

# Self-Driving Cars

## Lecture 10 – Object Detection

Prof. Dr.-Ing. Andreas Geiger

Autonomous Vision Group

University of Tübingen / MPI-IS

EBERHARD KARLS  
**UNIVERSITÄT**  
TÜBINGEN



e l l i s  
European Laboratory for Learning and Intelligent Systems

# Agenda

**10.1** Introduction

**10.2** Performance Evaluation

**10.3** Sliding Window Object Detection

**10.4** Region Based CNNs

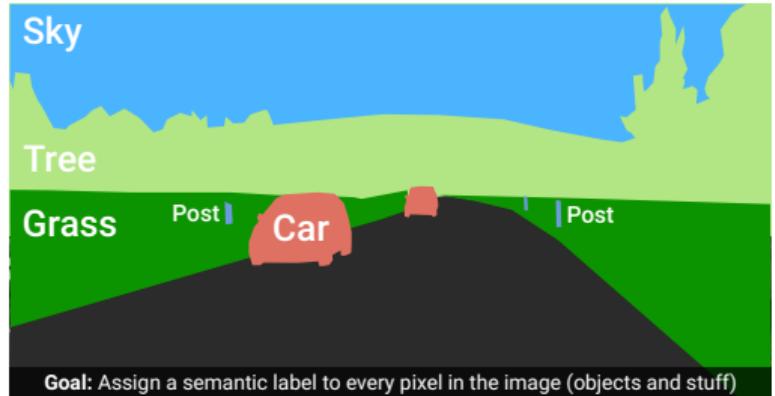
**10.5** 3D Object Detection

# 10.1

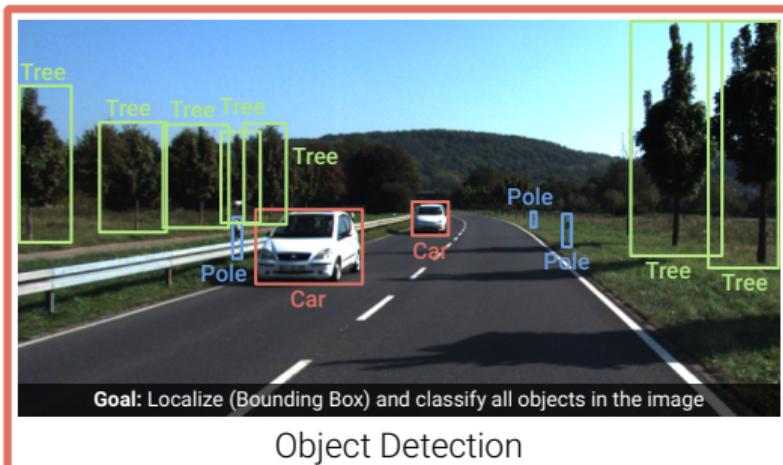
## Introduction



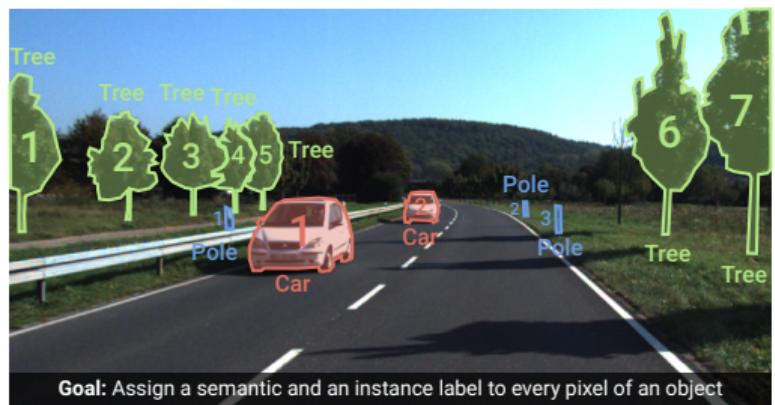
## Image Classification



## Semantic Segmentation



## Object Detection

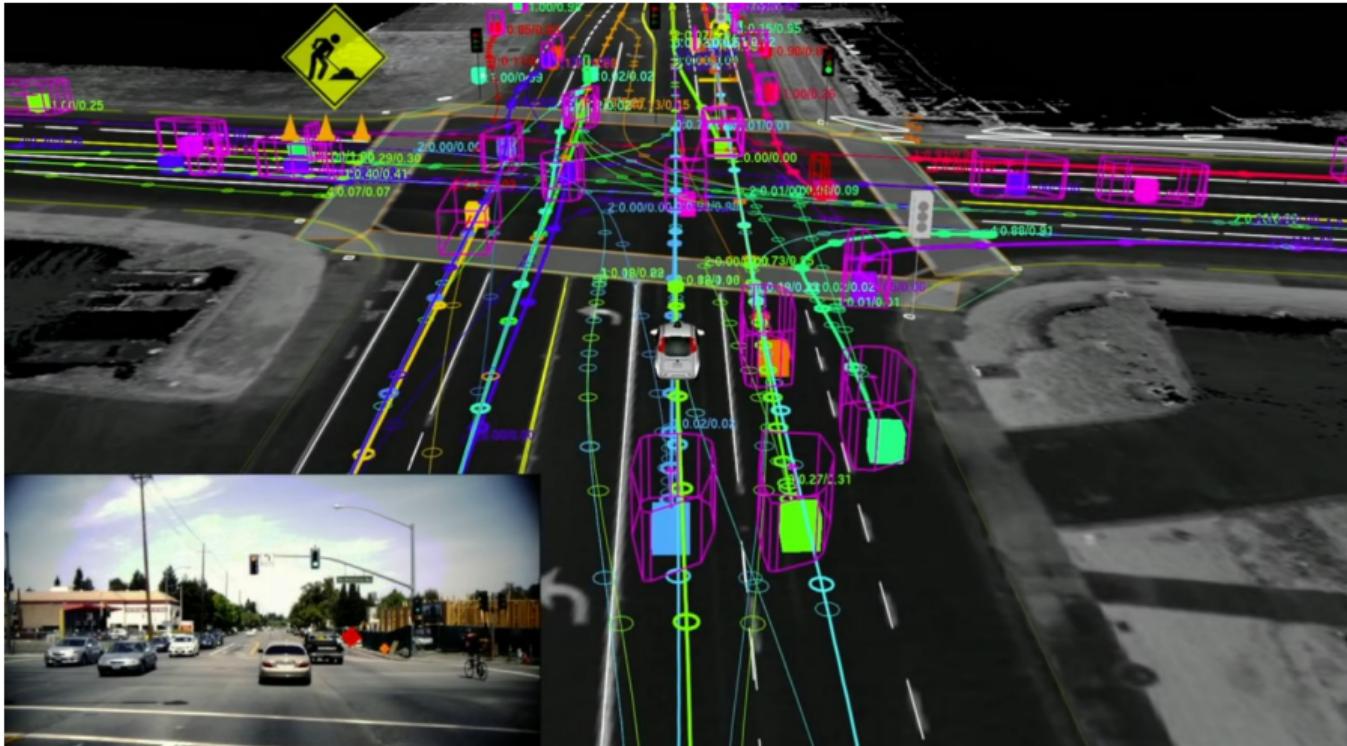


## Instance Segmentation

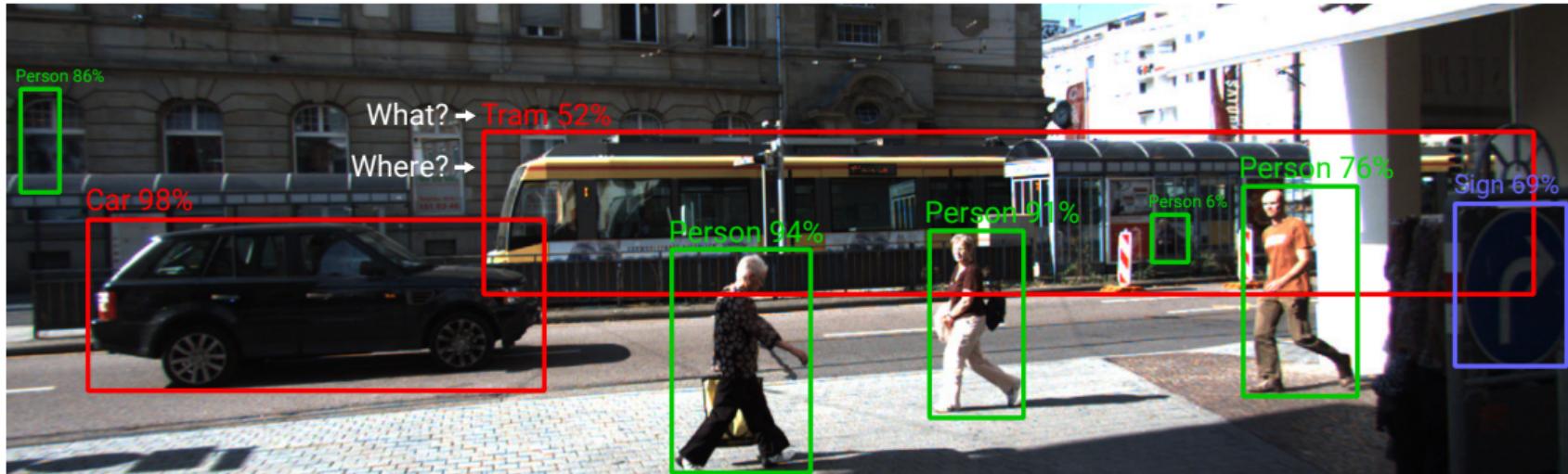
# Motivation



# Motivation



# Problem Setting



## Problem Setting:

- ▶ **Input:** RGB Image or laser range scan.
- ▶ **Output:** Set of 2D/3D bounding boxes with category label and confidence
- ▶ There are many ( $\infty$ ) possible boxes and even the number of objects is unknown

# Problem Setting



## Problem Setting:

- ▶ **Input:** RGB Image or laser range scan.
- ▶ **Output:** Set of 2D/3D bounding boxes with category label and confidence
- ▶ There are many ( $\infty$ ) possible boxes and even the number of objects is unknown

# Challenges

## Challenges: Large Number of Image Categories



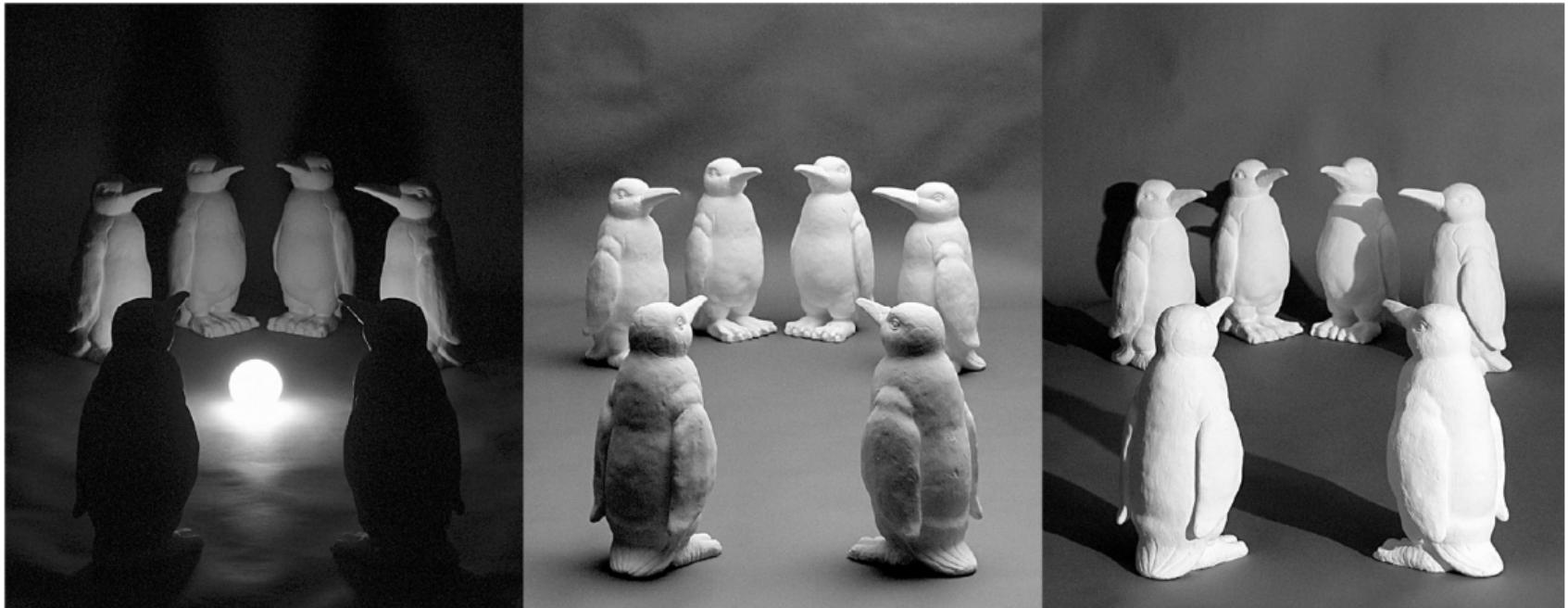
# Challenges: Intra-class Variation



# Challenges: Viewpoint Variation



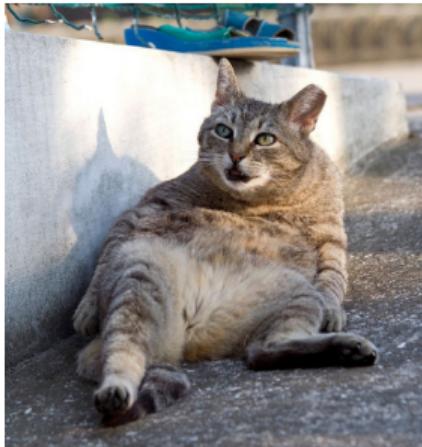
# Challenges: Illumination Changes



# Challenges: Background Clutter



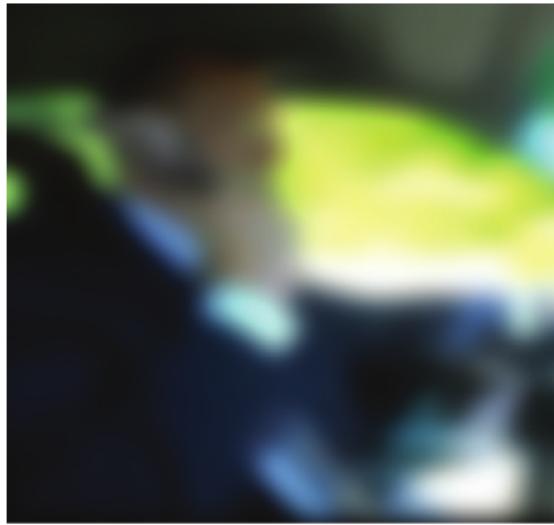
# Challenges: Deformation



# Challenges: Occlusion

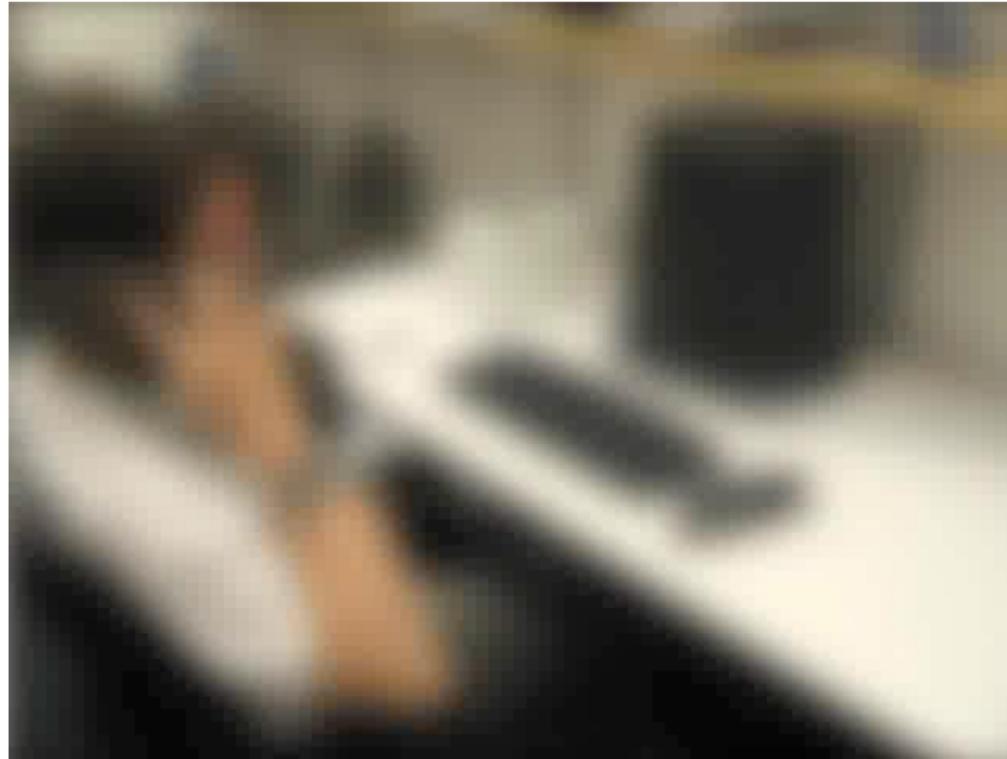


# Context Matters



- ▶ Contextual information is important! What do you see here?
- ▶ But it can also be misleading ...

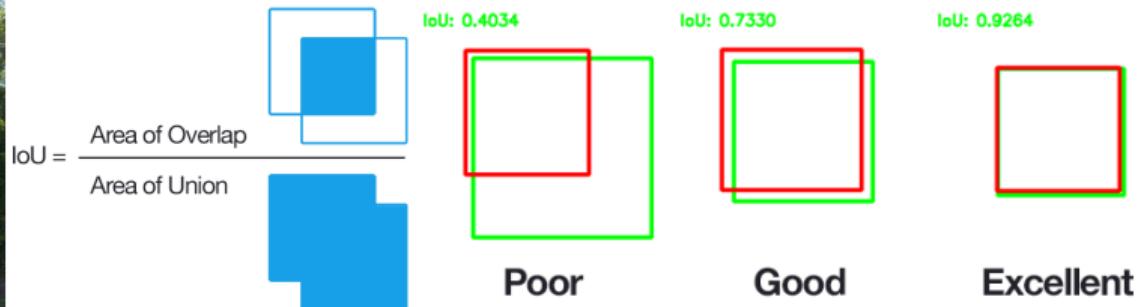
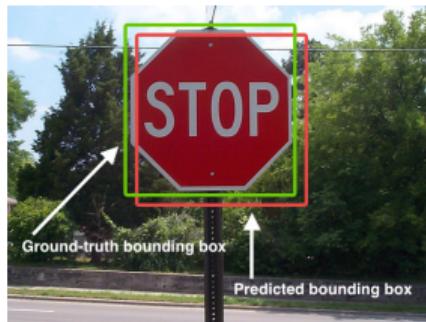
# Context Matters



## 10.2

# Performance Evaluation

# How to Measure Detection Performance?



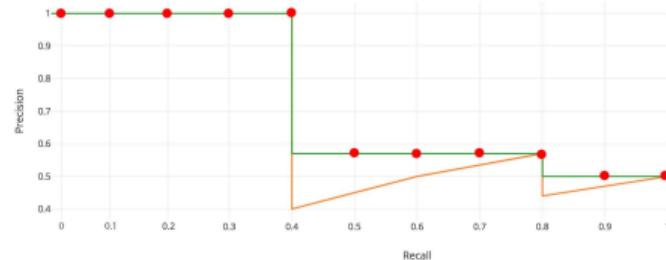
## Measuring Bounding Box Alignment wrt. Ground Truth:

- ▶ IoU = Intersection-over-Union (**predicted bbox** vs. **true bbox**)
- ▶ Detection considered successful if  $\text{IoU} > 0.5$
- ▶ How to fairly measure detection performance in case of multiple objects?
- ▶ Number of detections depends on detector threshold and is detector specific

# How to Measure Detection Performance?

## Average Precision Metric:

1. Run detector with varying thresholds
2. Assign detections to closest object
3. Count TP, FP, FN
4. Compute **Average Precision (AP)**



**True Positives TP:** Number of true objects correctly detected ( $\text{IoU} > 0.5$ )

**False Negatives FN:** Number of true objects not detected ( $\text{IoU} < 0.5$ )

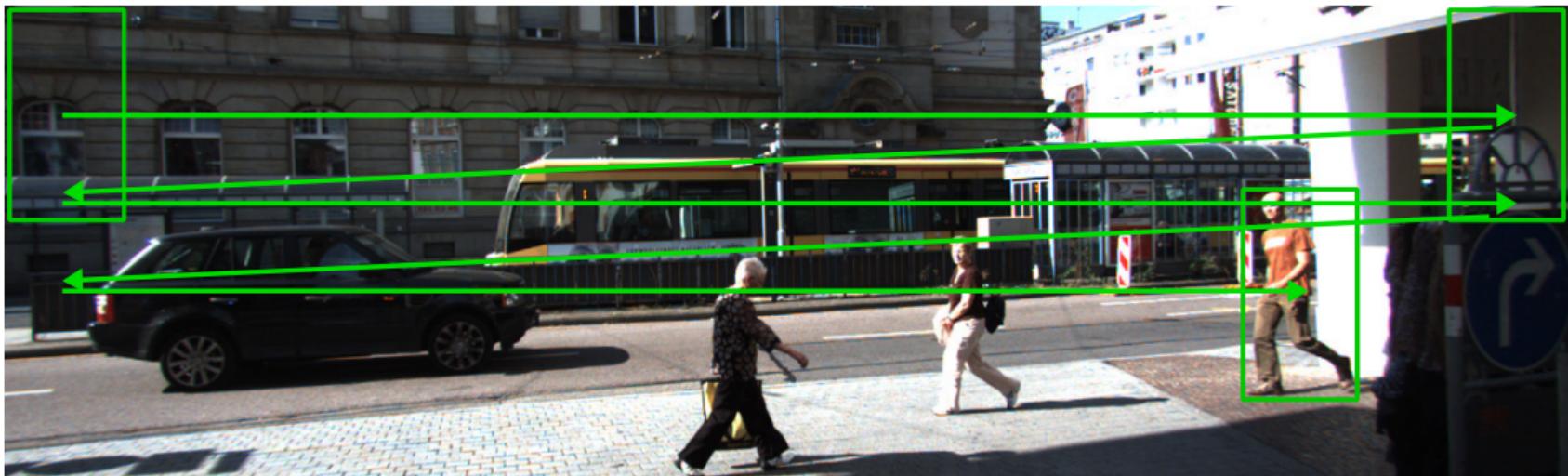
**False Positives FP:** Wrong detections

$$\begin{aligned}\text{Precision } P &= \frac{TP}{TP + FP} \\ \text{Recall } R &= \frac{TP}{TP + FN} \\ \text{Avg. Prec. } AP &= \frac{1}{11} \sum_{R \in \{0, \dots, 1\}} \max_{R' \geq R} P(R')\end{aligned}$$

## 10.3

# Sliding Window Object Detection

# Sliding Window Object Detection



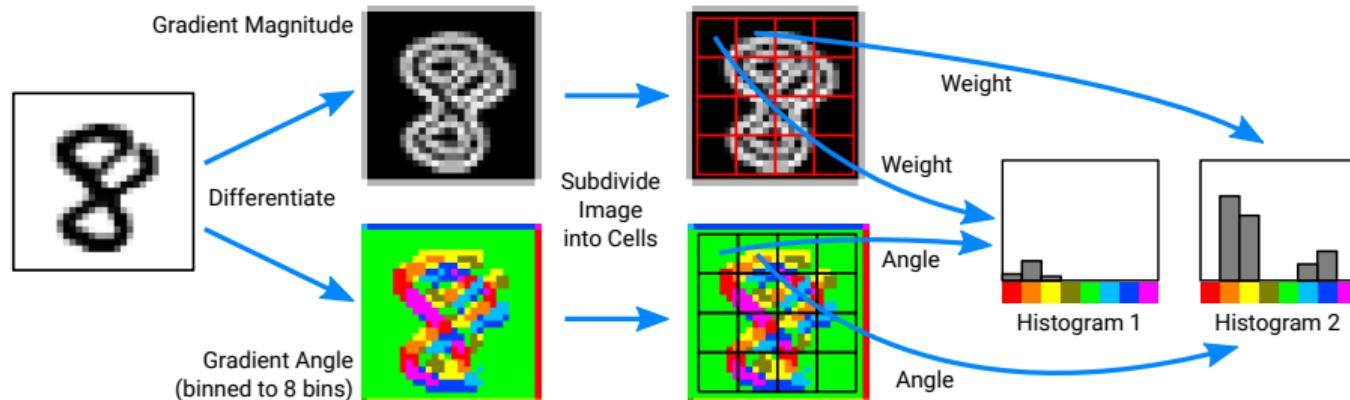
## Sliding Window Detection:

- ▶ Run **sliding window** of fixed size over the image; extract features for each window
- ▶ **Classify each crop** (object vs. background) using SVM, random forest, boosting, ..
- ▶ **Search across aspect ratios/scales** to recover objects of varying size/distance
- ▶ **Non-maxima suppression** (=clustering) to retrain only 1 prediction per object

# Histograms of Oriented Gradients

## Which feature space to use?

- ▶ Simplest: RGB pixel space  $\Rightarrow$  neither viewpoint nor illumination invariant
- ▶ Better: Histograms of Oriented Gradients (HoG) (defacto standard for >10 years)
  - ▶ Idea: represent patches with histograms (gradient angles weighted by magnitude)

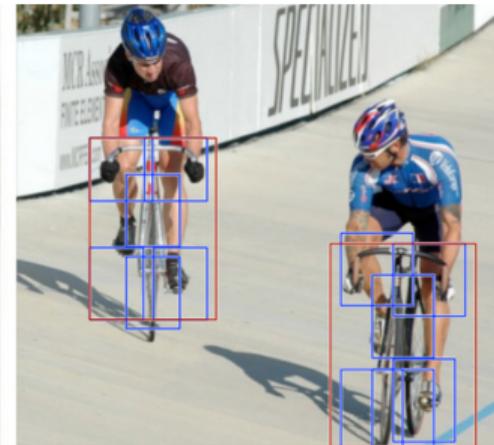
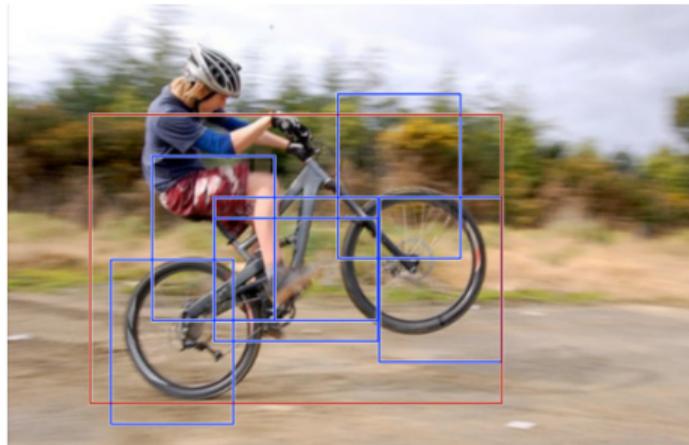
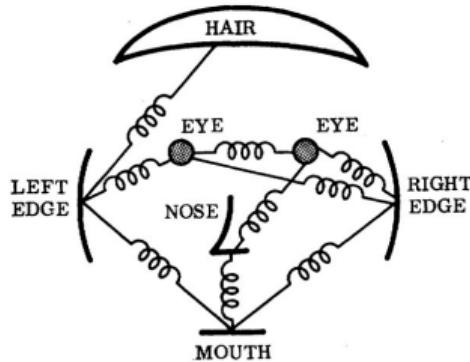


- ▶ HoG is invariant to small deformations (translation, scale, rotation, perspective)

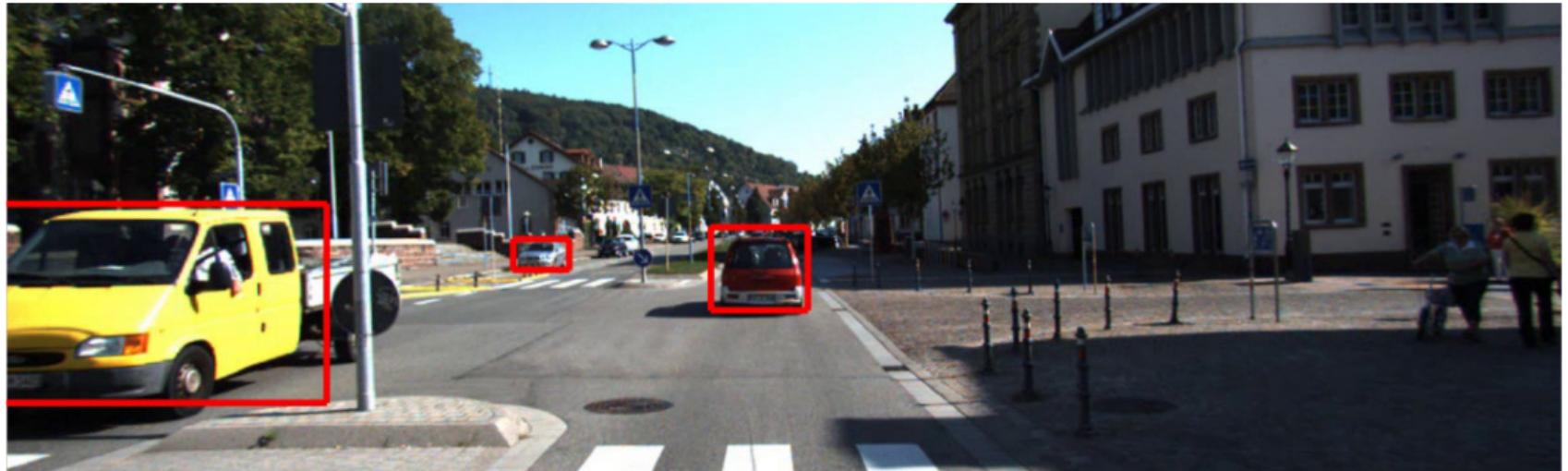
# Part Based Models

## Part-based Models:

- ▶ Idea: Model object based on its parts, **model distribution of part configurations**
- ▶ Allows for even more invariance (non-rigid deformations etc.)
- ▶ However: **slower inference** and not much gain wrt. multi-view HoG models



# Results on KITTI



## 10.4

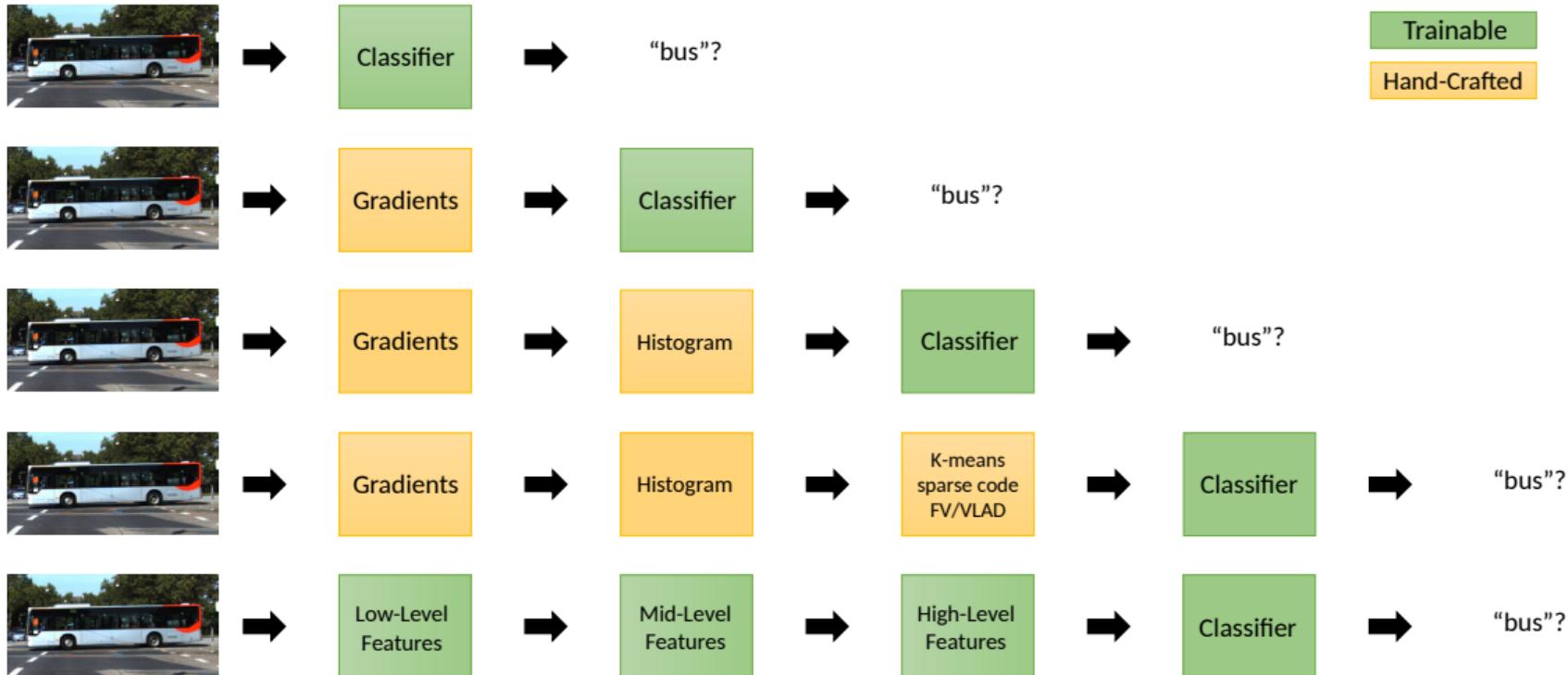
### Region Based CNNs

# Recognition - A Success Story?

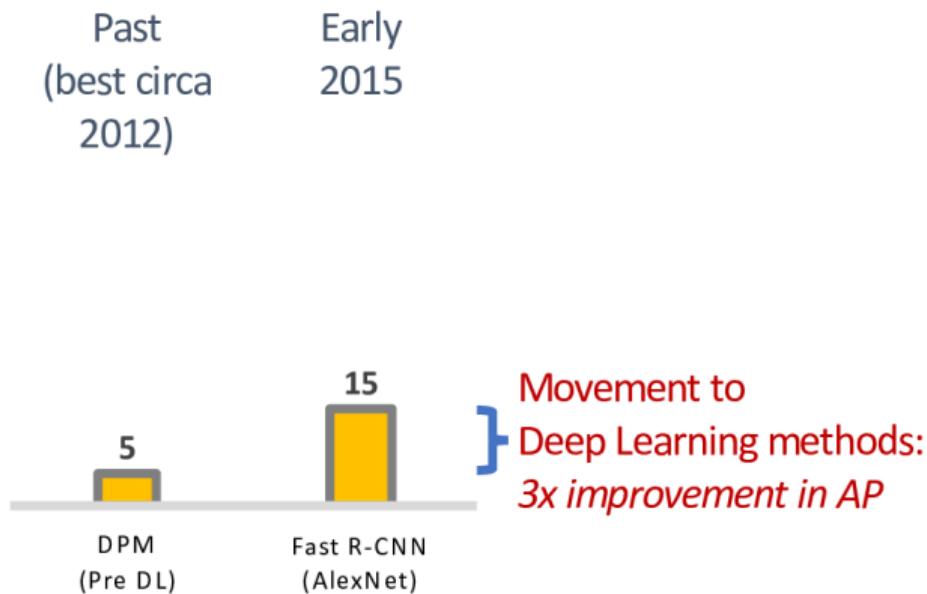
Classical approaches **haven't lead to major breakthroughs**: only for specific non-safety-critical tasks (e.g., face detection), they have seen great success. **Why?**

- ▶ Computer vision features are very difficult to hand-engineer
- ▶ It is unclear how a good representation should look like
- ▶ Furthermore, sliding window approaches are slow in practice
- ▶ Inference in complicated part-based models is even slower
- ▶ Computation is often not efficiently re-used
- ▶ But: deep learning has changed this ..

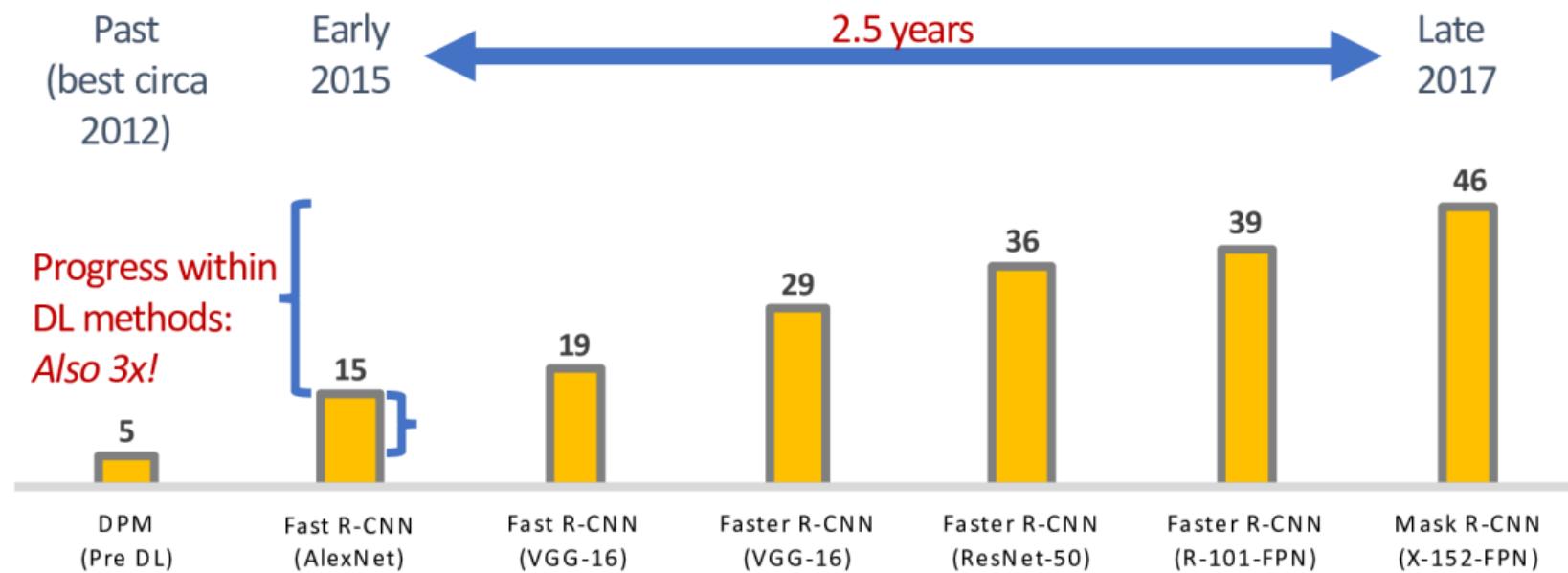
# Hand-crafted Representations vs. Learned Representations



# Hand-crafted Representations vs. Learned Representations



# Hand-crafted Representations vs. Learned Representations



# Object Detection with Deep Neural Networks

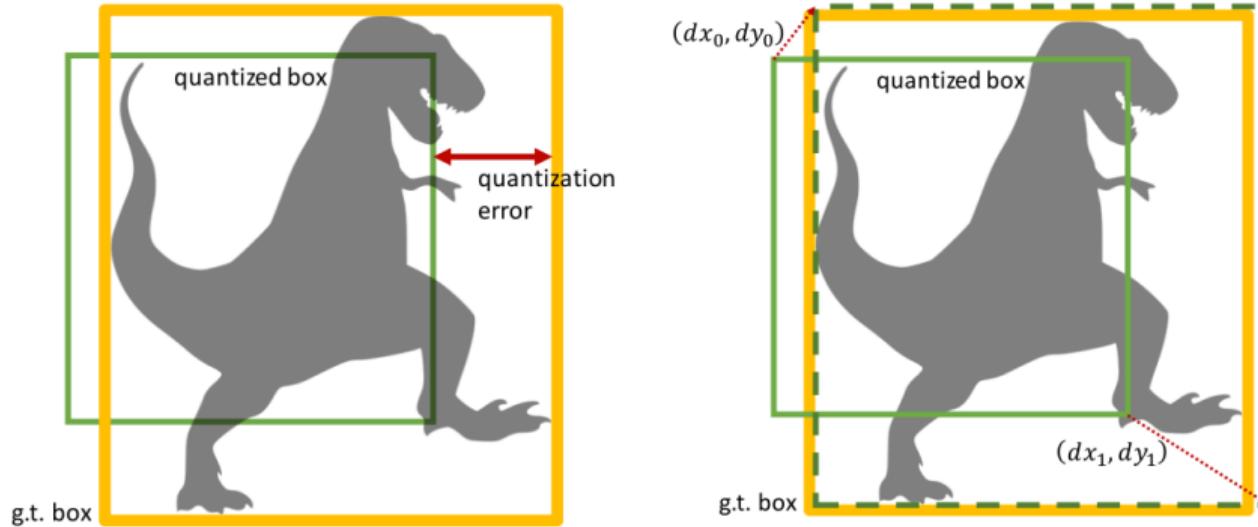
Slide Credits: Ross Girshick

# Proposal-based Object Detection

## How can we detect objects using deep neural networks?

- ▶ Naïve idea: Classify all possible boxes in the image using a classification network
- ▶ Problem: Too many boxes to classify (even if space is discretized)
- ▶ Better idea:
  1. **Detect**  $\sim 2k$  candidate bounding boxes that might contain an object (quantize)  
⇒ Choose set such that it is overcomplete (i.e., it should contain all boxes)
  2. **Classify and refine** the location the boxes using a deep neural network (regress)
- ▶ This approach is called a proposal-based (2-stage) object detector
- ▶ Note: We have split the original object detection problem into 2 proxy tasks

# Proposal-based Object Detection

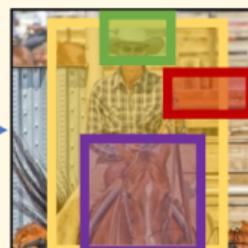
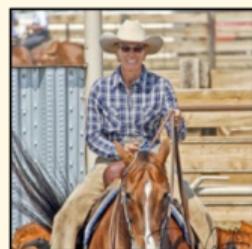


- ▶ Approximation with finite set of boxes leads to **quantization error** (quantize)
- ▶ Recover loss of localization accuracy by **regressing the location offset** (regress)
- ▶ Remove redundant predictions using non-maximum-suppression (clustering)

# R-CNN: Region-based Convolutional Neural Network

Per-image computation

$I:$

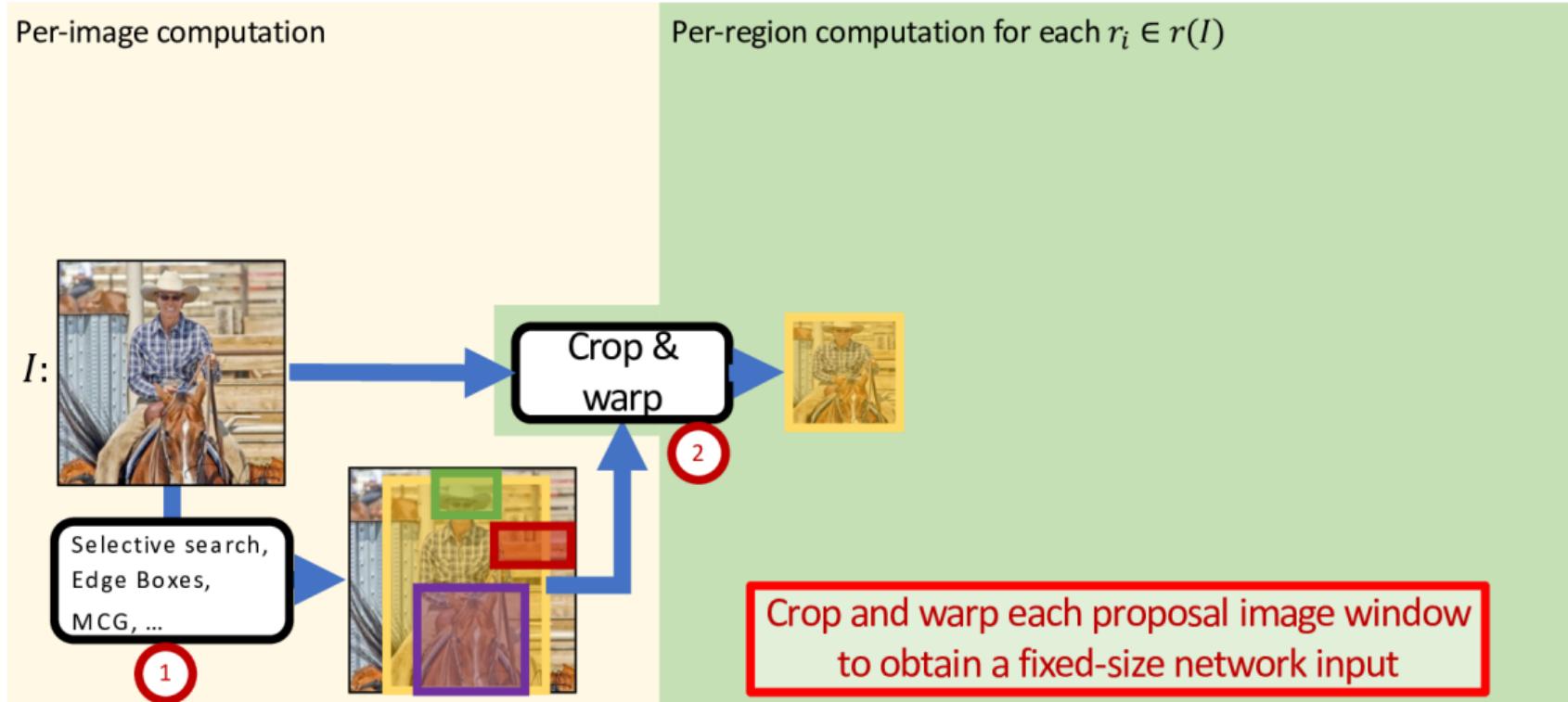


Selective search,  
Edge Boxes,  
MCG, ...

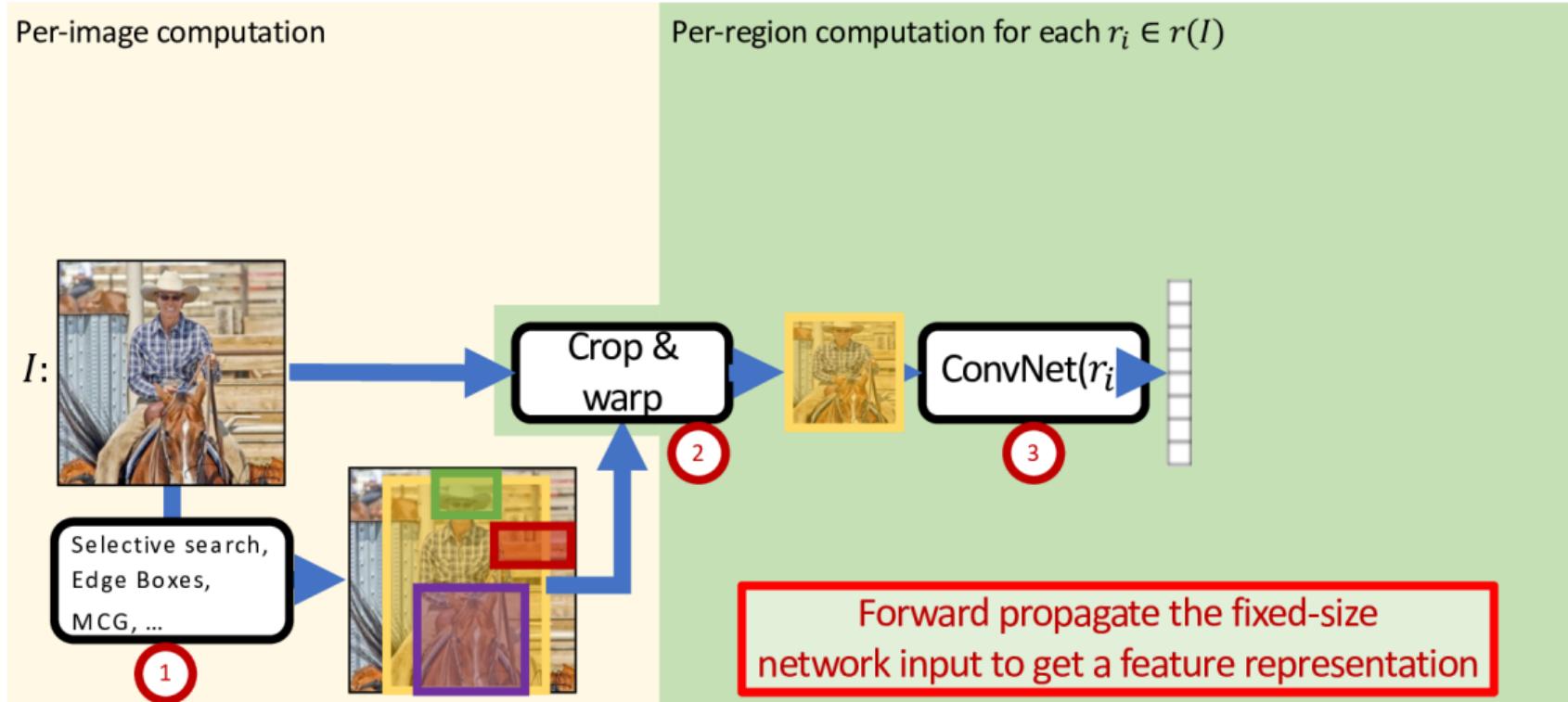
1

Use an off-the-shelf region/object/detection  
proposal algorithm (~2k proposals per image)

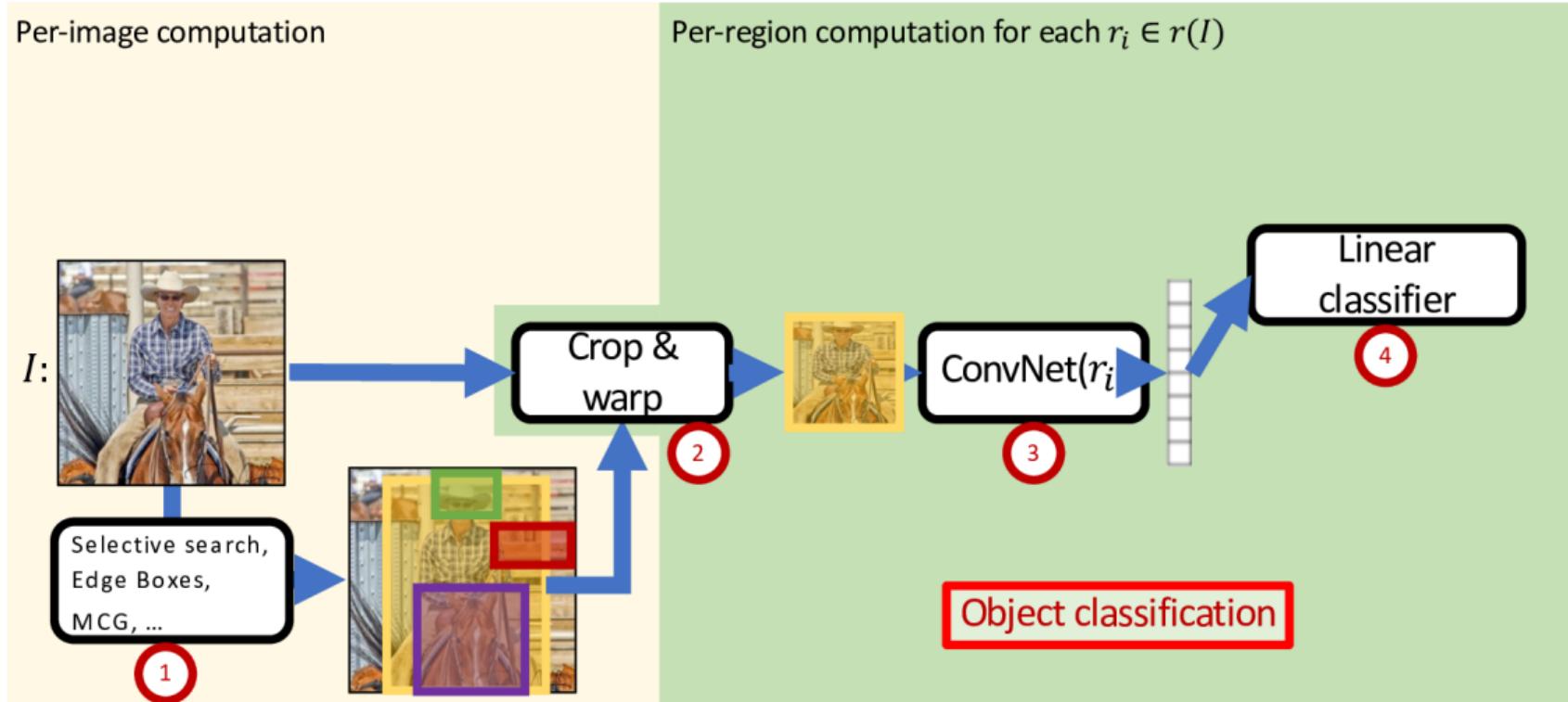
# R-CNN: Region-based Convolutional Neural Network



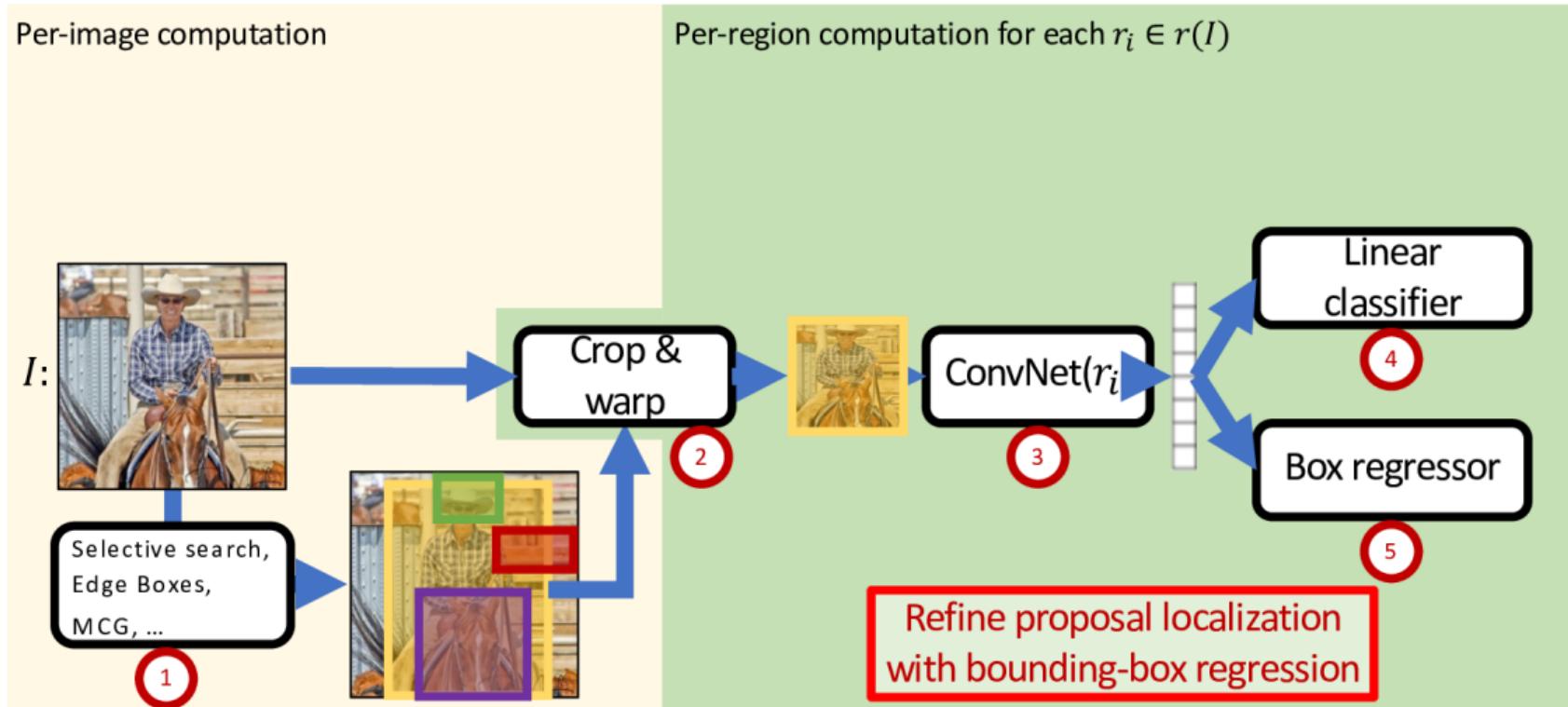
# R-CNN: Region-based Convolutional Neural Network



# R-CNN: Region-based Convolutional Neural Network



# R-CNN: Region-based Convolutional Neural Network



# Generalized Framework

Per-image computation

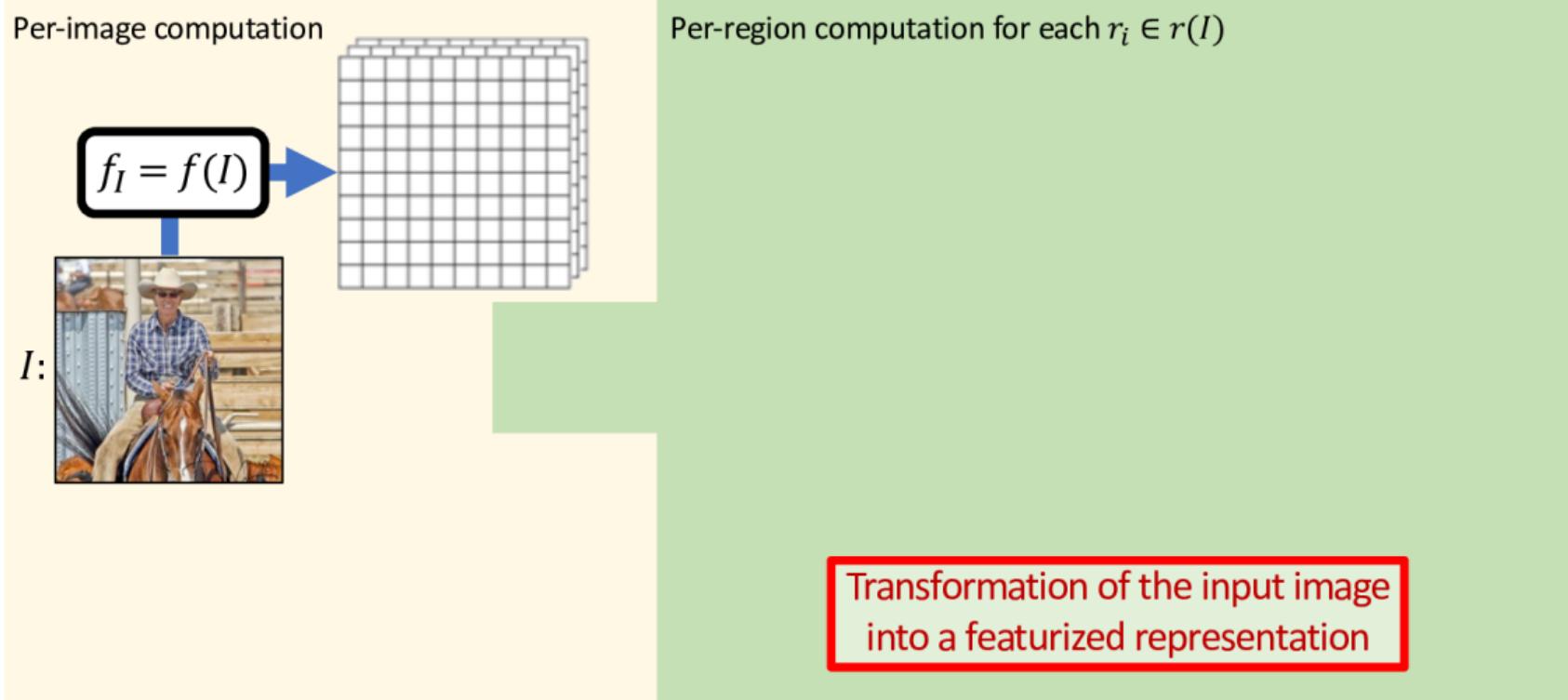


$I$ :

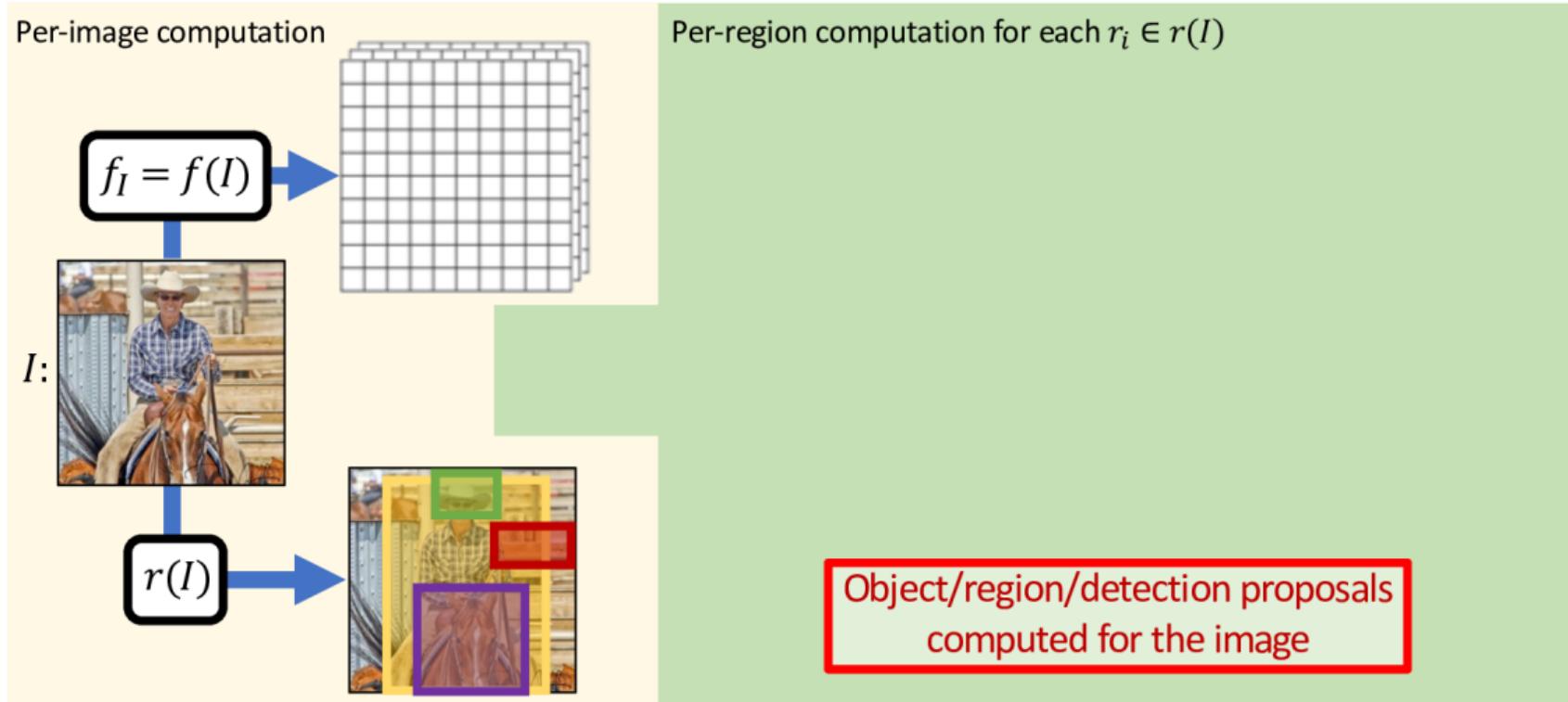
Per-region computation for each  $r_i \in r(I)$

Input image  
per-image operations | per-region operations

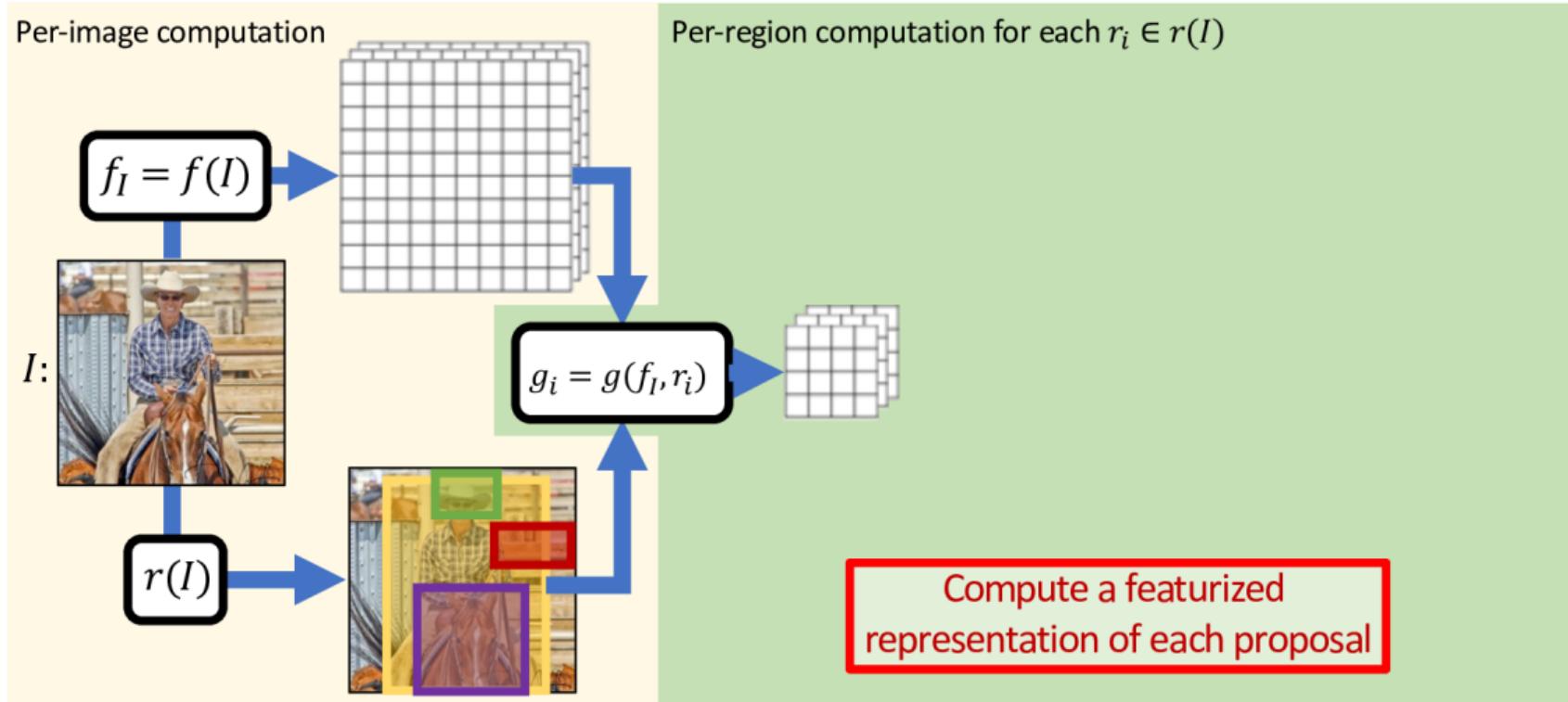
# Generalized Framework



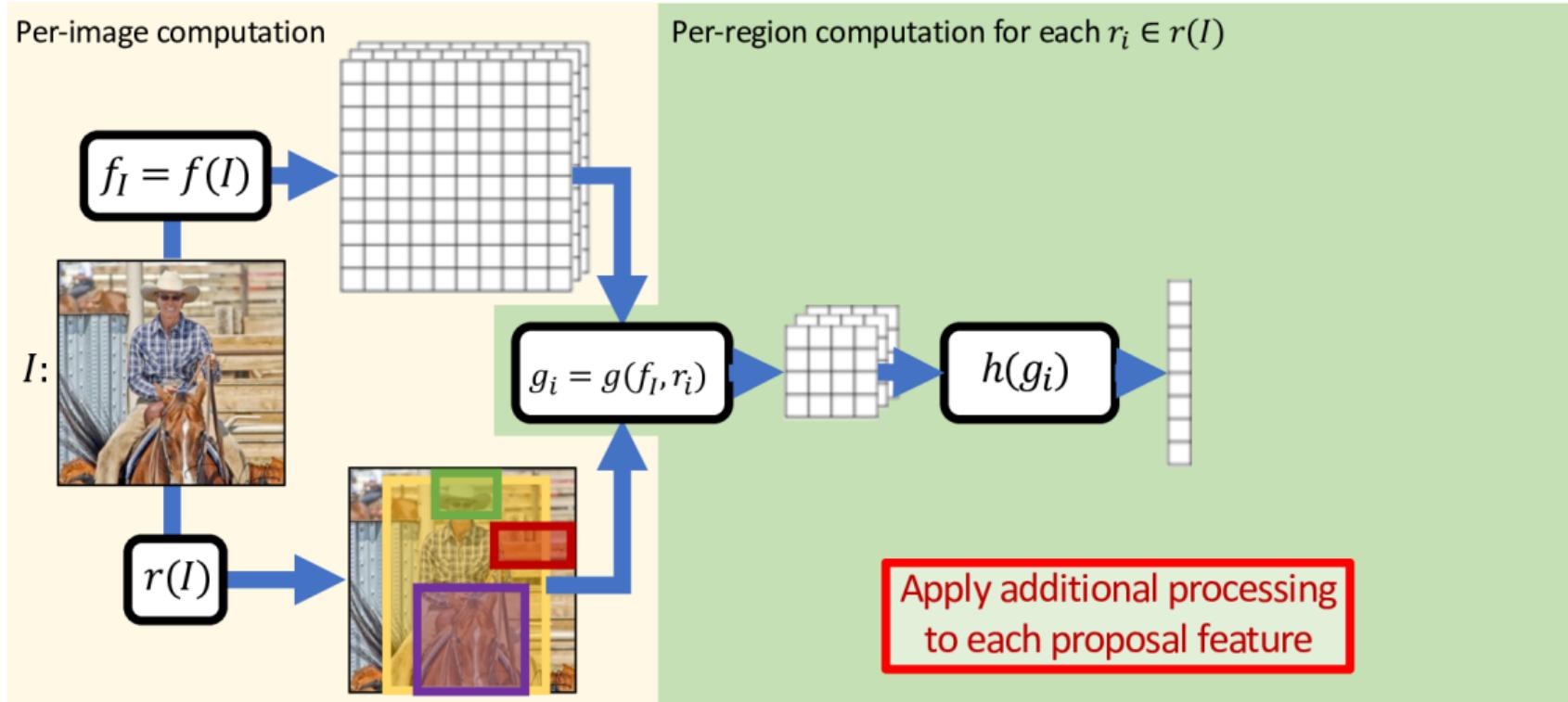
# Generalized Framework



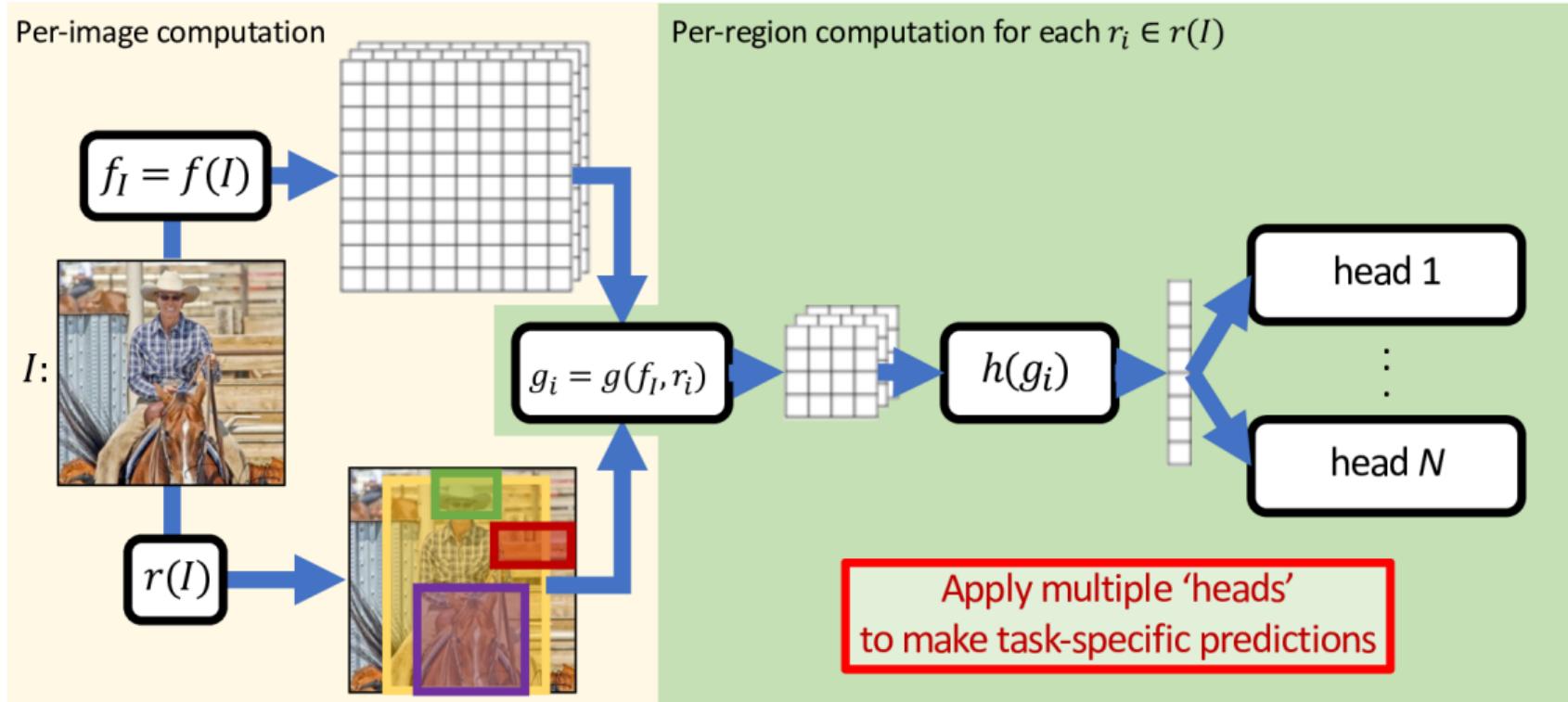
# Generalized Framework



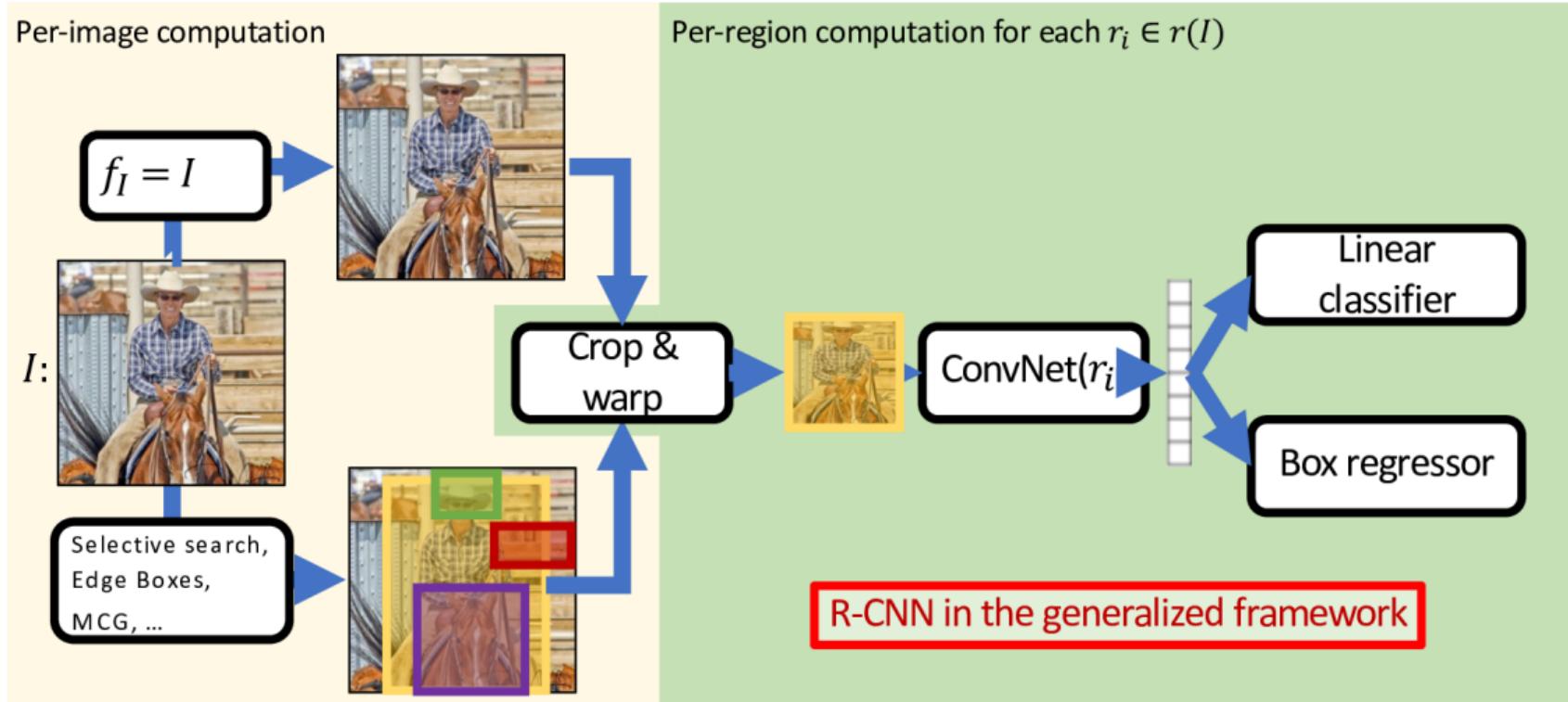
# Generalized Framework



# Generalized Framework



# R-CNN in Generalized Framework

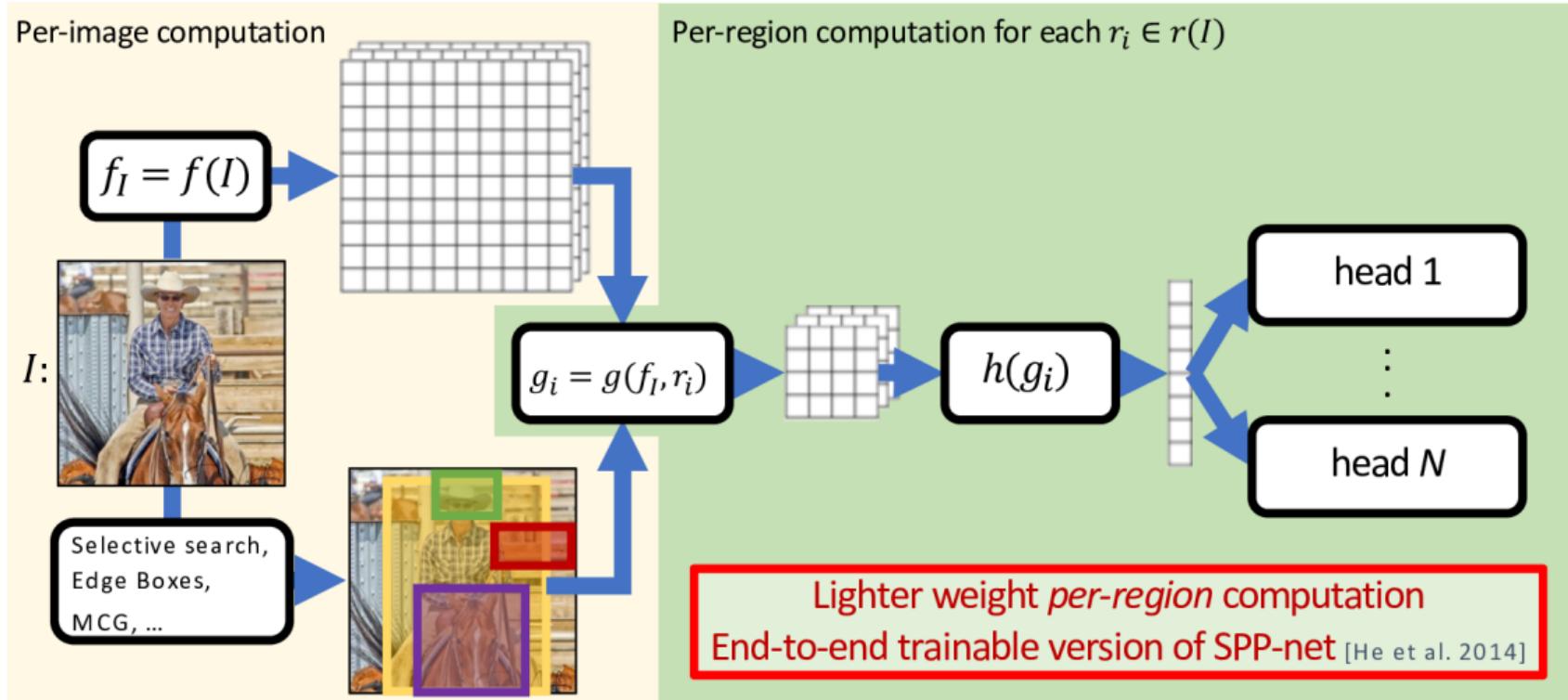


# R-CNN in Generalized Framework

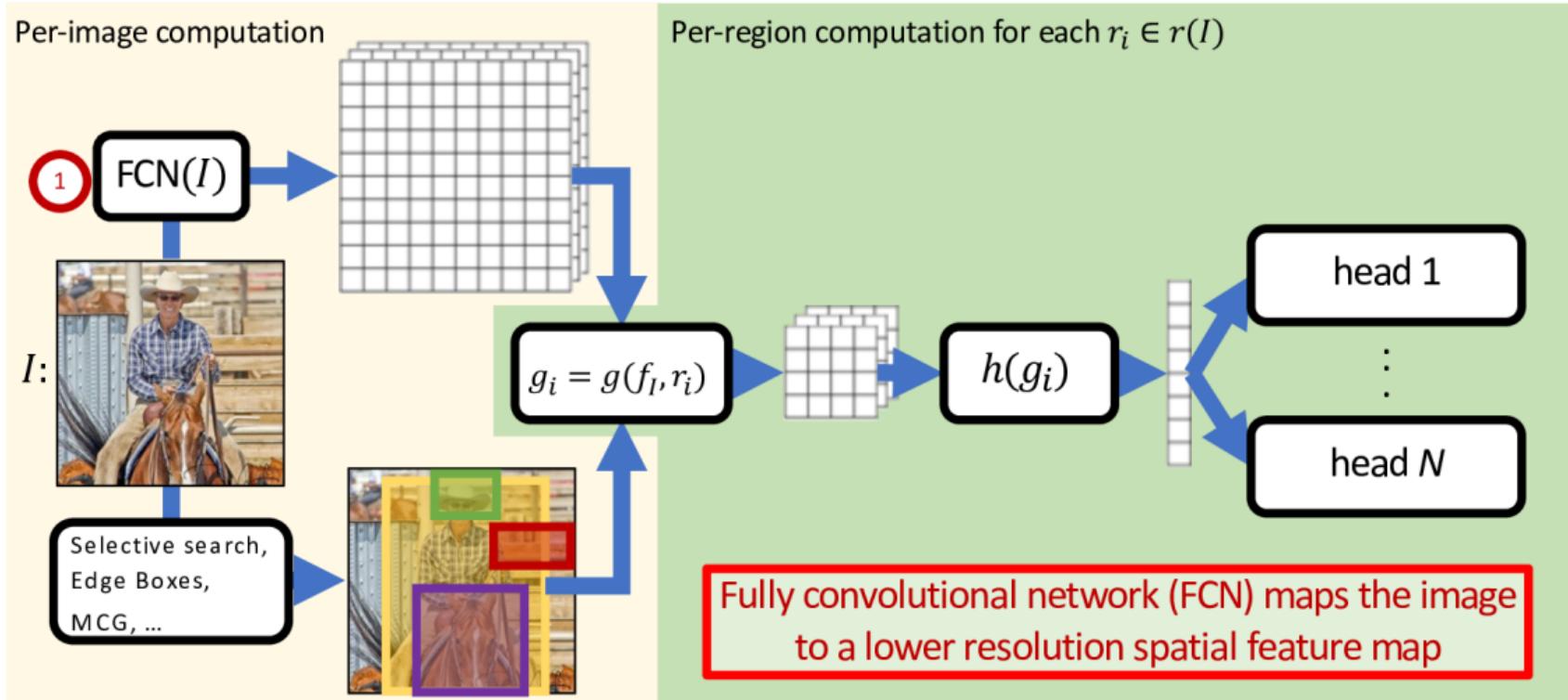
## **What is the problem with R-CNN?**

- ▶ Heavy per-region computation (e.g., 2000 full network evaluations)
- ▶ No computation/feature sharing
- ▶ Slow region proposal method adds to runtime
- ▶ Generic region proposal techniques have limited recall

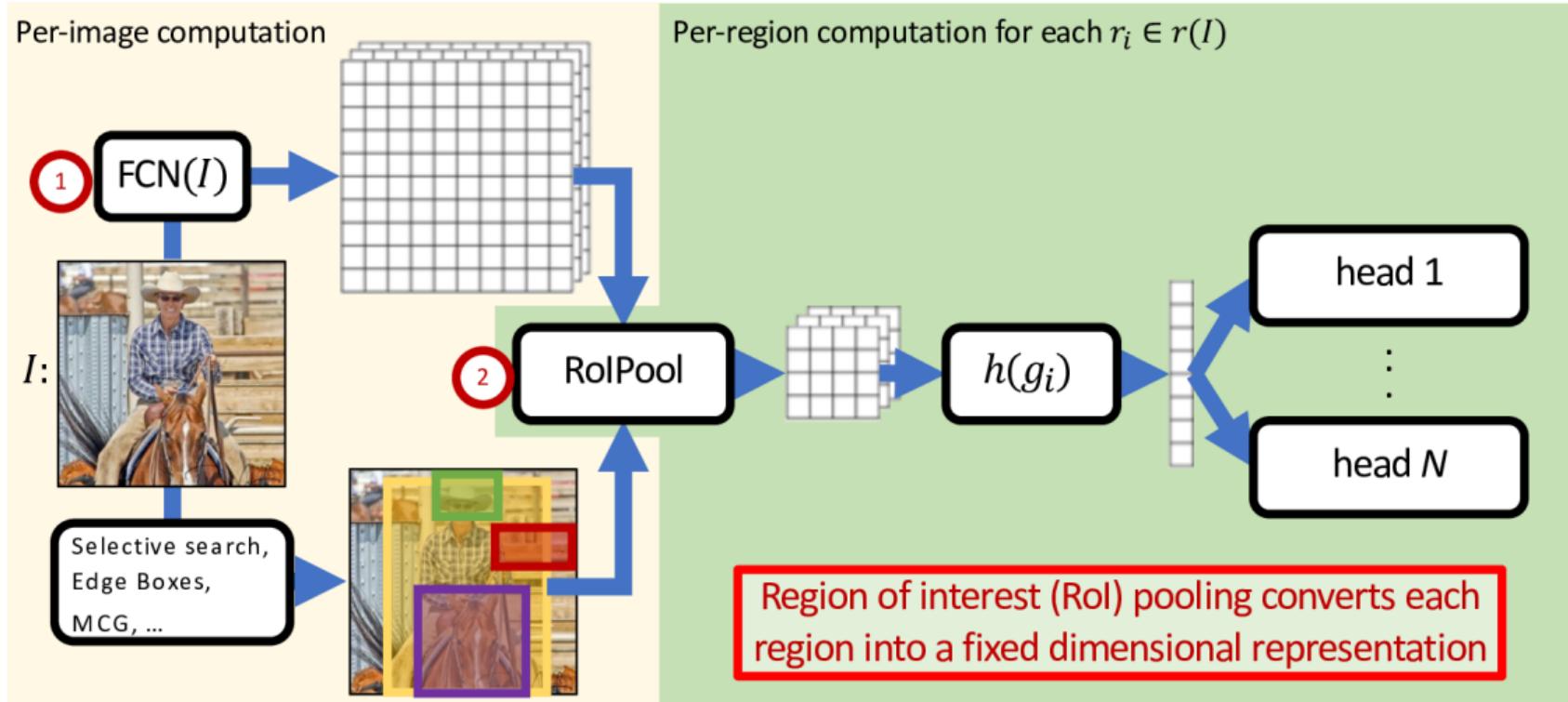
# Fast R-CNN



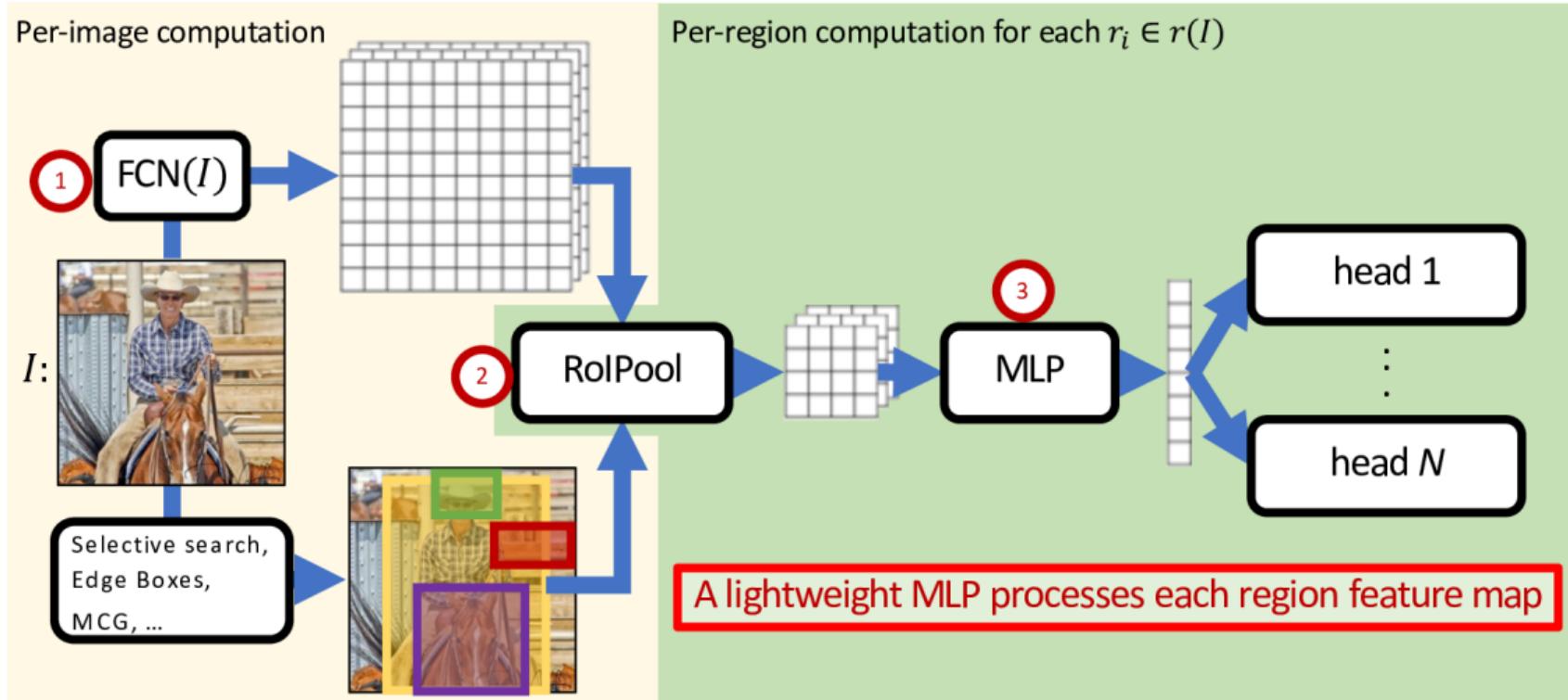
# Fast R-CNN



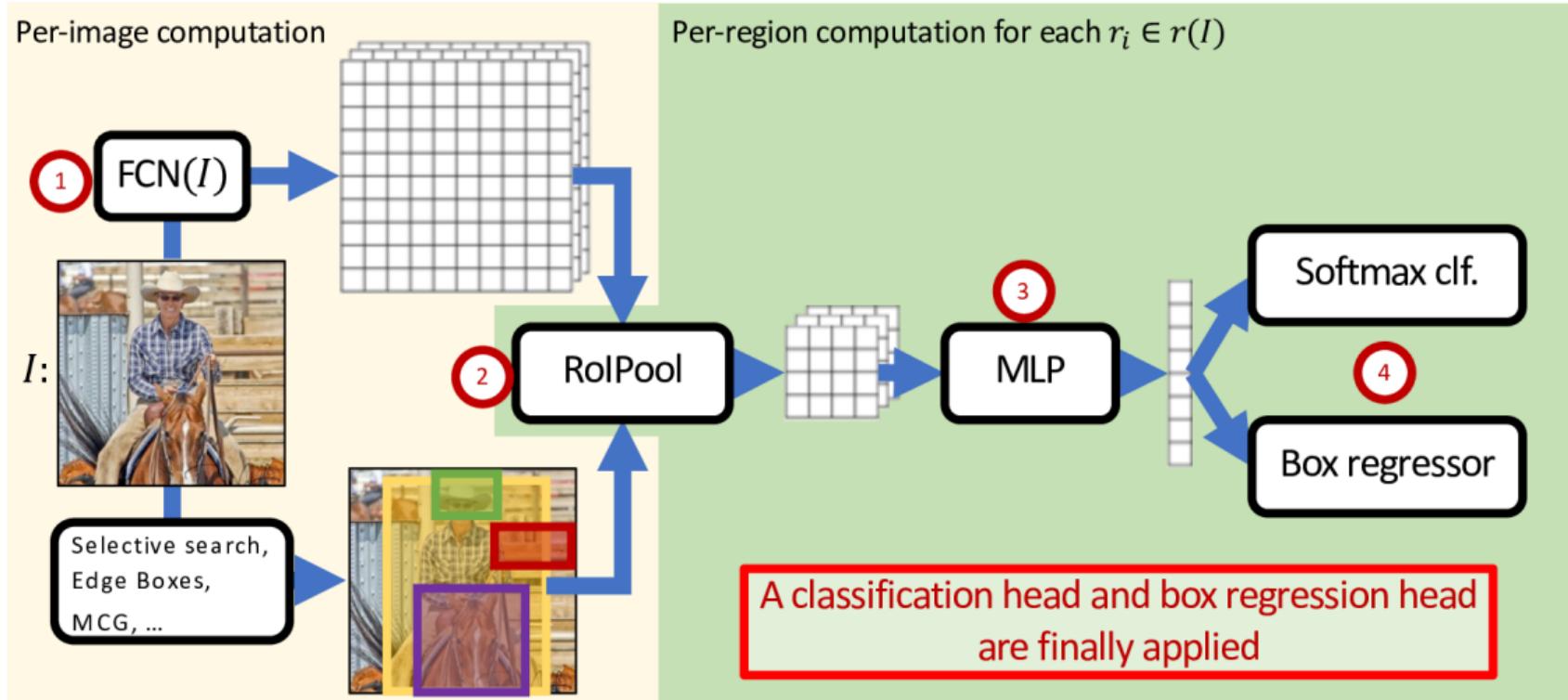
# Fast R-CNN



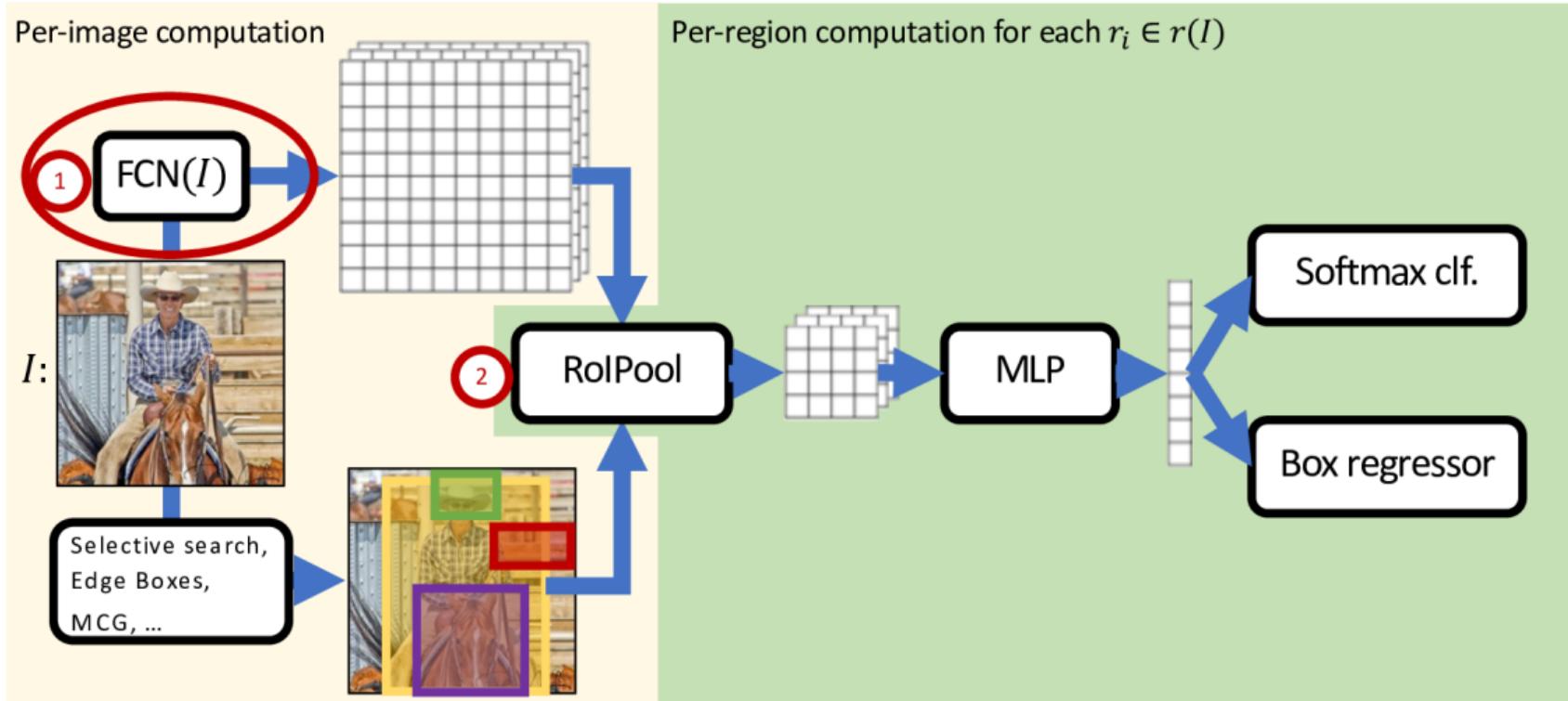
# Fast R-CNN



# Fast R-CNN



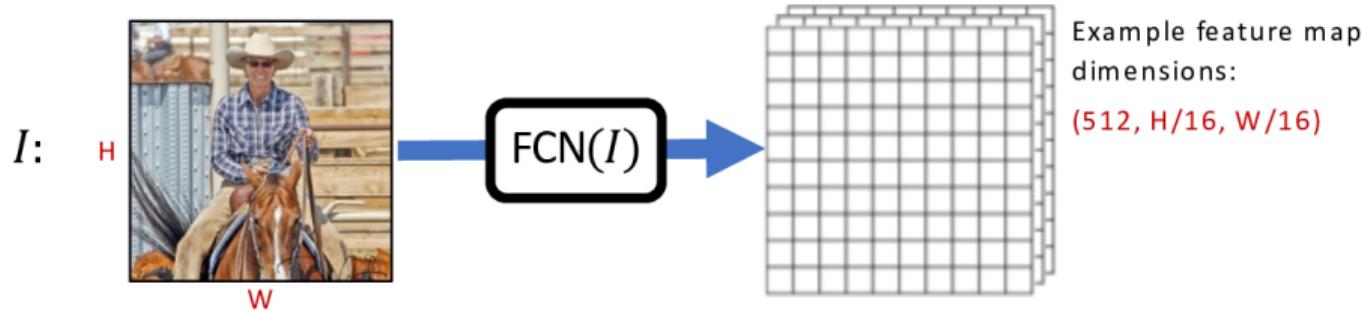
# Fast R-CNN



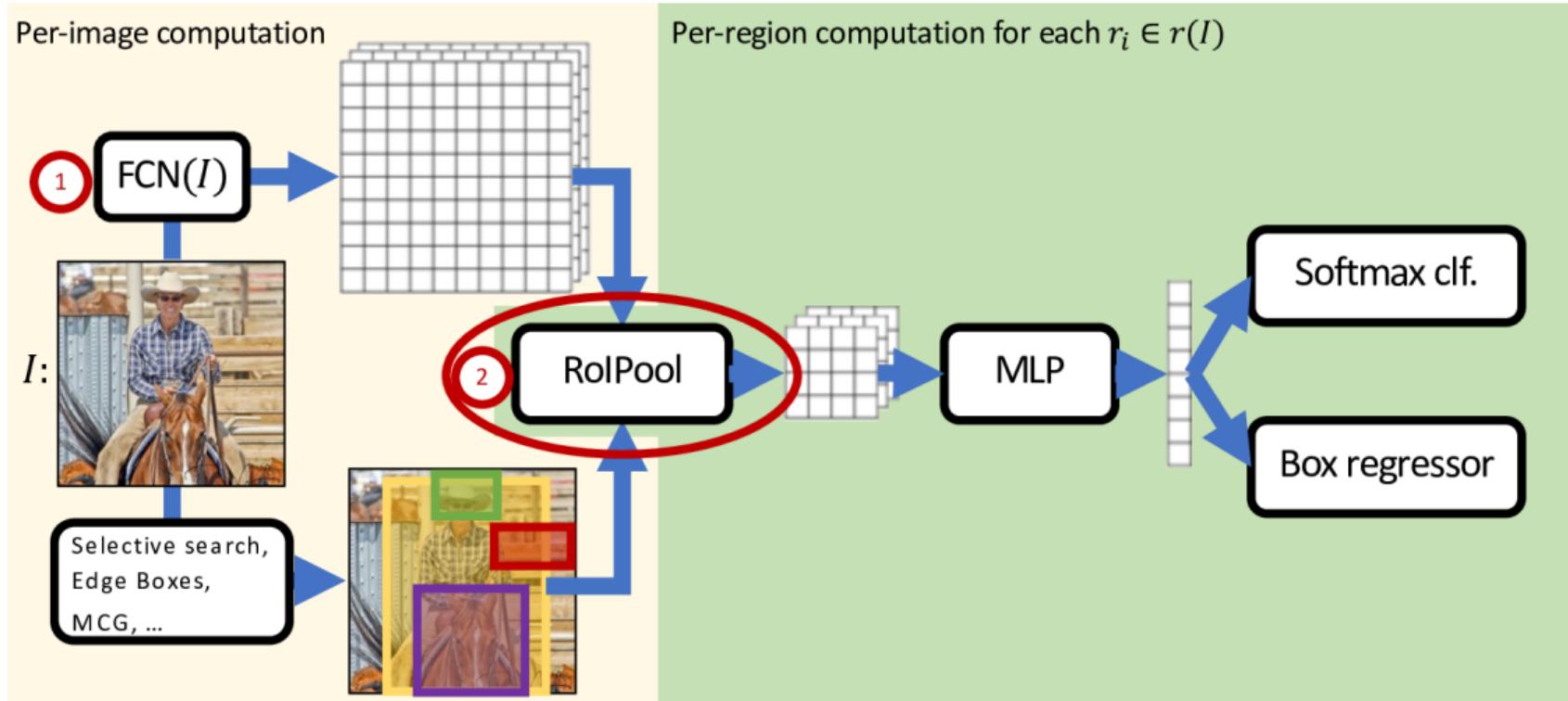
# Fast R-CNN

## Backbone:

- ▶ Use any standard convolutional network as “backbone” architecture
  - ▶ E.g.: AlexNet, VGG, ResNet, Inception, ResNeXt, DenseNet, ...
- ▶ **Remove global pooling** ⇒ output spatial dims proportional to input spatial dims
- ▶ A good network exploits the strongest recognition backbone (features matter!)

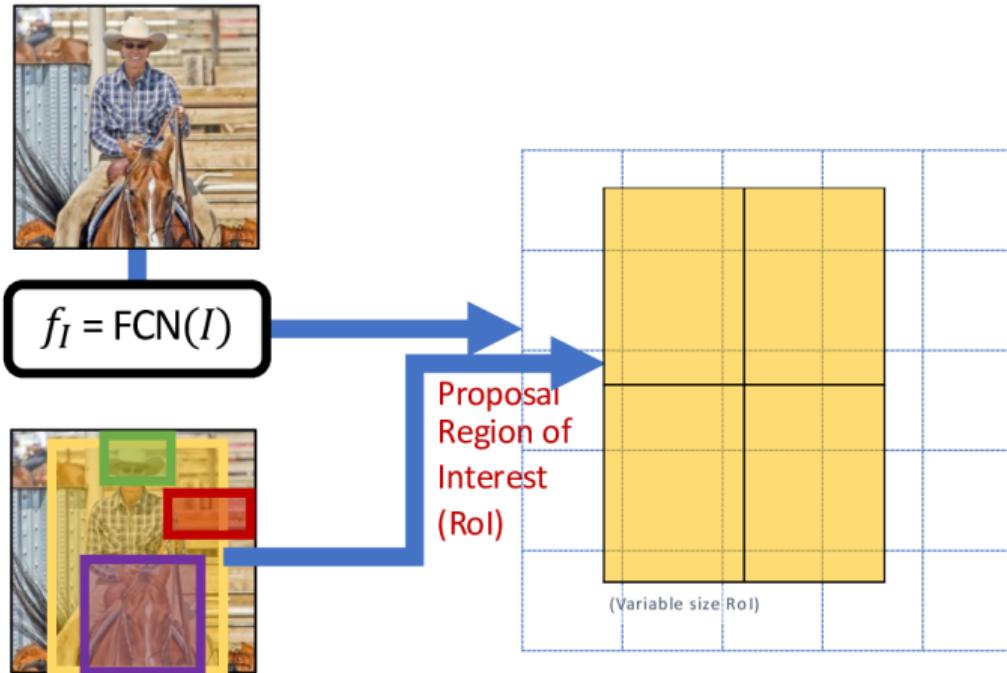


# Fast R-CNN



# Fast R-CNN

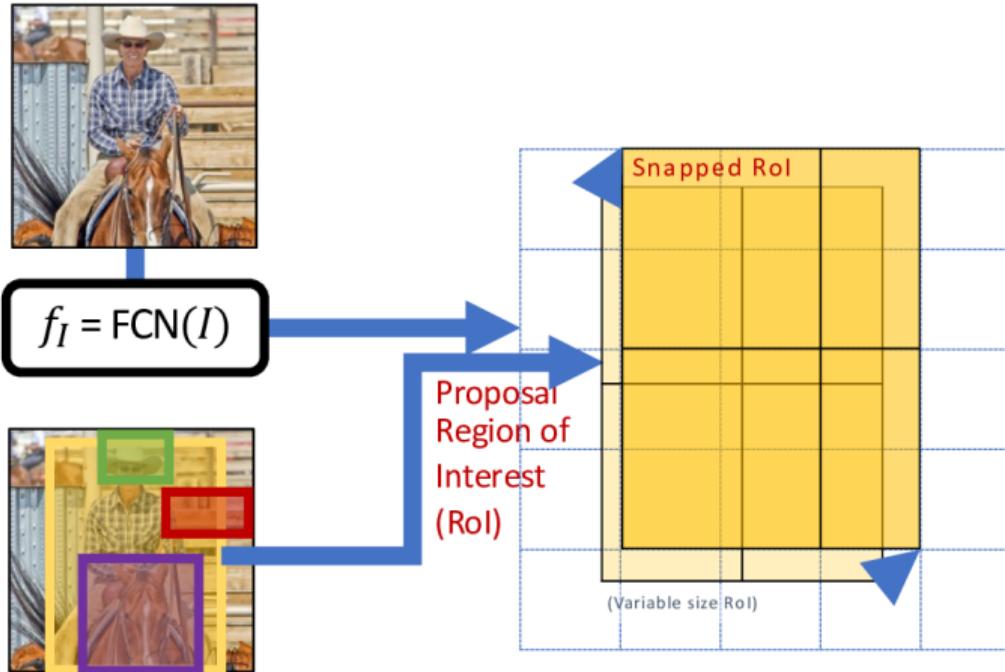
**Region-of-Interest (RoI) Pooling on each Proposal:**



Key innovation in SPP-net  
[He et al. 2014]

# Fast R-CNN

**Region-of-Interest (RoI) Pooling on each Proposal:**



Key innovation in SPP-net  
[He et al. 2014]

# Fast R-CNN

## Region-of-Interest (RoI) Pooling on each Proposal:

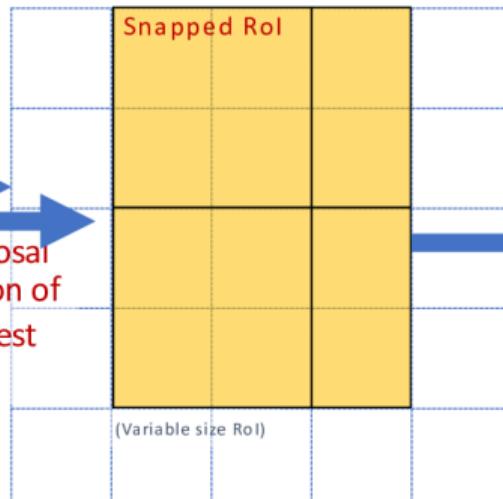


Transform arbitrary size proposal into a fixed-dimensional representation (e.g., 2x2)

$$f_I = \text{FCN}(I)$$



Proposal  
Region of  
Interest  
(RoI)



RoIPool  
transform

(Fixed dimensional  
representation)

Feature value  
is **max** over input  
cells



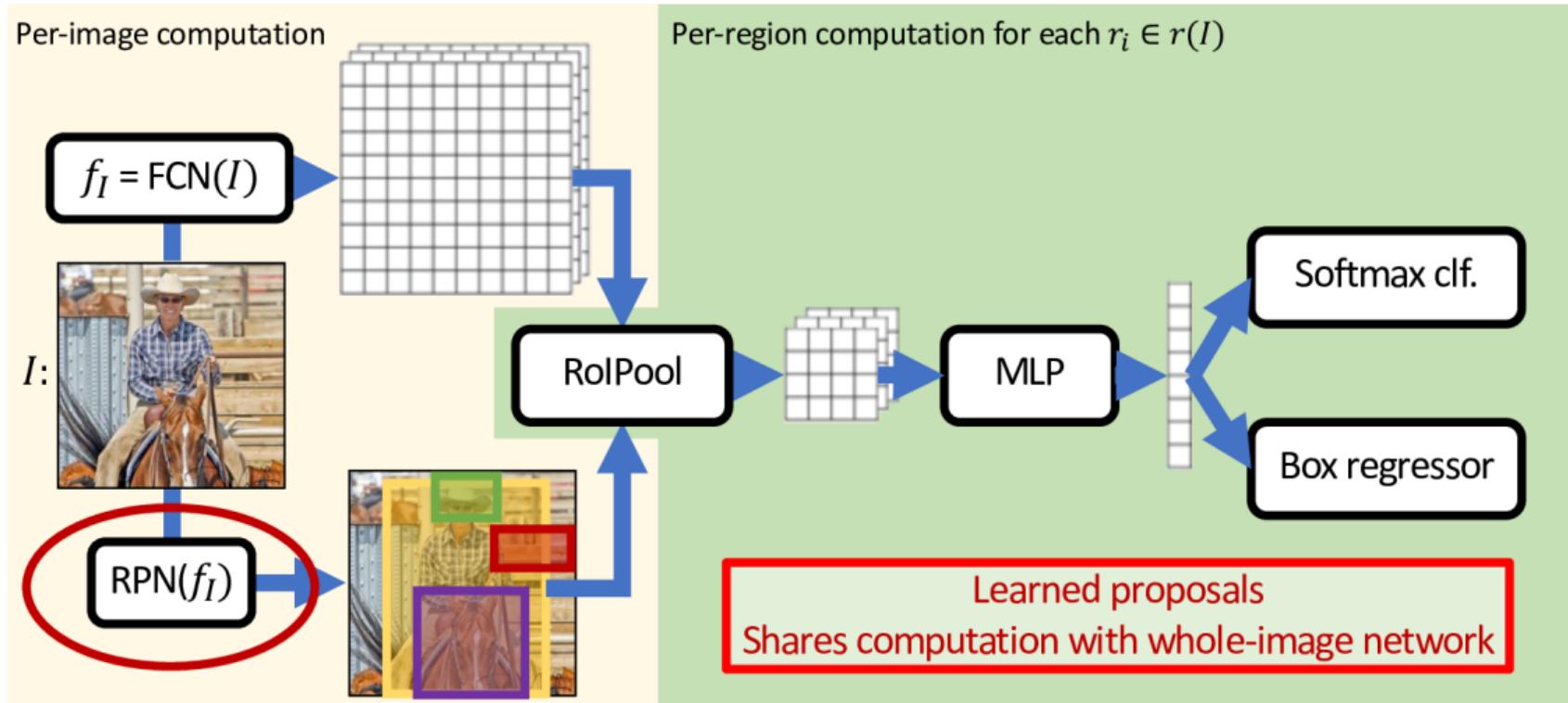
MLP

# Fast R-CNN

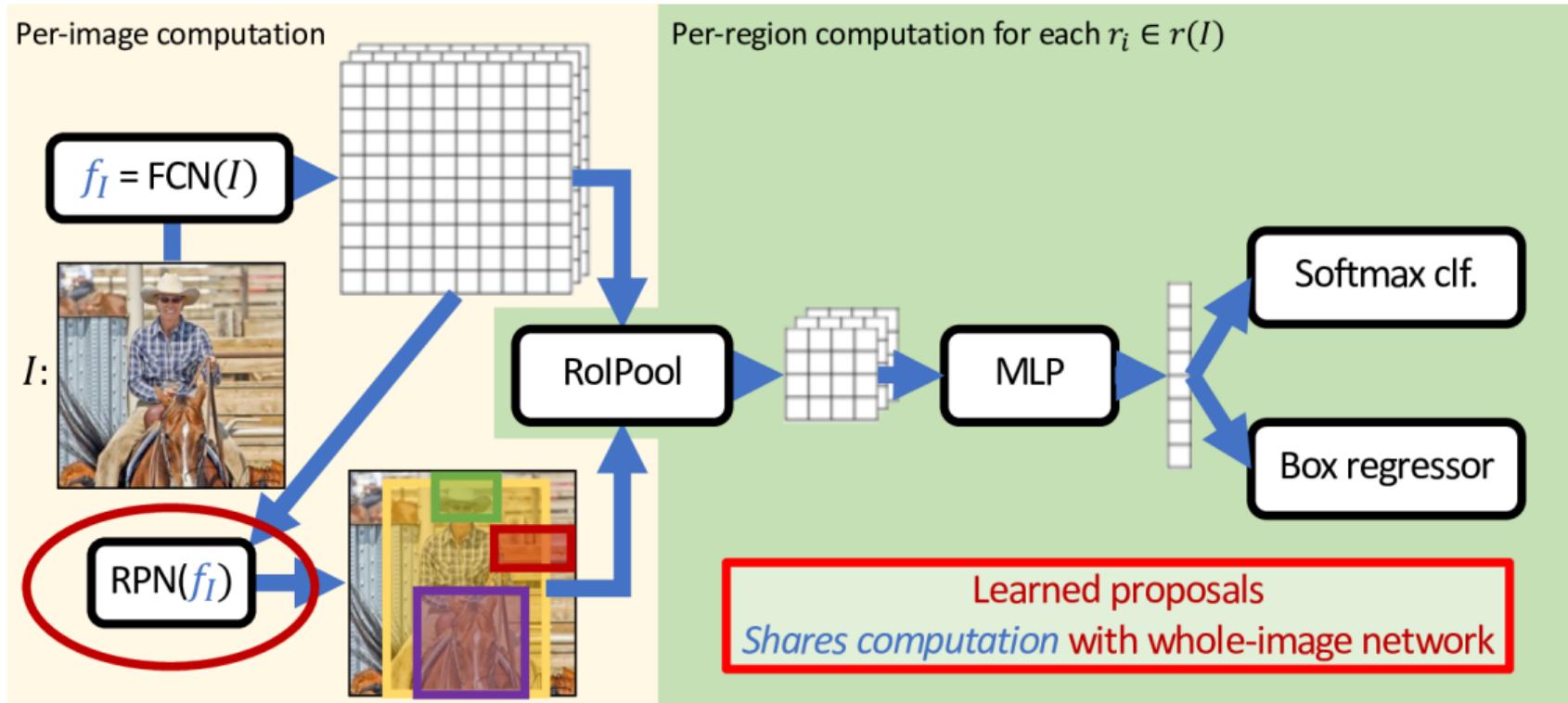
## What is the problem with Fast R-CNN?

- ▶ ~~Heavy per-region computation (e.g., 2000 full network evaluations)~~
- ▶ ~~No computation/feature sharing~~
- ▶ Slow region proposal method adds to runtime
- ▶ Generic region proposal techniques have low recall

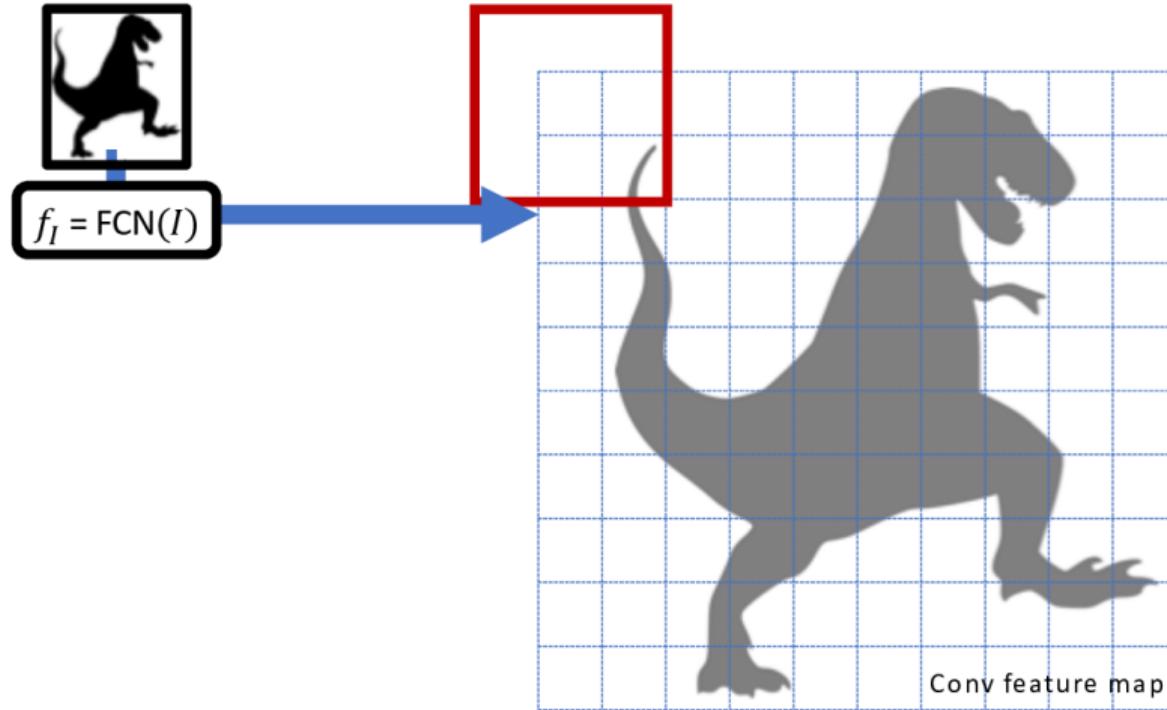
# Faster R-CNN



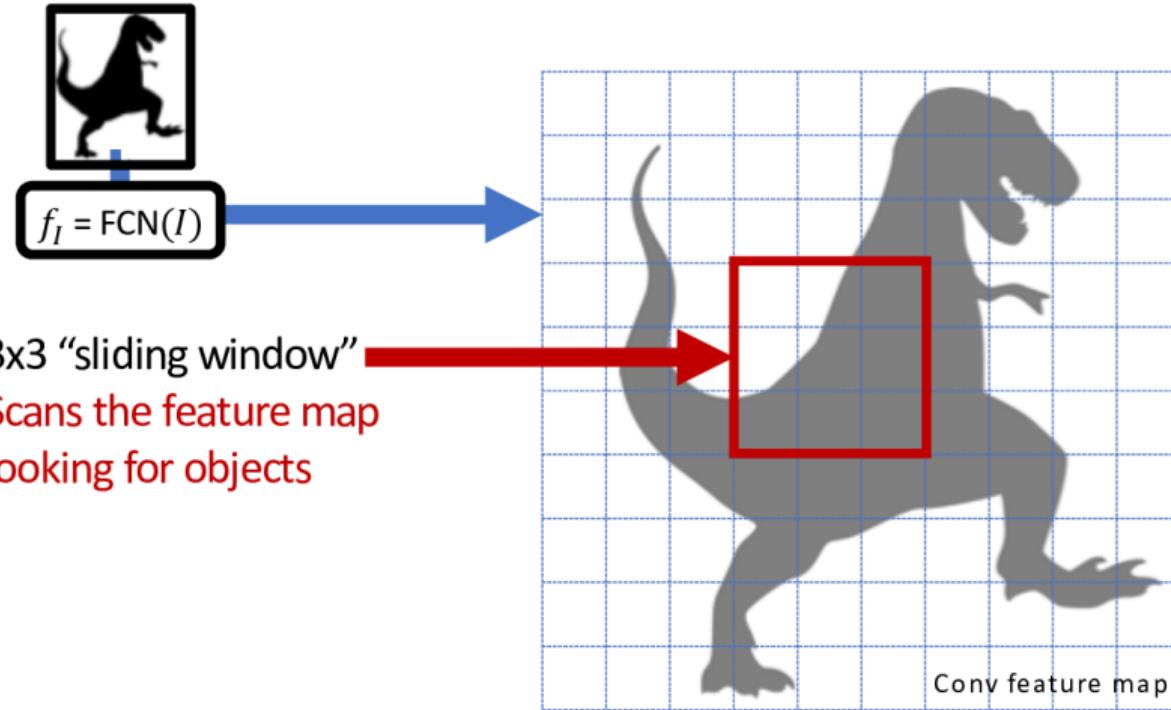
# Faster R-CNN



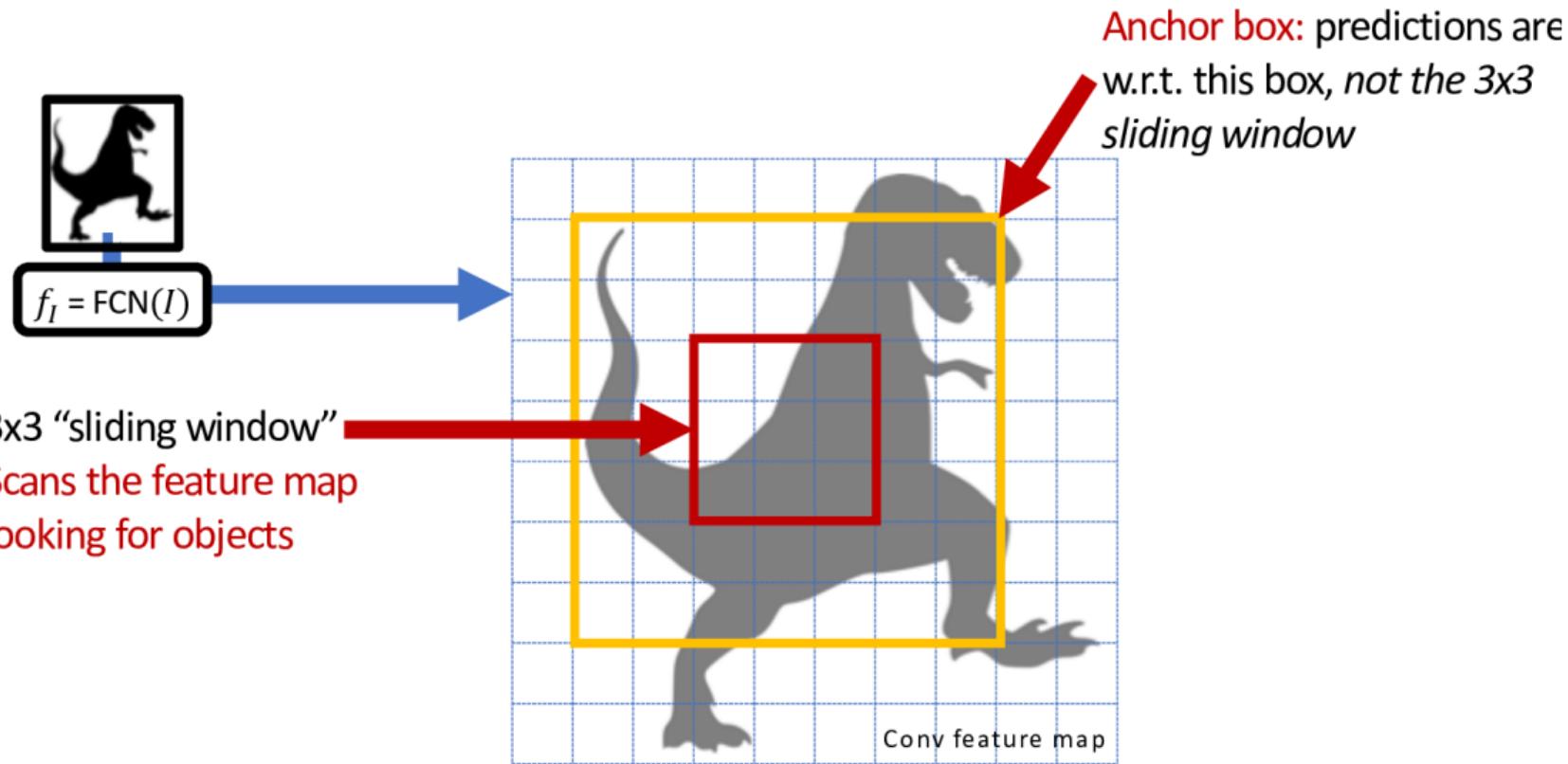
# Faster R-CNN: Region Proposal Network (RPN)



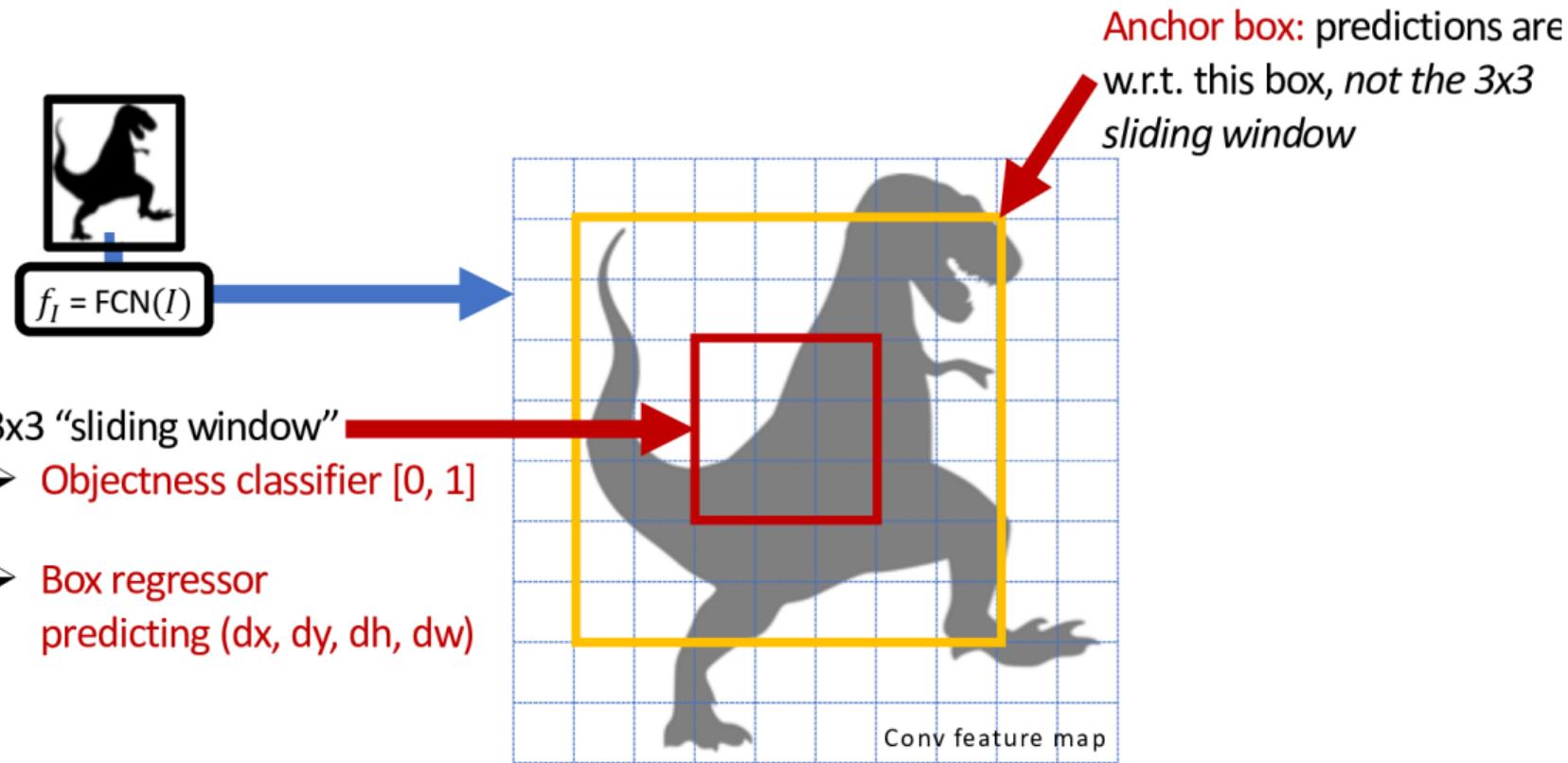
# Faster R-CNN: Region Proposal Network (RPN)



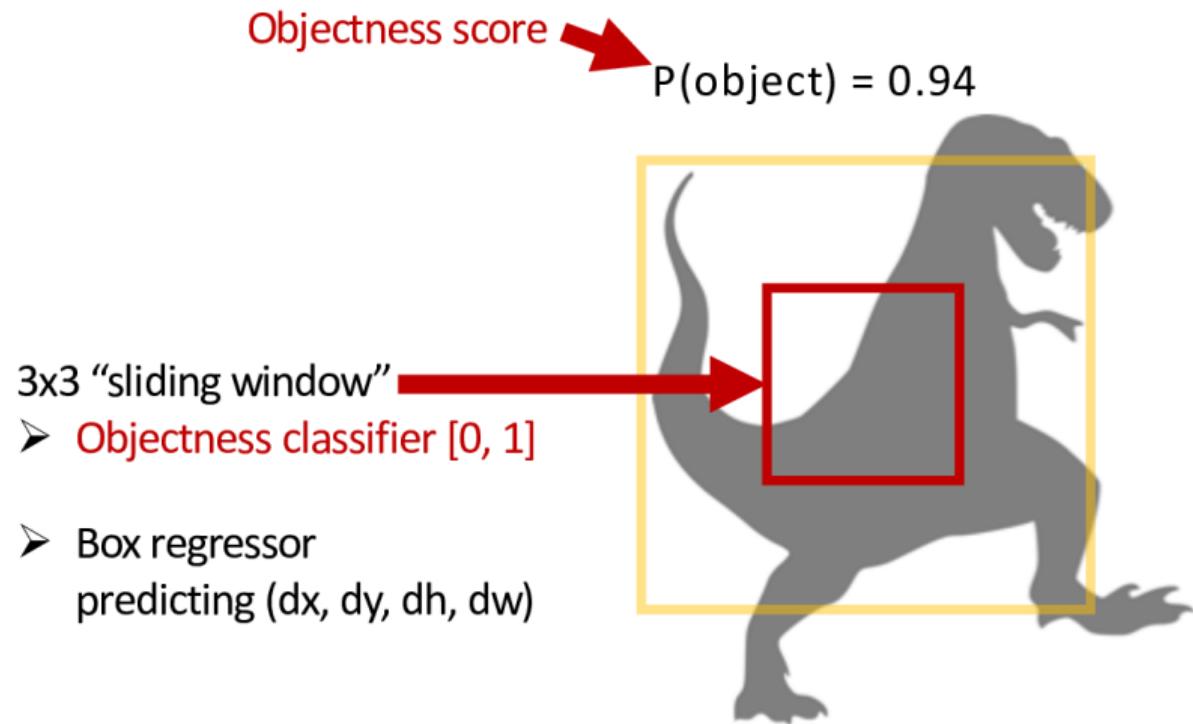
# Faster R-CNN: Region Proposal Network (RPN)



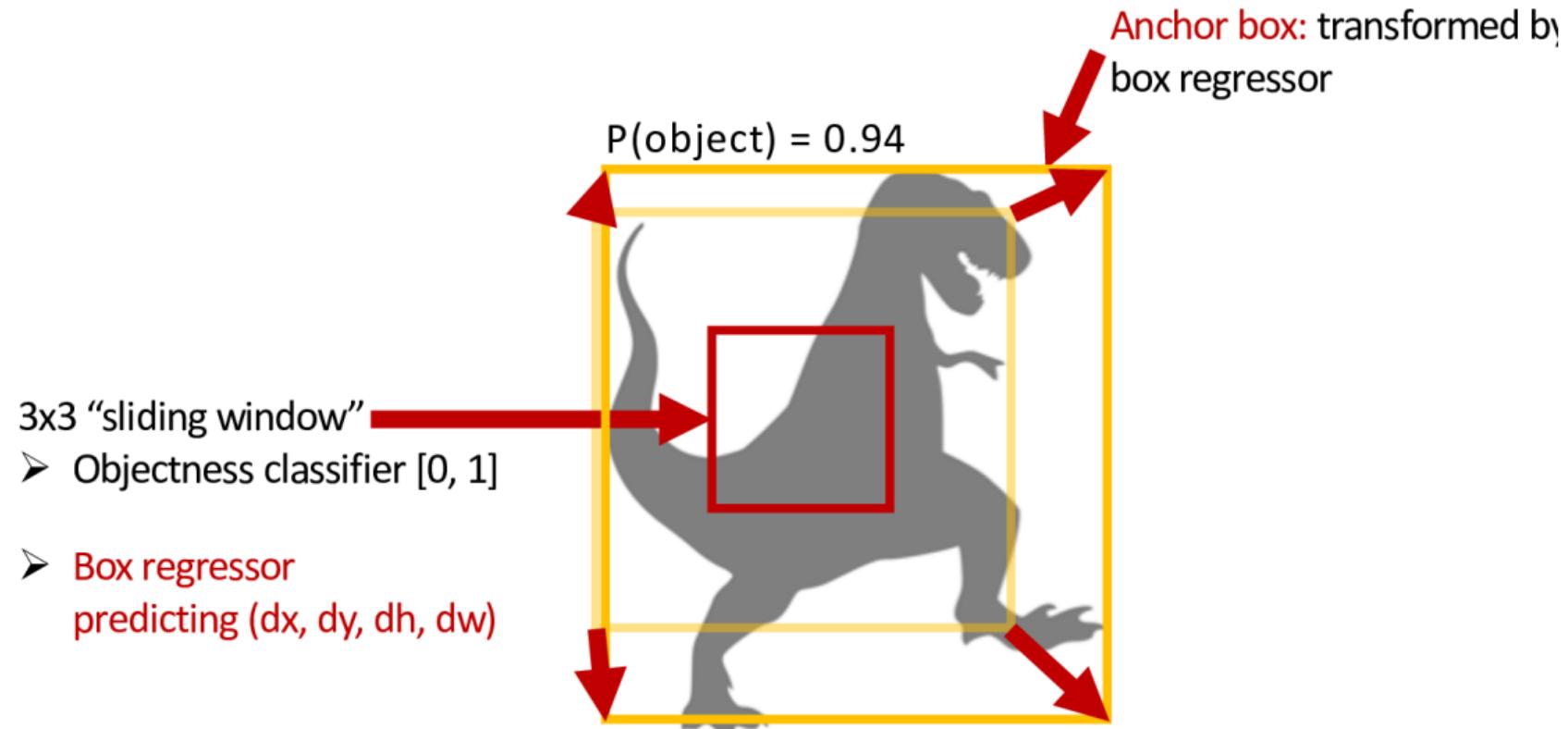
# Faster R-CNN: Region Proposal Network (RPN)



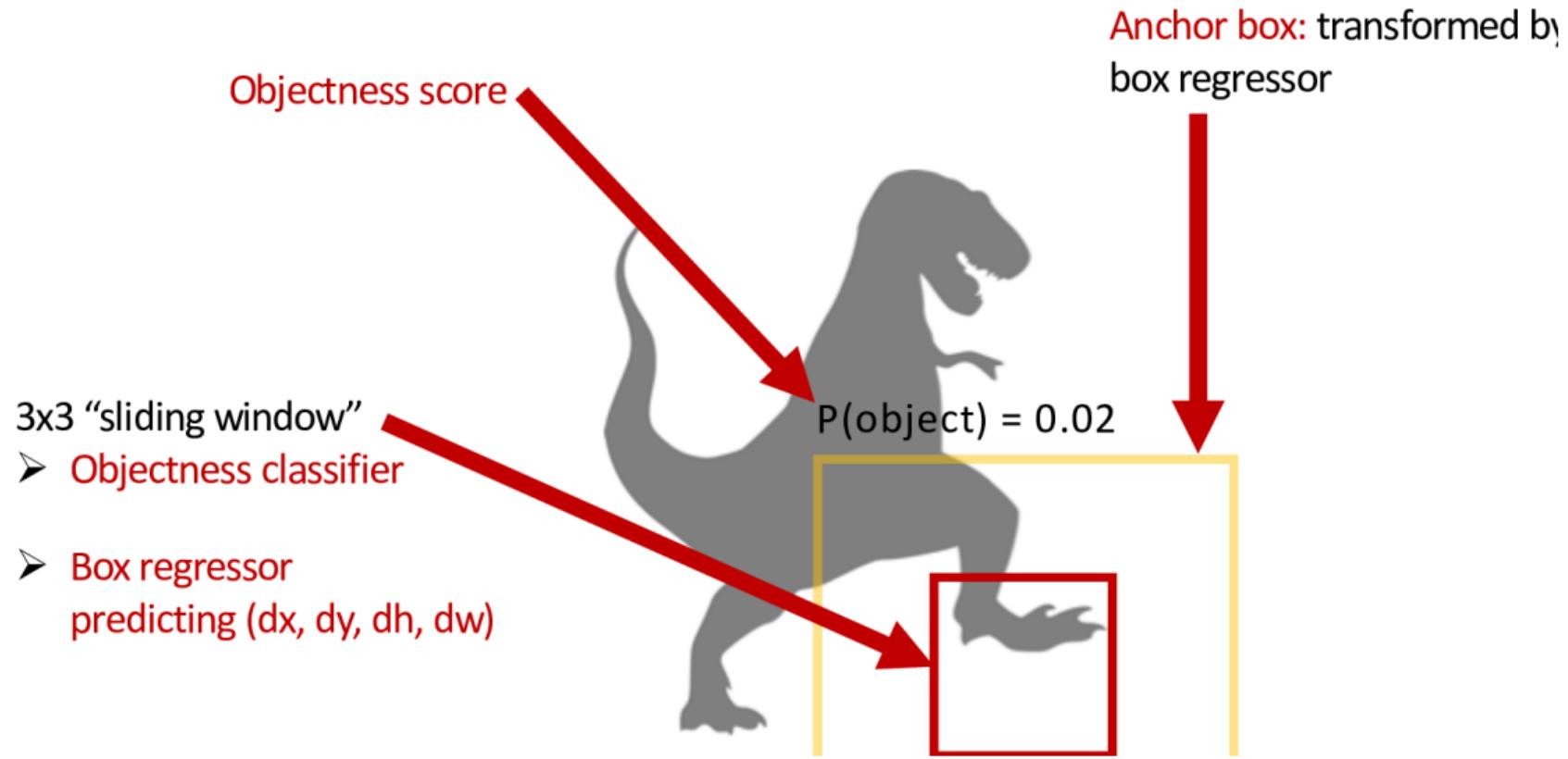
# Faster R-CNN: Region Proposal Network (RPN)



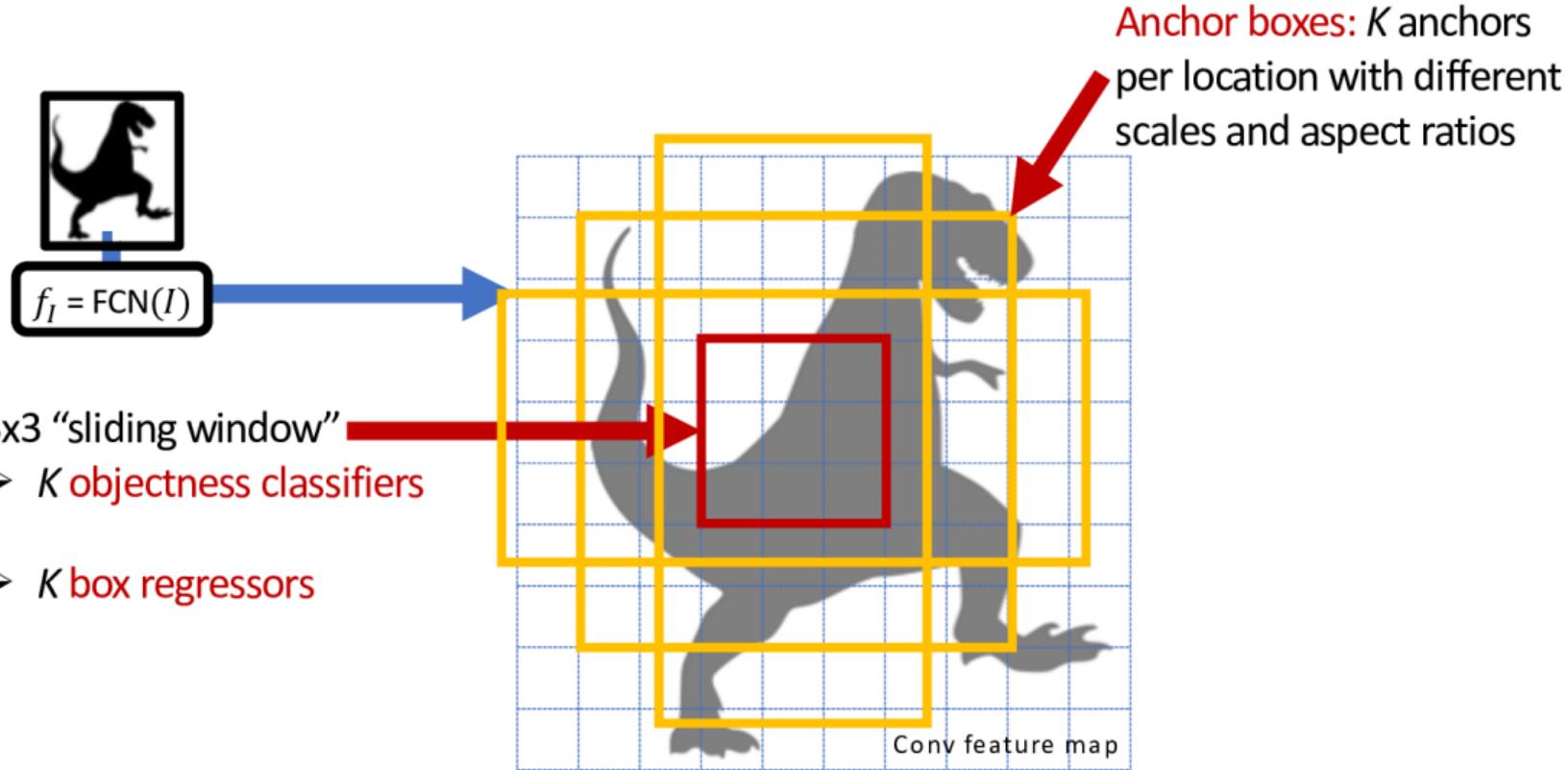
# Faster R-CNN: Region Proposal Network (RPN)



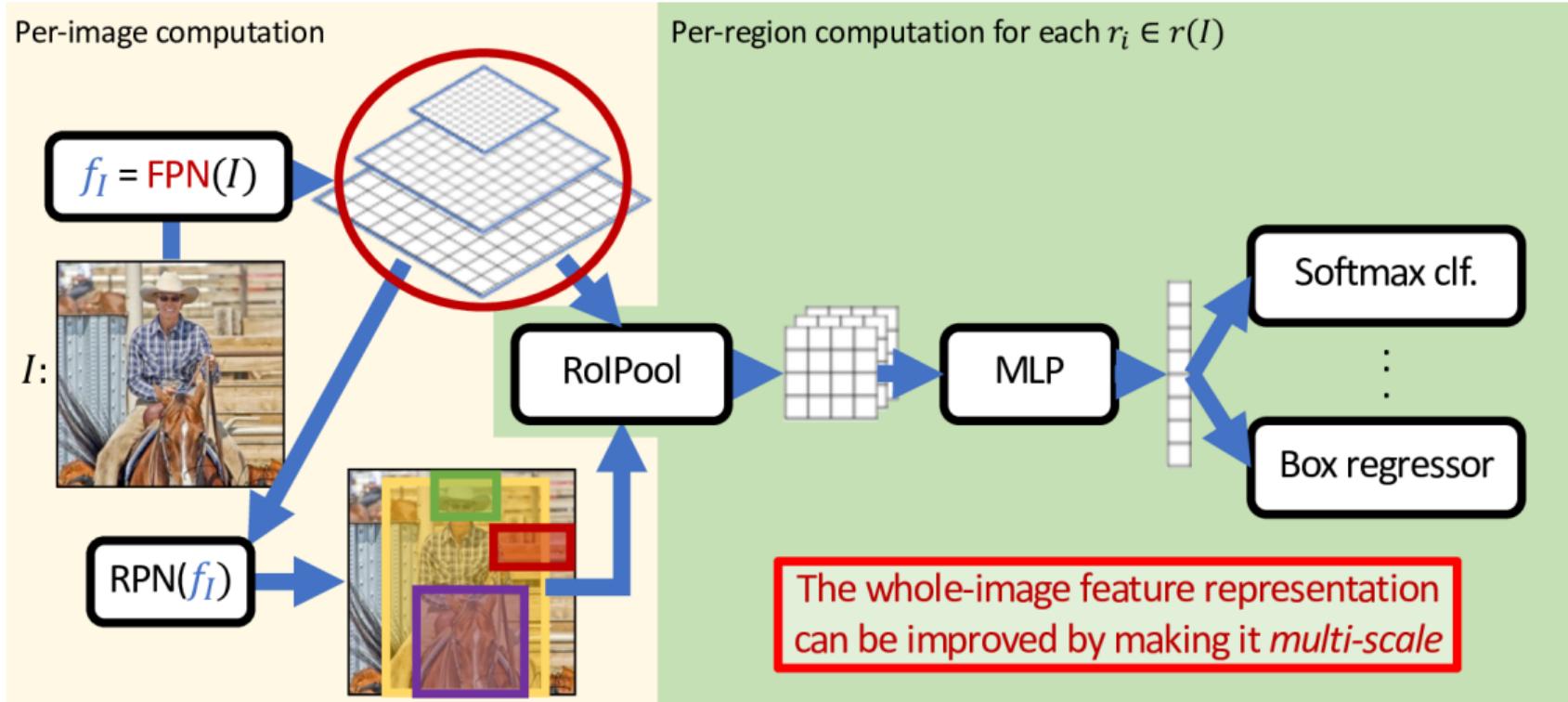
# Faster R-CNN: Region Proposal Network (RPN)



# Faster R-CNN: Region Proposal Network (RPN)

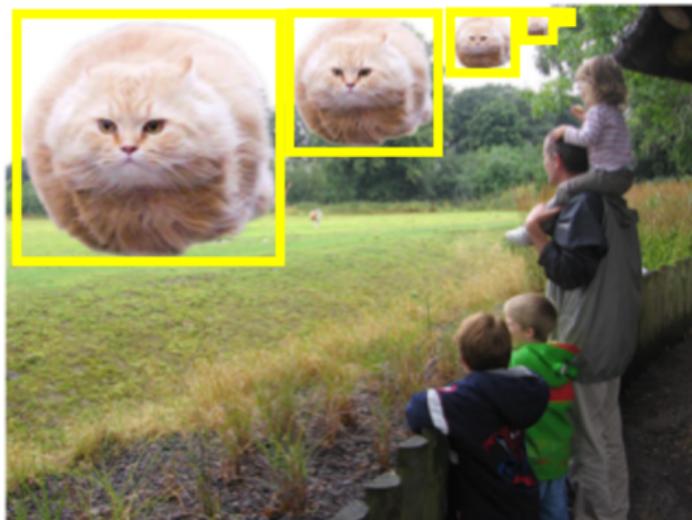


# Feature Pyramid Network



# Feature Pyramid Network

**Goal of Feature Pyramid Network:** Improve Scale Equivariance



Detectors need to

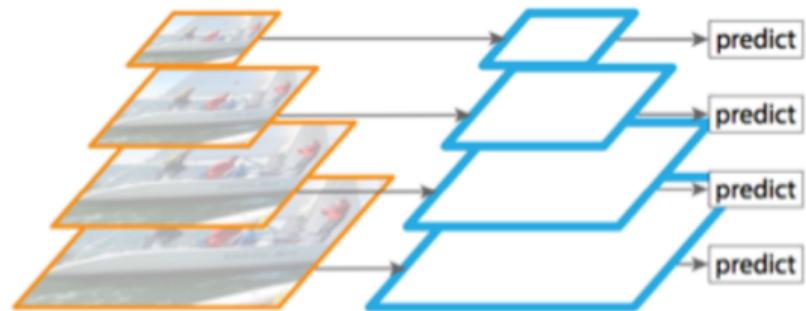
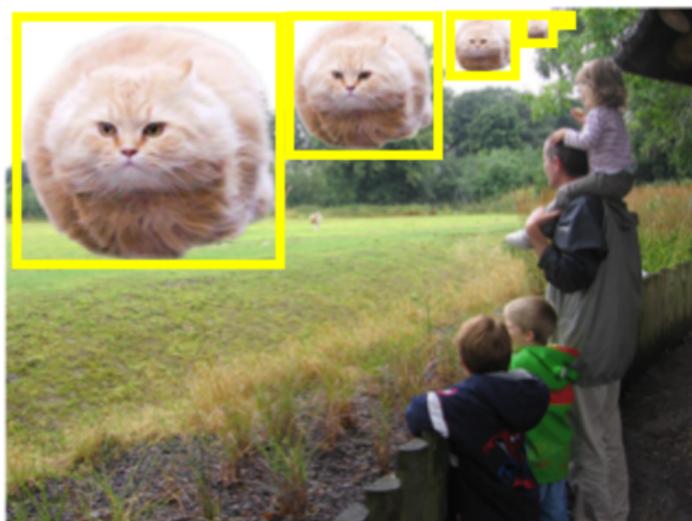
1. classify and
2. localize

objects over a **wide range of scales**

FPN improves this ability

# Feature Pyramid Network

Strategy 1: Image Pyramid



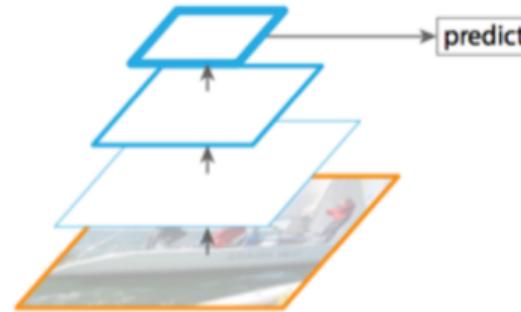
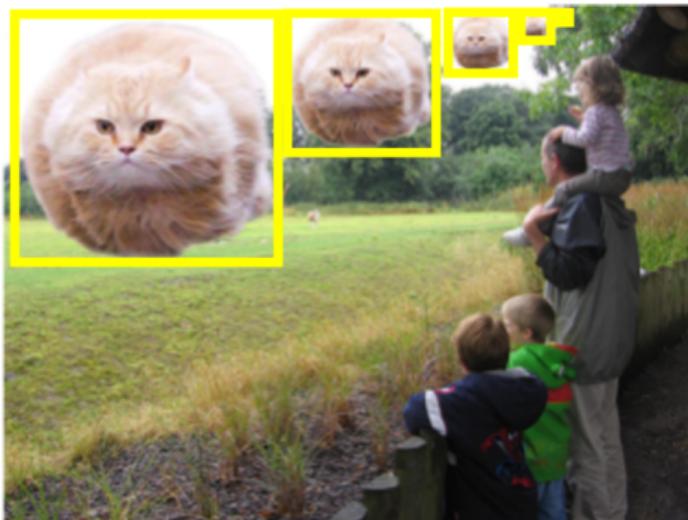
(a) Featurized image pyramid

Standard solution – *slow!*

(E.g., Viola & Jones, HOG, DPM, SPP-net,

# Feature Pyramid Network

Strategy 2: Multi-scale Features (Single-scale Map)

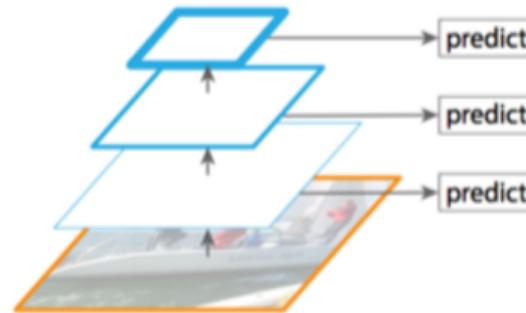
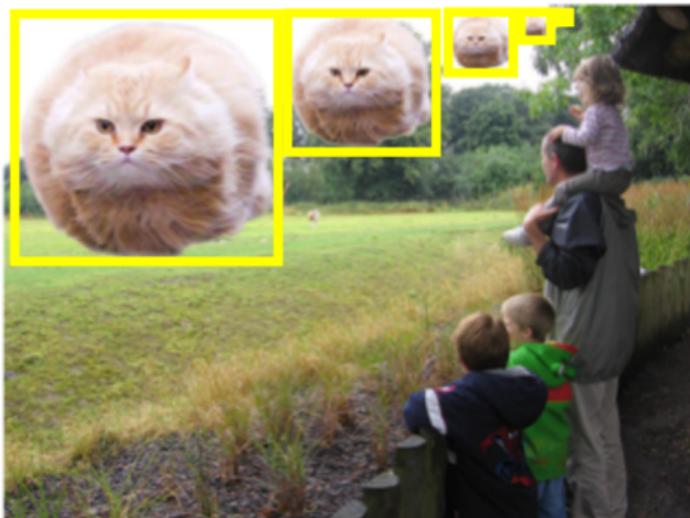


**(b) Single feature map**

**Leave it all to the features – *fast, suboptimal***  
(E.g., Fast/er R-CNN, YOLO, ...)

# Feature Pyramid Network

## Strategy 3: Naïve In-network Pyramid

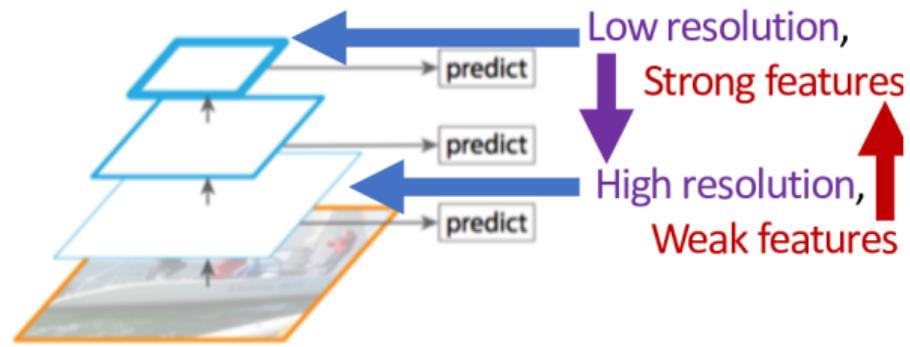
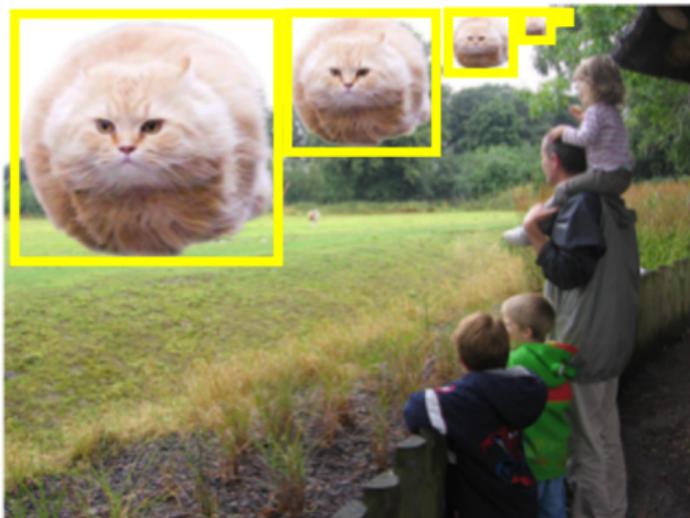


(c) Pyramidal feature hierarchy

Use the internal pyramid – *fast, suboptimal*  
(E.g.,  $\approx$  SSD, ...)

# Feature Pyramid Network

## Strategy 3: Naïve In-network Pyramid

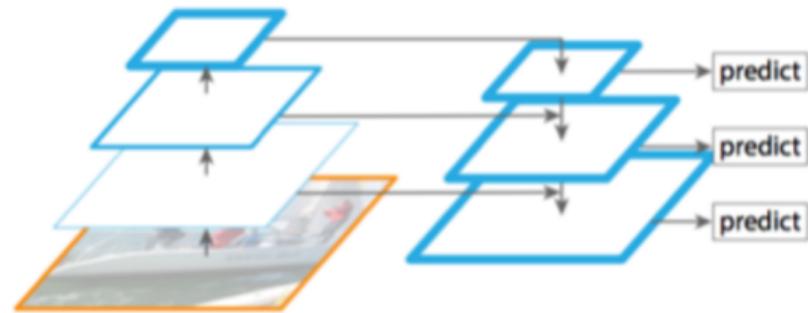
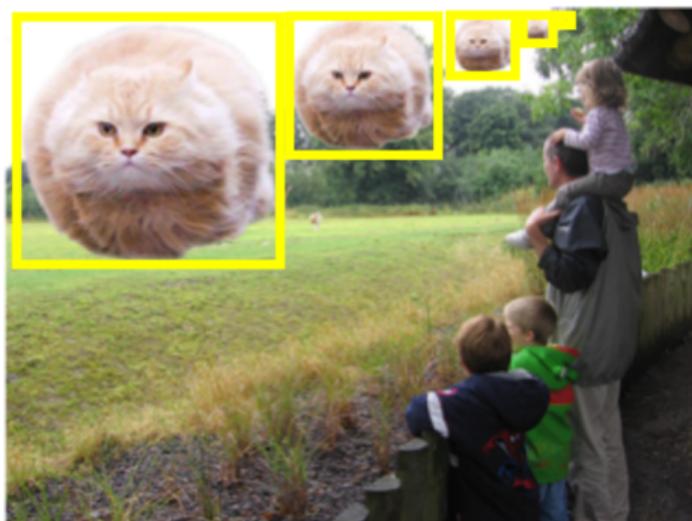


(c) Pyramidal feature hierarchy

Use the internal pyramid – *fast, suboptimal*  
(E.g.,  $\approx$  SSD, ...)

# Feature Pyramid Network

## Strategy 4: Feature Pyramid Network

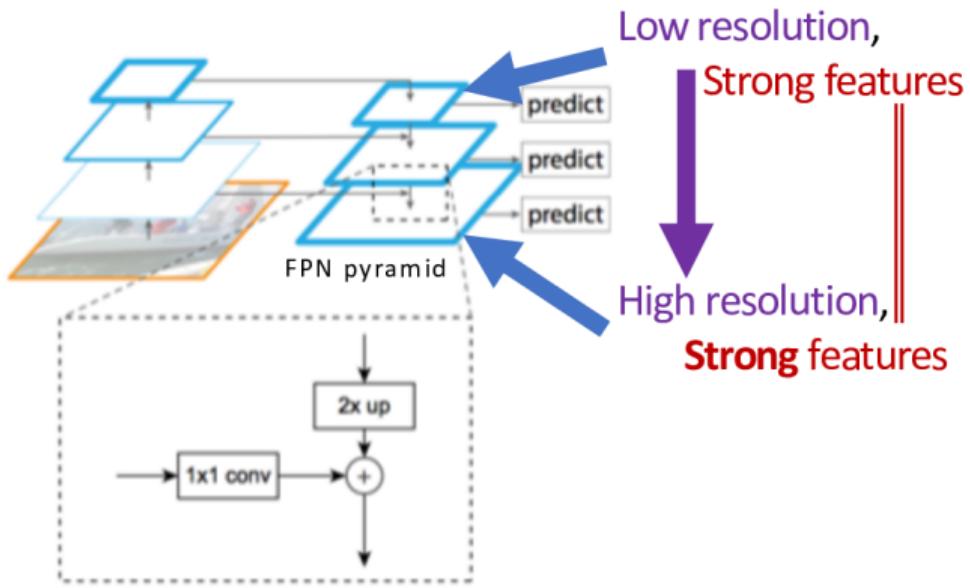
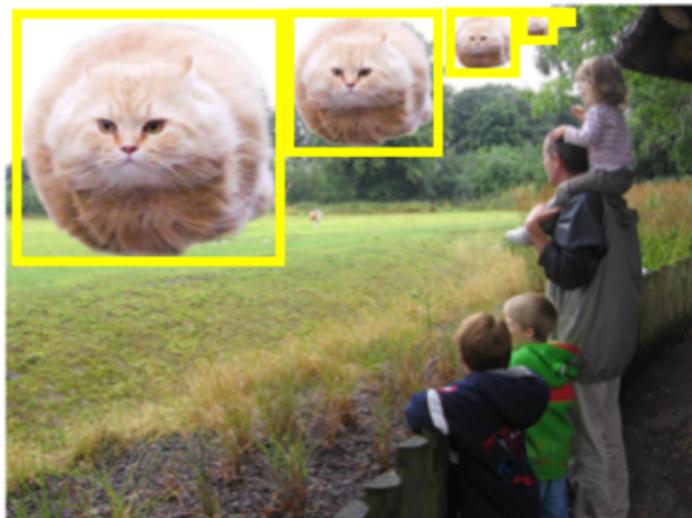


(d) Feature Pyramid Network

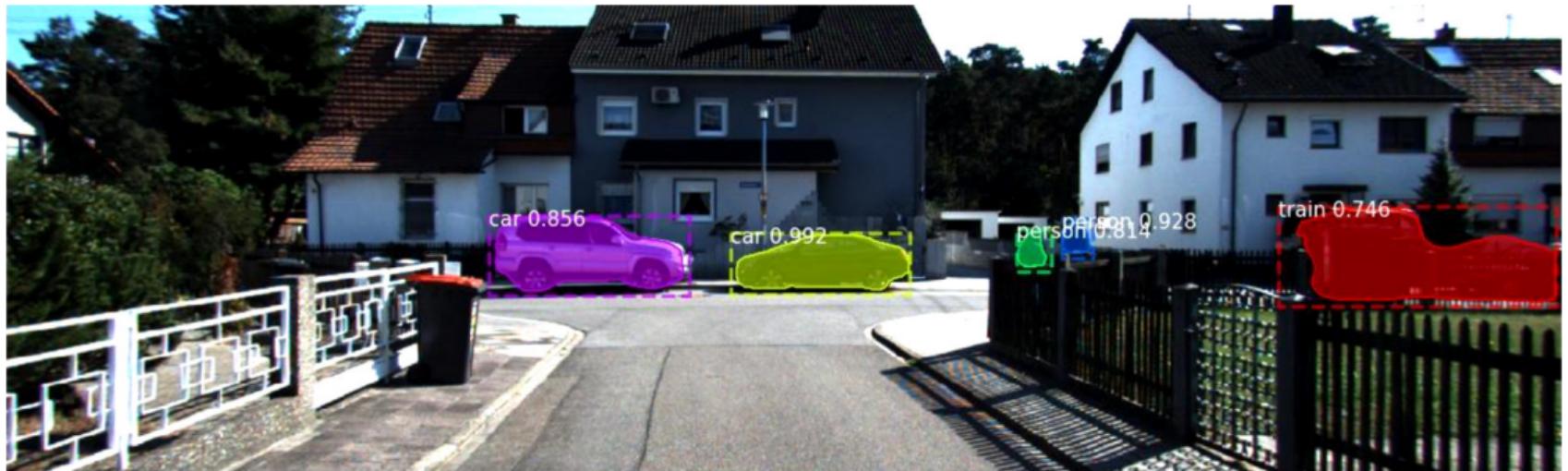
Top-down enrichment of high-res features –  
*fast, less suboptimal*

# Feature Pyramid Network

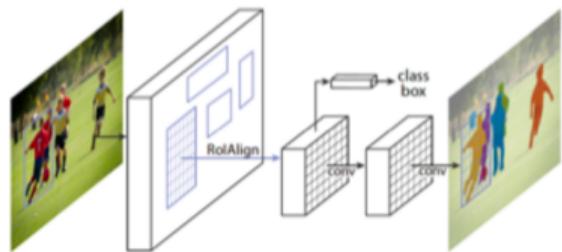
## Strategy 4: Feature Pyramid Network



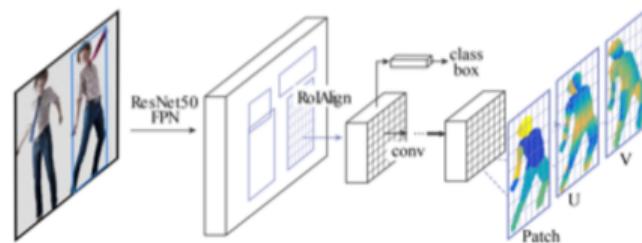
# Results



# Generalization to other Output Modalities



**Mask R-CNN**  
[He, Gkioxari, Dollár, Girshick]



**DensePose**  
[Güler, Neverova, Kokkinos]

- It is easy to add more/other output heads to this framework
  - Mask R-CNN: Predicting an instance segmentation mask per detection
  - DensePose: Predict object coordinates (relative 3D location on body)
  - Mesh R-CNN: Predict triangular mesh per object

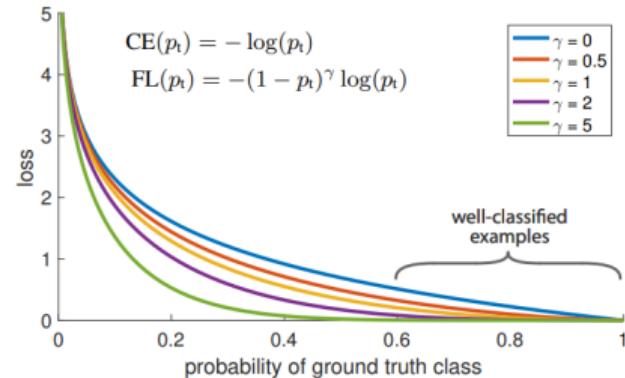
# Detectron2 for PyTorch



# Foreground-Background Imbalance

## Problem:

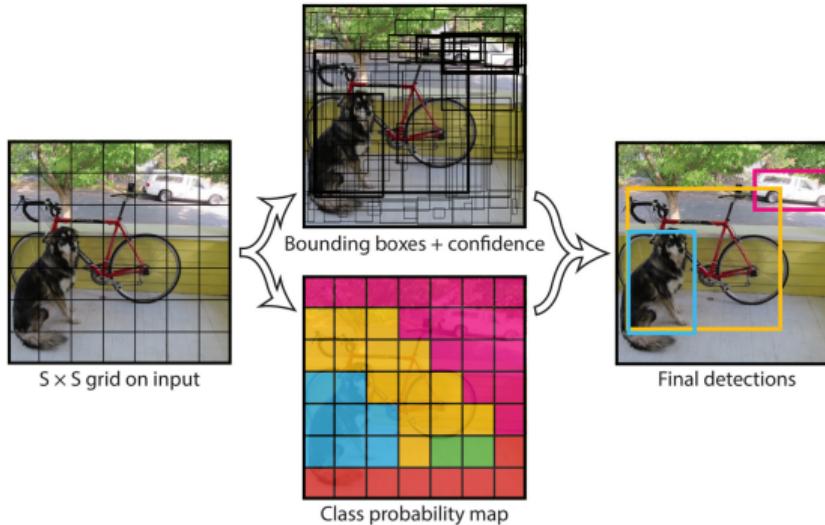
- ▶ Number of pos/neg boxes heavily imbalanced
- ▶ Before quantization: infinite imbalance
- ▶ After quantization: 0.01-0.1% foreground boxes
- ▶ Learning from imbalanced data is difficult
- ▶ Ignoring minority class  $\Rightarrow \sim 100\%$  accuracy



## Solutions:

1. Loss function which pays attention to hard examples, e.g., Focal Loss
2. Cascades, e.g., proposal-based detectors (2nd stage sees more balanced data)
3. Quantization, e.g., single-stage detectors (imbalance small by construction)

# Single Stage Detection



- ▶ YOLO and SSD: Alternative to proposal-based pruning of output space
- ▶ Regress boxes and predict class probability in single stage at  $7 \times 7$  locations
- ▶ Faster, but worse performance due to dramatic subsampling of output space  
(large quantization error hard to correct in practice  $\Rightarrow$  low AP)

# Paradigm Shift: DEtection TRansformer (DETR)

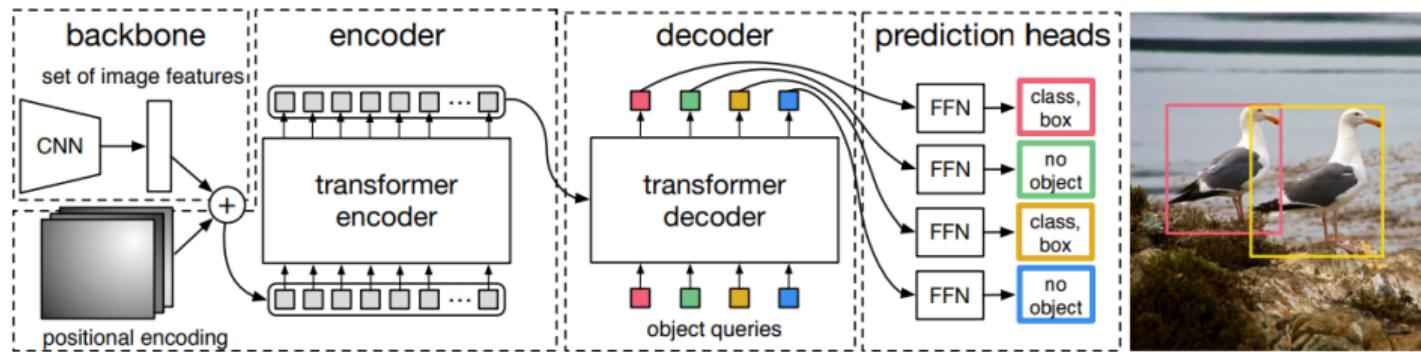
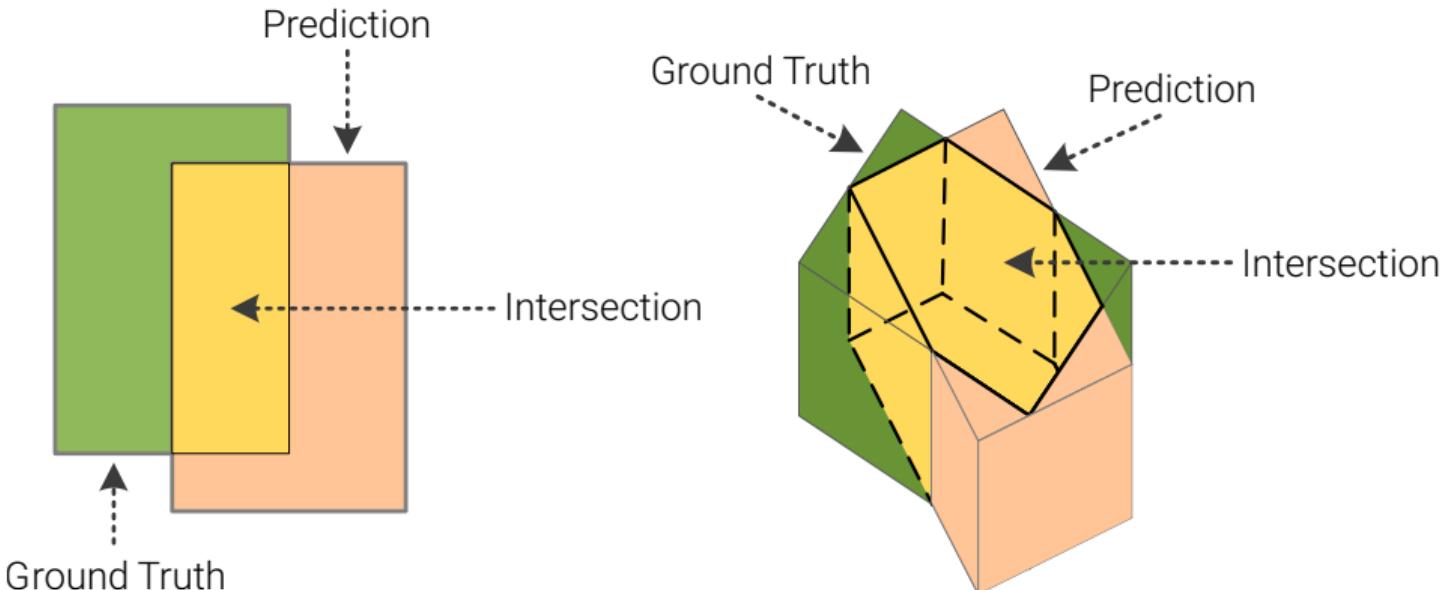


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a “no object” class.

# 10.5

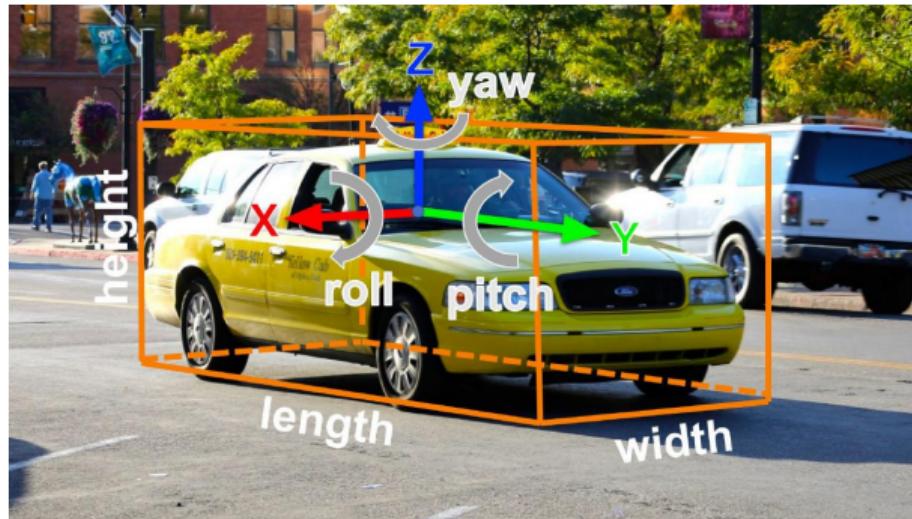
## 3D Object Detection

# 2D vs. 3D Object Detection



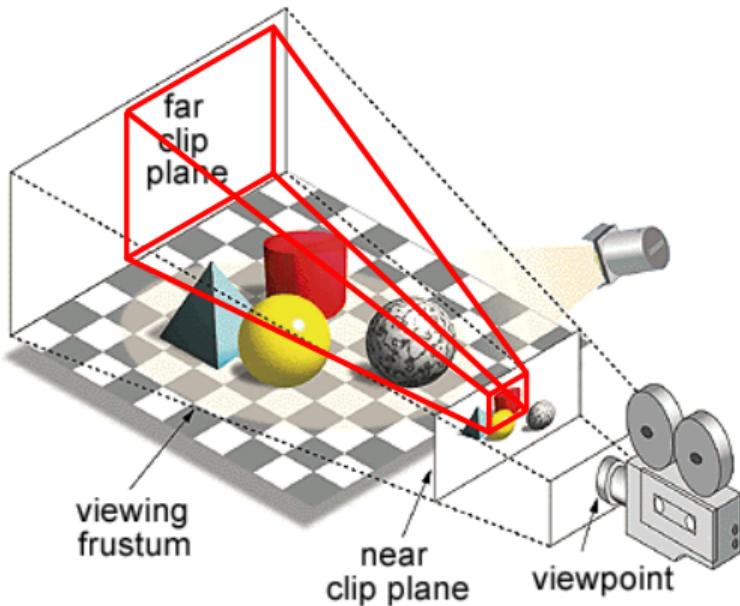
- ▶ Goal of 3D object detection: predict 3D bounding box for each object of interest
- ▶ Performance evaluation: 3D IoU can be calculated similarly to 2D IoU

# 2D vs. 3D Bounding Box

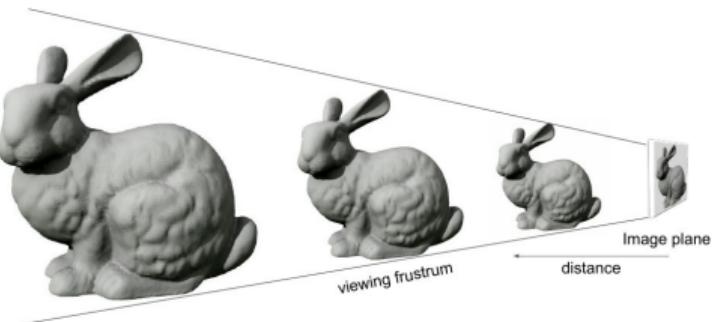


- ▶ **2D Object Detection**  
2D Box:  $(x, y, w, h)$
- ▶ **3D Object Detection**  
3D Box:  $(x, y, z, w, h, l, r, p, y)$
- ▶ **Simplification:**  
Assume zero roll and pitch
- ▶ Much harder than 2D detection
- ▶ Difficulty depends on input:  
2D image vs. 3D point cloud

# Image-based 3D Object Detection



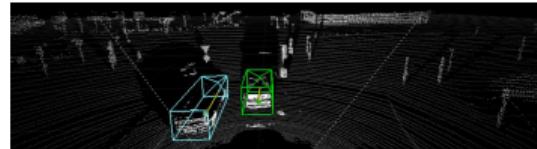
- ▶ A 2D bounding box in the image corresponds to a **frustum** in 3D
- ▶ The object can be **located anywhere** in the camera viewing frustum
- ▶ Objects of **different scales** and distances may look exactly the same:



# Input Modality



Image



Lidar

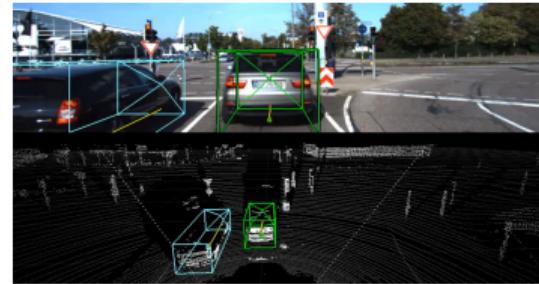


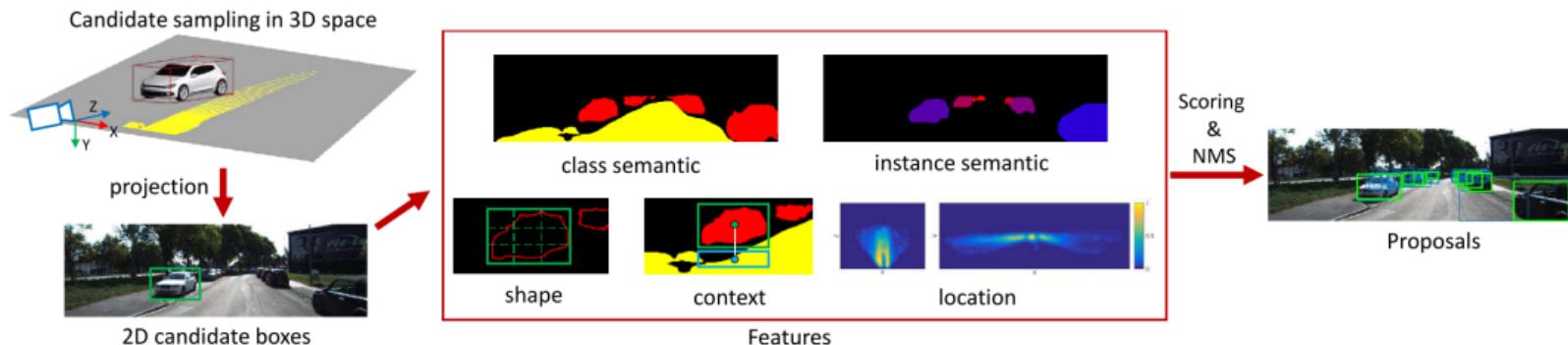
Image + Lidar

**Difficulty depends on input modality:**

- ▶ Regressing 3D boxes from **monocular images** requires good object size priors
- ▶ **Stereo** information helps to localize the box in 3D space (but quadratic error)
- ▶ **Lidar** sensors are sparse but provide good 3D accuracy at far range
- ▶ Several methods **combine** 2D image and Lidar information (complementary)

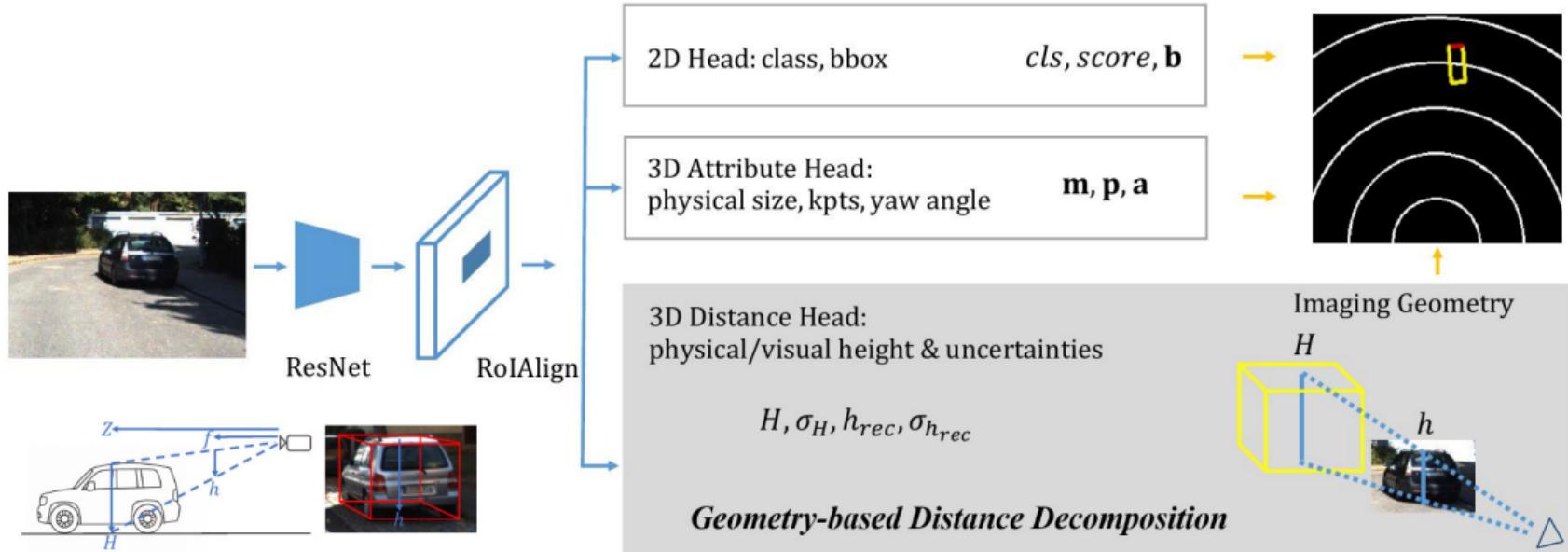
# Image-based 3D Object Detection

# Monocular 3D Object Detection for Autonomous Driving



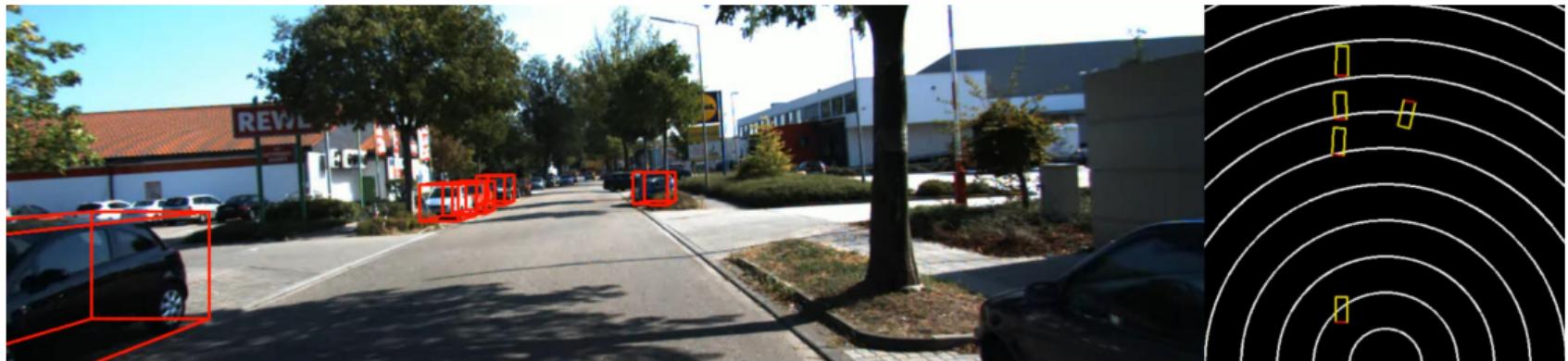
1. Sample **candidate 3D boxes** on the ground plane in non-road areas (gray)
2. Project 3D boxes to **image plane** in order to extract features
3. Feature-based **scoring**: semantics, instance, contour, shape, context, location
4. Final scoring and 2D refinement of top proposals using Fast R-CNN

# MonoRCNN: Monocular 3D Object Detection

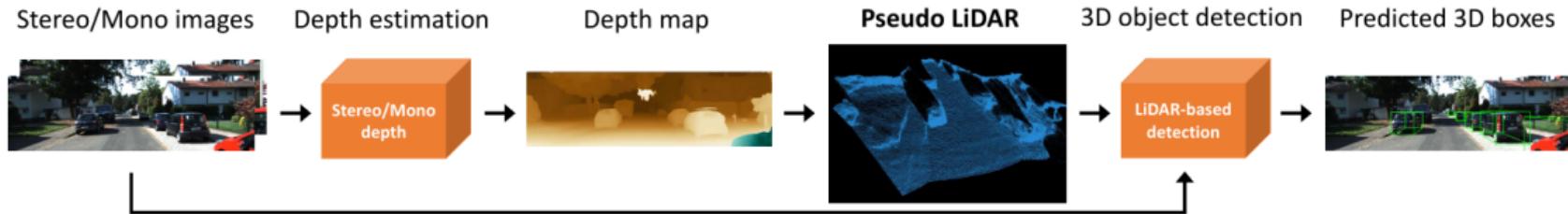


- ▶ Estimating **metrically accurate distance** is difficult from monocular images
- ▶ Idea: Regress physical and image height (+uncertainties)  $\Rightarrow Z = \frac{fH}{h}$

# MonoRCNN: Monocular 3D Object Detection



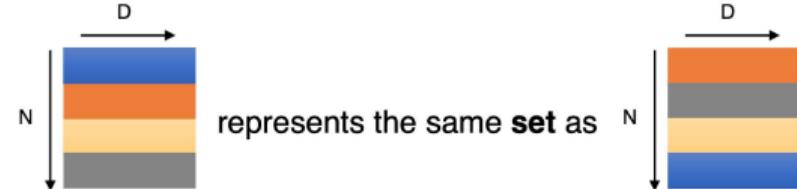
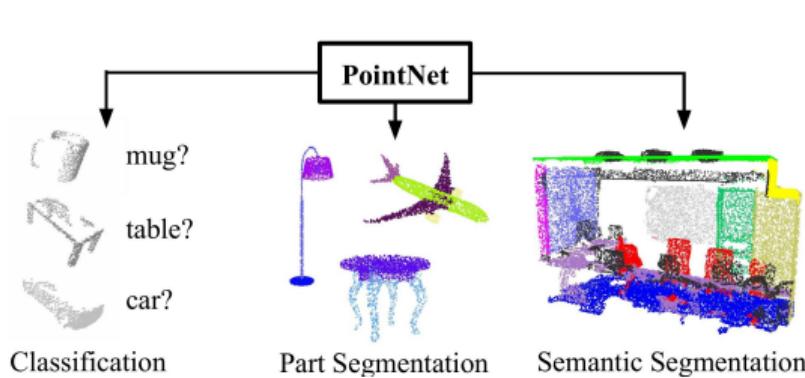
# Pseudo-LiDAR: 3D Object Detection from Point Clouds



- ▶ Given stereo or monocular images, first **predict a depth map**
- ▶ Backproject the depth map into a **3D point cloud** ("pseudo-Lidar")
- ▶ Apply **standard Lidar-based detection** algorithms to this point cloud
- ▶ Performance claimed to be competitive with Lidar-based detectors
- ▶ For a recent critical discussion, refer to [Simonelle et al., ICCV 2021]

# Lidar-based 3D Object Detection

# Learning with Point Clouds



$$f(x_1, x_2, \dots, x_n) \equiv f(x_{\pi_1}, x_{\pi_2}, \dots, x_{\pi_n}), \quad x_i \in \mathbb{R}^D$$

Examples:

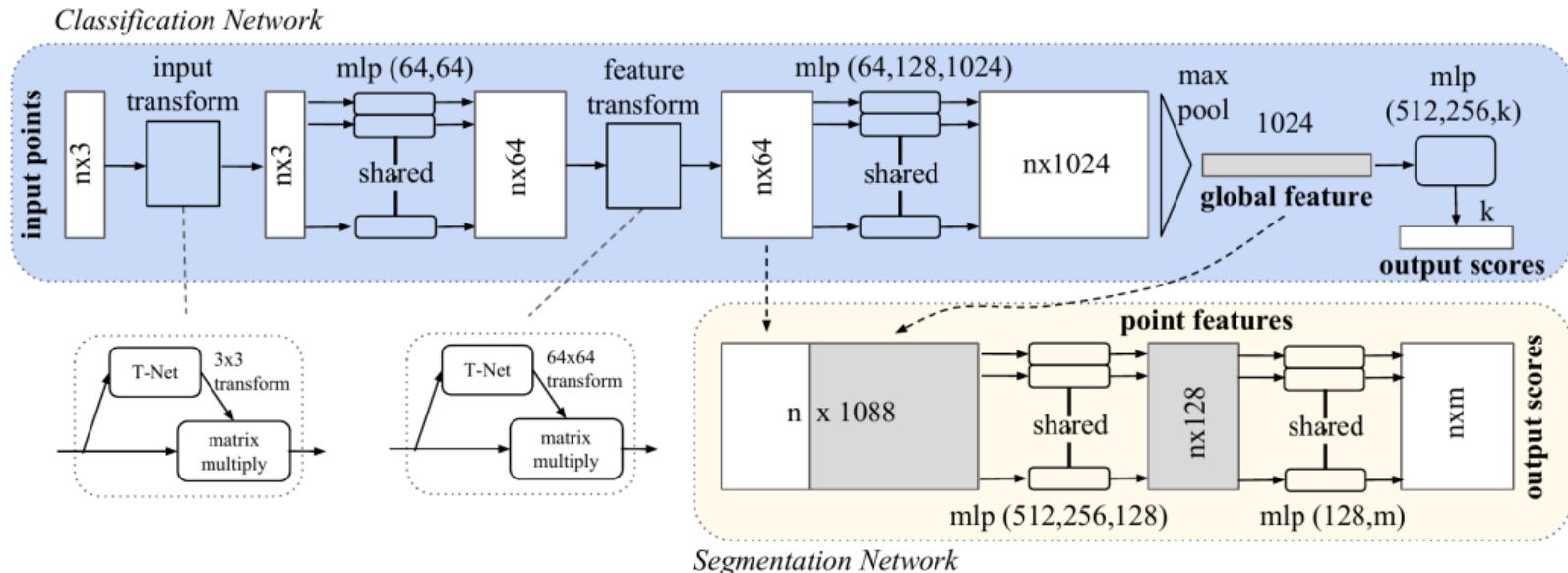
$$f(x_1, x_2, \dots, x_n) = \max\{x_1, x_2, \dots, x_n\}$$

$$f(x_1, x_2, \dots, x_n) = x_1 + x_2 + \dots + x_n$$

**Challenges** when learning with point clouds:

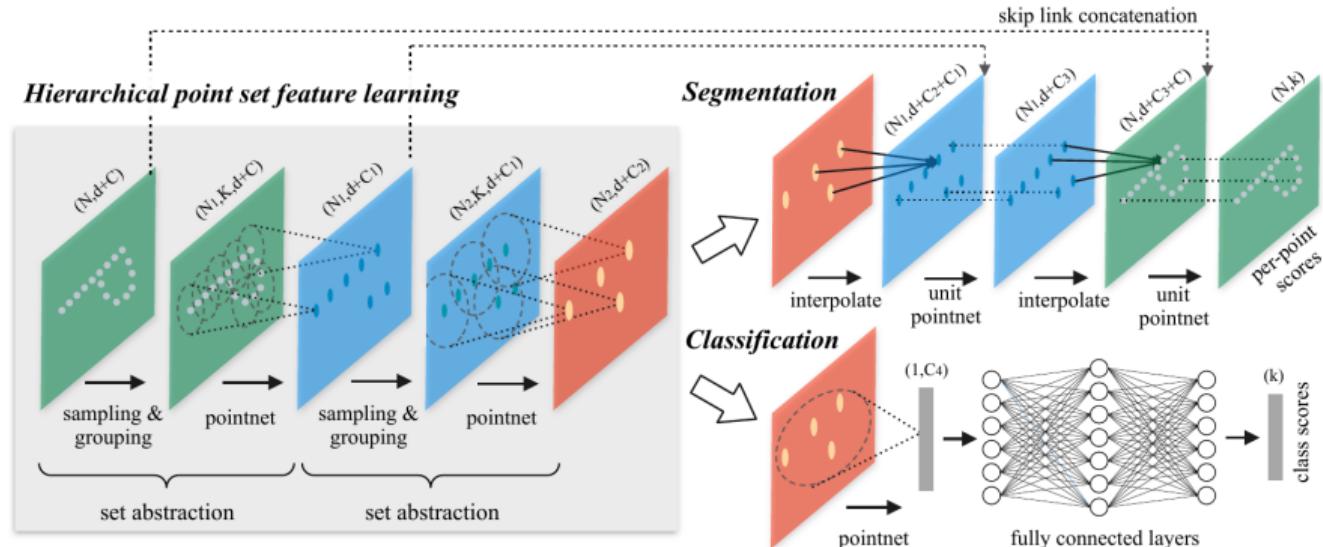
- The learned model must accommodate point clouds of **arbitrary size**
- The learned model needs to be invariant to all ( $N!$ ) **permutations**
- The learned model should be invariant to **rotations** of the point cloud

# PointNet



- ▶ PointNet applies MLP per point, followed by **max-pooling** and a final (global) MLP
- ▶ Affine input/feature transformations for geometric invariance (small PointNets)

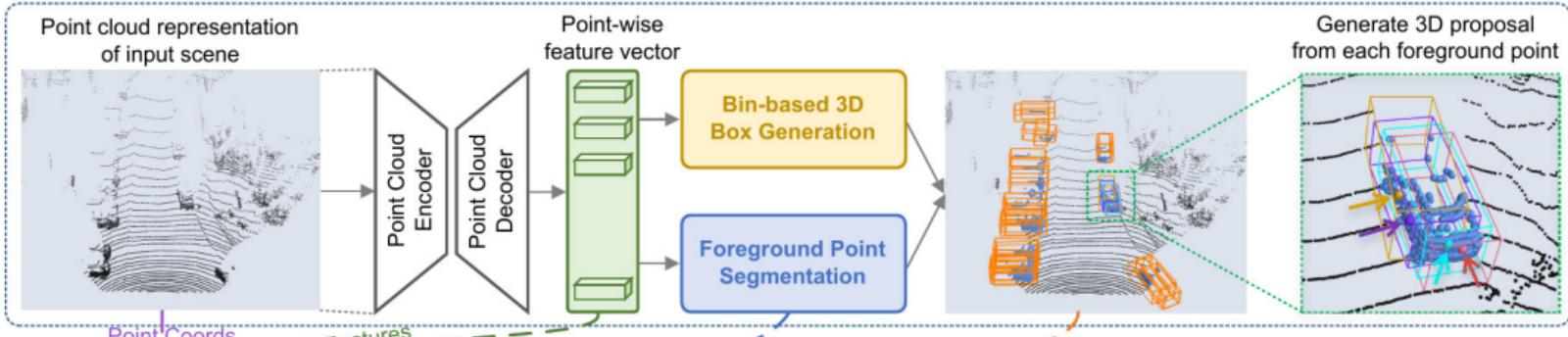
# PointNet++



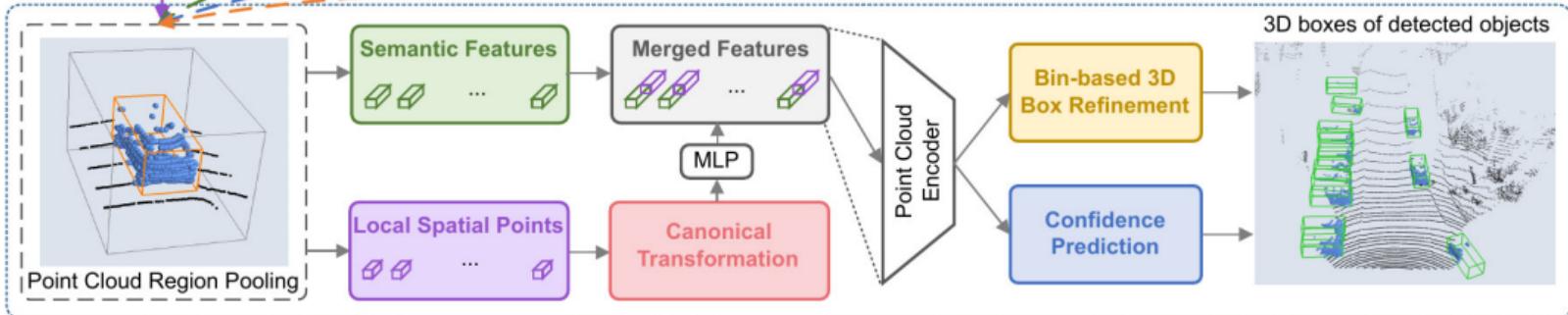
- ▶ PointNet does not capture local structures and depends on global coordinates
- ▶ PointNet++ applies PointNet **recursively** on **nested partitions** of the point set
- ▶ It learns features with increasing contextual scales  $\Rightarrow$  “Multi-scale PointNet”

# PointRCNN: Faster R-CNN for Point Clouds

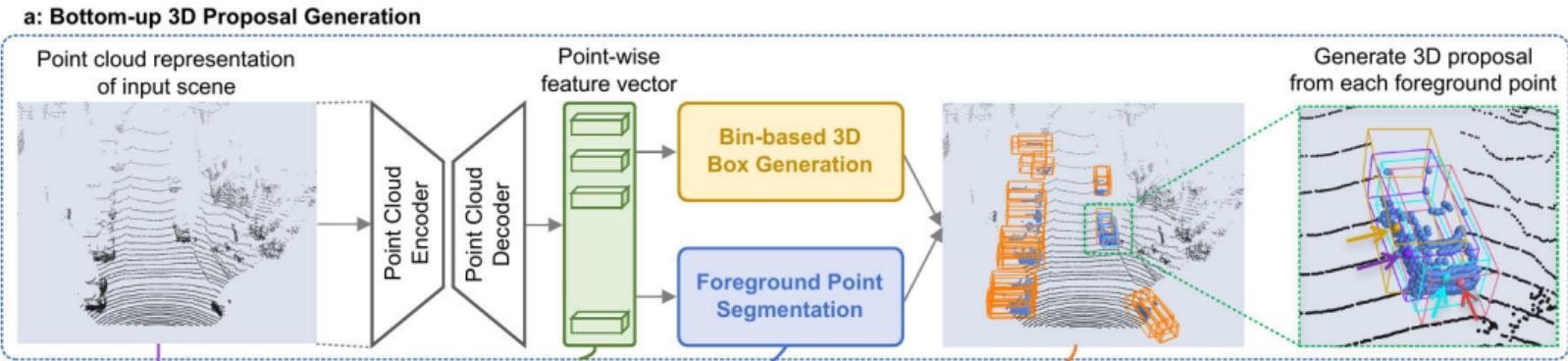
## a: Bottom-up 3D Proposal Generation



## b: Canonical 3D Box Refinement

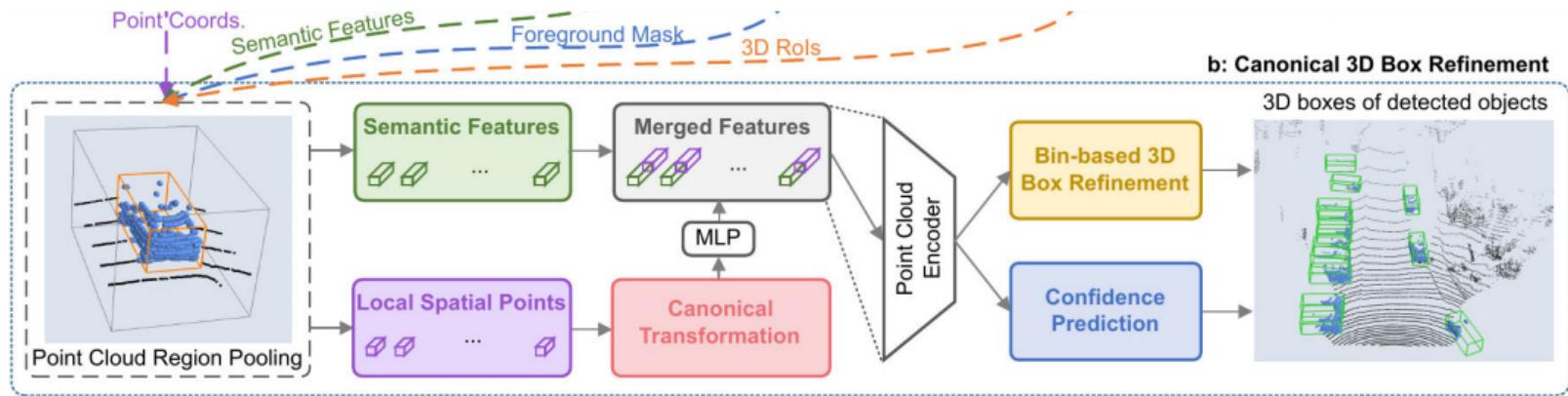


# PointRCNN: Faster R-CNN for Point Clouds



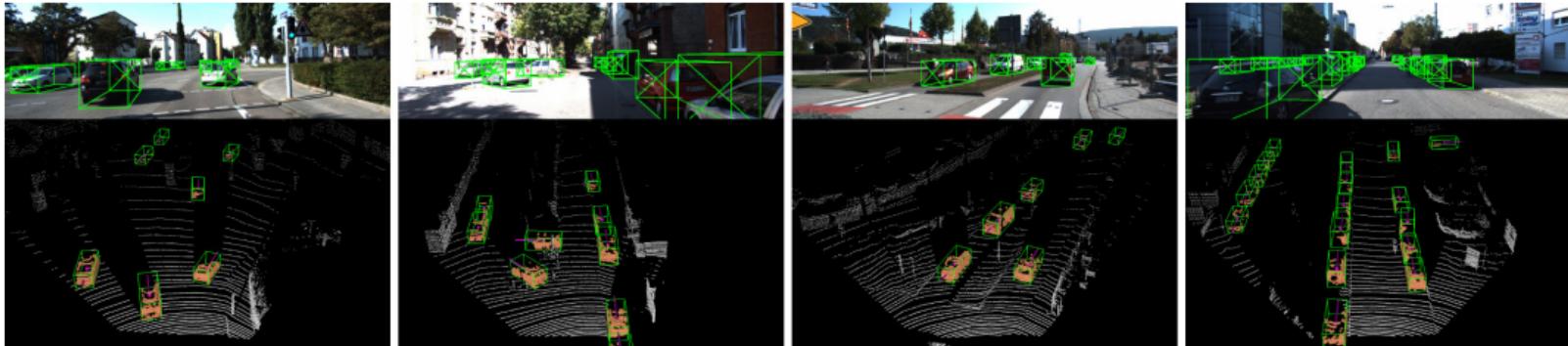
1. Backbone learns discriminative point-wise features with **PointNet++**
2. **3D box proposal regression** (like Faster R-CNN) and foreground segmentation (combination of regression and classification+regression (“bin-based”) losses)
3. Pass top 100 proposals to box refinement stage

# PointRCNN: Faster R-CNN for Point Clouds



1. **Pooling** of 3D point features based on 3D box proposals
2. Transform the pooled points to canonical (local) 3D box coordinates
3. **3D box refinement** by predicting residuals as in Faster R-CNN  
(combination of regression and classification+regression ("bin-based") losses)

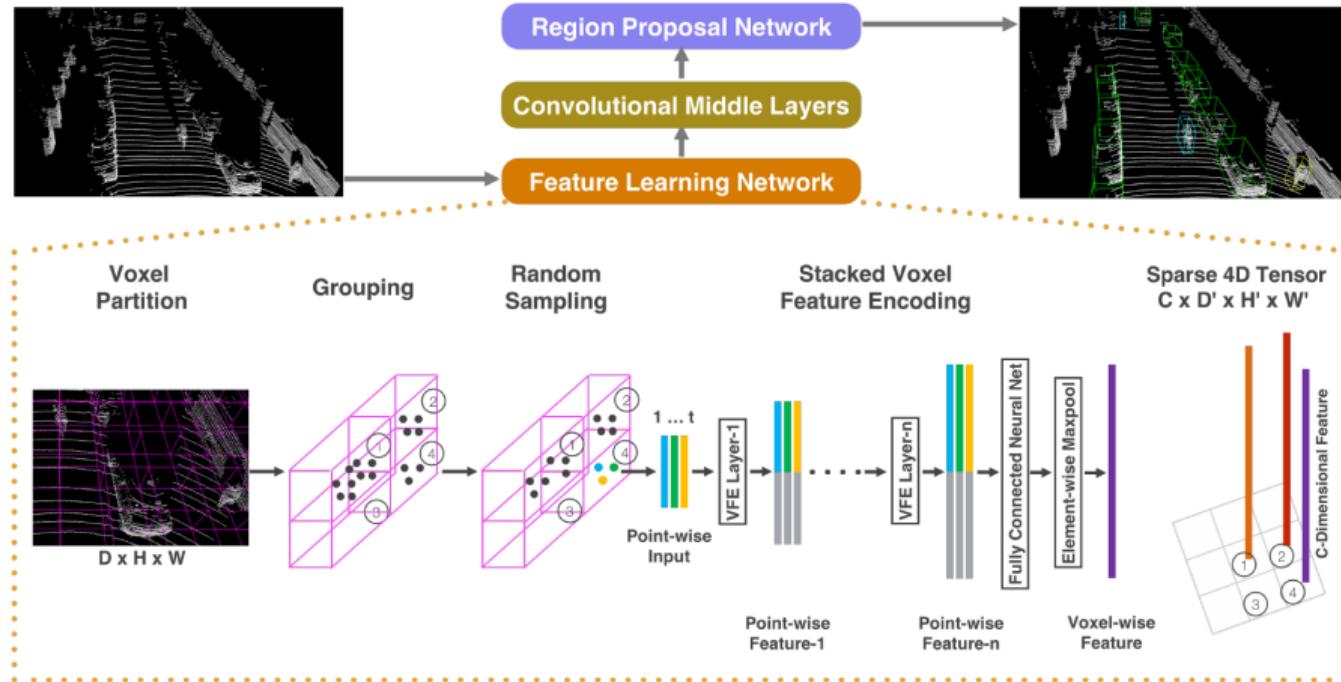
# PointRCNN: Faster R-CNN for Point Clouds



Method	Modality	Car (IoU=0.7)			Pedestrian (IoU=0.5)			Cyclist (IoU=0.5)		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
MV3D [4]	RGB + LiDAR	71.09	62.35	55.12	-	-	-	-	-	-
UberATG-ContFuse [17]	RGB + LiDAR	82.54	66.22	64.04	-	-	-	-	-	-
AVOD-FPN [14]	RGB + LiDAR	81.94	71.88	66.38	50.80	42.81	<b>40.88</b>	64.00	52.18	46.61
F-PointNet [25]	RGB + LiDAR	81.20	70.39	62.19	<b>51.21</b>	<b>44.89</b>	40.23	71.96	56.77	50.39
VoxelNet [43]	LiDAR	77.47	65.11	57.73	39.48	33.69	31.51	61.22	48.36	44.37
SECOND [40]	LiDAR	83.13	73.66	66.20	51.07	42.56	37.29	70.51	53.85	46.90
Ours	LiDAR	<b>85.94</b>	<b>75.76</b>	<b>68.32</b>	49.43	41.78	38.63	<b>73.93</b>	<b>59.60</b>	<b>53.59</b>

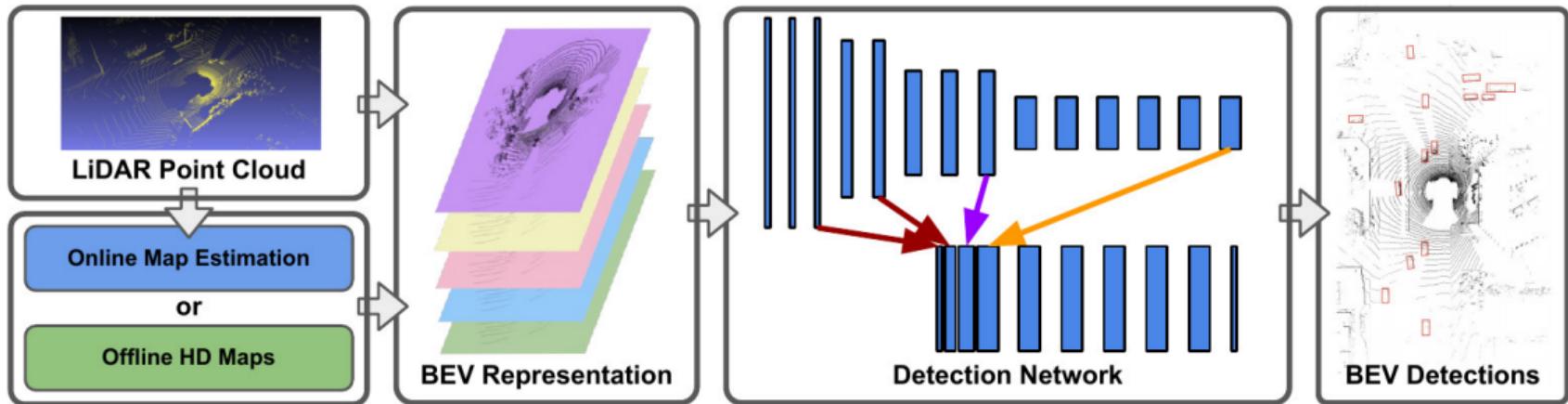
- Outperforms multi-modal methods, except for pedestrians which are small

# VoxelNet: Single Stage Detection with 3D Convolutions



- Alternative: Encode points into **voxel representation** and use 3D convolutions

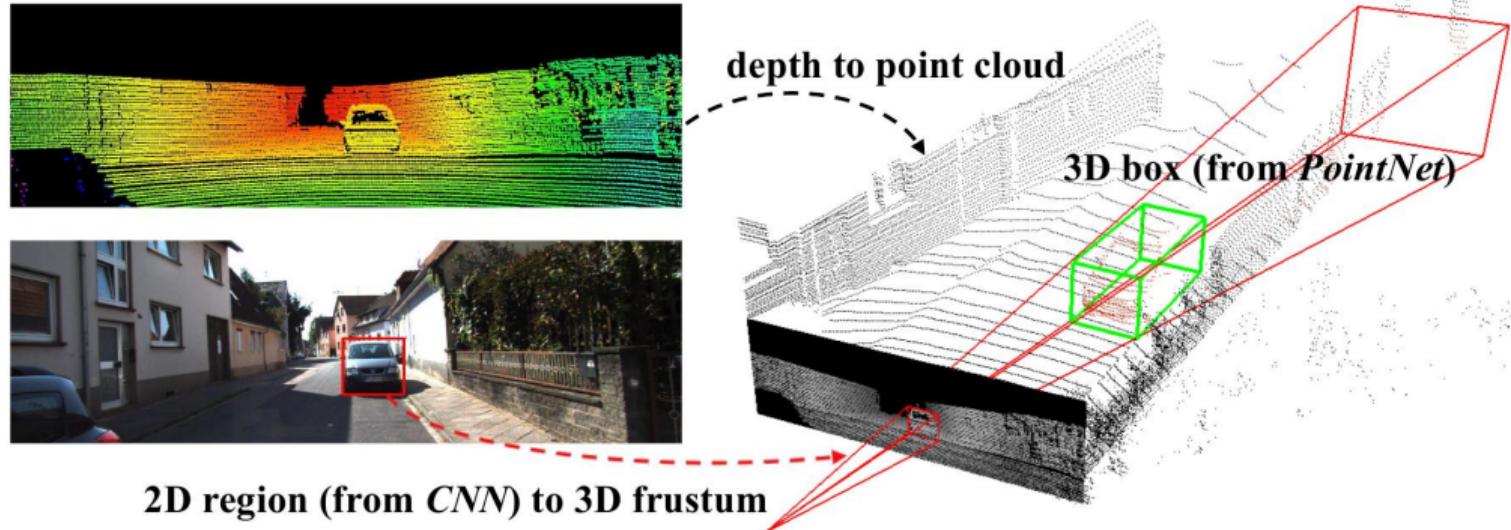
# PIXOR/HDNet: 2D BEV Representation



- ▶ Representation of Lidar and map data in **2D BEV**
- ▶ **Z-dimension** treated as **feature channel** ⇒ efficient 2D convolutions
- ▶ Road maps (HD maps) provide strong priors that can boost 3D detectors

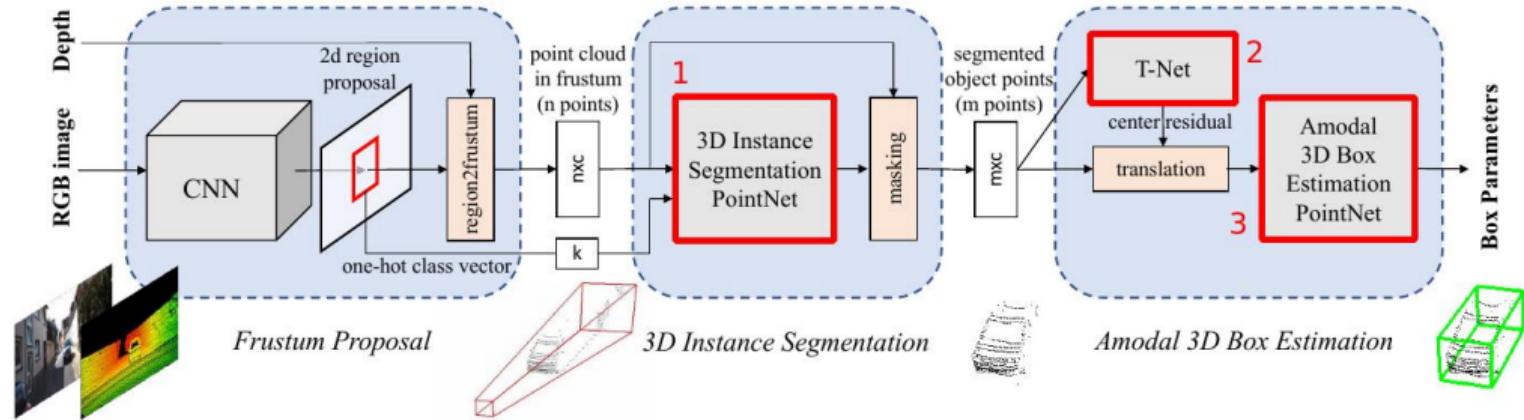
# Multi-modal 3D Object Detection

# Frustum PointNet: Detect 3D Boxes in Object Frustum



1. Generate **2D object proposals** in the RGB image
2. Extract point cloud from **3D object frustum**
3. **Predict 3D bounding box** from points in frustum via PointNet

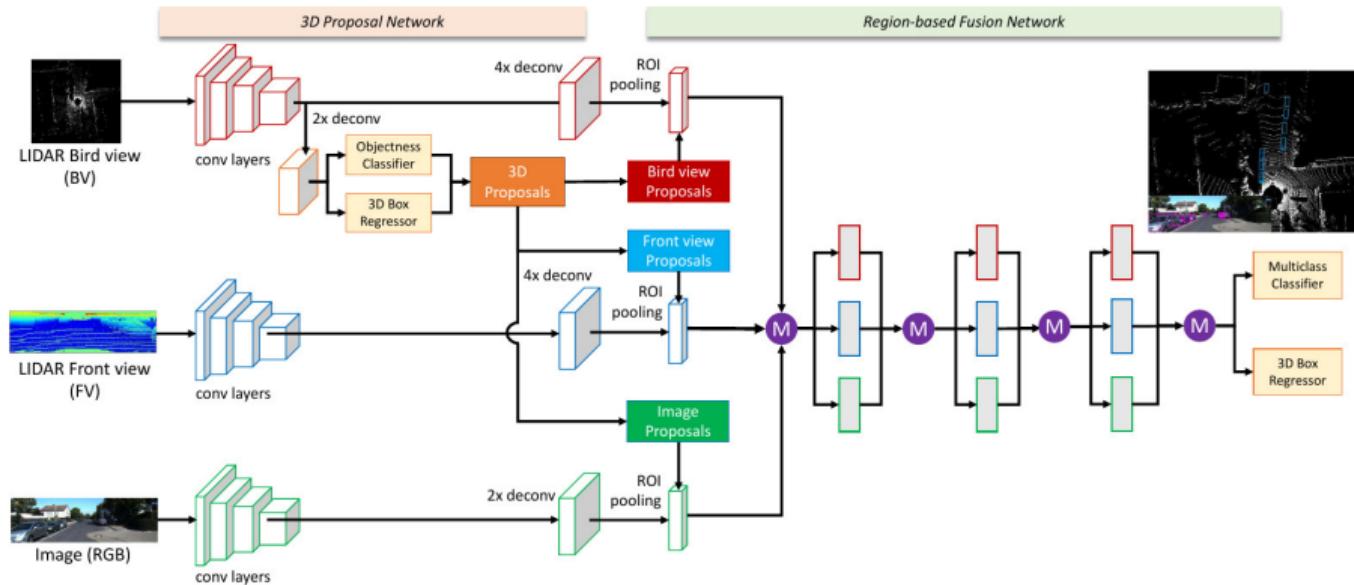
# Frustum PointNet



3D box prediction problem is split into **3 stages**: (all implemented via PointNets)

1. **Segmentation** of RGB point cloud (object vs. background)
2. **Translation** of points such that centroid aligns with 3D box center
3. **3D bounding box regression** ( $x, y, z, w, h, l, y$ )

# MV3D: Combining Multiple 2D Views (RGB/Lidar Projections)



1. Generate 3D proposals from Lidar BV and project into **3 modalities** (Proposals)
2. **Fuse ROI-pooled features** from all modalities, regress box residuals (Refinement)

# MV3D: Combining Multiple 2D Views (RGB/Lidar Projections)

Method	Data	IoU=0.25			IoU=0.5			IoU=0.7		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Mono3D [3] 3DOP [4]	Mono	62.94	48.2	42.68	25.19	18.2	15.52	2.53	2.31	2.31
	Stereo	85.49	68.82	64.09	46.04	34.63	30.09	6.55	5.07	4.1
VeloFCN [16]	LIDAR	89.04	81.06	75.93	67.92	57.57	52.56	15.20	13.66	15.98
Ours (BV+FV)	LIDAR	96.03	88.85	88.39	95.19	87.65	80.11	71.19	56.60	55.30
Ours (BV+FV+RGB)	LIDAR+Mono	<b>96.52</b>	<b>89.56</b>	<b>88.94</b>	<b>96.02</b>	<b>89.05</b>	<b>88.38</b>	<b>71.29</b>	<b>62.68</b>	<b>56.56</b>

Table 2: **3D detection performance:** Average Precision ( $AP_{3D}$ ) (in %) of 3D boxes on KITTI validation set.



## Summary

- ▶ Localizing and recognizing objects is crucial for self-driving
- ▶ However, there are ( $\infty$ ) many possible boxes and number of objects unknown
- ▶ Challenges: appearance/viewpoint variation, illumination, clutter, occlusion
- ▶ Detection performance is typically measured using average precision
- ▶ Classic sliding-window detection methods use hand-crafted features
- ▶ Deep learning has boosted recognition performance by 10x in few years
- ▶ Two-stage detection methods (Faster R-CNN, Mask R-CNN) are state-of-the art
- ▶ Feature pyramid networks help detecting features at various scales
- ▶ 3D detection methods rely on RGB images, Lidar point clouds or both
- ▶ Lidar information is crucial for accurate 3D localization