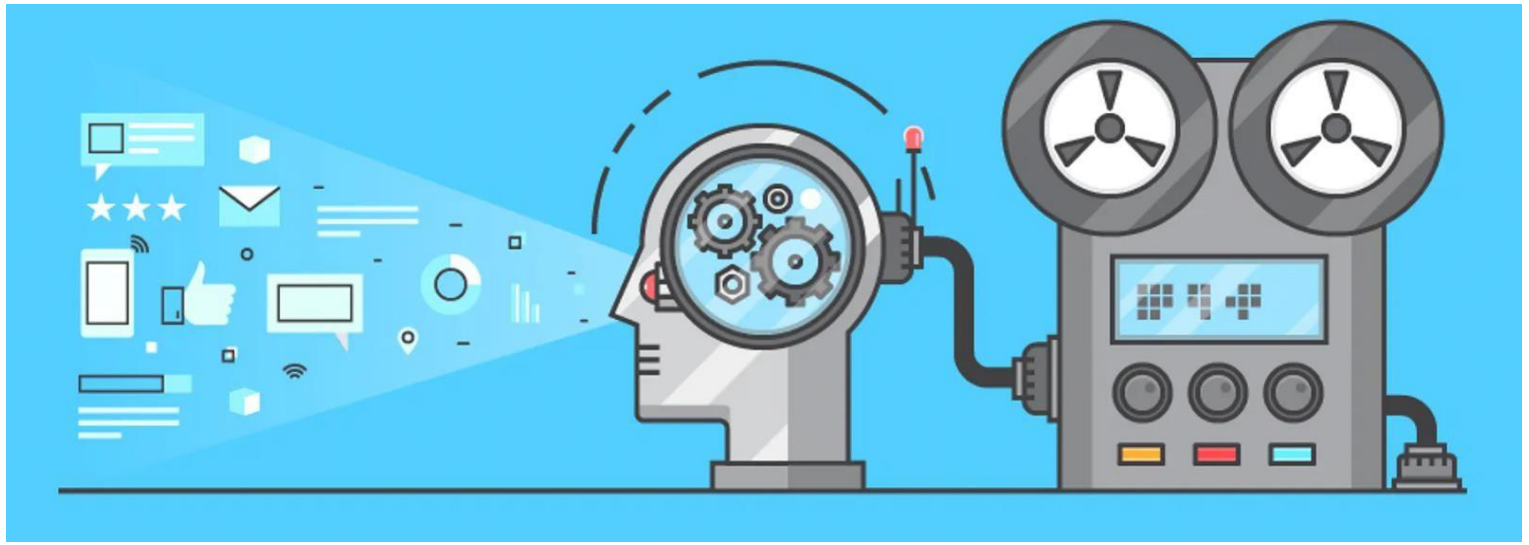
Deep Learning and Neural Networks

Topic 9:

Object Detection



Ricardo Abel Espinosa Loera, McS

Researcher in DL & Computer Vision

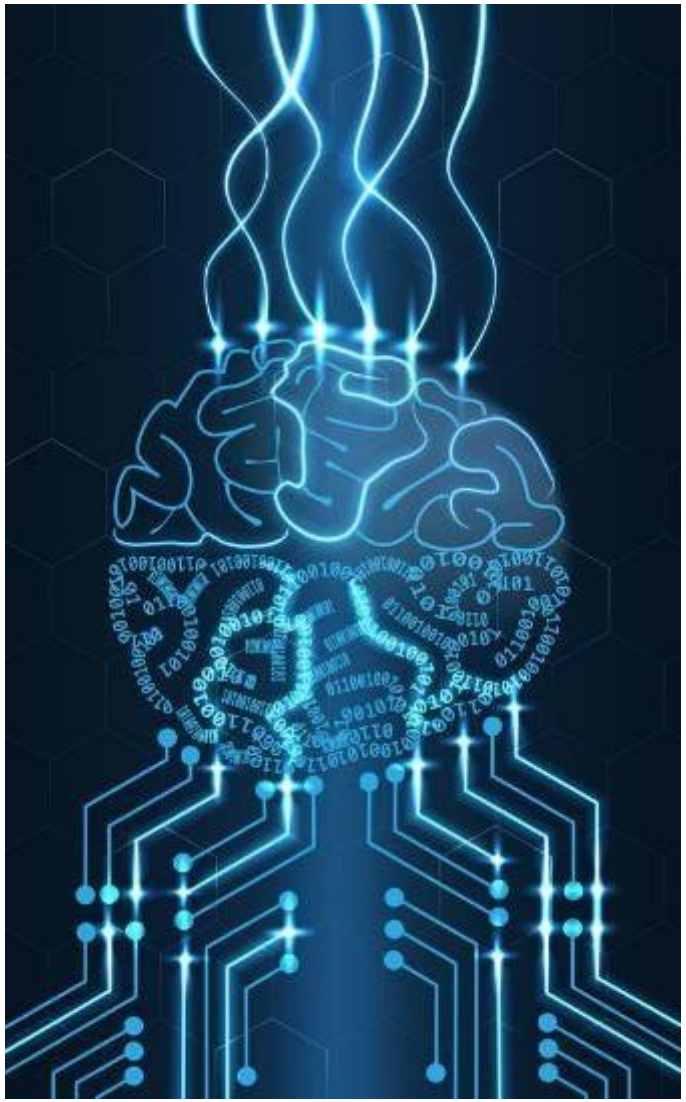
Agenda – Outline of the class 1

Introduction
A brief history and main advances
The current "fuss" about AI

Trends in AI
Main areas of application and trends
Markets showing the highest growths

Neural Nets and Deep Learning
A very gentle and brief introduction to Convolutional Neural Networks

Introduction to Object Detection
Technical evolution in the last 20 years



Agenda – Outline of the class 2

Object Detection

Early milestones

Challenges and constraints

Methods, datasets and metrics

Modern DL-based approaches

Two stage detectors: RCNN, variants

One stage detectors: YOLO, SSD, etc

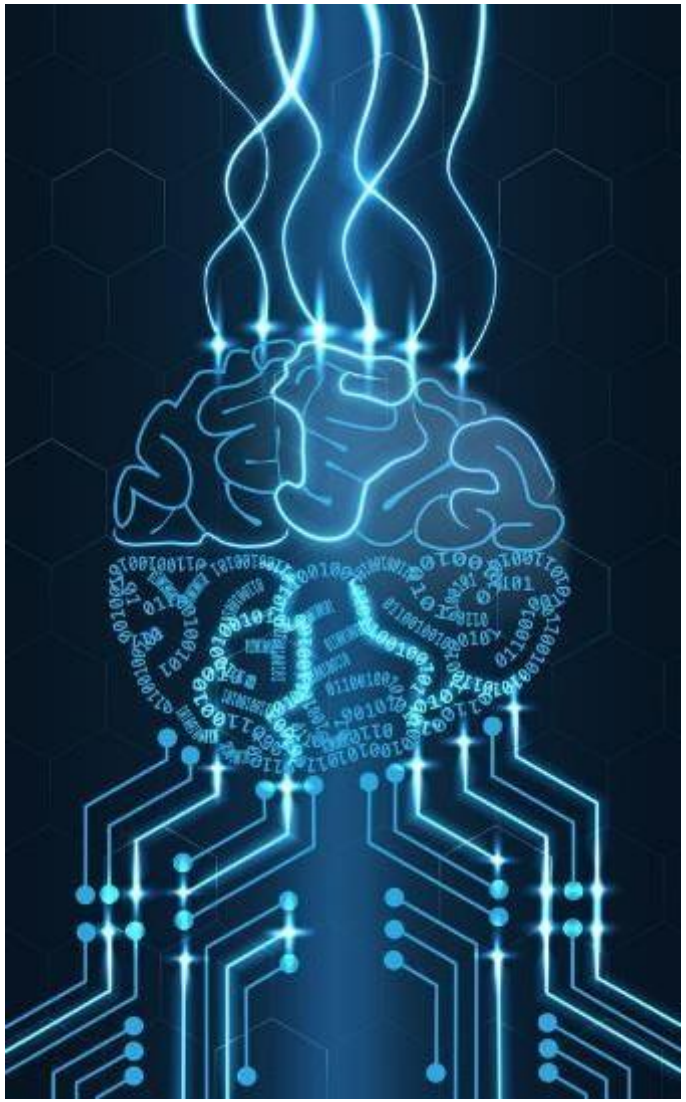
Comparison in terms of performance

Object Detection in Google Colab

Introduction to the tool

Data Augmentation

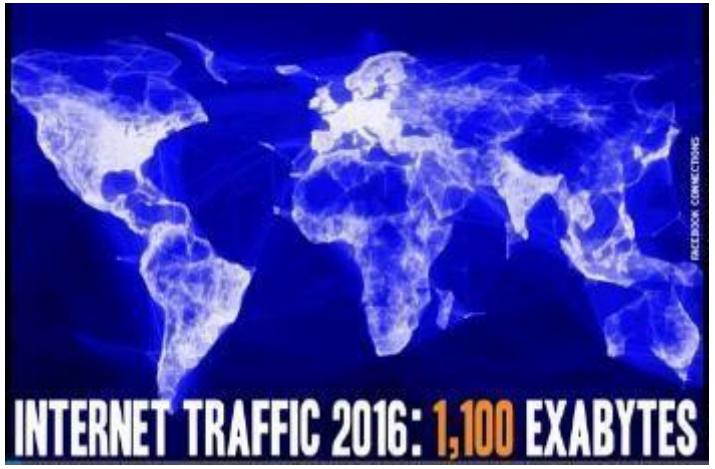
Training and Testing



Introduction:

A brief intro to the progresses made in AI

So, neural networks are trendy now, what happened? **1: Big Data**



Introduction:

A brief intro to the progresses made in AI

Machine learning requires of humongous amounts of data!!!

The internet produces an exponential growth of information

The NLP & computer vision fields suffered a major revolution

NLP, apps:

From mobile computing, but also from web

Computer Vision:

From web images, videos on Youtube, smartphones

Introduction:

A brief intro to the progresses made in AI

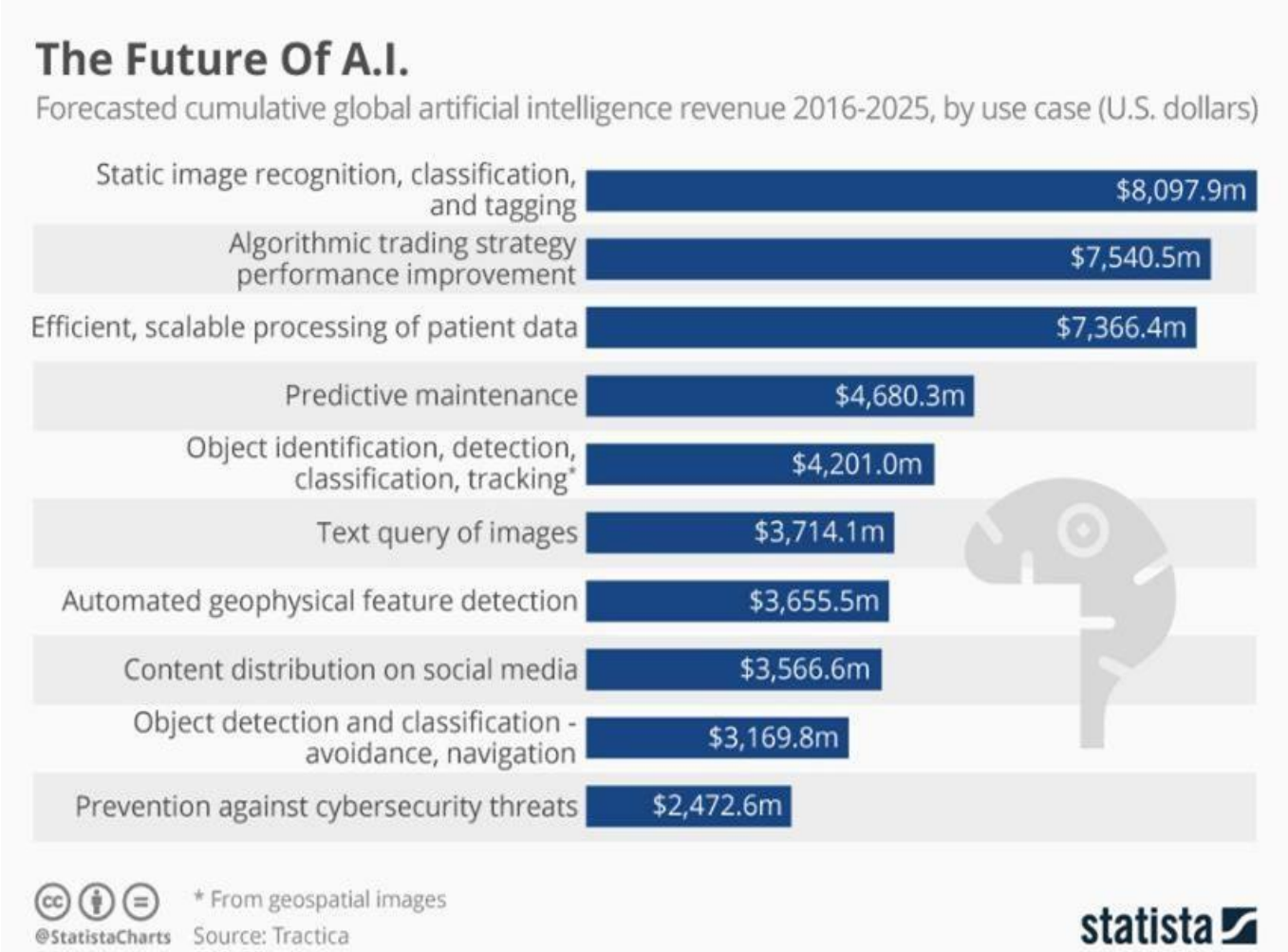
So, neural networks are trendy now, what happened? **2: Hardware**

Even if Moore Law is now over, the development of GPUs and their use has enabled: big data and DEEP LEARNING

Marenostrum at the Barcelona SuperComputing Center

Trends:

What are the main AI application areas?

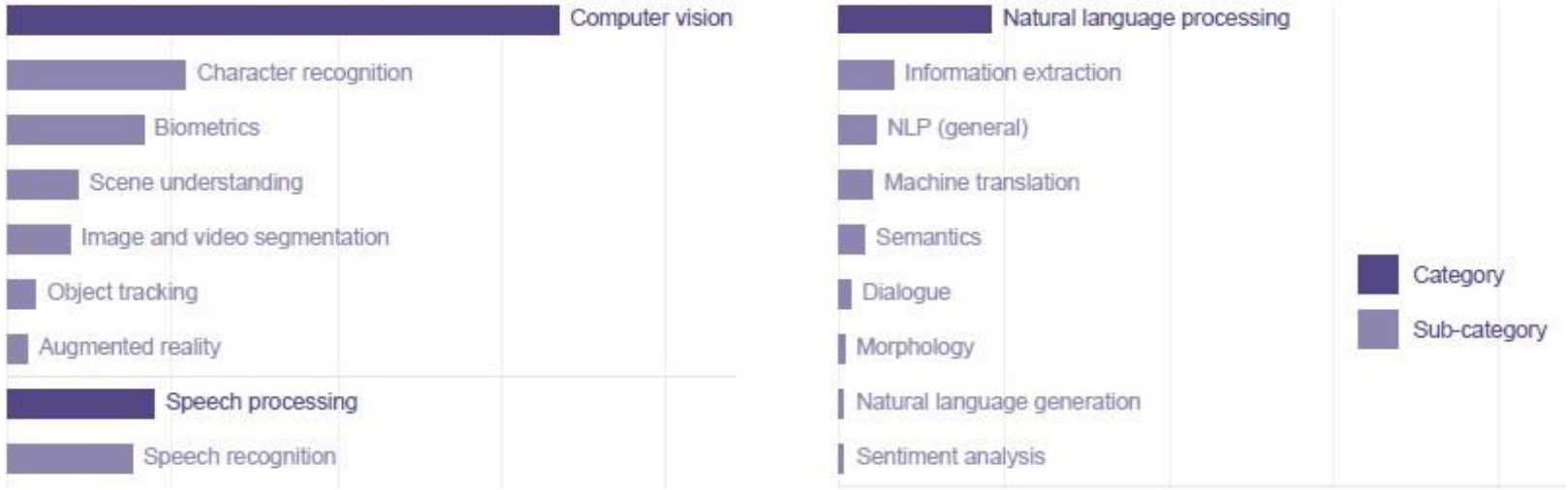


Trends:

What are the main AI application areas?

Figure 3.10. Patent families for functional application categories and sub-categories

Computer vision represents 49 percent of patent families related to a functional application



Trends:

What are the main AI application areas?

Computer Vision

Biometrics

Character Recognition

Scene Understanding

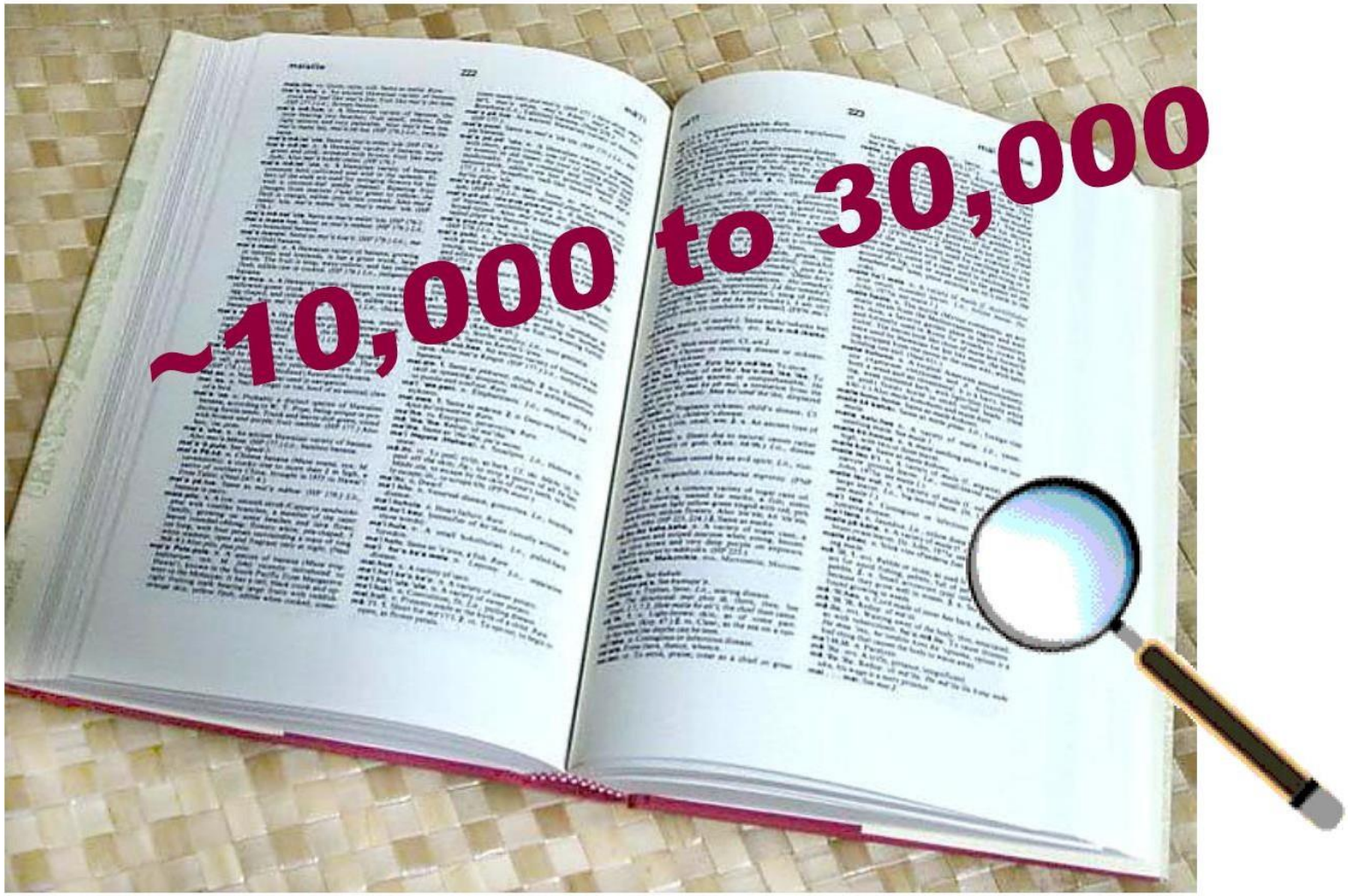
Tasks Beyond Classification

What is next?

Holy grail
a model that achieves
human level
scene understanding

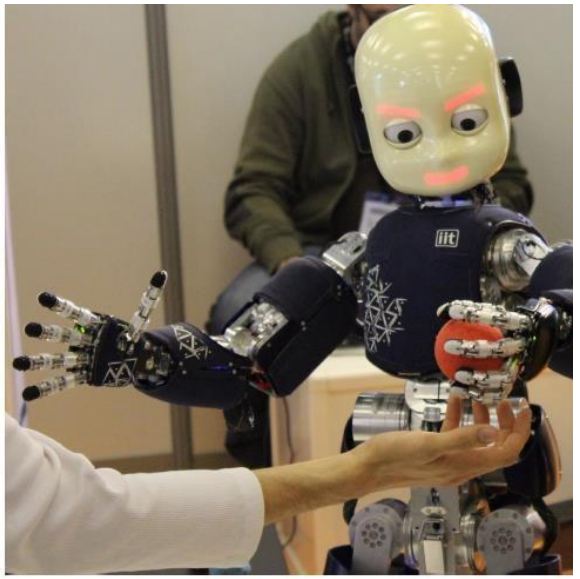
Tasks Beyond Classification

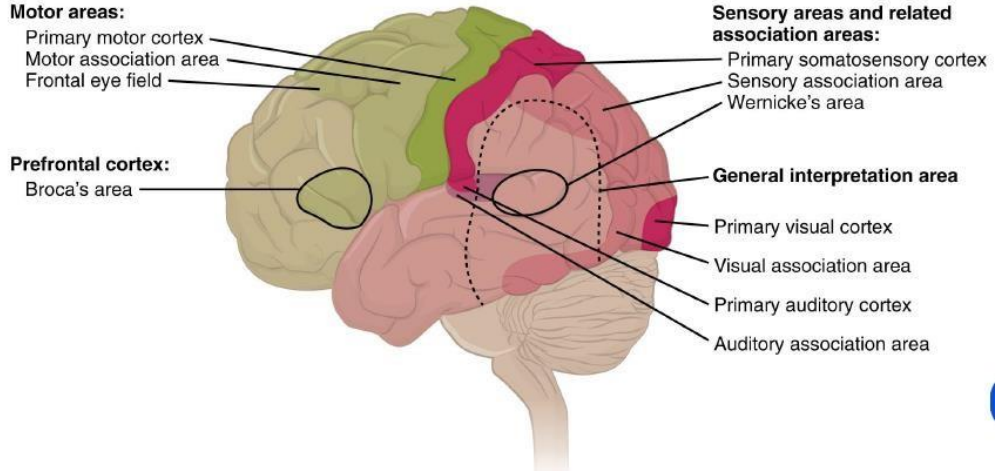
What is next?



Tasks Beyond Classification

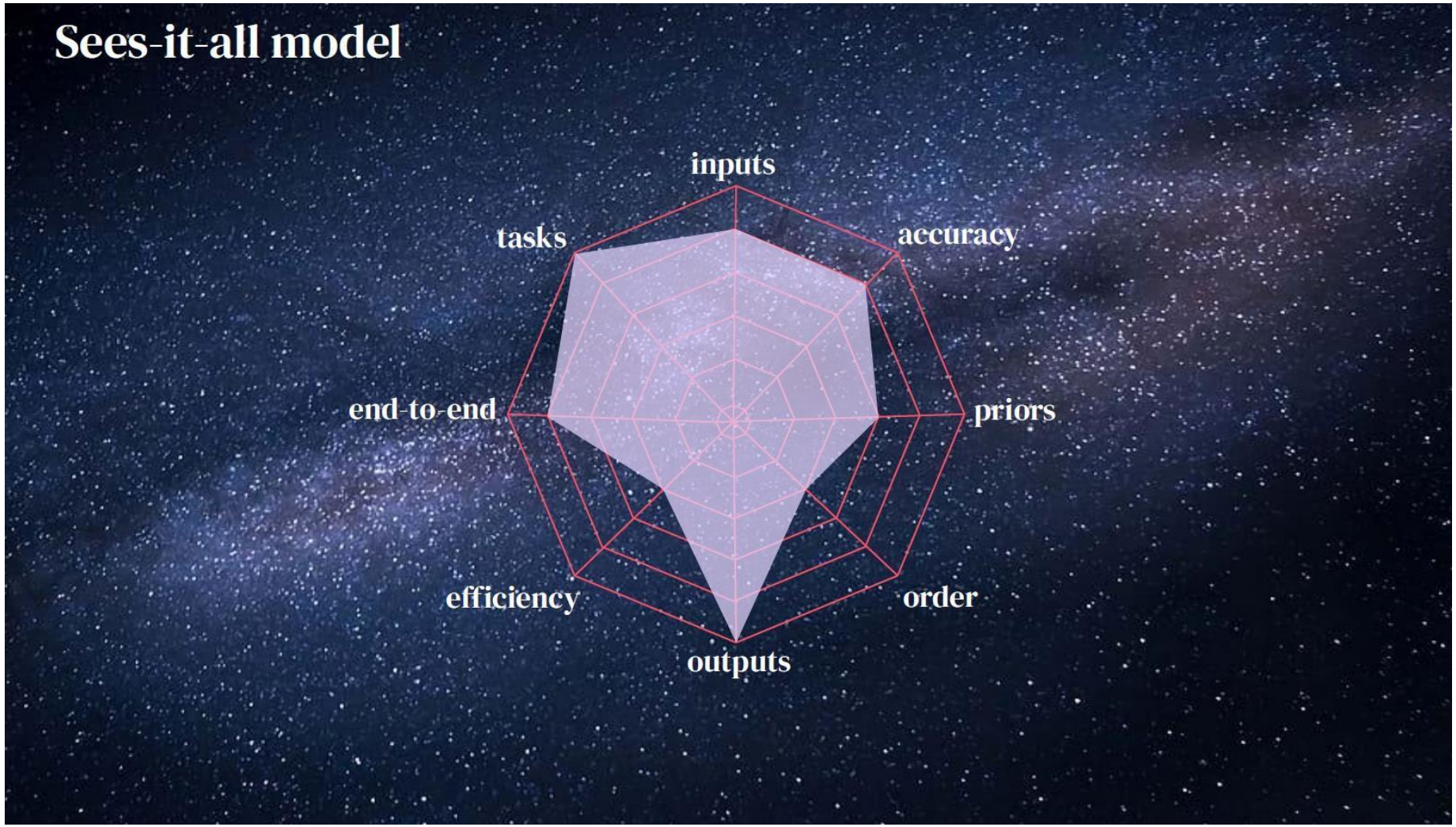
What is next?





Tasks Beyond Classification

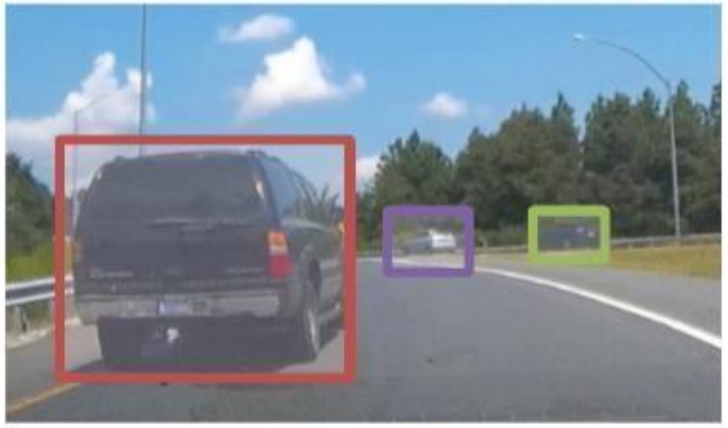
What is next?



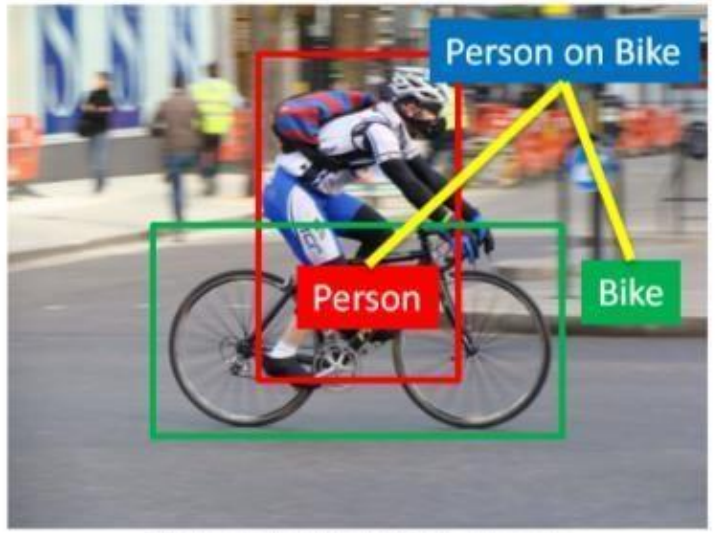
Tasks Beyond Classification

What is next?

There are many visual recognition problems that are related to image classification, such as object detection, image captioning



- Object detection
- Action classification
- Image captioning
- …



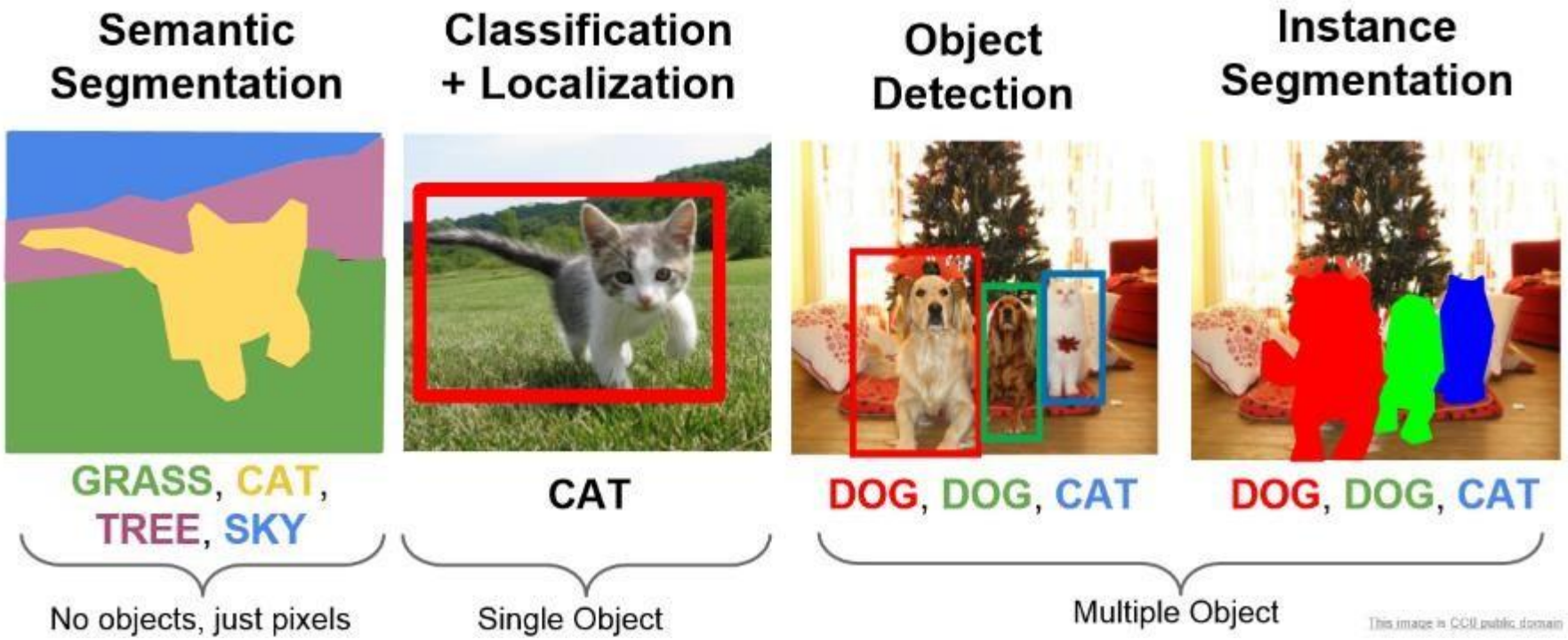
Tasks Beyond Classification

What is next?

Fast-forward to today: ConvNets are in other vision tasks

Object detection is related to other problems in computer vision

It can help in reducing the computational burden in many algorithms



Tasks Beyond Classification

What is next?

Fast-forward to today: ConvNets are in other vision tasks

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

2D Object Detection

DOG, DOG, CAT

Object categories + 2D bounding boxes

3D Object Detection

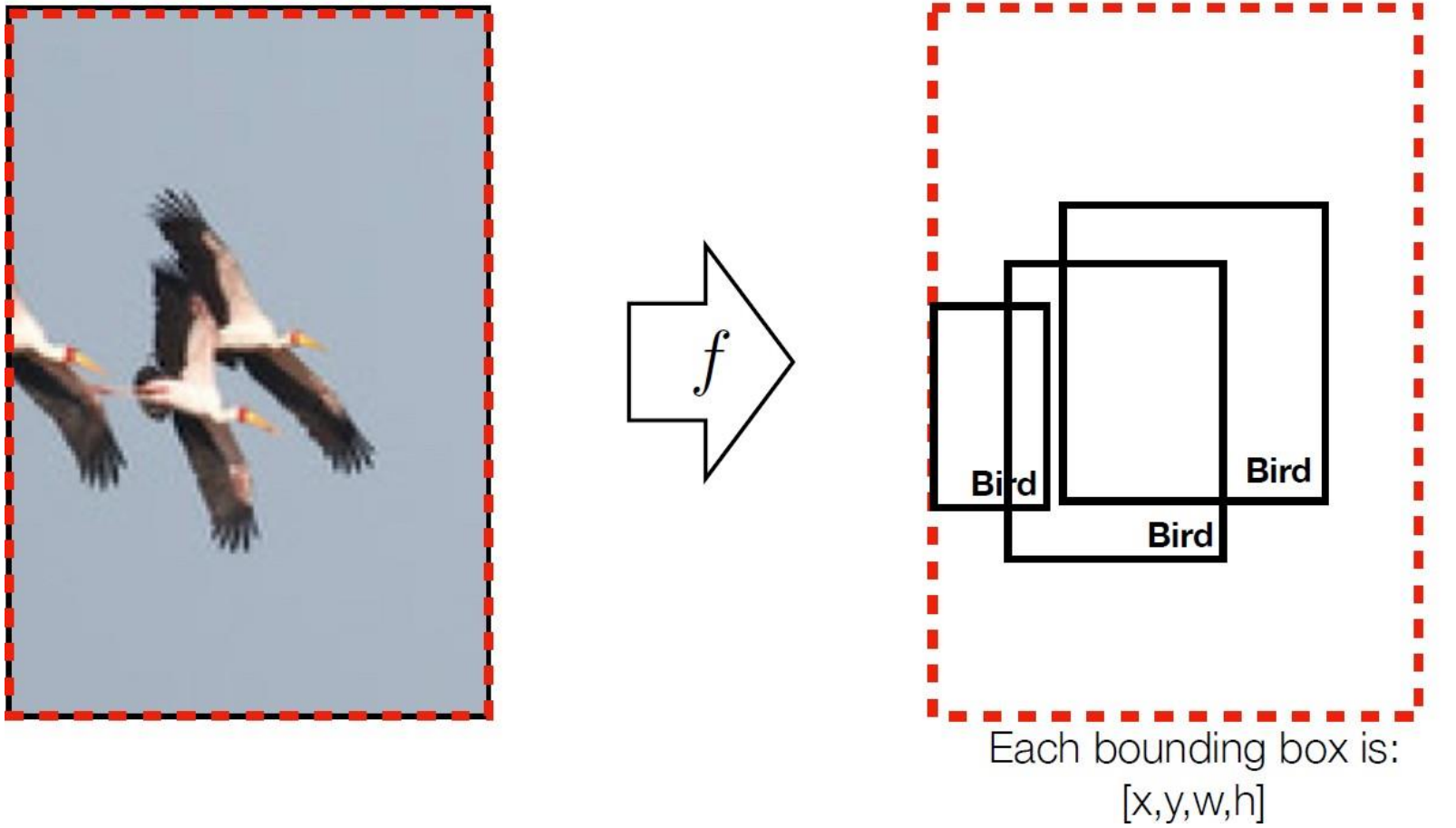
Car

Object categories + 3D bounding boxes

In this talk we will focus on the second case, describing the main progresses made in DL-based object detection in recent years

Object Detection

A very gentle introduction and overview



Object Detection

A very gentle introduction and overview

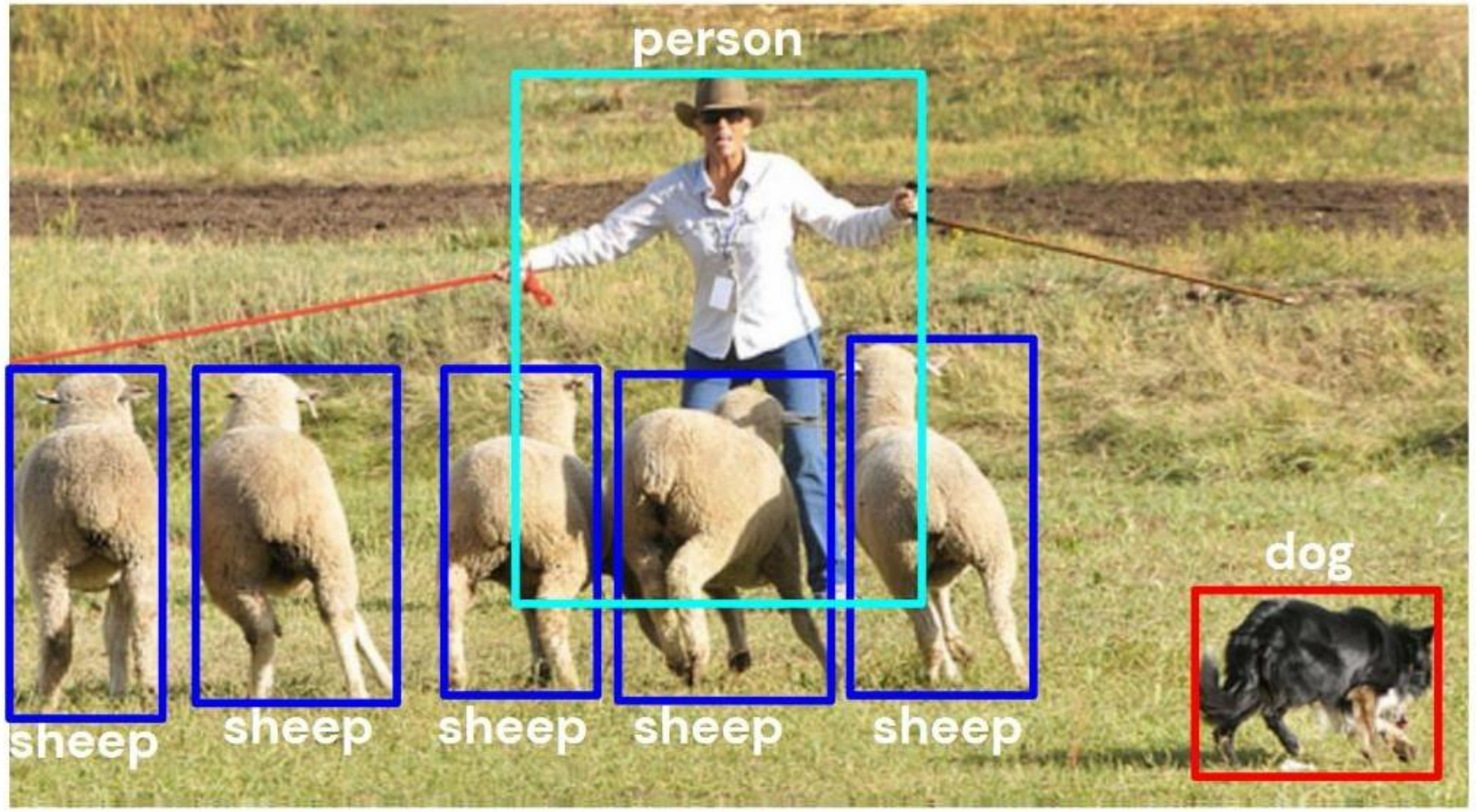
Multi-task problem

Classification and localisation

Image from COCO dataset – *Microsoft COCO: Common Objects in Context, Lin et al, 2014*

Object Detection

A very gentle introduction and overview



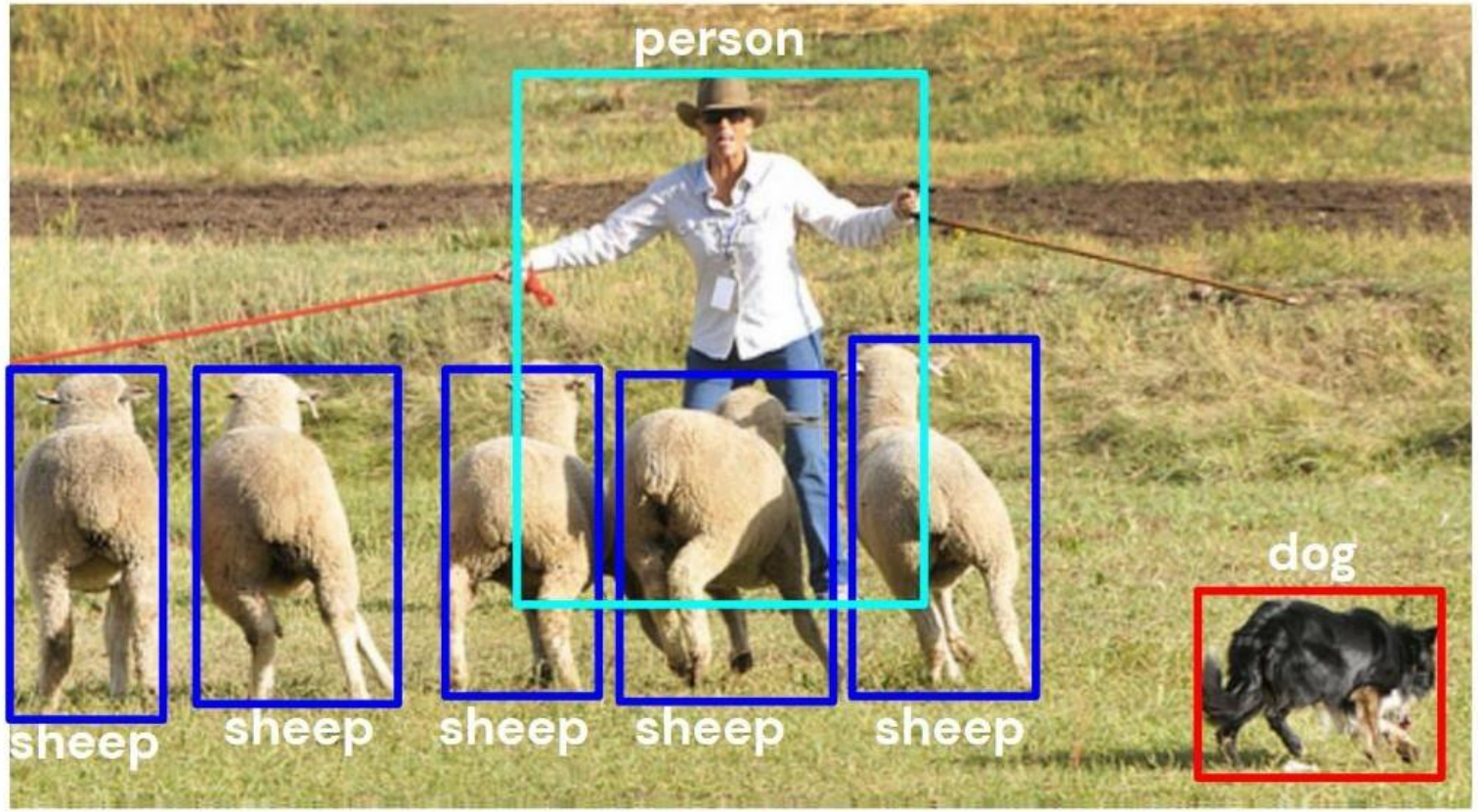
Object Detection

A very gentle introduction and overview

Dataset

$N_{\text{train}}, N_{\text{test}}$ samples

$$
\begin{aligned}
&\{\text{'image'} : p \in [0,1], H \times W \times 3, \\
&\ \text{'objects'} : \\
&\ [\\
&\ \{\text{'label'} : \text{one\_hot}(N), 1 \times N, \\
&\ \ \text{'bbox'} : (x_c, y_c, h, w) \in \mathbb{R}, 1 \times 4\}, \\
&\ \{\text{'label'} : \text{one\_hot}(N), 1 \times N, \\
&\ \ \text{'bbox'} : (x_c, y_c, h, w) \in \mathbb{R}, 1 \times 4\}, \\
&\ \vdots \\
&\ ]\}
\end{aligned}
$$



Object Detection

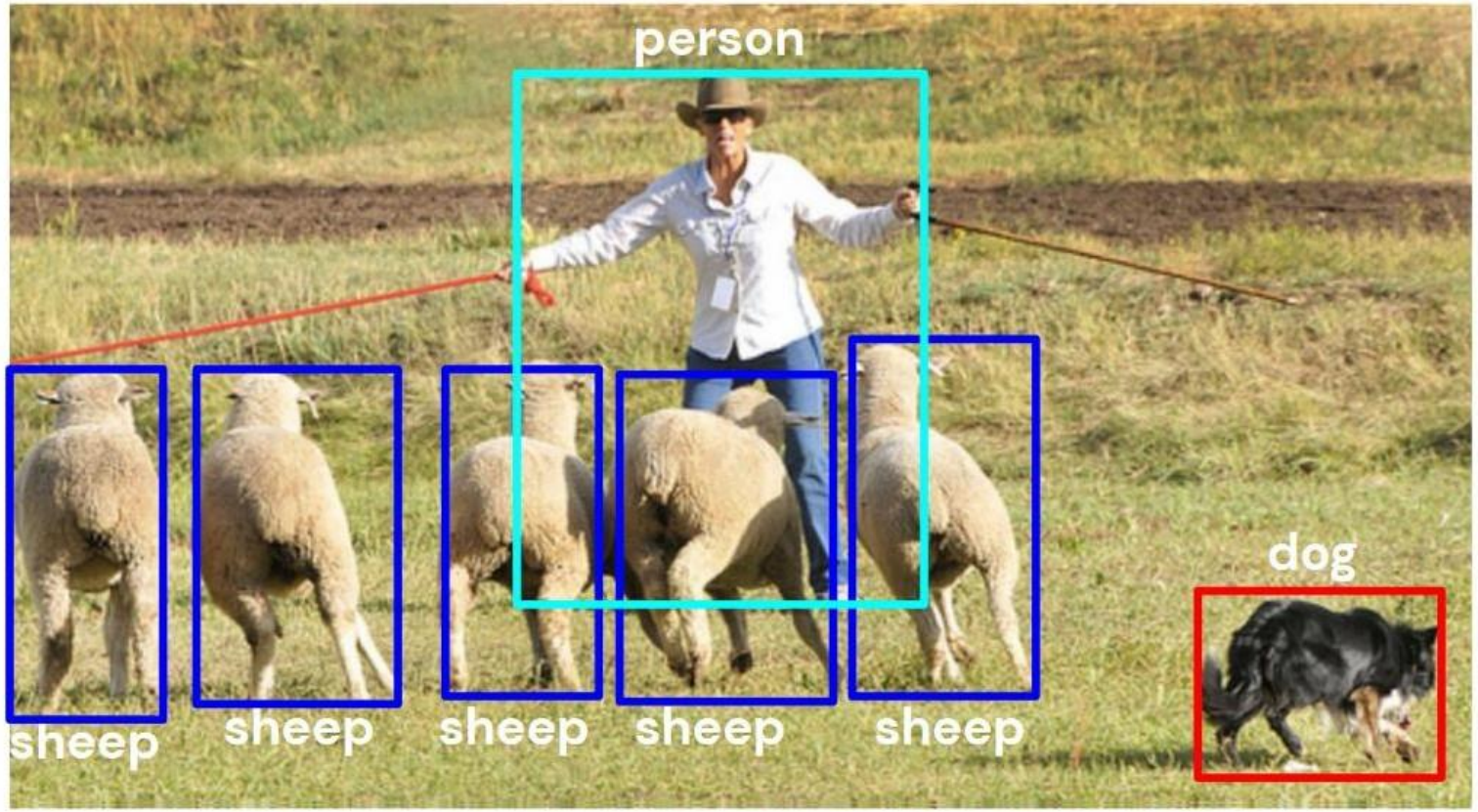
A very gentle introduction and overview

Dataset

$N_{\text{train}}, N_{\text{test}}$ samples

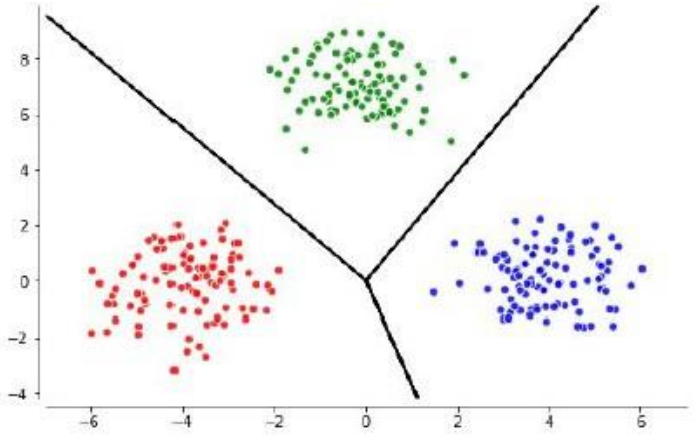
$$\{\text{'image'} : p \in [0,1], H \times W \times 3,$$
$$\text{'objects'} :$$
$$[$$
$$\{\text{'label'} : \text{one\_hot}(N), 1 \times N,$$
$$\text{'bbox'} : (x_c, y_c, h, w) \in \mathbb{R}, 1 \times 4\},$$
$$\{\text{'label'} : \text{one\_hot}(N), 1 \times N,$$
$$\text{'bbox'} : (x_c, y_c, h, w) \in \mathbb{R}, 1 \times 4\},$$
$$\vdots$$
$$]\}$$

How to learn to predict bbox coordinates?



Object Detection

Recap: Softmax +cross entropy for classification

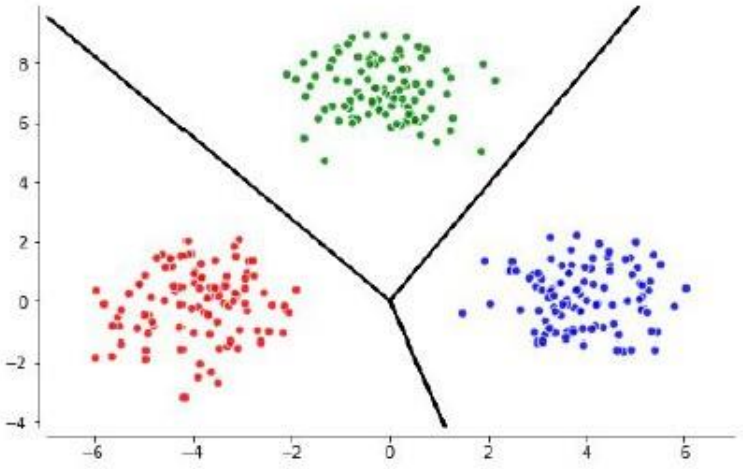


$$\ell_{\mathrm{CE}}(f_{\mathrm{sm}}(\mathbf{x}), \mathbf{t}) = -\sum_{j=1}^{k} \mathbf{t}_j \log[f_{\mathrm{sm}}(\mathbf{x}_j)] = -\sum_{j=1}^{k} \mathbf{t}_j [\mathbf{x}_j - \log \sum_{l=1}^{k} e^{\mathbf{x}_l}]$$

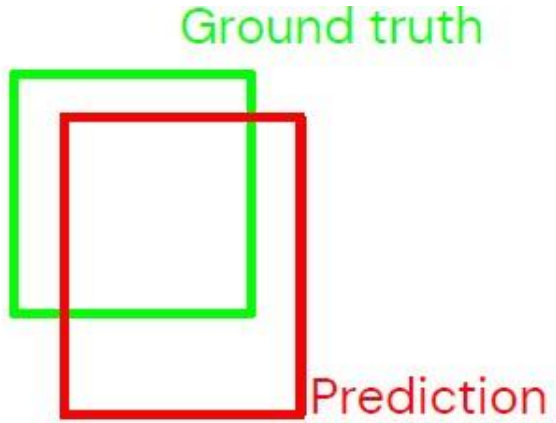
Assign data points to categories; output is discrete.

Object Detection

Bounding box prediction



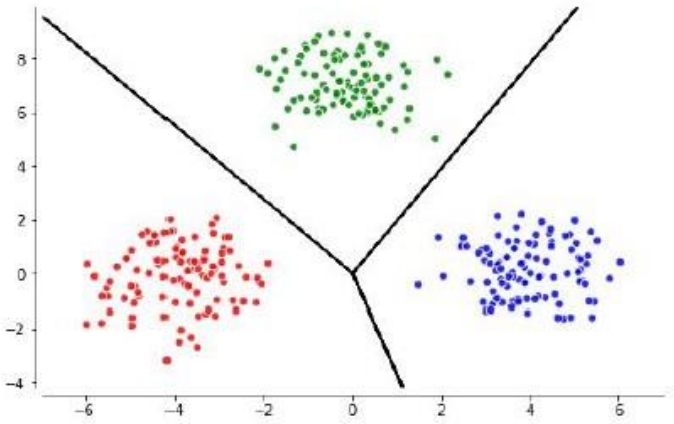
Classification



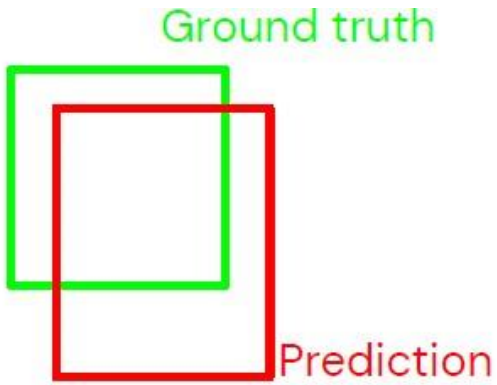
Mistakes are not quantifiable in classification; the data is not *ordered*.

Object Detection

Bounding box prediction



Classification



Regression

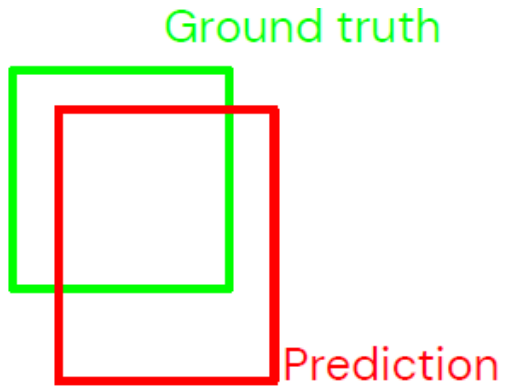
In classification, the output is discrete*, in regression the output is continuous.

*in classification the output is continuous too, but it represents the probability distribution over the possible classes

Object Detection

Quadratic loss for regression

$$\ell_2(\mathbf{x}, \mathbf{t}) = \|\mathbf{t} - \mathbf{x}\|^2$$



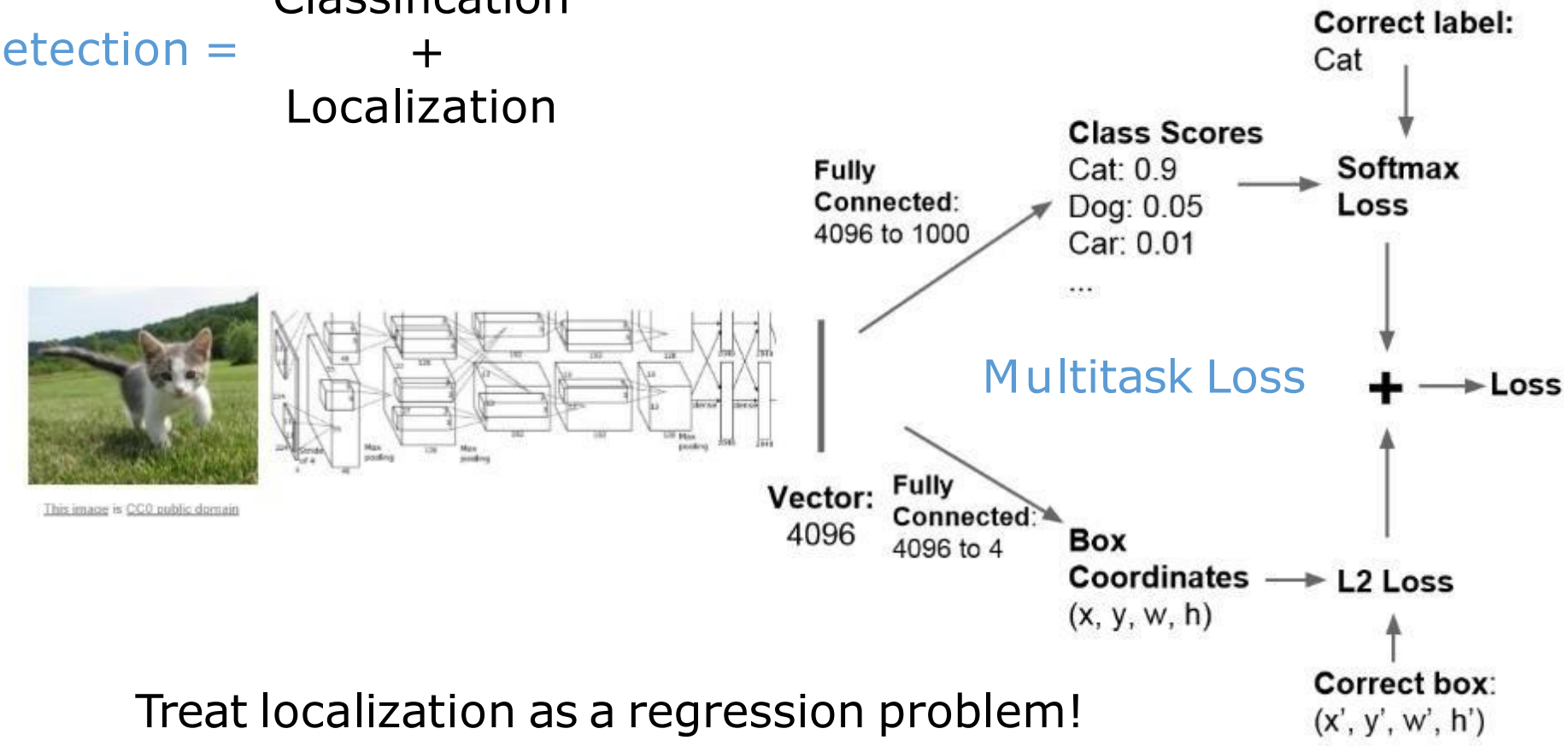
Minimise the mean squared error over samples.

Object Detection

Quadratic loss for regression

Fast-forward to today: Object Detection Ⅱ

Detection = Classification + Localization



Treat localization as a regression problem!

Object Detection

Summary: Classification vs Regression

Property	Classification	Regression
Basic	map inputs to predefined classes	map inputs to continuous values
Output	discrete values	continuous values
Nature of the data	unordered data	ordered data
Algorithms	logistic regression, decision trees, neural networks	linear regression, neural networks

Object Detection

Quadratic loss for regression

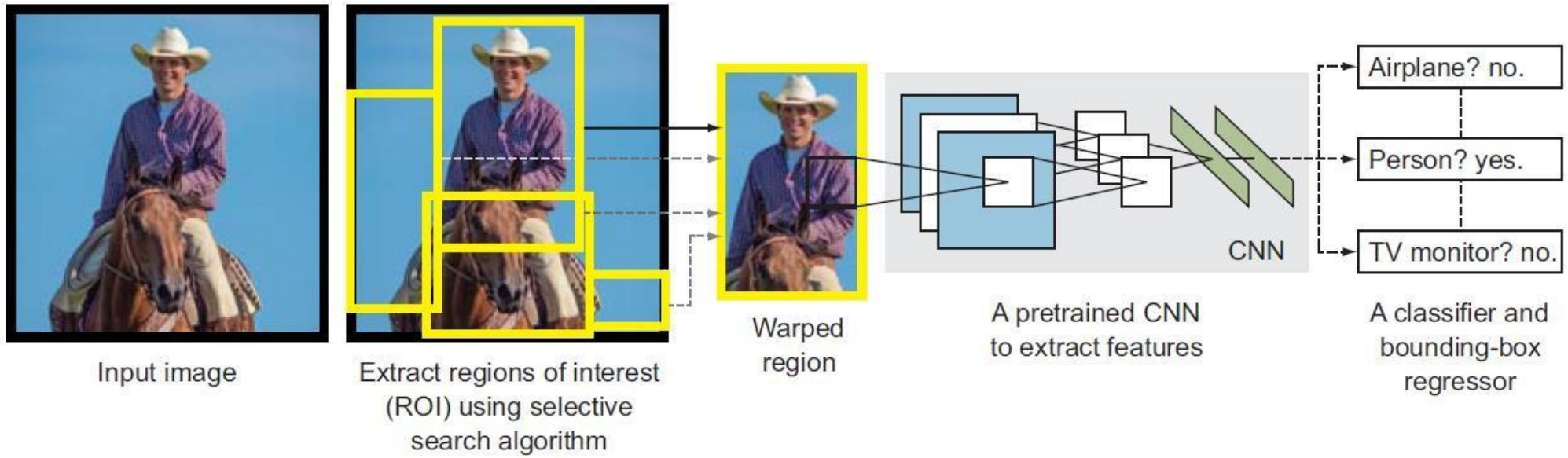
Fast-forward to today: ConvNets are in other vision tasks

Object Detection as Classification

↓

Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Problem:

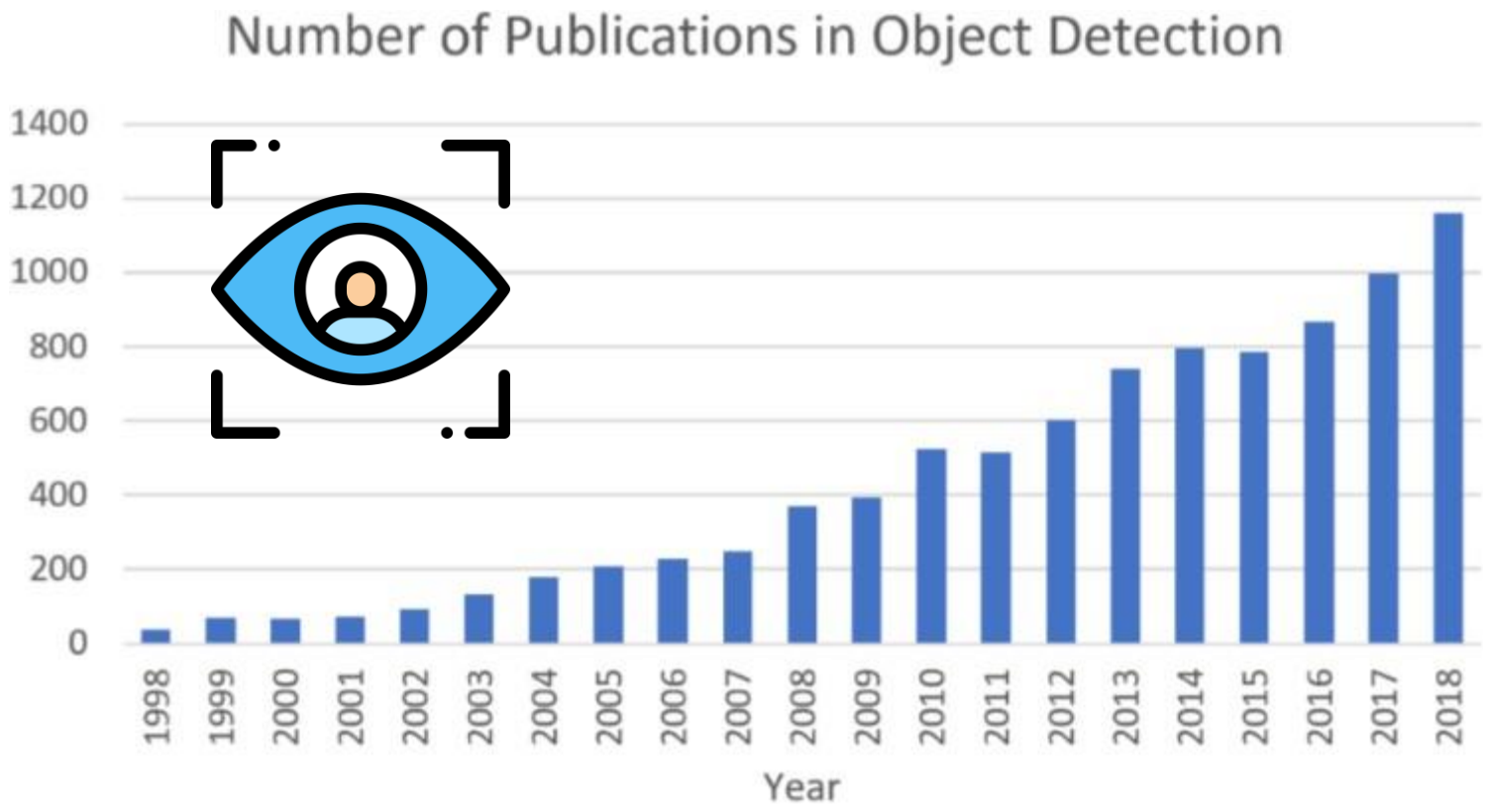
Need to apply CNN to huge nb of locations, scales, and aspect ratios

very computationally expensive!

Introduction:

Object Detection in the last 20 years

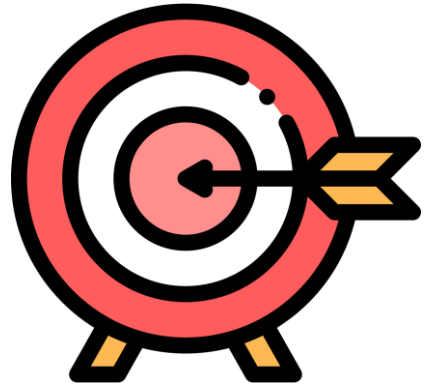
Basics: Object detection I



Data from Google scholar advanced search: allintitle: "object detection" AND "detecting objects".

Introduction:

Object Detection in the last 20 years



Basics: Object detection II

Primarily interested in

1. Highly structured objects (cars, faces,)

2. Articulated objects (humans, horses, dogs)

Bounding boxes used for coarsely defining the spatial location of an object and for evaluation purposes

bicycle, person

bird

bottle, sofa, person

bus, car, person

Can be of two flavors: specific instance or specific categories

Introduction:

Object Detection in the last 20 years

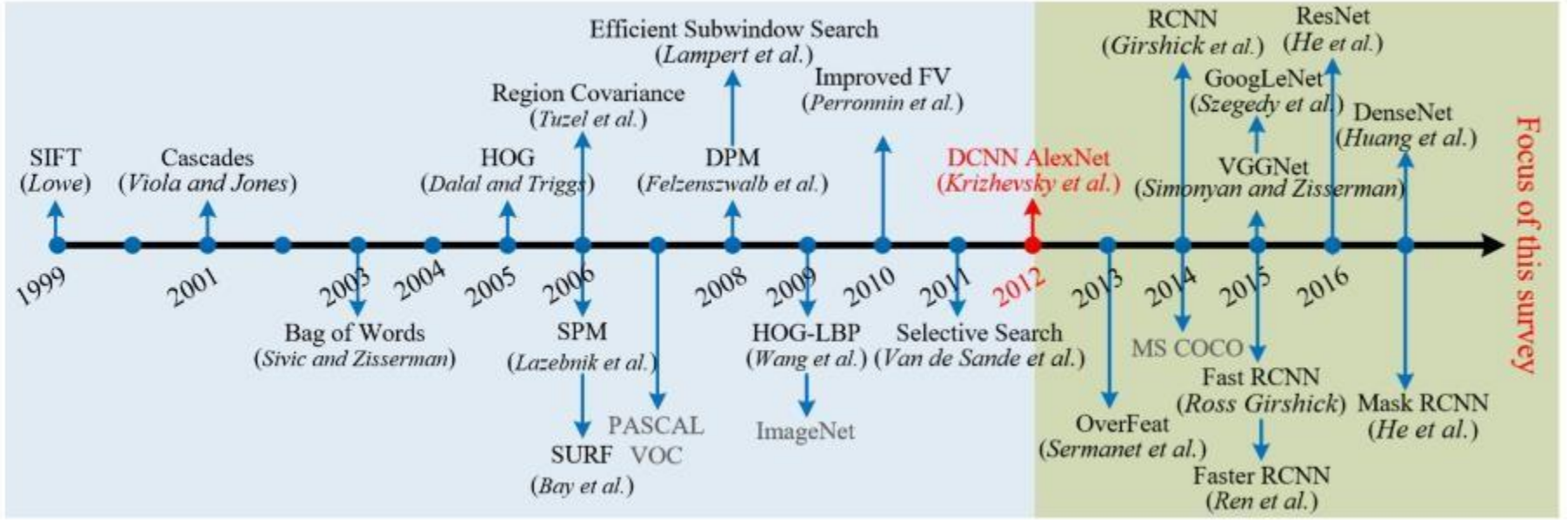
Basics: Object detection Ⅲ

Traditional computer vision: a single category (faces, pedestrians)

In recent years: more challenging goal of generic object detection

Traditional approaches struggled at classification, we saw

Deep Learning-based detectors have taken since 2012



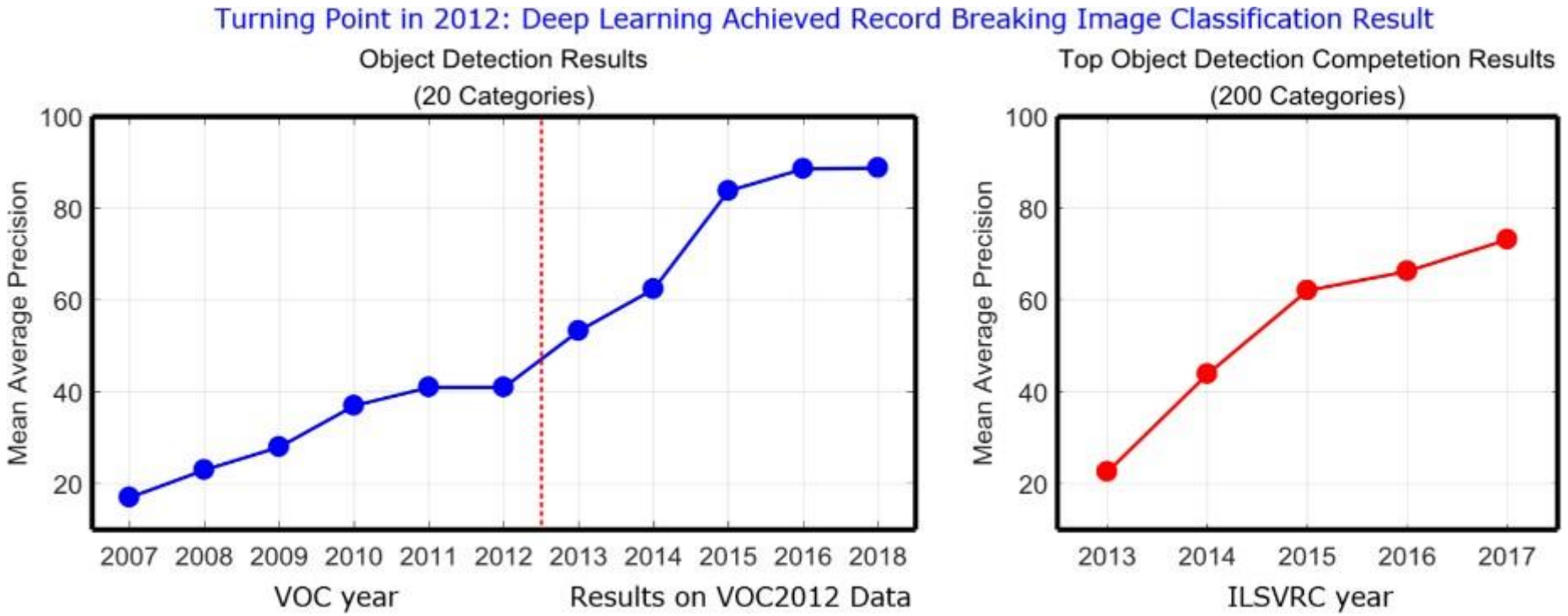
Introduction:

Object Detection in the last 20 years

Basics: Object detection IV

Object detection is challenging task, but DL methods have shown great promise and are being widely deployed

Due in large part to increasingly large training sets



Introduction:

Object Detection in the last 20 years

Basics: Object detection V

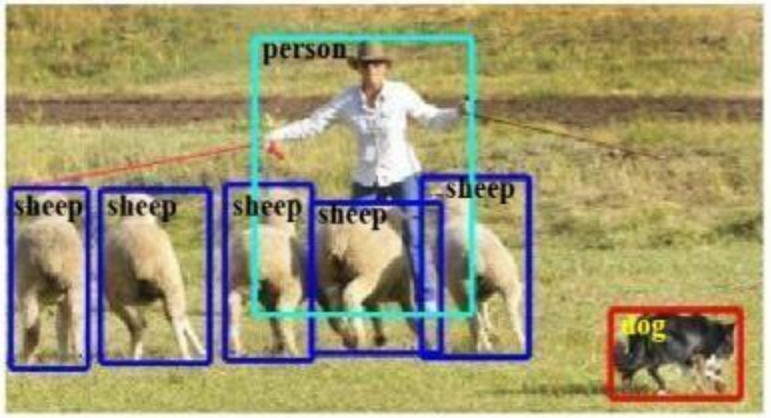
As we saw, the internet has been a major driving force here, both for gathering, crowdsourcing and hosting large training datasets

Introduction:

Object Detection in the last 20 years

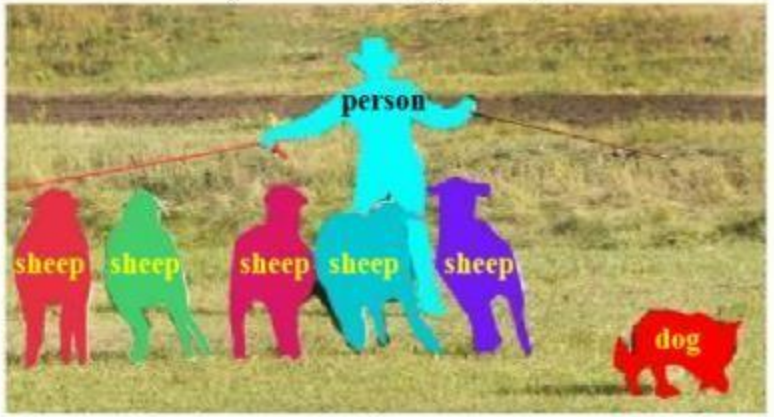
Basics: Object detection VI

(a) Object Classification



(b) Generic Object Detection (Bounding Box)

(c) Semantic Segmentation

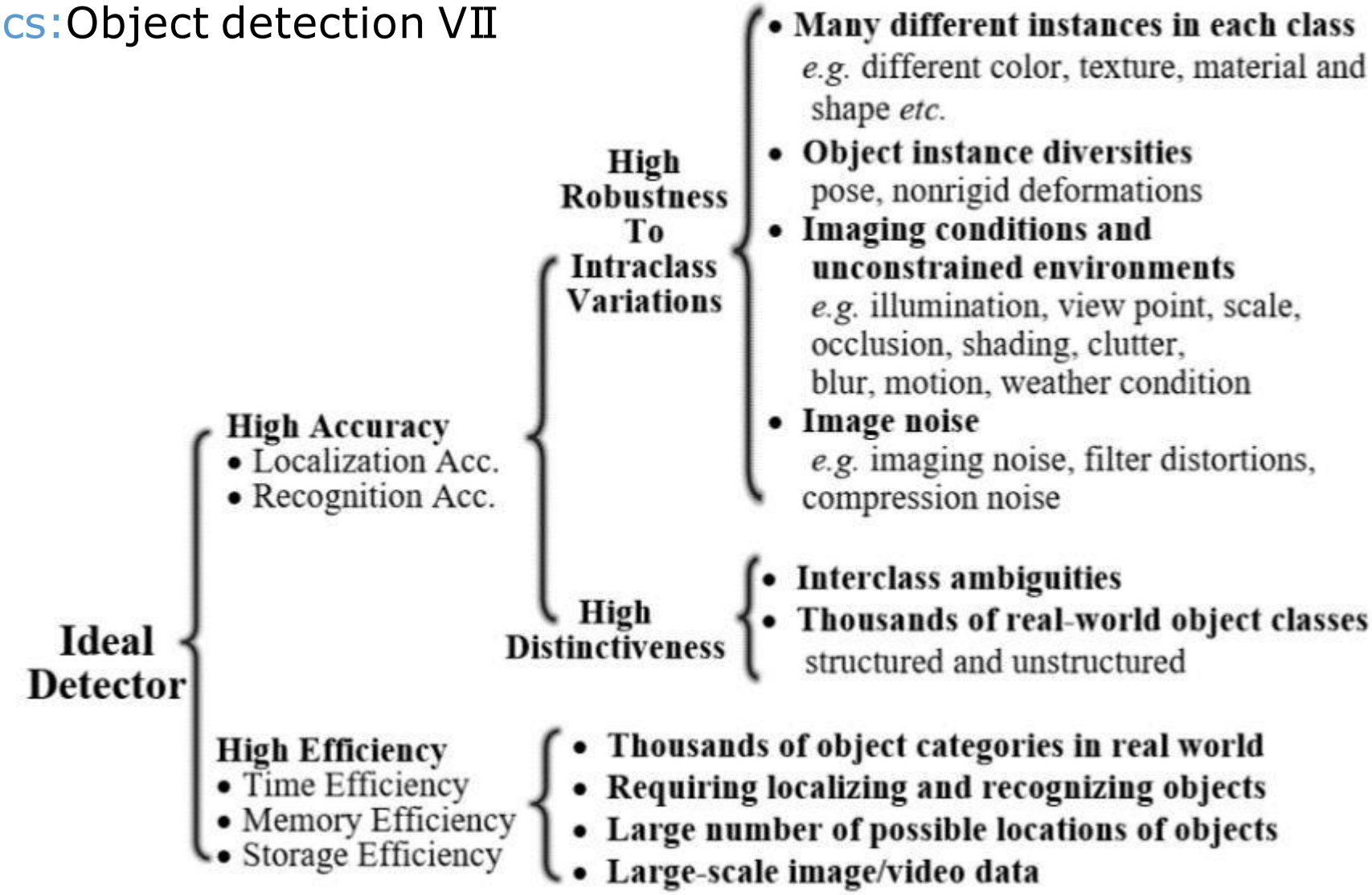


(d) Object Instance Segmetation

Introduction:

Object Detection in the last 20 years

Basics: Object detection VII



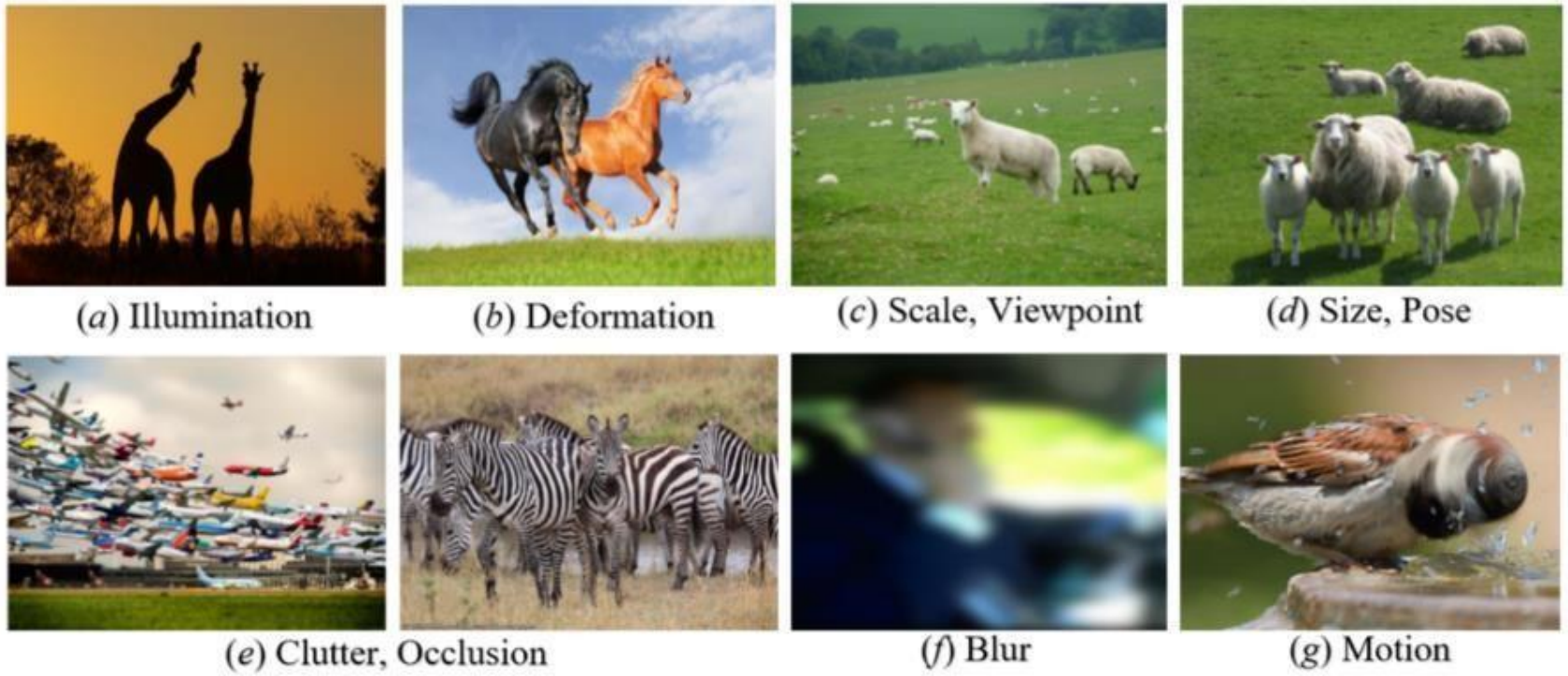
Introduction:

Object Detection in the last 20 years

Challenges: Object detection I

In contrast to just image or object classification, object detection is usually expected to run on real-time video

This requirement introduces several tough challenges



(a) Illumination (b) Deformation (c) Scale, Viewpoint (d) Size, Pose

(e) Clutter, Occlusion (f) Blur (g) Motion

Introduction:

Object Detection in the last 20 years

Challenges: Object detection II

In addition to those, there are accuracy related challenges due to

1) Vast range of intraclass variations (intrinsic factors, imaging cond.)

2) A huge number of object categories (discrimination power)

(*h*) Different instances of the "chair" category

(*i*) Small Interclass Variations: four different categories

Introduction:

Object Detection in the last 20 years

Introduction:

Object Detection in the last 20 years

Challenges: Efficiency

The exponentially increasing number of images calls for efficient and scalable detectors for applications on constrained devices

However mobile/wearable devices have limited computational capabilities and storage space, in which case an efficient object detector is critical.

For efficiency, the challenges stem from the need to localize and recognize all object instances of very large number of object categories, and the very large number of possible locations and scales within a single image

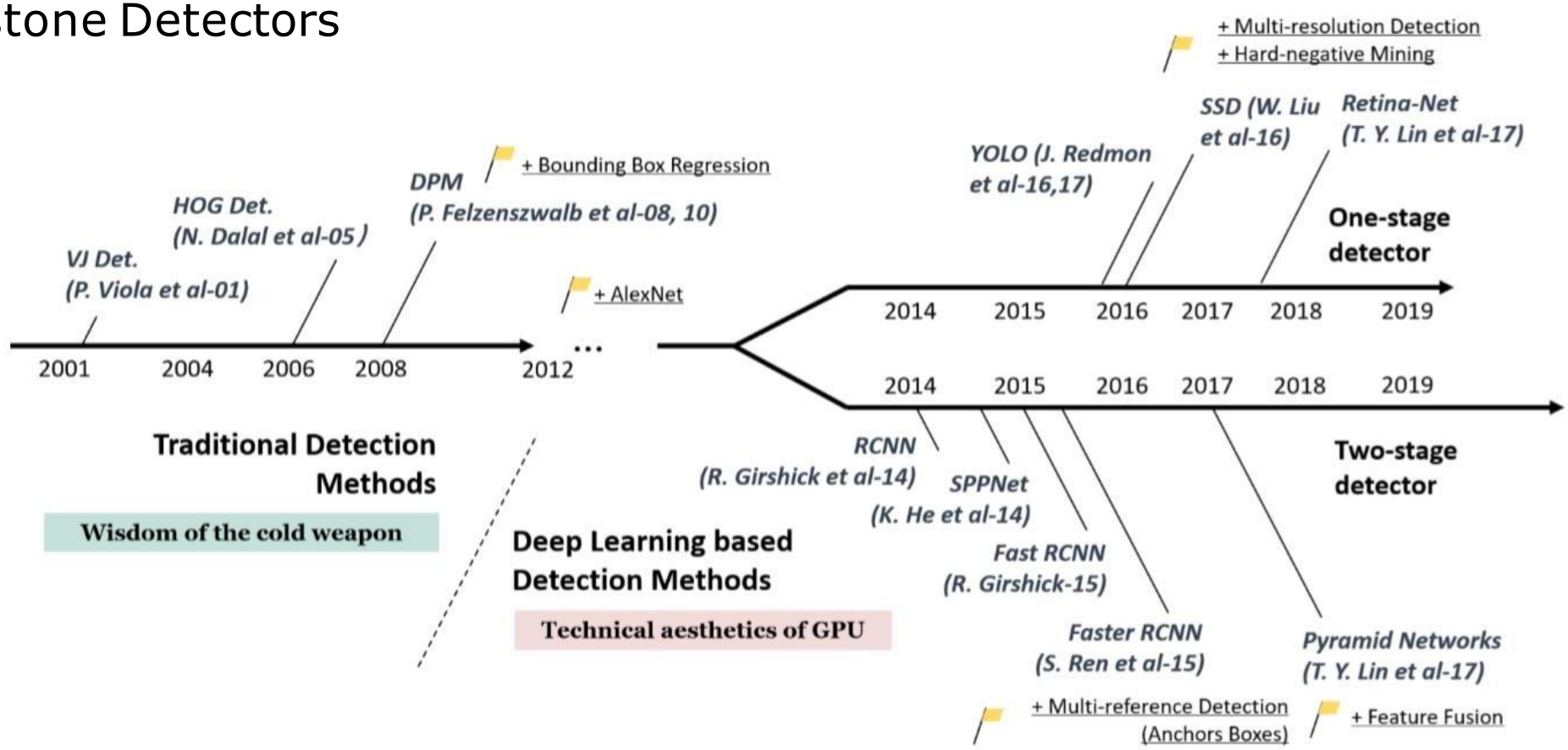
A further challenge is that of scalability: A detector should be able to handle unseen objects, unknown situations, and rapidly increasing image data.

Technical evolution in generic object detection

Evolution of deep-learning based object detectors

A Roadmap of Object Detection

Milestone Detectors



Technical evolution in generic object detection

Early naive approaches I

Object detection (before 2000) did not follow a unified detection philosophy like sliding window detection

Detectors at that time were usually designed based on low-level and mid-level vision as follows

Components, shapes and edges

Recognition-by-components: long time a core idea of image recognition and object detection

Initial approaches framed the problem as a measurement of similarity between the object components, shapes and contours

Despite some promising results, these approaches were not scalable and machine learning-based methods began to prosper.

Technical evolution in generic object detection

Early naive approaches II

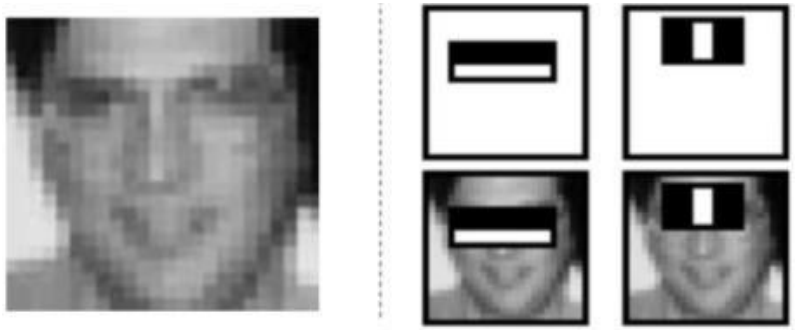
Machine Learning-based detection

Multiple periods: statistical models of appearance (before 1998), wavelet feature (98-2005) & gradient-based (05-2012) representations

Statistical: Eigenfaces
Used for face detection in real time

(a) Eigenfaces (M. Turk et al, 1991)

Wavelet: Haar Transform
Used by Viola-Jones for face and
Other applications in real-time



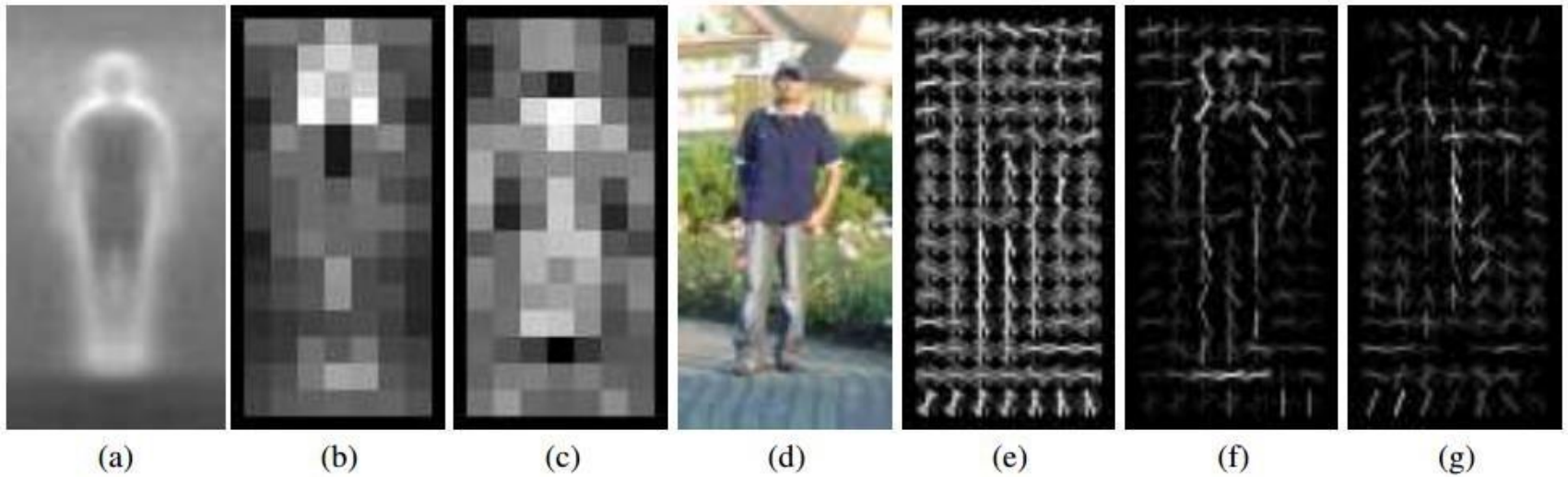
(d) Haar wavelet (P. Viola et al, 2001)

Gradient Based: HoG
Mainly for pedestrian detection

Technical evolution in generic object detection

Early naive approaches: **HOG**

Pedestrians



Histograms of Oriented Gradients for Human Detection, Dalal and Triggs, CVPR 2005

AP ~77%

More sophisticated methods: AP ~90%

(a) average gradient image over training examples
(b) each "pixel" shows max positive SVM weight in the block centered on that pixel
(c) same as (b) for negative SVM weights
(d) test image
(e) its R-HOG descriptor

Technical evolution in generic object detection

Early naive approaches: **HOG**

1. Compute gradients in the region to be described
2. Put them in bins according to orientation
3. Group the cells into large blocks
4. Normalize each block
5. Train classifiers to decide if these are parts of a human

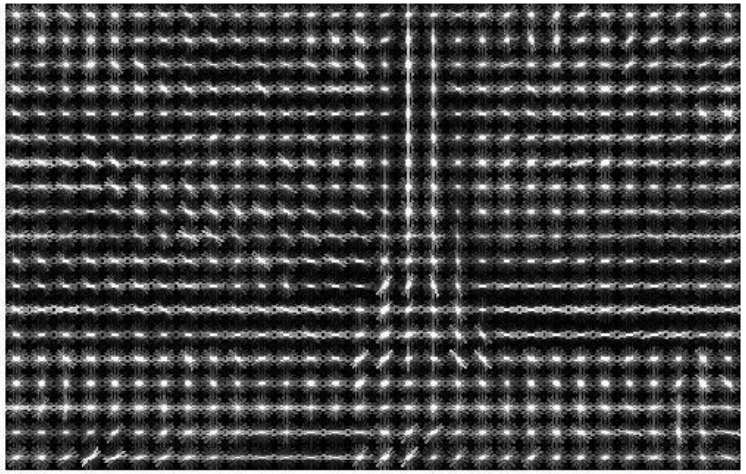


Image credit: N. Snavely

Technical evolution in generic object detection

Early naive approaches: **HOG**

Details

- Gradients

  [-1 0 1] and $[-1\ 0\ 1]^T$ were good enough filters.

- Cell Histograms

  Each pixel within the cell casts a weighted vote for an orientation-based histogram channel based on the values found in the gradient computation. (9 channels worked)

- Blocks

  Group the cells together into larger blocks, either R-HOG blocks (rectangular) or **C-HOG** blocks (circular).

Technical evolution in generic object detection

Early naive approaches: **HOG**

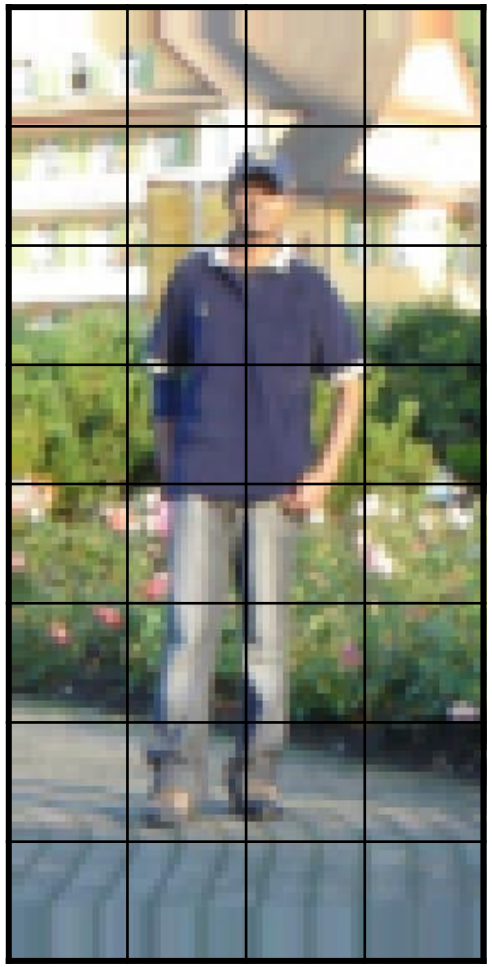
Example: Dalal-Triggs pedestrian detector

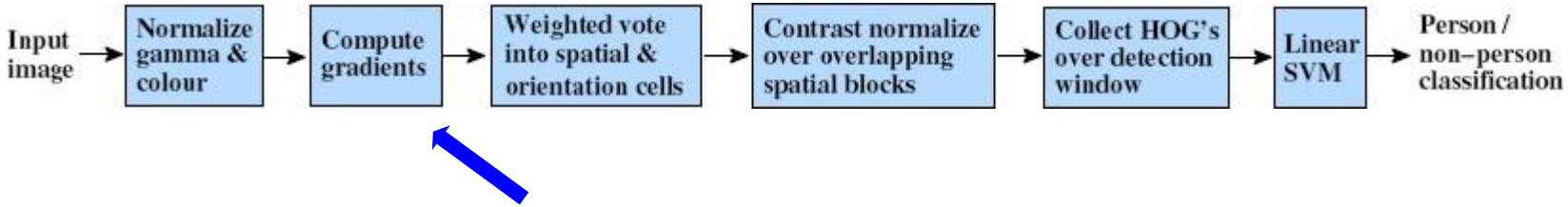
1. Extract fixed-sized (64x128 pixel) window at each position and scale
2. Compute HOG (histogram of gradient) features within each window
3. Score the window with a linear SVM classifier
4. Perform non-maxima suppression to remove overlapping detections with lower scores

Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05

Technical evolution in generic object detection

Early naive approaches: **HOG**





-1	0	1

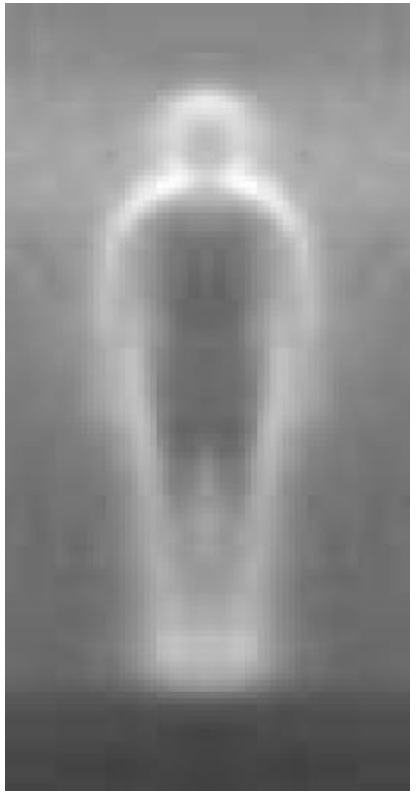
centered

-1	1

uncentered

1	-8	0	8	-1

cubic-corrected



0	1
-1	0

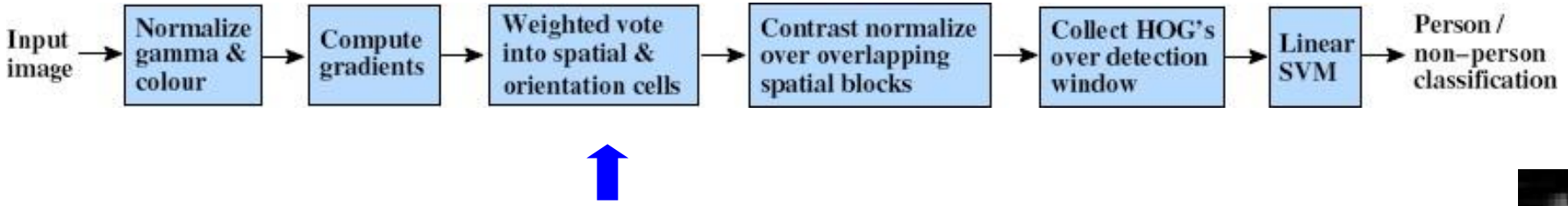
diagonal

-1	0	1
-2	0	2
-1	0	1

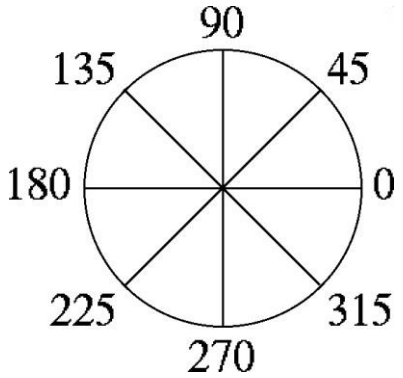
Sobel

Technical evolution in generic object detection

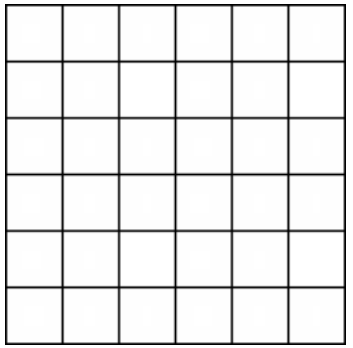
Early naive approaches: **HOG**

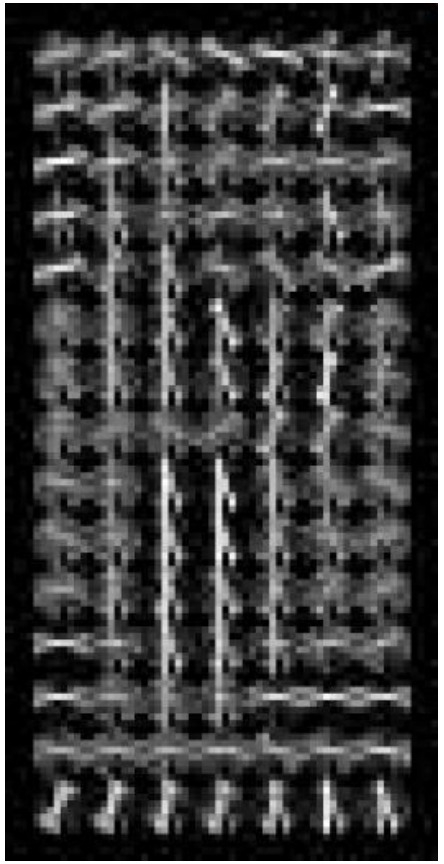


Orientation: 9 bins (for unsigned angles)



Histograms in 8x8 pixel cells



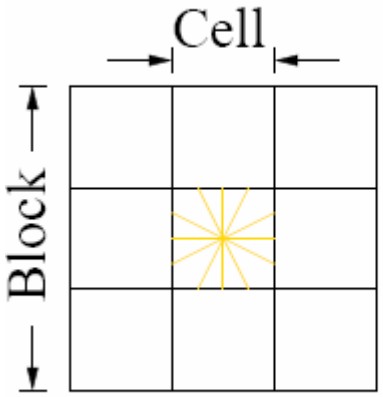


Technical evolution in generic object detection

Early naive approaches: **HOG**

R-HOG

Normalize with respect to surrounding cells

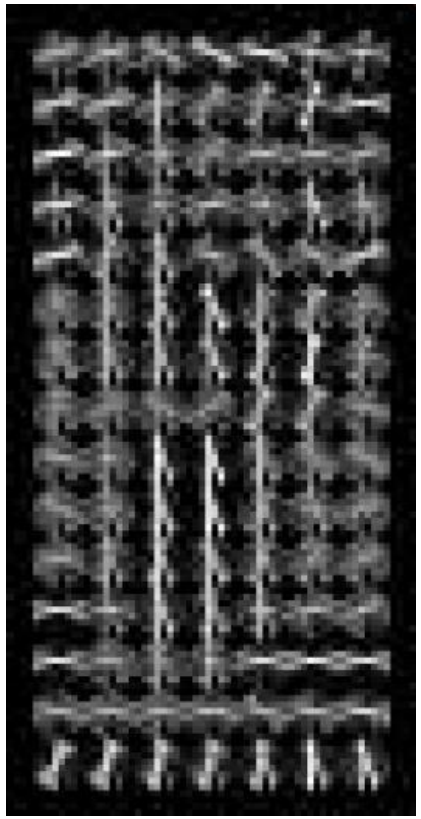


$$L2-norm: v \longrightarrow v/\sqrt{||v||_2^2+\epsilon^2}$$

Technical evolution in generic object detection

Early naive approaches: **HOG**

X=



features = 15 x 7 x 9 x 4 = 3780

orientations

cells

normalizations by neighboring cells

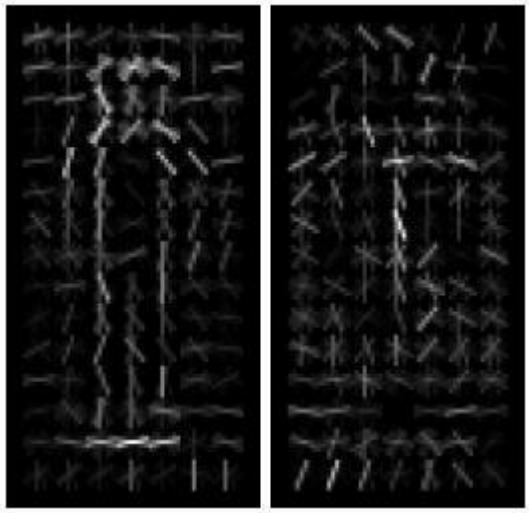
Technical evolution in generic object detection

Early naive approaches: **HOG**

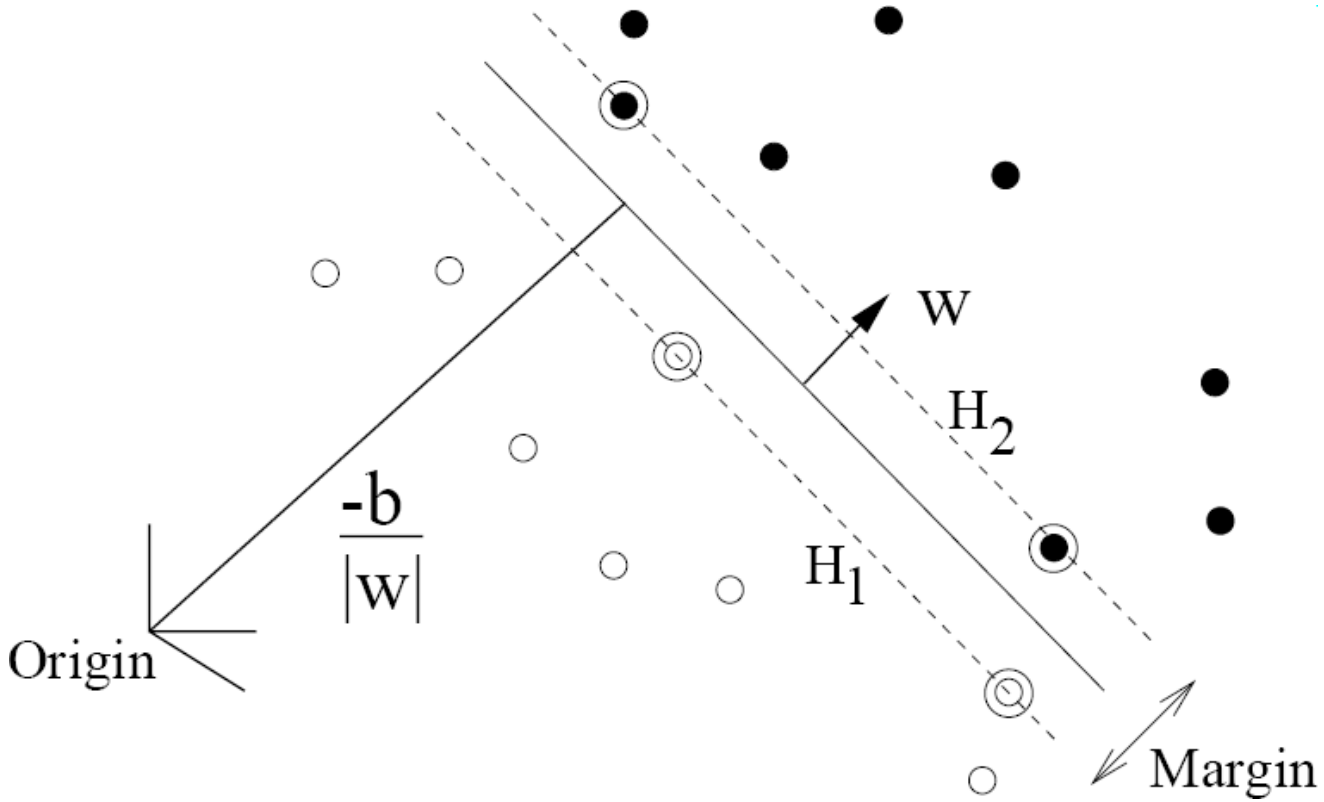
Training set

Technical evolution in generic object detection

Early naive approaches: **HOG**

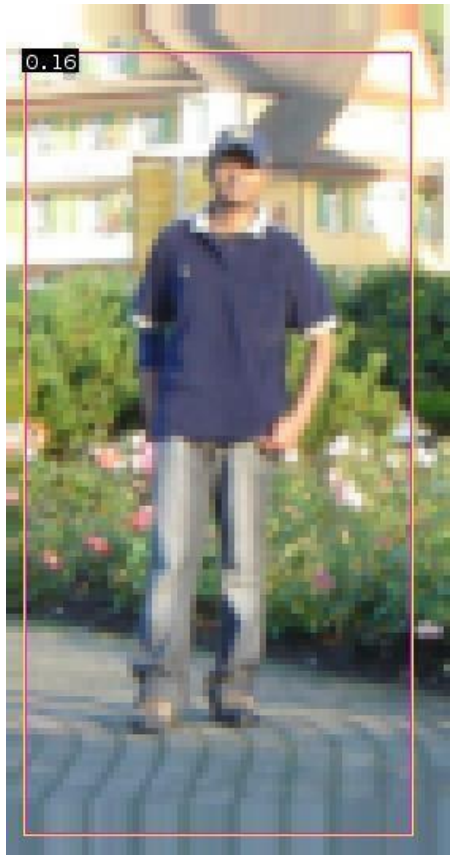


pos w neg w



Technical evolution in generic object detection

Early naive approaches: **HOG**



$$0.16 = w^T x - b$$

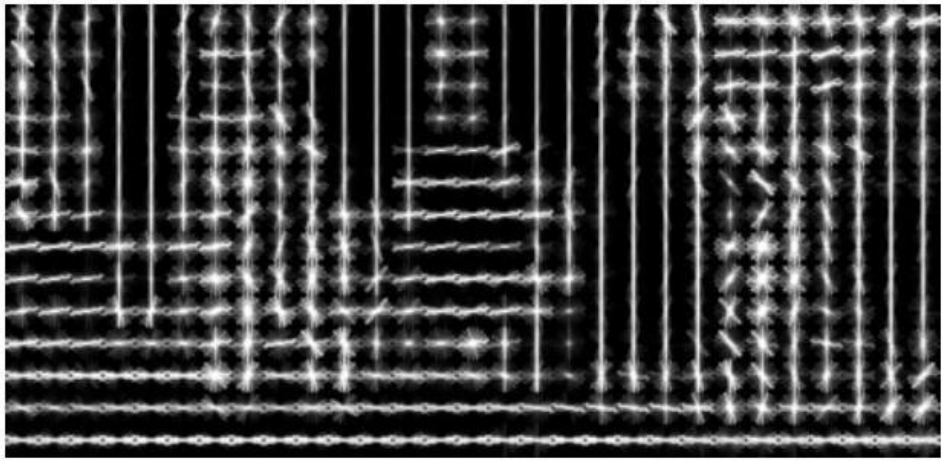
$$sign(0.16) = 1$$

➡ pedestrian

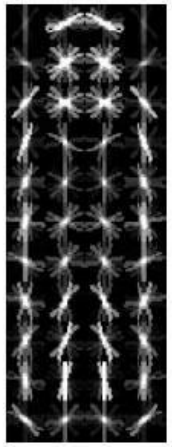
Technical evolution in generic object detection

Early naive approaches: **HOG**

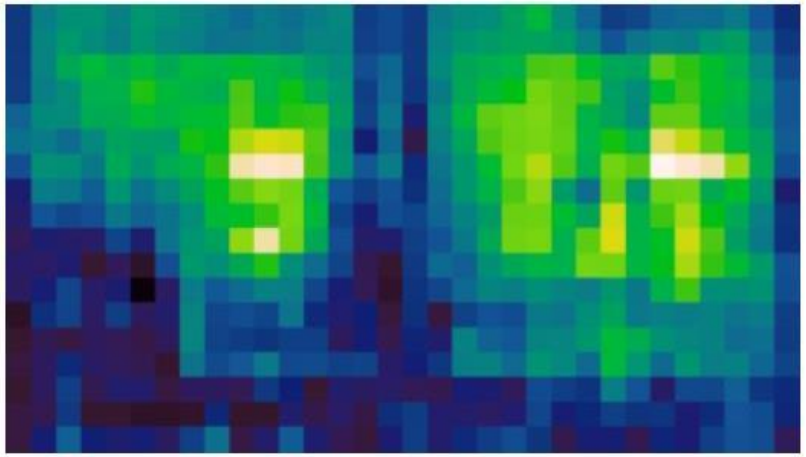
HOG feature map



Template



Detector response map



Technical evolution in generic object detection

Early naive approaches: **HOG**

Detection examples

Technical evolution in generic object detection

Sliding window…. The best method?

Technical evolution in generic object detection

Sliding window…. The best method?

Each window is separately classified

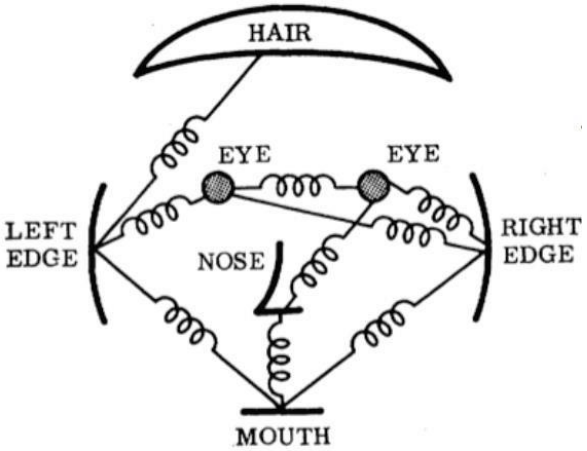
Technical evolution in generic object detection

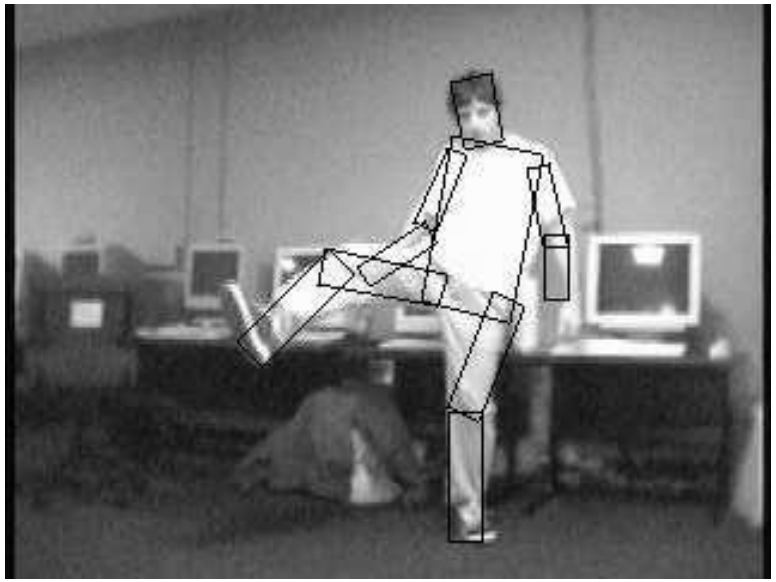
Deformable Parts Model

- Takes the idea a little further
- Instead of one rigid HOG model, we have multiple HOG models in a spatial arrangement
- One root part to find first and multiple other parts in a tree structure.

Articulated parts model

- Object is configuration of parts
- Each part is detectable



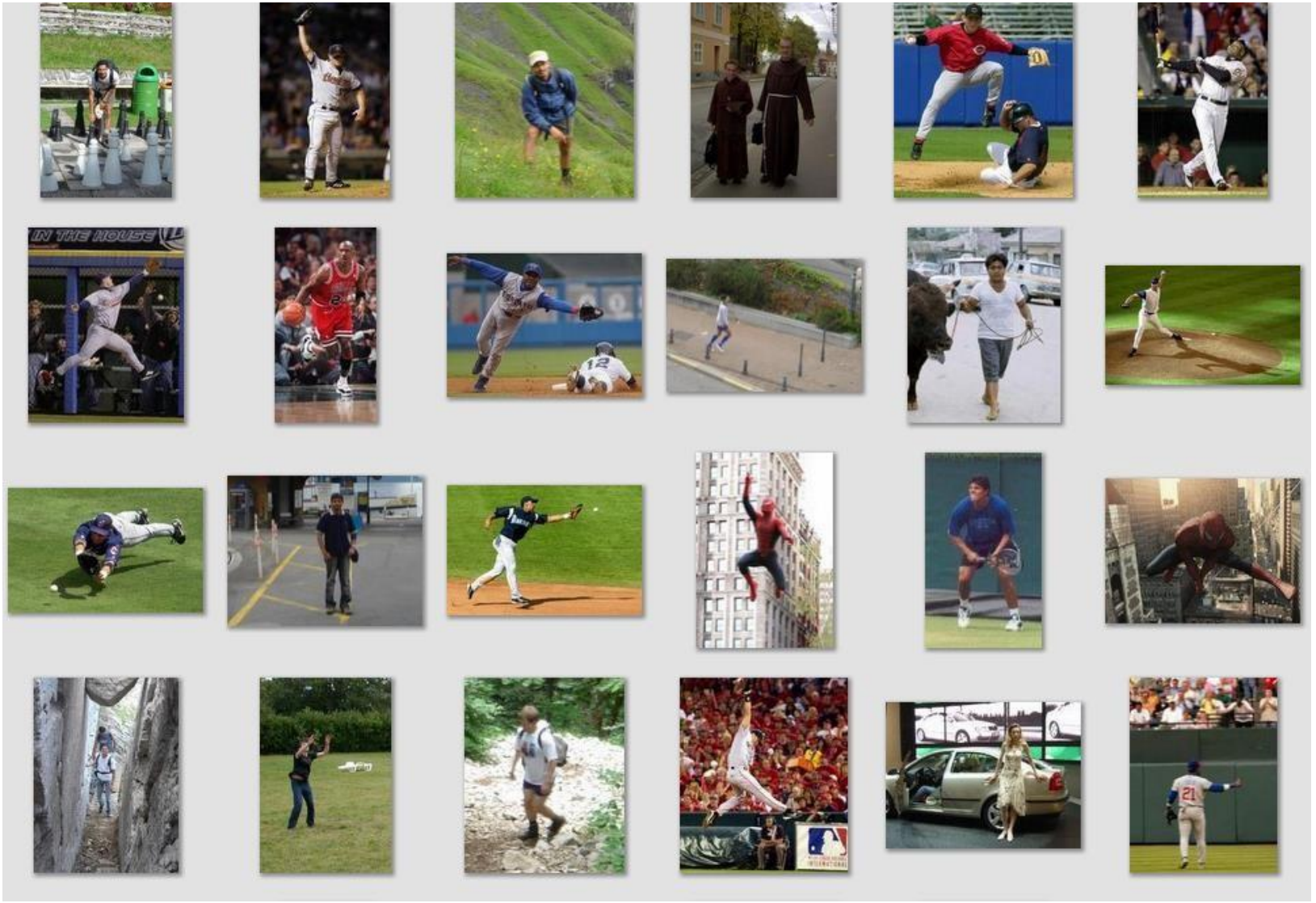


Technical evolution in generic object detection

Deformable Parts Model

Technical evolution in generic object detection

Deformable Parts Model

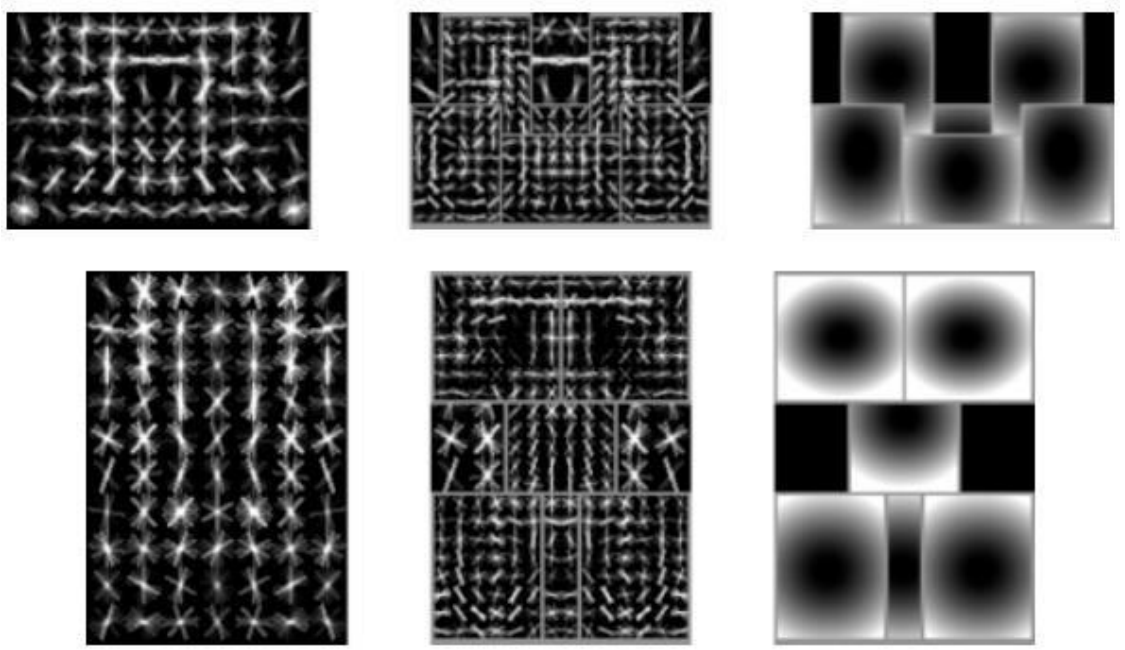


Technical evolution in generic object detection

Deformable Parts Model

Detections

Template Visualization



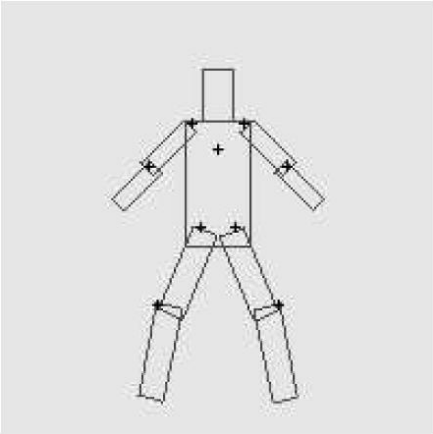
root filters
coarse resolution

part filters
finer resolution

deformation
models

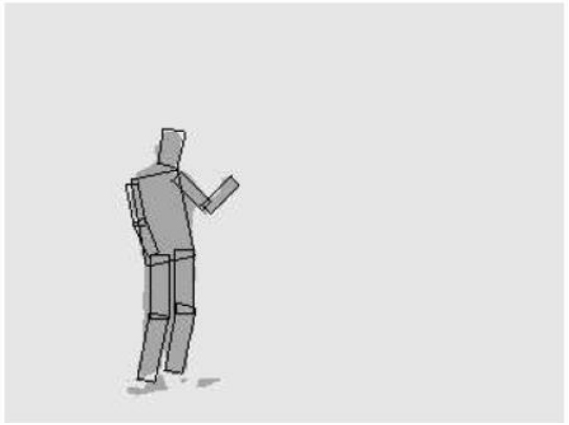
Technical evolution in generic object detection

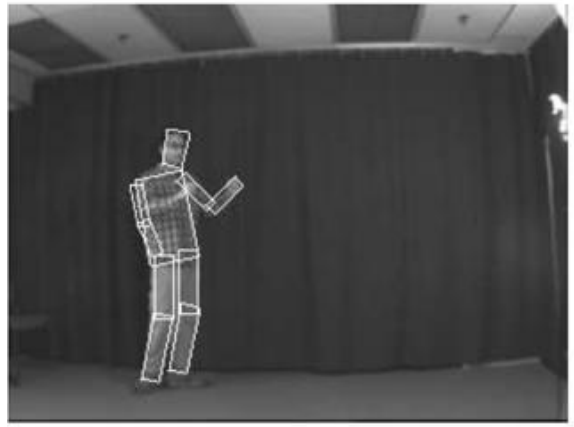
Deformable Parts Model



$$P(L|I,\theta) \propto \left(\prod_{i=1}^{n} p(I|l_i, u_i) \prod_{(v_i,v_j)\in E} p(l_i, l_j|c_{ij}) \right)$$

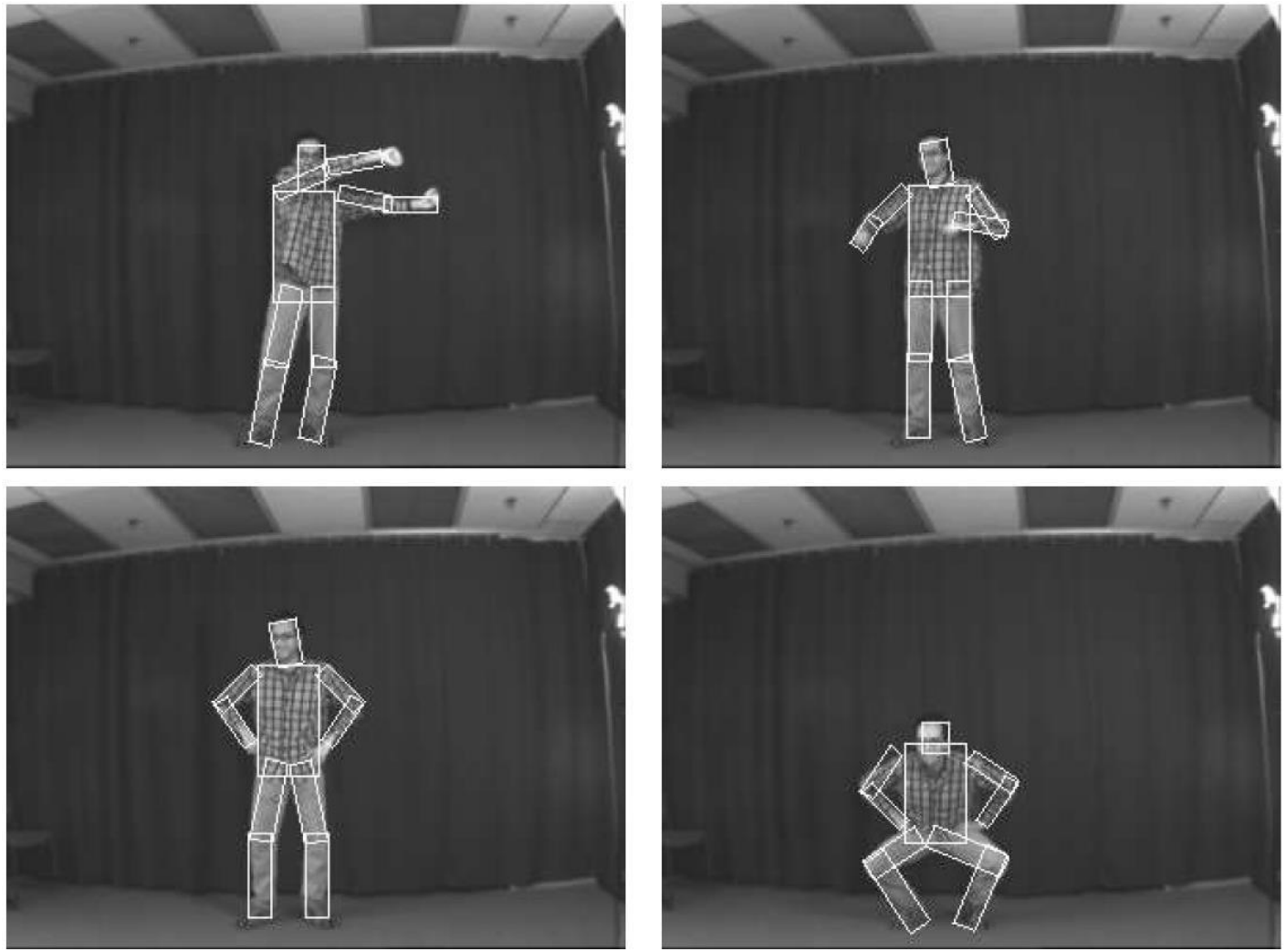
Appearance likelihood **Geometry likelihood**





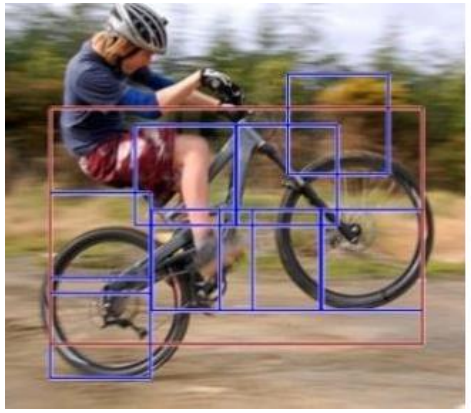
Technical evolution in generic object detection

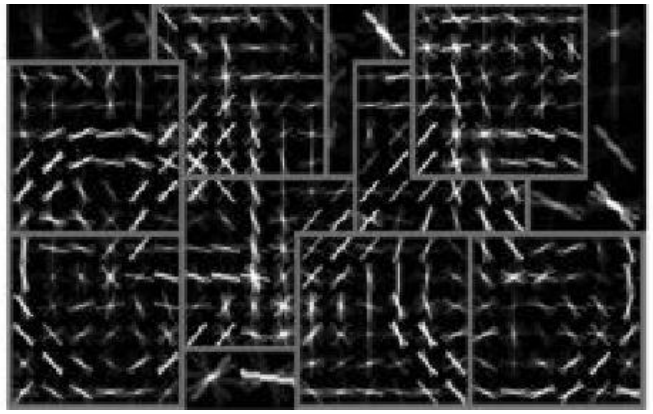
Deformable Parts Model



Technical evolution in generic object detection

Deformable Parts Model





1. Strong low-level features based on HOG
2. Efficient matching algorithms for deformable part-based models (pictorial structures)
3. Discriminative learning with latent variables (latent SVM)

Felzenszwalb et al., 2008, 2010, 2011, 2012

Technical evolution in generic object detection

Deformable Parts Model

Why did gradient-based models work?

Average gradient image

Technical evolution in generic object detection

Deformable Parts Model

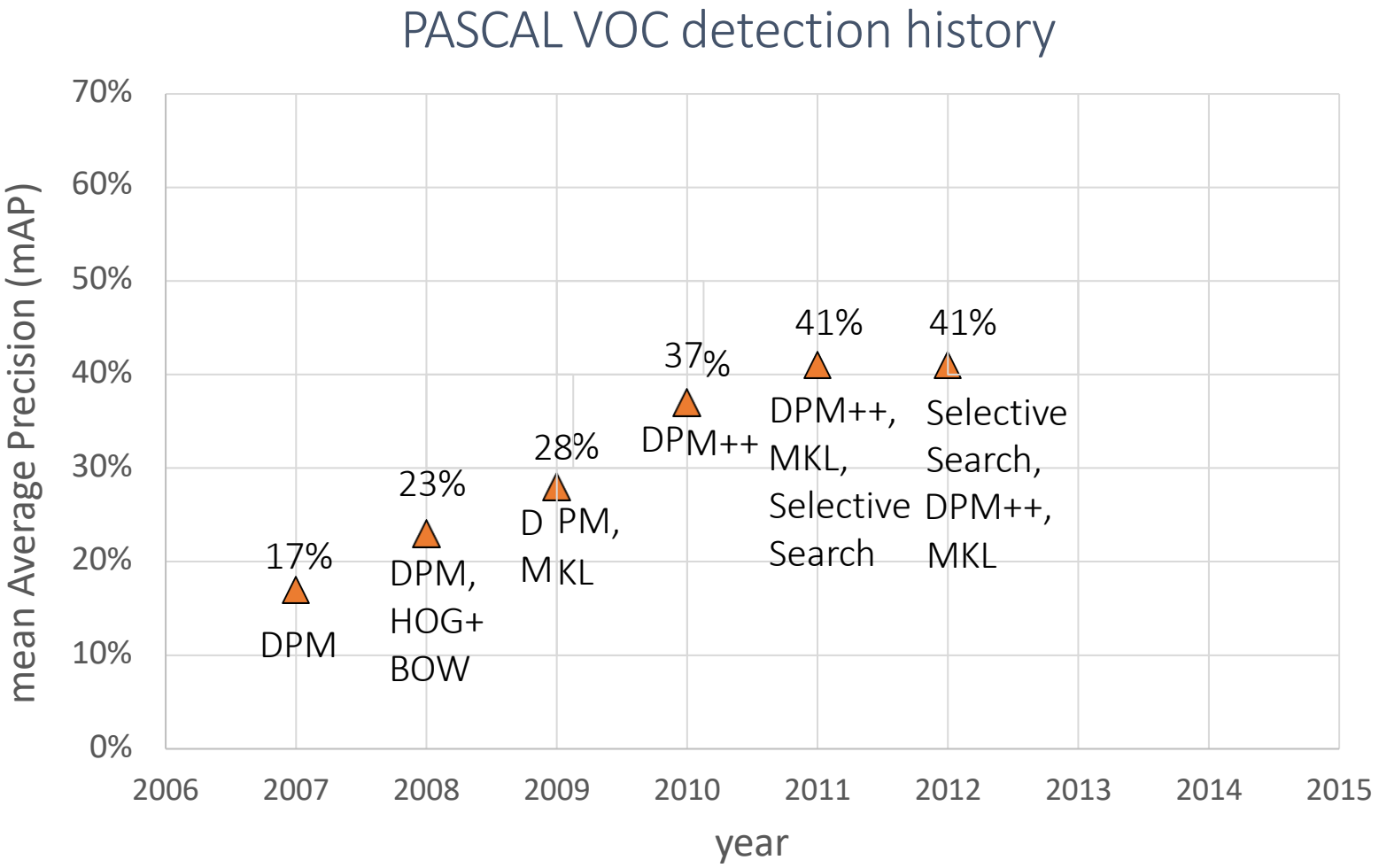
Generic categories (what is does not work as well)

Can we detect people, chairs, horses, cars, dogs, buses, bottles, sheep …?

PASCAL Visual Object Categories (VOC) dataset

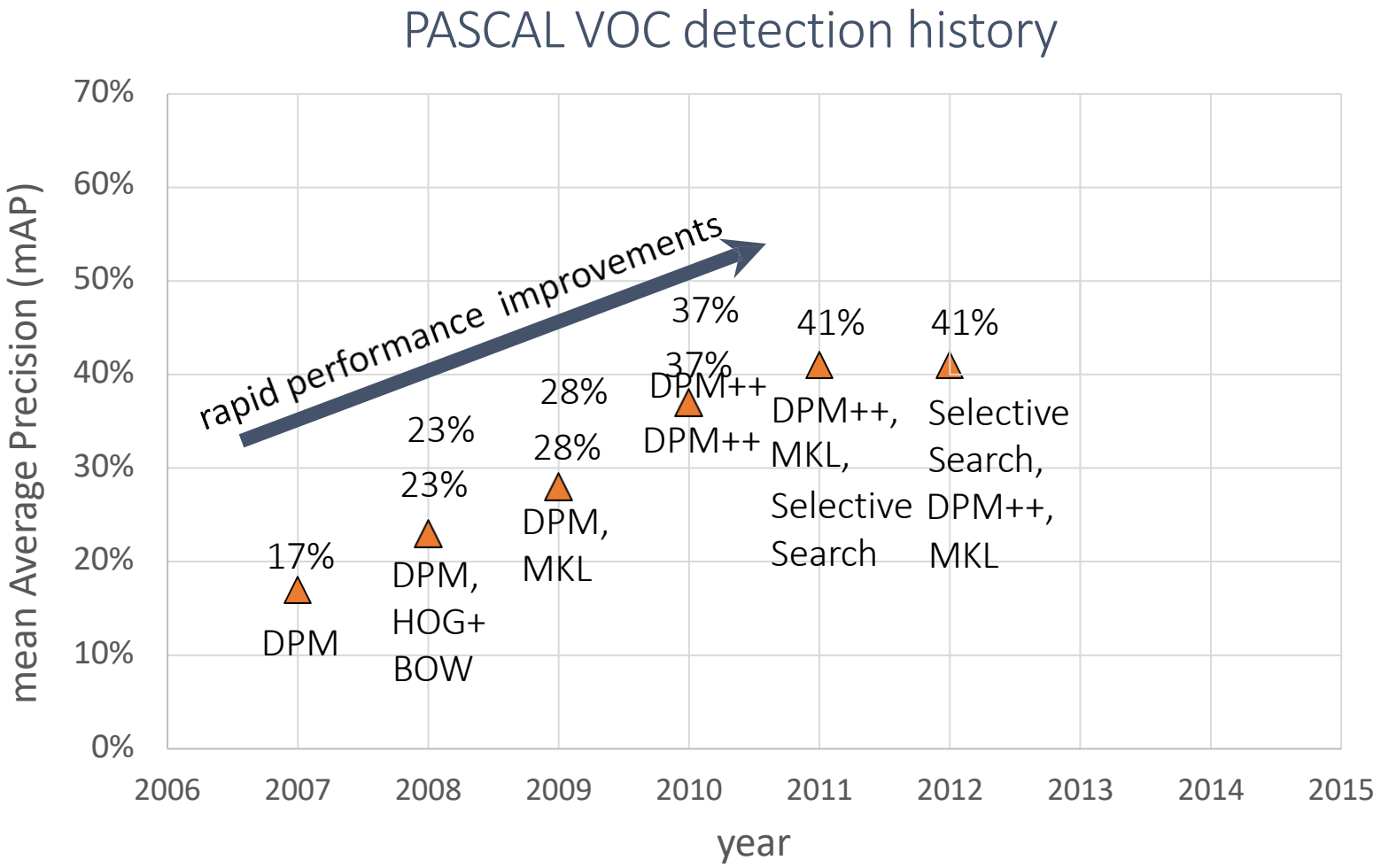
Technical evolution in generic object detection

Deformable Parts Model



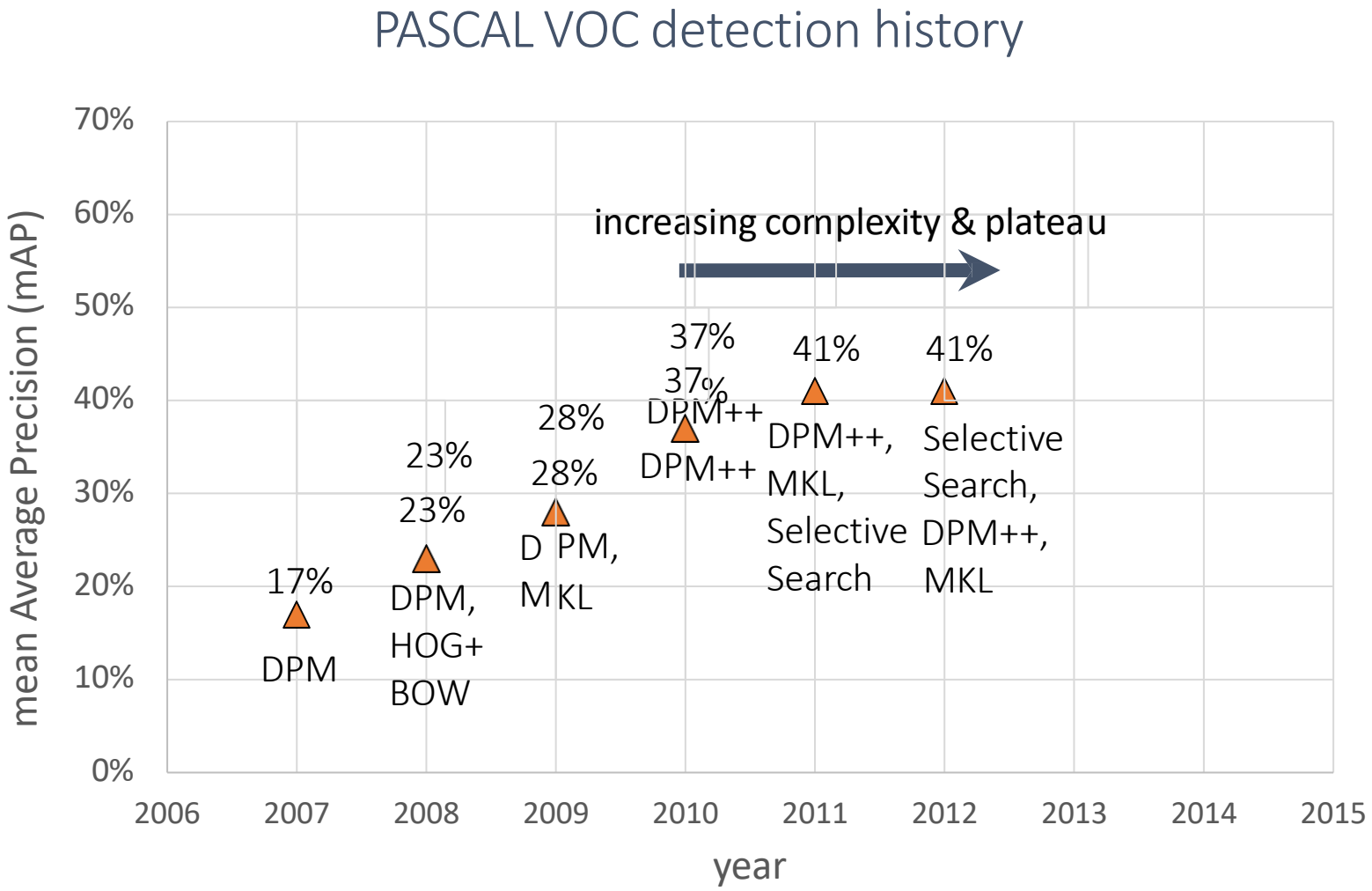
Technical evolution in generic object detection

Deformable Parts Model



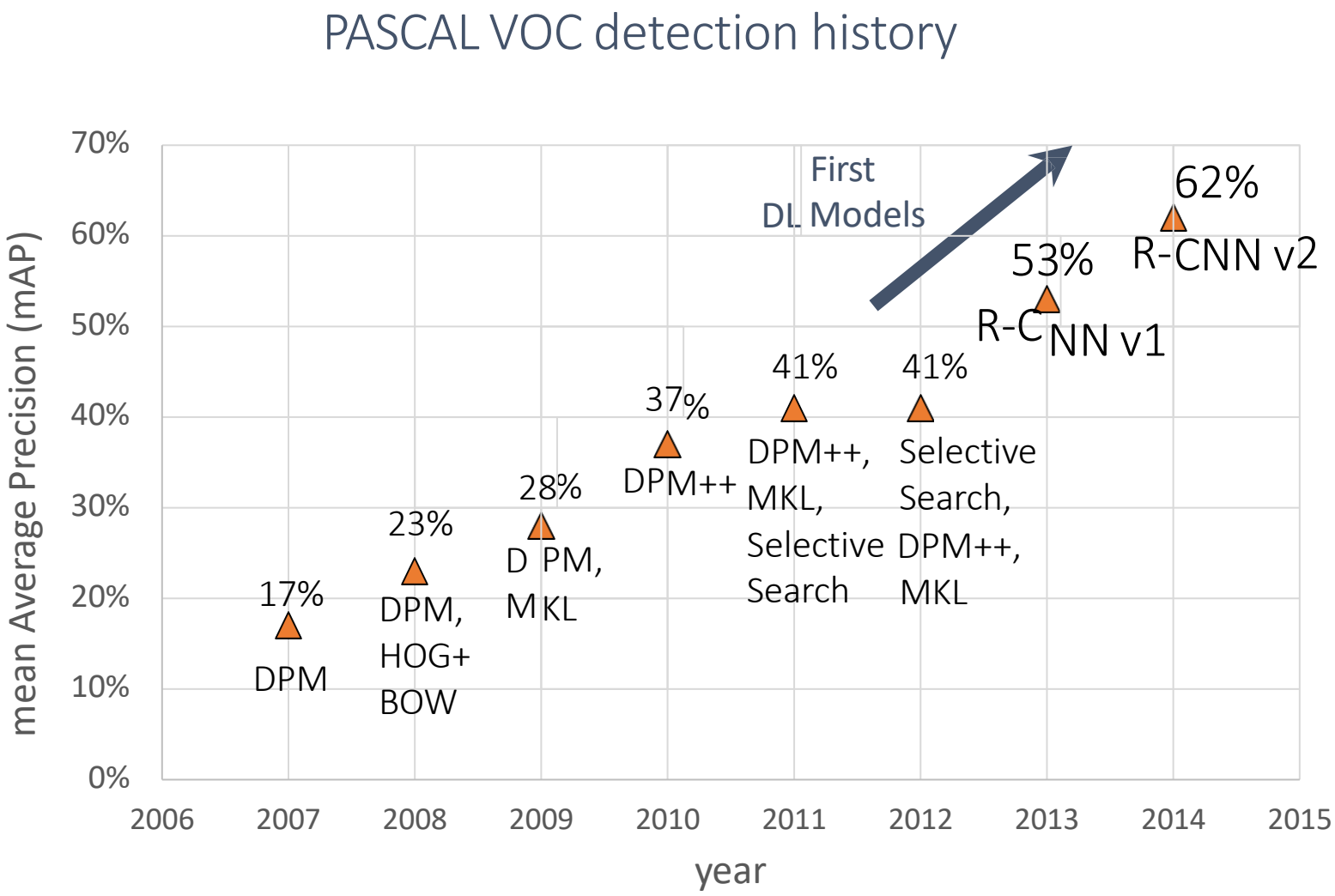
Technical evolution in generic object detection

Deformable Parts Model



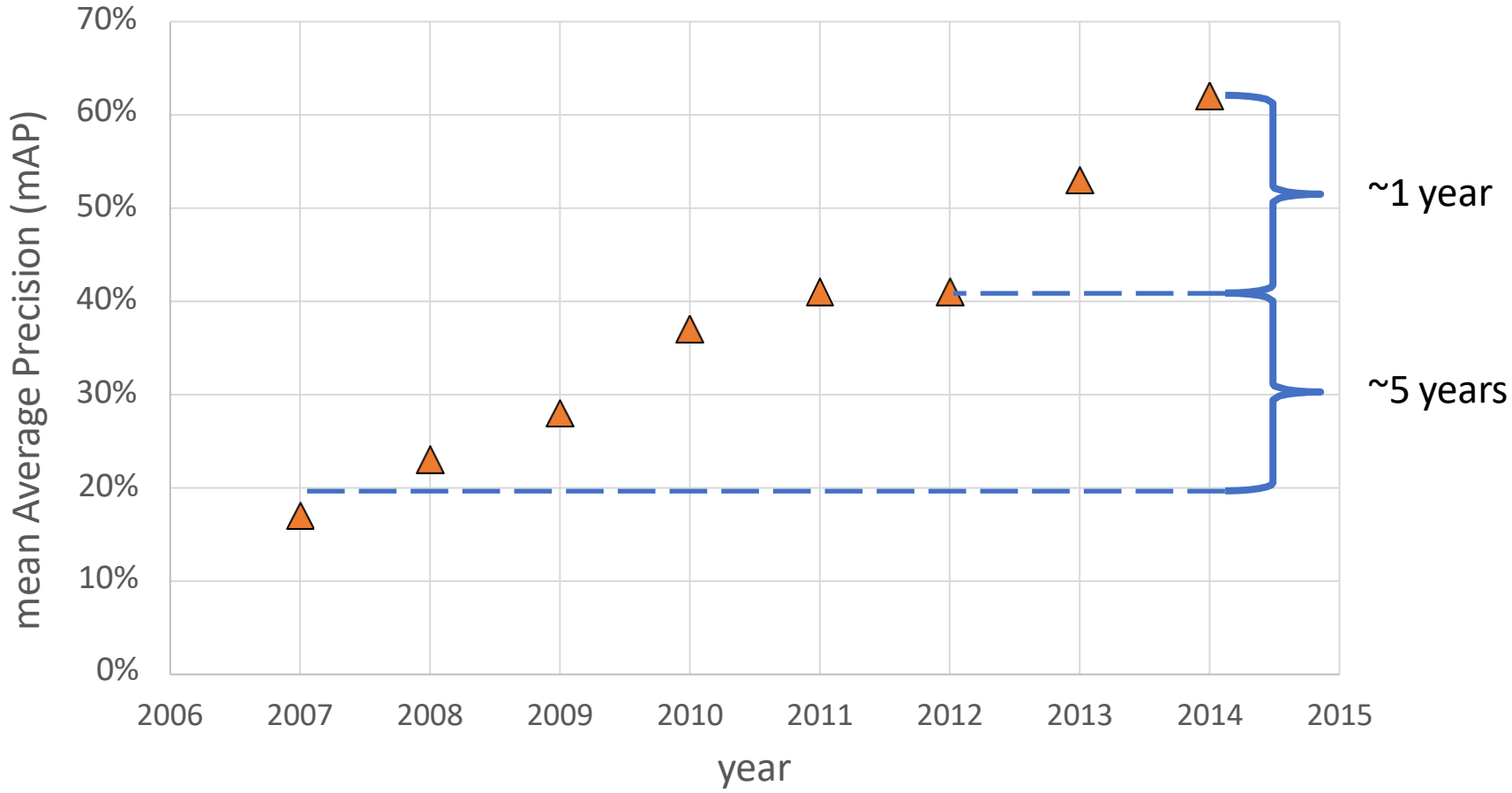
Technical evolution in generic object detection

Deformable Parts Model



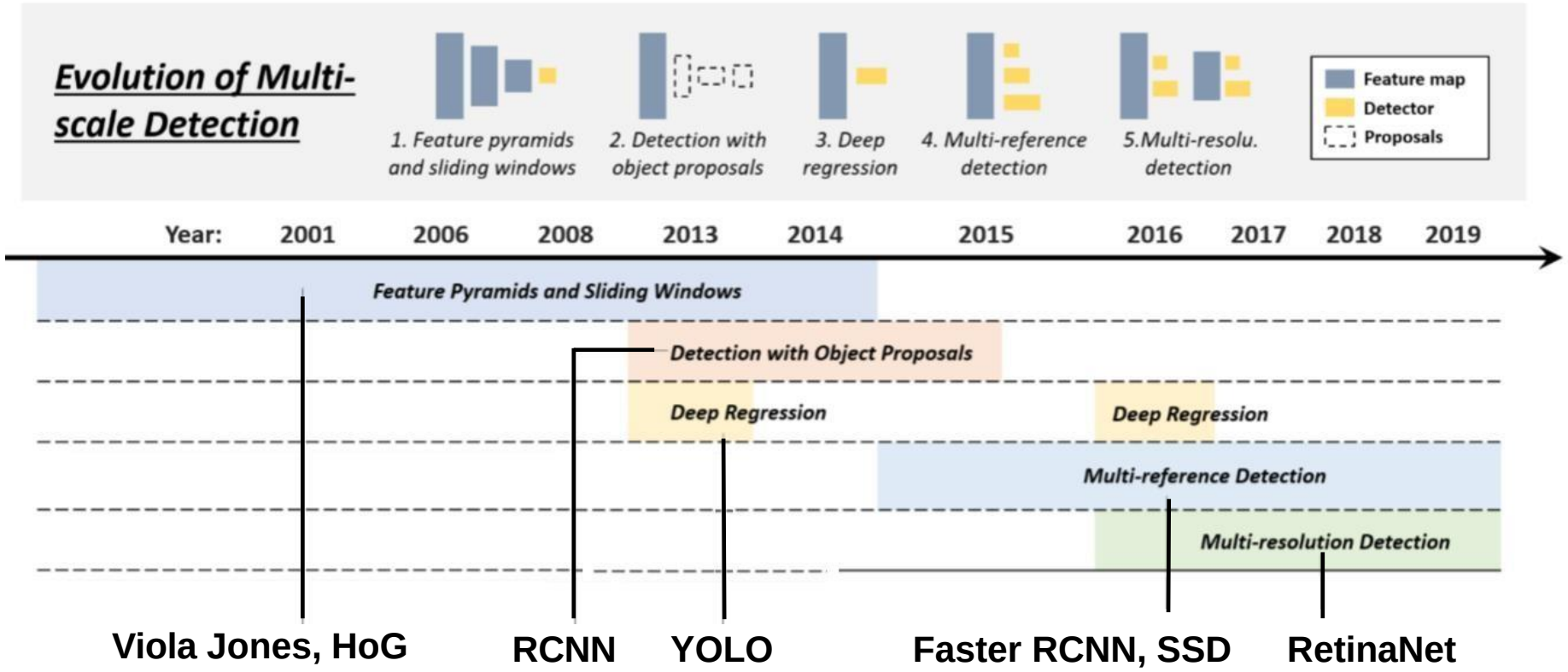
Technical evolution in generic object detection

Deformable Parts Model



Technical evolution in generic object detection

Multi-scale object detection



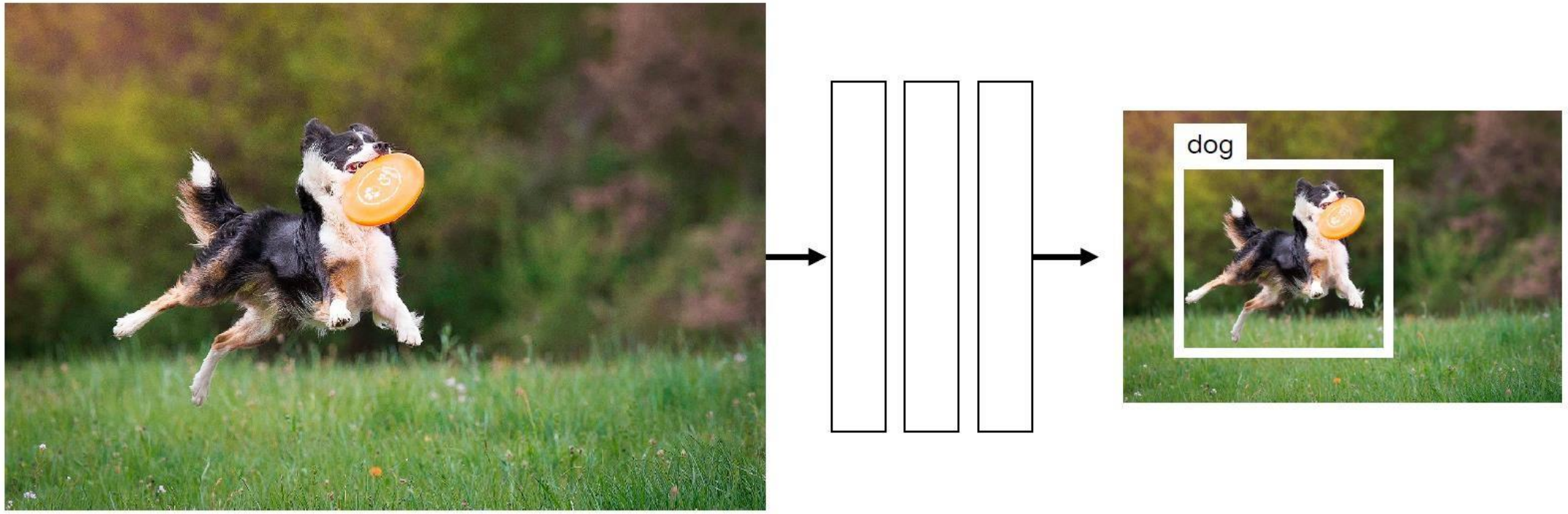
Detection of objects of different sizes and different aspect ratios

We will discuss the evolution of detectors considering this aspect

Technical evolution in generic object detection

Multi-scale object detection

Bounding Box Regression

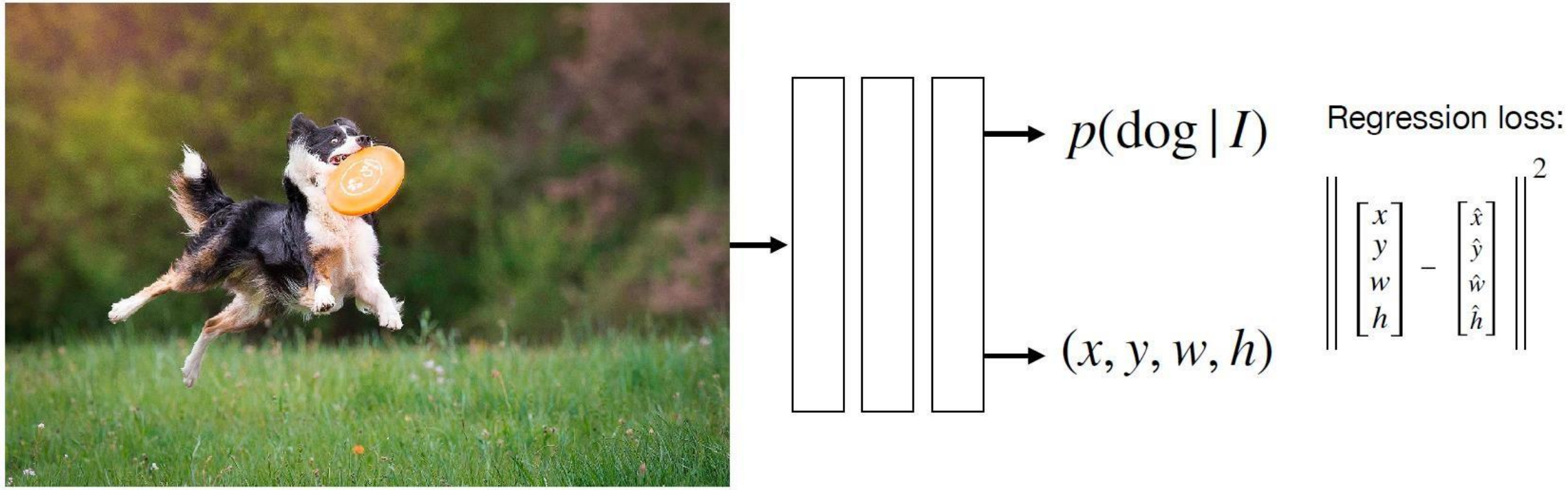


<u>Problem:</u> Doesn’t scale with multiple objects

Technical evolution in generic object detection

Multi-scale object detection

Bounding Box Regression

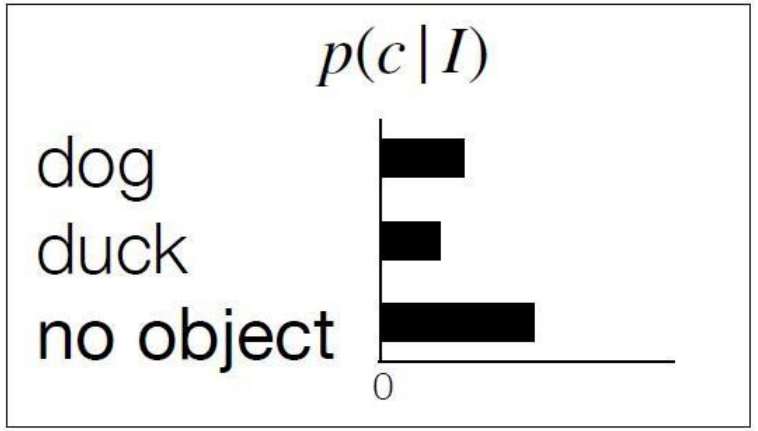


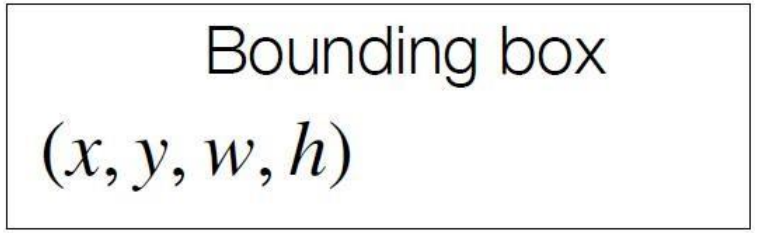
<u>Problem:</u> Doesn't scale with multiple objects

Technical evolution in generic object detection

Multi-scale object detection

Sliding window



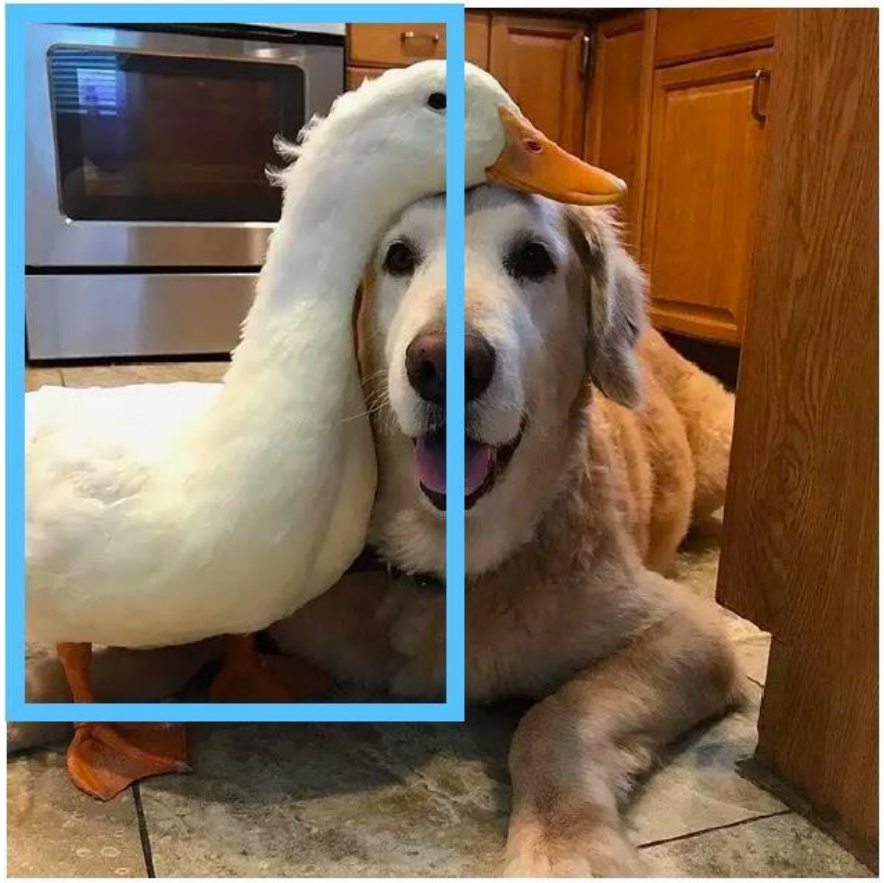


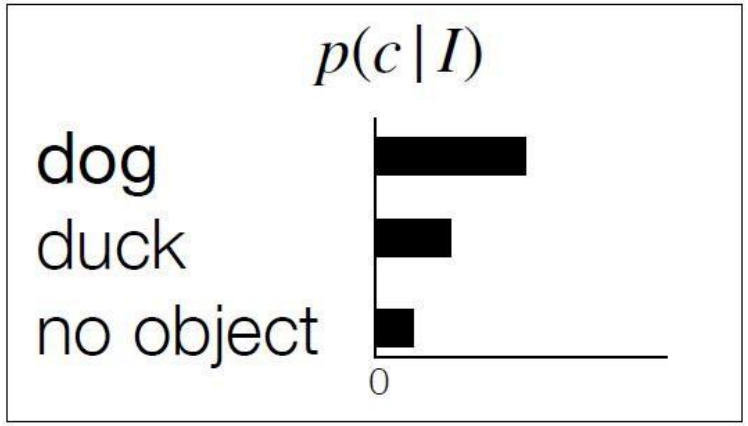
<u>Problem:</u> It requires multiple scales and ratios, very computationally expensive

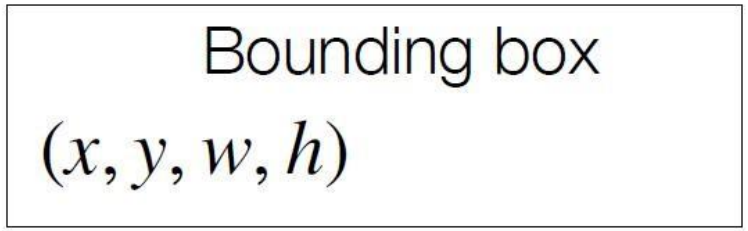
Technical evolution in generic object detection

Multi-scale object detection

Sliding window





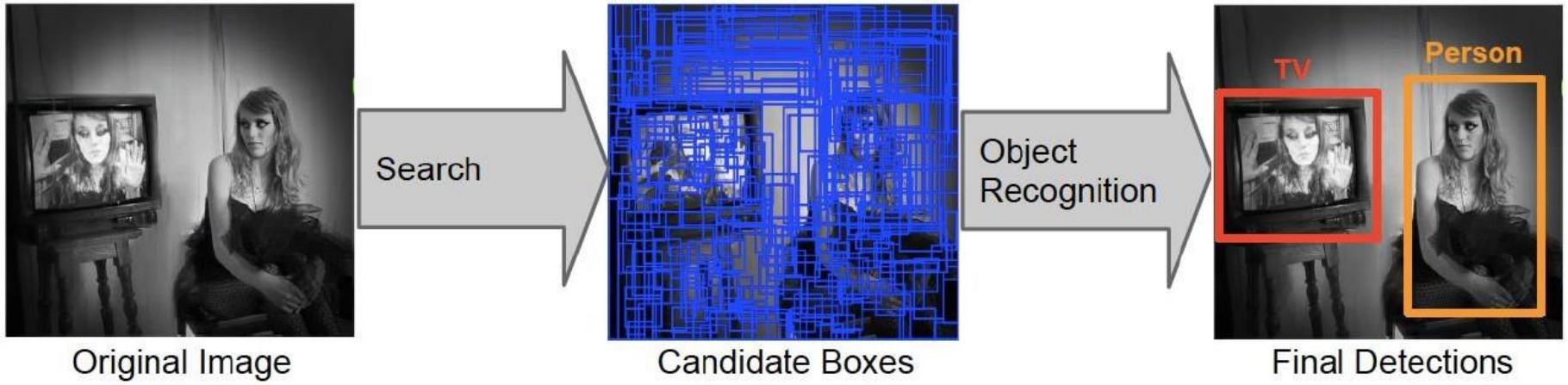


<u>Problem:</u> It requires multiple scales and ratios, very computationally expensive

Technical evolution in generic object detection

Multi-scale object detection

Object proposals



- Problem: evaluating a detector is very expensive
  - An image with n pixels has $O(n^2)$ windows
- Only generate and evaluate a few hundred **region proposals** for regions that are "likely" to be an object of interest.

Technical evolution in generic object detection

Features pyramids +sliding windows

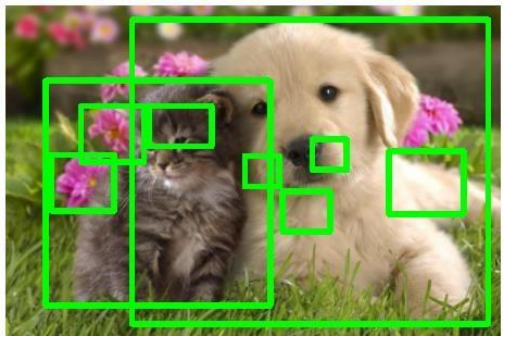
Early detection methods (Viola Jones, HoG): designed to detect objects with "fixed aspect ratios" (faces, upright pedestrians)

Girshick proposed the mixture model, using methods like DPM

The detection of various aspect ratios was not considered at the time, and many methods not worked beyond classification

As datasets become more complex, a question naturally arises:

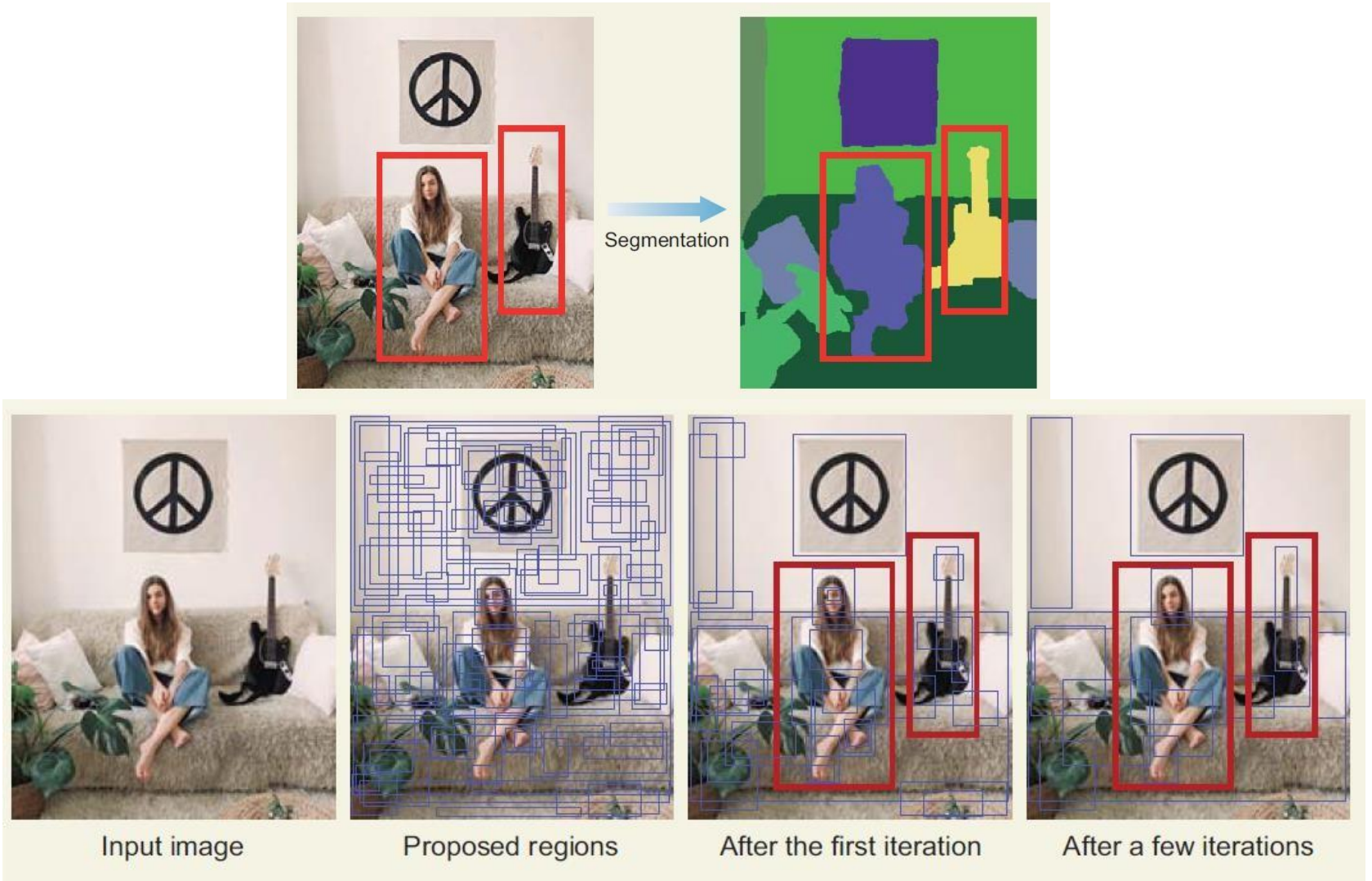
Is there a unified multi-scale approach?



Object Proposals

Technical evolution in generic object detection

Features pyramids +sliding windows



Technical evolution in generic object detection

Detection with object proposals

It refers to a group of class-agnostic candidate boxes that are likely to contain any objects, what is the main idea?

Avoiding exhaustive sliding windows search across an image…

It should meet the following criteria:

1) High Recall Rate, 2) High Loc. Accuracy, 3) Low Processing Time

There are three main approaches, but DL-based have prevailed

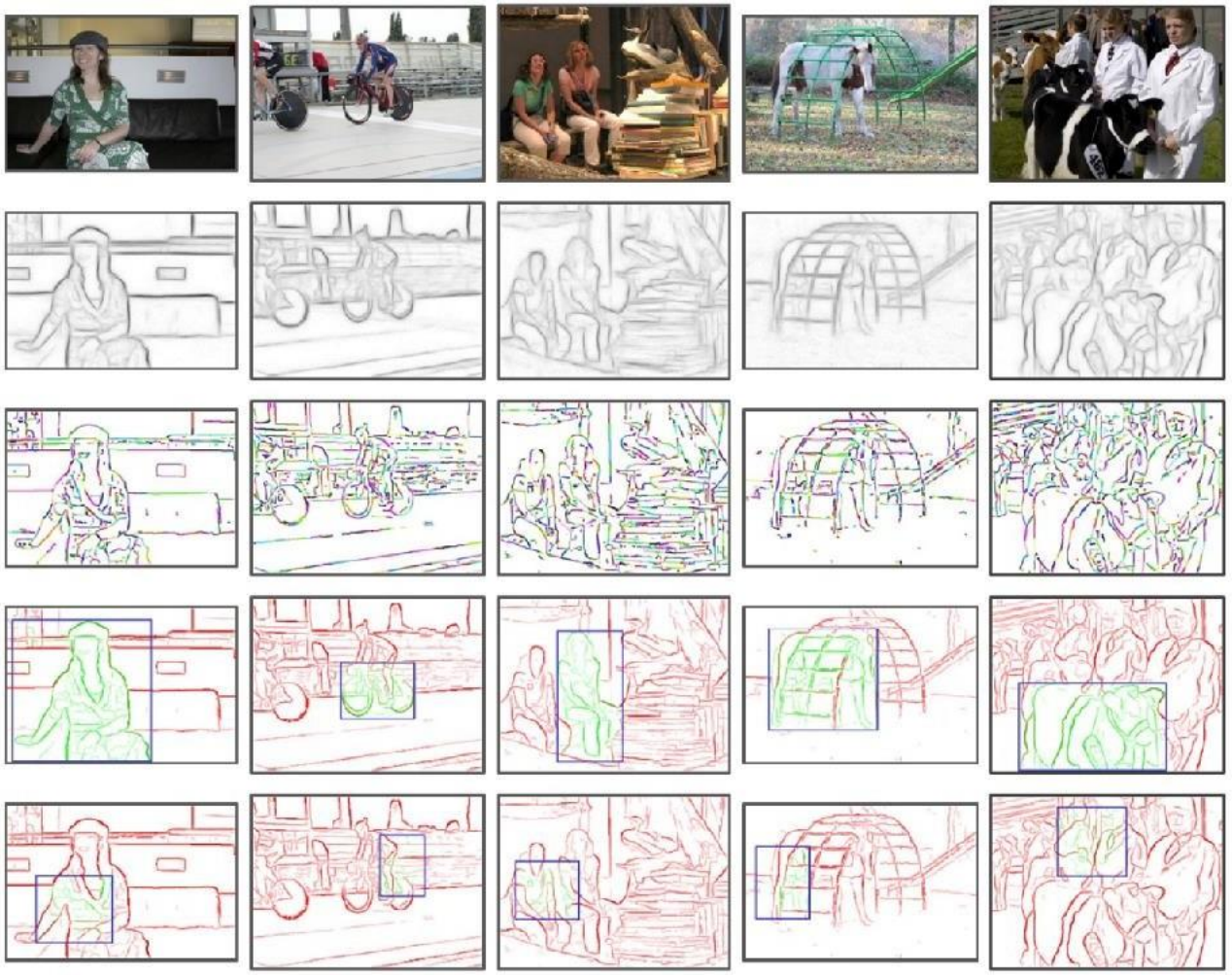
Early methods based on saliency detection: very poor localization

Deep Learning methods moved the research from bottom-up to

top-down learning-based methods for searching candidate boxes

Technical evolution in generic object detection

Detection with object proposals

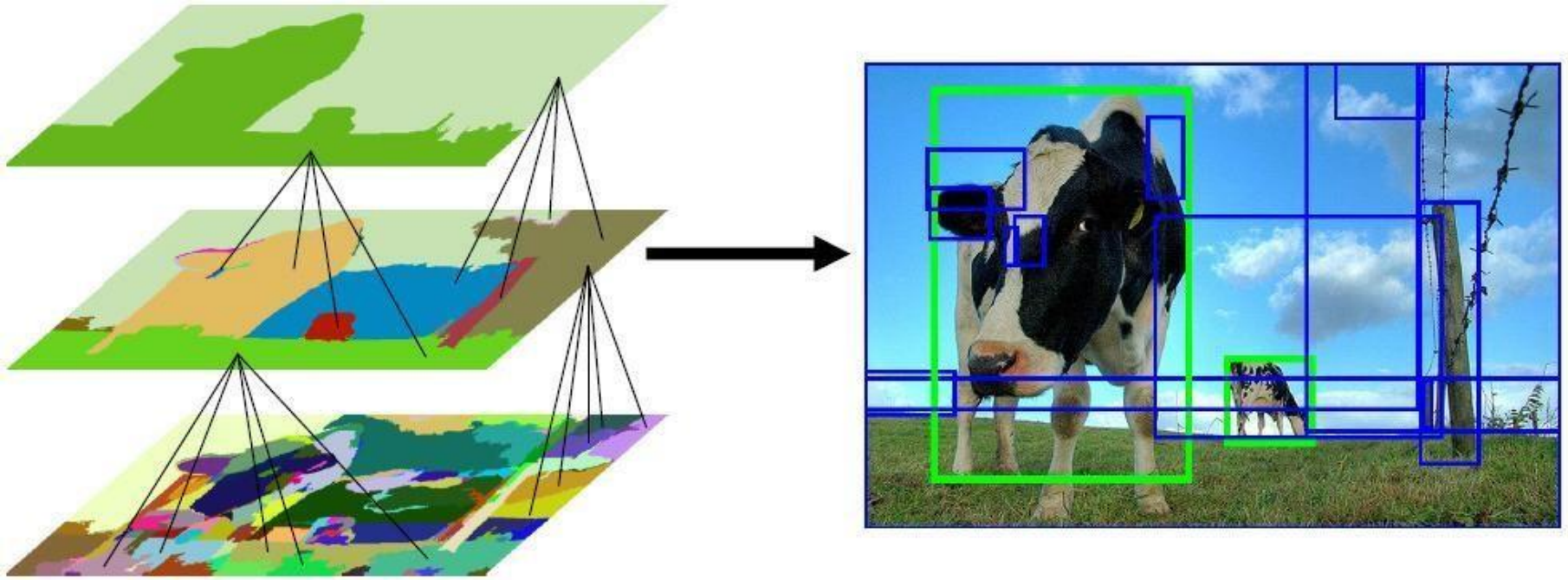


- Example: edge boxes [Zitnick & Dollar, 2014]
- Heuristic: detect edges, group them into contours
- Rank each window based on number of contours in window
- These are the only windows our detector will see

Technical evolution in generic object detection

Detection with object proposals

- Use segmentation to produce ~5K candidates

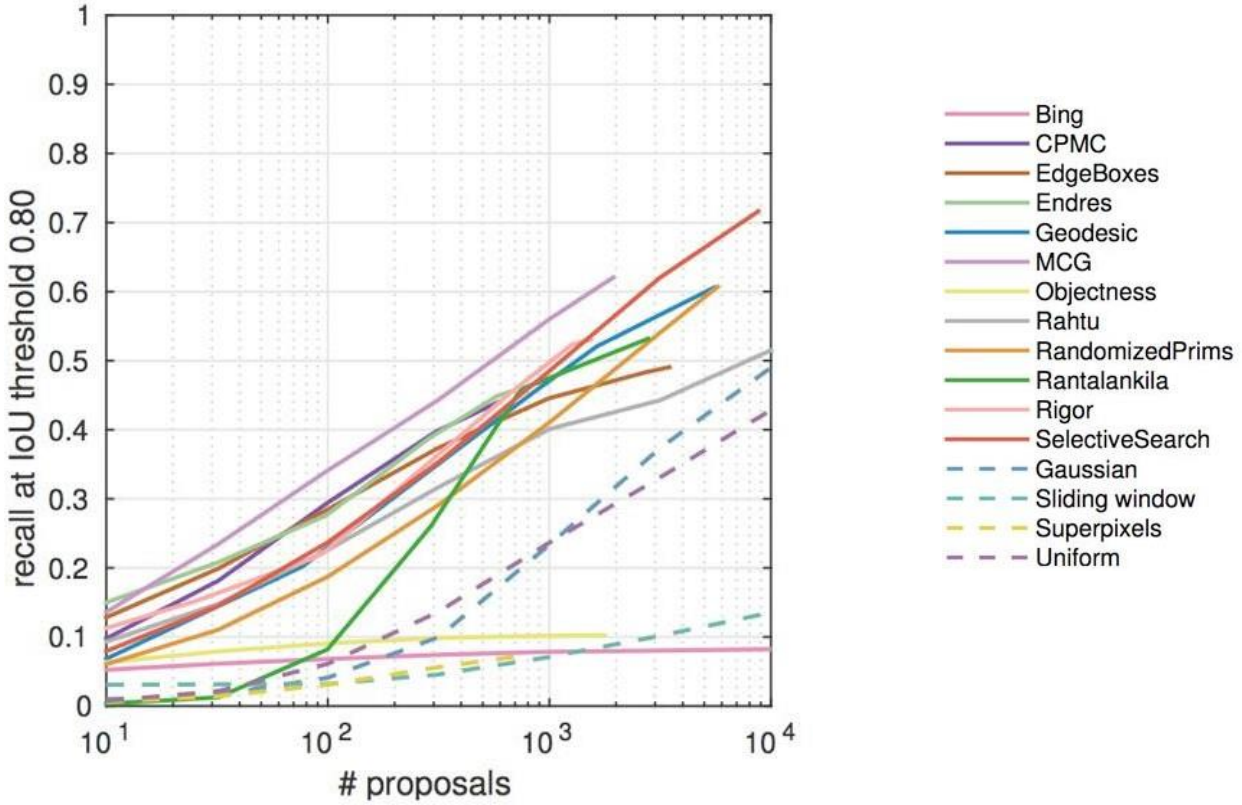


Selective Search for Object Recognition
J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, A. W. M. Smeulders
In International Journal of Computer Vision 2013.

Technical evolution in generic object detection

Detection with object proposals

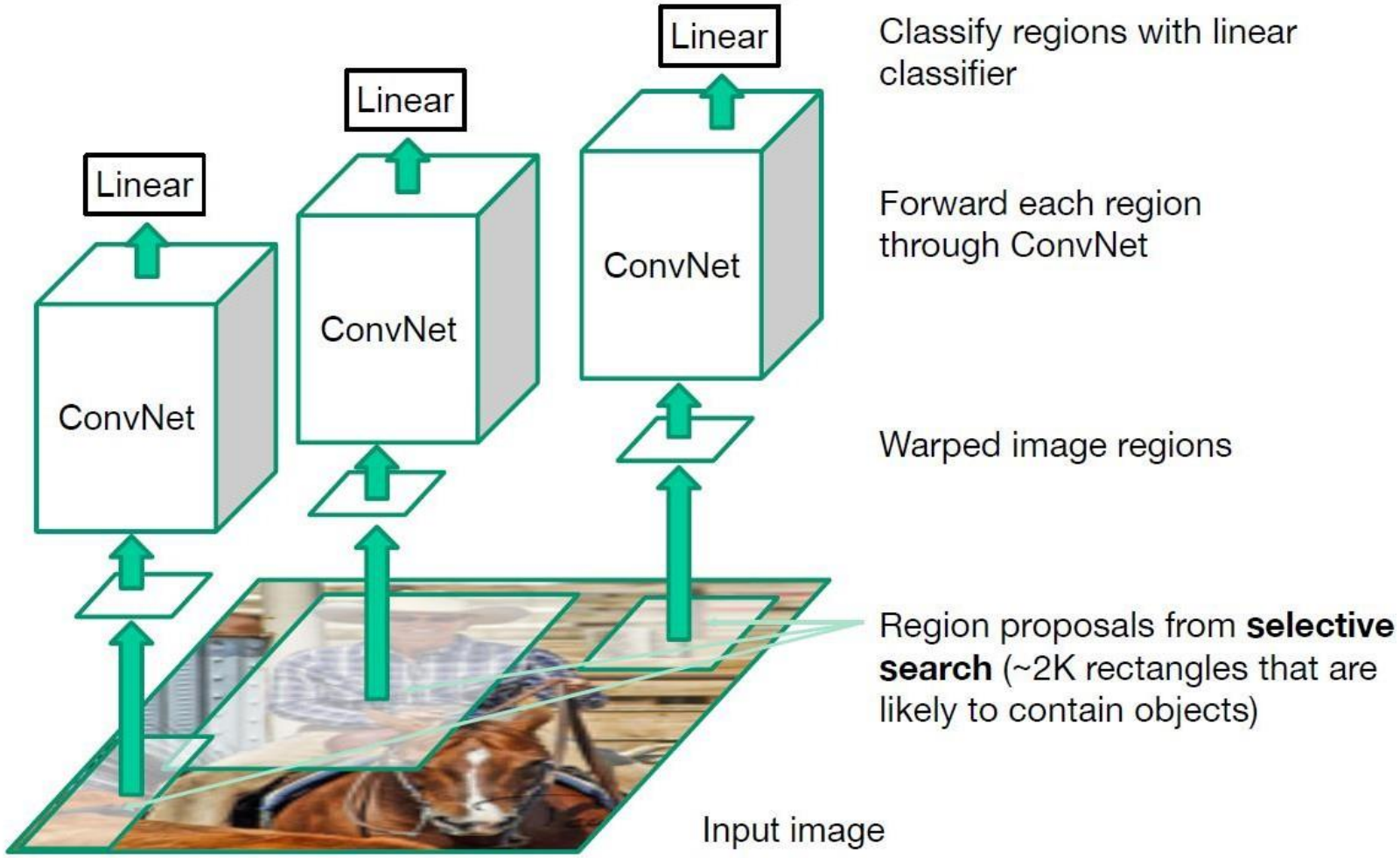
- Tens of ways of generating candidates (“proposals”)
- What fraction of ground truth objects have proposals near them?



What makes for effective detection proposals? J. Hosang, R. Benenson, P. Dollar, B. Schiele. In TPAMI

Technical evolution in generic object detection

Detection with object proposals: **RCNN**



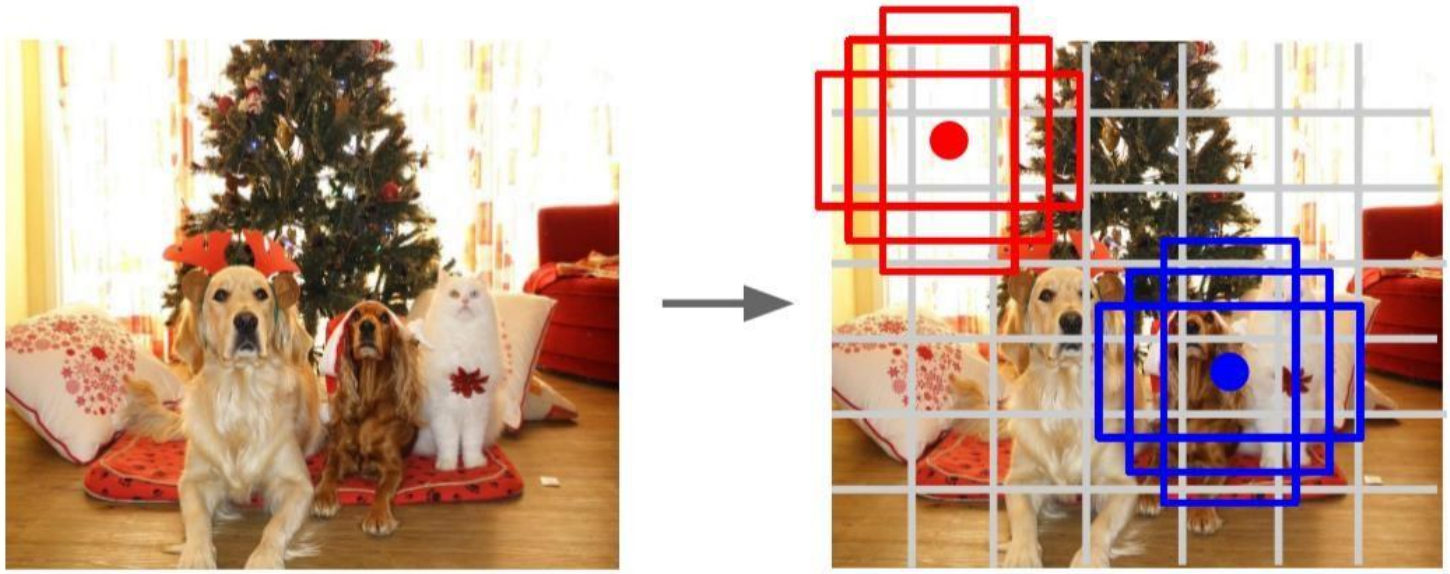
R. Girshick, J. Donahue, T. Darrell, and J. Malik, Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation, CVPR 2014.

Technical evolution in generic object detection

Detection with object proposals :Deep Regression

Increasing power of GPUs lead people to use "brute force" methods for dealing with multi-scale detection

Deep Regression's idea: to directly predict the coordinates of a BB based on deep learning features (i.e. from AlexNet)



Input image
3 x H x W

Divide image into grid
7 x 7

Technical evolution in generic object detection

Detection with object proposals: Multi-reference

Deep regression had issues with localization (i.e. for small objects)

Multi-reference: pre-define a set of reference boxes (anchors) with different sizes and aspect ratios at different locations of the image

A typical loss of each predefined anchor box consists of 2 parts:

1. Cross entropy loss for category recognition
2. L1/L2 regression loss

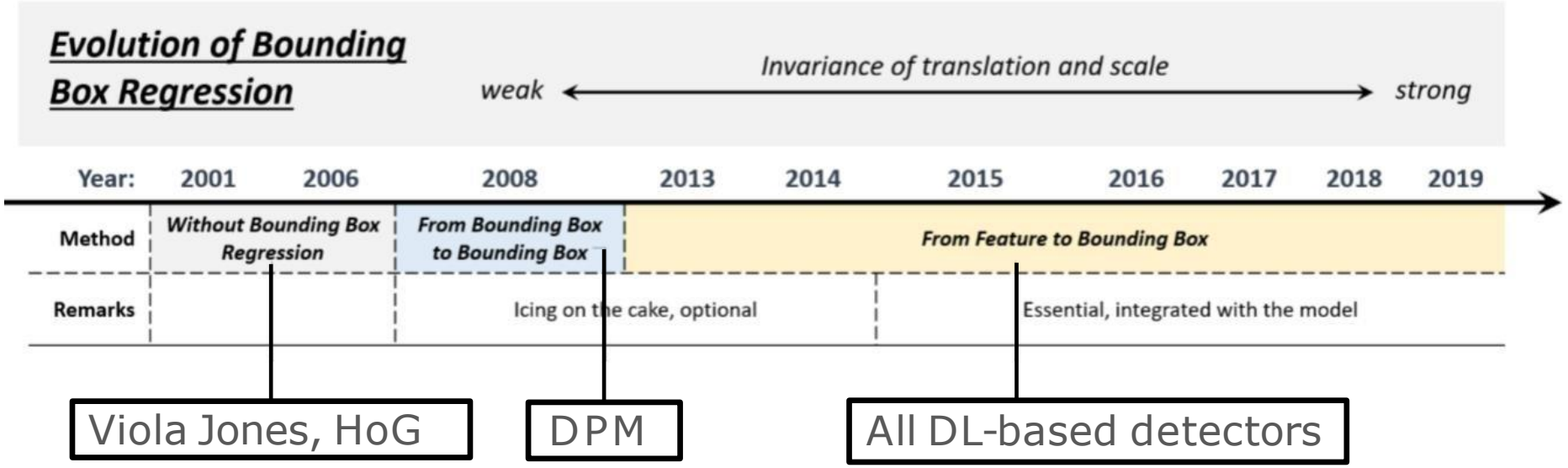
$$L(p, p^*, t, t^*) = L_{cls.}(p, p^*) + \beta I(t) L_{loc.}(t, t^*)$$

$$I(t) = \begin{cases} 1 & \text{IOU}\{a, a^*\} > \eta \\ 0 & \text{else} \end{cases}$$

Another technique is multi-resolution detection, detecting objects of different scales at different layers of the network

Technical progresses in DL-based object detection

Bounding Box Regression I



The BB regression is an important technique in object detection. It aims to refine the location on the initial anchor box.

Initial methods like VJ and HoG did not use BB regression and considered the sliding window as the detection result need for

Very dense pyramid and slide the detector densely in each location

Technical progresses in DL-based object detection

Bounding Box Regression I

Bounding Box Regression: from BB to BB

BB regression was first used in DPM, but it acted more as a post-processing block, mostly optional

Later, Girshick formulated the solution as a LS regression problem based on the complete configuration of an object hypothesis

This method yielded noticeable improvements on PASCAL criteria

Bounding Box Regression: from features to BB

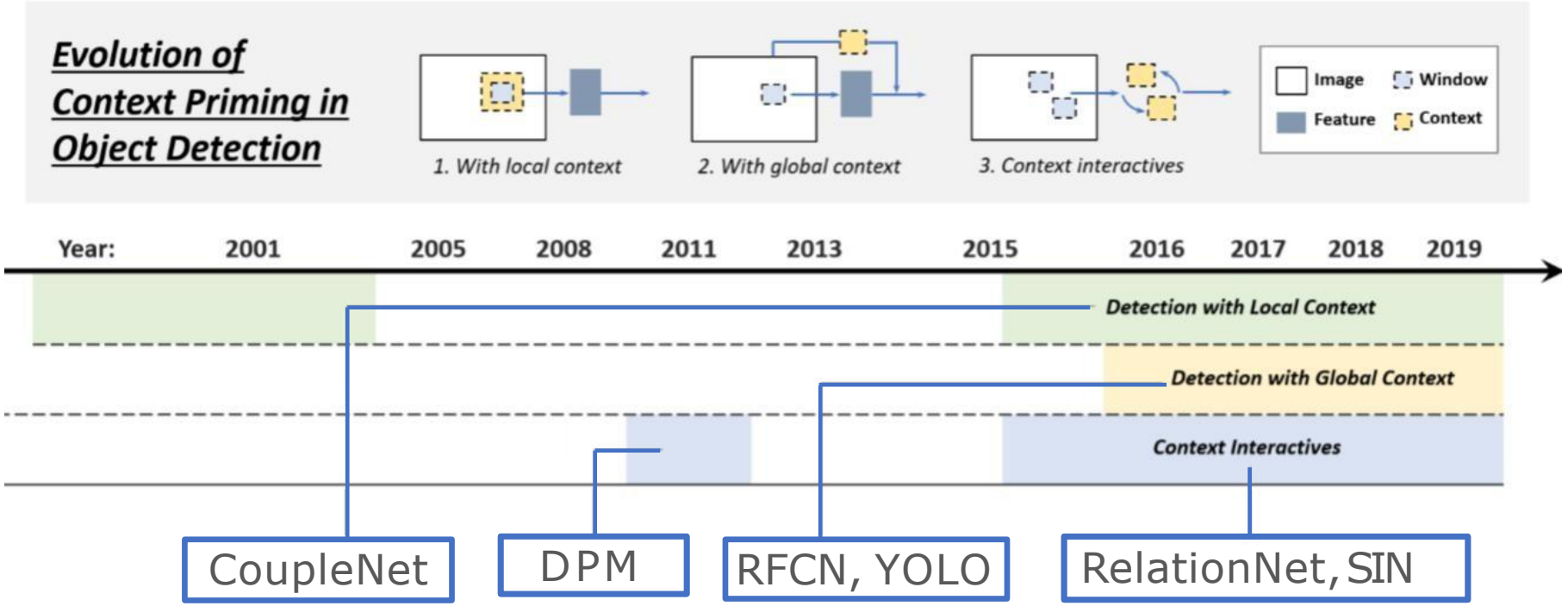
With Faster RCNN is fully integrated into the pipeline and trained in an end-to-end fashion

Also, BB regression evolved to predicting the BB directly based on features obtained from a CNN

Other regression loses as smooth-L1 were introduced

Technical progresses in DL-based object detection

Context priming for improved OD I



Visual detectors are usually embedded in a typical context.

Taking advantage of the associations of objects and its context has bee used to improved detection: context priming

Technical progresses in DL-based object detection

Context priming for improved OD II

Local context

Visual Information in the area surrounding the object to detect

It substantially improves the detection performance

Global context

Exploits scene configuration as additional source of information

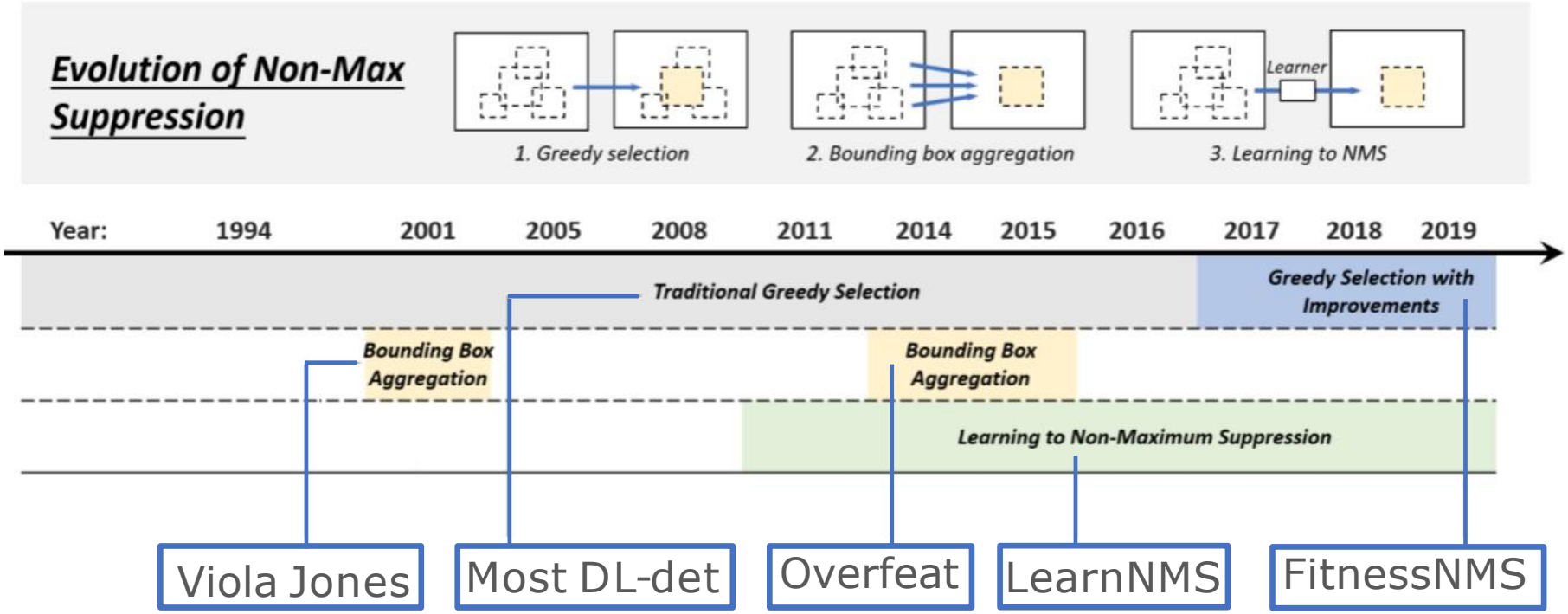
DL-based: larger receptive fields or through recursive CNNs

Context interactives

Information conveyed by the interaction of visual elements on the scene, exploring modelling dependencies among objects

Technical progresses in DL-based object detection

Evolution of non-maximum suppression I



Post-processing step to remove detection ambiguities

As neighboring windows usually have similar detection scores

Technical progresses in DL-based object detection

Evolution of non-maximum suppression Ⅱ

Greedy Selection

BB with the maximum score is selected following a threshold

Perform in a greedily manner, still one of the strongest methods

BB Aggregation

Combining or clustering multiple overlapped regions

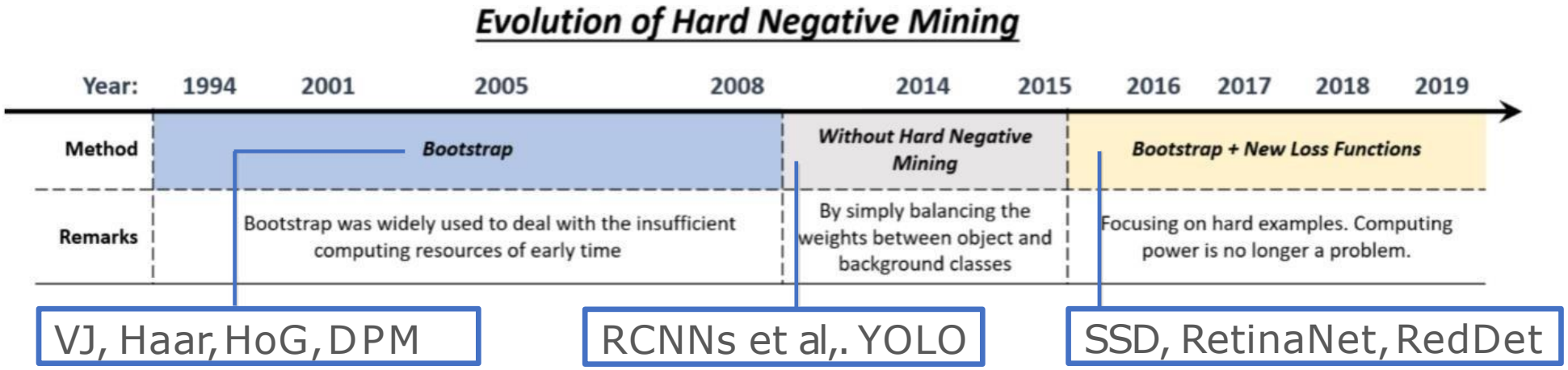
Takes into account object relationships and their spatial layout

Learning to NMS

They think NMS as a filter to rescore all raw detections and to train NMS as part of the CNN in an end-to-end fashion

Technical progresses in DL-based object detection

Evolution of hard negative mining I



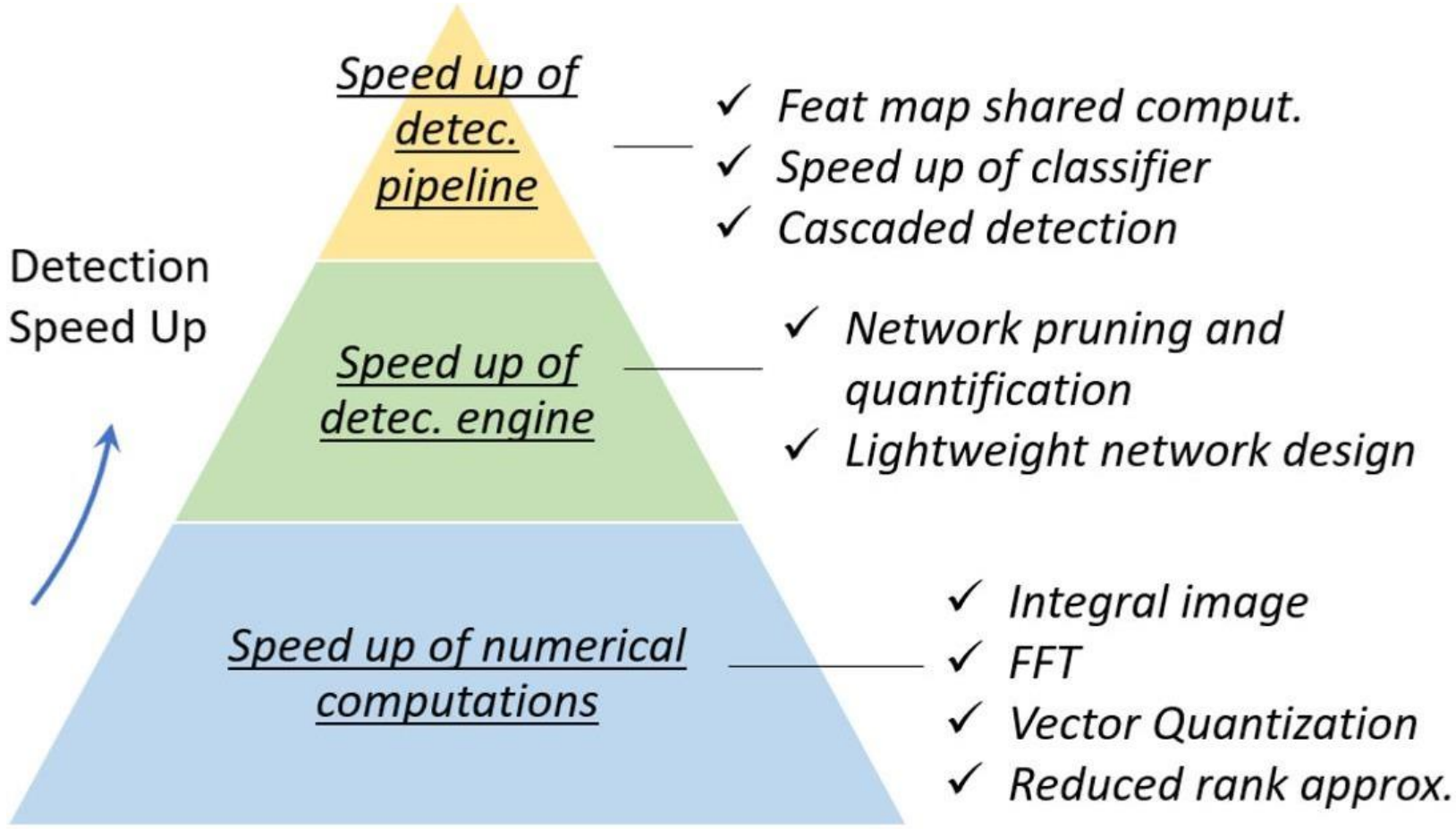
The training of an object detector is essentially an imbalanced data learning problem: 104~105 background windows per object

Hard negative mining aims to deal with the problem of imbalanced data during training.

DL-based methods simply balance the weights between the positive and negative windows +bootstrapping approaches

Technical progresses in DL-based object detection

Developments for speeding up object detection

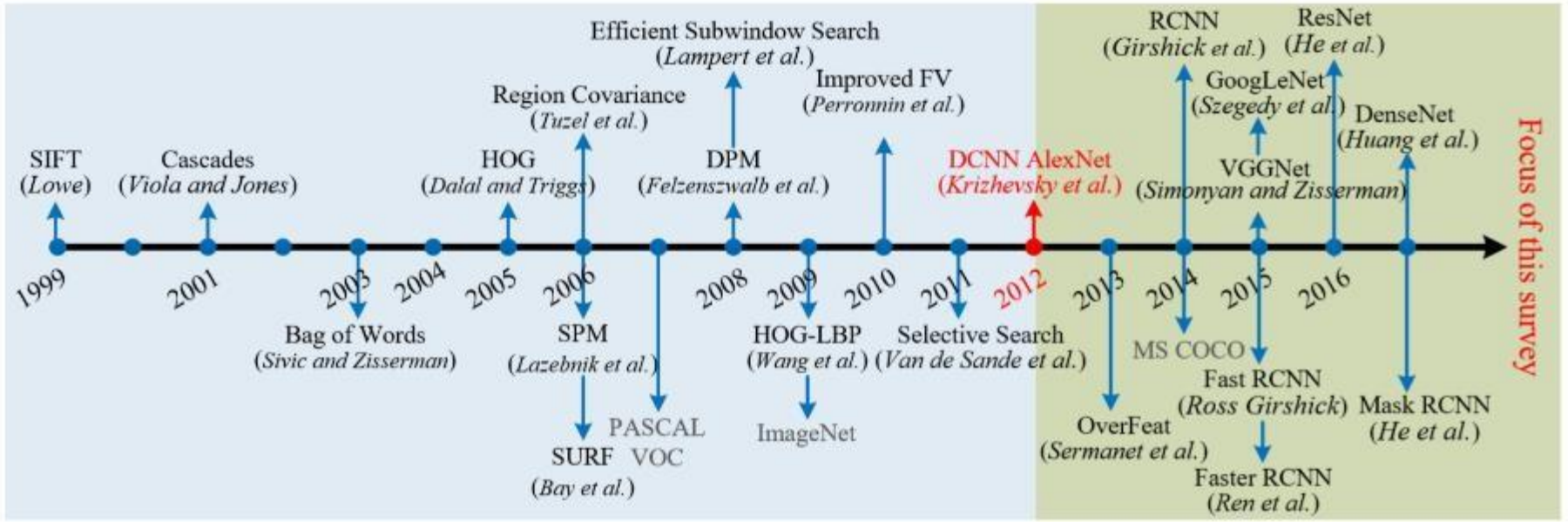


Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone Detectors**

Early research: template matching and part-based models
Specific categories with rigid spatial layouts (faces, pedestrians)
Hand-crafted features +statistical classifiers (HoG +SVM)
Useful in many applications, but couldn't cope with generic O.D.

AlexNet was a turning point in computer vision

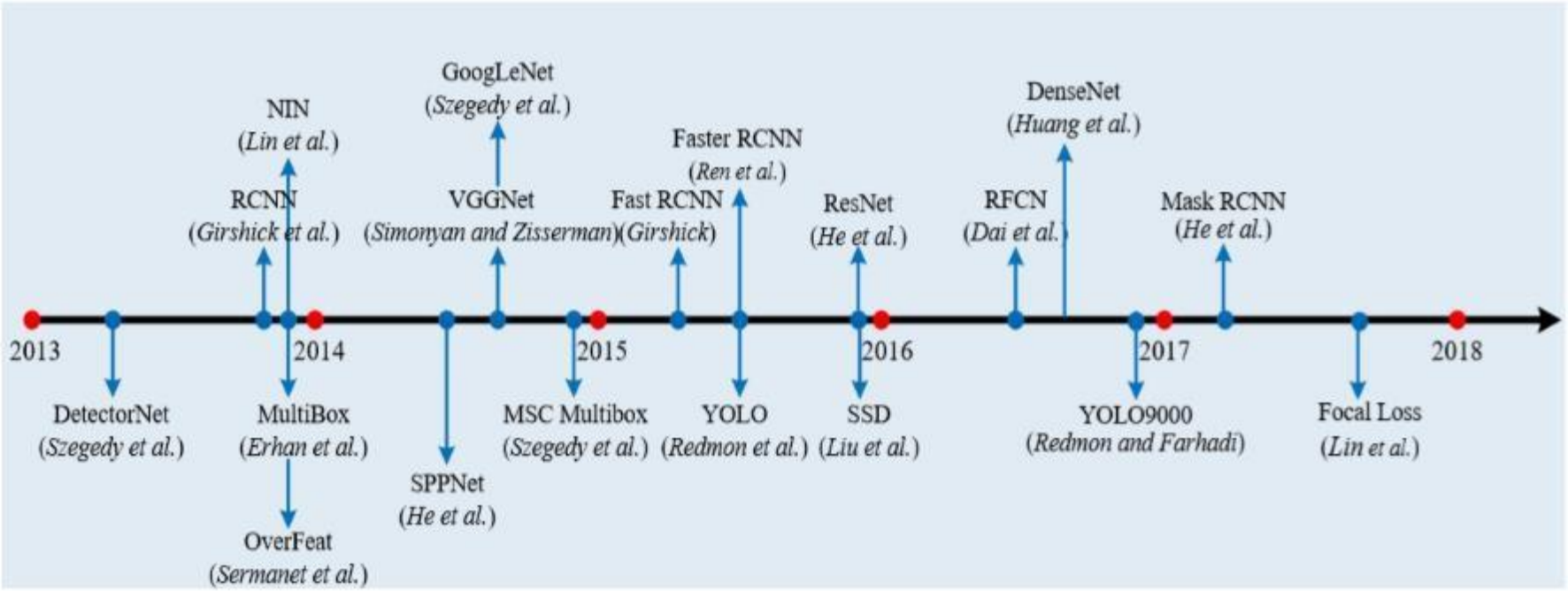


Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone CNN Detectors**

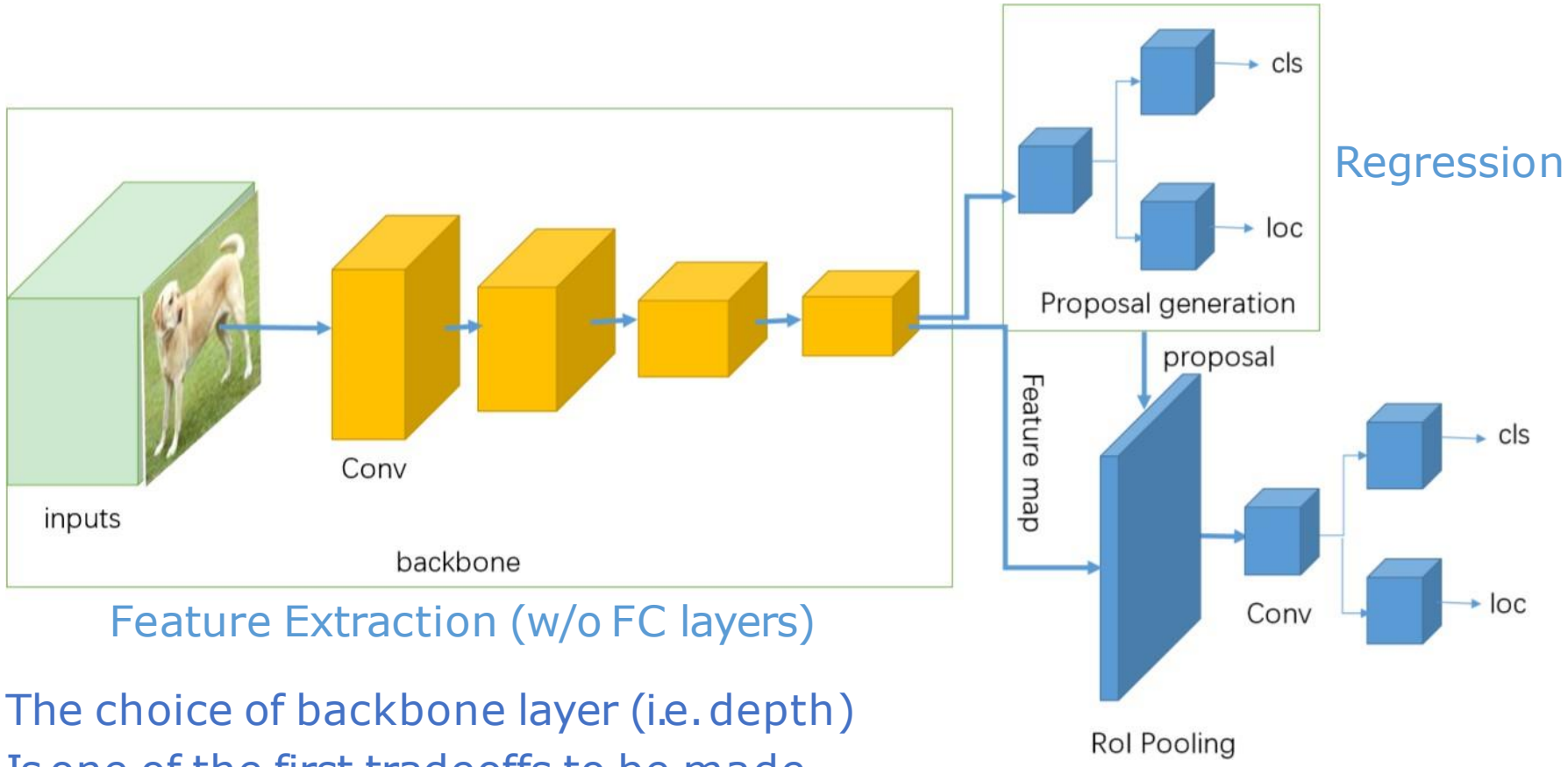
Girschick to the lead to break the deadlocks in 2014 by proposing the Regions with CNNs features (RCNN) for object detection

Since, object detection started to evolve at unprecedented speed!!!



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**



Feature Extraction (w/o FC layers)

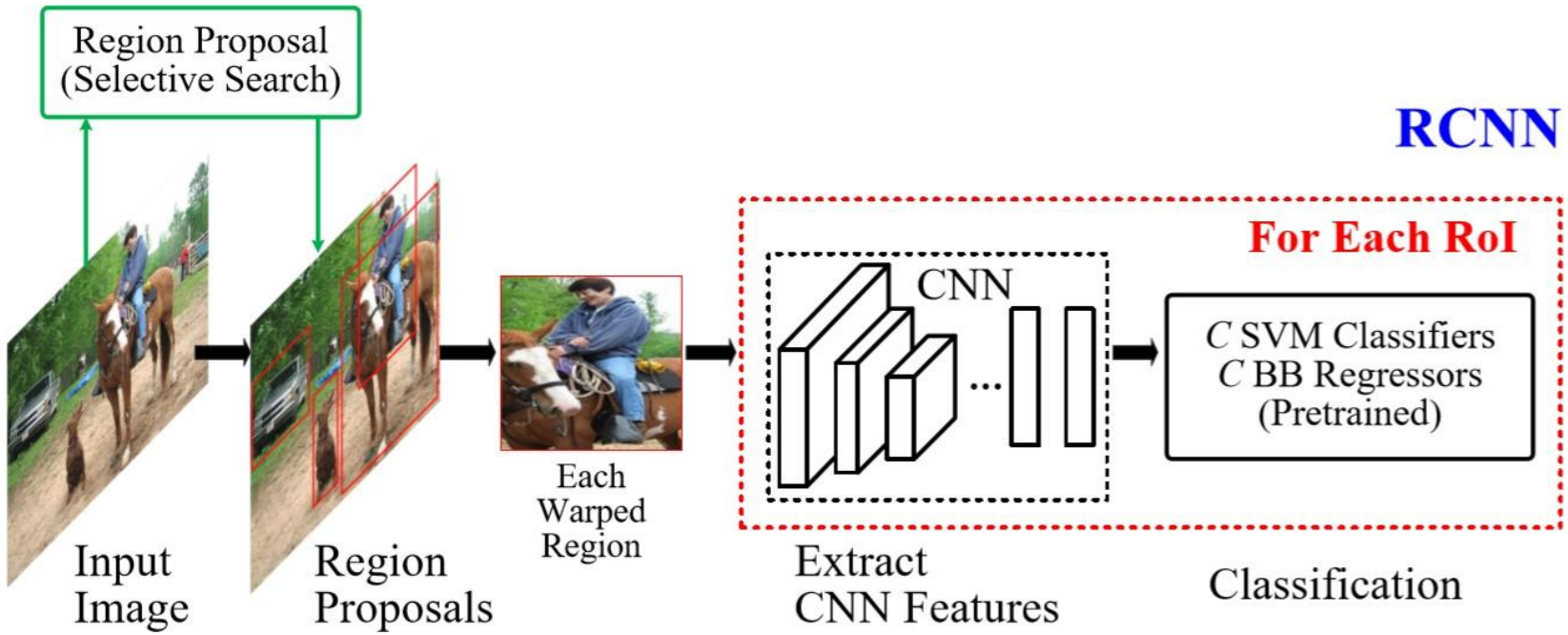
The choice of backbone layer (i.e. depth)
Is one of the first tradeoffs to be made
when designing a detector → Accuracy vs Performance

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**

R-CNN

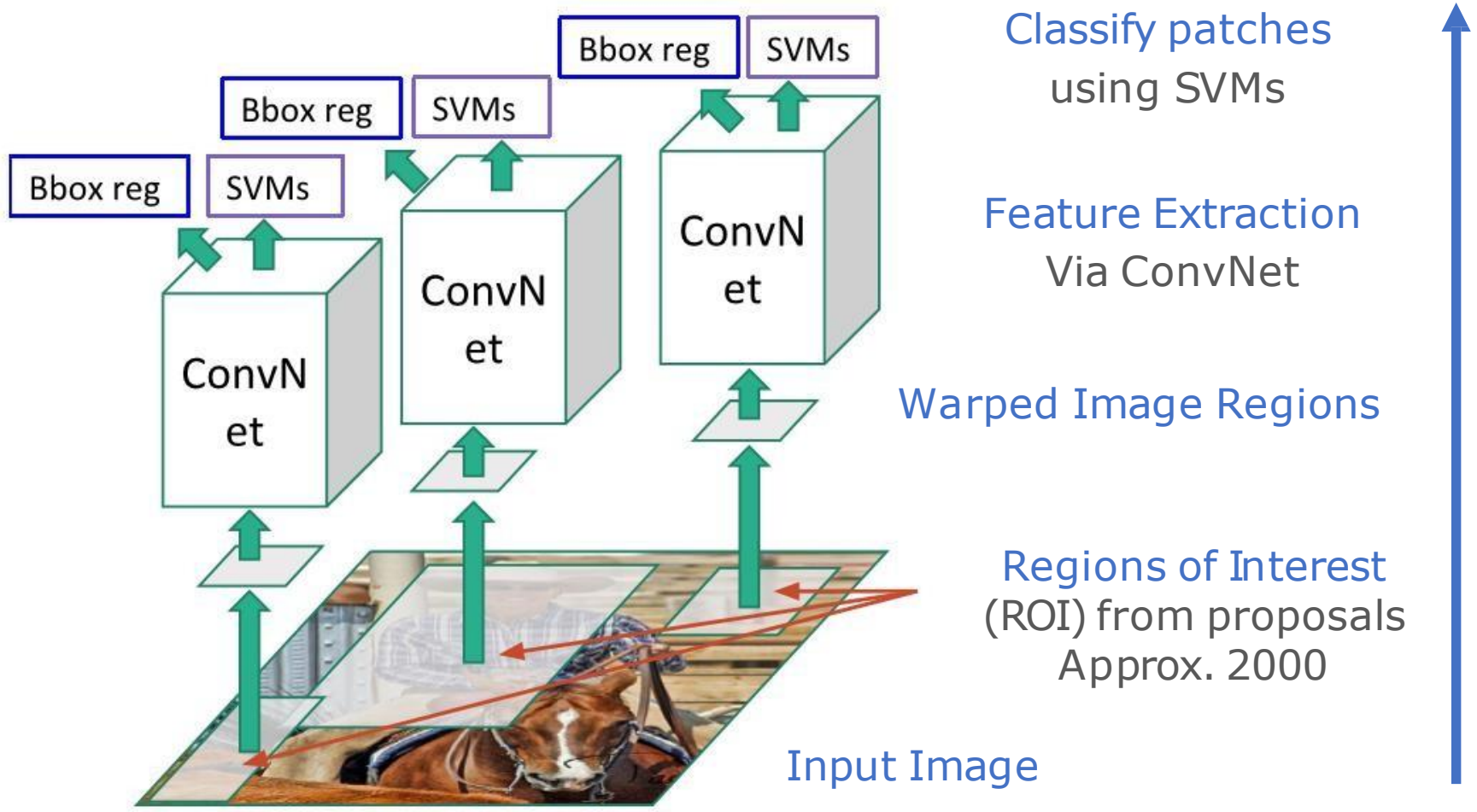
Extraction of object proposal for feature extraction +SVM
Boost in performance (mAP) from 33.7% (DPM) to 58.5%
But: slow detection speed (redundant feature comp. on R.Ps)



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**

R-CNN



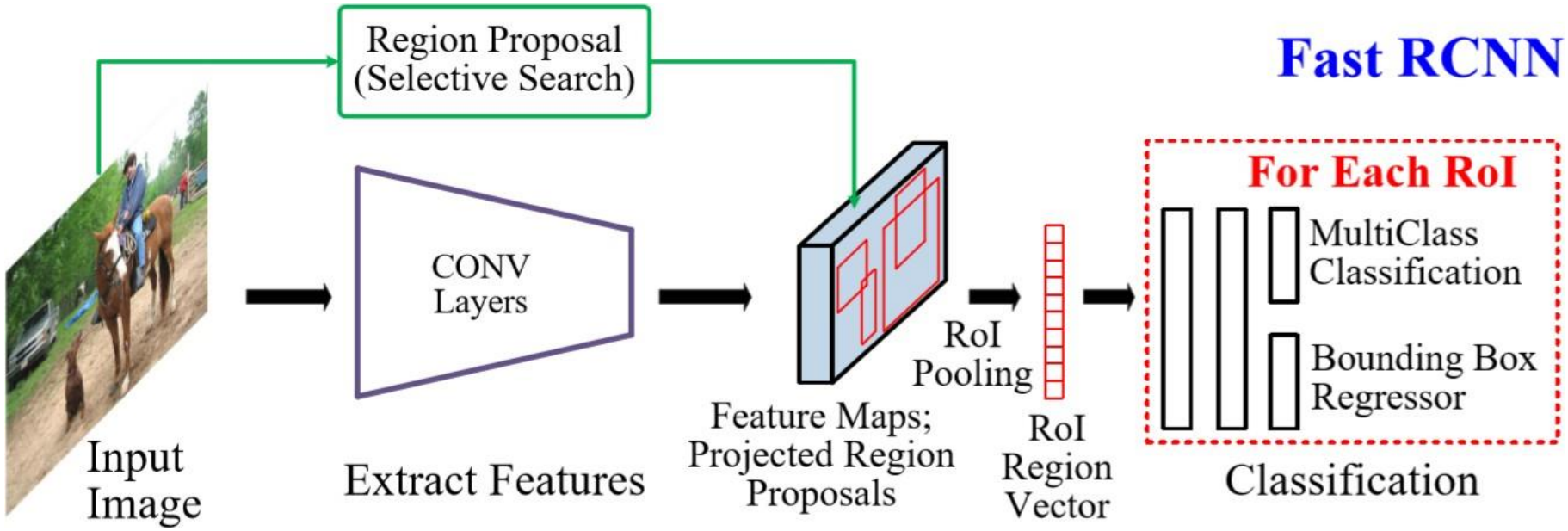
Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**

Fast - RCNN

Simultaneously train detector and B.B. regressor (9x faster)
Boost in performance (mAP) from 58.5% (RCNN) to 70.0%
Still slow detection speed (due to region proposal detection)

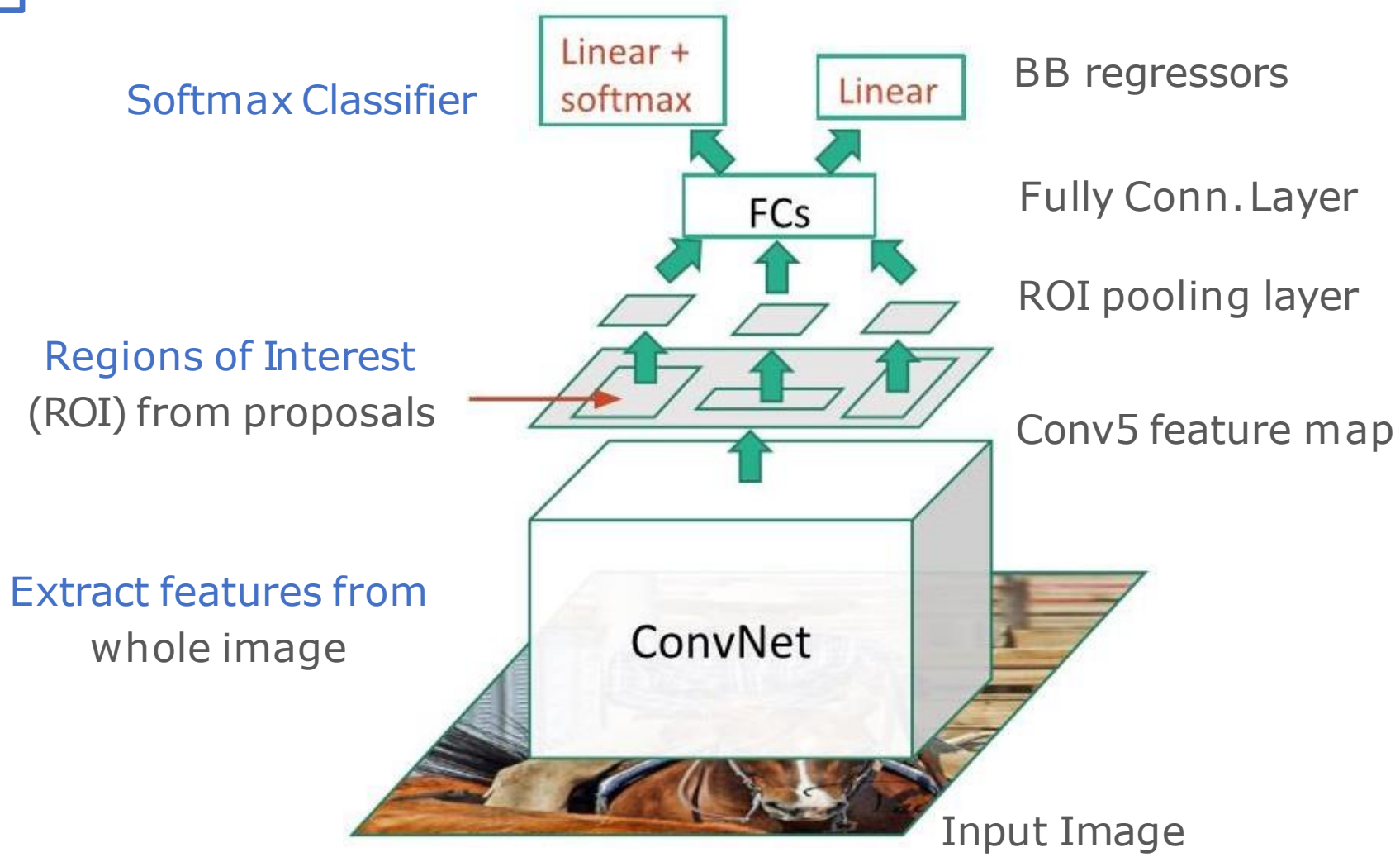
Could we generate object proposals with a CNN model?



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**

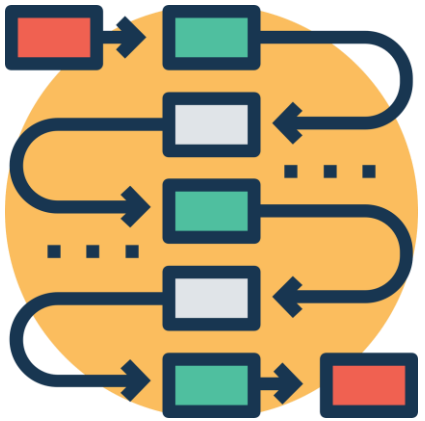
Fast - RCNN



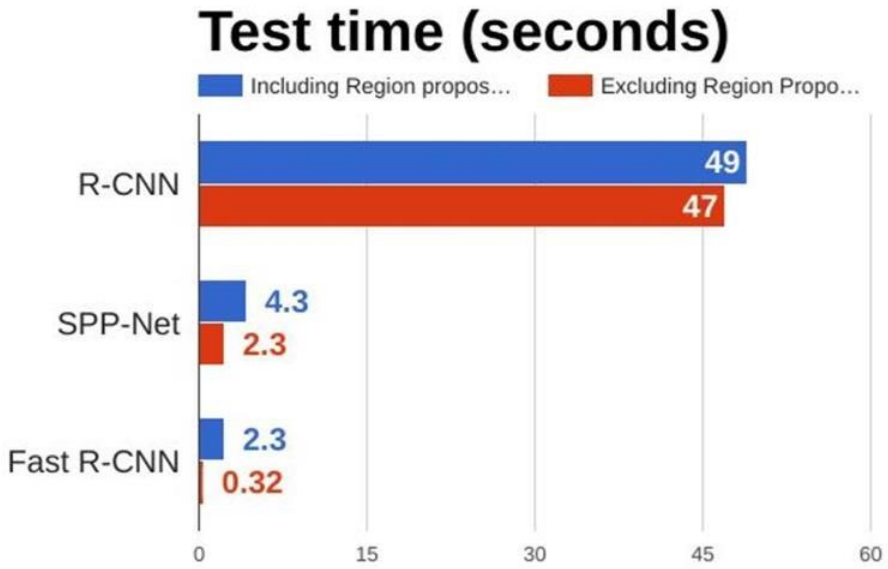
Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**

Fast - RCNN



R-CNN vs SPP vs Fast R-CNN



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**

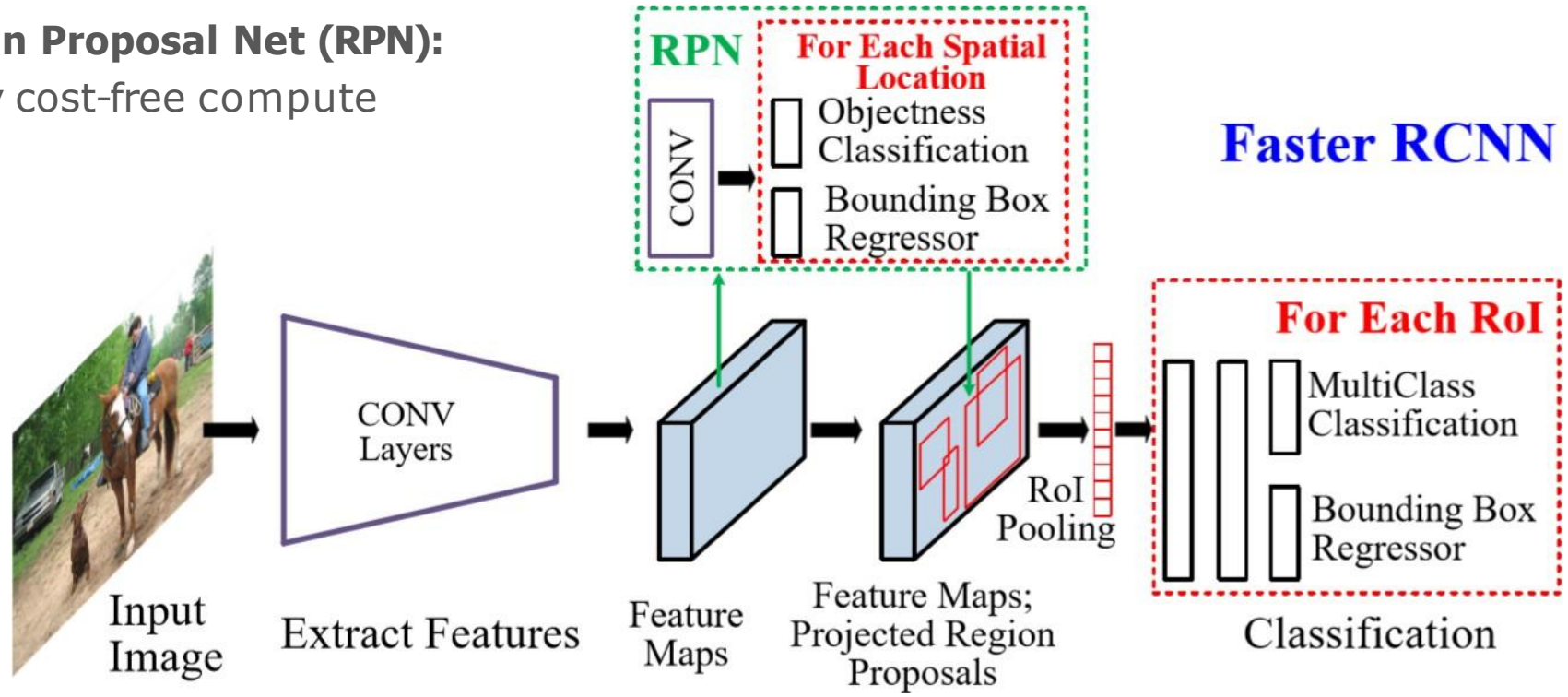
Faster - RCNN

Could we generate O.P with a CNN model?

Yes! First end-to end, near real-time detector

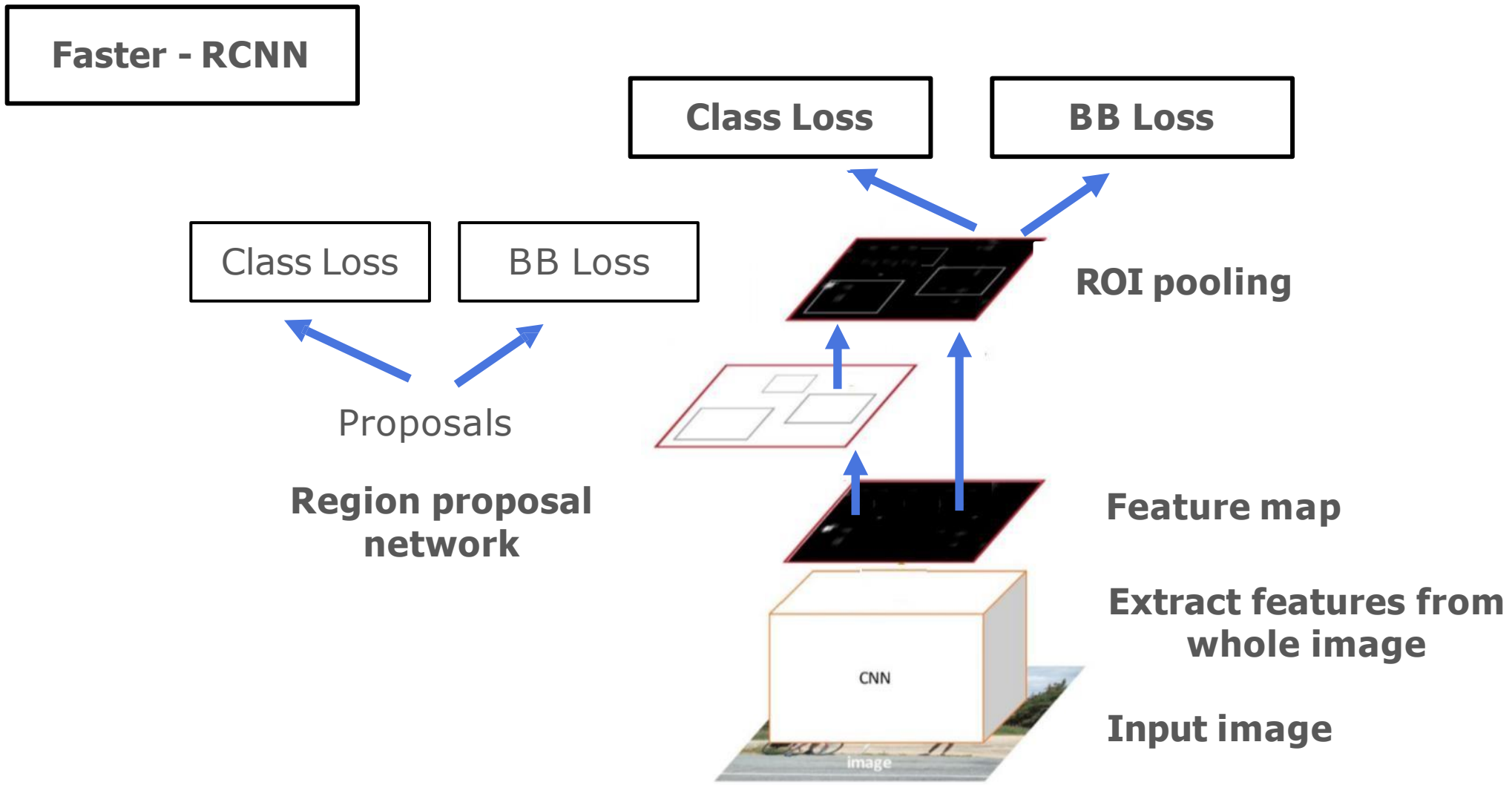
VOC07 mAP: 73.2% 17 FPS with ZFNet

This method introduces the **Region Proposal Net (RPN):** nearly cost-free compute



Introduction: Object Detection in the last 20 years

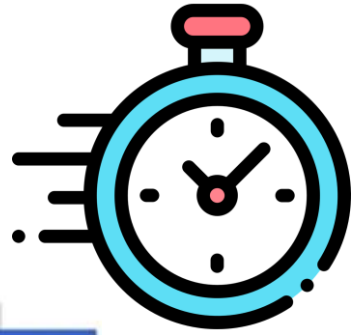
A roadmap of object detection: **Milestone 2-stage Detectors**

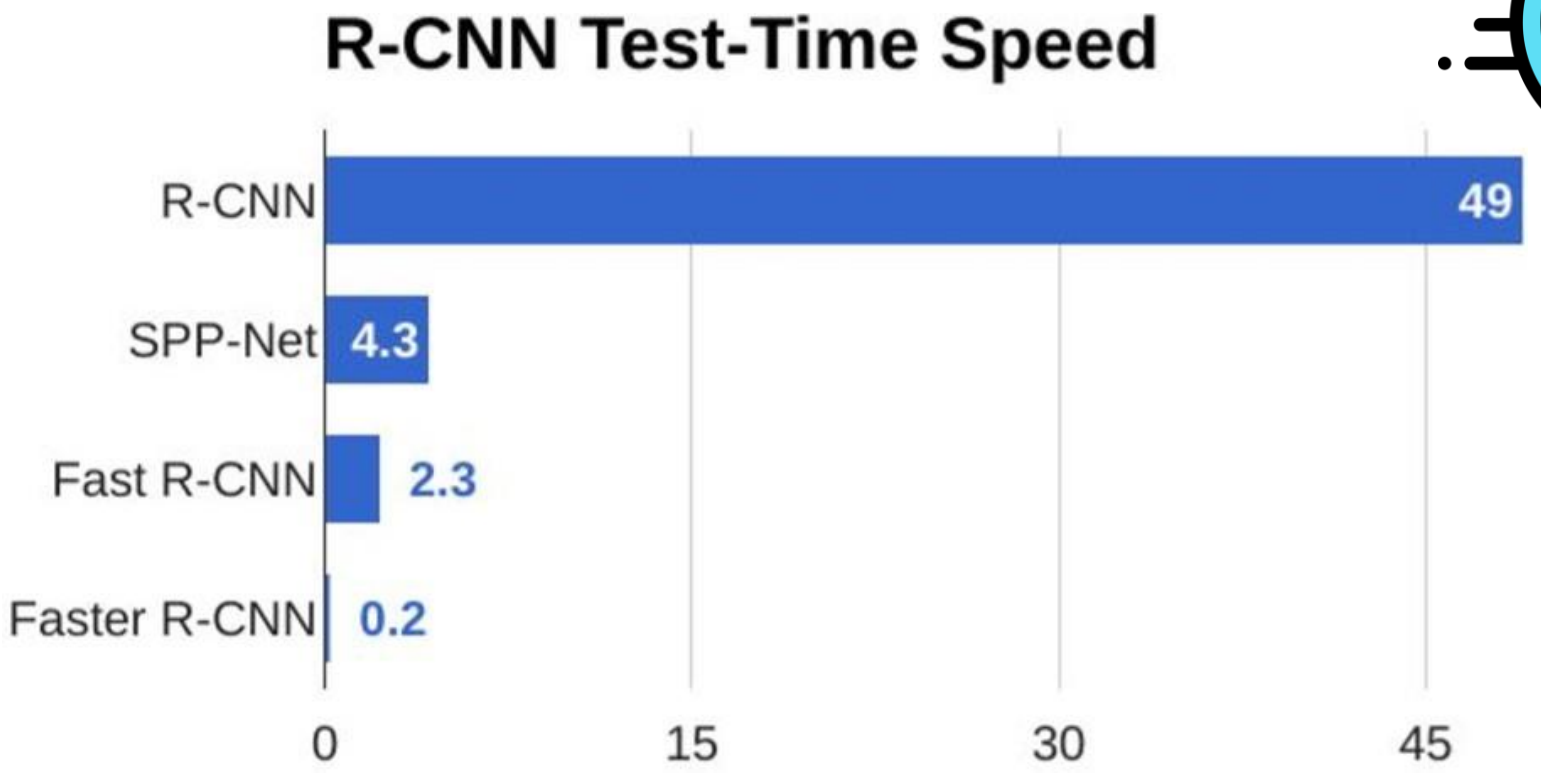


Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**

Faster - RCNN

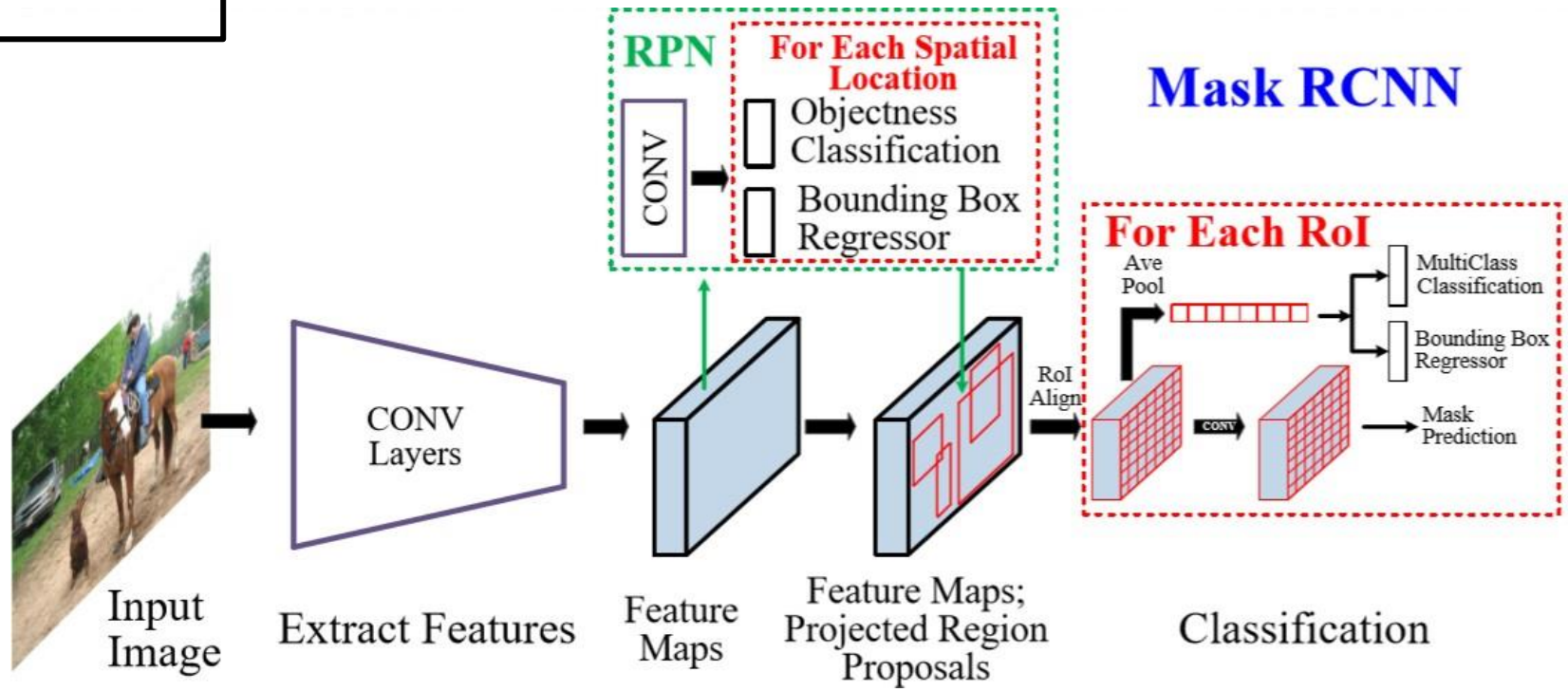




Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**

Mask - RCNN



An extension of Faster RCNN for pixel-wise instance segmentation

Identical fist stage (RPN) but in second branch for binary outputs

Mask-RCNN achived top-results for segmentation and BB detection

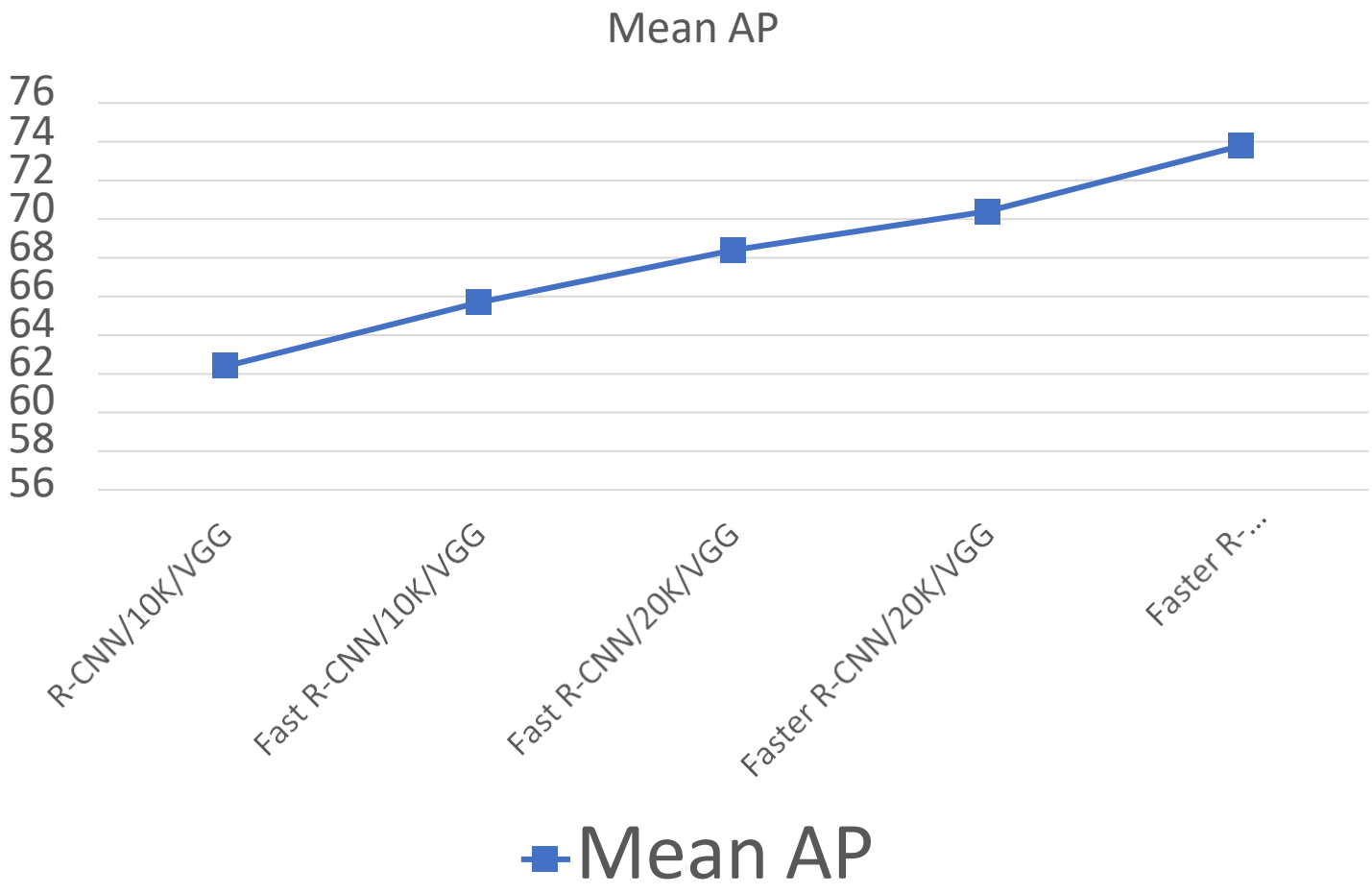
Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**

Mask - RCNN

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 2-stage Detectors**

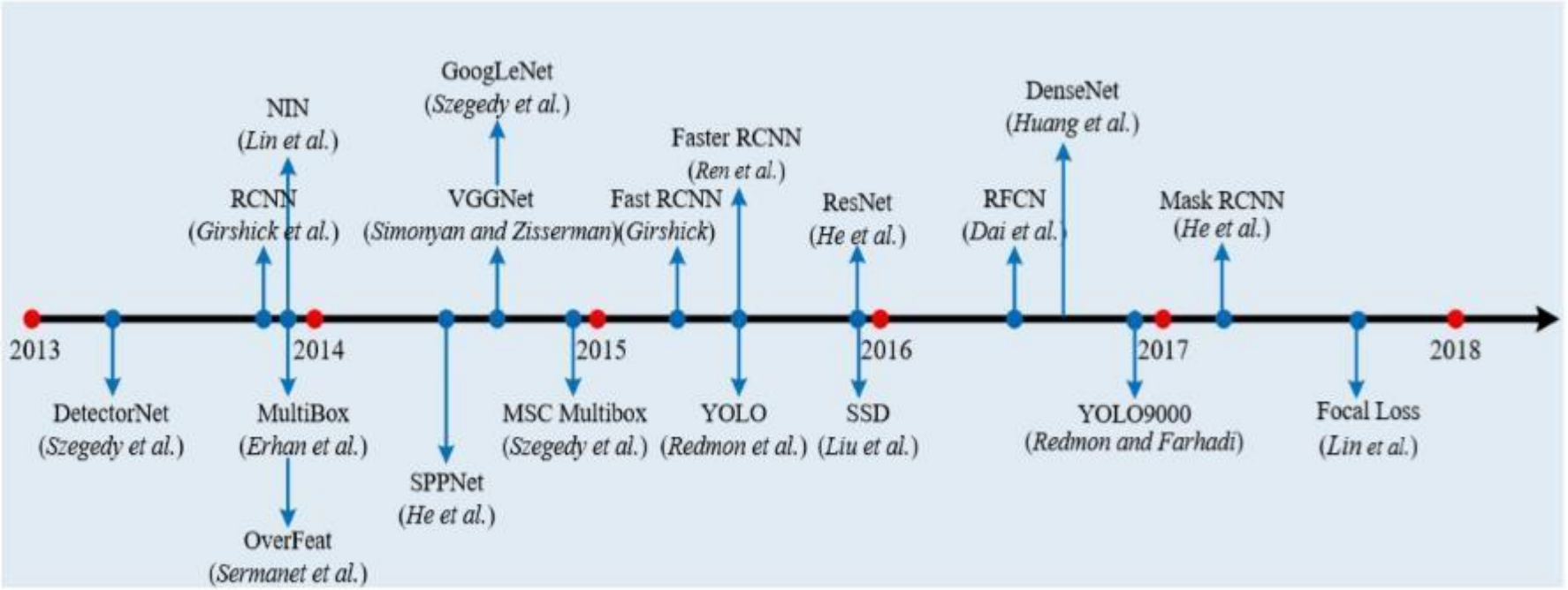


Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 1-stage Detectors**

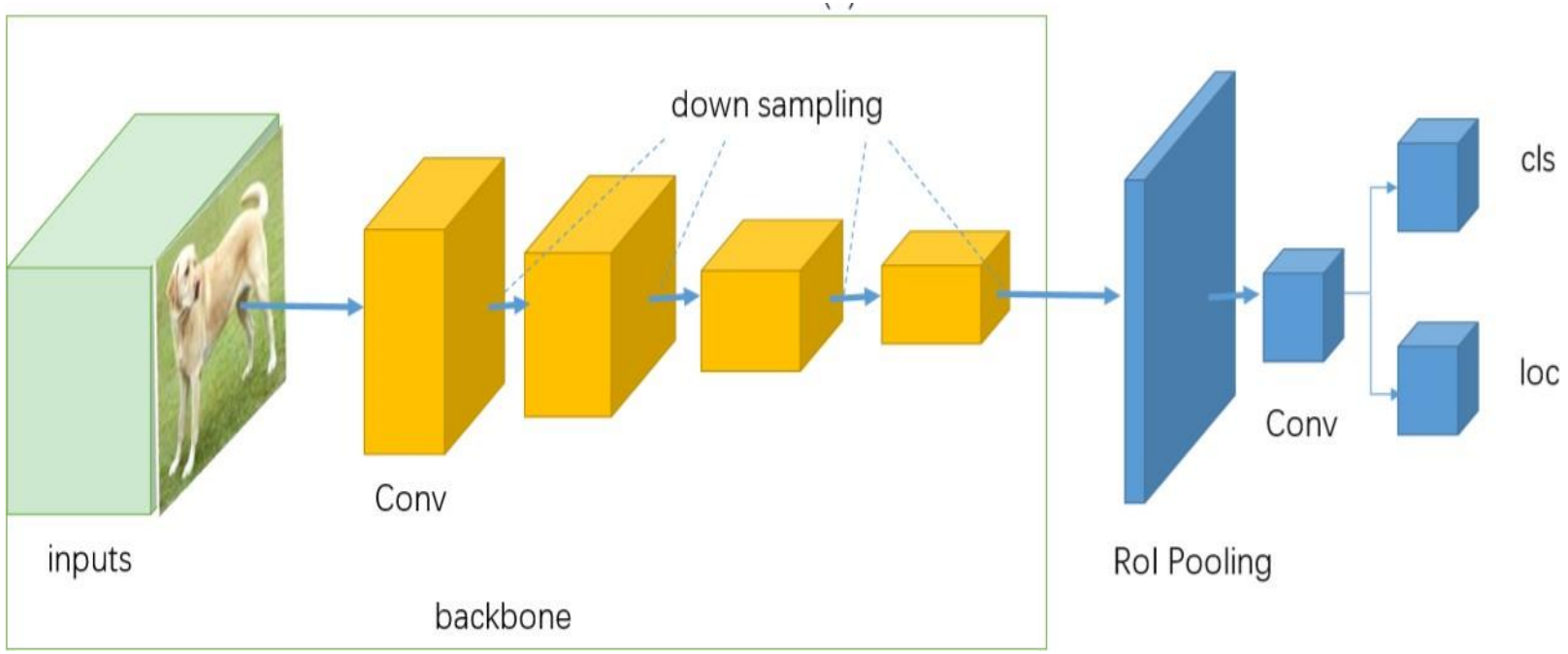
These approaches abandon the **proposal +verification paradigm,** applying instead a **single CNN to the full image**

Several optimizations for improving **accuracy/speed tradeoff**



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 1-stage Detectors**



The choice of backbone layer (i.e. depth)
Is one of the first tradeoffs to be made
when designing a detector → **Accuracy vs Performance**

Introduction: Object Detection in the last 20 years

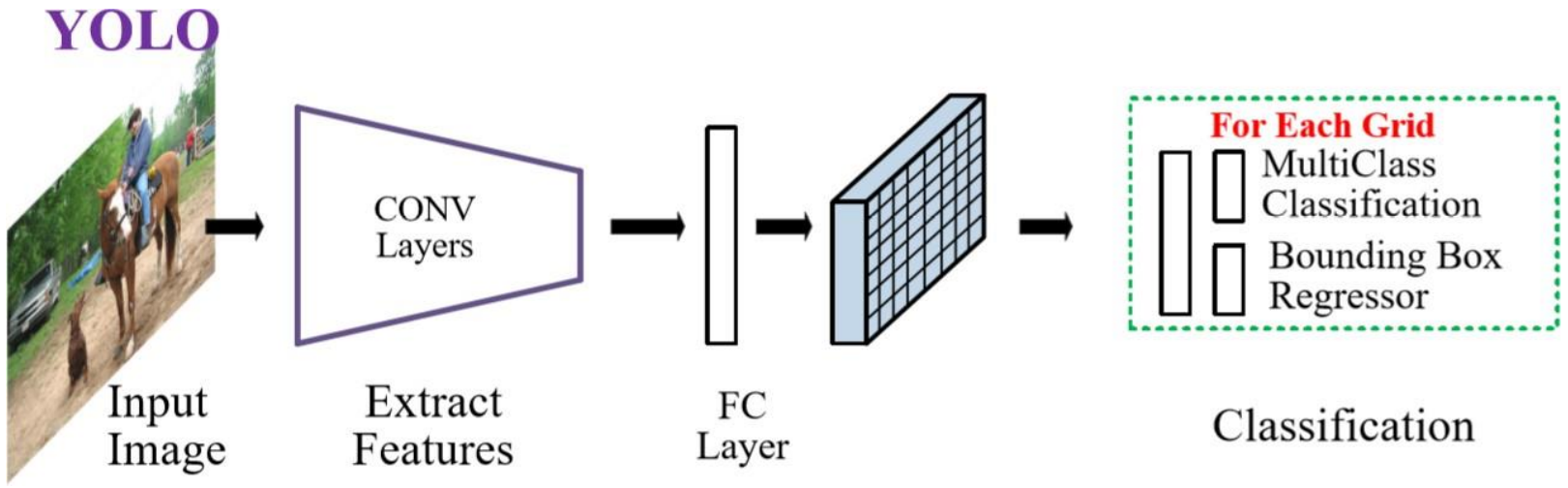
A roadmap of object detection: **Milestone 1-stage Detectors**

YOLO

You Only Look Once, Redmon (2016)

Unified detector: **from pixels to classes and B.B.s**

It directly **predicts detection** using small set of candidate regions
Each grid predicts **C class probabilities**, B locations and scores
It uses the **entire image** for prediction (context), **less FPs**
Fast by design, it can run **@ 45 FPS** and FAST Yolo @155 FPS
More prone to **localization errors**, due to **coarse division** of grids



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 1-stage Detectors**

YOLO version 2 **You Only Look Once, improvements?**

Batch Normalization: Used for accelerating CNN training

Conv with Anchor Boxes: It introduces anchor boxes like in Fater-RCNN, improving recall (7%)

Dimension Clusters for BBs: Uses k-means on the training
Set bounding boxes to get good priors (5%better)

Fire Grained Features: Concatenating multi-layer features

Multi-scale training: Randomly choosing images of different sizes to improve robustness on small object detection

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 1-stage Detectors**

YOLO version 3

You Only Look Once, improvements?

Multi-label classification: Independent logistic classifiers to adapt to more complex datasets (overlapping labels)

Multi-scale: It makes use of three different scale feature maps to achieve improved bounding box predictions

The last convolutional later predicts a 3D tensor encoding class predictions, objectness and bounding box

Feature Extractor: Deeper and more robust, DarkNet, which was inspired by ResNet

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 1-stage Detectors**

SSD

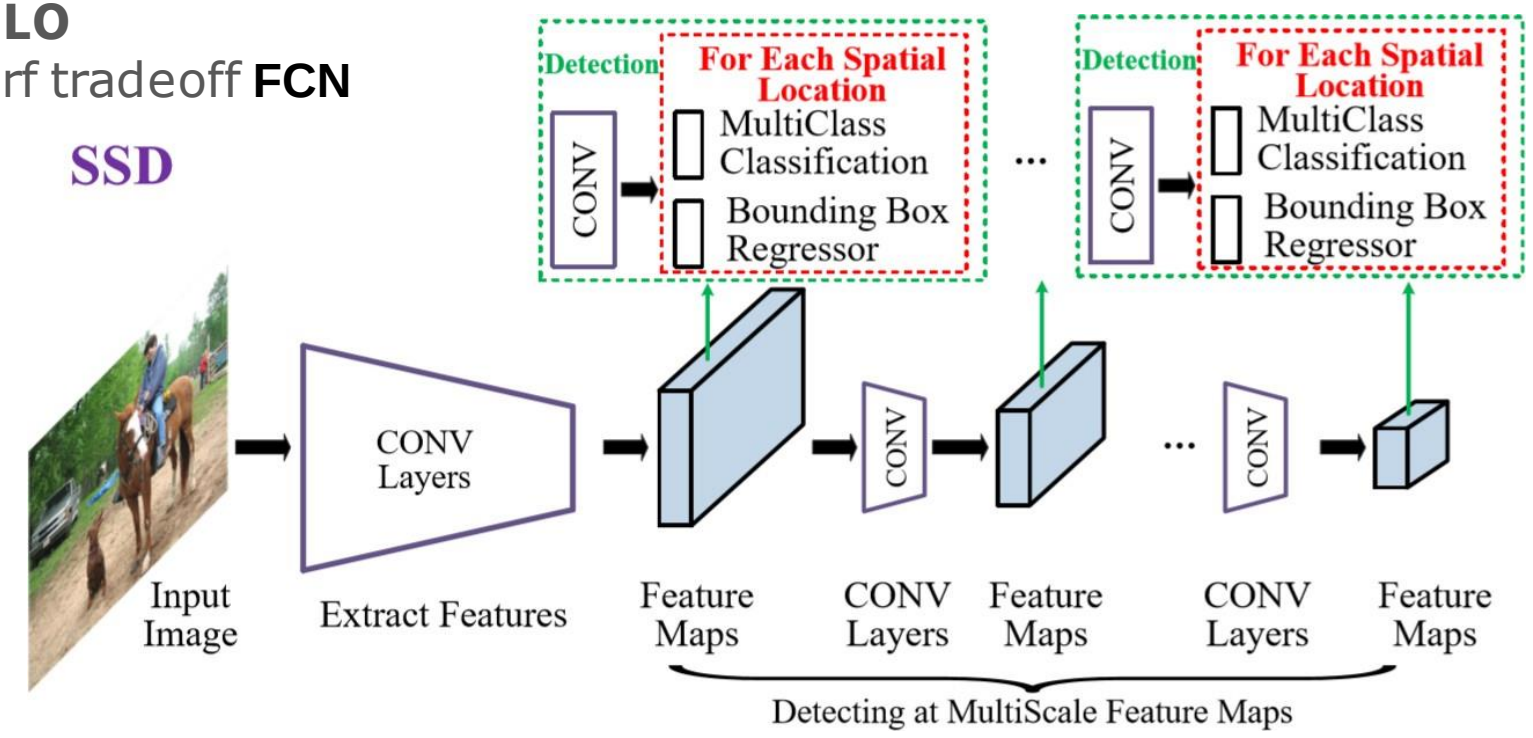
Single Shot Detector, combines RPN/Faster RCNN

1st Idea: real-time, w/o sacrificing accuracy

2nd Idea: shallower layers for **better localization**

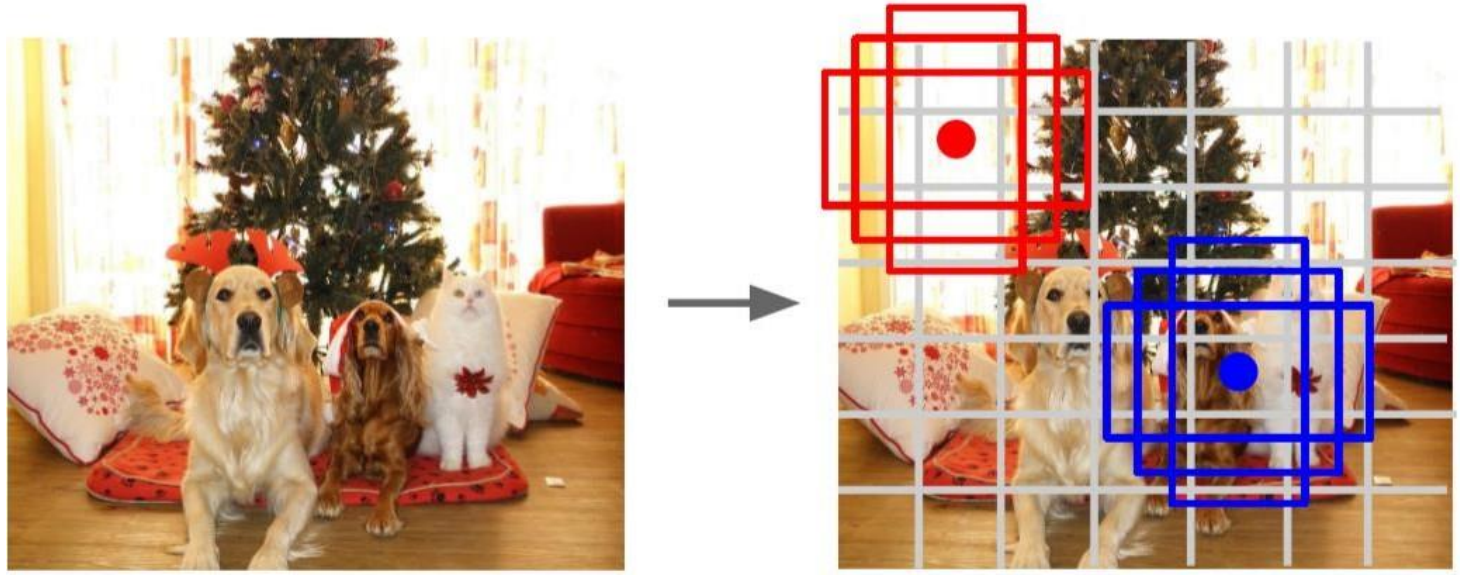
Uses anchors as YOLO

But achieves Acc/Perf tradeoff **FCN**



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 1-stage Detectors**



Input image
3 x H x W

Divide image into grid
7 x 7

Within each grid cell: - Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence) - Predict scores for each of C classes (including background as a class)

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 1-stage Detectors**

Base Network
VGG16
ResNet-101
Inception V2
Inception V3
Inception
ResNet
MobileNet

Object Detection architecture
Faster R-CNN
R-FCN
SSD

Image Size
 # Region Proposals
…

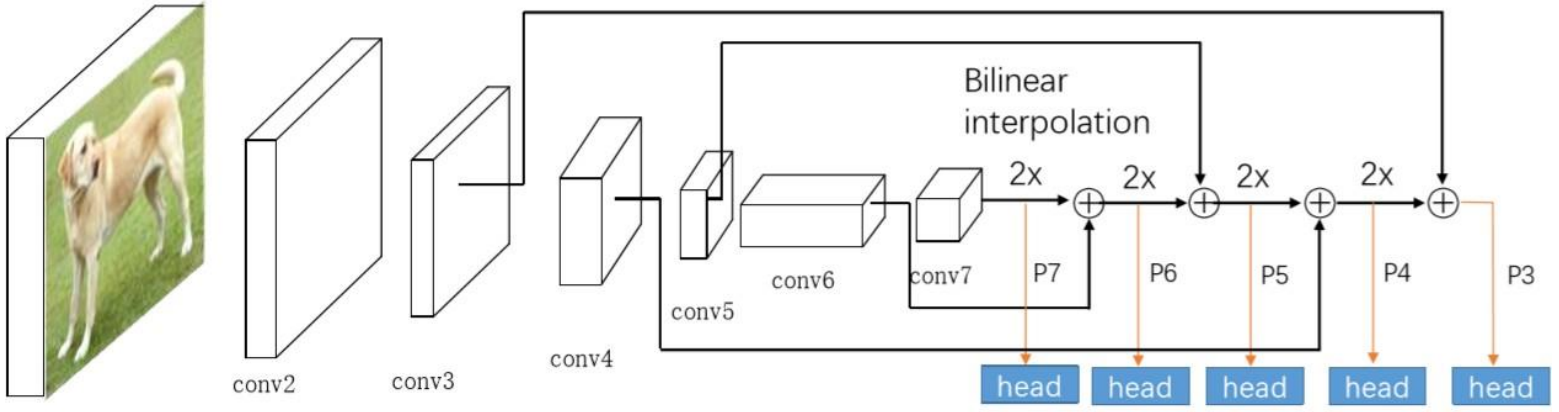
Takeaways
Faster R-CNN is slower but more accurate

SSD is much faster but not as accurate

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 1-stage Detectors**

RetinaNet

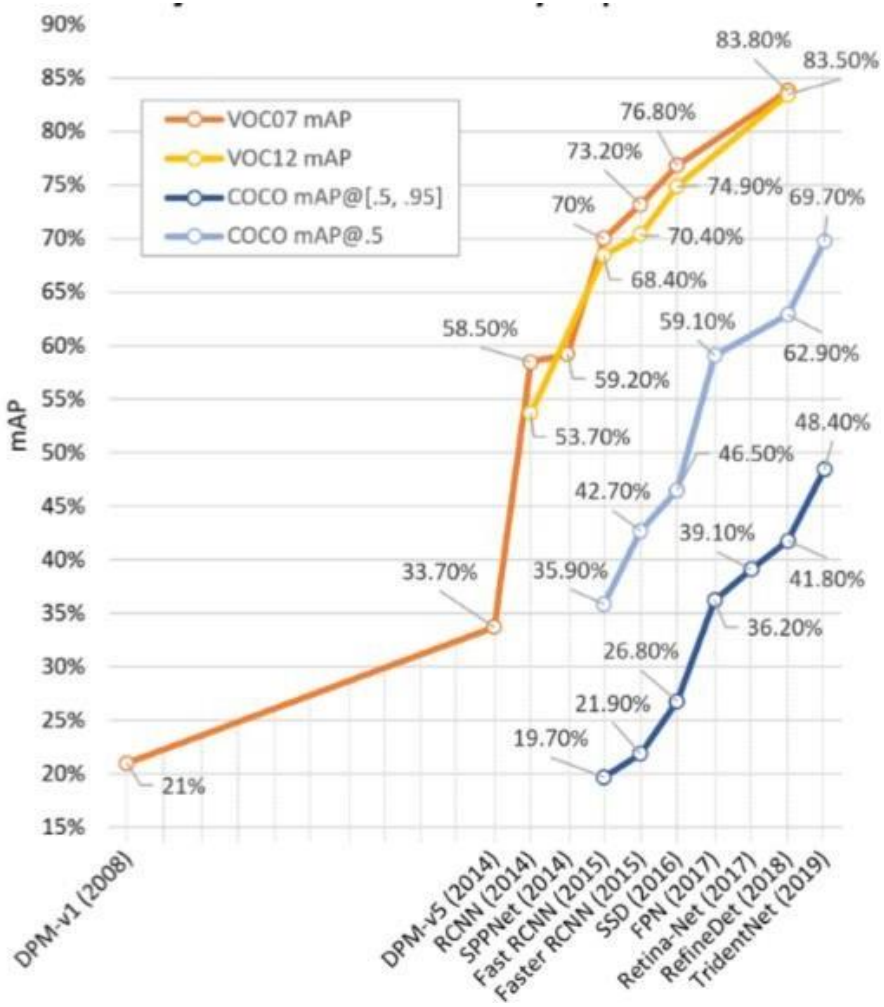


Uses **focal loss** as classification loss function
It down-weights the **loss assigned to "easy" examples**.
It avoids the **vast number of easy negative examples** during training

RetinaNet inherits the fast speed of previous one-stage detectors while greatly overcomes the disadvantage of one-stage detectors difficult to train unbalanced positive and negative examples.

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Milestone 1-stage Detectors**



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Datasets**

Pascal VOC

Visual Object Classes (VOC)

One of the most important datasets & **early computer vision challenges**

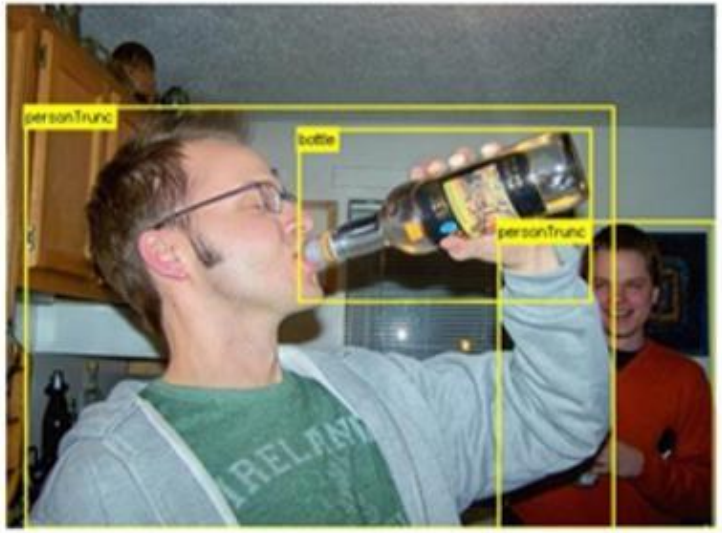
VOC07 & VOC12

5k training + **12k** annot. objects

12k training + **27k** annot. Objects

20 classes of objects

With larger datasets, not used much anymore (**mostly as test-bed**)



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Datasets**

ILSVRC

ImageNet Large Scale Visual Recognition

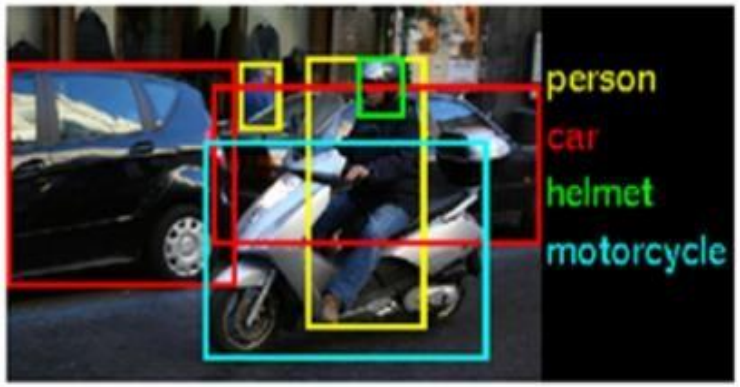
It pushed forward the **state of the art**

200 classes of visual objects!

Number of its image/instances is **two orders of magnitude larger** than VOC

ILSVRC14

517k training images and **534k** annotated objects



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Datasets**

MS-COCO

Most **challenging detection dataset** available to this day (since 2015)

Less categories than ILSVRC, but it contains more **object instances**

MS-COCO-17

164k training +**897k** annot. objects

80 classes of objects

Bounding Boxes +Segmentation

(to aid in localization)

More small objects (1% of image)

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Datasets**

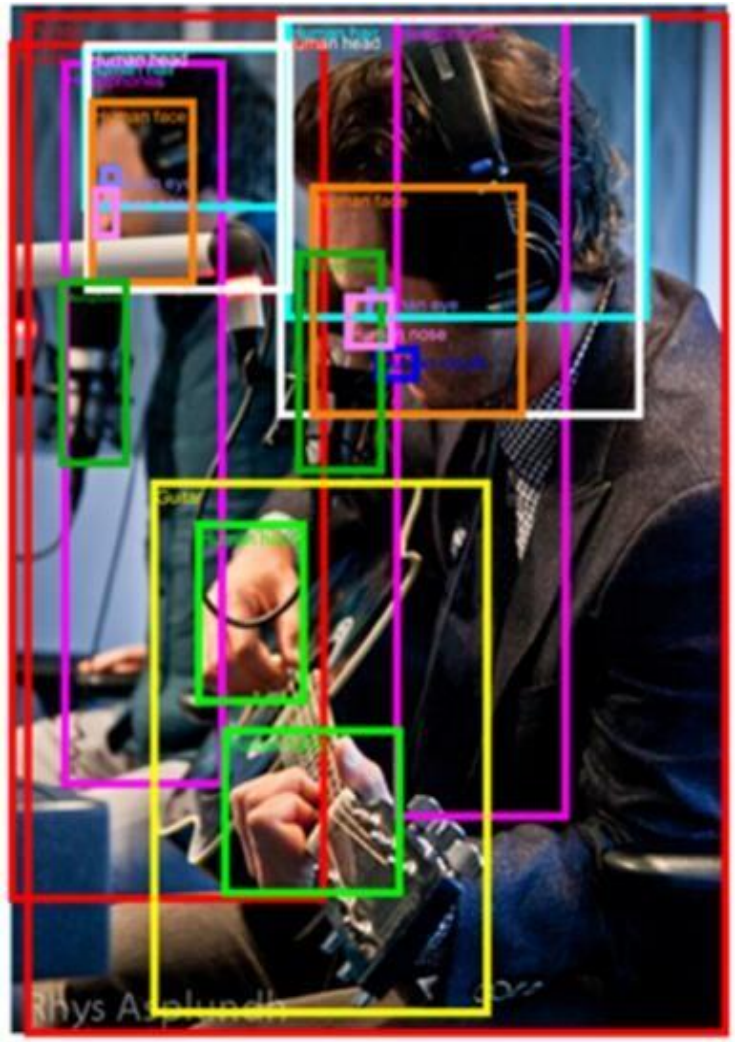
MS-COCO

Open Images Detection (OID)
More recent dataset (since 2018)

MS-COCO+, at unprecedented scale **(600 object categories!)**

Two tasks for this dataset
1. Standard object detection
2. Visual relationship, detecting parings of objects

1,910k images, 15,440 annotations



Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Datasets**

Pedestrian

Very important for **smart cities** applications: **transportation, activity recognition**

Dataset	Year	Description	#Cites
MIT Ped.	2000	One of the first pedestrian detection datasets. Consists of ~500 training and ~200 testing images (built based on the LabelMe database). url: http://cbcl.mit.edu/software-datasets/PedestrianData.html	1515
INRIA	2005	One of the most famous and important pedestrian detection datasets at early time. Introduced by the HOG paper [12]. url: http://pascal.inrialpes.fr/data/human/	24705
Caltech	2009	One of the most famous pedestrian detection datasets and benchmarks. Consists of ~190,000 pedestrians in training set and ~160,000 in testing set. The metric is Pascal-VOC @ 0.5 IoU. url: http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/	2026
KITTI	2012	One of the most famous datasets for traffic scene analysis. Captured in Karlsruhe, Germany. Consists of ~100,000 pedestrians (~6,000 individuals). url: http://www.cvlibs.net/datasets/kitti/index.php	2620
CityPersons	2017	Built based on CityScapes dataset [63]. Consists of ~19,000 pedestrians in training set and ~11,000 in testing set. Same metric with CalTech. url: https://bitbucket.org/shanshanzhang/citypersons	50
EuroCity	2018	The largest pedestrian detection dataset so far. Captured from 31 cities in 12 European countries. Consists of ~238,000 instances in ~47,000 images. Same metric with CalTech.	1

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Datasets**

Face Detection

Very important for **smart cities** applications:
Security, surveillance, biometrics

Dataset	Year	Description	#Cites
FDDB [65]	2010	Consists of ~2,800 images and ~5,000 faces from Yahoo! With occlusions, pose changes, out-of-focus, etc. url: http://vis-www.cs.umass.edu/fddb/index.html	531
AFLW [66]	2011	Consists of ~26,000 faces and 22,000 images from Flickr with rich facial landmark annotations. url: https://www.tugraz.at/institute/icg/research/team-bischof/lrs/downloads/aflw/	414
IJB [67]	2015	IJB-A/B/C consists of over 50,000 images and videos frames, for both recognition and detection tasks. url: https://www.nist.gov/programs-projects/face-challenges	279
WiderFace [68]	2016	One of the largest face detection dataset. Consists of ~32,000 images and 394,000 faces with rich annotations i.e., scale, occlusion, pose, etc. url: http://mmlab.ie.cuhk.edu.hk/projects/WIDERFace/	193
UFDD [69]	2018	Consists of ~6,000 images and ~11,000 faces. Variations include weather-based degradation, motion blur, focus blur, etc. url: http://www.ufdd.info/	1
WildestFaces [70]	2018	With ~68,000 video frames and ~2,200 shots of 64 fighting celebrities in unconstrained scenarios. The dataset hasn't been released yet.	2

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Datasets**

Text Detection

Very important for **smart cities** applications:
information retrieval, AR

Dataset	Year	Description	#Cites
ICDAR	2003	ICDAR2003 is one of the first public datasets for text detection. ICDAR 2015 and 2017 are other popular iterations of the ICDAR challenge [72, 73]. url: http://rrc.cvc.uab.es/	530
STV	2010	Consists of ~350 images and ~720 text instances taken from Google StreetView. url: http://tc11.cvc.uab.es/datasets/SVT_1	339
MSRA-TD500	2012	Consists of ~500 indoor/outdoor images with Chinese and English texts. url: http://www.iapr-tc11.org/mediawiki/index.php/MSRA_Text_Detection_500_Database_(MSRA-TD500)	413
IIIT5k	2012	Consists of ~1,100 images and ~5,000 words from both streets and born-digital images. url: http://cvit.iiit.ac.in/projects/SceneTextUnderstanding/IIIT5K.html	165
Syn90k	2014	A synthetic dataset with 9 million images generated from a 90,000 vocabulary of multiple fonts. url: http://www.robots.ox.ac.uk/~vgg/data/text/	246
COCOText	2016	The largest text detection dataset so far. Built based on MS-COCO, Consists of ~63,000 images and ~173,000 text annotations. https://bgshih.github.io/cocotext/.	69

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Datasets**

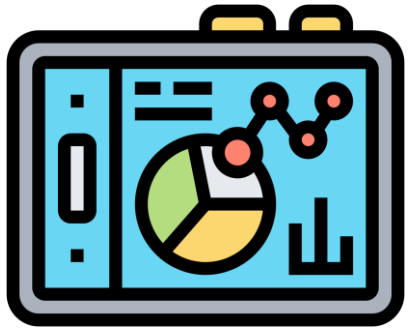
Traffic Signs

Very important for **smart cities** applications:
transportation, physical asset management

Dataset	Year	Description	#Cites
TLR [79]	2009	Captured by a moving vehicle in Paris. Consists of ~11,000 video frames and ~9,200 traffic light instances. url: http://www.lara.prd.fr/benchmarks/trafficlightsrecognition	164
LISA [80]	2012	One of the first traffic sign detection dataset. Consists of ~6,600 video frames, ~7,800 instances of 47 US signs. url: http://cvrr.ucsd.edu/LISA/lisa-traffic-sign-dataset.html	325
GTSDB [81]	2013	One of the most popular traffic signs detection dataset. Consists of ~900 images with ~1,200 traffic signs capture with various weather conditions during different time of a day. url: http://benchmark.ini.rub.de/?section=gtsdb&subsection=news	259
BelgianTSD [82]	2012	Consists of ~7,300 static images, ~120,000 video frames, and ~11,000 traffic sign annotations of 269 types. The 3D location of each sign has been annotated. url: https://btsd.ethz.ch/shareddata/	224
TT100K [83]	2016	The largest traffic sign detection dataset so far, with ~100,000 images (2048 x 2048) and ~30,000 traffic sign instances of 128 classes. Each instance is annotated with class label, bounding box and pixel mask. url: http://cg.cs.tsinghua.edu.cn/traffic%2Dsign/	111
BSTL [84]	2017	The largest traffic light detection dataset. Consists of ~5000 static images, ~8300 video frames, and ~24000 traffic light instances. https://hci.iwr.uni-heidelberg.de/node/6132	21

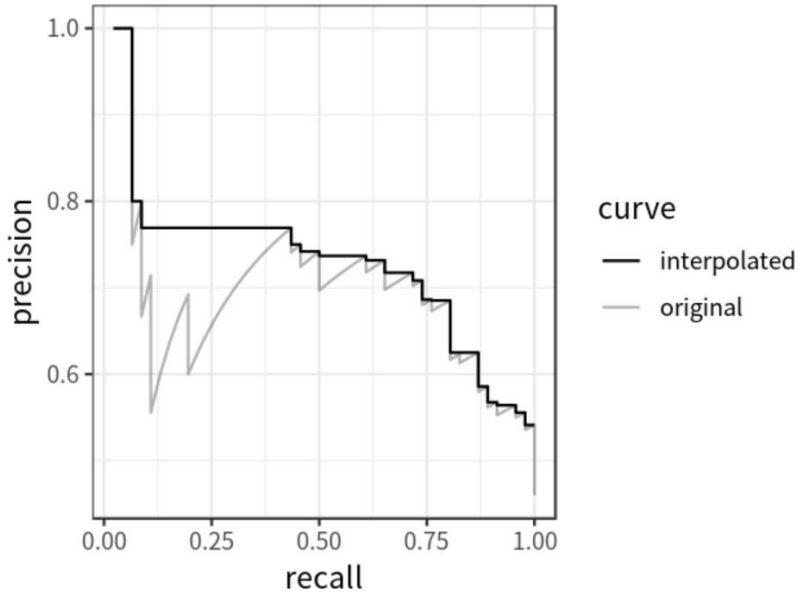
Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Metrics**



In recent years, the most frequently used **metric for evaluation for object detection**

Defined as the **average precision under different recalls**, evaluated per category



Typically **interpolated** using

$$p_{interp}(r) = \max_{r' \geq r} p(r')$$

AP is the area under the **precision-recall curve**

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) p_{interp}(r_{i+1})$$

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Metrics**

- ML usually gives scores or probabilities, so threshold

- Too low → many detections →

  low precision, high recall

- Too high → few detections →

  high precision, low recall

Right tradeoff depends on application

Detecting cancer cells in tissue: need high recall

Detecting edible mushrooms in forest: need high precision

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Metrics**

Mean Avg. Precision

The **calculation of AP only involves one class**. However, in object detection, there are usually **K>1classes**.

Thus, in order to compare the performance over all object categories, the **mean AP (mAP)** averaged over all the object categories is usually used as the **final metric of performance**

Mean average precision (mAP) is defined as the mean of AP across all K classes:

$$mAP = \frac{\sum_{i=1}^{K} AP_i}{K}$$

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Metrics**

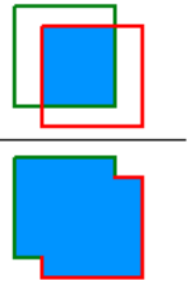
Confidence and IoU

Confidence score, is the **probability** that an anchor box contains an object. It is usually **predicted by a classifier.**

Intersection over Union (IoU), is defined as the area of the intersection divided by the area of the union of a predicted **bounding box (Bp)** and a **ground-truth box (Bgt):**

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$



Both **confidence score** and IoU are used as the criteria that determine whether a detection is a true positive or a false positive

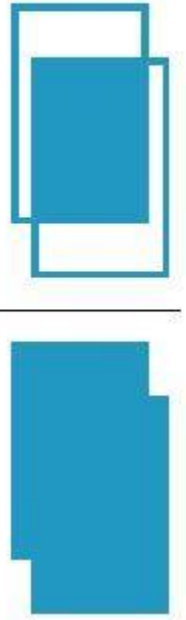
Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Metrics**

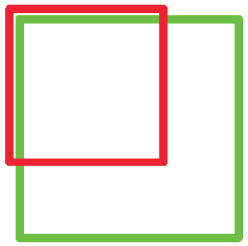
Predicted person bounding box

Ground truth person bounding box

$$\text{Score} = \frac{\text{Area of overlap}}{\text{Area of union}}$$

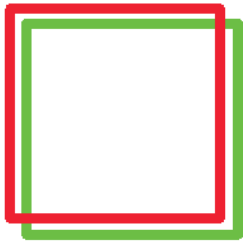


IoU: 0.4034



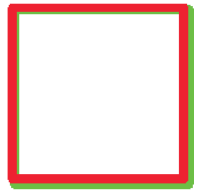
Poor

IoU: 0.7330



Good

IoU: 0.9264



Excellent

Introduction: Object Detection in the last 20 years

A roadmap of object detection: **Metrics**

IoU and mAP

Typically a **combination of mAP and 0.5 IoU threshold** has become the de facto metric for object detection for years

However, after 2014 due to the **popularity of MS-COCO datasets**, researchers started to pay **more attention to the accuracy of the BB**

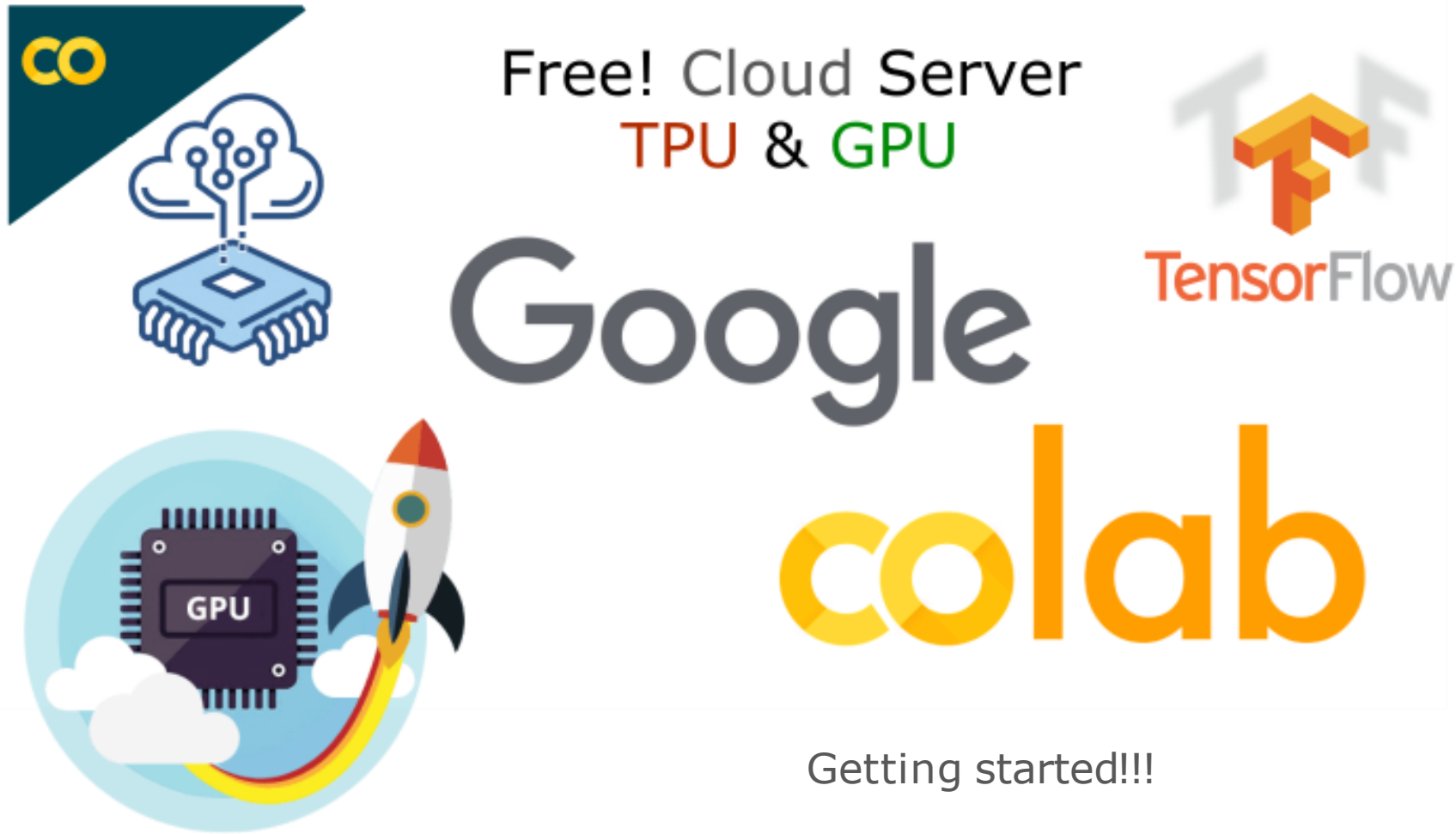
Instead of using a fixed IoU threshold, MS-COCO **AP is averaged over multiple IoU** threshold between **0.5 (coarse)** and **0.95 ("perfect")**

This change of metric has **encouraged more accurate localization** and may be of great importance in real-world applications

There are other proposals, but so fat the **VOC/COCO-based mAP** is still the **most frequently used evaluation metric** for object detection

Introduction to OD using Google Colab

Why using Google Colab for your projects I



Getting started!!!

https://www.youtube.com/watch?v=inN8seMm7UI

Introduction to OD using Google Colab

Why using Google Colab for your projects Ⅲ: Roadmap

What are we doing next?

- Brief introduction to **Google Colab**
- Train a **Yolo v3** using Darknet via **Colab 12GB-RAM GPU**.
- Discuss how to deal with some of the **limitations of Colab**
    - Working directly from the files on your computer.
    - **Configure your notebook** to install everything you need
    - Get your trained weights on your computer during the training.
    - While the **notebook is training** you can **check how it is going** using your trained weights in your computer.

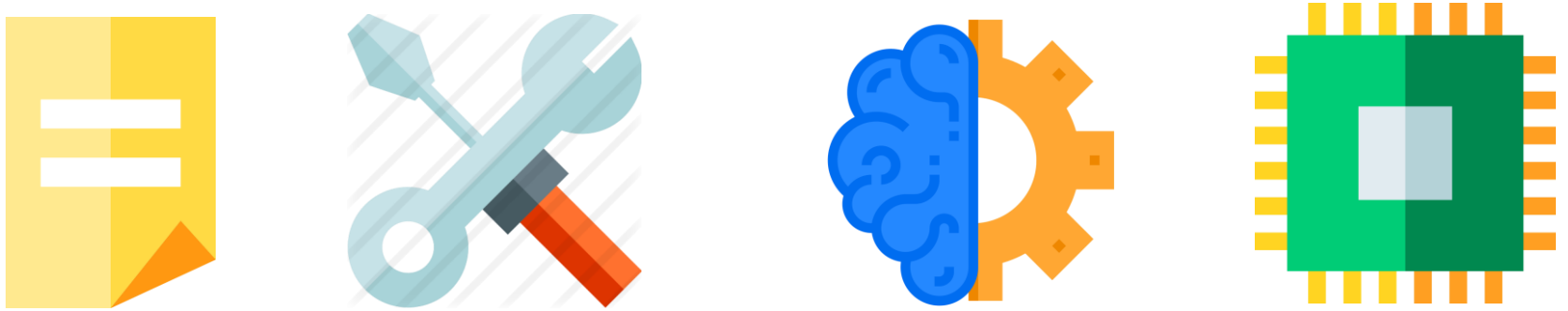
Introduction to OD using Google Colab

Why using Google Colab for your projects IV: What it is?

Colab is a free cloud service for machine ML education and research.

It provides a **runtime fully configured for deep learning** and…

Free-of-charge access to a robust GPU!!!



Colab is the perfect place to do your research, tutorials or projects.

Obviously, not everything can be wonderful… Let us see why!

Introduction to OD using Google Colab

Why using Google Colab for your projects V: Limitations

Colab runtime is volatile. Your Virtual Machine (VM) will blow up after 12 hours and..... **will disappear in the space!**

- This means that your **VM and all files are lost after 12 hours.**
- After 12 hours **you'll have to reconfigure your runtime** in order to start training again.

- **You work with a remote VM.** You don't have direct access to the VM filesystem. You have to upload your files in order to be used and download the files created during the training.

Introduction to OD using Google Colab

Why using Google Colab for your projects VI: Fixes I

How can we solve these limitations?

Remote filesystem - Google has included Drive API on the notebooks, making very easy to map your Google Drive as a VM drive.

Besides, we can synchro one folder of our computer to Google Drive. That's it! now we'll have direct access to your Colab filesystem.

You'll be able to work with your YOLO config files locally and test on the notebook instantly.

Introduction to OD using Google Colab

Why using Google Colab for your projects VI: Fixes II

How can we solve these limitations?

Volatile VM. Files are lost every 12 hours - Google drive to the rescue again. We'll save our files directly to the mapped drive.

Reconfiguring entire runtime every time - Basically, speeding up the process.

We can configure the entire runtime to train YOLOv3 model using Darknet in less than a minute and just with one manual interaction.

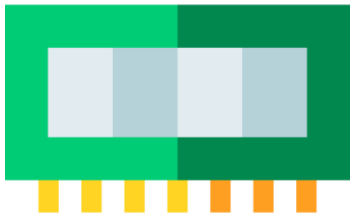
Introduction to OD using Google Colab

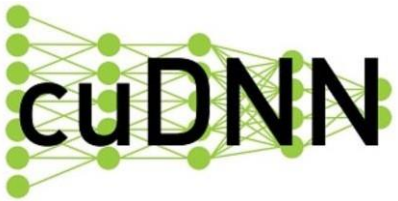
Running YOLO in Google Colab I

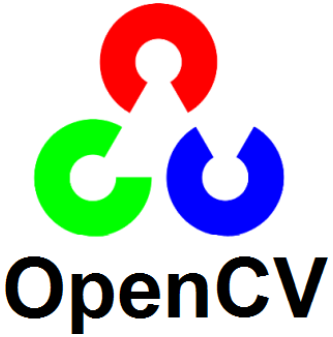
What we need to run YOLO in Darknet?

To train a **YOLO mod**el using darknet we need the following

- A powerful **GPU**
- Nvidia **CUDA** and **cuDNN**
- **OpenCV**: Open Source Computer Vision Library
- **Darknet** model







Introduction to OD using Google Colab

Running YOLO in Google Colab II

Colab notebooks have almost all of them pre-configured for you!

We need configure for ourselves the following:

To run Darknet

1. Configure **Runtime type to use GPU**. (Only the first time)
2. Get access to **Google Drive** and map as network drive
3. Install **cuDNN**
4. Clone and compile **Darknet**

To train YOLO

1. YOLO model configuration file (yolo-…-.cfg)
2. An Image data set
3. Data configuration file (obj.data)
4. The pre-trained weights file (file.weights)

Introduction to OD using Google Colab

Running YOLO in Google Colab III

Creating the notebook

Notebooks can be shared. I can create a notebook and share with you, but if you open it, you will have your own VM runtime to play with. You can make a copy of the notebook and apply your own code.

You can access to the notebook in two different ways

1. You can access if shared to you

2. You can Clone a Github repo and upload to your Google Drive. From there, you'll be able to access and work on it.

https://github.com/AlexeyAB/darknet

Introduction to OD using Google Colab

Running YOLO in Google Colab IV

What's YOLO and Darknet?

YOLO, acronym of **You Only Look Once** is a state-of-the-art, real-time object detection system created by **Joseph Redmon**.

Darknet, also created by **Joseph Redmon** is an open source neural netw framework written in C and CUDA.

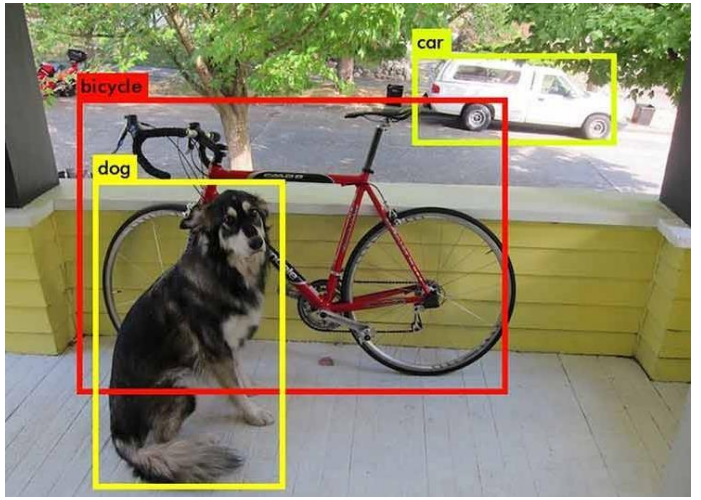
It is fast, easy to install, and supports **CPU and GPU computation**.

What you have to prepare before you start training the model, **is the dataset.**

Next, we present a cheat sheet of all the config files and dataset configuration you need in order to train your YOLO model

Introduction to OD using Google Colab

Running YOLO in Google Colab V



General flow for training a model

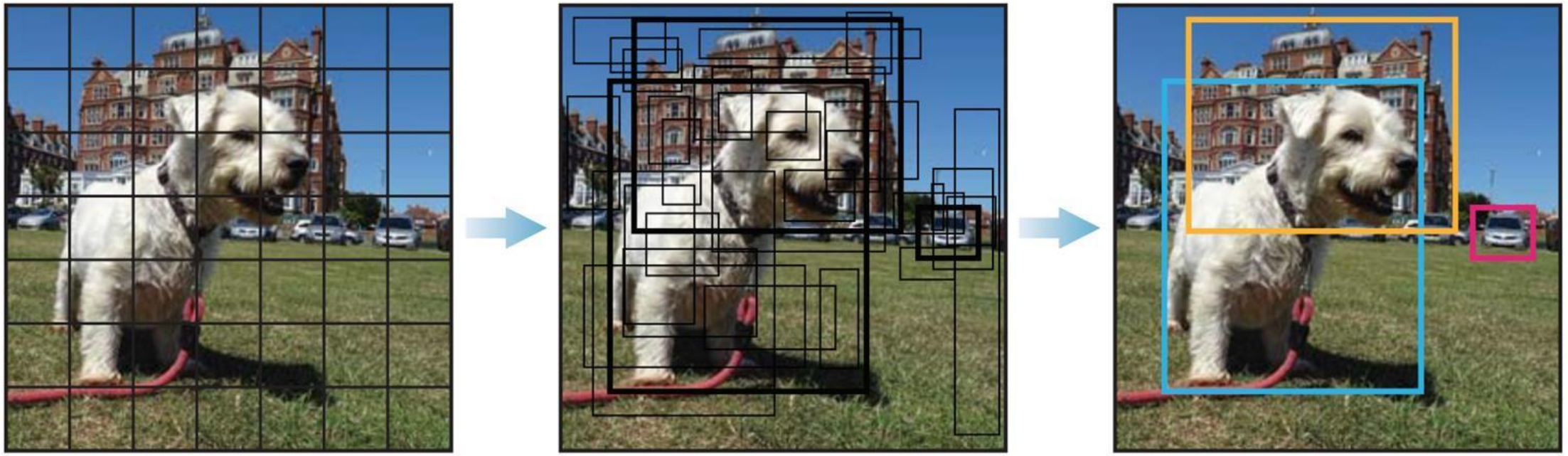
To train a model we follow these steps:

- Preparing the dataset
- Input an image we know what contains
- Process this through our neural network and get a result
- Compare the result to what we know it really contains
- Adjust the weights in order to minimize the difference between what we obtained and we know we had.

This process is repeated until we get the best weights we can.
Once done, store the model weights and config for future use

Introduction to YOLO v3

The You Only Look Once model



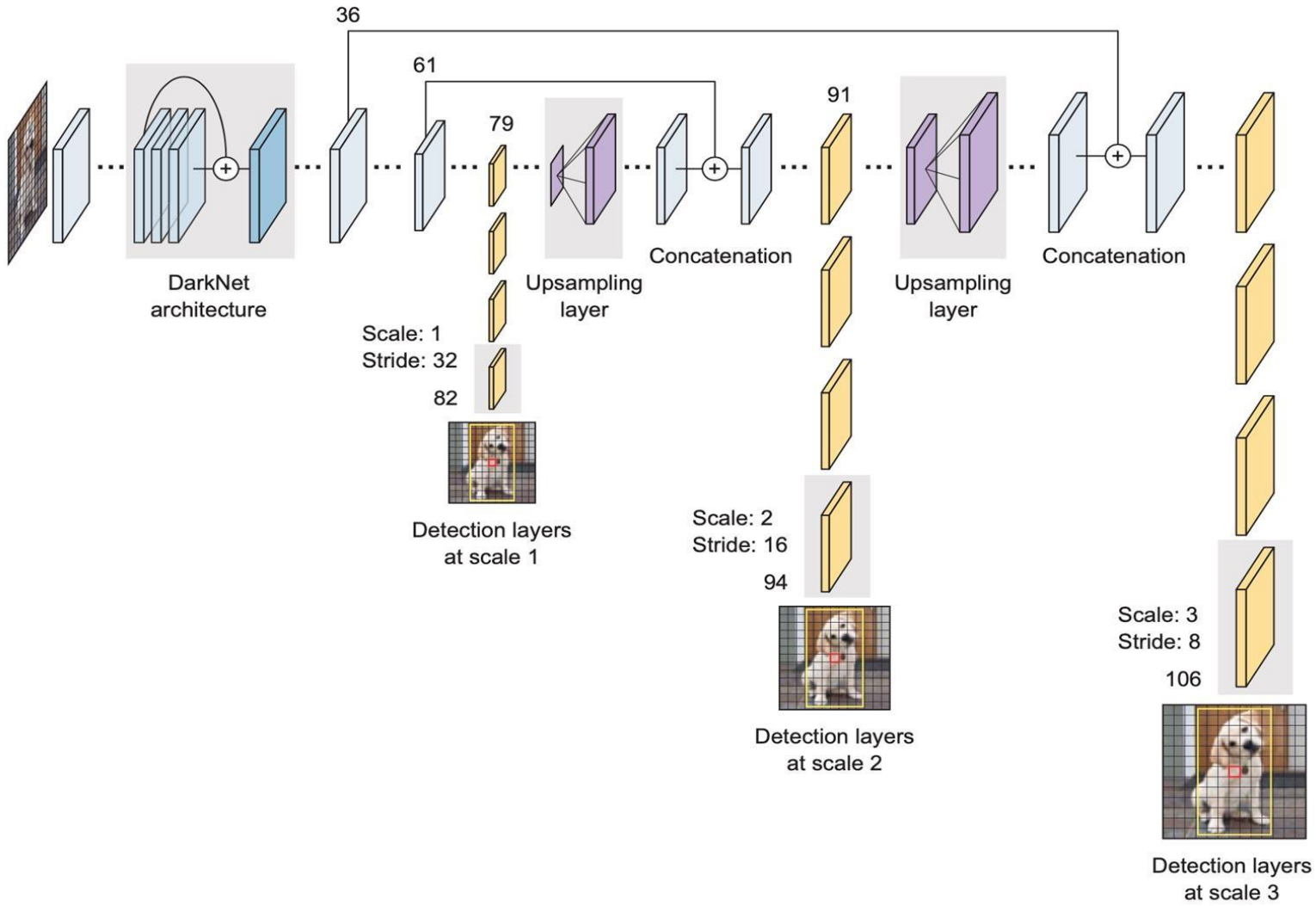
Introduction to YOLO v3

Using Darknet as backbone

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	34	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

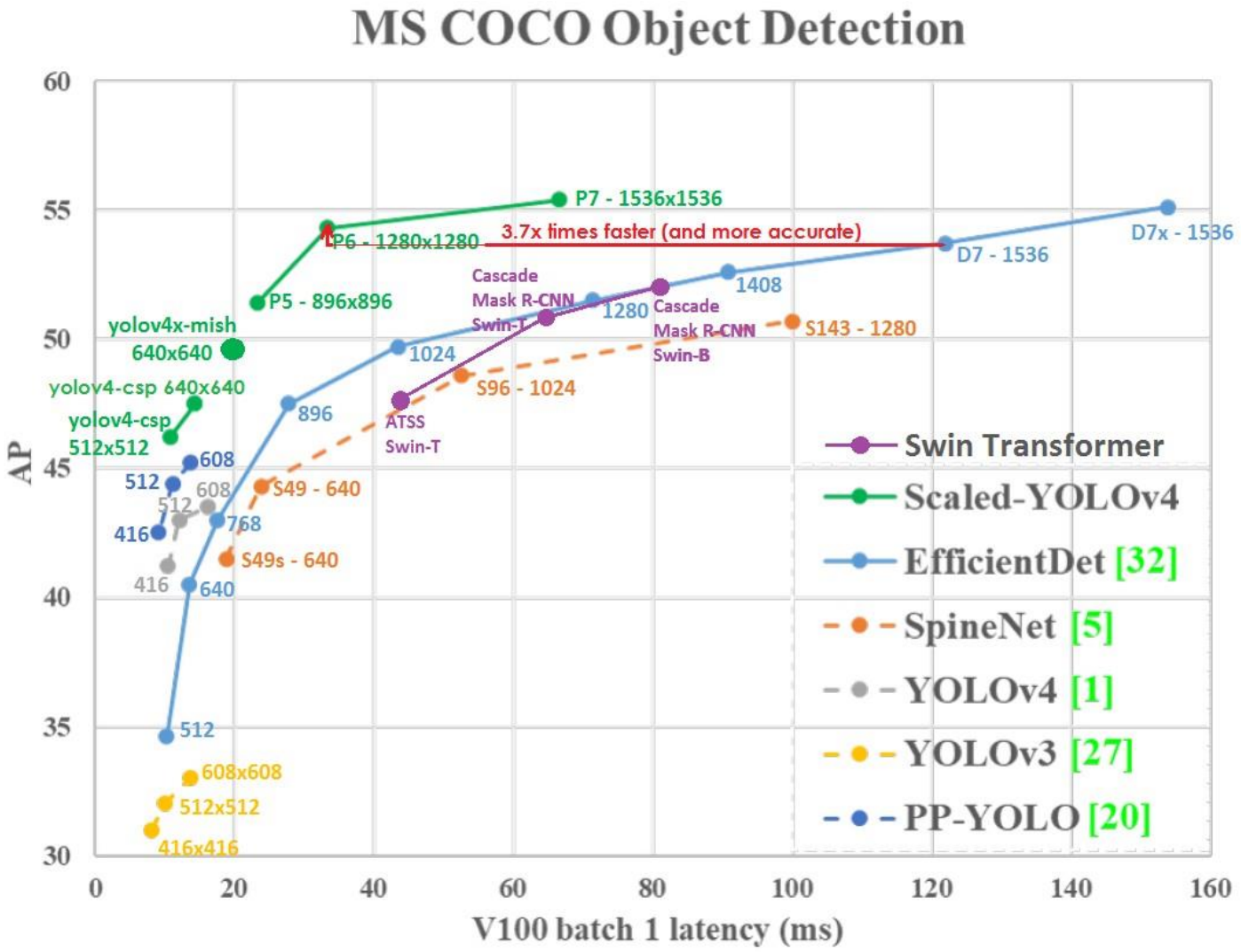
Introduction to YOLO v3

The YOLO v3 architecture



Introduction to YOLO v3

The YOLO v3 compared against other state of the art Object Detectors



Introduction to YOLO v3

Running YOLO in Google Colab VI

Preparing the dataset: This is a crucial step and performance of your model depend on the quality of data you collected.

a) Search for **images and videos** related your problem domain

b) Be aware of what **problem you are going to solve.**

c) Use a **labeling an annotation tool**, there are several out there

d)The **dataset** can come from images of your own, crowdsourced o from videos, annotation tools support them all

e) As we are used a deep learning framework, **image augmentation** is a must to avoid overfitting issues

Introduction to YOLO v3

Running YOLO in Google Colab VII

Preparing the dataset (video)

If You have **video** and want to take a screenshot then use **ffmpeg**

Link: https://ffmpeg.org/ Doc: https://ffmpeg.org/ffmpeg.html

```
$ sudo apt install ffmpeg
$ ffmpeg -i input.mp4 output.avi
```

Use the fps command to change the frames per second

```
$ ffmpeg -i monkey2.mp4 -vf fps=1/3 thumb%04d.jpg -hide_banner
```

Introduction to YOLO v3

Running YOLO in Google Colab IX

Preparing the dataset (images)

We will use **labelImg** for this task: **https://github.com/tzutalin/labelImg**

```
Carloss-MacBook-Air:~ axelvega$ git clone https://github.com/tzutalin/labelImg
Cloning into 'labelImg'...
remote: Enumerating objects: 1864, done.
remote: Counting objects: 100% (104/104), done.
remote: Compressing objects: 100% (74/74), done.
remote: Total 1864 (delta 47), reused 66 (delta 25), pack-reused 1760
Receiving objects: 100% (1864/1864), 232.82 MiB | 3.46 MiB/s, done.
Resolving deltas: 100% (1094/1094), done.
```

Also you have to install **Qt5** depending on your system
Ubuntu Linux:

```
sudo apt-get install pyqt5-dev-tools
sudo pip3 install -r requirements/requirements-linux-python3.txt
make qt5py3
python3 labelImg.py
```

Introduction to YOLO v3

Running YOLO in Google Colab IX

MacOs:

```
pip3 install pyqt5 lxml # Install qt and lxml by pip

make qt5py3
python3 labelImg.py
```

Windows:

Open cmd and go to the labelImg directory

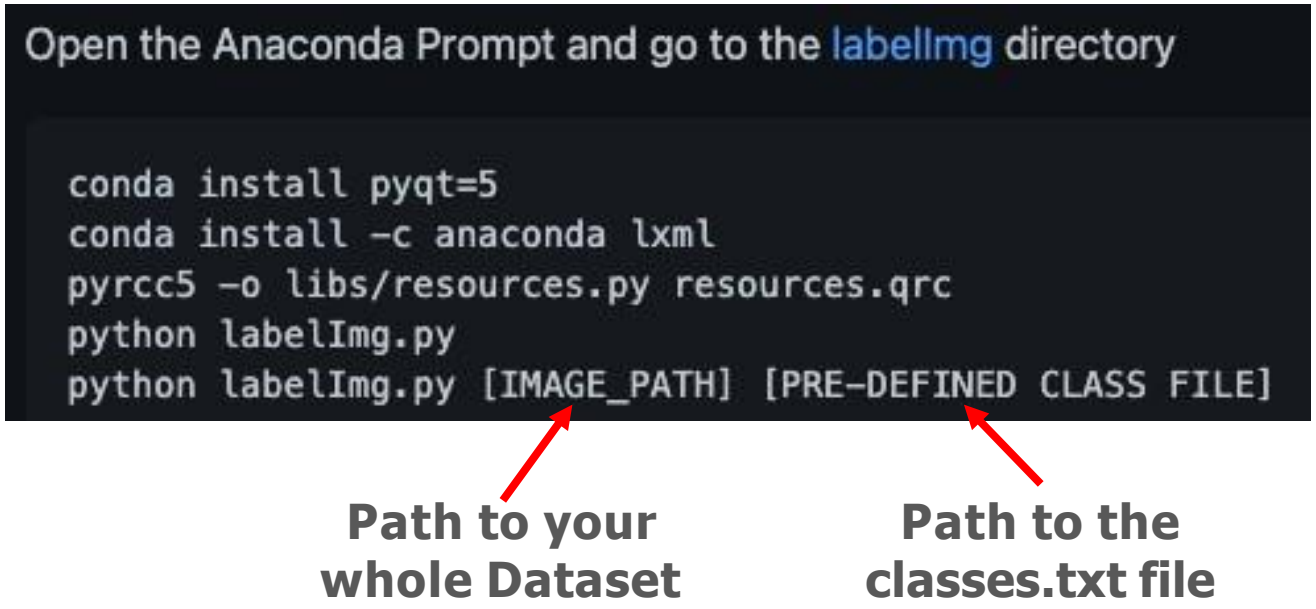
```
pyrcc4 -o libs/resources.py resources.qrc
For pyqt5, pyrcc5 -o libs/resources.py resources.qrc

python labelImg.py
python labelImg.py [IMAGE_PATH] [PRE-DEFINED CLASS FILE]
```

Introduction to YOLO v3

Running YOLO in Google Colab X

Windows +Anaconda



If you present some issues with the installation of labelImg in Anaconda Prompt, you should revise the following solution:

https://medium.com/@sanghuynh_73086/how-to-install-labelimg-in-windows-with-anaconda-c659b27f0f

Introduction to YOLO v3

Running YOLO in Google Colab X

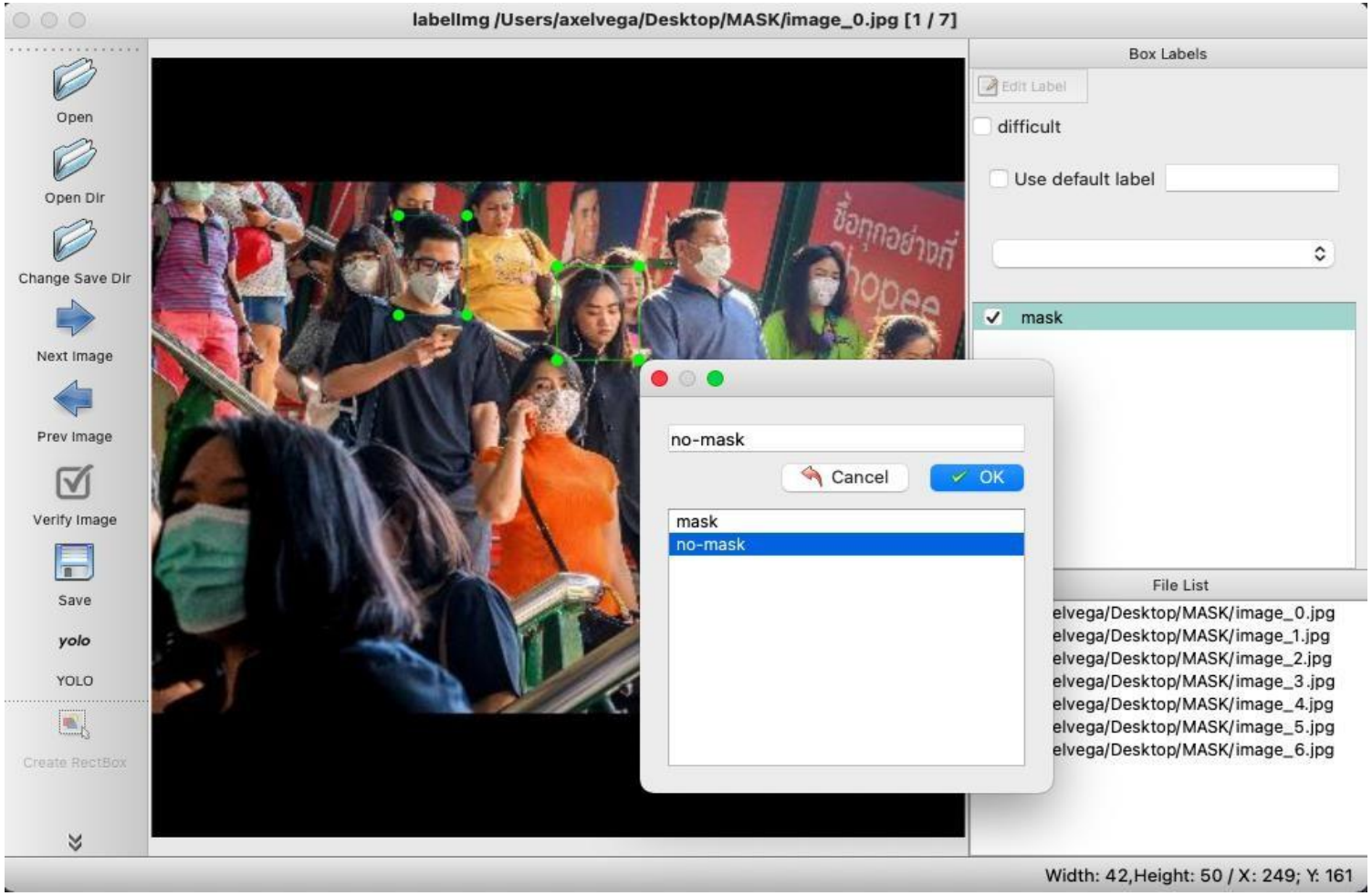
mask/no-mask example:

```
Carloss-MacBook-Air:labelImg axelvega$ python3 labelImg.py /Users/axelvega/Deskt
op/MASK/ /Users/axelvega/Desktop/classes.txt
```

Introduction to YOLO v3

Running YOLO in Google Colab XI

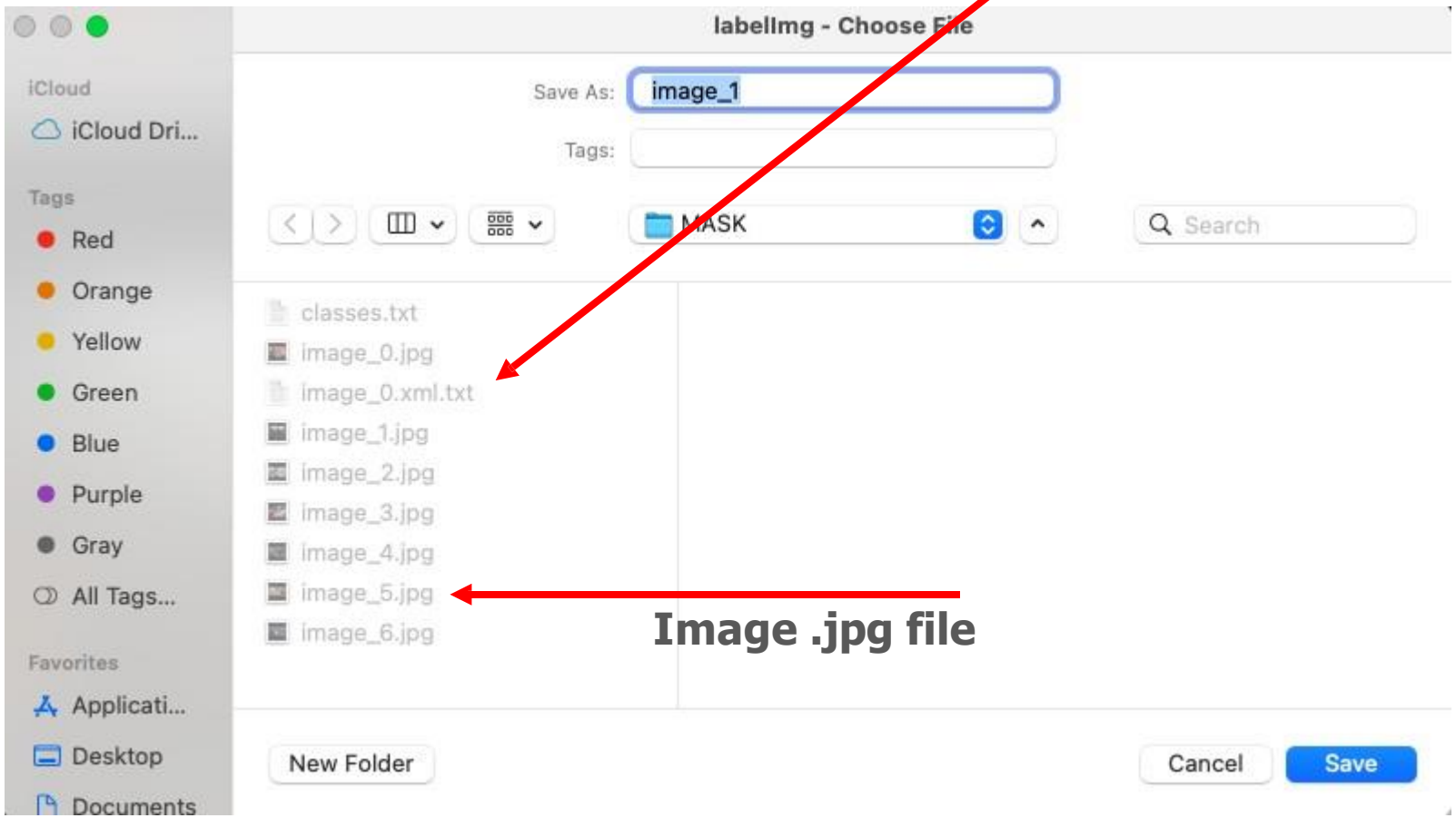
Preparing the dataset (images)



Introduction to YOLO v3

Running YOLO in Google Colab XII

Preparing the dataset (images)



Introduction to YOLO v3

Running YOLO in Google Colab XIII

Preparing the dataset (images)

Introduction to YOLO v3

Running YOLO in Google Colab XIII

Preparing the dataset (images): creating train.txt and test.txt files

Once we have created our folder **obj** with the images and the annotations, the next step is to create two more files: **train.txt** and **test.txt**.

- For avoiding complications with the following implementation it is important to keep the names of the images folder like **obj** and the names of .txt flies like **train.txt** and **test.txt**.
- In our case we split up the dataset into 90% for training and 10% for testing.

CAUTION! The **train.txt** and **test.txt** files should contain the paths to each image, thus it is not necessary to make a folder for training and a folder for test.

data/obj/NAME_OF_IMAGE.jpg

Introduction to YOLO v3

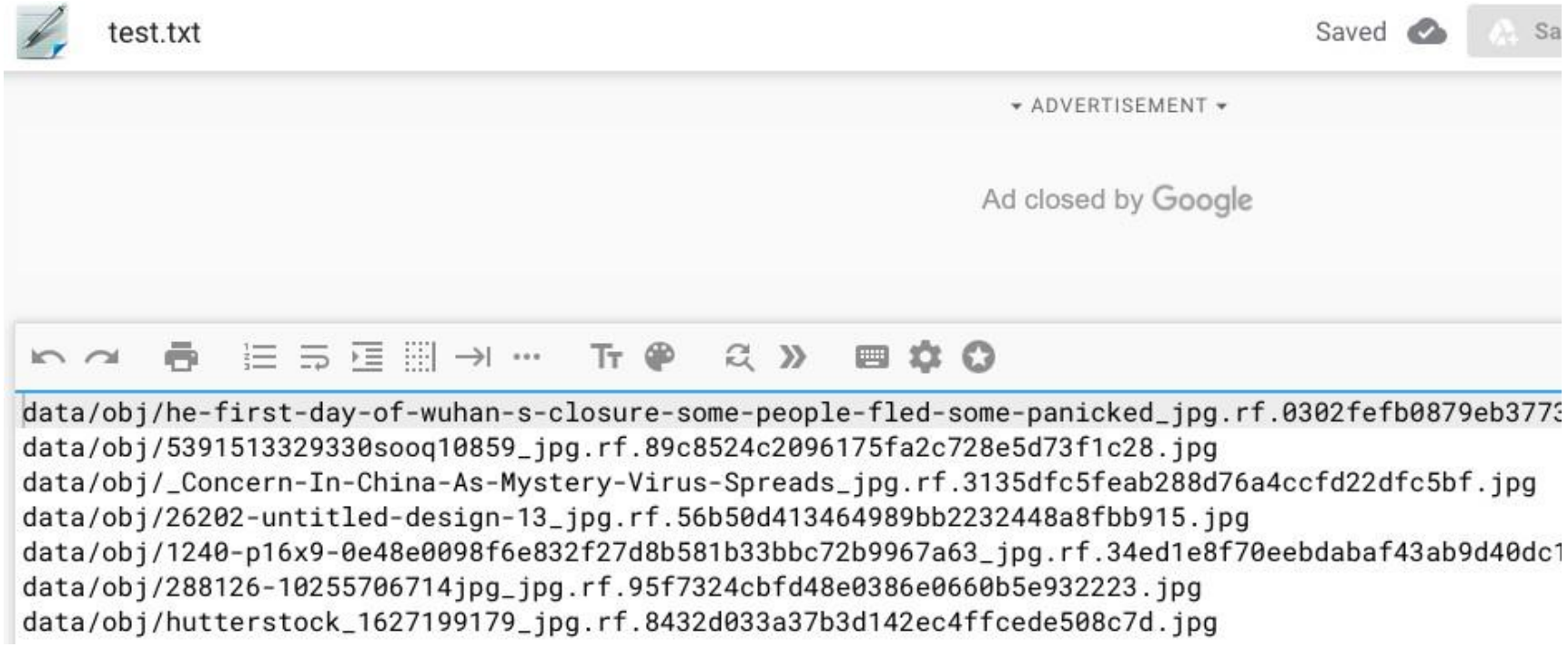
Running YOLO in Google Colab XIII

Preparing the dataset (images): creating train.txt and test.txt files

Introduction to YOLO v3

Running YOLO in Google Colab XIII

Preparing the dataset (images): creating train.txt and test.txt files



Training YOLO v3 over a custom dataset using Google Colab

Mounting Google Drive

Connect your Google Drive account to your Google Colab script

```
# Mount Google Drive

from google.colab import drive
drive.mount('/content/drive')

# Change to desired directory
%cd /content/drive/MyDrive/Tutorial_MICAI/YOLOv3
%ls
```

Training YOLO v3 over a custom dataset using Google Colab

Preparing Darknet backbone

Clone the Darknet repository from GitHub

```
# clone darknet repo
!git clone https://github.com/AlexeyAB/darknet

Cloning into 'darknet'...
remote: Enumerating objects: 15316, done.
remote: Total 15316 (delta 0), reused 0 (delta 0), pack-reused 15316
Receiving objects: 100% (15316/15316), 13.72 MiB | 14.30 MiB/s, done.
Resolving deltas: 100% (10406/10406), done.
```

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab XIV

Configuring YOLO v3

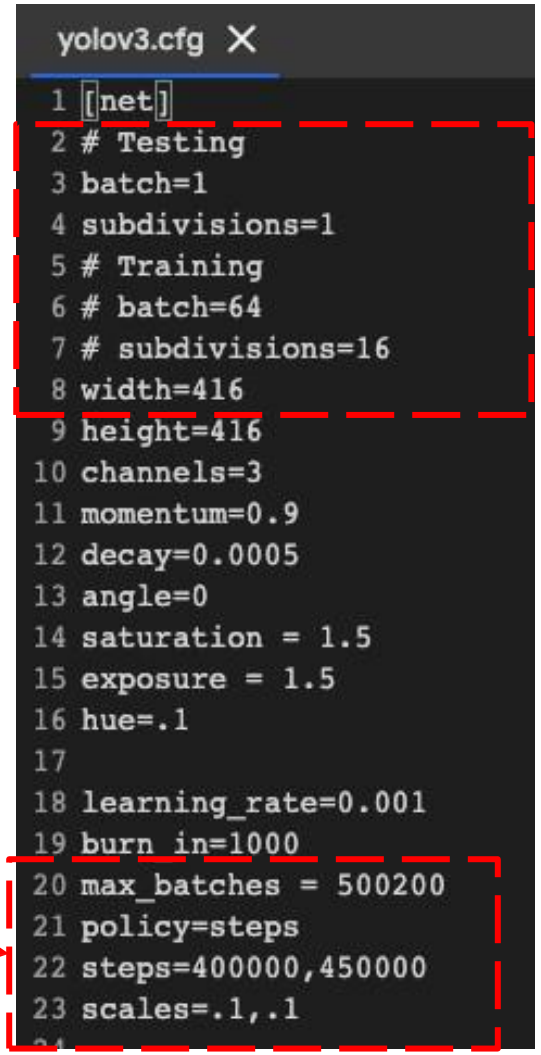
Our main configuration file is :

yolov3.cfg and

For custom dataset you have to edit it since it is configured for COCO Microsoft Dataset with 80 classes.

If you are training for different number of classes.

Values for COCO Dataset which should be modified according to yours



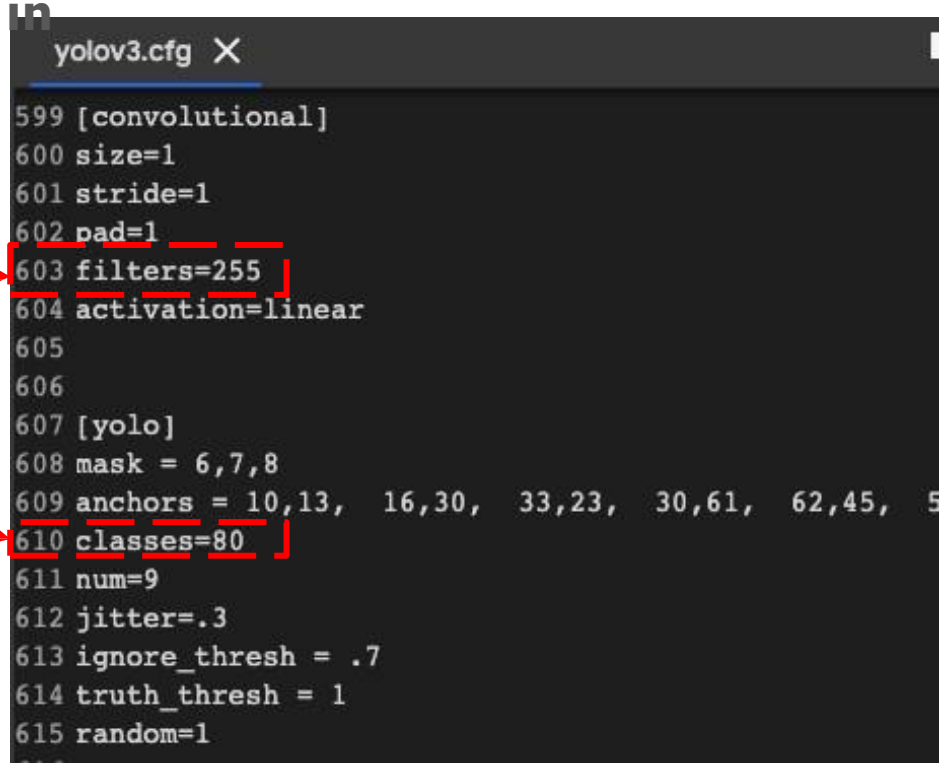
Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab XVI

Changing the number of classes and filters in convolutional layers

filters = (classes+5)*3 = 21

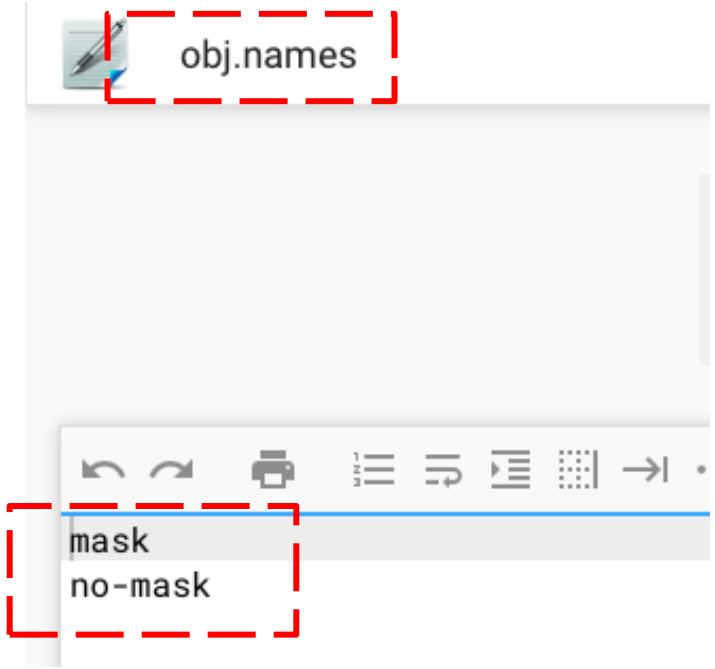
Number of classes = 2



Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab

Creating a file named obj.names containing the selected labels

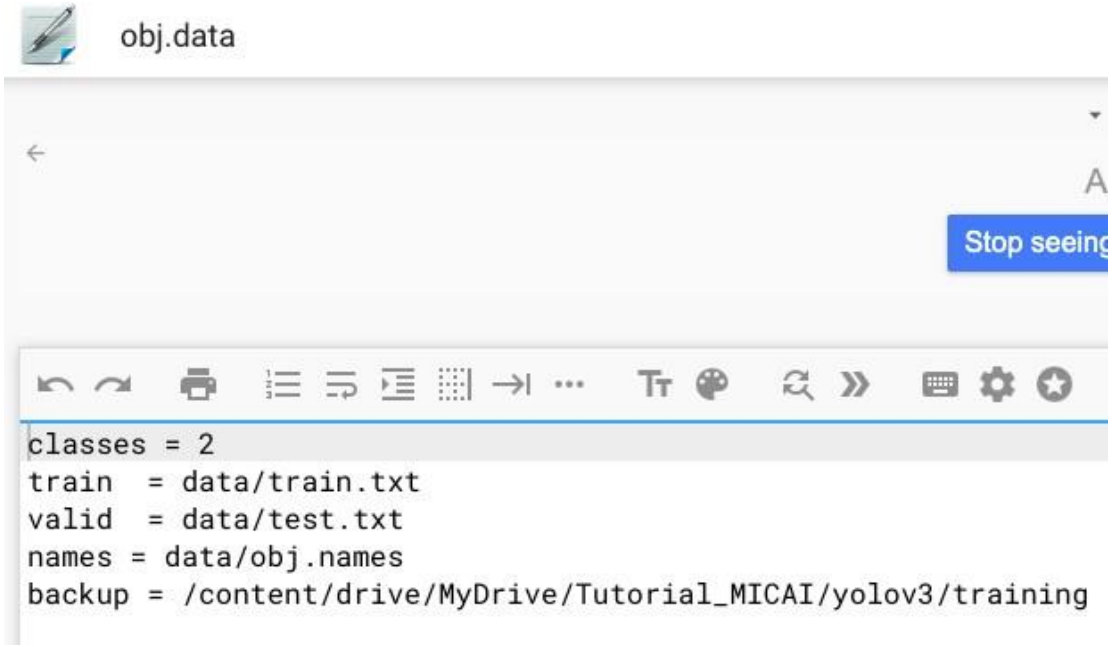


Make sure your labels are same as you did during making annotations file using labelImg

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab

Creating a file named obj.data containing the paths to the training, testing, configuration and backup directories



Make sure your paths are properly defined, else the model will not work

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab

Edit makefile in order to enable OPENCV and GPU which are going to be used during training and testing

```
# change makefile to have GPU and OPENCV enabled
# also set CUDNN, CUDNN_HALF and LIBSO to 1

%cd darknet/
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

Don’t forget to change to darknet directory.
Also enable CUDNN, CUDNN_HALF, and LIBSO by changing their values to 1

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab

Run make command to build Darknet

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab

Prepare data/ and cfg/ folders within Darknet. Remove unnecessary files. Copy required files

```
# Clean the data and cfg folders first except the labels folder in data which is required

%cd data/
!find -maxdepth 1 -type f -exec rm -rf {} \;
%cd ..

%rm -rf cfg/
%mkdir cfg
```

```
[ ] # Unzip the obj.zip dataset and its contents so that they are now in /darknet/data/ folder
    #!unzip "/content/drive/MyDrive/Tutorial_MICAI/yolov3/obj.zip" -d "/content/drive/MyDrive/Tutorial_MICAI/yolov3/"
    !cp -r /content/drive/MyDrive/Tutorial_MICAI/yolov3/obj -d data/
```

```
[ ] # Copy the yolov4-custom.cfg file so that it is now in /darknet/cfg/ folder

    !cp /content/drive/MyDrive/Tutorial_MICAI/yolov3/yolov3.cfg cfg

    # verify if your custom file is in cfg folder
    !ls cfg/
```

```
    yolov3.cfg
```

```
[ ] # Copy the obj.names and obj.data files from your drive so that they are now in /darknet/data/ folder

    !cp /content/drive/MyDrive/Tutorial_MICAI/YOLOv3/obj.names data
    !cp /content/drive/MyDrive/Tutorial_MICAI/YOLOv3/obj.data  data

    # verify if the above files are in data folder
    !ls data/
```

```
    labels  obj  obj.data  obj.names
```

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab

Prepare data/ and cfg/ folders within Darknet. Remove unnecessary files. Copy required files

```
# Copy train, valid and test file to data/ directory
%cp /content/drive/MyDrive/Tutorial_MICAI/YOLOv3/train.txt data/
%cp /content/drive/MyDrive/Tutorial_MICAI/YOLOv3/test.txt data/
%cp /content/drive/MyDrive/Tutorial_MICAI/YOLOv3/valid.txt data/
```

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab

Download pre-trained weights file from the following URL

http://pjreddie.com/media/files/darknet53.conv.74

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab

Time to start training our custom YOLOv3 object detector

Execute the following command:

./darknet detector train [data] [configuration] [weights] [arguments]

```
# Time to train our custom YOLOv3 detector

!./darknet detector train data/obj.data cfg/yolov3.cfg darknet53.conv.74 -dont_show -mjpeg_port 8090 -map
```

```
 (next mAP calculation at 1000 iterations)
 34: 1246.143066, 1189.019897 avg loss, 0.000000 rate, 15.523874 seconds, 2176 images, 31.139845 hours left
 MJPEG-stream sent.
Loaded: 0.000081 seconds
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.206141), count: 1, class_loss = 172.732620, iou_loss = 1.406052, total_loss = 174.138672
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.390412), count: 27, class_loss = 773.013794, iou_loss = 17.386536, total_loss = 790.400330
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.304934), count: 8, class_loss = 2767.779297, iou_loss = 8.843750, total_loss = 2776.623047
 total_bbox = 12890, rewritten_bbox = 0.015516 %
```

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab

Evaluate our custom YOLOv3 object detector

Execute the following command:

./darknet detector map [data] [configuration] [weights] [arguments]

```
## Check the mAP for the saved weights of your interest

!./darknet detector map data/obj.data cfg/yolov3.cfg /content/drive/MyDrive/Tutorial_MICAI/training/yolov3_last.weights -iou_thresh 0.50
```

Keep in mind that you can define the Intersection over Union threshold

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab

Test our custom YOLOv3 object detector over an image

Change cfg file to test mode

```
# Change your cfg file to test mode
%cd cfg
!sed -i 's/batch=64/batch=1/' yolov3.cfg
!sed -i 's/subdivisions=16/subdivisions=1/' yolov3.cfg
%cd ..
```

Execute the following command

./darknet detector test [data] [configuration] [weights] [arguments]

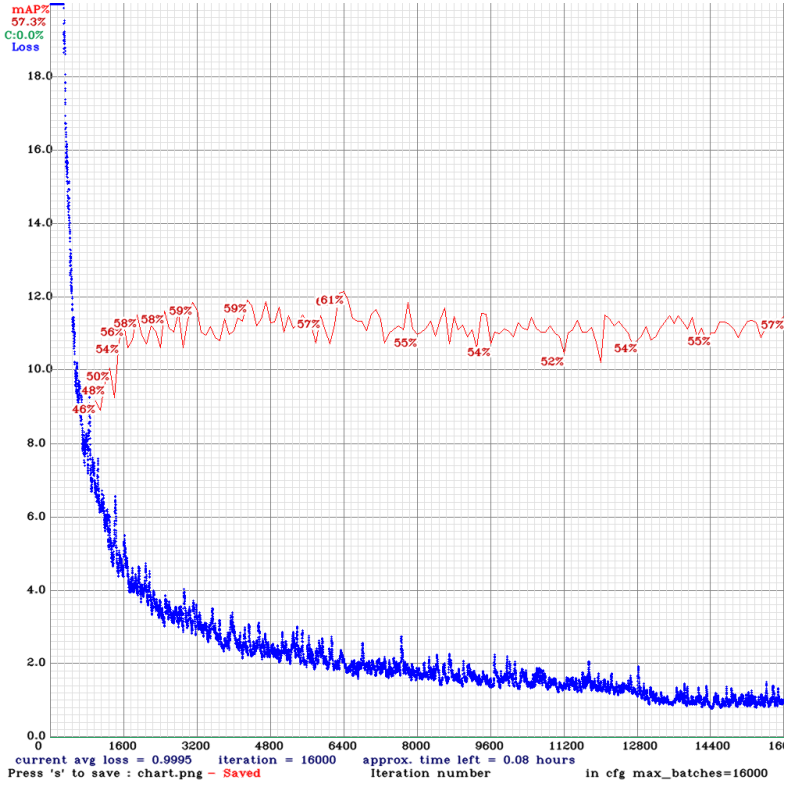
```
# Test custom YOLOv3 object detector over an image
%cd /content/drive/MyDrive/Tutorial_MICAI/YOLOv3/darknet/
!./darknet detector test data/obj.data cfg/yolov3.cfg /content/drive/MyDrive/Tutorial_MICAI/YOLOv3/training/yolov3_last.weights /content/drive/MyDrive/Tutorial_MICAI/YOLOv3/obj/image_1.jpg -thresh 0.3
imShow('predictions.jpg')
```

Keep in mind that you can define the Intersection over Union threshold

Training YOLO v3 over a custom dataset using Google Colab

Running YOLO in Google Colab XVI

YOLOv3 model performance



Test over an image