Visual scene real-time analysis for Intelligent Vehicles:

Objects visual detection and recognition / categorization

Pr. Fabien MOUTARDE
Center for Robotics,
MINES ParisTech
PSL Université Paris

Fabien.Moutarde@mines-paristech.fr
http://people.mines-paristech.fr/fabien.moutarde

Outline

- <u>Motivations: ADAS and autonomous driving</u>
- Objects visual *DETECTION*
- Objects visual *RECOGNITION*:
    - usual *features* used
    - Machine-Learning algorithms
- Traffic Sign Detection and Recognition (TSR)
- Cars & Pedestrians detection with adaBoost

A self-driving car is a mobile robot!

Perception → Reasoning Decision → Action → Perception

ROBOTIC LOOP

« Ingredients » of an Autonomous Vehicle

Robot ➔ perceive + reason + act

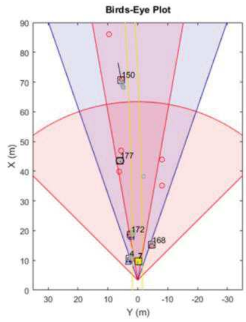
<u>An Autonomous Vehicle therefore needs:</u>

- Sensors
- « Intelligents » algorithms
  - for perception
  - for trajectory planning
  - for control
- Embedded calculator(s)
- Actuators (« drive by wire »)

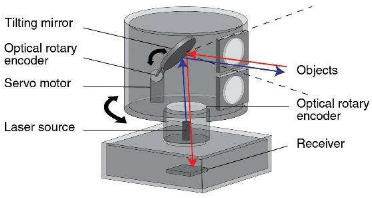
Sensors for Autonomous Vehicles

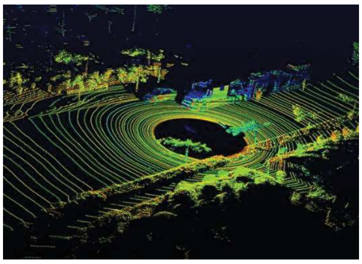
- **Classic cameras** ***[range ~500m, wide field of view]***

- **Radar(s)** ***[range ~200m, <u>narrow</u> field of view]***



- **LIDAR** ***[range ~100m, FoV from ~60° to 360°]***





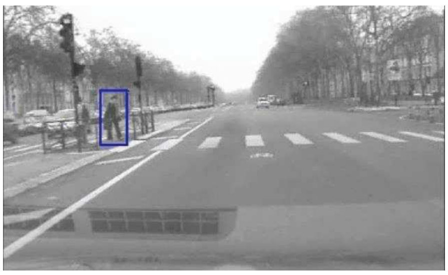
- **Ultrasound, etc…**

Examples of visual objects detection & recognition for IV

- **Traffic Sign detection and Recognition (TSR)**

- **Traffic Lights Detection**

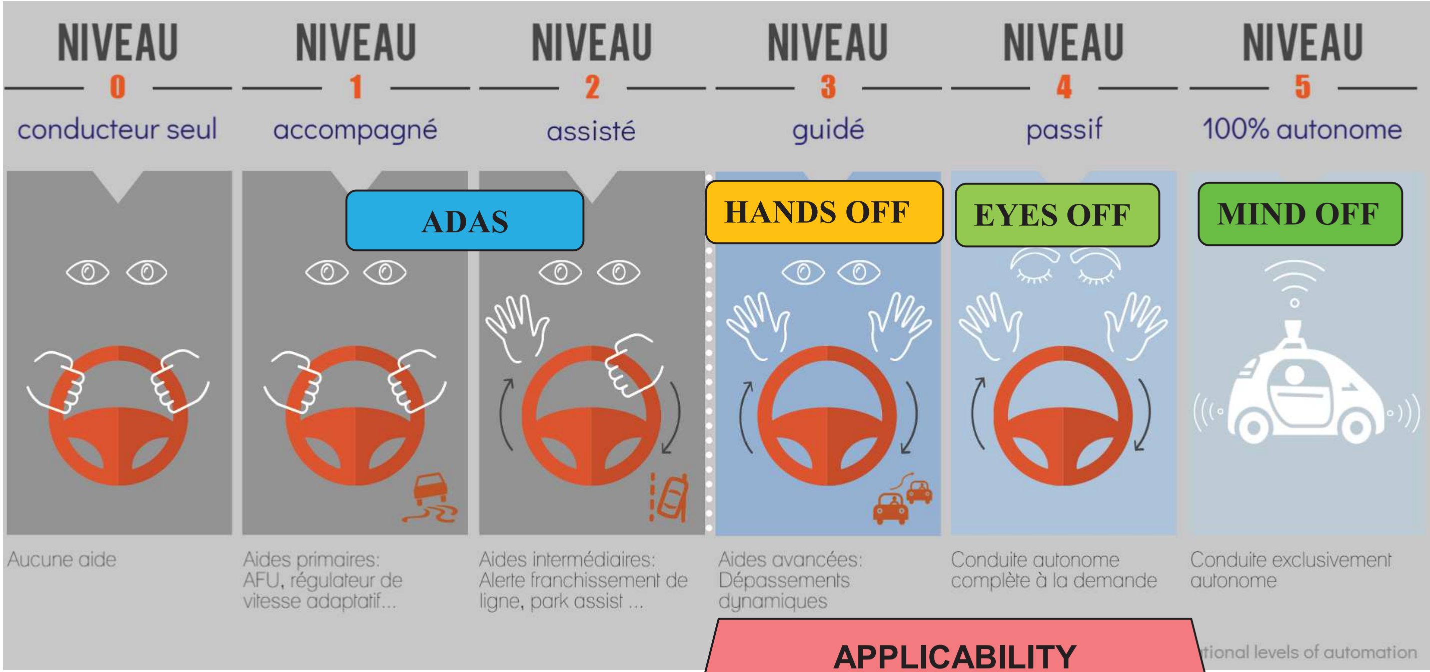
- **Cars and pedestrians visual detection**



All these videos = research conducted @ center for Robotics of MINES ParisTech

Partial, total or conditional autonomy?

The 5 « automation levels » for vehicles defined by SAE

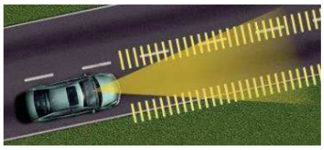


APPLICABILITY
CAN BE CONDITIONAL
(e.g. RESTRICTED TO ONLY
MOTORWAYS, …)

What are ADAS?

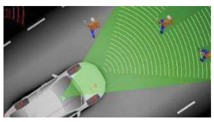
Acronym of Advanced Driving Assistance Systems = Intelligent functions for safer and/or easier driving

- ***Warning or Information***

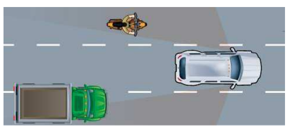


  - Lane Departure Warning (LDW)

  - Forward Collision Warning (FCW)



  - Pedestrian Collision Warning



  - Blind Spot Monitoring

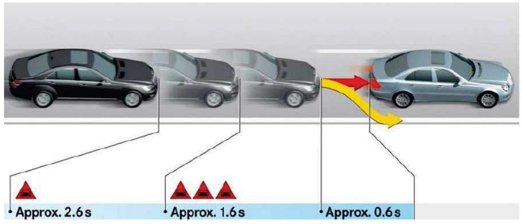
  - Speed Limit Assistant
  - Driver Attention Warning
  - Night vision
  - …

Active ADAS

- ***Active systems*** (ADAS that ACT on the vehicle, rather than just only warn the driver)
  - Adaptive Cruise Control (ACC)

  - Lane Keeping (LK)

  - Autonomous Emergency Braking

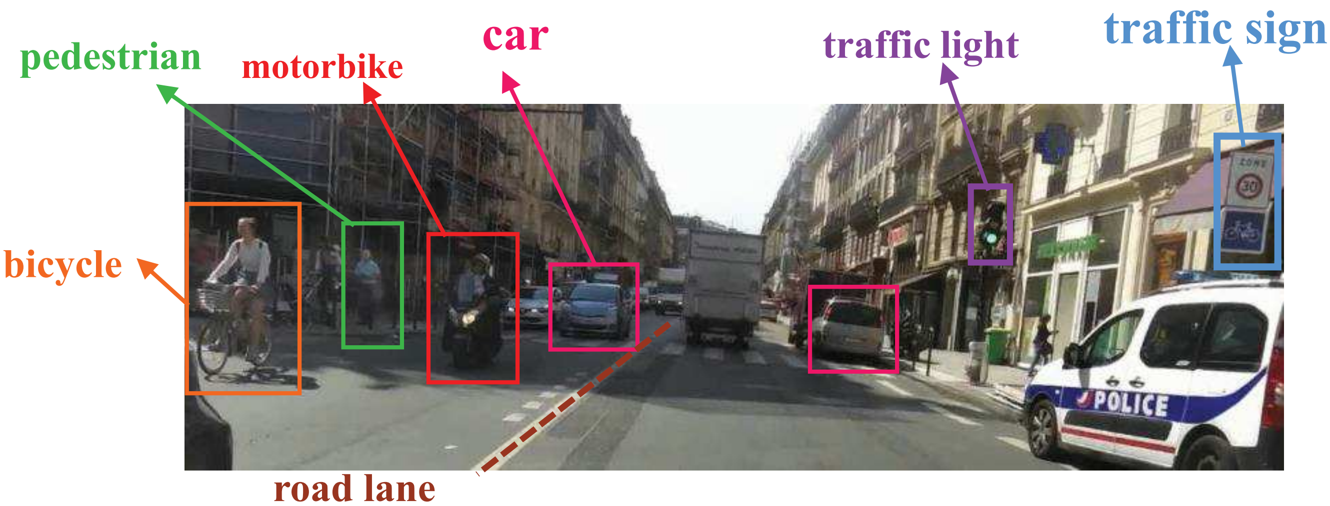


  - Automated Parking
  - …

More detailed information: see for instance https://mycardoeswhat.org/

Real-time visual scene understanding

Main goal = localize and categorize "objects"



Key componant for driving assistance (ADAS) & automated driving

Strong real-time constraint:
process at least ~20 frames/second

Objects to be detected and recognized

- Road lanes
- <u>Traffic signs</u>
- <u>Traffic lights</u>
- <u>Cars</u>, vans, trucks
- <u>Motorbikes</u>
- <u>Bicycles</u>
- <u>Pedestrians</u>
- etc…

Summary on *MOTIVATIONS*

- Intelligent functions for safer and/or easier driving are called *ADAS* (= *<u>A</u>dvanced <u>D</u>riving <u>A</u>ssistance <u>S</u>ystems*)
- There are several different types of ADAS, such as Forward Collision Warning (FCW), Blind Spot Monitoring (BSM), Lane Keeping, Adaptive Cruise Control (ACC), Automated Parking, etc
- Many of these ADAS, and automated driving, requires *real-time on-board analysis of video from cameras, in order to interpret ("understand") the visual scene*, and in particular to *detect and categorize* in the images *objects* such as: *cars, pedestrians, bicycles, motorbikes, traffic signs* and *traffic lights*

Outline

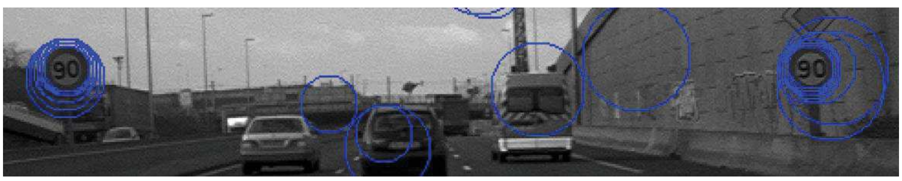
- **Motivations: ADAS and autonomous driving**
- **<u>Objects visual *DETECTION*</u>**
- **Objects visual *RECOGNITION*:**
  - **usual *features* used**
  - **Machine-Learning algorithms**
- **Traffic Sign Detection and Recognition (TSR)**
- **Cars & Pedestrians detection with adaBoost**

Objects visual <u>*DETECTION*</u>

For <u>*objects*</u>, visual scene analysis often performed in TWO (or three) STEPS:

Detection → **Recognition** ⇢ **Temporal Tracking** ⇢ **Detection**

<u>Detection = find WHERE in the image</u>
are (maybe) located interesting objects



Candidate locations for searched objects

Recognized objects

Objects visual *detection* approaches

Visual detection can be done using:

- Template matching
- Shape cues
- Color cues
- Window scanning *with classifier*
- Keypoints
- Segmentation

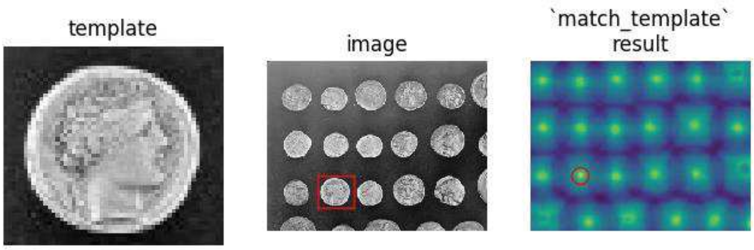
Objects visual *detection* by TEMPLATE MATCHING

Mostly for detection of nearly invariant patterns (like *traffic signs*)

- **Principle:** compare a reference image (template) of object with all possible positions/sizes (cross-correlation)

For each position compute a similarity measure (e.g. SAD)
→ « heatmap »

$$SAD(x, y) = \sum_{i=0}^{T_{\text{rows}}} \sum_{j=0}^{T_{\text{cols}}} \text{Diff}(x + i, y + j, i, j)$$



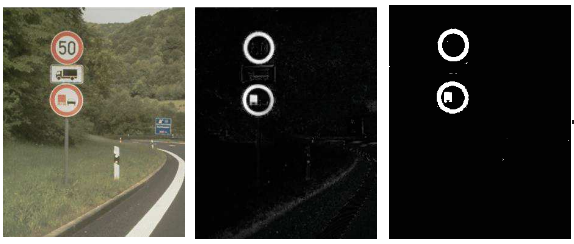
Problems: high computation time
+ handling of luminosity&contrast variations
+ handling of orientation variation, and of deformation

Objects visual *detection* by COLOR

For objects with standardized *(e.g. Traffic Signs)* or specific color *(e.g. skin)*

Principle: ≈ thresholding in color space

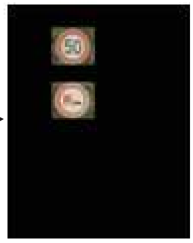
[color pixels usually coded as 3 intensities for the 3 primary colors Red, Green and Blue]



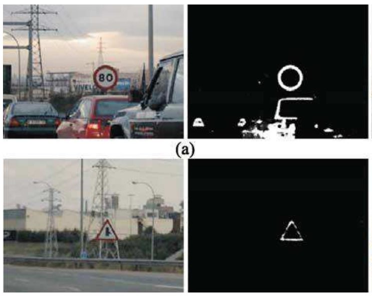
Red enhancement

Thresholding

→ Extract connected componants → Filter →



Candidate bounding boxes

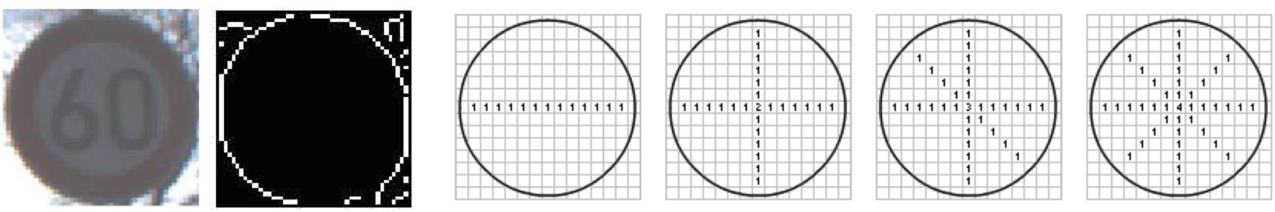


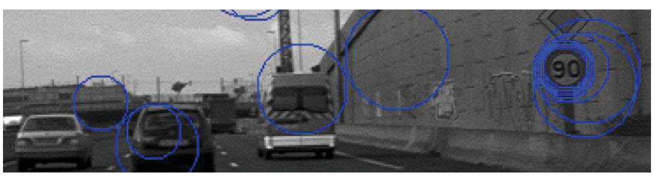
Problems:
- sometimes many parasite detections
- high variability of color appearance (especially in RGB!)

Objects visual *detection* by SHAPE

For objects with fixed and rather specific *shape*

- **Principle:**
  - General case: template-matching on *contours* image
  - For « simple » shapes (lines, circles, polygons like triangles, rectangles,…) efficiently feasible using Hough transform (center voting by Canny edges) or Radon transform



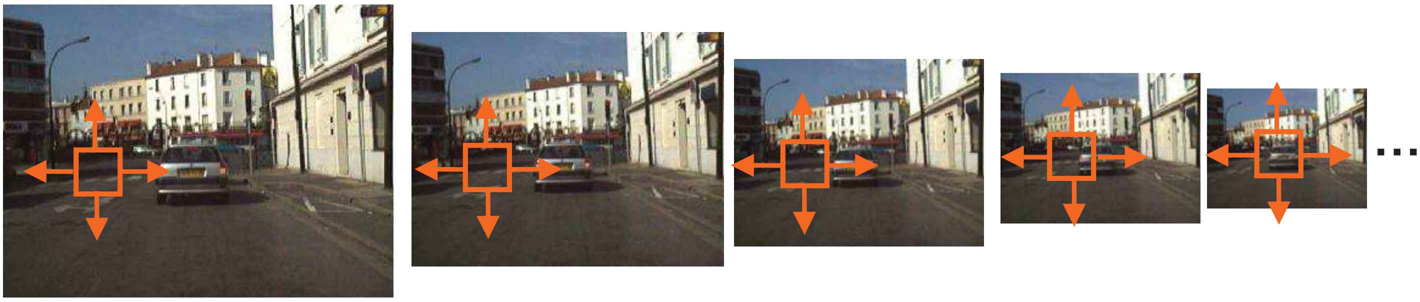


Problems:
- Rather computer-intensive
- Some shape are not so rare (rectangles!!)

Multi-scale detection by Window-scanning with classifier

Principle:

- Build a <u>pyramid</u> of *down-sampled* images
- Scan each level of pyramid with a <u>sliding fixed-size detection window</u> ➔ tens of thousand of sub-images



- Apply a single common classifier on all sub-images to determine if it is a bounding-box around searched object

Kind of Template-matching using classifier output as similarity measure

Objects visual *detection* by KEYPOINTS

<u>Keypoint =</u> « salient » point (e.g. corners, etc)

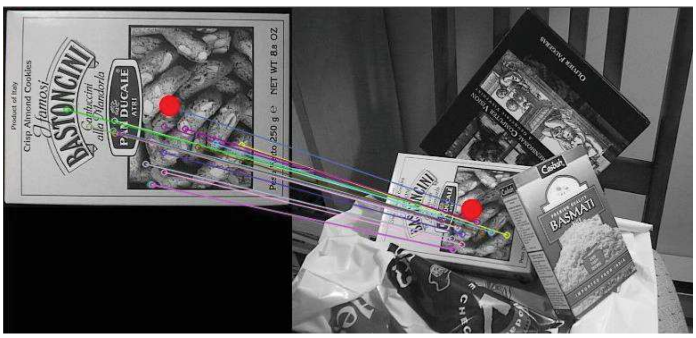
– Detection by Harris or SIFT or SURF or FAST or …

– Description by SIFT/SURF/ORB/…

<u>Detector</u> should ideally be « *repeatable* » i.e. select *same points whatever the scale, rotation, lighting…*

<u>Descriptor</u> should ideally be *invariant under change of scale/rotation/lighting/…*

So that several keypoints can always be matched



Keypoints detectors and descriptors

Very large number of variants of detectors and descriptors successively invented over time

Detectors

1988: Harris
1999: SIFT
2006: SURF, FAST
2011: ORB
…

YEARS

Descriptors

1999: SIFT
2006: SURF
2010: BRIEF
2011: ORB
…

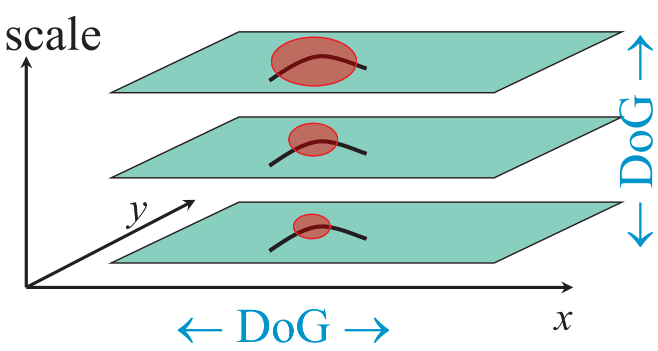
SIFT = *Scale Invariant Feature Transform*
SURF = *Speeded Up Robust Features*
FAST = *Features from Accelerated Segment Test*
BRIEF = *Binary Robust Independent Elementary Features*
ORB = Oriented FAST and Rotated BRIEF

SIFT keypoints

Scale Invariant Feature Transform proposed by Lowe in 1999

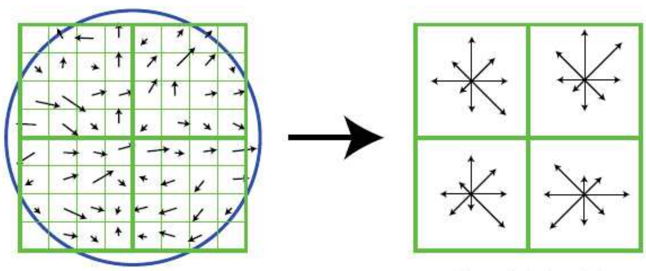
Detector

Max and mins of Difference of Gaussians (DoG) applied in scale space to a series of smoothed and resampled images.



Descriptor

Summarizes spatial distribution of gradient orientations around keypoint in a 128D vector

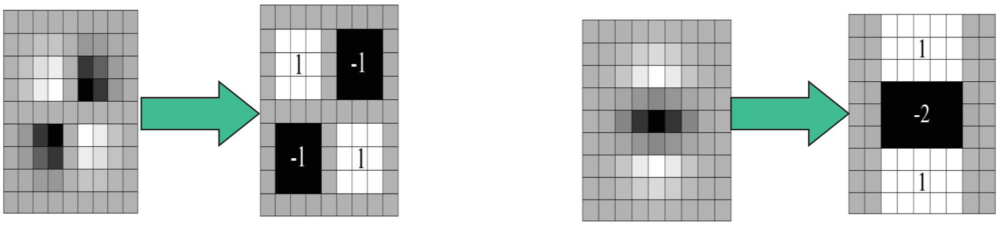


SURF keypoints

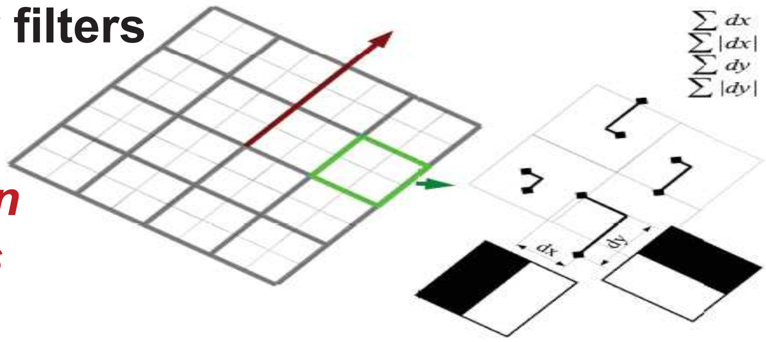
<u>S</u>peeded <u>U</u>p <u>R</u>obust <u>F</u>eatures

proposed by Bay et al. in 2006

<u>Detector:</u> approximation with Haar filters of blob detection by determinant of Hessian (→ speed-up with integral image)



<u>Descriptor:</u> based on Haar filters responses around keypoint



Much faster to compute than SIFT (but « blob » keypoints rather than corners)

Keypoints matching and filtering

- **Precompute keypoints *<u>locations and descriptors</u>* on object to find**
- **Compute keypoints locations and descriptors on « query » (image where we search object)**
- **Find keypoints in query with descriptors similar to a keypoint in object**
- **Filter false matches by geometric checking (RANSAC)**

Advantage: intrinsicly multi-scale search, thanks to scale invariance of keypoint detector and descriptor

Problem: can search/find only a *specific* image pattern

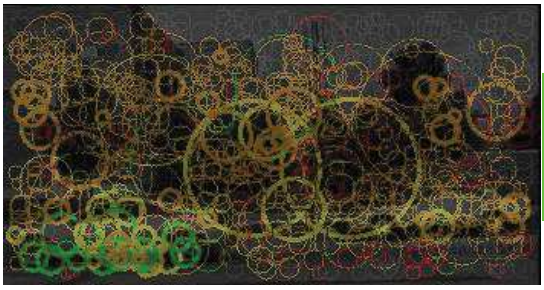
Keypoints categorization

If looking for objects of a CATEGORY (rather than a particular pattern/sub-image), need to first build a filter for discriminating keypoints that are specific of the type of searched objects

- **Extract keypoints on many examples of each category (car, pedestrian, etc…)**

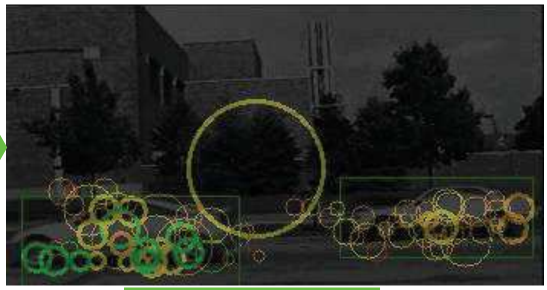
- **Train a classifier on a labelled dataset of *keypoints descriptors*, that predicts category_of_object = f(descriptor)**

Objects category visual detection by keypoints

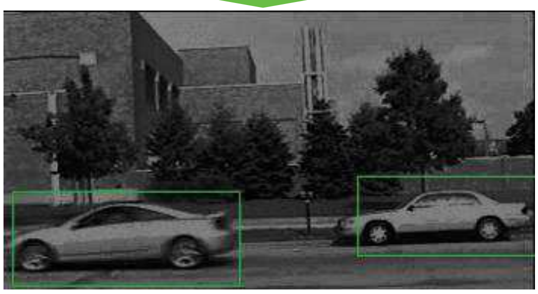


SURF keypoints

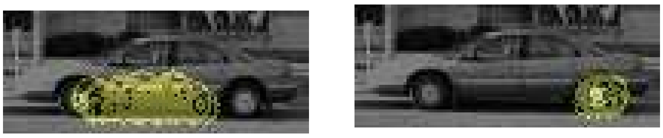
Filtering of CAR keypoints by classifier



CAR objects localization



Semantic interpretability of keypoints



+ Potential for multi-categories simultaneous detection

[Result of research conducted by the center for Robotics of MINES ParisTech]

Summary on visual objects *DETECTION*

Detection = find WHERE in the image
are (maybe) located interesting objects

Detection is a first stage often applied before recognition (which is then applied only on candidate objects output by detection)

Visual objects detection can be done using various types of approaches:

- Template matching
- Shape cues
- Color cues
- Window scanning *with classifier*
- Keypoints matching

Outline

- Motivations: ADAS and autonomous driving
- Objects visual *DETECTION*
- Objects visual *RECOGNITION*:
  - usual *features* used
  - Machine-Learning algorithms
- Traffic Sign Detection and Recognition (TSR)
- Cars & Pedestrians detection with adaBoost

Objects visual RECOGNITION

<u>Robust</u> visual recognition requires independance wrt:

- **Image size**
- **Centering small offsets**
- **Rotations (at least small ones)**
- **Luminosity & contrast**

➔ Generally NOT input pixels directly into classifier, but rather use « <u>FEATURES</u> » computed on image to be classified

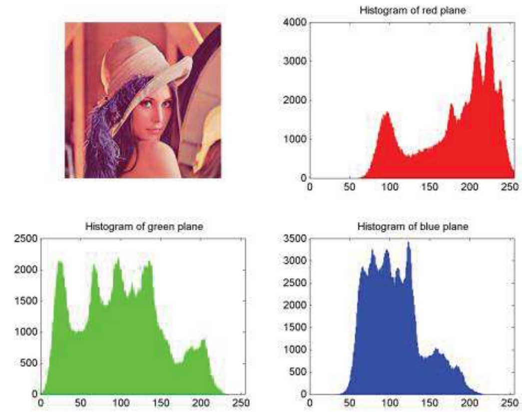
Traditional Machine Learning Flow

Visual FEATURES

<u>Main feature types:</u>

- **Histogram of pixel luminance or color**
- **…**
- **Histogram of Orientations of Gradients (HOG)**
- **Keypoint descriptors, Bag of Word (BoW)**

Luminance or color *Histogram features*



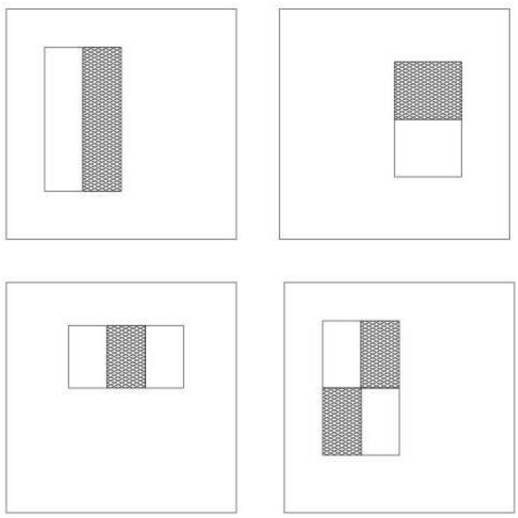
Problems:

- High variability with luminosity/contrast
  - ➔ normalize (histogram equalization)
  - ➔ other color space (YUV, HSV, …)
- *Often not sufficiently discriminative*

The *Viola&Jones features* for object detection: Haar-like filters

4 rectangular feature types:

- *two-rectangles feature* types (horizontal/vertical)
- *three-rectangles feature* type
- *four-rectangles feature* type



Feature output:

Σ(pixels in grey rectangles)

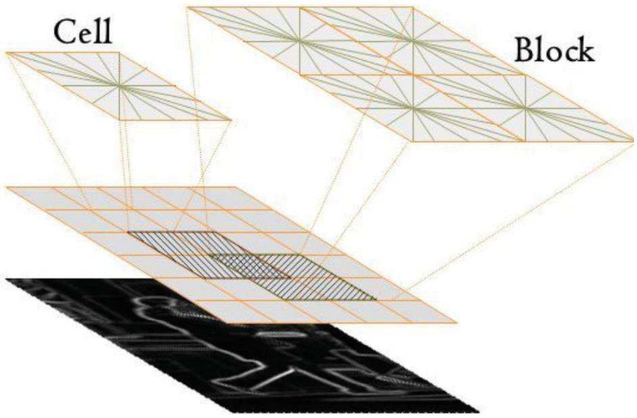
- Σ(pixels in white rectangles)

HOG features

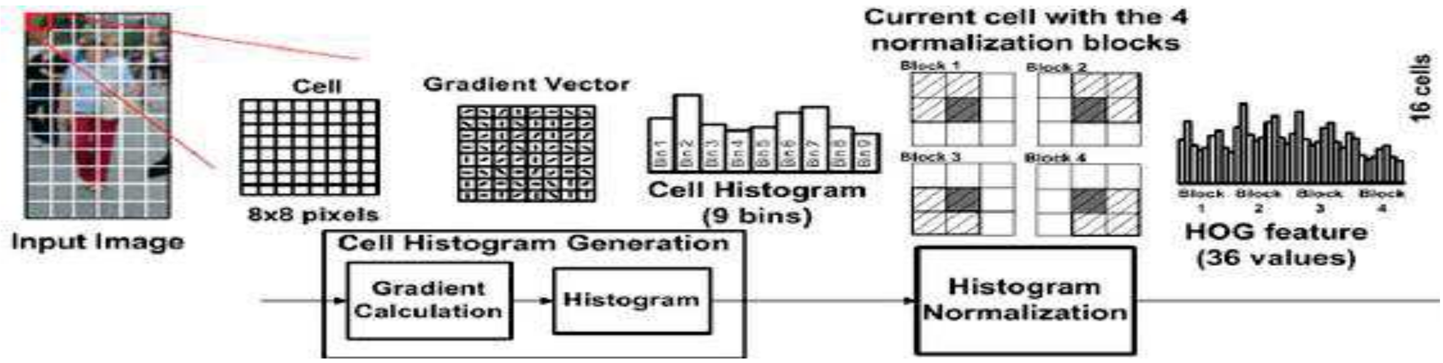
<u>H</u>istogram of <u>O</u>rientations of <u>G</u>radients
popularized by Dalal & Triggs in 2005

<u>Principle:</u>

- Computation of vertical and horizontal gradients with 1D derivative masks `[-1 0 1]` and `[-1 0 1]`^{T}
- Accumulation (weighted by gradient magnitude) of gradient orientations in cell bins
- Normalization within overlapping blocks



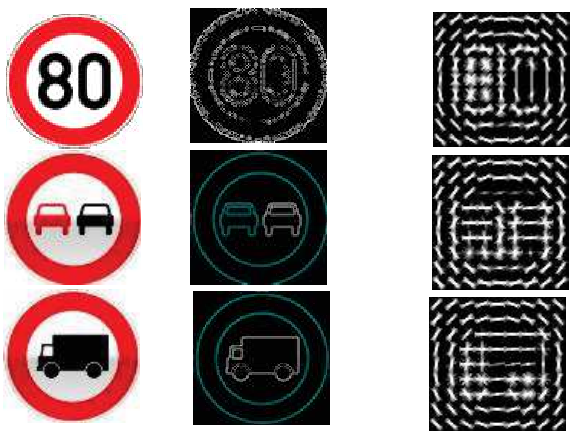
HOG descriptor details



Characterize distribution of contours' orientations

<u>Parameters:</u>

- Cell size (in pixels)
- Number of histogram bins for each cell
- Block size (in cells)

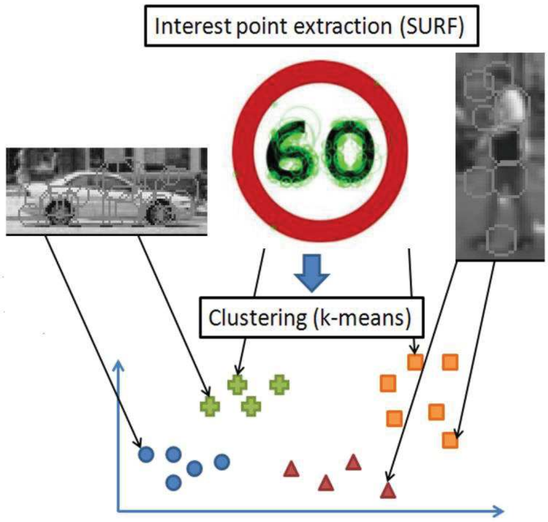


Bag-of-(visual) Words (BoW)

Inspired from text analysis in which a piece of text is represented by a sparse vector of the number of occurrences of each word of a dictionary

Adapted to images using *keypoints descriptors* as a representation of image content:

- **descriptor vectors are quantized (usually by K-means partitioning) into a codebook of « visual words »**
- **An (sub-)image is represented by an histogram of codebook occurences**



Summary on *VISUAL FEATURES*

- **Visual features are characteristics computed on an image to be classified, that describe its content, and will be fed into classifier for recognition**
- **Common types of visual features include:**
  - **Histogram of pixel luminance or color**
  - **Haar-like filters**
  - **Histogram of Orientations of Gradients (HOG)**
  - **Keypoint descriptors, Bag of Word (BoW)**

Outline

- **Motivations: ADAS and autonomous driving**
- **Objects visual *DETECTION***
- **<u>Objects visual *RECOGNITION*:</u>**
  - **usual *features* used**
  - **<u>Machine-Learning algorithms</u>**
- **Traffic Sign Detection and Recognition (TSR)**
- **Cars & Pedestrians detection with adaBoost**

Object visual <u>*RECOGNITION / CATEGORIZATION*</u>

- <u>***RECOGNITION =***</u>
  <u>***determine WHAT are the detected objects***</u>
  (ie assign a type/class to each one)
- **It is therefore a <u>*classification task*</u>: for traffic sign recognize its type (eg Speed Limit to 50 km/h), and for other objects CATEGORIZE them as car / pedestrian / bicycle etc (or false alarm)**
- **Classifiers are generally obtained by applying a Machine-Learning algorithm on *visual features* computed on candidate sub-image (rather than on raw pixels)**

What is statistical Machine-Learning (ML)?

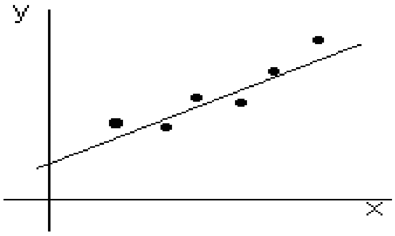
(Statistical) ***Machine Learning*** = **Building an empirical** (i.e. data-driven) **mathematical model**, for automated classification, regression, clustering, or behavior rule

Most simple « Machine-Learning » example:

Least Squares Linear Regression

= find a and b minimizing $K=\Sigma_i (y_i-a.x_i-b)^2$

so that (straight) line $y=ax+b$ fits the points



For objects visual recognition or categorization

Supervised Machine-Learning

Examples (input-output) $(x_1,y_1), (x_2,y_2), \dots, (x_n, y_n)$ →

H (parameterized) family of mathematical models →

Hyper-parameters for training algorithm →

LEARNING ALGORITHM

(usually based on optimization technique)

→ $h \in \mathbf{H}$ so that $h(x_i) \approx y_i$

Optimization methods used by ML include:
- Gradient descent
- Quadratic programming
- Decision tree inference
- …

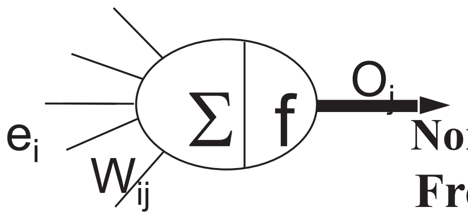
ML (shallow) algorithms for visual object recognition

Main *shallow* (ie not-deep) Machine-Learning algorithms used:

- MLP Neural Networks
- Support Vector Machines (SVM)
- Random Forets
- Boosting

Multi-Layer Neural Networks (MLP)

One neuron



$$O_j = f\left(W_{0j} + \sum_{i=1}^{n_j} W_{ij} e_i \right)$$

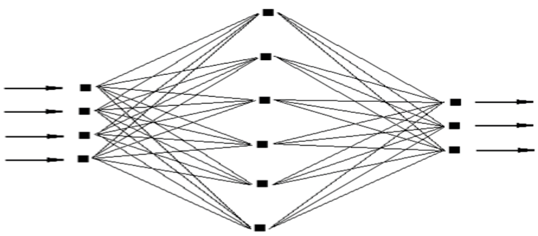
Non-linear transform(Linear combination of inputs)

Free parameters: set of « synaptic weights » W_{ij}

Network: usually 1 input layer + 1 hidden layer + 1 output layer

Main parameter: size of hidden layer

Input: vector of features



Output: typically, 1 per class (with 1-vs-all encoding, and actual values ~ probabilities)

Training: random initialization of W_{ij} weights

+ Iterative *gradient descent minimizing error function*

$$E(W) = \sum_p \left(Y_p - D_p \right)^2$$

MLP training hyper-parameters

Architecture:

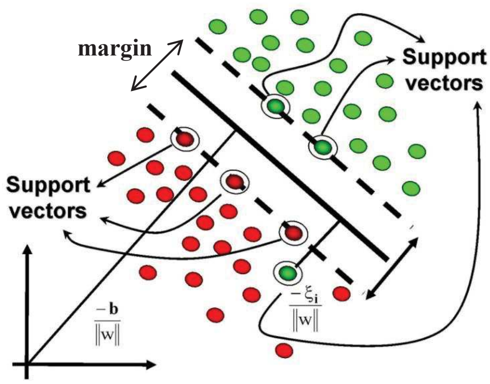
- usually 1 input layer + ONLY 1 hidden layer
  + 1 output layer
- Main parameter:
  size (number of neurons) of hidden layer

Optimization:

- Type of gradient descent algorithm
- Main parameter for *standard* gradient: learning step
  + momentum*
- Number of iterations

Linear Support Vector Machines (SVM)

Provide *optimal (maximal margin) hyperplane separator* in input space



Training: quadratic programming to solve convex optimization

$$\arg\min_{\mathbf{w},b}\max_{\alpha\geq 0}\left\{\frac{1}{2}\|\mathbf{w}\|^2-\sum_{i=1}^{n}\alpha_i[y_i(\mathbf{w}.\mathbf{x}_i)-b)-1]\right\}$$

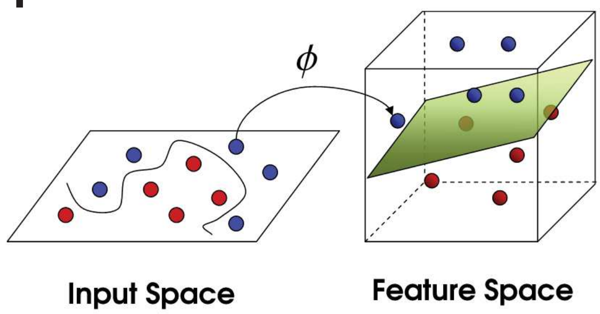
Linear SVM output:

$$h(X)=b+\sum_{i=1}^{N_S}\beta_{S(i)}\,X\cdot X_{S(i)}=b+X\cdot\left(\sum_{i=1}^{N_S}\beta_{S(i)}\,X_{S(i)}\right)$$

with $X_{s(i)}$ = ***Support Vectors***

Kernel for *non-linear* SVM

Classes are often NOT linearly separable. But linear SVM can be applied in a *transformed* space in which classes are hopefully linearly separable



Kernel "trick": **use a transform $\Phi(X)$ *implicitly* defined by $\Phi(X_1).\ \Phi(X_2)=k(X_1,X_2)$ [with k = KERNEL]**

➔ **Training same as linear SVM, with just *replacing $w.x_i$ by $k(w,x_i)$***

$$\arg\min_{\mathbf{w},b}\max_{\alpha\geq 0}\left\{\frac{1}{2}\|\mathbf{w}\|^2-\sum_{i=1}^{n}\alpha_i[y_i(\mathtt{k(w,x_i)}-b)-1]\right\}$$

Non-linear* SVM output:** $h(X)=\sum_{i=1}^{N_S}\beta_{s(i)}k(X_{s(i)},X)+b$ **with $X_{s(i)}$ = *Support Vectors

SVM training hyper-parameters

Kernel:

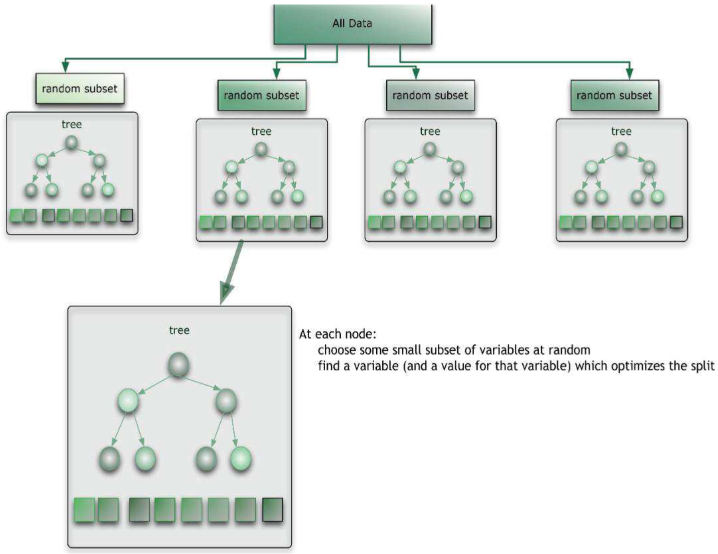
- Type (linear or polynomial or Gaussian)
- Kernel param (degree for polynomial, sigma for Gaussian)

Optimization:

- tolerance parameter C !!!

Random Forest

- A Random Forest is a <u>set of N Decision Trees</u> (typically N ~ tens, hundreds or more)
- Each Decision Tree is learnt on a <u>≠ *random* subset of</u> training <u>examples</u>, using only a <u>*randomly* chosen and *small* set of coordinates</u>
- The output of the Random Forest is the <u>majority vote by all trees</u>



RandomForest training hyper-parameters

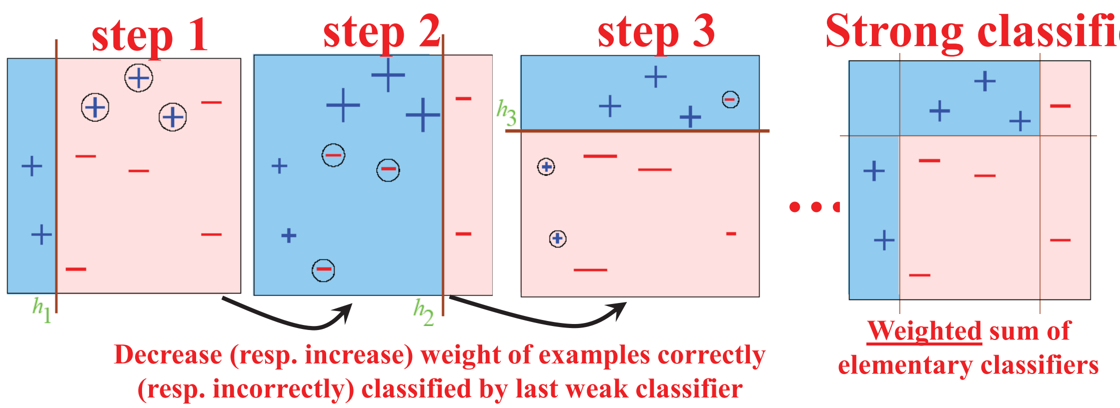
<u>Size = number of trees</u>

<u>Max-depth of trees</u>

<u>Randomization:</u>
- <u>% of randomly chosen training examples for each tree</u>
- <u>% of random input coordinates used in each tree</u>

Boosting

adaBoost principle: weighted vote of a "committee" of "weak classifiers" obtained by successive weightings of examples



➔ **Final STRONG classifier:** $H(x) = \mathrm{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

adaBoost Algorithm [Freund&Schapire 1995]

Given $(x_1, y_1), \ldots, (x_m, y_m)$ with: $x_i \in X,\ y_i \in Y = \{-1, +1\}$

Initialize weights with: $D_1(i) = \frac{1}{m}$

For $t = 1, \ldots, T$:

- In features family, find $h_t : X \to \{-1, +1\}$ minimizing error (weighted using $D_t(i)$ for examples)

$$h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y(i) \neq h_j(x_i)]$$

Weak-Learner

- Check if ε_t<0.5, otherwise STOP
- Evaluate weak-classifier with $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ where ε_t is the weighted error of h_t
- Update weights with:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Adaptation of example weights:
↗ for those incorrectly classified by h_t
↘ for those correctly classified by h_t

➔ **Final STRONG classifier:** $H(x) = \mathrm{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

adaBoost training hyper-parameters

Weak-Learner:

Algo used?

If feature selection, which family (Haar, HOG, controlPoints)?

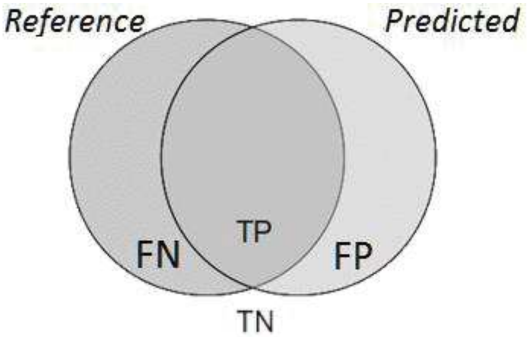
Number of Weak-Classifiers to assemble

Comparison of main "shallow" ML algorithms

	MLP Neural Network	SVM	Boosting	Random Forest
Many classes	+	--	--	++
Large dimension of input		-		++
Many examples		-		
Easy to train	-	++	+++	
Feature handling			Selection	
Fast recognition		+		++
Robustness to data noise	+	++		++

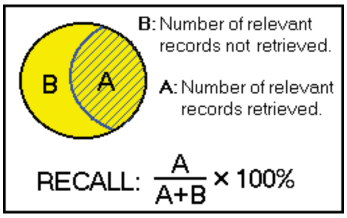
Choice of a particular ML model/algorithm should ideally be done empirically: try all of them and keep best performing!
It can also be influenced by characteristics of training data (# of classes, dimension of input, # of examples), by relative ease of training, and by execution speed of recognition

Different types of classification errors

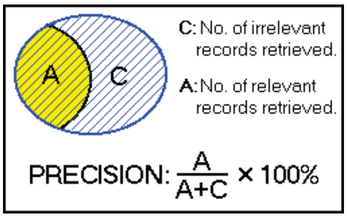


	predicted as positive	predicted as negative
positive	TP	FN
negative	FP	TN

False Negatives (« missed ») vs False Positives



Recall: percentage of relevant examples successfully predicted/retrieved



Precision: percentage of actually relevant examples among all those returned by the classifier

Recall and precision formulas

	predicted as positive	predicted as negative
positive	TP	FN
negative	FP	TN

Recall (sensitivity) True Positive rate

$$= \frac{\text{Nb of } \underline{\text{correct}} \text{ positive predictions}}{\text{Nb of } \textit{real} \text{ positives}} = \frac{TP}{TP + FN}$$

Precision (specificity)

$$= \frac{\text{Nb of } \underline{\text{correct}} \text{ positive predictions}}{\text{Nb of positive } \textit{predictions}} = \frac{TP}{TP + FP}$$

Classification performance metrics

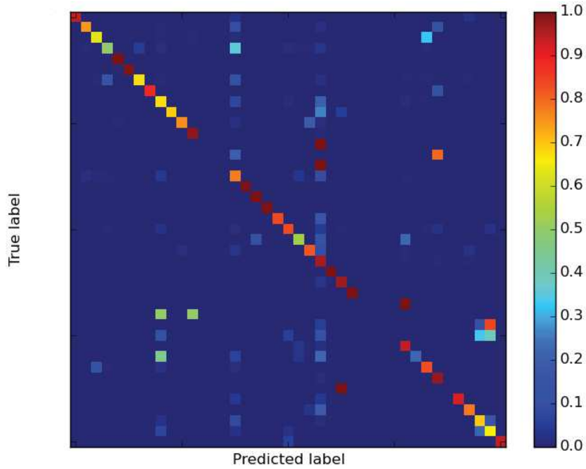
- **Recall (sensitivity)** ≈ proportion of « not missed »
  ≈ « exhaustivity » level

- **Precision (specificity)** ≈ reliability of predicted labels

- **Confusion matrix**: predicted label v.s. true label

True positive	False positive
True negative	False negative

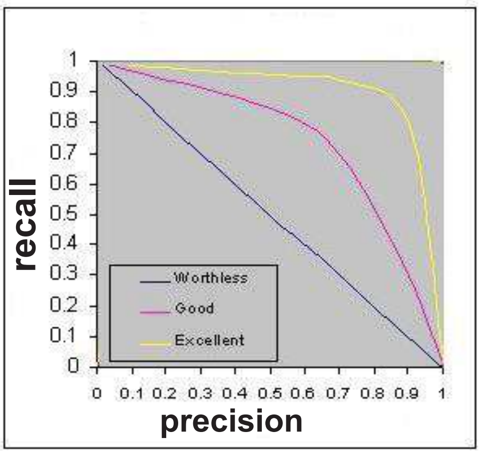
C.Matrix	1	2	3	4	5	6	ACTUAL	RECALL
1	339	15	5	0	0	0	359	94.43%
2	15	305	14	0	0	0	334	91.32%
3	6	10	242	0	0	0	258	93.80%
4	0	0	0	302	30	0	332	90.96%
5	0	0	0	15	368	0	383	96.08%
6	0	0	0	0	0	394	394	100.00%
PREDICTED	360	330	261	317	398	394	2060	94.43%
PRECISION	94.17%	92.42%	92.72%	95.27%	92.46%	100.00%	94.51%	94.66%



Precision-recall trade-off and curve

Classifier C1 predicts better than C2
iff C1 has better recall *and* precision

+ Trade-off between recall and precision



➔ Compare precision-recall *curves*!

For numeric comparison (or if curves cross each other), Area Under Curve (AUC)

Notion and importance of GENERALIZATION

« LEARNING = INFER/APPROXIMATE
<u>*+ GENERALIZE !!*</u> »

Given a FINITE set of examples (x_1,y_1), (x_2,y_2), … , (x_n, y_n), where $x_i \in \Re^d$ are input vectors, and $y_i \in \Re^s$ are *target output* values, we search a function h that « <u>*fits AND GENERALIZE*</u> best » the underlying actual function f defined by $y_i=f(x_i)$+noise

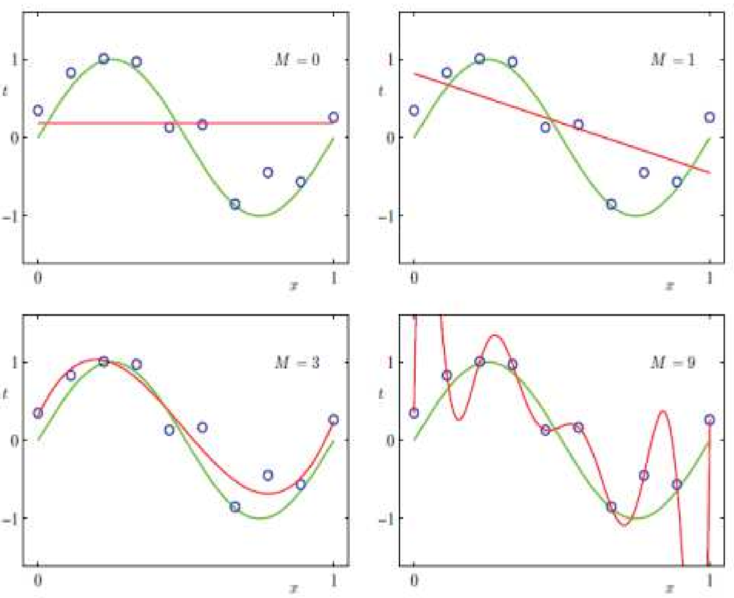
$\Rightarrow$ goal = minimize the GENERALIZATION error

$$E_{gen} = \int \|h(x)-f(x)\|^2 \, p(x)\,dx$$

(where p(x)=probability distribution of x)

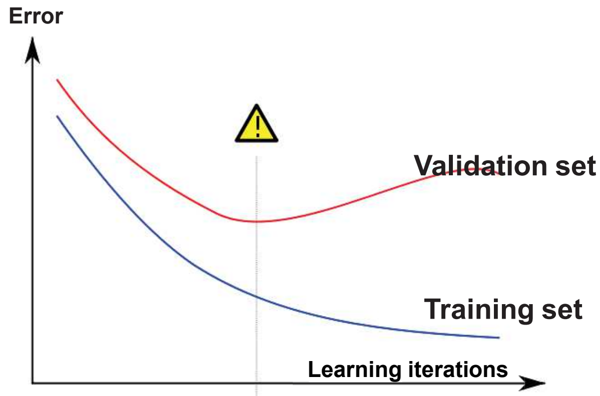
What is « overfitting »?

What can be measured (and minimized!) is only the EMPIRICAL error on examples: $E_{emp} = \left(\Sigma_i \|h(x_i)-y_i\|^2 \right)/n$



Fitting a data set to different orders of polynomials
[from Bishop, "Pattern Recognition and Machine Learning"]

<u>Over-fitting detection</u>
for an iterative algorithm

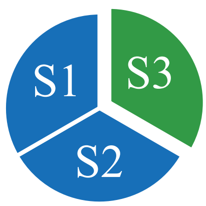


Training methodology: ALWAYS use validation-set or cross-validation!

For maximizing GENERALIZATION (and avoid overfitting), it is essential to choose/optimize all training parameters with VALIDATION:

- either with a *separate* validation set (random splitting of examples into Training+Validation)
- or with CROSS-VALIDATION:

  estimate error on several subsets used as validation (k-fold or « leave-one-out »), then average errors



3-fold cross-validation :

- train on S1∪S2 and evaluate on S3
- train on S1∪S3 and evaluate on S2
- train on S2∪S3 and evaluate on S1
- Average (errS1, errS2, errS3)

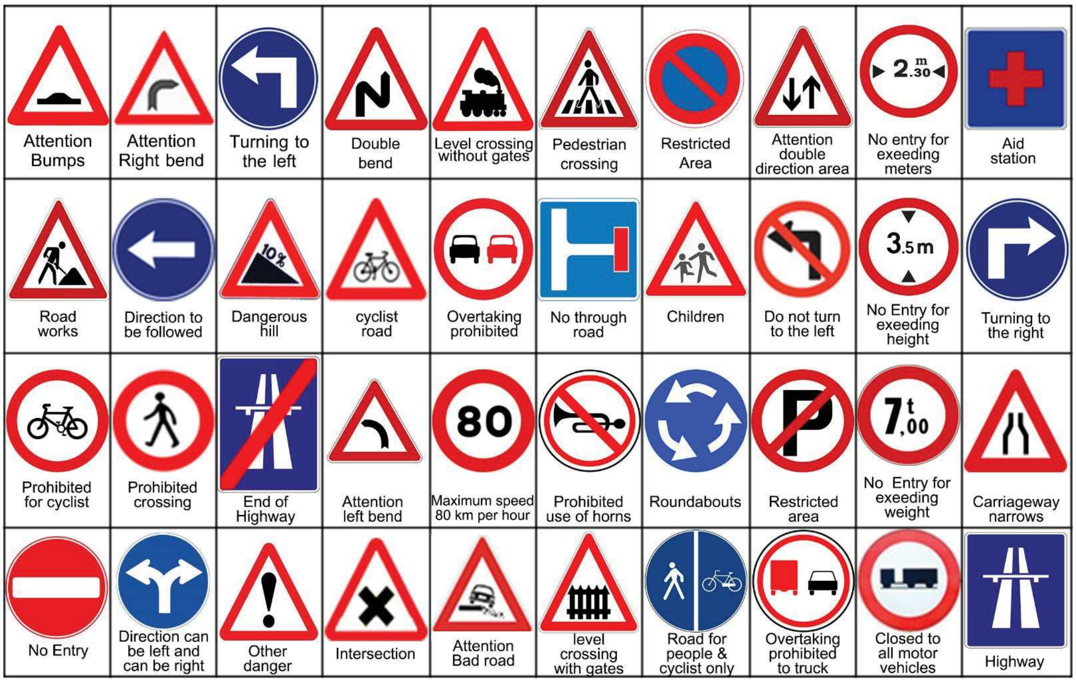
Summary on *shallow Machine-Learning algorithms for visual objects recognition*

- Visual recognition is generally performed using Machine-Learning (ML) *applied on visual features*
- ML = Building an empirical (i.e. data-driven) mathematical model, eg for automated classification
- Main shallow ML algorithms used for visual object recognition include:
  - MLP Neural Networks
  - Support Vector Machines (SVM)
  - Random Forests
  - adaBoost

Outline

- **Motivations: ADAS and autonomous driving**
- **Objects visual *DETECTION***
- **Objects visual *RECOGNITION*:**
  - **usual *features* used**
  - **Machine-Learning algorithms**
- **<u>Traffic Sign Detection and Recognition (TSR)</u>**
- **Cars & Pedestrians detection with adaBoost**

Traffic Signs



Shape, colors and pictograms ≈ standardized
(but national variations & totally different in USA…)

Traffic Sign detection and Recognition

3 main steps:

1. Where are traffic signs?
   → *Detection* by color or/and shape
2. What traffic sign is it?
   → Use pattern recognition (→ require use of some Machine-Learning)
3. Temporal integration (tracking)
   → Position prediction, better confidence estimation, and handle temporary occlusions

Main challenges:
- real-time detection (signs are small !)
- robustness to illumination changes

Traffic Signs DETECTION

- Often done by COLOR THRESHOLDING
  ➔ fast, but poor robustness to illumination changes
- Alternative or complement: SHAPE DETECTION (circles, triangles, rectangles) using Hough
  ➔ robust, and OK even on greyscale,
  BUT very computer-intensive if ≠ optimized
- Best = using COLOR AND SHAPE
  Color → candidate regions
  Shape detection restricted to those regions

Traffic Sign RECOGNITION (TSR)

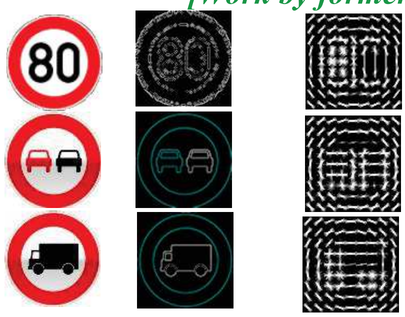
- **Very little intrinsic variation of object**
  → main recognition challenge = *robustness to illumination & contrast changes + small 3D rotations*
- **Large number of classes (~100)**
- **Input feature for classification?**
  - Vector of pixel values??
  - HoG (Histogram of Orientations of Gradients)
  - …
- **ML algo used: *Neural Nets, Random Forest*, boosting, SVM (but 2 last = BINARY classifiers → less convenient)**

MINES_ParisTech's approach for Traffic Sign Recognition (TSR)

[Work by former PhD student Fatin Zaklouta]

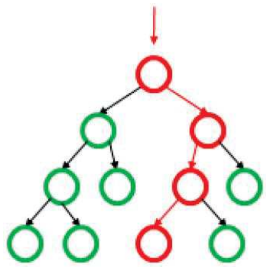
German Traffic Sign Recognition benchmark (GTSRb)

43 classes, 26640 training images
2569 test images



Gradient (Sobel)

Histogram of Orientations of Gradients (HOG)



Use set of random trees

Machine-Learning algorithm used: random forest

Principle: 1/ Grow large (typically 500) set of "random" trees, with each node testing 1 of the 1000-3000 HoG componants (node = best split); 2/ Labels of leaves computed based on most frequent class of training examples ending in it; 3/ *Classify by majority vote of trees*

Best student paper @ICAR'2011
3rd best competition result: 96,1% (vs 99,5% and 98,3%)

MINES_ParisTech's TSR result

Outline

- Motivations: ADAS and autonomous driving
- Objects visual *DETECTION*
- Objects visual *RECOGNITION*:
  - usual *features* used
  - Machine-Learning algorithms
- Traffic Sign Detection and Recognition (TSR)
- <u>Cars & Pedestrians detection with adaBoost</u>

Cars or pedestrians visual detection

Main challenge: *very large intra-class variability!!*

➔ **Requires LARGE dataset of examples**

pedestrians

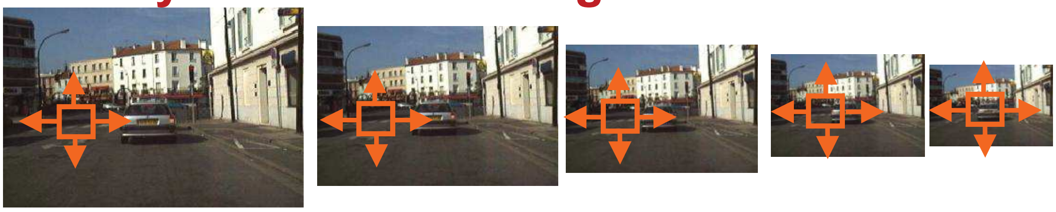
« non-pedestrians »

($N \geq 10^3$-10^4 positives + $\geq 10^4$-10^5 negatives)

↓

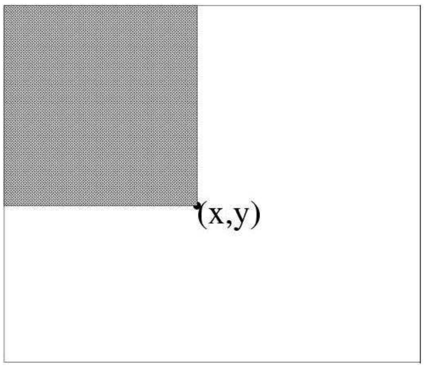
Machine-Learning ⟶ **Image classifier**

+ Detection by window-scanning ➔ classifier must be FAST



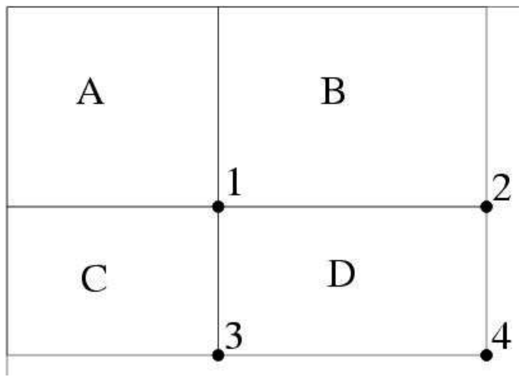
…

Integral image



- **Definition**: The *integral image* at location (*x*,*y*), is the sum of the pixel values above and to the left of (*x*,*y*), inclusive.
- It can be computed in one single pass with nb_pixels additions.

Using the integral image representation one can compute the value of any rectangular sum in constant time.
For example the integral sum inside rectangle D we can compute as:
***ii*(4) + *ii*(1) – *ii*(2) – *ii*(3)**



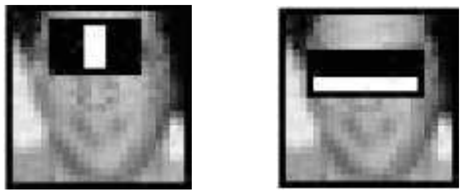
➔ **VERY FAST COMPUTATION of ViolaJones features**

Boosting as feature selection (and weighting)

adaBoost = weighted vote by a committee of ***"weak classifiers"*** obtained by iterative weightings of examples

➔ Final STRONG classifier: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

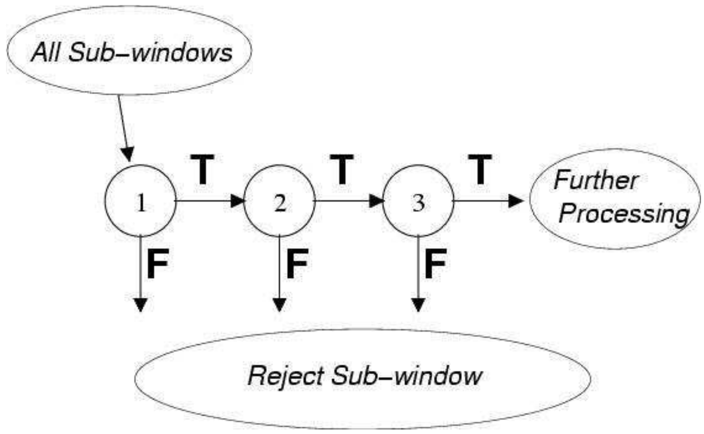
Idea of Viola&Jones in 2001: ***use as weak classifier very simple boolean features selected in a family*** (e.g. all Haar-like features) ⇔ Weak Learner = search of feature with lowest weighted error



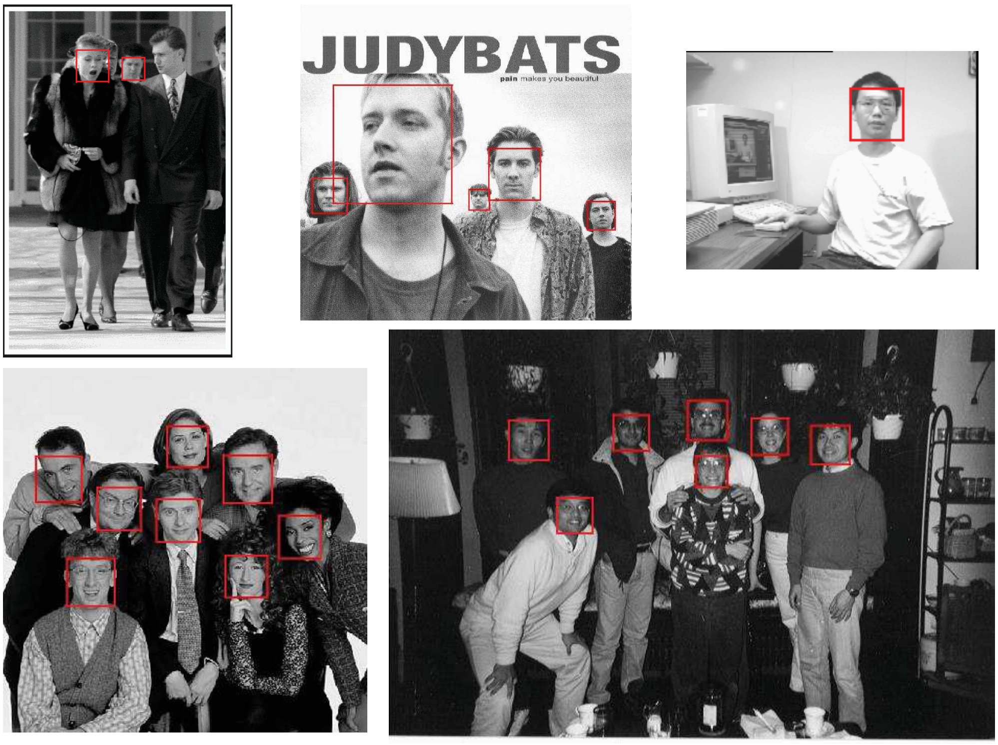
Using a 24x24 pixels detection window, with all possible combinations of horizontal&vertical location and scale of Haar, the full set of features has 45,396 ≠ features (and ~10 times more in a 32x32 window) ➔ brute-force exhaustive search possible!

Speed-up by « Attentional » Cascade

- Simple, boosted classifiers can reject many negative sub-windows and still detect all positive instances
- Cascade of progressively more complex classifiers
  → good detection performance with less processing (most negative sub-windows eliminated by simplest classifiers at beginning of cascade)



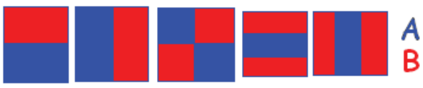
« Success story »: now standard face-detection approach



Result of multi-scale window-scanning with strong classifier obtained by boosting of Haar filters (Viola&Jones, 2001)

Main families of Weak Classifiers for boosting

- **Haar-like (Viola-Jones) = most commonly used features**



if $\left|SumPixels(A) - SumPixels(B)\right| > Threshold$ then True else False

  - **Relatively fast computation with integral image**
  - **Mostly based on horizontal/vertical contrasts**

Some work showed improved results with extended feature set [Treptow & Zell, CEC'2004]

- **HOG (Histogram of Oriented Gradient) – based features**

  [Zhu et al., CVPR'2006, Mitsubishi] [Pettersson et al., IV'2008, NICTA]

  - **More detailed/discriminative information**
  - **Tricky to make it fast enough**
  - **Not so good results on object classes with too shallow gradients**

- **Pixel-pairs comparisons**

  [Baluja et al., ICIP'2004, Google/CMU] [Leyrit et al., IV'2008, LASMEA]

  - **Extremely low computation time**
  - **Less discriminative ➔ more WC, or more complex classif required**

- **Control-points features** [CAOR/Mines ParisTech work since 2004]

Outcome of boosting with ≠ feature families

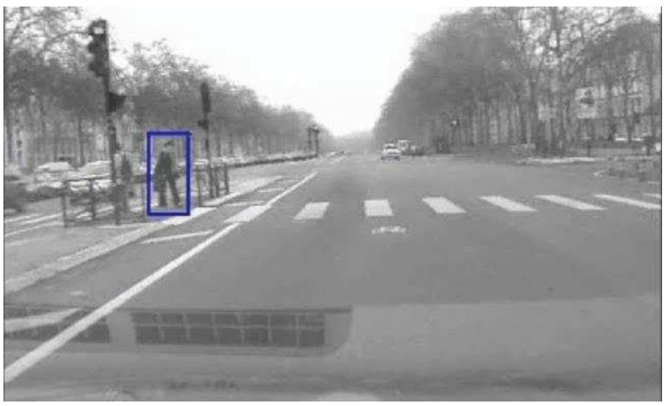
Typical connected-Control-Points selected during Adaboost training

For comparison, typical Adaboost-selected Haar features

Example result of car & pedestrian detection with boosting

Cars (from behind) : ~ 95% detection with < 1 false alarm / image

[Research conducted @ center for Robotics of MINES ParisTech]



Pedestrian (daytime) : ~80% detection with < 2 false alarms / image

(Intermediate) Conclusions

Until outbreak in 2013 of Deep-Learning with Convolutional Neural Networks, state-of-the-art in real-time visual object detection and recognition or categorization for Intelligent Vehicles was:

- **For Traffic Signs, Color and/or Shape detection + Random Forest recognition**
- **For more complex/variable categories (cars, pedestrians, etc…) boosting selection of weak features, or SVM classification using HOG**

These techniques are still those used in most already existing products

NB: in most cases, fusion with information by processing of input from other sensors: radar, lidar, …

NB: Deep-Learning approaches for visual scene analysis in a separate course