Section3

Semantic Segmentation, object detection, instance segmentation

Computer vision tasks

- Computer vision tasks are the main engines to drive the development of deep learning. Start with the image classification to more complete task like semantic segmentation and tracking task. We will see what we have solved in the past and what are the challenges that need to be solved.
- Image classification → Object detection → Instance segmentation → Semantic segmentation

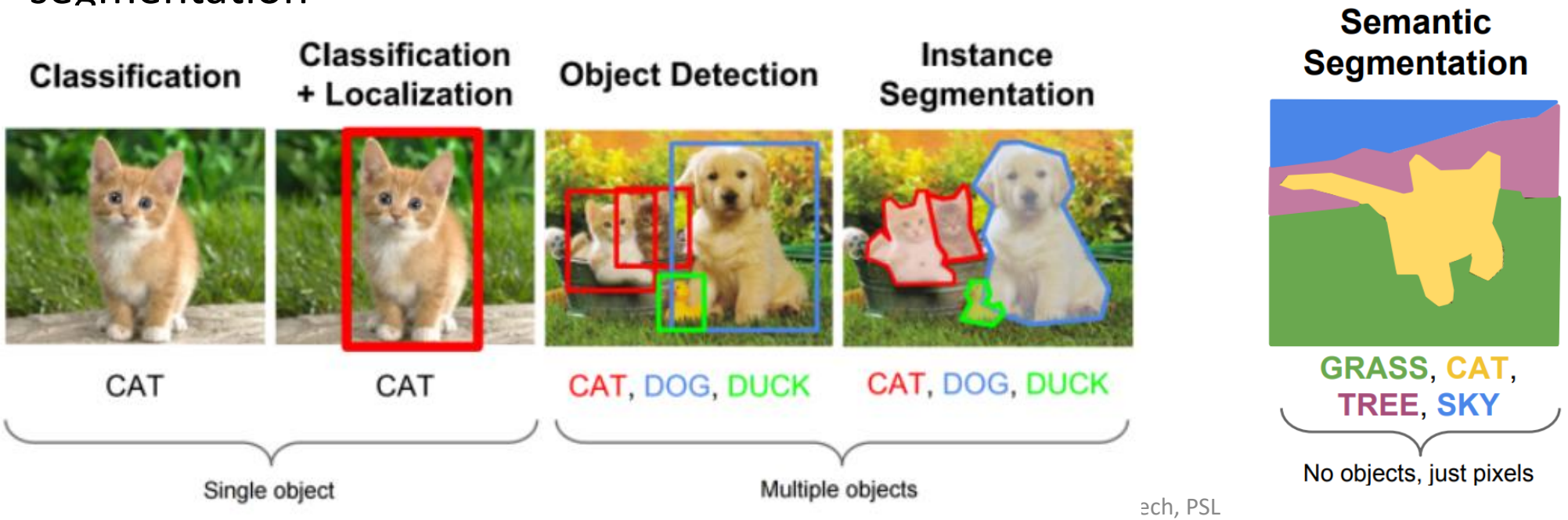
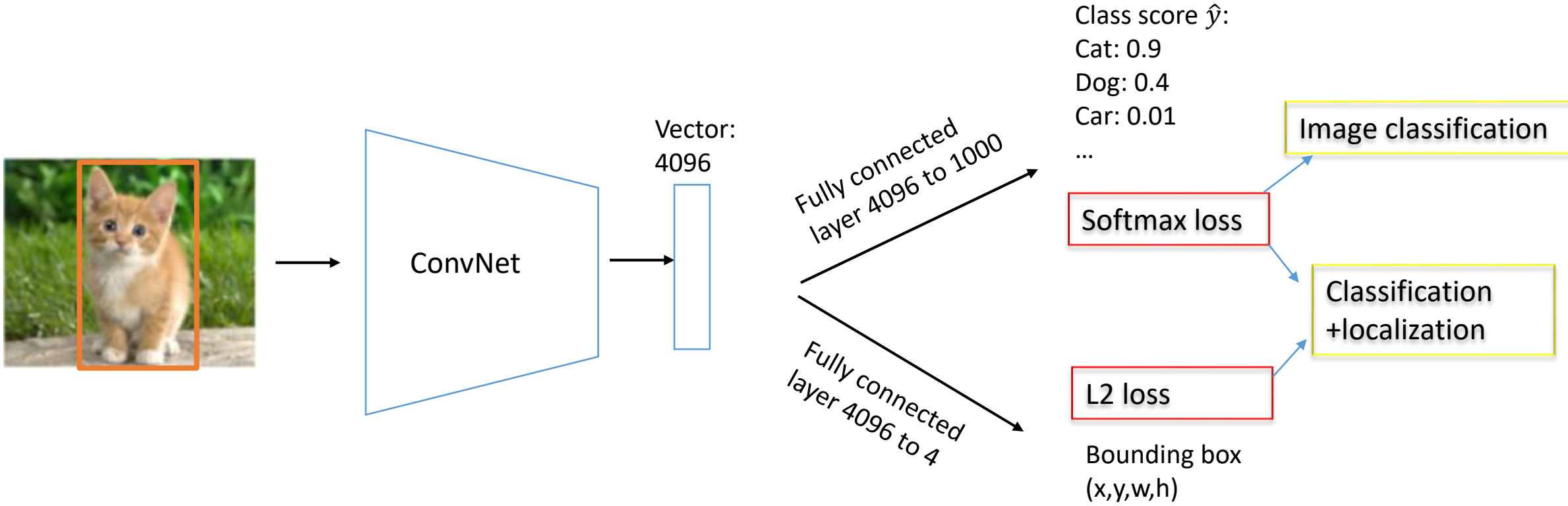


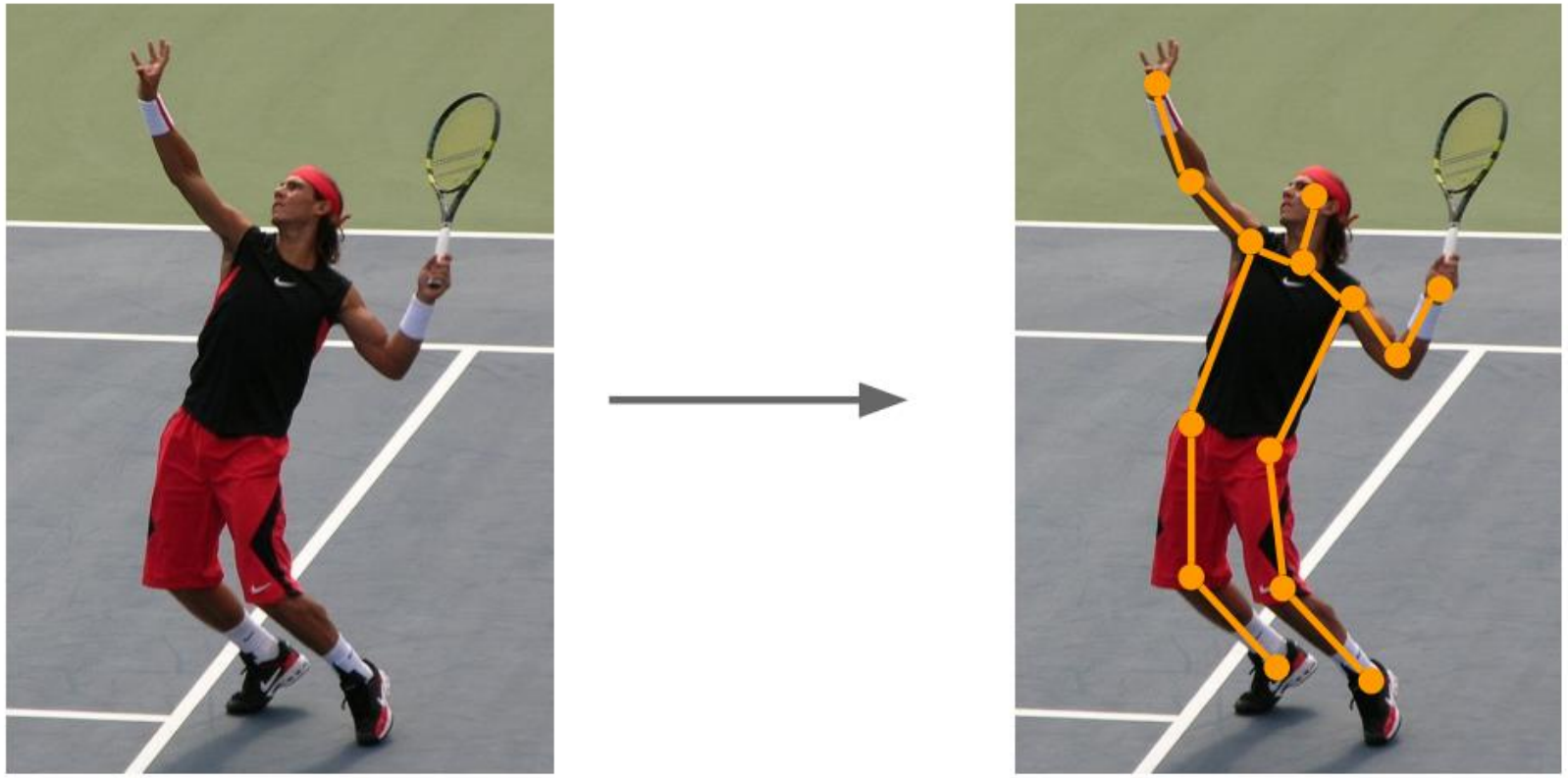
Image tasks



Here we treat localization as a regression problem and extend this task to be a multi-tasks

Aside the bounding box

- Human pose prediction is by adding more regression tasks at the end

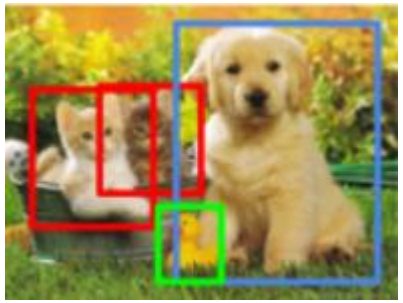


Represent pose as a set of 14 joint positions:

Left / right foot	(x1,y1)
Left / right knee	(x3,y3)
Left / right hip	
Left / right shoulder	
Left / right elbow	
Left / right hand	
Neck	
Head top	(x14,y14)

Johnson and Everingham, "Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation", BMVC 2010

Object detection



ConvNet

Vector:
4096

Fully connected
layer 4096 to 1000

Fully connected
layer 4096 to 4

Class score $\hat{y}$:
Cat: 0.9
Dog: 0.4
Car: 0.01
…

Image classification

Softmax loss

Classification
+localization

L2 loss

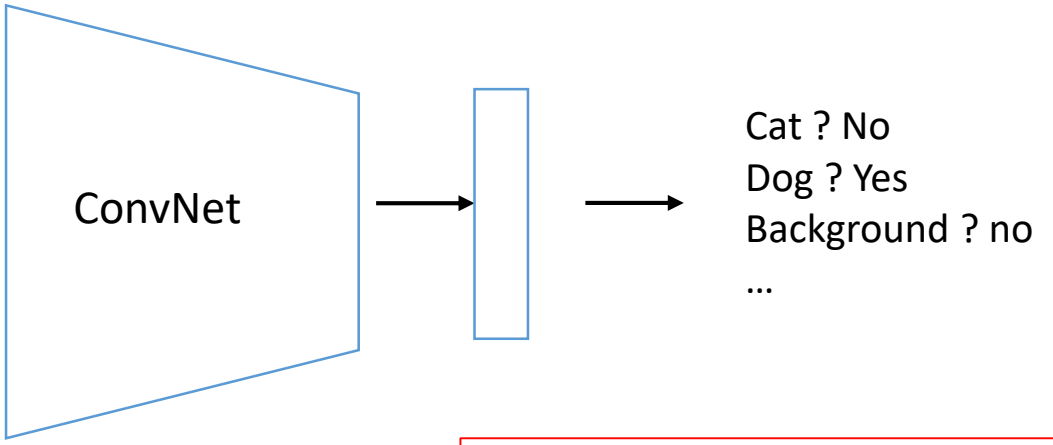
Bounding box
(x,y,w,h)

Object detection task is more close to complete computer vision task. However, can we treat this problem like the method before?

No! Unlike human pose estimation, we don't know how many objects in one image. We can't fixed the size of output in this way

First approach: sliding window

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

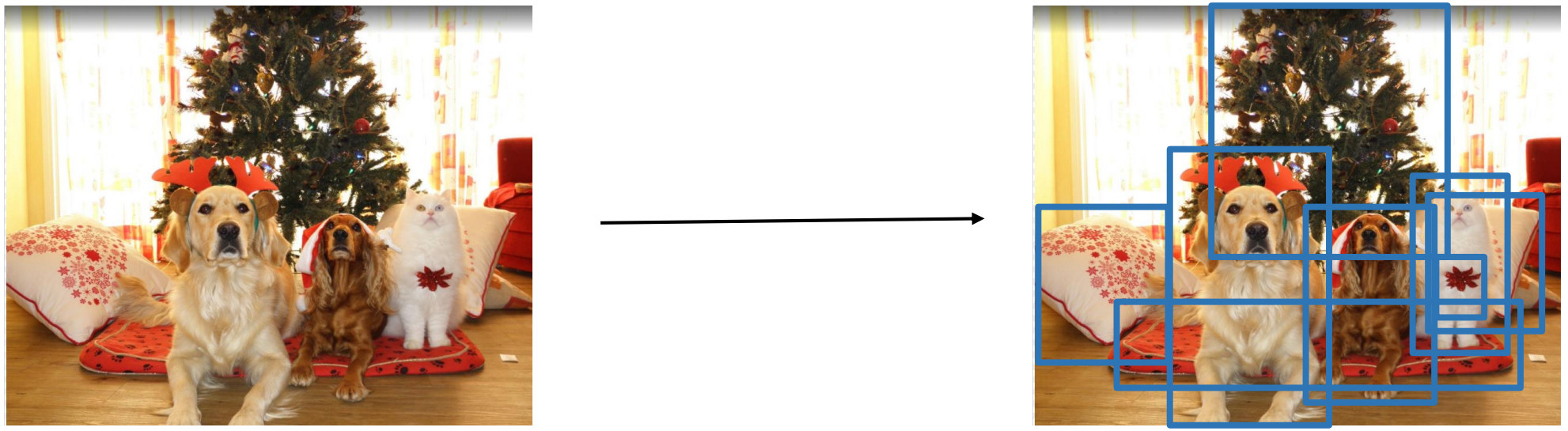


Cat ? No
Dog ? Yes
Background ? no
…

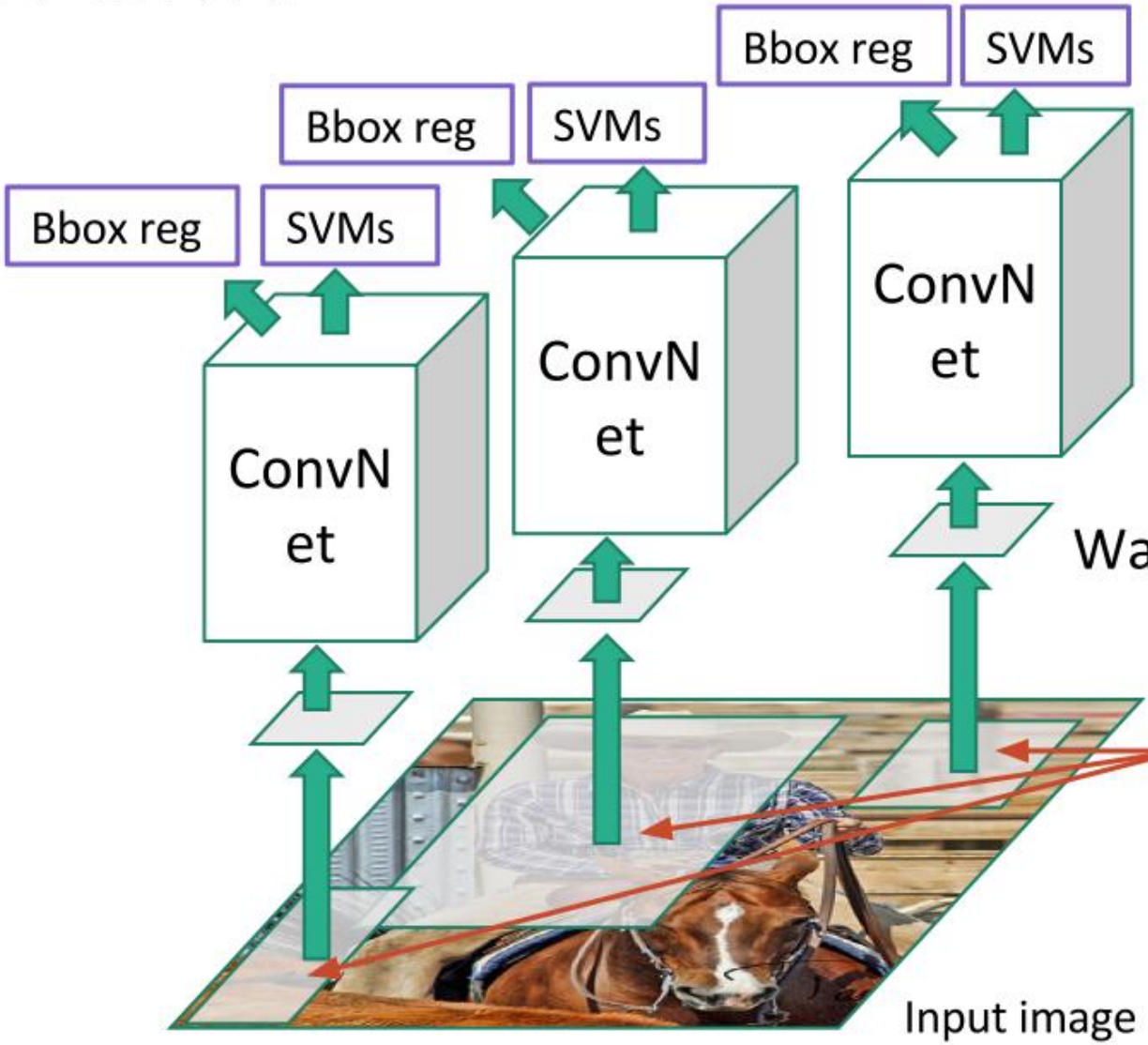
Problem: Need to apply CNN to huge number of locations and scales, very computationally expensive!

First approach: region proposals

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 1000 region proposals in a few seconds on CPU



R-CNN



Linear Regression for bounding box offsets

Classify regions with SVMs

Forward each region through ConvNet

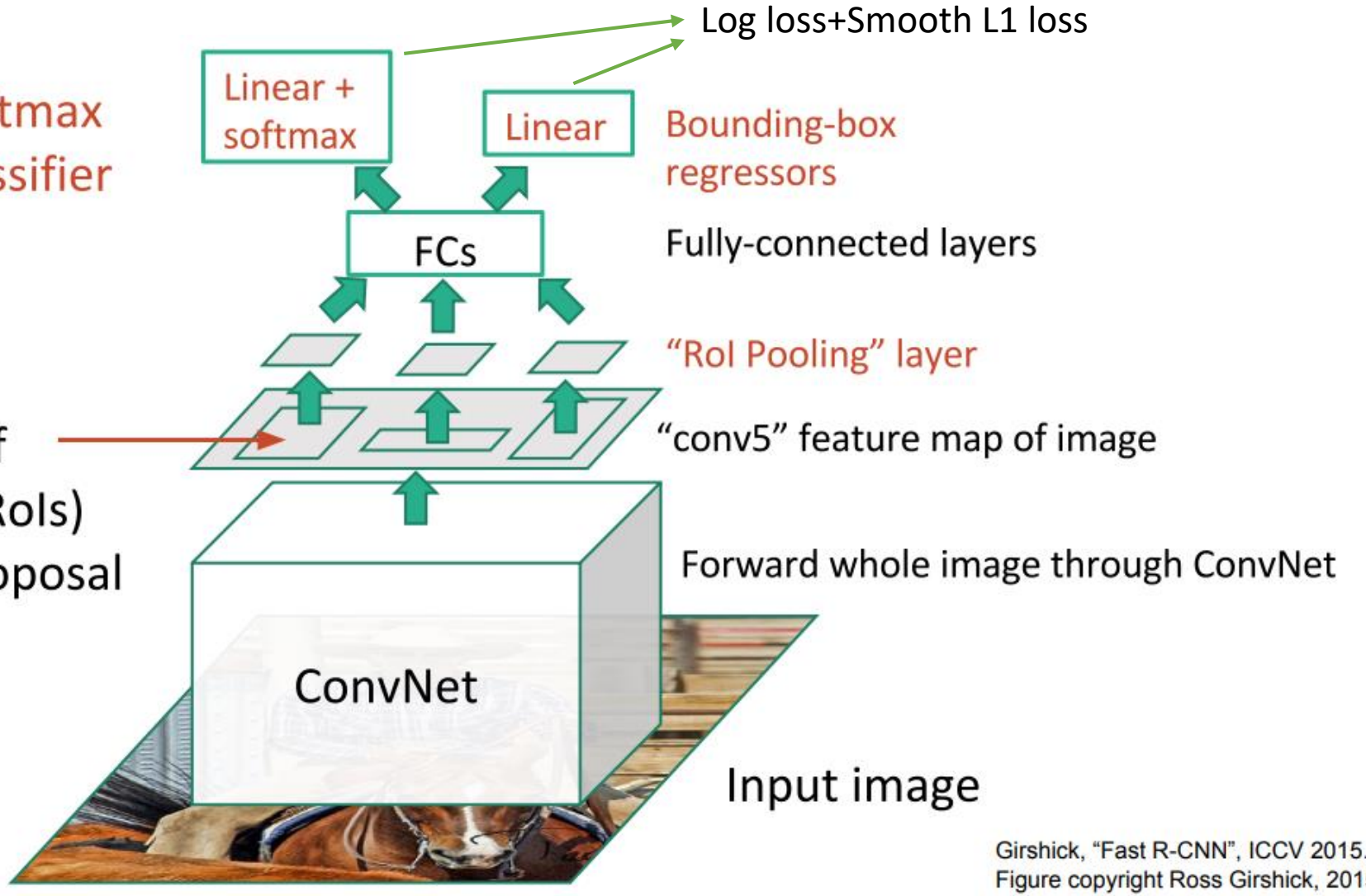
Warped image regions

Regions of Interest (RoI) from a proposal method (~2k)

- Very slow + approximate bounding-boxes
- Inference time per image is around 47s with VGG16

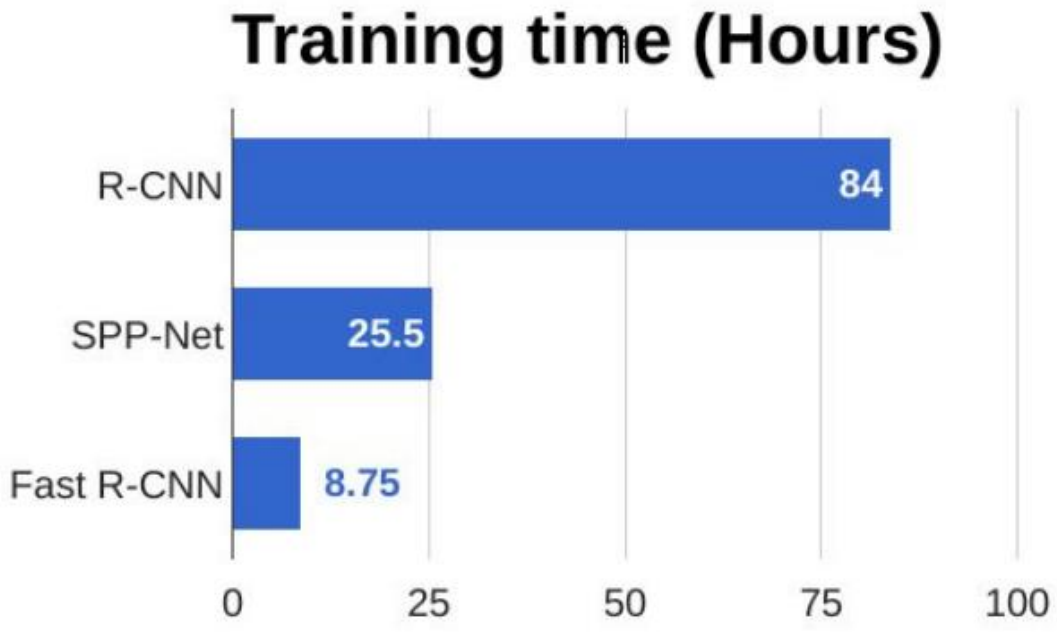
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

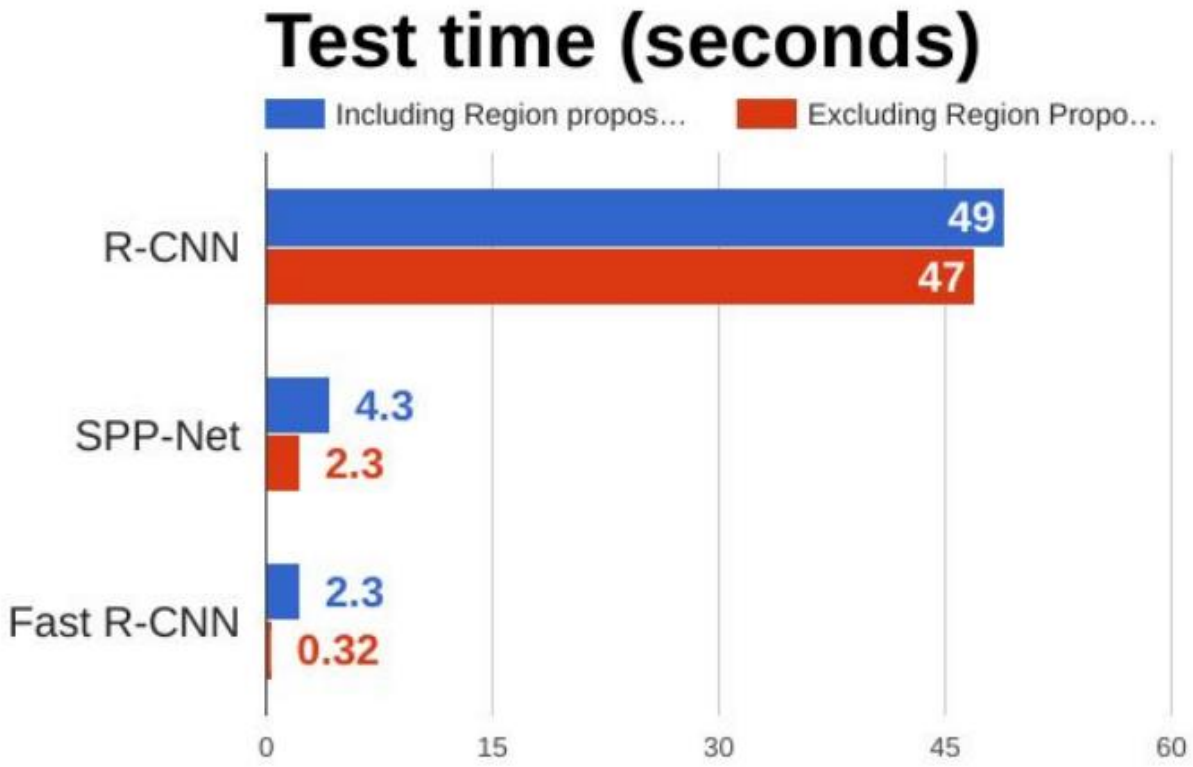
Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN vs SPP vs Fast R-CNN





Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014
Girshick, "Fast R-CNN", ICCV 2015

Faster R-CNN:

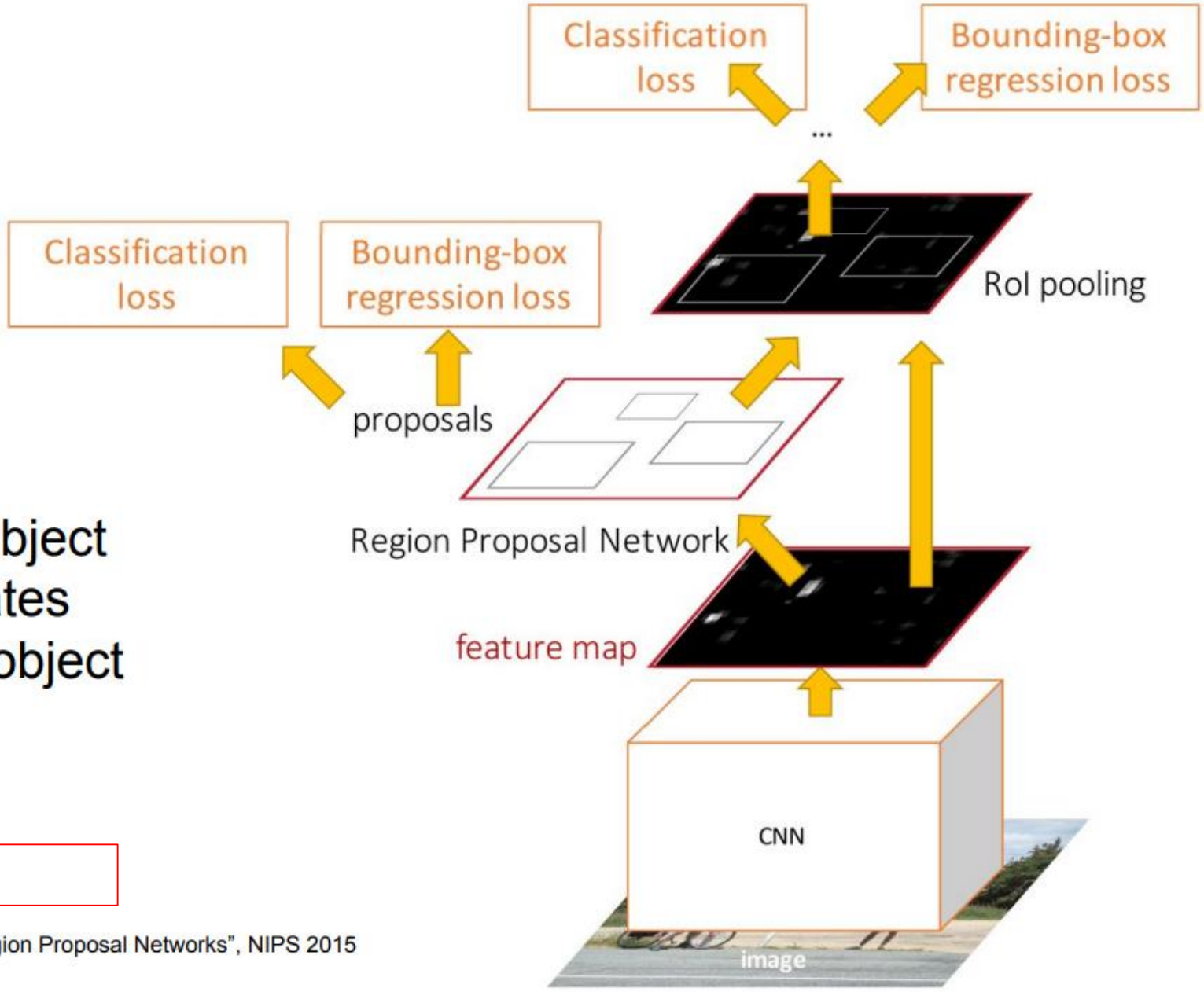
Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

Jointly train with 4 losses:

1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates

With this solution, test time drop to 0.2 s

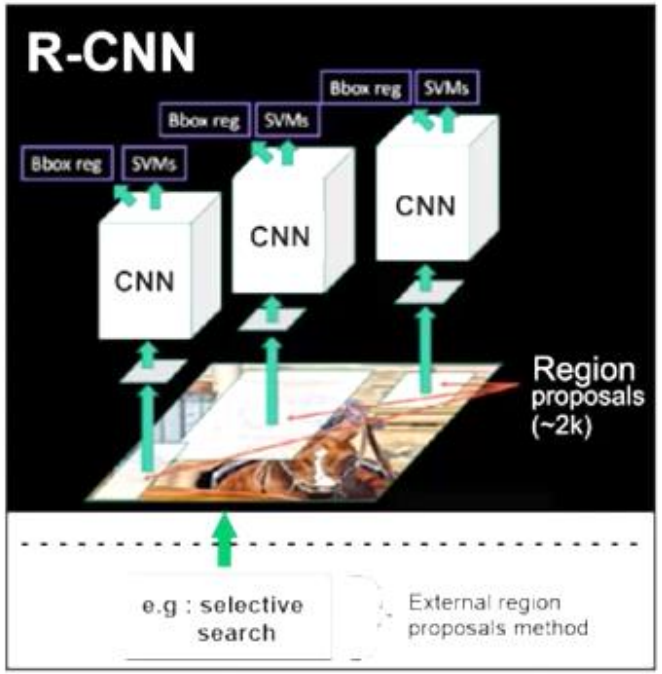


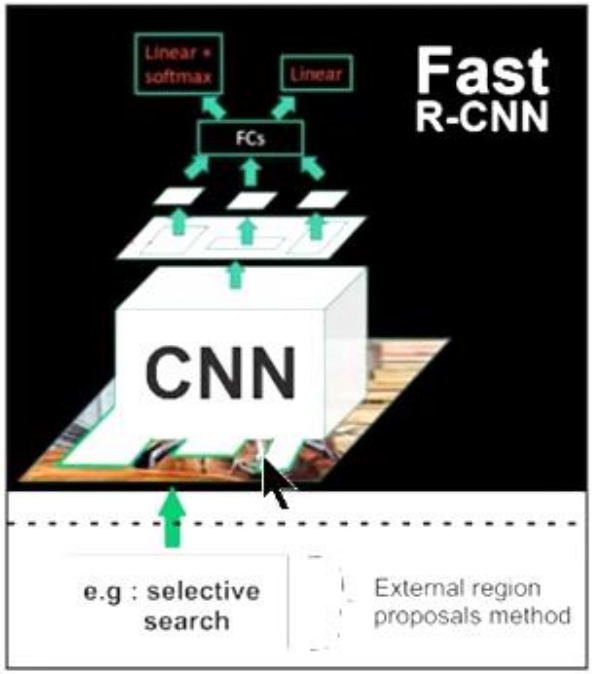
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

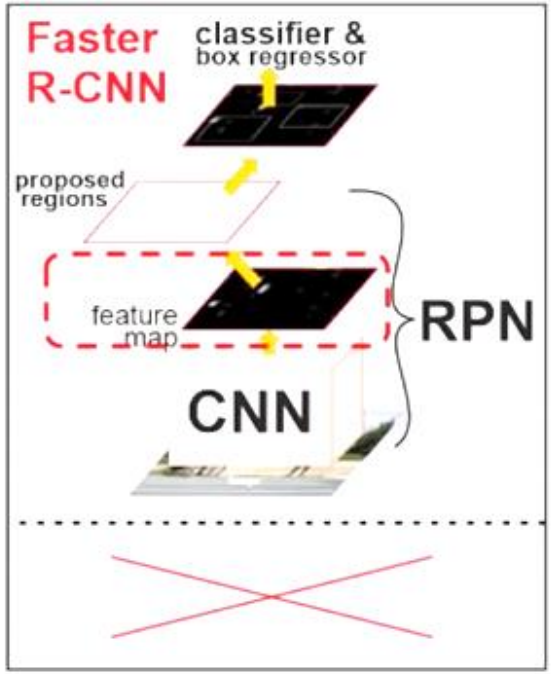
R-CNN family

- History
  - R-CNN: Selective search → Cropped Image → CNN
  - Fast R-CNN: Selective search → Crop feature map of CNN
  - Faster R-CNN: CNN → Region-Proposal Network → Crop feature map of CNN
- Best performances, but longest run-time
- End-to-end, multi-task loss

[https://github.com/endernewton/tf-faster-rcnn]







	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x
mAP (VOC 2007)	66.0%	66.9%	66.9%

Mean Average Precision

Metric for object detections

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

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

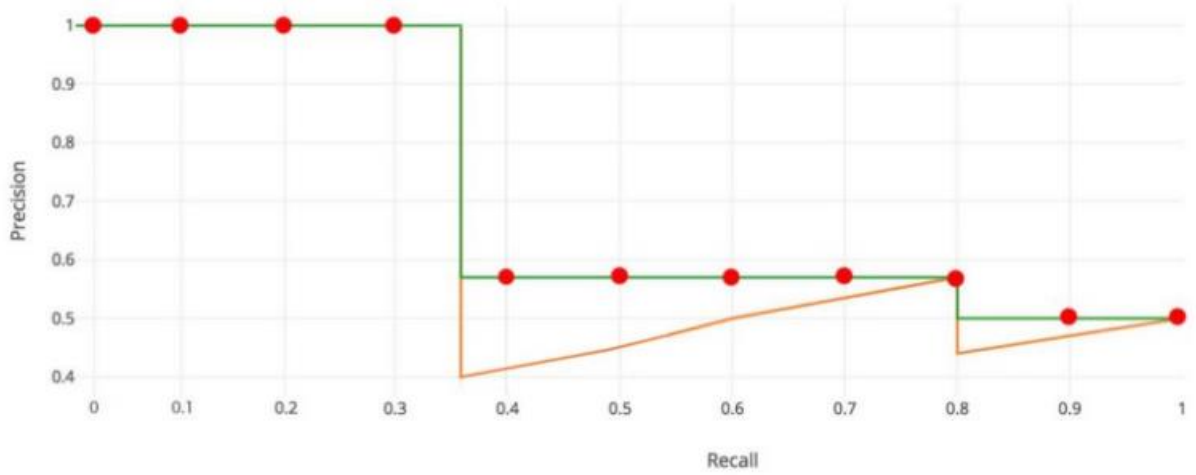
TP = True positive

TN = True negative

FP = False positive

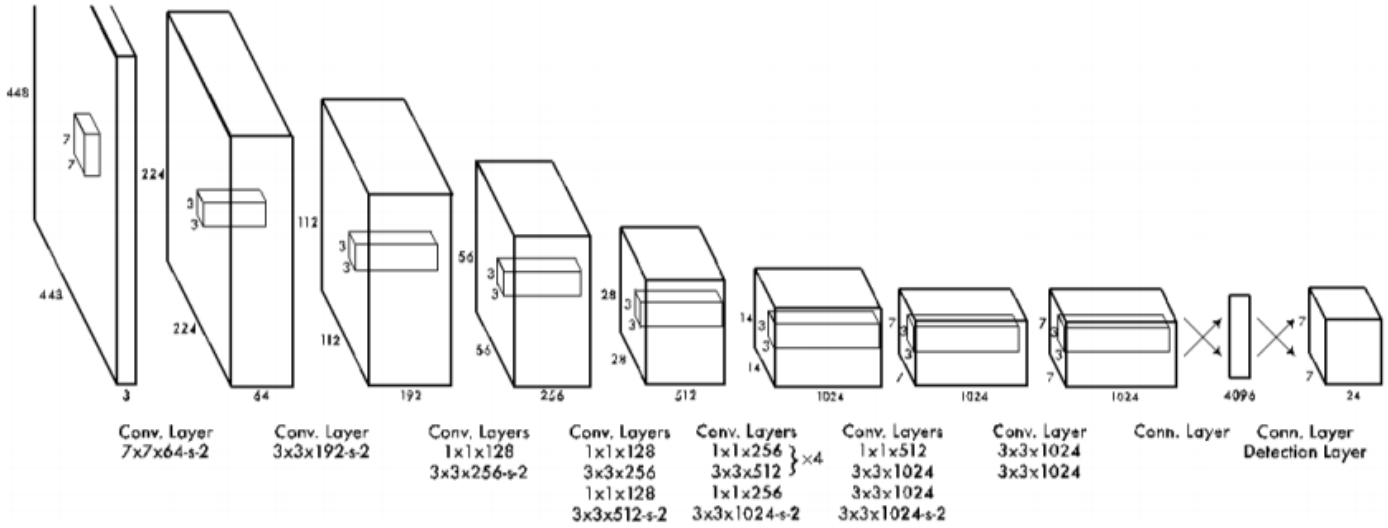
FN = False negative

Precision: how accurate if your prediction
Recall: how good you find all the positives
IoU: intersection over union
AP: average precision: summation of the area under precision-recall curve



Detection without proposals: YOLO

You Only Look Once



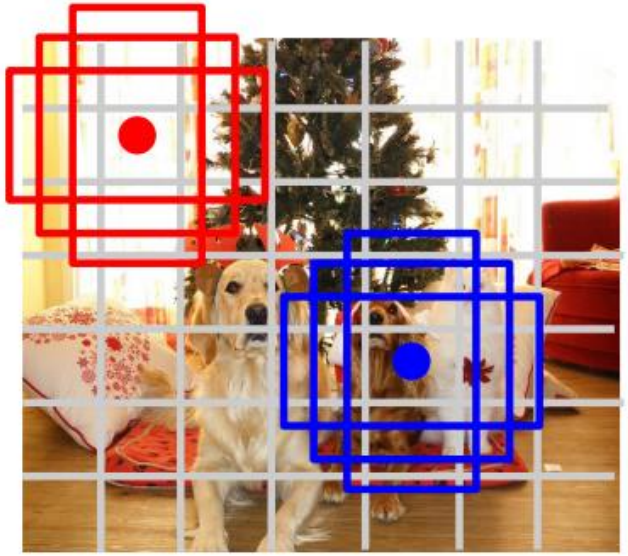
- **Modified GoogleNet/Inception**
- **Super fast (21~155 fps)**
- **Finds objects in image grids *in parallel***
- **Only slightly worse performance than Faster R-CNN**

YOLO limitations and v2 improvements

- Non-maximal suppression (NMS):
  - Since YOLO uses 7x7 grid then if an object occupies more than one grid this object may be detected in more than one grid
  - For each class (cars, pedestrians, cats,….) do:
    1. Discard all boxes with confidence **C**<**C** -threshold (for example **C**<0.5)
    2. Sort the predictions starting from the highest confidence **C.**
    3. Choose the box with the highest **C** and output it as a prediction .
    4. Discard any box with IOU>IOU-threshold with the box in the previous step .
    5. Start again from step (3) until all remaining predictions are checked.
- Limitations
  - It can only detect maximum 49 objects
  - Groups of small objects
  - Unusual aspect ratios
  - High error of localization
- YOLOv2 (2016):
  - add Batch Normalization
  - Increase input image from 224x224 to 448x448
  - Add anchor boxes: multi-object prediction per grid cell
  - Backbone: Darknet19

Detection without proposals: YOLO

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

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)

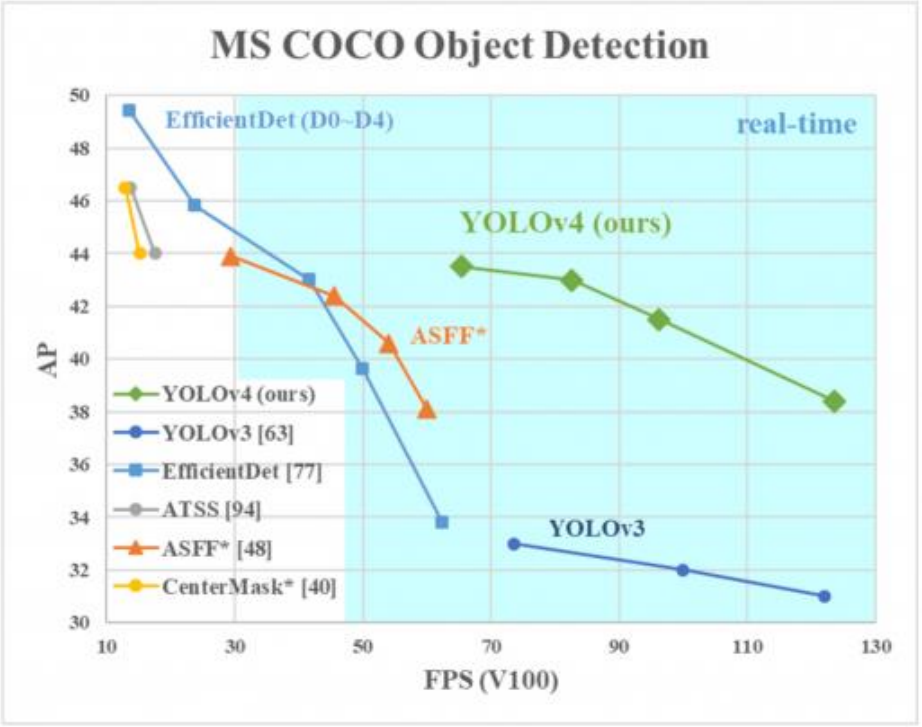
Output:
7 x 7 x (5 * B + C)

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

Slide from:[http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf]

YOLO families

- YOLO9000 (2017):
  - It is a real-time framework for detecting more than 9000 object categories by jointly optimizing detection and classification. During training, they mix images from both detection and classification datasets.
- YOLOv3 (2018):
  - Backbone: Darknet53
  - better performance for small objects
  - worse performance on medium and larger size objects.
  - YOLOv3 predicts boxes at 3 different scales
- YOLOv4 (Alexey Bochkovskiy et al.,2020):
  - bag of freebies: CutMix and Mosaic data augmentation, DropBlock regularization, Class label smoothing
  - bag of specials: Mish activation, Cross-stage partial connections (CSP), Multi- input weighted residual connections (MiWRC)
- YOLOv5 (no published paper)



The speed and accuracy of YOLO v4 (source: YOLO v4 paper)

YOLO versions

YOLO (darknet) - https://pjreddie.com/darknet/yolov1/ (C++)

YOLO v2 (darknet) - https://pjreddie.com/darknet/yolov2/ (C++)

- Better and faster - 91 fps for 288 x 288

YOLO v3 (darknet) - https://pjreddie.com/darknet/yolo/ (C++)

https://github.com/ultralytics/yolov3 (Pytorch)

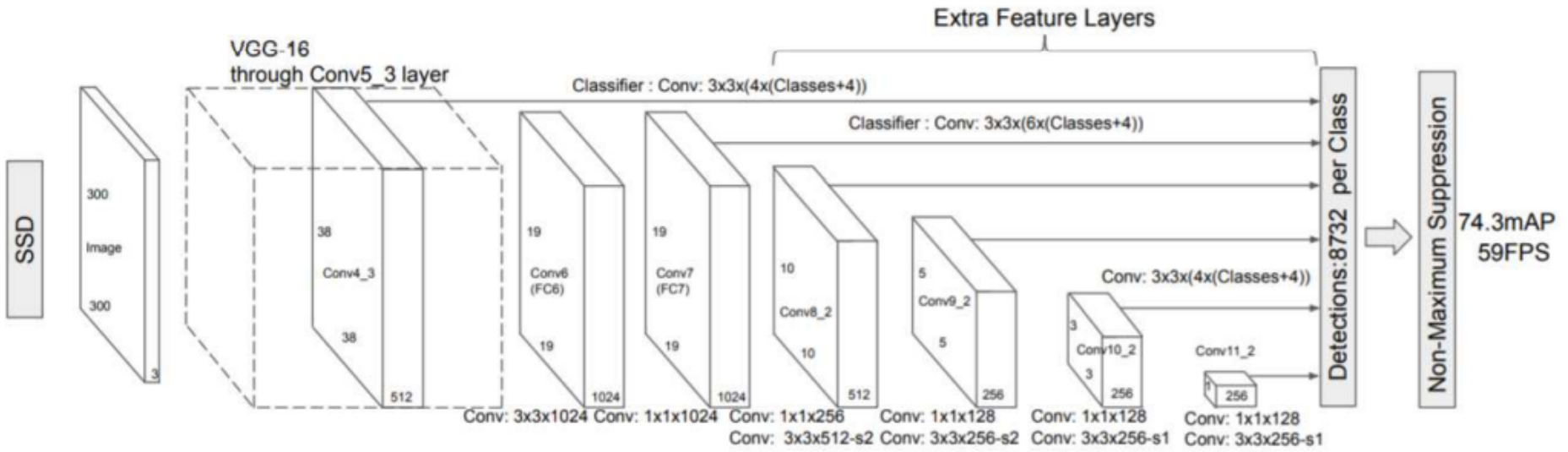
YOLO (caffe) - https://github.com/xingwangsfu/caffe-yolo

YOLO (tensorflow) - https://github.com/thtrieu/darkflow

YOLO v5 (pytorch) - https://github.com/ultralytics/yolov5

SSD (Single Shot Detector)

Architecture



Slower but more accurate than YOLO
Faster but less accurate than Faster_R-CNN

SSD available source code versions

SSD (caffe) - https://github.com/weiliu89/caffe/tree/ssd

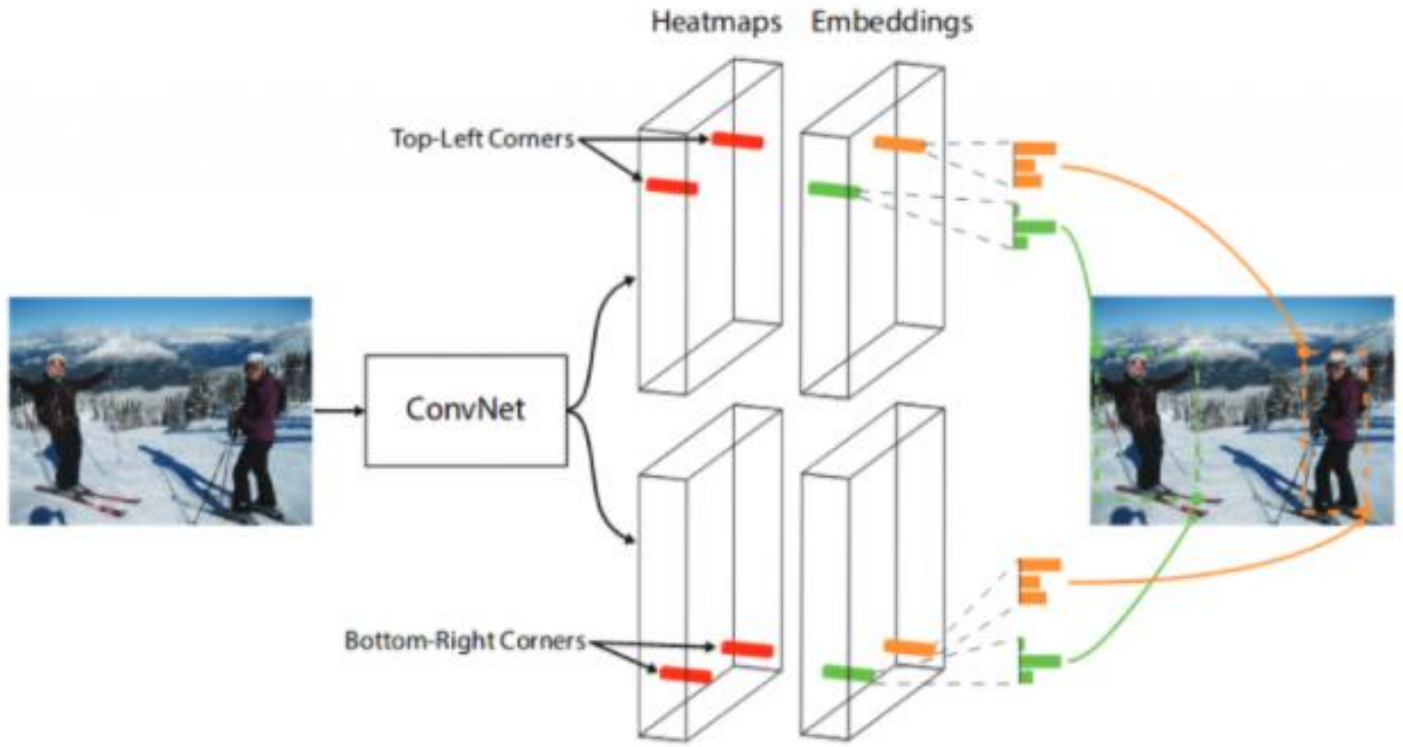
SSD (tensorflow) - https://github.com/balancap/SSD-Tensorflow

SSD (pytorch) - https://github.com/amdegroot/ssd.pytorch

CornerNet

Hei Law, Jia Deng "CornerNet: Detecting Objects as Paired Keypoints", *European Conference on Computer Vision (ECCV), 2018*

- Detecting Objects as Paired Key points: the top-left corner and the bottom-right corner
- No anchor boxes
- Proposed corner pooling – that helps the network better localize corners
- Backbone: Hourglass-104



$$L = L_{det} + \alpha L_{pull} + \beta L_{push} + \gamma L_{off}$$

L_{det} is the detection loss, which is responsible for proper corner detection and is a variant of focal loss;
L_{pull} is the grouping loss, used to pull corners of one object together;
L_{push}, is, on the opposite, used to separate corners of different objects;
L_{off} is the smooth L1 loss used for offset correction;
and α, β and γ are parameters, which are set to 0.1, 0.1 and 1 respectively.

Object detection variables

- Base Network
  - VGG16
  - ResNet-101
  - Inception V2
  - Inception V3
  - Inception
  - ResNet
  - MobileNet
- Object Detection architecture
  - Faster R-CNN
  - R-FCN
  - SSD
  - YOLO

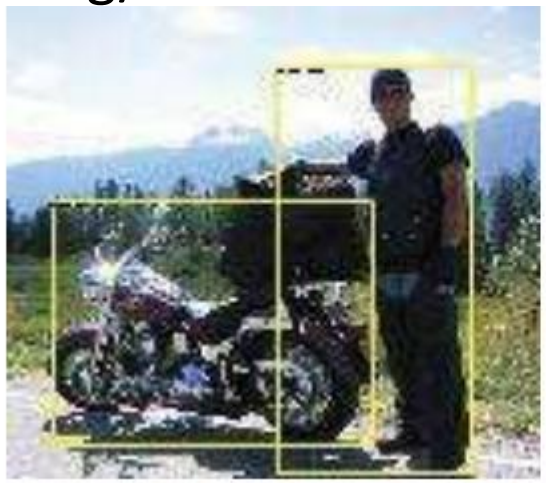
Recommended reading:
Huang et al, “Speed/accuracy trade-offs for modern convolutional object detectors”, CVPR 2017

R-FCN: Dai et al, “R-FCN: Object Detection via Region-based Fully Convolutional Networks”, NIPS 2016
Inception-V2: Ioffe and Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, ICML 2015
Inception V3: Szegedy et al, “Rethinking the Inception Architecture for Computer Vision”, arXiv 2016
Inception ResNet: Szegedy et al, “Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning”, arXiv 2016
MobileNet: Howard et al, “Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv 2017

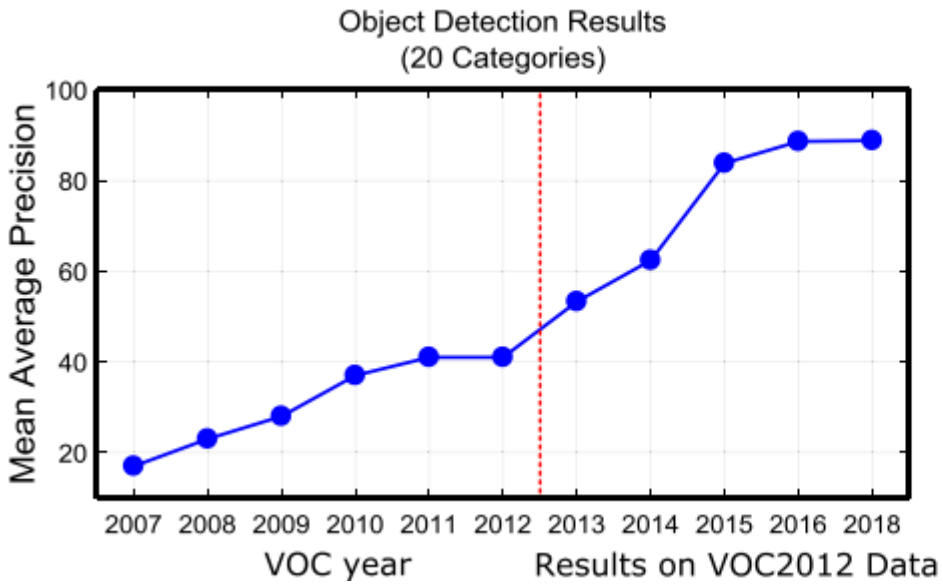
Training sets for Visual objects detection

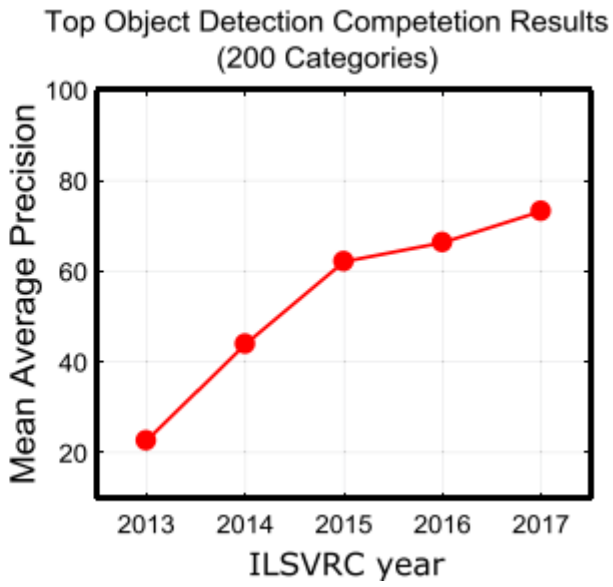
- Training a visual objects detector requires a training set contain images WITH BOUNDINGBOXES (or even mask) ANNOTATION
- Two main « reference » training sets of this type:
  - Pascal VOC (Visual Object Class)

    http://host.robots.ox.ac.uk/pascal/VOC/
  - Coco (Common Objects in Context) [more classes + MASK annotations] http://cocodataset.org/



An overview of recent object detection performance

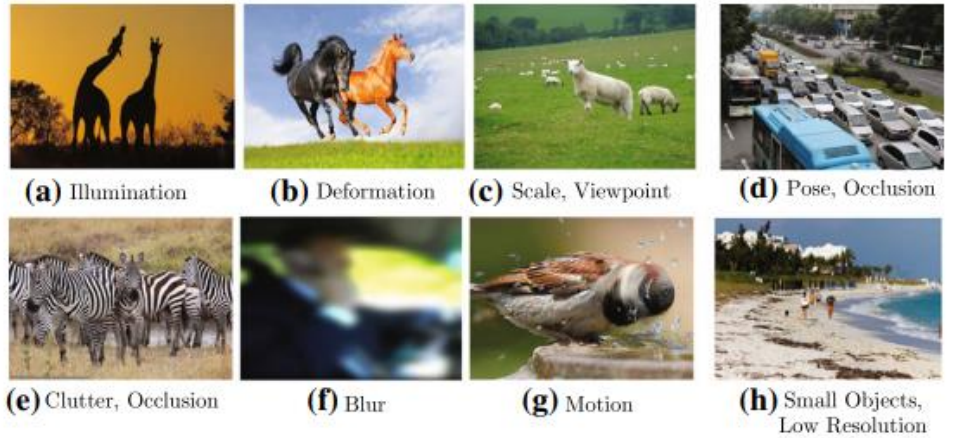




Turning point at 2012: Deep learning achieved record breaking Image Classification Result

The difficulties for image classification

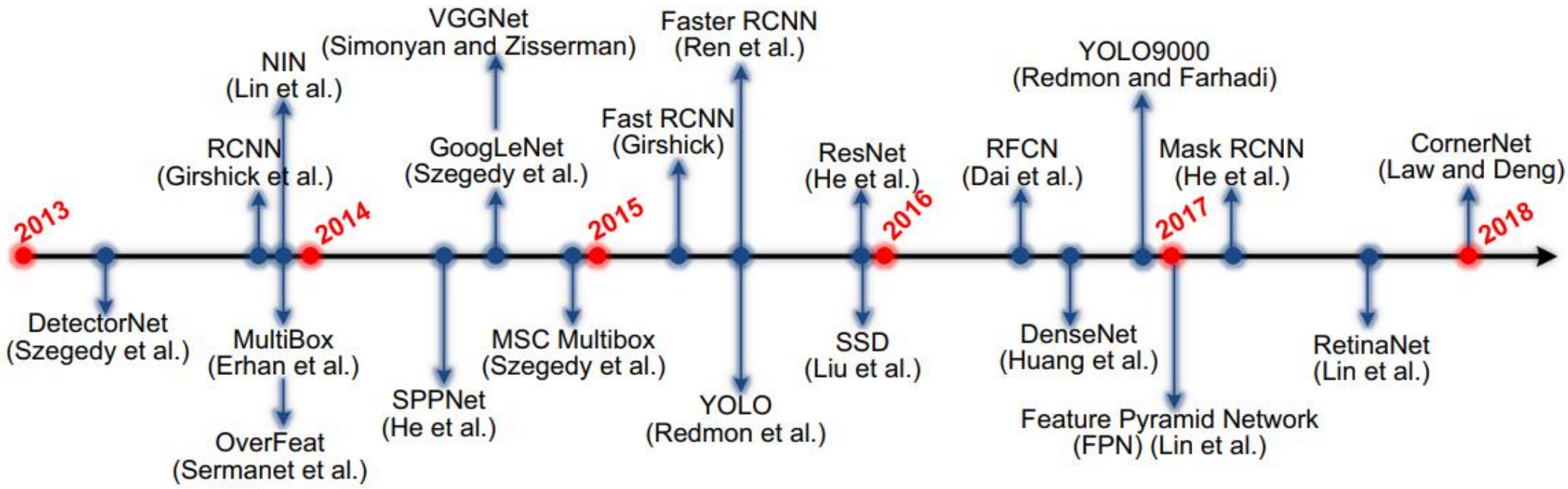
- (a-h) Changes in appearance of the same class with variations in imaging conditions
- (i) There is an astonishing variation in what is meant to be a single object class
- (j) In contrast, the four images in j appear very similar, but in fact are from four different object classes.
- Most images are from ImageNet (Russakovsky et al. 2015) and MS COCO (Lin et al. 2014)



(i) Different instances of the "chair" category

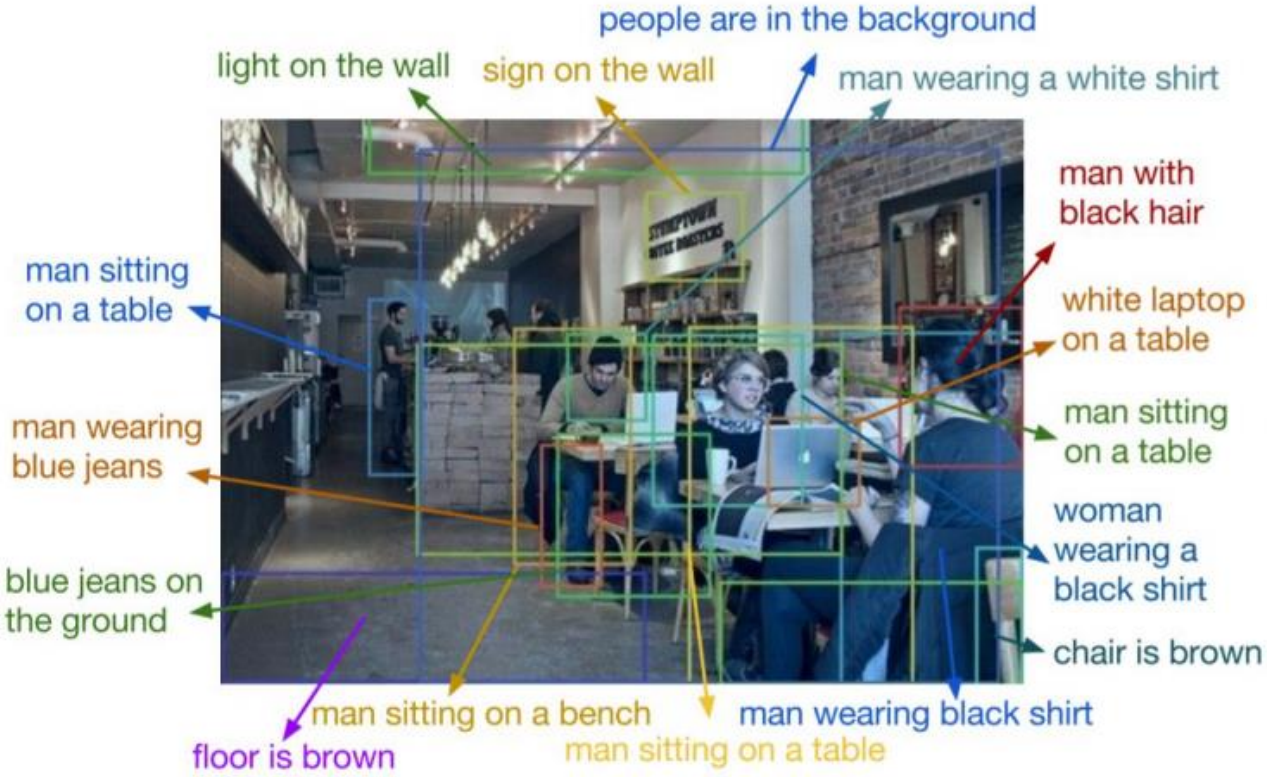
(j) Small Interclass Variations: four different categories

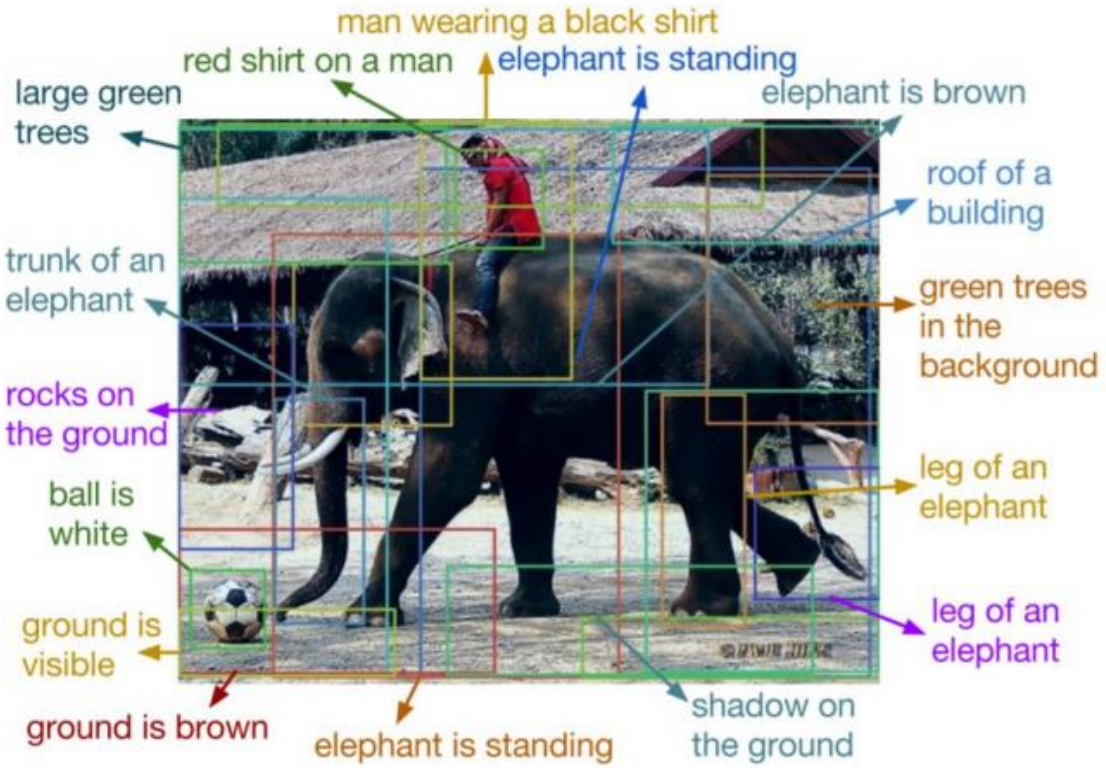
Milestones in generic object detection



[Liu, L., Ouyang, W., Wang, X. *et al.* Deep Learning for Generic Object Detection: A Survey. *Int J Comput Vis* **128,** 261–318 (2020)]

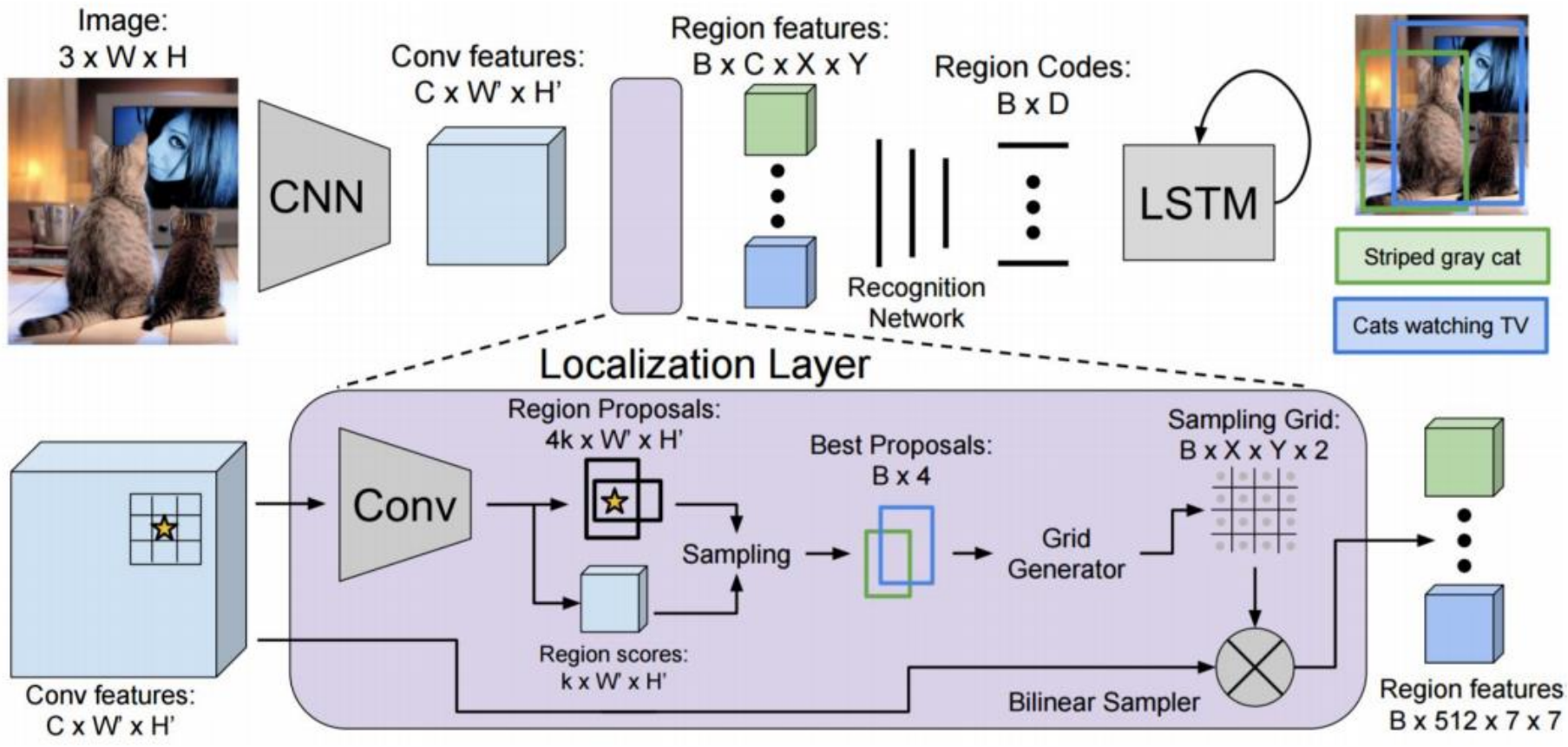
More applications with object detection





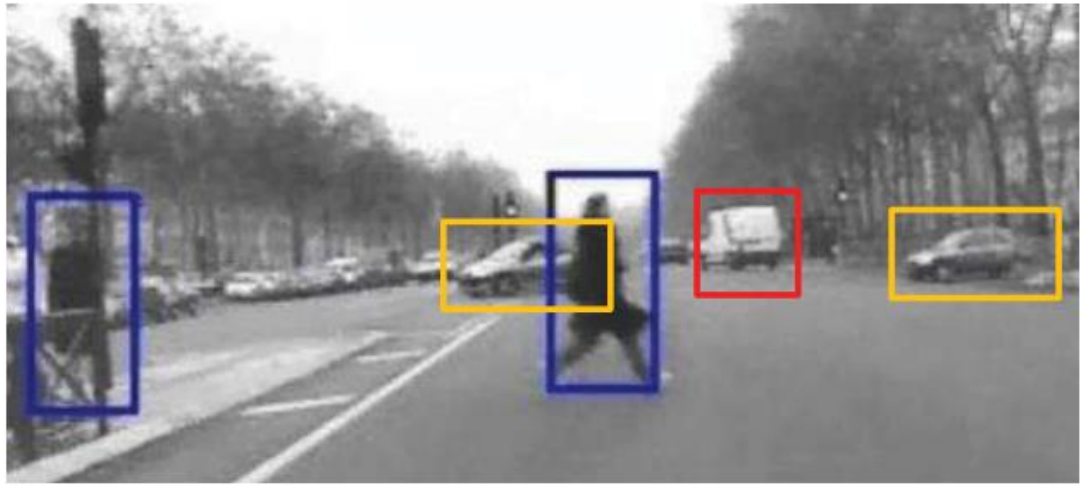
Johnson, Karpathy, and Fei-Fei, "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", CVPR 2016
Figure copyright IEEE, 2016. Reproduced for educational purposes.

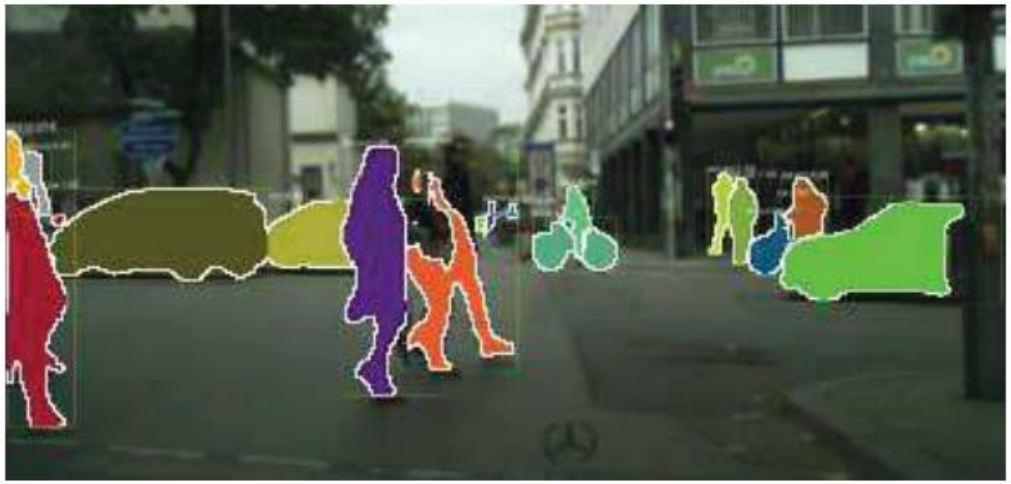
Dense captioning



Johnson, Karpathy, and Fei-Fei, "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", CVPR 2016
Figure copyright IEEE, 2016. Reproduced for educational purposes.

Drawbacks of object detections





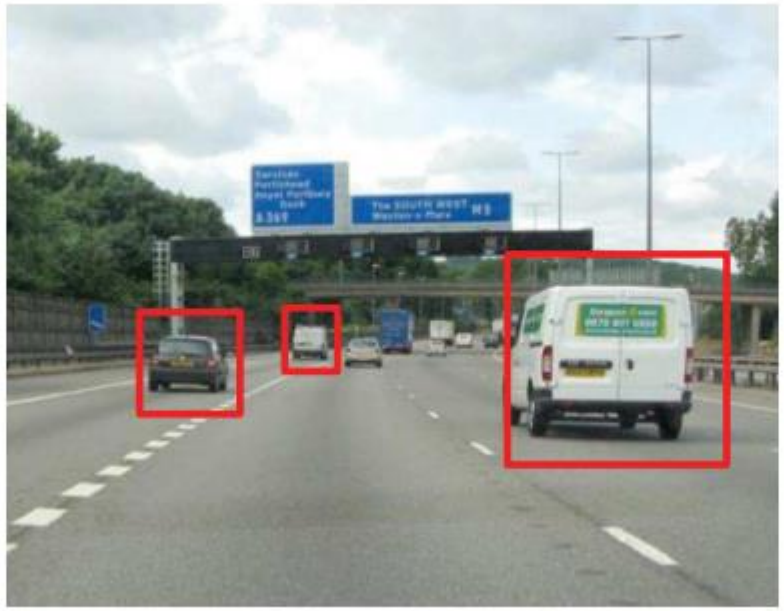
- Problem for objects without sharp boundaries (trees, …) or very dense group of objects (crowd of pedestrians, …)
- Only « compact » objects are categorized (what about « road », « sidewalk », « building », …?)

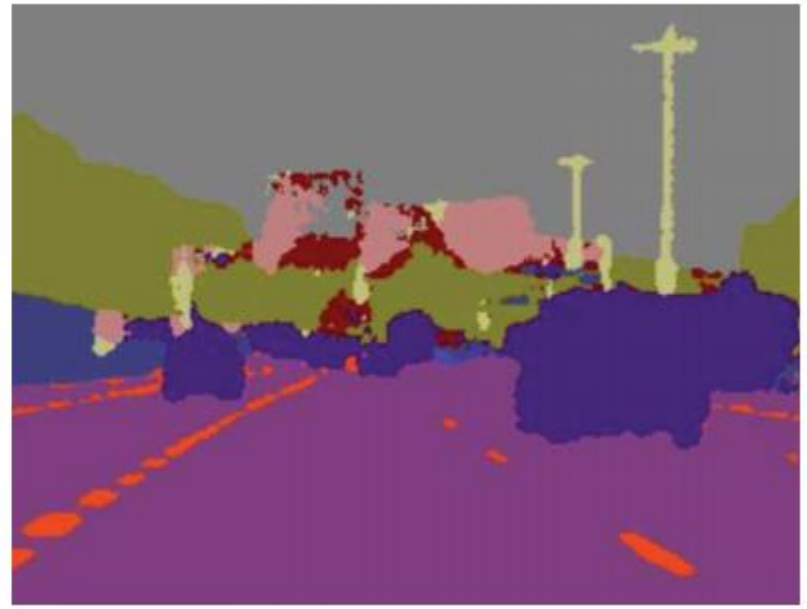
What is image segmentation

- Identify groups of continuous pixels that go together

Advantage of Semantic (full) segmentation

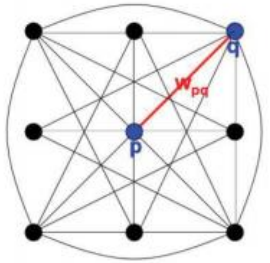
- One single semantic segmentation →all interesting object categories (cars, pedestrians, signs, etc…) and categorization of whole image
- Can also categorize non-compact areas (road, sky, buildings, trees, traffic lanes…)





Many approaches for image segmentation

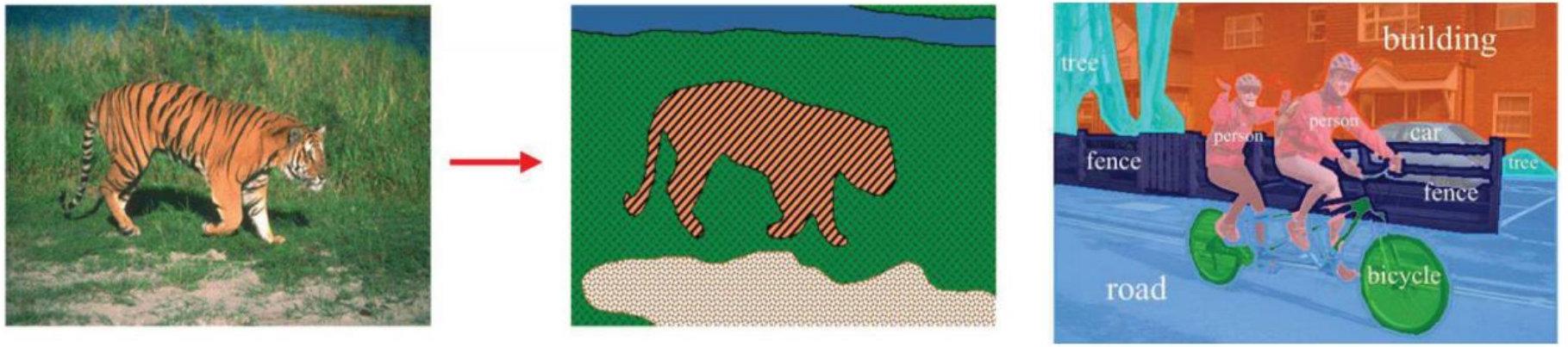
- Clustering (K-means, GMM, MeanShift, …)
- Graph-based (graph-cuts)



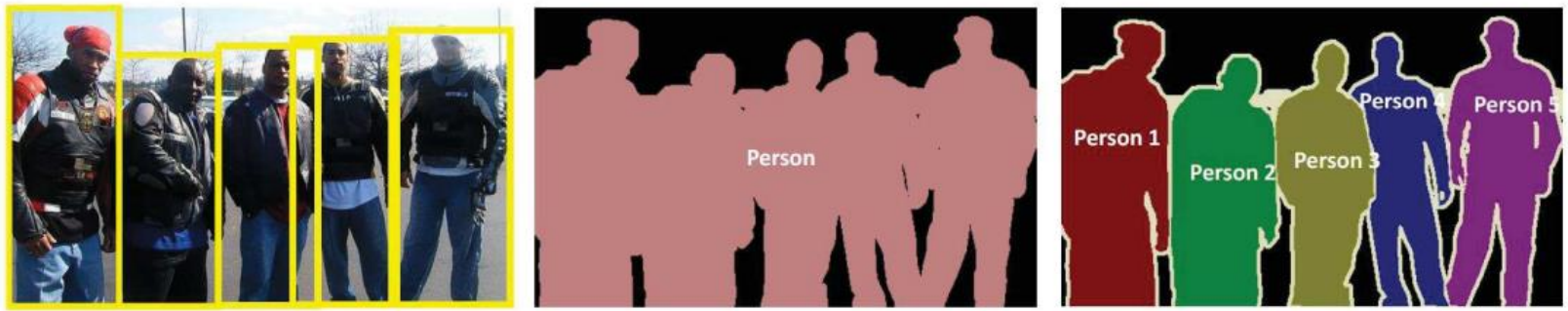
- Node (vertex) for every pixel
- Edge between pairs of pixels, (p,q)
- Affinity weight w_{pq} for each edge
  - w_{pq} measures similarity
  - Similarity is inversely proportional to difference (in color and position…)

- Mathematical Morphology (watershed, etc…)
- Energy minimization (Conditional Random Fields)
- Deep-Learning

What is semantic segmentation?

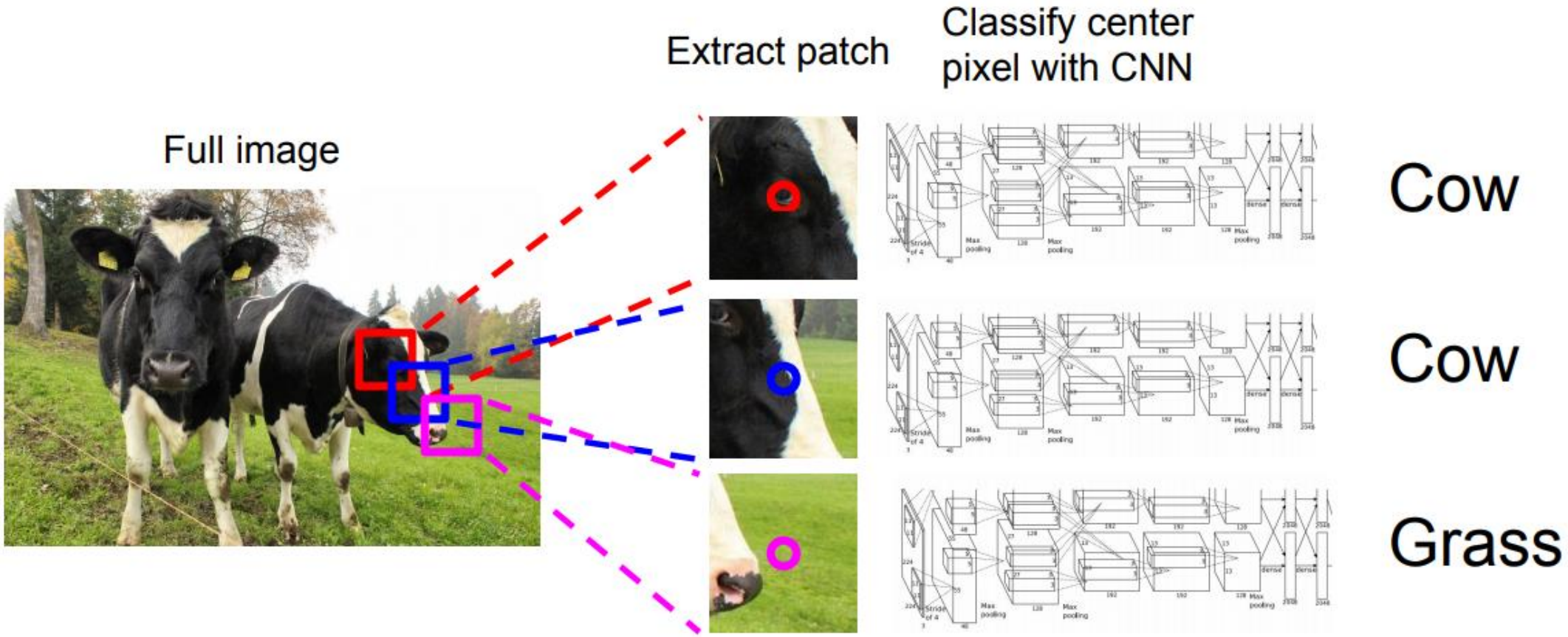


SEMANTIC segmentation:
« go together » = same « type of object »
≠ from just grouping pixels with similar colors or texture



Objects detection **Semantic Segmentation** Instance Segmentation

Deep-Learning approach idea 1: sliding window

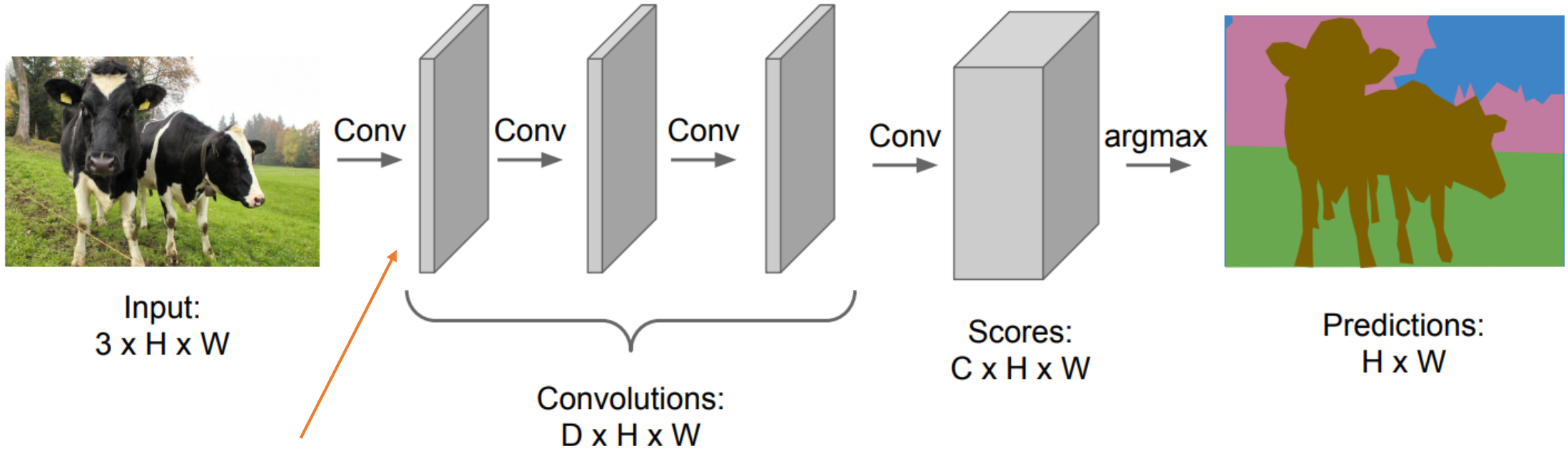


Very inefficient because each patch doesn't share features

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Deep-Learning approach idea 1: fully convolutional

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!

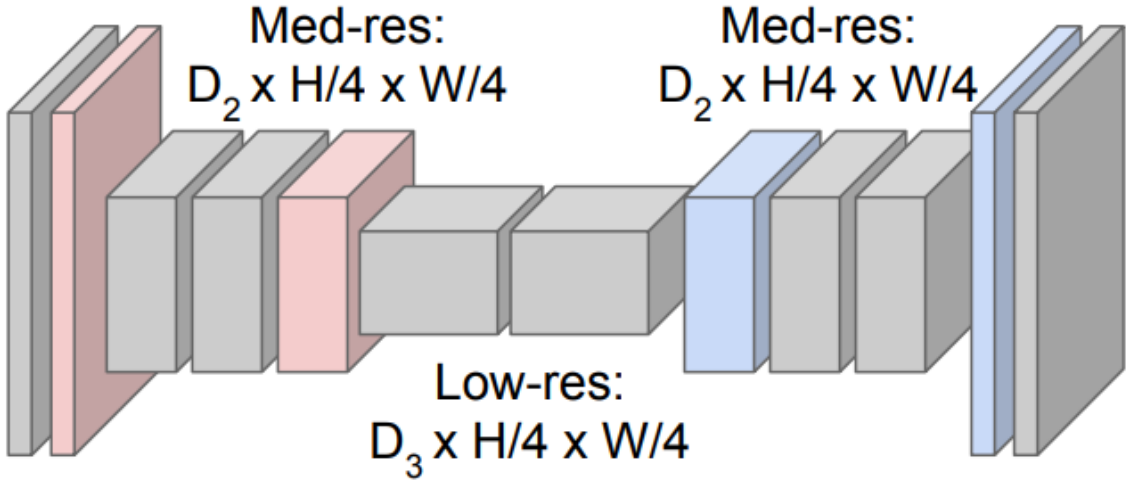


The first layer of convolution will be very expensive due to the resolution of image

Deep-Learning approach idea 1: fully convolutional

- Solution: using downsampling (stride, pooling) and upsampling (?)

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Uppooling

Nearest Neighbor

1	2
3	4

→

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Bed of nails

1	2
3	4

→

1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

But how and why we choose the upper let corner?

Max Uppooling

- A better and more sense way of bed of nails is to remember which element was max

Max pooling

2	5	3	1
1	3	7	4
3	1	2	1
2	2	4	8

→

5	7
3	8

→ Rest of network → … →

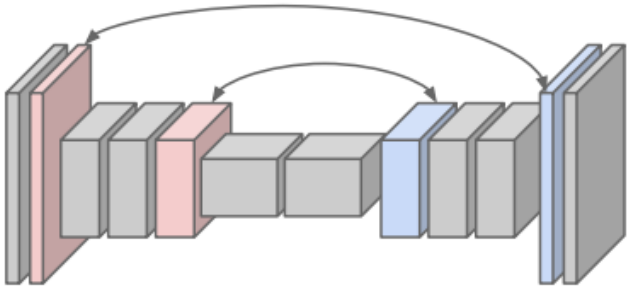
1	2
3	4

→

Max uppooling:
use position from pooling layer

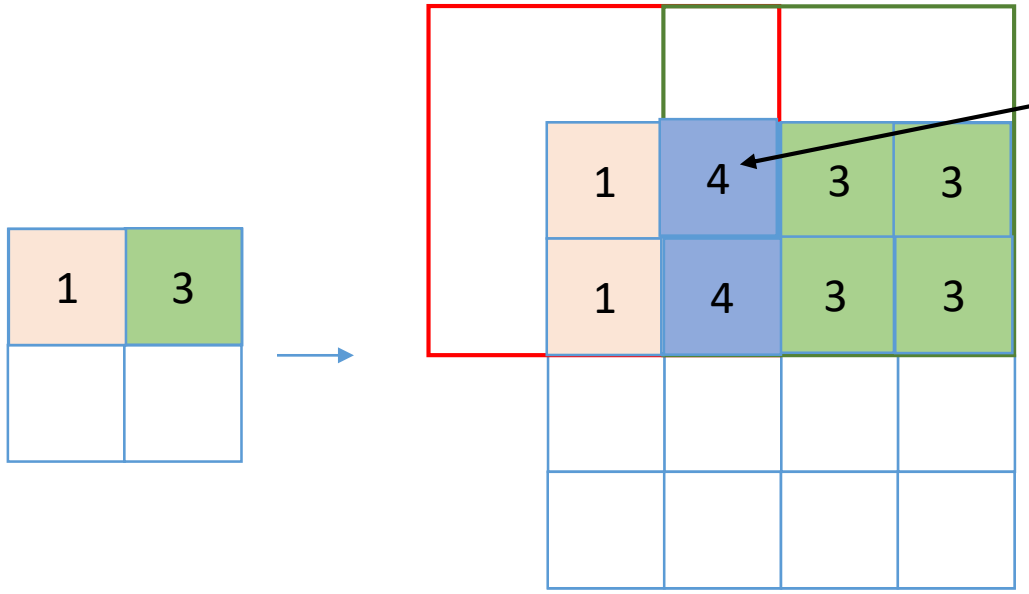
0	5	0	0
0	0	7	0
3	0	0	0
0	0	0	8

Corresponding pairs of downsampling and upsampling layers



Learnable upsampling: transpose convolution

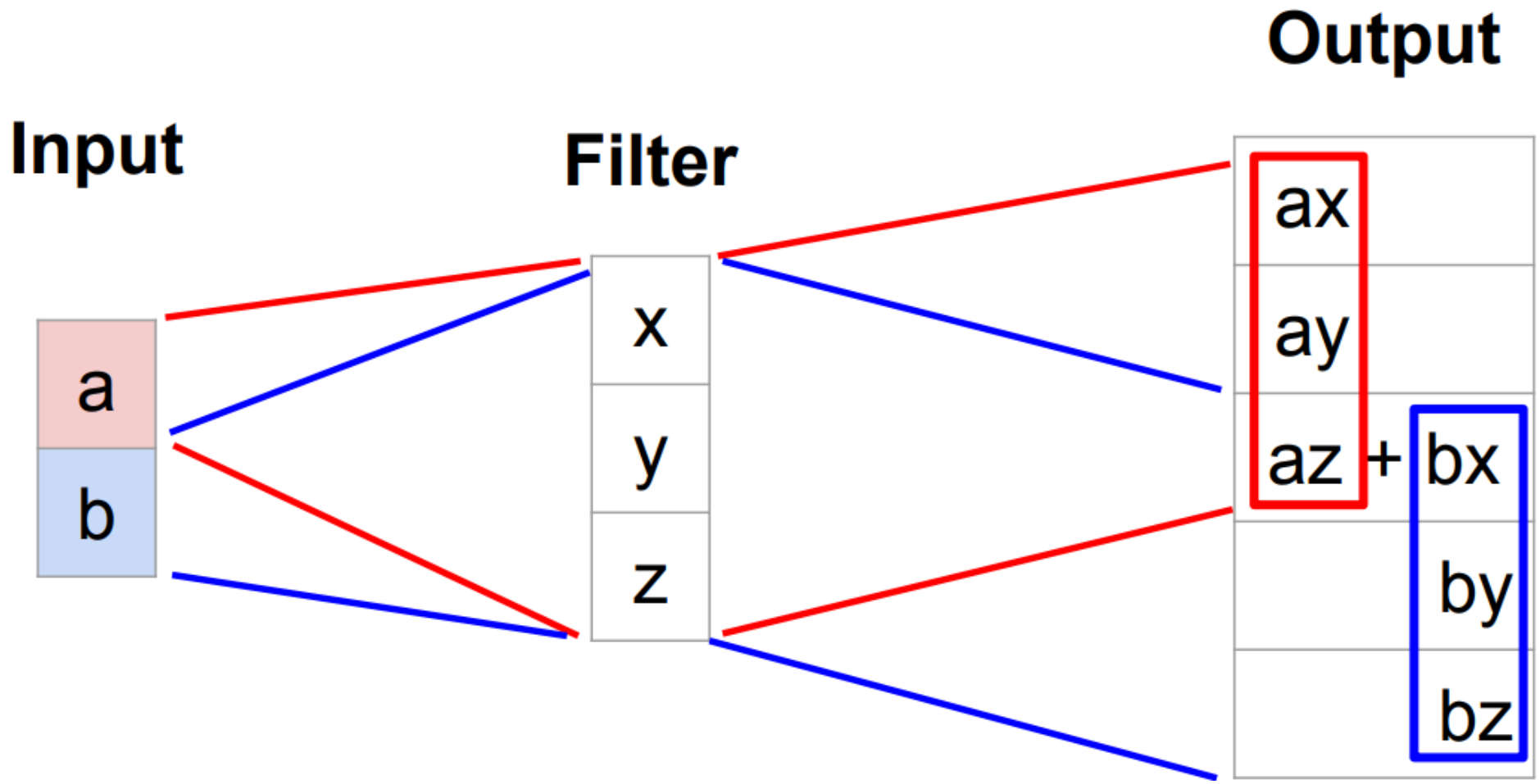
3x3 transpose convolution, stride 2 pad 1



Sum where output overlaps

- Other names:
  - Deconvolution (bad)
  - Upconvolution
  - Fractionally strided convolution
  - Backward strided convolution

Transpose convolution 1D example



Convolution as matrix operation

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

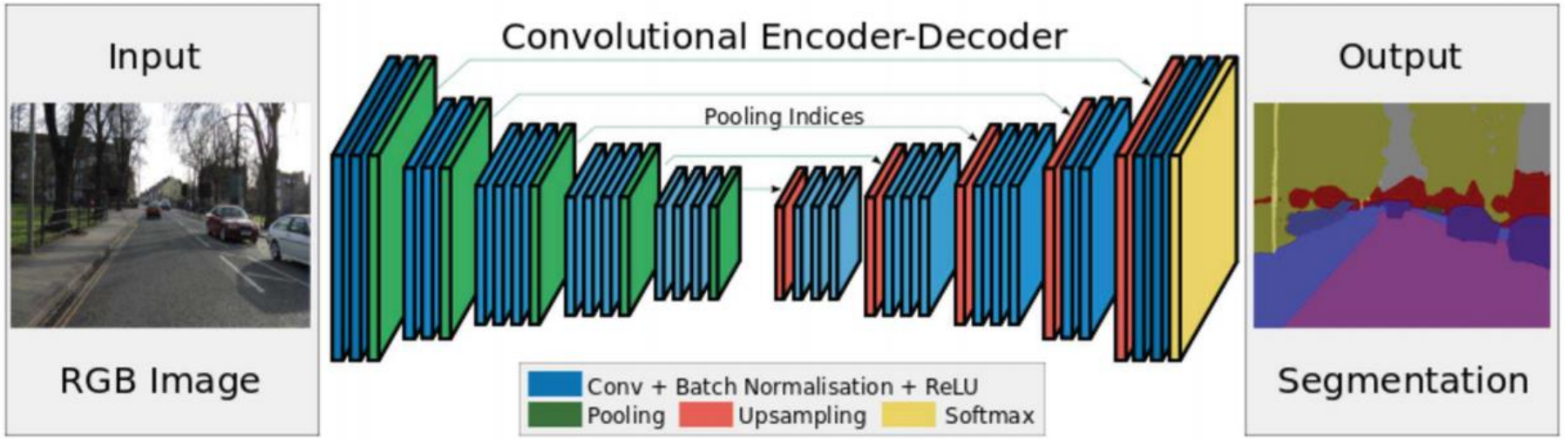
Convolution transpose multiplies by the transpose of the same matrix:

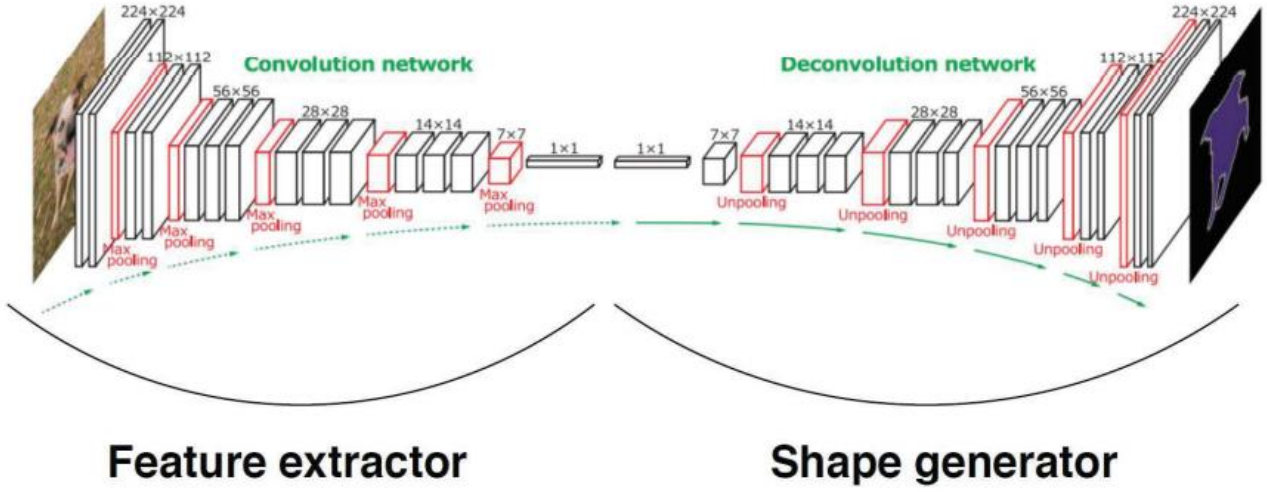
$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

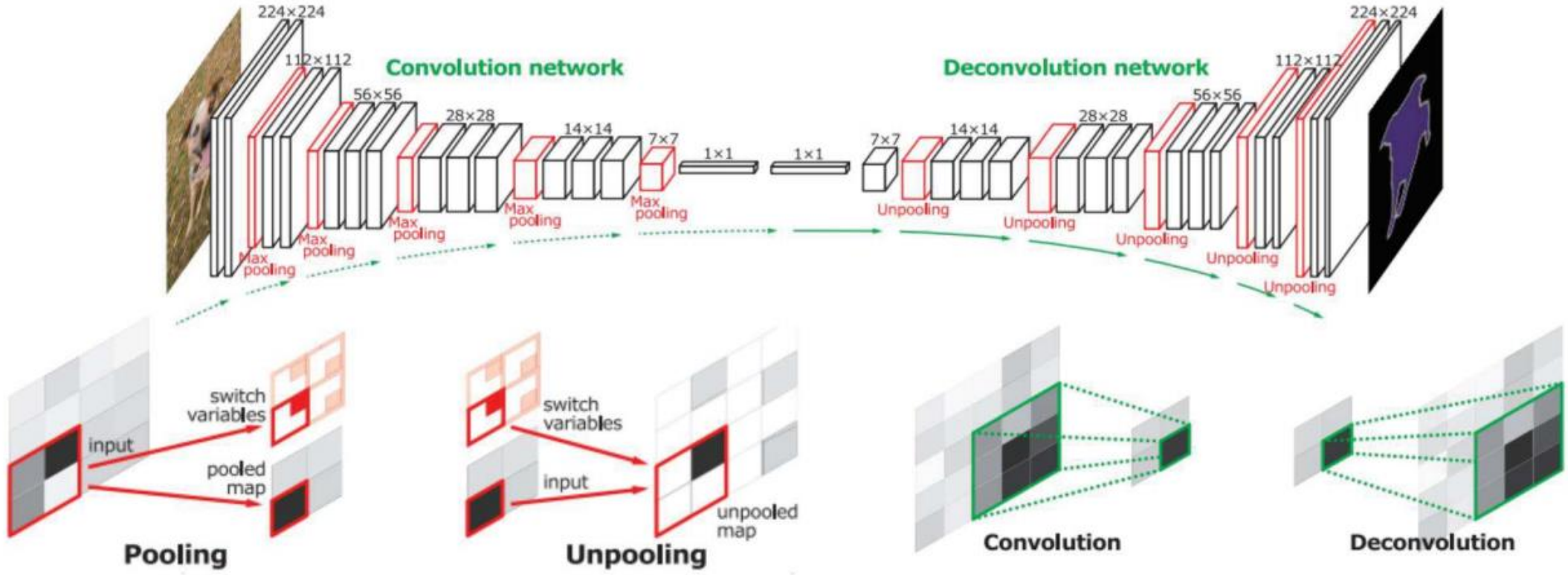
When stride=1, convolution transpose is just a regular convolution (with different padding rules)

Convolution encoder-decoder



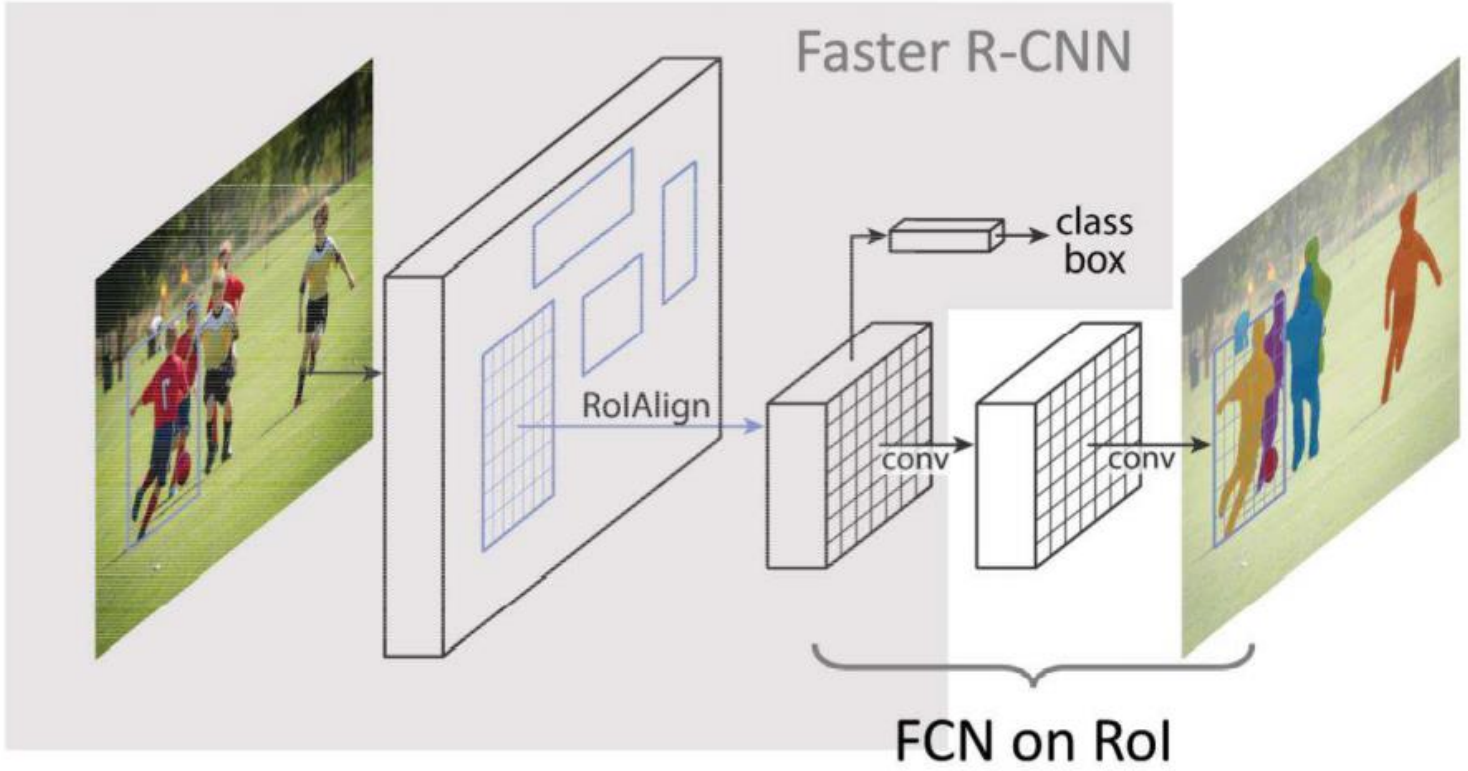


Encoder-Decoder network



Mask R-CNN

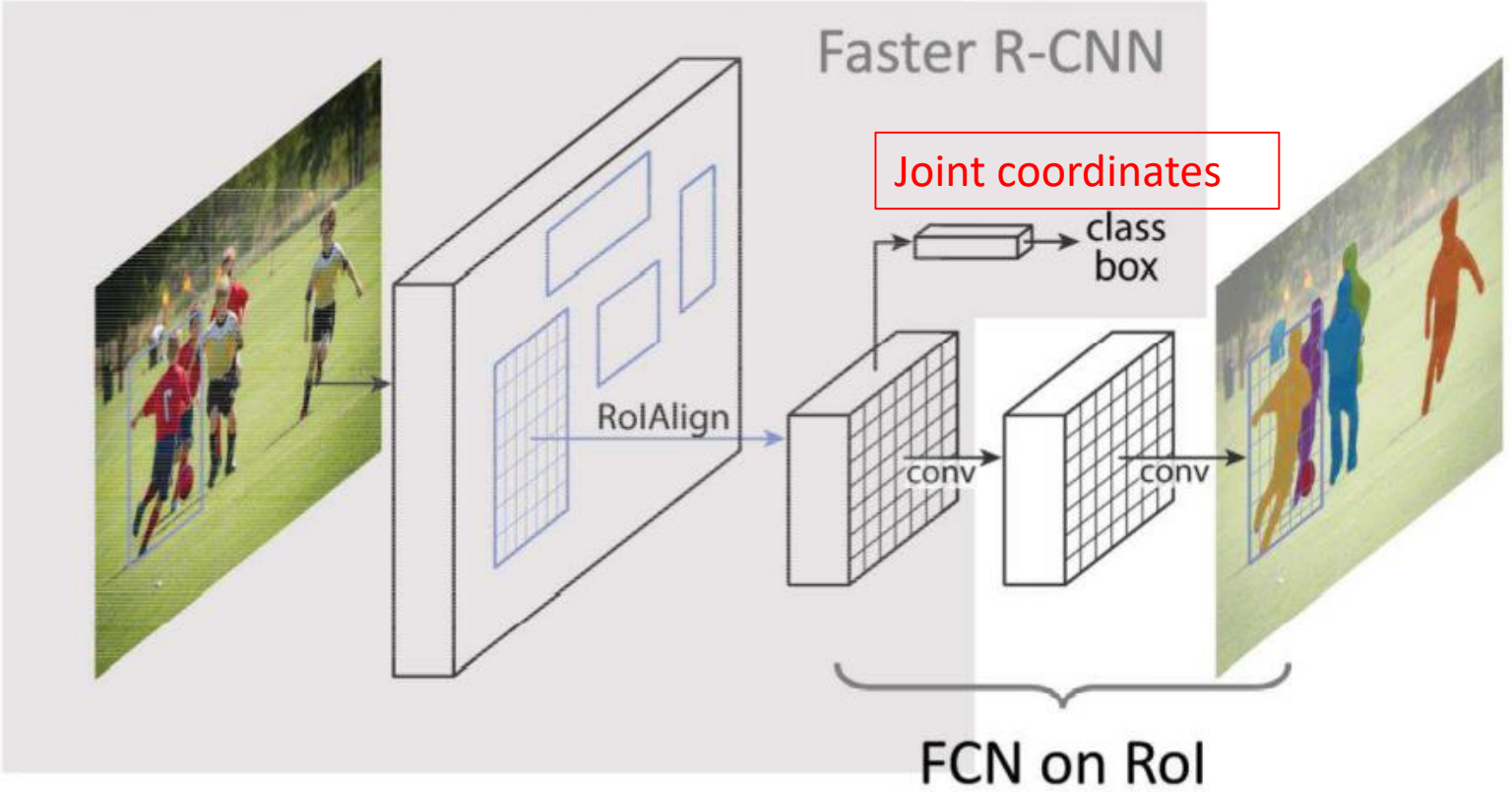
Mask R-CNN = **Faster R-CNN** with **FCN** on RoIs



Mask R-CNN architecture extract detailed contours and shape of objects instead of just bounding-boxes

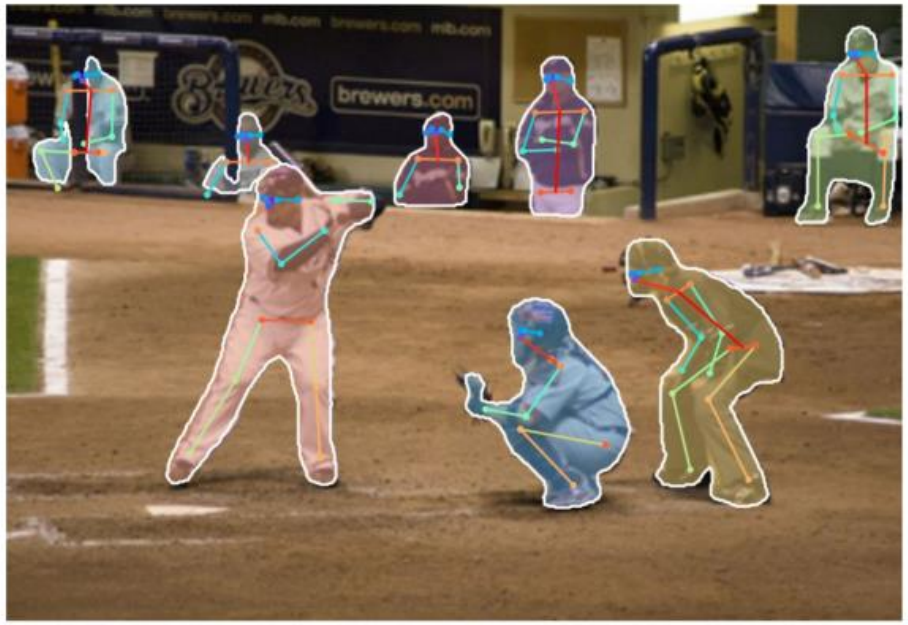
Addition function of Mask R-CNN

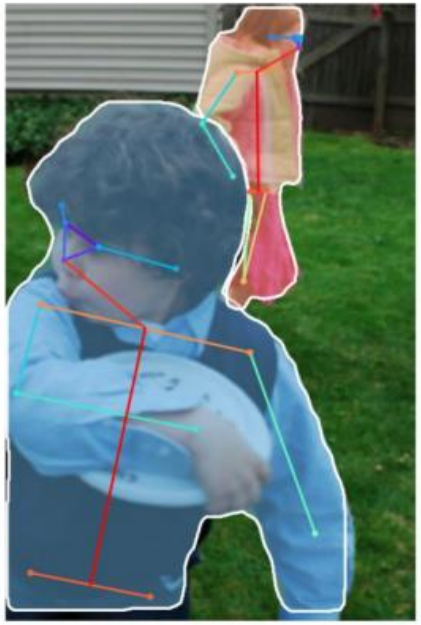
Mask R-CNN = **Faster R-CNN** with **FCN** on RoIs



Mask R-CNN architecture extract detailed contours and shape of objects instead of just bounding-boxes

Results from Mask R-CNN

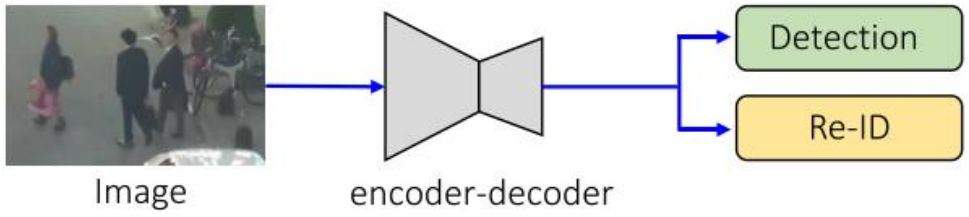


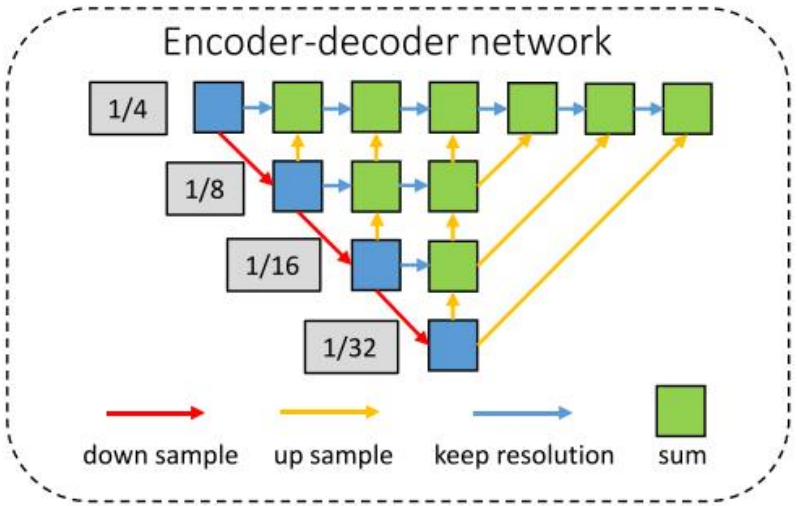



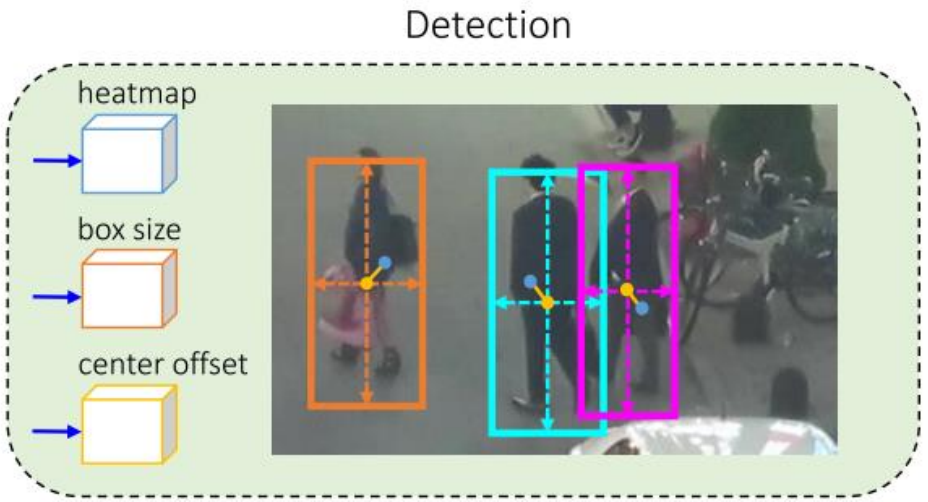
And many other competitors

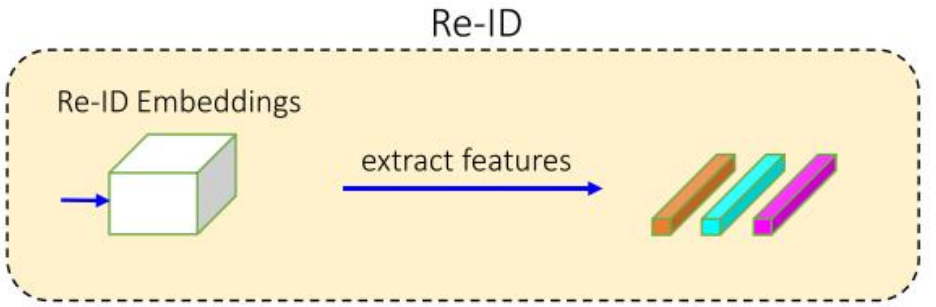
- 2015: U-Net (Keras)
  - https://github.com/zhixuhao/unet
- RefineNet (2016)
- DeepLab (Caffe)
  - https://github.com/Robotertechnik/DeepLab
- DeepLabv3 (Tensorflow)
  - https://github.com/NanqingD/DeepLabV3-Tensorflow

Tracking





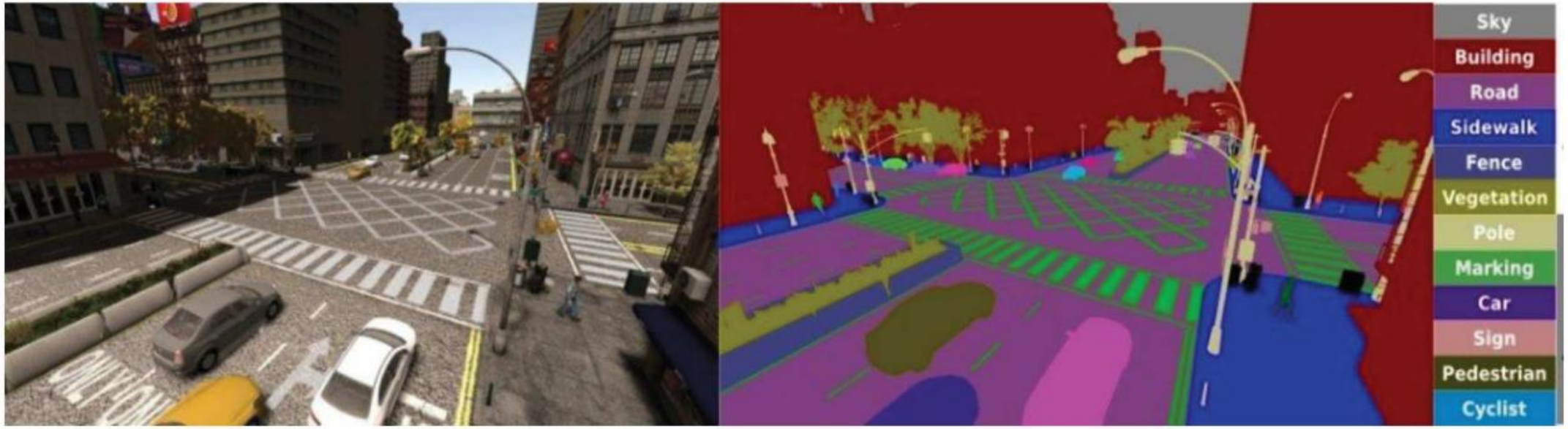




Zhang, Y., Wang, C., Wang, X., Zeng, W., & Liu, W. (2020). FairMOT : On the Fairness of Detection and Re-Identification in Multiple Object Tracking. *ArXiv Preprint ArXiv:2004.01888*. https://github.com/ifzhang/FairMOT.

Dataset for training segmentation

- It is very expensive to create dataset for training segmentation
- Synthetic images are popular in this case. They are very real.



Example from SYNTHIA

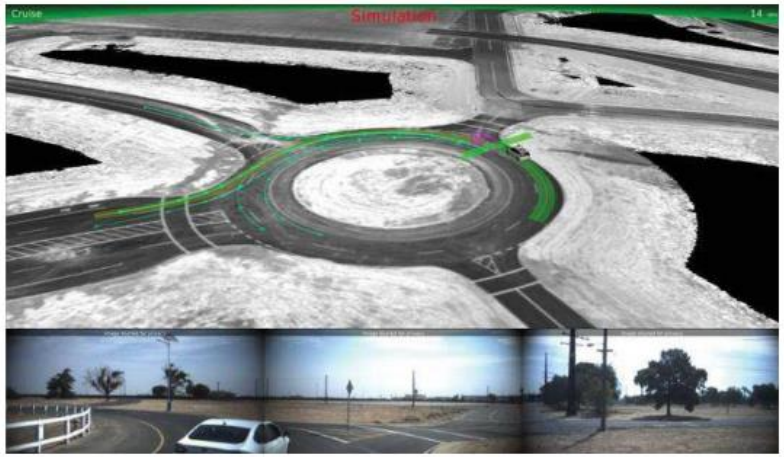
(http://synthia-dataset.net)

Interest of synthetic images for Machine-Learning

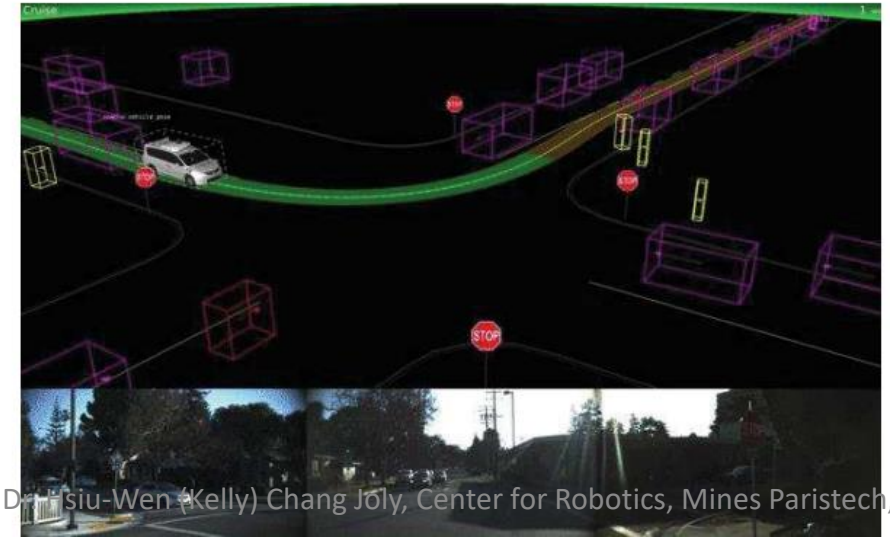
- Possible to generate as many as needed at nearly no cost (in particular compared to recording while driving)
- Easy to generate controlled variability in environment, luminosity conditions, scenarii, etc + also images « dangerous situations »
- NO NEED FOR MANUAL LABELLING: ground truth (ie target value) for classifiers, localizers, and semantic segmentation provided automatically

Simulators dedicated to Autonomous Vehicles

Scenario-buiding with CarCraft by Google/Waymo



Simulation of a virtual scenario in XView by Google/Waymo



CARLA open-source urban driving simulator

- Still few driving simulators adapted for DL and RL, and best ones not totally mature

Simulateur	GTA	DeepDrive.io	AirSim	CARLA[1]
Flexibilité	− −	++	++	++
Variété	++	− −	−	+
Complexité/Réalisme	++	− −	−	−
Objets mobiles	++	− −	− −	+
Vitesse éxecution	− −	+	+	+
Multi-agent	− −	−	−	++

→ Choice of CARLA

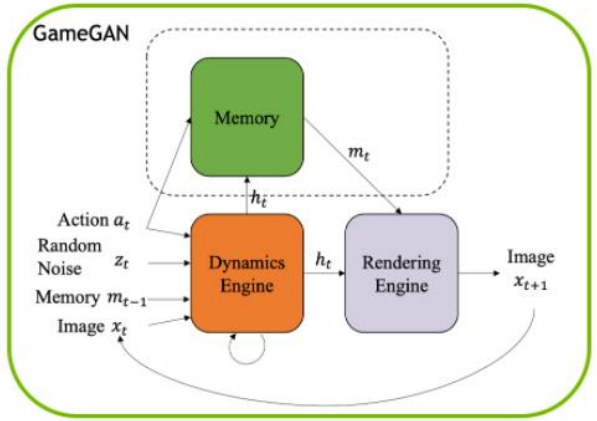
[1] A. Dosovitskiy: CARLA: An Open Urban Driving Simulator (2017)

Synthetic images use in ML/DL for IV

- Initial training of a classifier / segmented / controller only on simulated images / videos / scenarios
- Possible to then adaptation to real-world by fine-tuning on REAL images/video datasets
- Cheaper / more extensive testing than on real-world videos
- REINFORCEMENT LEARNING in simulation !

We ask computer to do it?!

- Is it possible we ask the computer to create the dataset for us?
- The answer becomes more and more clear to us in the past few year
- GameGAN:
  - generative model that learns to visually imitate a desired game by ingesting screenplay and keyboard actions during training



- Dall-e (OpenAI, 2021):
  - creates images from text captions for a wide range of concepts expressible in natural language.
  - https://openai.com/blog/dall-e/

Question?

hsiu-wen.chang_joly@mines-paristech.fr