployment, we
  1. Qualified GD_e with 3 fine-tune operations using the 25%, 50%, and 75% of the images collected in *e*, obtaining: QD_e^{25}, QD_e^{50}, and QD_e^{75}
  2. The last 25% of images collected in *e* are used for testing

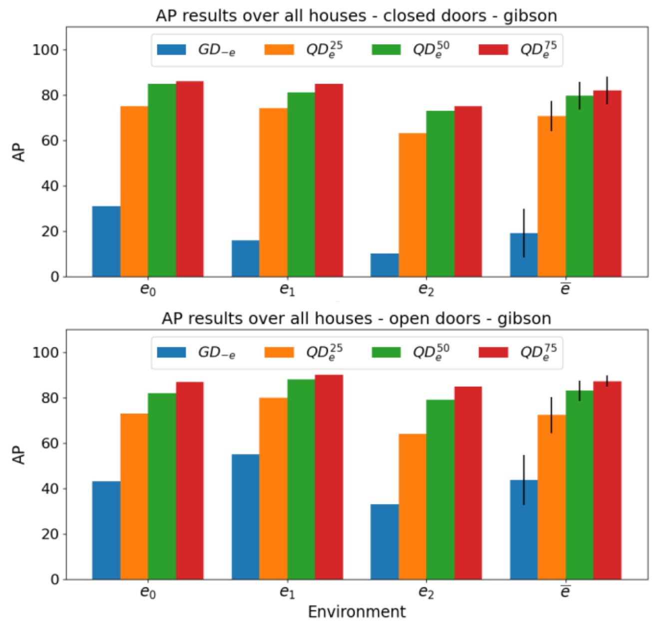


Results in the Real World

The performance are measured using the AP (average precision) metric, which is computed over the two different object categories: closed and open doors

Average results over all environments

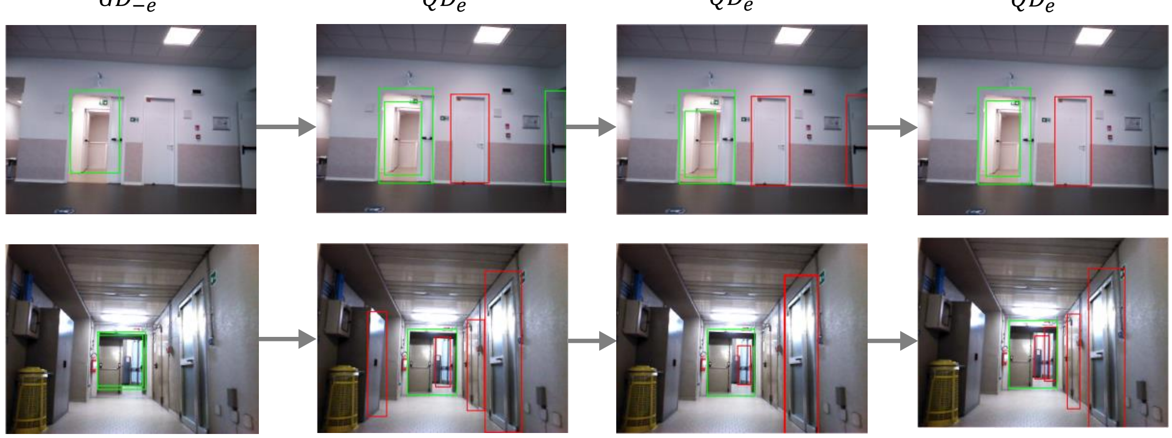
Exp.	Label	AP	Inc.
GD_{-e}	Closed	15	-
	Open	39	-
QD_e^{25}	Closed	63	484%
	Open	67	74%
QD_e^{50}	Closed	74	17%
	Open	78	19%
QD_e^{75}	Closed	78	5%
	Open	85	1%



Results in the Real world

The detection accuracy increases with consecutive fine–tune operations

GD_{-e} QD_e^{25} QD_e^{50} QD_e^{75}

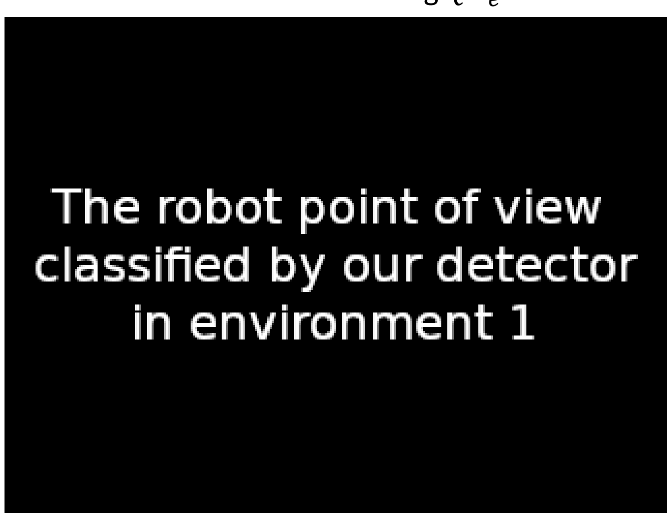


Results in the Real World

Fine–tuning with only the 25% of new images is enough to classify challenging examples

Results in Simulation

The robot's POV using QD_e^{75}



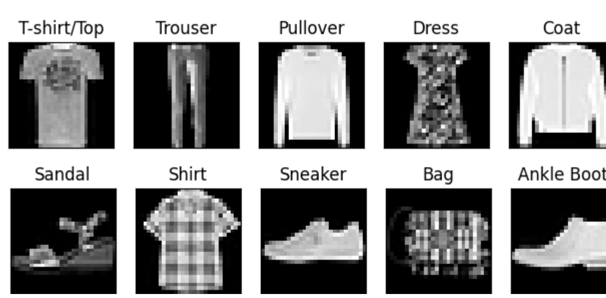
Outline

- What is Robotic Vision?
- Feature based methods
- A feature based method for Door Detection
- Deep Learning in Computer Vision (CNN)
- Deep Learning in RV for Door Detection
- <u>**A CNN for image classification: practical example**</u>

A CNN at Work on Image Classification

In this example, we will perform image classification on the dataset **FashionMNIST**:
- It is a dataset of 70k images 28x28
- The images depict 10 different categories of clothes

This example is implemented using **PyTorch**:
- An open-source framework for Machine Learning
- Particularly used for the implementation of Deep Neural Network
- Available in Python, C++, Java



Dataset Preparation

Download of the train and test sets (60k + 10k images)

Split the test set in validation and test sets (1k + 9k images)

Set the PyTorch classes to load the datasets from disk

```python
# Download del dataset FashionMnist direttamente da PyTorch (train)
fashion_mnist_train = torchvision.datasets.FashionMNIST('/files/', train=True, download=True,
                                transform=torchvision.transforms.Compose([
                                    torchvision.transforms.ToTensor(),
                                ]))
# Download del dataset FashionMnist direttamente da PyTorch (test)
fashion_mnist_test = torchvision.datasets.FashionMNIST('/files/', train=False, download=True,
                                transform=torchvision.transforms.Compose([
                                    torchvision.transforms.ToTensor(),
                                ]))

# Split del test set in validation e test set
len_validation_set = int(0.1 * len(fashion_mnist_test))
len_test_set = int(0.9 * len(fashion_mnist_test))
validation, test = torch.utils.data.random_split(fashion_mnist_test, [len_validation_set, len_test_set])

# Creazione delle classi PyTorch per il caricamento del dataset da disco fisso
train_loader = torch.utils.data.DataLoader(fashion_mnist_train, batch_size=batch_size_train, shuffle=False)
validation_loader = torch.utils.data.DataLoader(validation, batch_size=batch_size_validation, shuffle=False)
test_loader = torch.utils.data.DataLoader(test, batch_size=batch_size_test, shuffle=False)
```

Sistemi Intelligenti Avanzati 2023/2024 104

Model Setting

Definition of the CNN's (trainable) layers

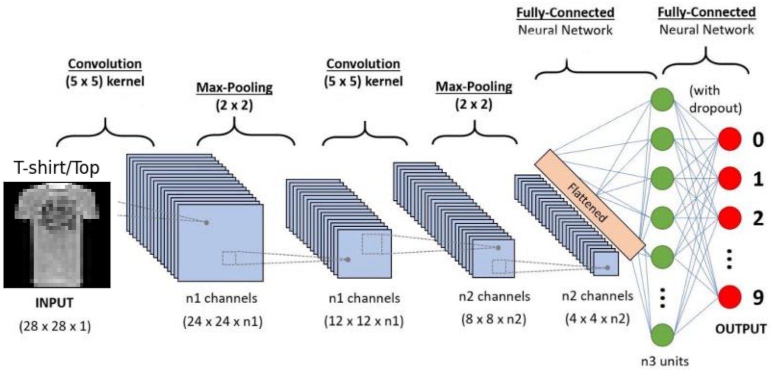
Definition of the forward propagation

Creation of the model and the optimizer (SGD)

```python
# Classe che implementa la CNN che è formata da:
#   - layer convoluzionale
#   - max-pooling
#   - layer convoluzionale
#   - max-pooling
#   - MLP di due layers
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2(x), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x)

# Creazione del modello
network = Net()

# Creazione dell'optimizer: stochastic gradient descent
optimizer = optim.SGD(network.parameters(), lr=learning_rate,
                      momentum=momentum)
```

Sistemi Intelligenti Avanzati 2023/2024 105

Assigning a Label to an Image



- The last layer of the model is composed by 10 neurons, to it returns an array $X = [x_0, \dots, x_9]$
- The activation function of the last layer is a log softmax, computer for each element of X

$$\text{logsoftmax}(x_i) = \log\left(\frac{e^{x_i}}{\sum_{j=0}^{i} e^{x_j}}\right)$$

- It normalizes the values of X between [0, 1] (**probabilities**)
- The position i with the highest value is considered the label assigned to the image by the model

```
labels = {
    0: "T-shirt/Top", 1: "Trouser", 2: "Pullover", 3: "Dress", 4: "Coat", 5: "Sandal", 6: "Shirt", 7: "Sneaker", 8: "Bag", 9: "Ankle Boot"
}
```


Training Phase

The model is set in training mode

Training cycle over batches

Training steps:
- Reset of the optimizer
- Forward propagation: the data in the batch are classified by the model
- Loss calculation
- Loss backpropagation
- Weights adjustment

```
# Funzione di training
def train(epoch):

    # Settaggio del modello in training mode
    network.train()

    # Ciclo for che itera sulle batch del training set
    for batch_idx, (data, target) in tqdm(enumerate(train_loader),
                                          total=len(train_loader), desc=f"Train epoch {epoch}",
                                          dynamic_ncols=True):

        # Reset dell'optimizer
        optimizer.zero_grad()

        # Forward propagation: tutti i dati del batch vengono classificati dal modello
        output = network(data)

        # Valutazione della loss function: negative log likelihood loss, già implementata in PyTorch
        # L'oggetto accetta come input l'output del modello (output) e le label ground truth (target)
        loss = F.nll_loss(output, target)

        # Backward propagation: l'errore della loss viene propagato all'indietro nella CNN
        loss.backward()

        # L'ottimizzatore (SGD) traina i pesi
        optimizer.step()
```

Loss Function

The loss function is the Negative Log Likelihood

- After the loss function application, the last layer returns a vector X that describes the probabilities assigned by the model to the 10 possible classes
- This Negative Log Likelihood loss computes $-\ln(x_i)$ where i is the correct label

Examples:

- Wrong classification:
  - $X = [0.1, 0.3, 0.5, 0.1]$, but the correct label is 3
  - $-\ln(x_3) = -\ln(0.1) = 2.3$
- Correct classification 1:
  - $X = [0.1, 0.3, 0.5, 0.1]$ and the correct label is 2
  - $-\ln(x_2) = -\ln(0.5) = 0.69$
- Correct classification 2:
  - $X = [0.1, 0.1, 0.7, 0.1]$ and the correct label is 2
  - $-\ln(x_2) = -\ln(0.7) = 0.35$

Testing Phase

The model is set in testing mode

Testing cycle

- Data classification
- Loss calculation
- Count of the correct predictions

Loss average and Accuracy calculation

```
def test(dataset):

    # Il modello viene messo in test mode
    network.eval()
    test_loss = 0
    correct = 0

    # Viene disabilitata l'ottimizzazione di PyTorch per il calcolo dei gradienti
    # che non serve durante il test
    with torch.no_grad():
        for data, target in dataset:
            output = network(data)
            test_loss += F.nll_loss(output, target, size_average=False).item()
            pred = output.data.max(1, keepdim=True)[1]
            correct += pred.eq(target.data.view_as(pred)).sum()

    # La loss, che inizialmente è la somma su tutti gli esempi,
    # viene mediata sul numero totale di esempi
    test_loss /= len(dataset.dataset)
    accuracy = 100. * correct / len(dataset.dataset)

    return test_loss, correct, accuracy
```

Model Training

In each epoch:

- The model is trained
- The error and the accuracy are calculated using the training set
- The error and the accuracy are calculated using the validation set

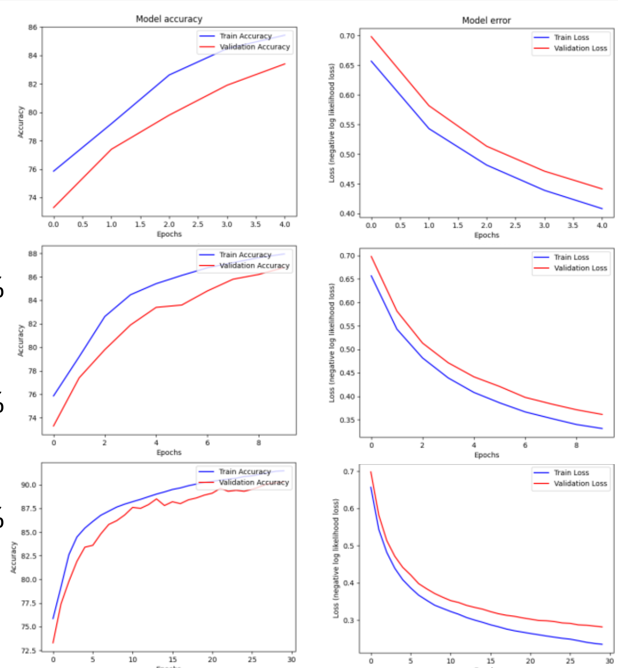
At the end, the model is tested with the test set

```
# Ciclo per trainare il modello sul numero di epoche
for epoch in range(n_epochs):
    # Train
    train(epoch)

    # Qui viene calcolato il training error
    loss, correct, accuracy = test(train_loader)
    print('\nEpoch: {} - Training error epoch: Avg. loss: {:.4f}, '
          'Accuracy: {}/{} ({:.0f}%)'.format(
        epoch, loss, correct, len(train_loader.dataset), accuracy))

    # Qui viene calcolato il validation error
    loss, correct, accuracy = test(validation_loader)
    print('Epoch: {} - Validation error epoch: Avg. loss: {:.4f}, '
          'Accuracy: {}/{} ({:.0f}%)'.format(
        epoch, loss, correct, len(validation_loader.dataset), accuracy))

loss, correct, accuracy = test(test_loader)
print('Test error: Avg. loss: {:.4f}, '
      'Accuracy: {}/{} ({:.0f}%)'.format(
    loss, correct, len(test_loader.dataset), accuracy))
```


Results

The CNN has been trained for 5, 10, and 30 epochs:

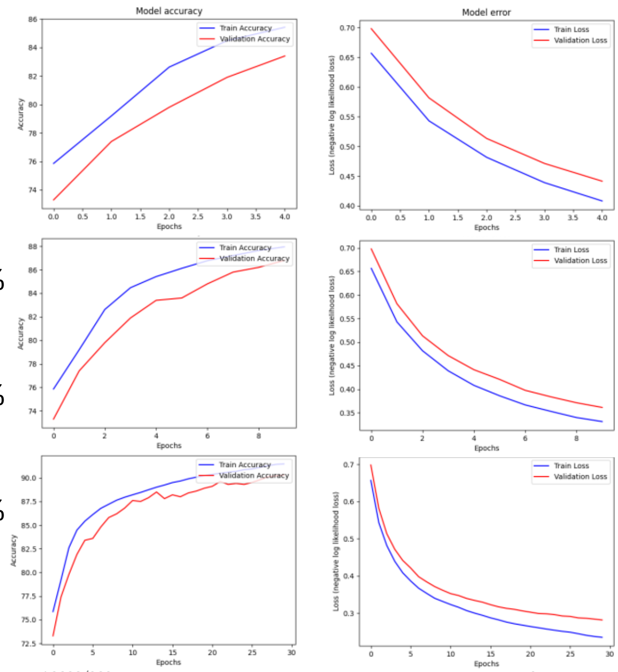
- **5 epochs**
  - **Accuracy:** training set = 85%, test set = 84%
  - **Loss:** training set = 0.40, test set = 0.43
- **10 epochs**
  - **Accuracy:** training set = 88%, test set = 87%
  - **Loss:** training set = 0.33, test set = 0.36
- **30 epochs**
  - **Accuracy:** training set = 91%, test set = 90%
  - **Loss:** training set = 0.23, test set = 0.30



Results

The CNN has been trained for 5, 10, and 30 epochs:

- **5 epochs**
  - **Accuracy:** training set = 85%, test set = 84%
  - **Loss:** training set = 0.40, test set = 0.43
- **10 epochs**
  - **Accuracy:** training set = 88%, test set = 87%
  - **Loss:** training set = 0.33, test set = 0.36
- **30 epochs**
  - **Accuracy:** training set = 91%, test set = 90%
  - **Loss:** training set = 0.23, test set = 0.30



Project: Fine-tune methods for deep learning modules

- Fine-tune is a promising paradigm to adapt neural networks to solve different or more refined tasks.
- A mobile robot can use this technique to specialize its onboard deep learning-based object detector to increase its performance in a specific environment.
- Despite this, the new examples must be accurately labelled. This task is extremely expensive if performed by a human technician.
- The goal is to study new approaches for selecting the most valuable data in order to maximize the impact of the fine-tune reducing the labelling effort