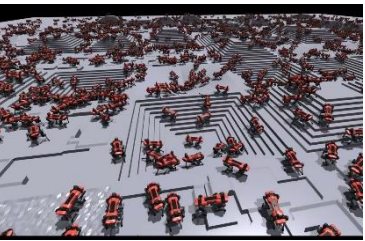
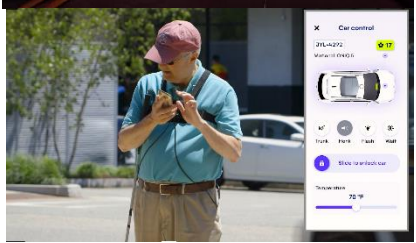
EC500: Robot Learning and Vision for Navigation



Eshed Ohn-Bar

Feb 22, 2023



- Research on assessment and Intervention of learning disabilities in first and second grade students with AI

- Interested? Please contact us.

SCC “Hacks”

- Reduce data size
- Reduce image size?
- Ask for higher capacity GPUs
- Careful with port numbers (see blackboard)

You are currently reading documentation for the "dev" branch of CARLA. This documentation refers to features currently in development and may result in unexpected behaviour. To read documentation for previous releases, select the desired version in the bottom, right-hand corner of the screen.

Tutorials

Here you will find the multitude of tutorials available to help you understand how to use CARLA's many features.

General

CARLA features

Retrieve simulation data — A step by step guide to properly gather data using the recorder.
Traffic manager — How to use traffic manager to guide traffic around your town.
Texture streaming — Modify textures of map objects in real time to add variation.
Instance segmentation camera — Use an instance segmentation camera to distinguish objects of the same class.
Bounding boxes — Project bounding boxes from CARLA objects into the camera.
Pedestrian bones — Project pedestrian skeleton into camera plane.
Control walker skeletons — Animate walkers using skeletons.

Building and integration

Build Unreal Engine and CARLA in Docker — Build Unreal Engine and CARLA in Docker.

Poll 1

Poll 2

Robot Learning and Vision for Navigation

Exercise 2 – Modular Pipeline

Release date: Thursday, 21 Feb. 2023 - **Deadline for Homework: Monday, 13 Mar. 2023 - 23:59**

For this exercise you need to submit a **.zip** folder containing your report as a **.pdf** file (up to 5 pages) and your **.py** code files.

As in the previous exercise, please use the provided code templates. Add your best choice of parameters to the `modular_pipeline.py` file.

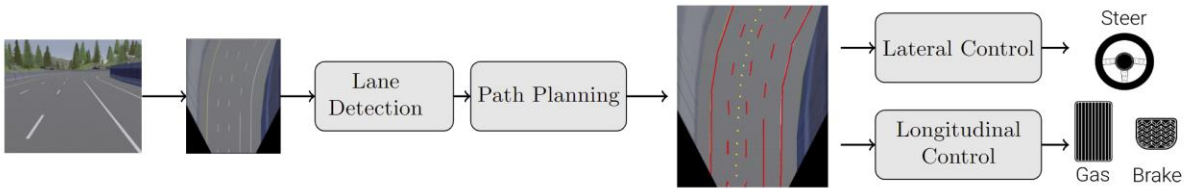


Figure 1: Modular pipeline consisting of a lane detection module, path planning and a control unit.

Homework 2:
Essentially implementing parts of Stanley (2005 DARPA challenge winner)

http://isl.ecst.csuchico.edu/DOCS/darpa2005/DARPA%202005%20Stanley.pdf

Lane Detection

a) Edge Detection:

- Translate the state image to a grey scale image and crop out the part above the car
  → LaneDetection.cut_gray()
- Derive the absolute value of the gradients of the grey scale image and apply thresholding to ignore unimportant gradients.
  → LaneDetection.edge_detection()
- Determine arguments of local maxima of absolute gradient per pixel row
  → LaneDetection.find_maxima_gradient_rowwise()
  Hint: use for example scipy.signal.find_peaks

Lane Detection

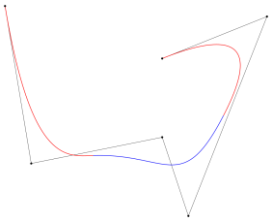
b) Assign Edges to Lane Boundaries:

- Find arguments of local maxima in the image row closest to the car
  → `LaneDetection.find_first_lane_point()`
  (already implemented)
- Assign the edges to the lane boundaries by successively searching for the nearest neighboring edge/maximum along each boundary
  → `LaneDetection.lane_detection()`

Lane Detection

c) Spline Fitting:

- Fit a parametric spline to each lane boundary
  → LaneDetection.lane_detection()
- Use scipy.interpolate.splprep for fitting and scipy.interpolate.splevfor evaluation



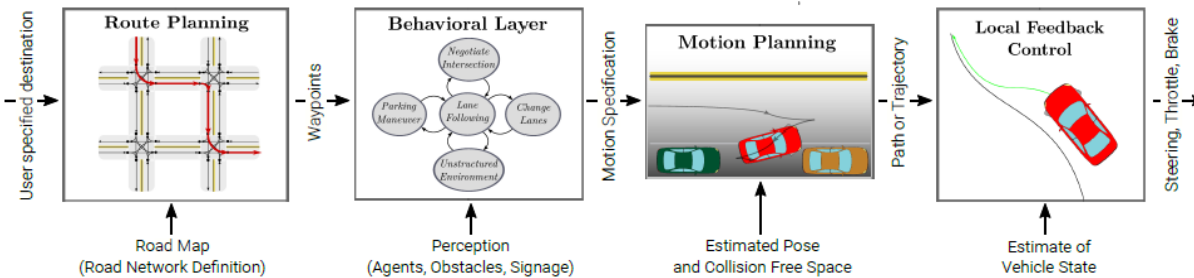
Given a list of points, which represents a curve in 2-dimensional space parametrized by s, find a smooth approximating spline curve $g(s)$.

Lane Detection

d) Testing:

- ► Find a good choice of parameters for the gradient threshold and the spline smoothness.
- ► Determine failure cases
- ► Add pictures to your report

Path Planning



- Break planning problem into a hierarchy of simpler optimization problems
- Each optimization problem tailored to its scope and level of abstraction
- Higher in the hierarchy means more abstraction
- Each optimization problem will have constraints and objective functions

Path Planning

- waypoint_prediction.py
- test_way_point_prediction.py for testing

a) Road Center:

- Use the lane boundary splines and derive lane boundary points for 6 equidistant spline parameter values
  → waypoint_prediction()
- Determine the center between lane boundary points with the same spline parameter
  → waypoint_prediction()

Path Planning

c) Target Speed Prediction:

- Implement a function that outputs the target speed for the predicted path in the state image , using

$$v_{\text{target}}(x_1, ..., x_N) = (v_{\text{max}} - v_{\text{min}}) \exp\left[-K_v \cdot \left| N - 2 - \sum_n \frac{(x_{n+1} - x_n) \cdot (x_n - x_{n-1})}{|x_{n+1} - x_n| \, |x_n - x_{n-1}|} \right|\right] + v_{\text{min}},$$

As initial parameters use: $v_{\text{max}} = 60, \quad v_{\text{min}} = 30$ and $K_v = 4.5.$

target_speed_prediction()

Template

- lateral control.py
- test lateral control.py for testing

a)Stanley Controller:

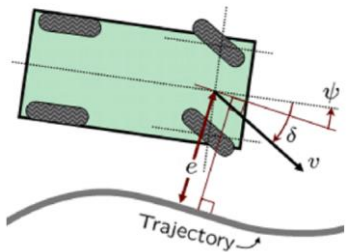
- Read section 9.2 in the Stanley paper

  http://isl.ecst.csuchico.edu/DOCS/darpa2005/DARPA%202005%20Stanley.pdf

  Explain the parts of the heuristic control law

Lateral Control

Lateral Control



- Stanley controller
- $\delta(t) = \psi(t) + \arctan(\frac{ke(t)}{v(t)})$
- Dampening term (rate) k

Lateral Control

b) Stanley Controller:

- ► Implement controller function given waypoints and speed
  → LateralController.stanley()
- ► Orientation error $\psi(t)$ is the angle between the first path segment to the car orientation
- ► Cross track error $d(t)$ is distance between desired waypoint at a spline parameter of zero to the position of the car
- ► Prevent division by zero by adding as small epsilon
- ► Describe the behavior of your car

c) Damping:

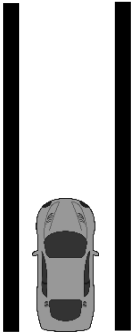
- ► Damping the difference between the steering command and the steering wheel angle of the previous step

$$\delta(t) = \delta_{SC}(t) - D \cdot (\delta_{SC}(t) - \delta(t - 1)).$$

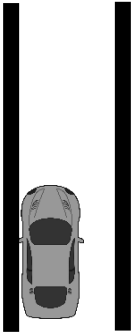
- ► Describe the behavior of your car

PID Control

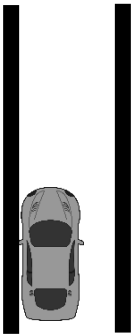
PID Control



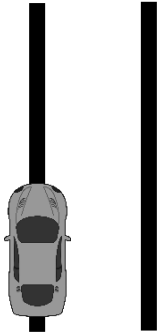
PID Control



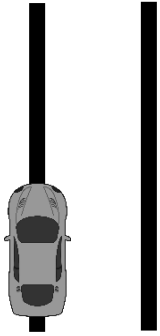
Skill: PID Control



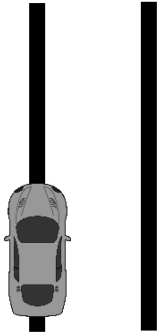
PID Control



PID Control



PID Control



Control ($u(t)$) is **proportional to the error**
(with respect to a **desired position**)

PID Control… Cruise Control example

Case 1: hold a speed of 65 MPH

Hill: “more gas”

Downhill: ”coast”

“Setpoint” == 65 MPH

Key to this working: Measure difference between **setpoint** and **measured** speed – “**Error**”

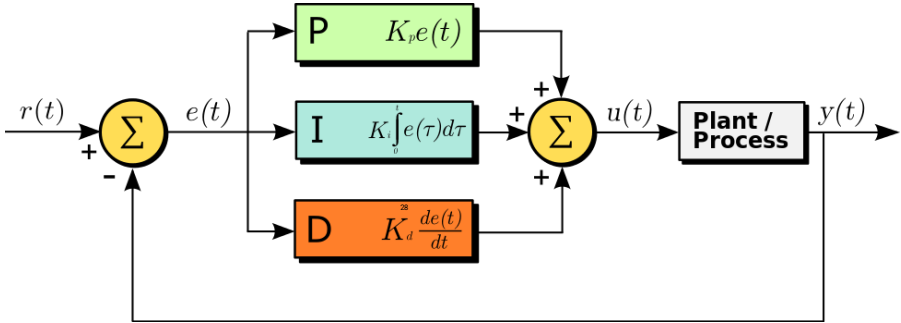
Can you hold a constant speed?

Bang-Bang Controller

- Feedback controller that switches abruptly between two states
- For instance, household thermostats
- Formulation

$$u(t) = \begin{cases} u_1 \; if \; e(t) > \tau \\ u_2 \quad \text{otherwise} \end{cases}$$

PID Controller



Mathematical formulation:

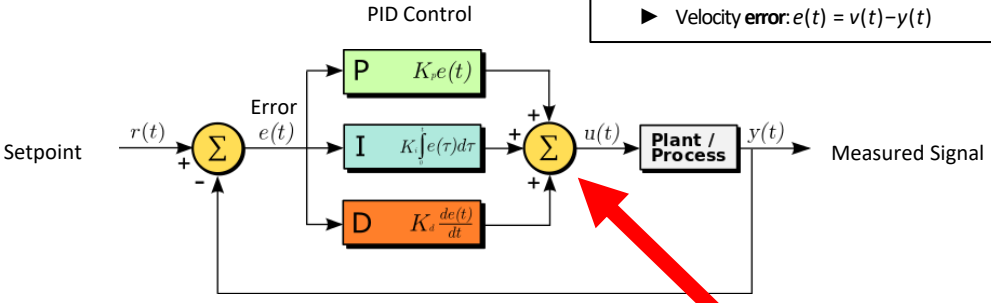
$$u(t) = K_p\, e(t) + K \int_0^t e(t')dt' + K_d \frac{de(t)}{dt}$$

with coefficients/parameters K_p, K_i and K_d

PID Control

EXAMPLE:

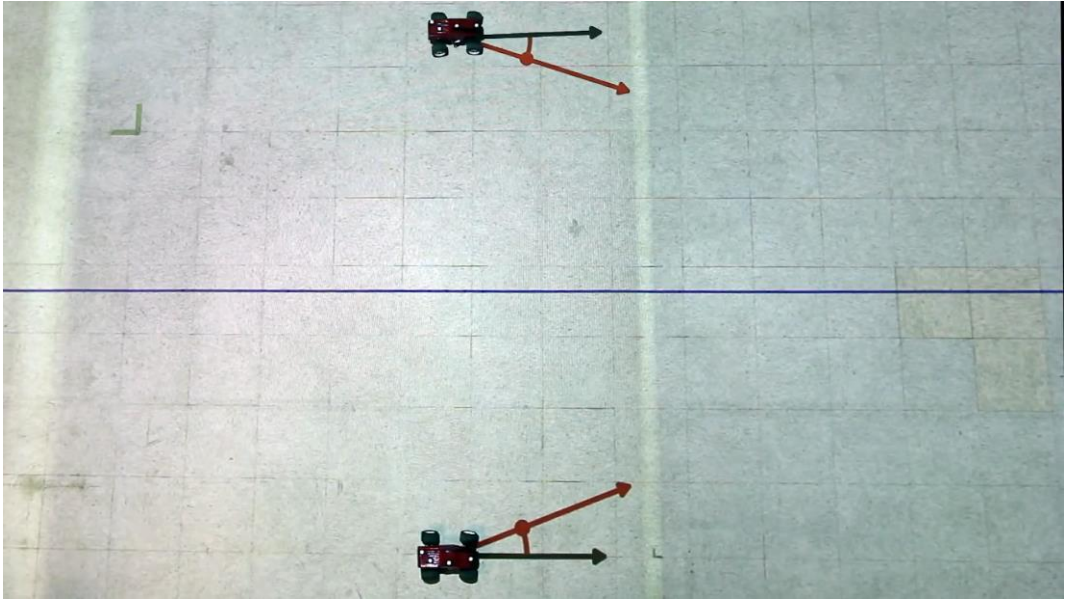
- **Set point** for velocity: $r(t) = v(t)$ =target velocity
- Control variable: $u(t)$ = gas/brake pedals
- Process variable: $y(t)$ = current velocity
- Velocity **error**: $e(t) = v(t) - y(t)$



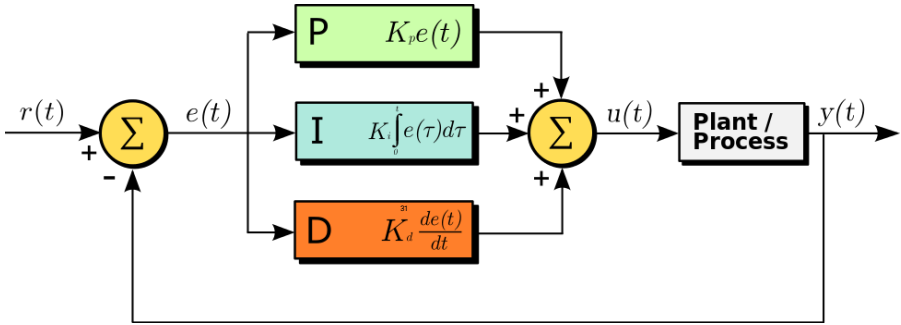
PID operates on "error" signal
Proportional – scaled with error signal
Integral – scaled with accumulated error
Differential/Derivative – scaled with change in error signal

Scale factors combined to apply to the actuator (the motor)
"more gas!"

PID Controller



PID Controller



- Using the **P** element alone leads to overshooting / oscillation
- The **I** element corrects residual errors by integrating past error measurements
- Adding a **D** element solves this problem by introducing a damping behavior

Longitudinal Control

Template

- longitudinal_control.py
- test_longitudinal_control.py for testing

a) PID Controller:

- Implement a PID control step for gas and braking
- Use a discretized version:

$$e(t) = v_{\text{target}} - v(t)$$

$$u(t) = K_p \cdot e(t) + K_d \cdot [e(t) - e(t-1)] + K_i \cdot \left[\sum_{t_i=0}^{t} e(t_i) \right]$$

where $u(t)$ is the control signal and $e(t)$ error signal

How to code a PID loop

Proportional – feedback signal function of distance (for example)
Differential – feedback signal function of rate of distance change
Integral – feedback signal function of accumulated error in distance

```
previous_error = 0
integral = 0

While(1)
{
  error = setpoint - measured_value
  integral = integral + error * dt
  derivative = (error - previous_error) / dt
  output = Kp * error + Ki * integral + Kd * derivative
  previous_error = error
  wait(dt)
}
```

K's – scale factors for each parameter

dt – time period for your feedback loop

b)Parameter Search:

- Run test lateral control.pyand have a look at plots of the target speed and speed to tune parameters (K_p, K_i, K_d)

- Start with (K_p = 0.01, K_i = 0, K_d =0)

- Only modify a single term at a time !!!!

Robot Learning and Vision for Navigation

Exercise 2 – Modular Pipeline

Release date: Thursday, 21 Feb. 2023 - **Deadline for Homework: Monday, 13 Mar. 2023 - 23:59**

For this exercise you need to submit a **.zip** folder containing your report as a **.pdf** file (up to 5 pages) and your **.py** code files.

As in the previous exercise, please use the provided code templates. Add your best choice of parameters to the `modular_pipeline.py` file.

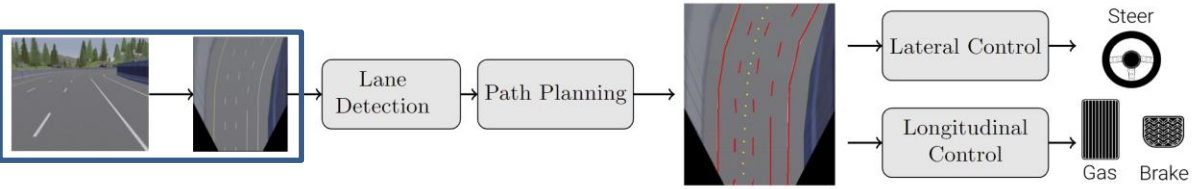
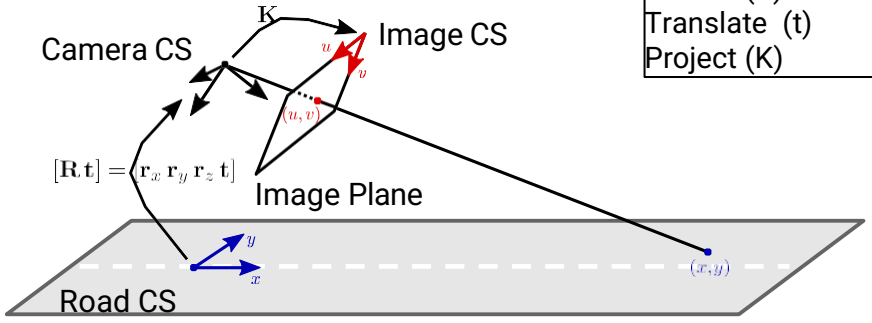


Figure 1: Modular pipeline consisting of a lane detection module, path planning and a control unit.

Inverse Perspective Mapping (IPM)

3D Operations:
Rotate (R)
Translate (t)
Project (K)

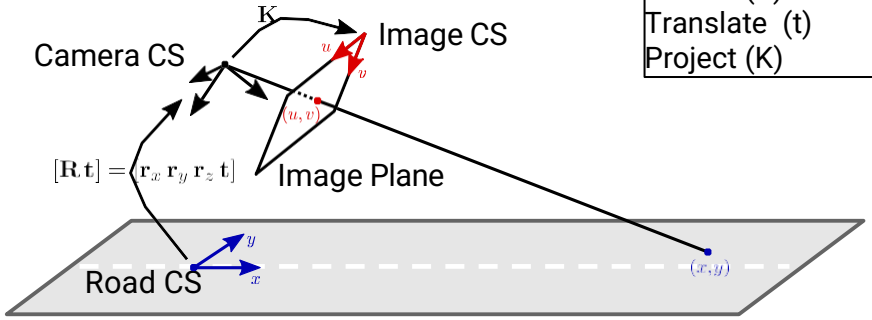


- IPM maps each pixel on the image plane to the respective ground plane point
- IPM is only possible if either per pixel depth or ground model (e.g., plane) known
- How to obtain ground plane (i.e., extrinsic parameters [**R t**])?

Bertozzi and Broggi: GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection. TIP, 1998.

Inverse Perspective Mapping (IPM)

3D Operations:
Rotate (R)
Translate (t)
Project (K)



$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \propto \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \overset{z=0}{\Rightarrow} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \propto \mathbf{K} \begin{bmatrix} \mathbf{r}_x & \mathbf{r}_y & \mathbf{t} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \propto \begin{bmatrix} \mathbf{r}_x & \mathbf{r}_y & \mathbf{t} \end{bmatrix}^{-1} \mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Bertozzi and Broggi: GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection. TIP, 1998.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \overbrace{\begin{bmatrix} f_x & 0 & s_x \\ 0 & f_y & s_y \\ 0 & 0 & 1 \end{bmatrix}}^{K} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x x_c + s_x z_c \\ f_y y_c + s_y z_c \\ z_c \end{bmatrix}$$

u,v – image coordinates (pixels)

x,y,z – camera coordinates

s_x, s_y – translate the origin

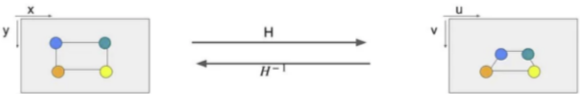
f_x, f_y – scales camera coordinates to image plane

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & s_x \\ 0 & f_y & s_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & s_x \\ 0 & f_y & s_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & T_x \\ R_{21} & R_{22} & T_y \\ R_{31} & R_{32} & T_z \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$H^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



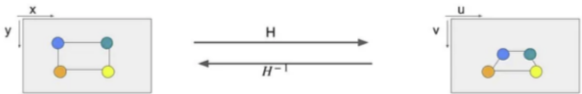
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & s_x \\ 0 & f_y & s_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & s_x \\ 0 & f_y & s_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & T_x \\ R_{21} & R_{22} & T_y \\ R_{31} & R_{32} & T_z \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$H^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Set rotation to a camera in top-down perspective/pose!



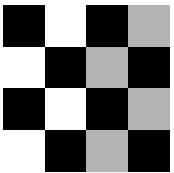
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Homography H (planar projective transformation)

3x3 Homography matrix

Punchline: For planar surfaces, 3D to 2D perspective projection reduces to a 2D to 2D transformation.

Punchline2: This transformation is INVERTIBLE!



pts1 - in image
pts2 - in real-world

SVD, can use: **fitgeotrans** in matlab, or opencv:

```
matrix = cv2.getPerspectiveTransform(pts1, pts2)
result = cv2.warpPerspective(frame, matrix, (500, 600))
```

Parametric Lane Marking Estimation

- Lane marking detection alone is not sufficient (why?)
- Only returns a set of pixels in image or road plane space
- In order to be useful for navigation, this needs to be transformed into a more ***semantically meaningful parametric model***
- Often low-dimensional parametric models (lines, polynomials, bezier curves, splines) are used
  - In the following: splines
- Note 1: This is a multi-model fitting problem!
- Note 2: Outliers must be handled (model must be robust)!

Parametric Lane Marking Estimation

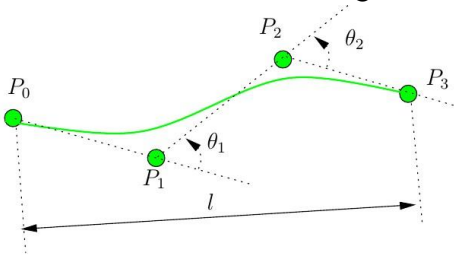


Fig. 7. Spline score computation.

Algorithm 1 RANSAC Spline Fitting

```
for i = 1 to numIterations do
    points=getRandomSample()
    spline=fitSpline(points)
    score=computeSplineScore(spline)
    if score > bestScore then
        bestSpline = spline
    end if
end for
```

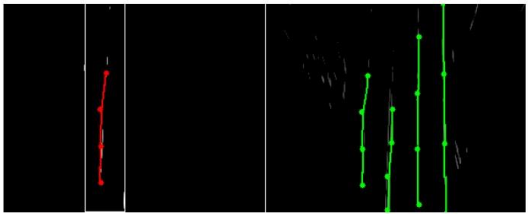


Fig. 8. RANSAC Spline fitting. Left: one of four windows of interest (white) obtained from previous step with detected spline (red). Right: the resulting splines (green) from this step

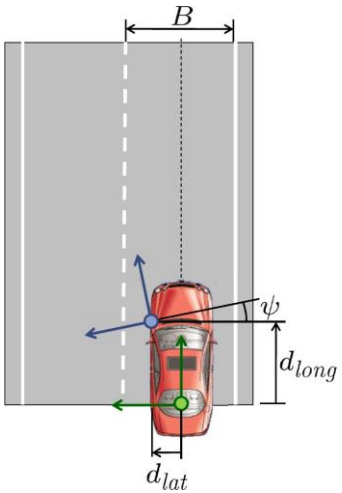
Fig. 10. Post-processing splines. Left: splines before post-processing in blue. Right: splines after post-processing in green. They appear longer and localized on the lanes.

Parametric Lane Marking Estimation

Aly: Real time detection of lane markers in urban streets. IV, 2008.

Task:
Given model for the lanes,
Fit it to observed data points

Model for Straight Lanes



Geometric model: (straight=highway scenario)

- Can we fit a single model for the entire lane?
- Lane width $B \in \mathbb{R}^+$
- Lateral vehicle offset wrt. centerline $d_{lat} \in \mathbb{R}$
- Longitudinal position $d_{long} \in \mathbb{R}$
- Yaw angle $\psi \in [-\pi, \pi]$

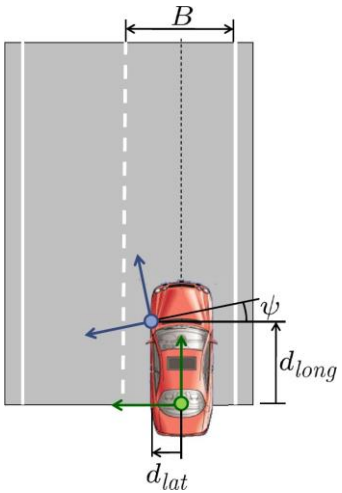
State:

$$\mathbf{s} = \begin{bmatrix} B \\ d_{long} \\ d_{lat} \\ \psi \end{bmatrix}$$

Summary:

- ►Detect lane markings in camera image
- ►Transform position of markings into vehicle coordinates
- ►Estimate pose parameters and lane width

Model for Straight Lanes



Geometric model: (straight=highway scenario)

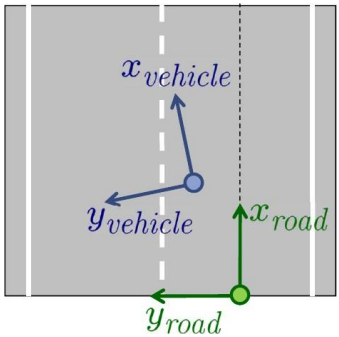
- Can we fit a single model for the entire lane?
- Lane width $B \in \mathbb{R}^+$
- Lateral vehicle offset wrt. centerline $d_{lat} \in \mathbb{R}$
- Longitudinal position $d_{long} \in \mathbb{R}$
- Yaw angle $\psi \in [-\pi, \pi]$

State:

$$\mathbf{s} = \begin{bmatrix} B \\ d_{long} \\ d_{lat} \\ \psi \end{bmatrix}$$

Model for Straight Lanes

Think rotation and translation



Transformation vehicle → road coordinates:

$$\begin{bmatrix} x_{road} \\ y_{road} \end{bmatrix} = \begin{bmatrix} d_{long} \\ d_{lat} \end{bmatrix} + \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix} \begin{bmatrix} x_{vehicle} \\ y_{vehicle} \end{bmatrix}$$

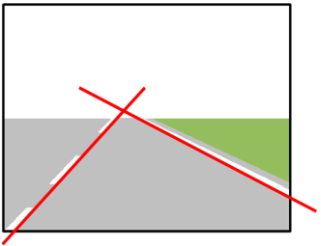
Transformation road → vehicle coordinates:

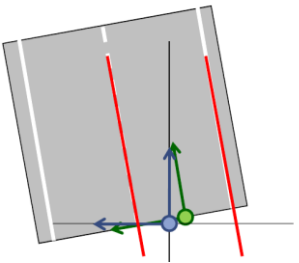
$$\begin{bmatrix} x_{vehicle} \\ y_{vehicle} \end{bmatrix} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} \left(\begin{bmatrix} x_{road} \\ y_{road} \end{bmatrix} - \begin{bmatrix} d_{long} \\ d_{lat} \end{bmatrix} \right)$$

Model for Straight Lanes

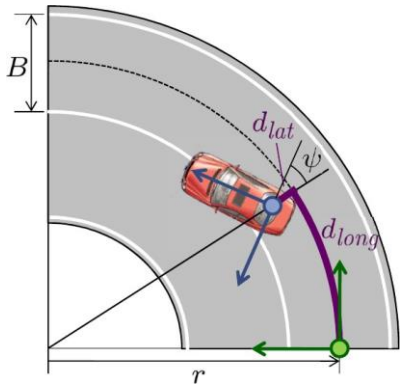
Summary:

- Detect lane markings in camera image
- Transform position of markings into vehicle coordinates
- Estimate pose parameters and lane width





Model for Circular Lanes



Geometric model:

- Lane width $B \in \mathbb{R}^+$
- Signed radius $r \in \mathbb{R}$ or curvature $\kappa = \frac{1}{r}$
- Lateral vehicle offset $d_{lat} \in \mathbb{R}$
- Longitudinal position $d_{long} \in \mathbb{R}$
- Yaw angle $\psi \in [-\pi, \pi]$

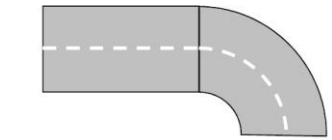
State:

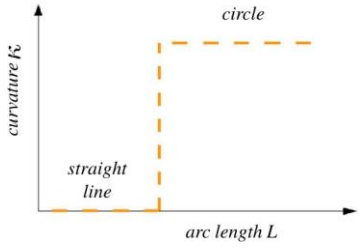
$$\mathbf{s} = \begin{bmatrix} B & r & d_{long} & d_{lat} & \psi \end{bmatrix}^T$$

Circular trajectories are not enough

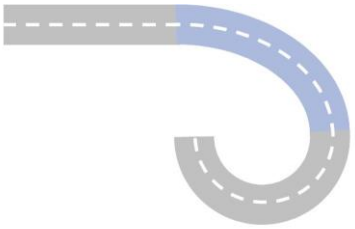
$$y(\ell) \approx \frac{1}{2}\kappa_0\ell^2 + \frac{1}{6}\kappa_1\ell^3$$

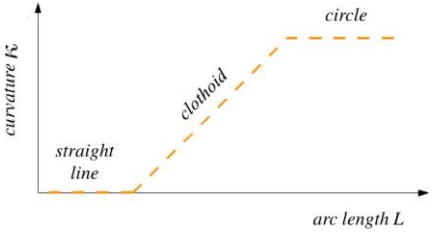
Circle-Straight Combination





Clothoid





Deep Convolutional Image Segmentation

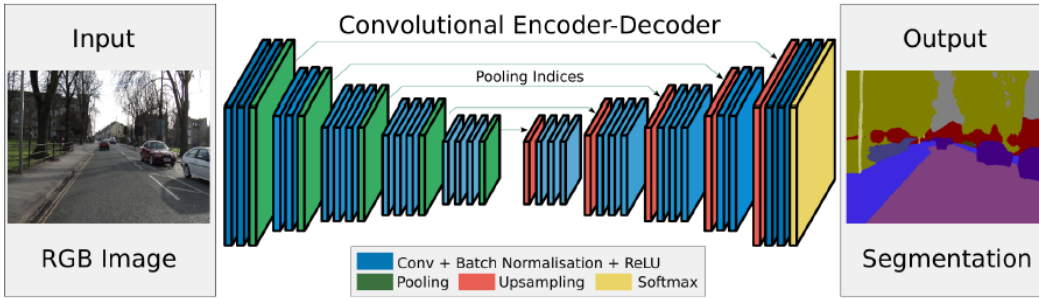


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

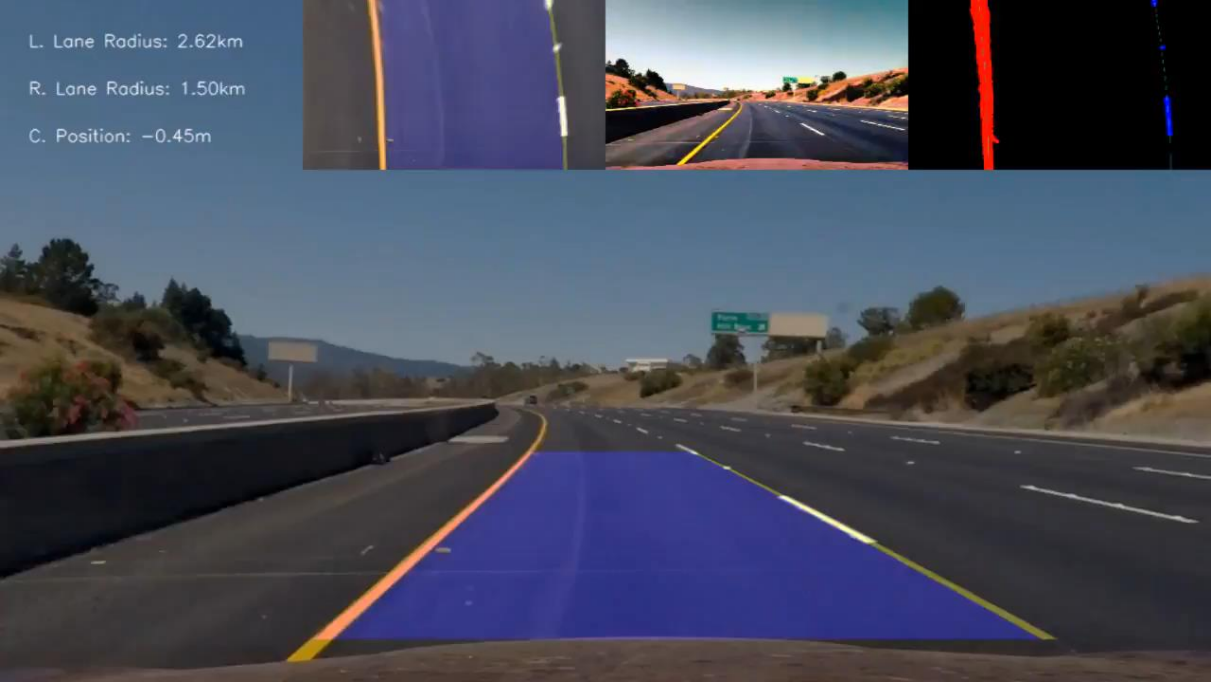
Badrinarayanan, Kendall and Cipolla: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. PAMI, 2017.

Further Readings

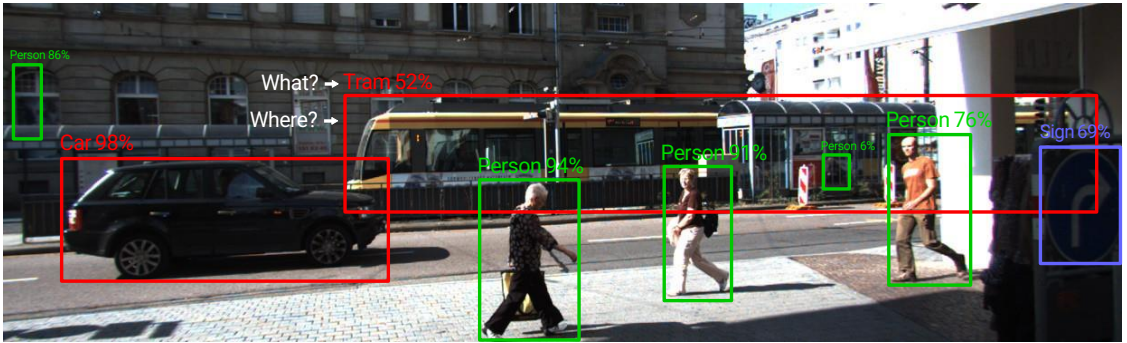
I Aly: Real time detection of lane markers in urban streets. IV, 2008.

I Badrinarayanan, Kendall and Cipolla: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. PAMI, 2017.

I Badino, Franke and Mester: Free Space Computation Using Stochastic Occupancy Grids and Dynamic Programming. ICCV Workshops, 2007.


L. Lane Radius: 2.62km
R. Lane Radius: 1.50km
C. Position: −0.45m

Problem Setting – Object Detection



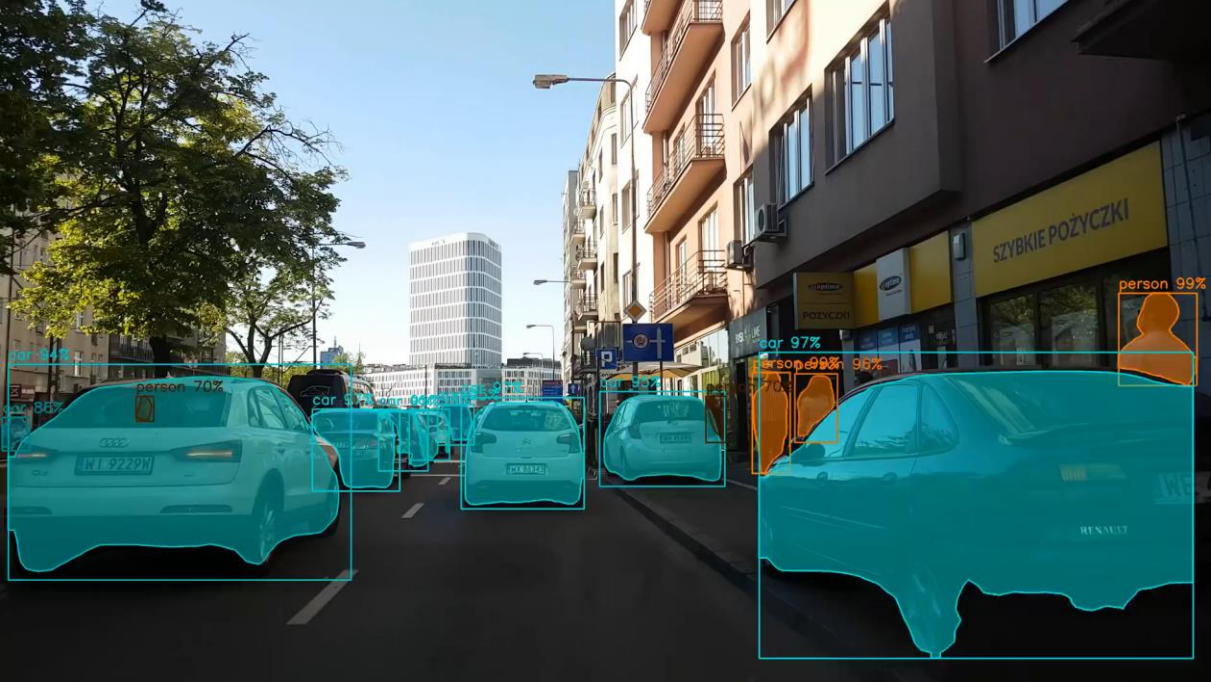
Problem Setting:

- **Input:** RGB Image or laser range scan
- **Output:** Set of 2D/3D bounding boxes with category label and confidence
- Note: Number of objects and object size not known a priori!

3D Object Detection

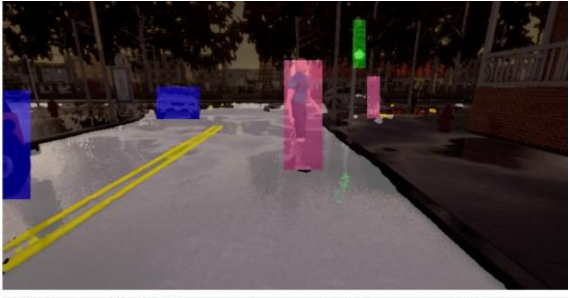
- **Input:** RGB Image or laser range scan
- **Output:** Set of 2D/3D bounding boxes with category label and confidence
- Note: Number of objects and object size not known a priori!


car 1.000
car 0.998
car 0.812
car 0.976
car 0.999
car 0.99
car 0.995
car 1.000
car 0.903
BADGLEY
MISCHKA


SZYBKIE POŻYCZKI
POŻYCZKI
person 99%
car 97%
person 96%
person 70%
RENAULT

Visual Abstractions – What are most relevant segmentation classes for learning a driving policy?

Trained with 6400 finely annotated images and 14 classes
Annotation time ≈ 7500 hours, policy success rate = 50%



Trained with 1600 coarsely annotated images and 6 classes
Annotation time ≈ 50 hours, policy success rate = 58%

Behl, Chitta, Prakash, Ohn-Bar and Geiger: Label Efficient Visual Abstractions for Autonomous Driving. IROS, 2020.

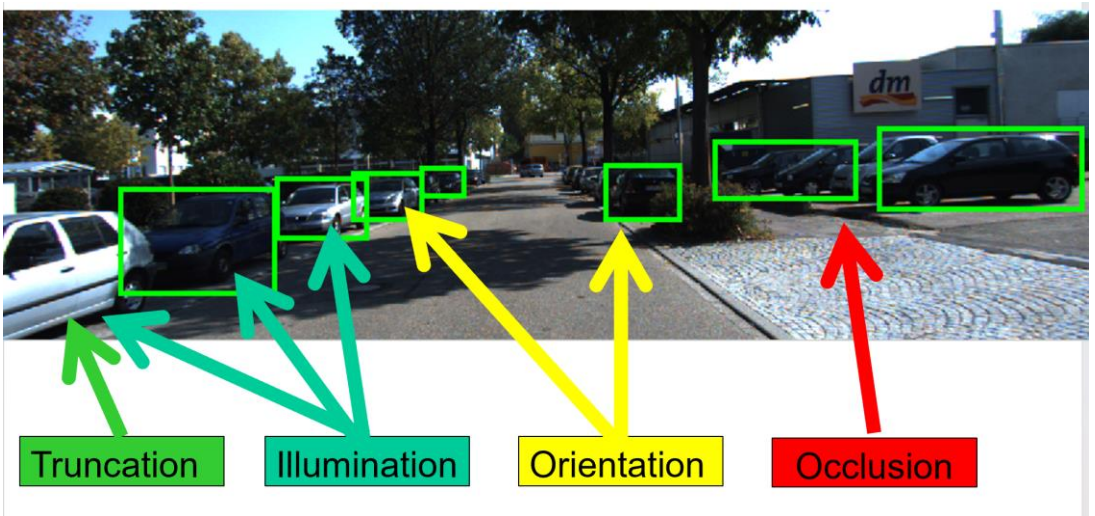
Why is detection hard(er)?

Precise localization

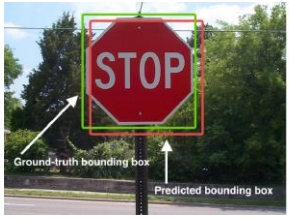
Why is detection hard(er)?

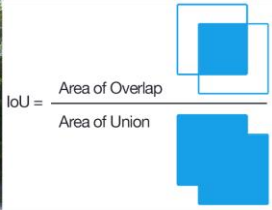
Small objects

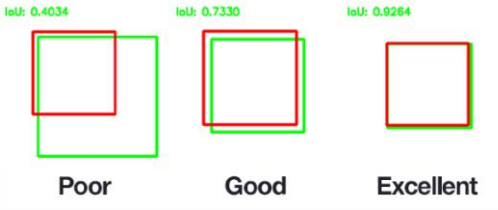
Object Detection Challenges



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 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!)

Matching detections to ground truth

- Match detection to most similar ground truth
  - highest IoU
- If IoU > 50%, mark as correct
- If multiple detections map to same ground truth, mark only one as correct
- **Precision** = #correct detections / total detections
- **Recall** = #ground truth with matched detections / total ground truth

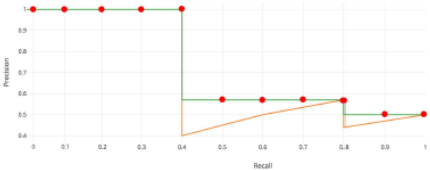
Tradeoff between precision and recall

- ML usually gives scores or probabilities, so threshold
- Too low threshold → too many detections → low precision, high recall
- Too high threshold → too 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

How to Measure Detection Performance?

Average Precision Metric:

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



True Positives TP: Number of objects correctly detected (IoU>0.5)

False Negatives FN: Number of objects not detected (IoU<0.5)

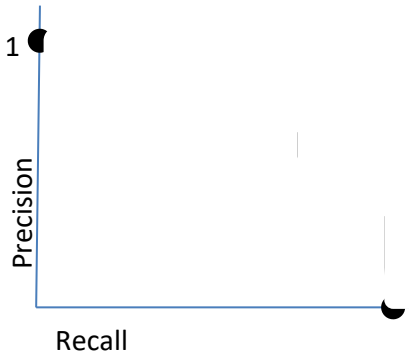
False Positives FP: Wrong detections

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

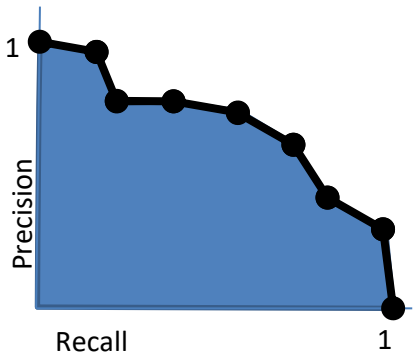
$$\text{Recall } R = \frac{TP}{TP+FN}$$

$$\text{Avg. Prec. } AP = \frac{1}{11} \sum_{R \in \{0,\ldots,1\}} \max_{R' \geq R} P(R')$$

Average precision



Average precision



https://www.cvlibs.net/datasets/kitti/

Object Detection and Orientation Estimation Evaluation

Cars

	Method	Setting	Code	Moderate	Easy	Hard	Runtime	Environment	Compare
1	VoCo			**97.11 %**	98.25 %	**94.46 %**	0.1 s	1 core @ 2.5 Ghz (Python + C/C++)	☐
2	VirConv-S			96.46 %	96.99 %	93.74 %	0.09 s	1 core @ 2.5 Ghz (C/C++)	☐
3	GraR-Vol		code	96.29 %	96.81 %	91.06 %	0.07 s	1 core @ 2.5 Ghz (Python + C/C++)	☐
	H. Yang, Z. Liu, X. Wu, W. Wang, W. Qian, X. He and D. Cai: Graph R-CNN: Towards Accurate 3D Object Detection with Semantic-Decorated Local Graph. ECCV 2022.								
4	GraR-Po		code	96.09 %	96.83 %	90.99 %	0.06 s	1 core @ 2.5 Ghz (Python + C/C++)	☐
	H. Yang, Z. Liu, X. Wu, W. Wang, W. Qian, X. He and D. Cai: Graph R-CNN: Towards Accurate 3D Object Detection with Semantic-Decorated Local Graph. ECCV 2022.								
5	NIV-SSD			96.06 %	96.89 %	88.63 %	0.03 s	1 core @ 2.5 Ghz (C/C++)	☐
6	SFD		code	96.05 %	**98.95 %**	90.96 %	0.1 s	1 core @ 2.5 Ghz (Python + C/C++)	☐
	X. Wu, L. Peng, H. Yang, L. Xie, C. Huang, C. Deng, H. Liu and D. Cai: Sparse Fuse Dense: Towards High Quality 3D Detection with Depth Completion. CVPR 2022.								
7	VPFNet		code	96.04 %	96.63 %	90.99 %	0.06 s	2 cores @ 2.5 Ghz (Python)	☐
	H. Zhu, J. Deng, Y. Zhang, J. Ji, Q. Mao, H. Li and Y. Zhang: VPFNet: Improving 3D Object Detection with Virtual Point based LiDAR and Stereo Data Fusion. IEEE Transactions on Multimedia 2022.								
8	VirConv-T			96.01 %	98.64 %	93.12 %	0.09 s	1 core @ 2.5 Ghz (C/C++)	☐
9	CASDC			95.97 %	98.64 %	92.99 %	0.1 s	1 core @ 2.5 Ghz (Python)	☐
10	TED		code	95.96 %	96.63 %	93.24 %	0.1 s	1 core @ 2.5 Ghz (C/C++)	☐
	H. Wu, C. Wen, W. Li, R. Yang and C. Wang: Transformation-Equivariant 3D Object Detection for Autonomous Driving. AAAI 2023.								
11	RDIoU		code	95.95 %	98.77 %	90.90 %	0.03 s	1 core @ 2.5 Ghz (Python + C/C++)	☐
	H. Sheng, S. Cai, N. Zhao, B. Deng, J. Huang, X. Hua, M. Zhao and G. Lee: Rethinking IoU-based Optimization for Single- stage 3D Object Detection. ECCV 2022.								
12	ACF-Net			95.95 %	96.64 %	93.17 %	n/a s	1 core @ 2.5 Ghz (C/C++)	☐
13	CLOCs		code	95.93 %	96.77 %	90.93 %	0.1 s	1 core @ 2.5 Ghz (Python)	☐
	S. Pang, D. Morris and H. Radha: CLOCs: Camera-LiDAR Object Candidates Fusion for 3D Object Detection . 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2020.								

Good old days

OC-DPM			64.42 %	73.50 %	52.40 %	10 s	8 cores @ 2.5 Ghz (Matlab)

ik, M. Stark, P. Gehler and B. Schiele: Occlusion Patterns for Object Class Detection. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2013.

LSVM-MDPM-sv			55.77 %	67.27 %	43.59 %	10 s	4 cores @ 3.0 Ghz (C/C++)

ger, C. Wojek and R. Urtasun: Joint 3D Estimation of Objects and Scene Layout. NIPS 2011.

enszwalb, R. Girshick, D. McAllester and D. Ramanan: Object Detection with Discriminatively Trained Part-Based Models. PAMI 2010.

DPM-C8B1			50.32 %	59.51 %	39.22 %	15 s	4 cores @ 2.5 Ghz (Matlab + C/C++)

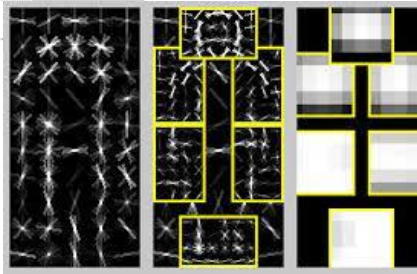
nous submission

AOG			36.87 %	44.41 %	30.29 %	3 s	4 cores @ 2.5 Ghz (Matlab)

nous submission

SVM-Res			30.38 %	35.02 %	24.87 %	10 s	4 cores @ 2.5 Ghz (Matlab)

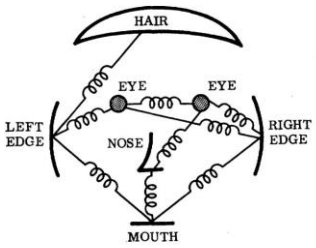
nous submission

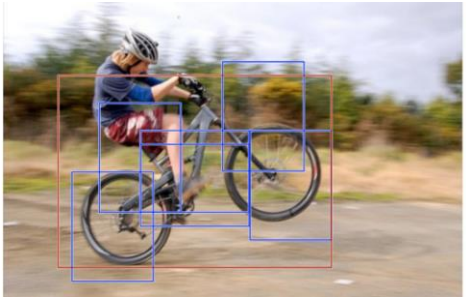


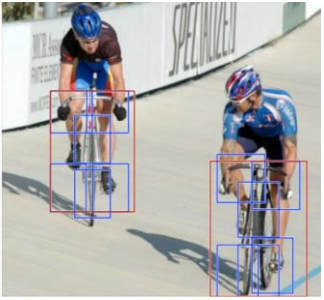
Part Based Models

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

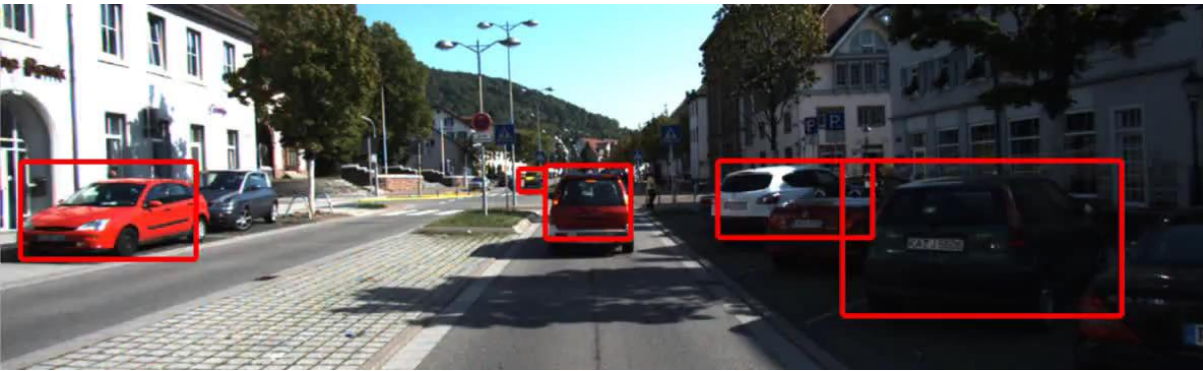






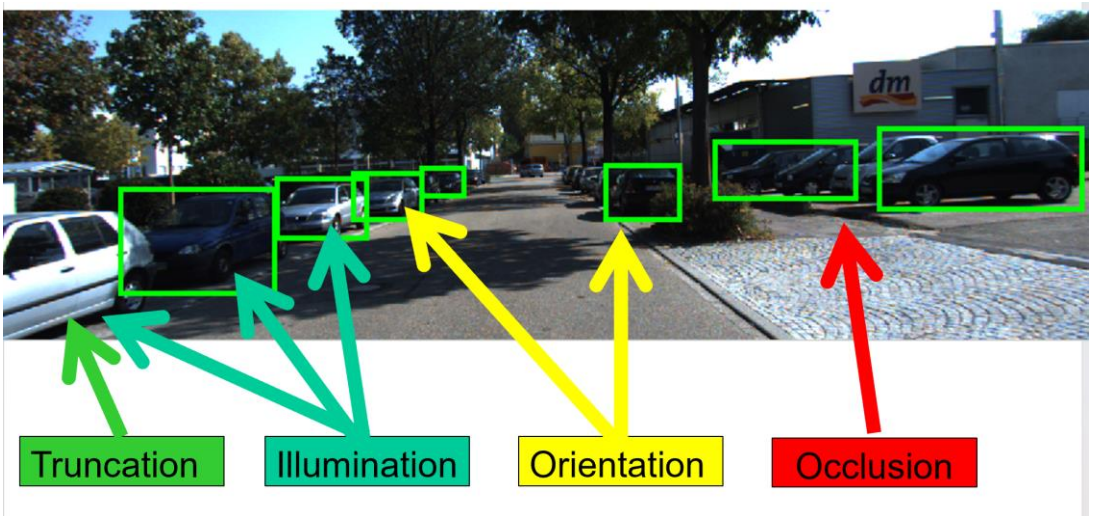
Object Detection with Discriminatively Trained Part Based Models. Felzenszwalb, Girshick, McAllester and Ramanan. PAMI, 2010.

Results on KITTI

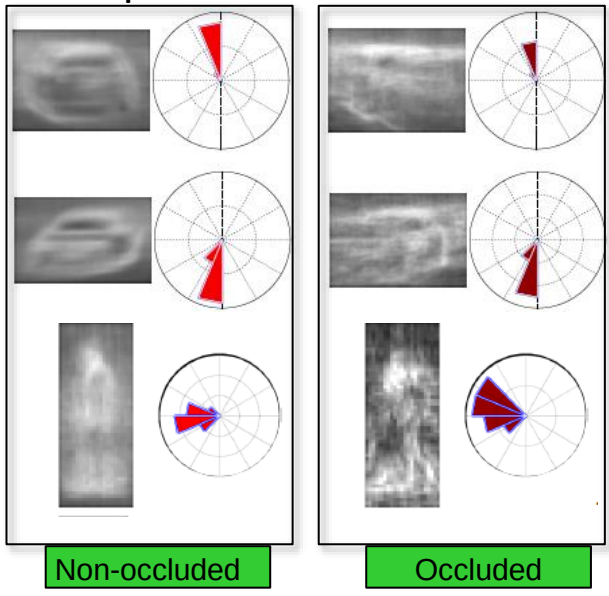


Object Detection with Discriminatively Trained Part Based Models. Felzenszwalb, Girshick, McAllester and Ramanan. PAMI, 2010.

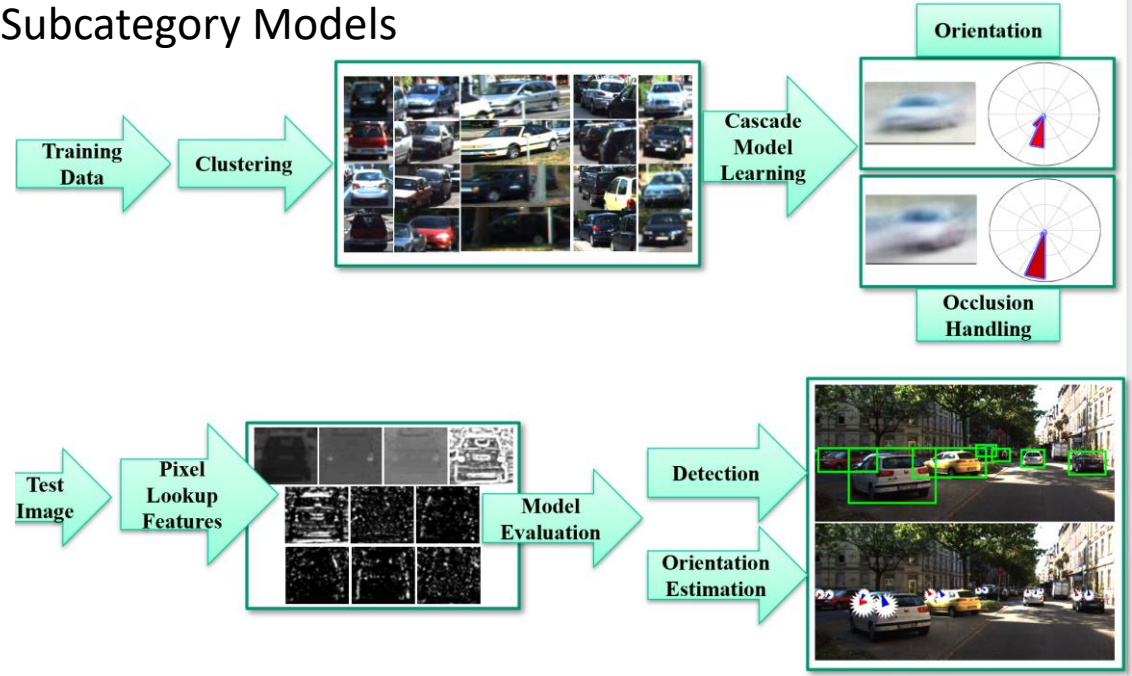
Object Detection Challenges



Key Idea: Learn Multiple Models



Subcategory Models



Object Detection and Orientation Estimation Evaluation

Cars

Rank	Method	Setting	Code	Moderate	Easy	Hard	Runtime	Environment	Compare
1	SubCat			64.94 %	80.92 %	50.03 %	0.3 s	6 cores @ 2.5 Ghz (Matlab + C/C++)	

E. Ohn-Bar and M. Trivedi: Learning to Detect Objects at Multiple Orientations and Occlusion Levels. Under submission 2014.

Rank	Method	Setting	Code	Moderate	Easy	Hard	Runtime	Environment	Compare
2	OC-DPM			64.42 %	73.50 %	52.40 %	10 s	8 cores @ 2.5 Ghz (Matlab)	

B. Pepik, M. Stark, P. Gehler and B. Schiele: Occlusion Patterns for Object Class Detection. IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2013.

Rank	Method	Setting	Code	Moderate	Easy	Hard	Runtime	Environment	Compare
3	LSVM-MDPM-sv			55.77 %	67.27 %	43.59 %	10 s	4 cores @ 3.0 Ghz (C/C++)	

A. Geiger, C. Wojek and R. Urtasun: Joint 3D Estimation of Objects and Scene Layout. NIPS 2011.

P. Felzenszwalb, R. Girshick, D. McAllester and D. Ramanan: Object Detection with Discriminatively Trained Part-Based Models. PAMI 2010.

Rank	Method	Setting	Code	Moderate	Easy	Hard	Runtime	Environment	Compare
4	DPM-C8B1			50.32 %	59.51 %	39.22 %	15 s	4 cores @ 2.5 Ghz (Matlab + C/C++)	

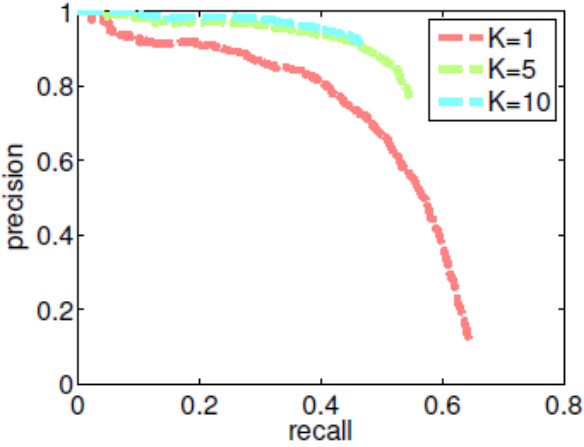
Anonymous submission

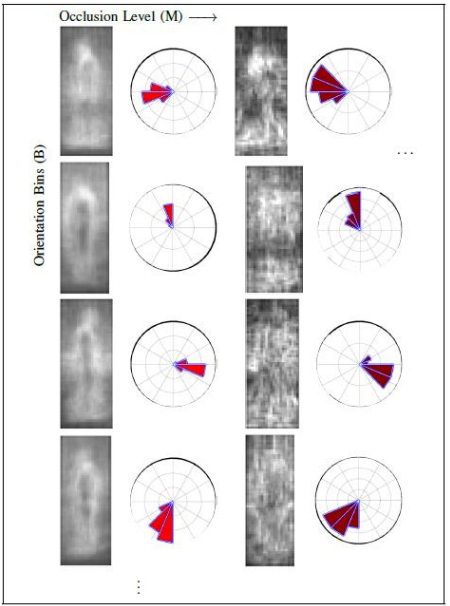
Rank	Method	Setting	Code	Moderate	Easy	Hard	Runtime	Environment	Compare
5	AOG			36.87 %	44.41 %	30.29 %	3 s	4 cores @ 2.5 Ghz (Matlab)	

Anonymous submission

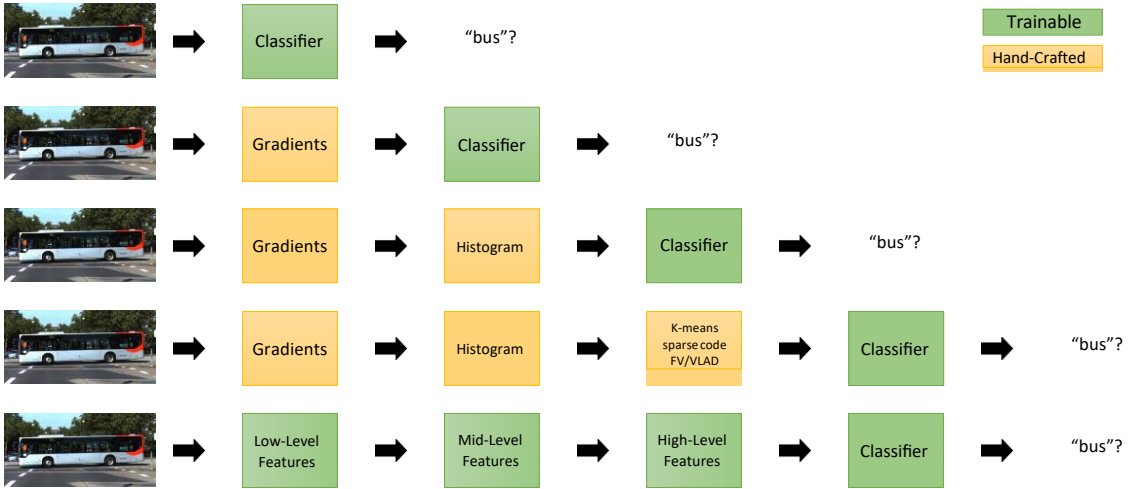
Rank	Method	Setting	Code	Moderate	Easy	Hard	Runtime	Environment	Compare
6	SVM-Res			30.38 %	35.02 %	24.87 %	10 s	4 cores @ 2.5 Ghz (Matlab)	

Anonymous submission


precision
recall
K=1
K=5
K=10
0
0.2
0.4
0.6
0.8
1

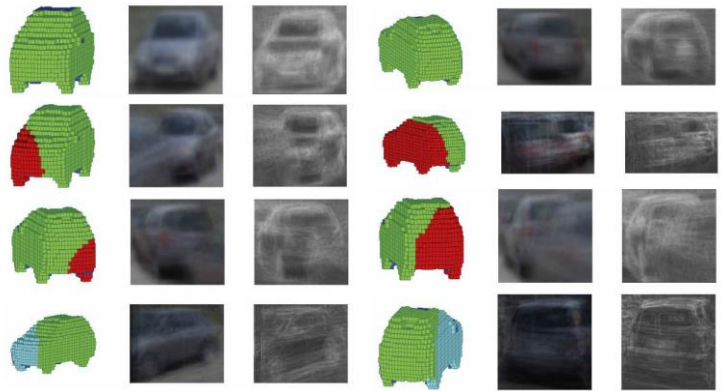

Occlusion Level (M) ⟶
Orientation Bins (B)

Hand-crafted Representations vs. Learned Representations



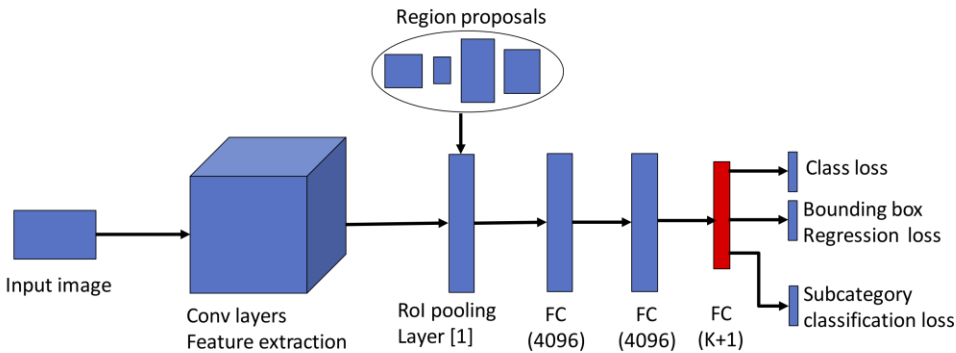
Data-Driven 3D Voxel Patterns for Object Category Recognition, Xiang et al., CVPR 2015

Cluster objects with similar 3D pose, occlusion and truncation.



Subcategory-aware Convolutional Neural Networks for Object Proposals and Detection, WACV 2017

Subcategory-aware Detection Network



[1] R. Girshick. Fast R-CNN. ICCV, 2015.

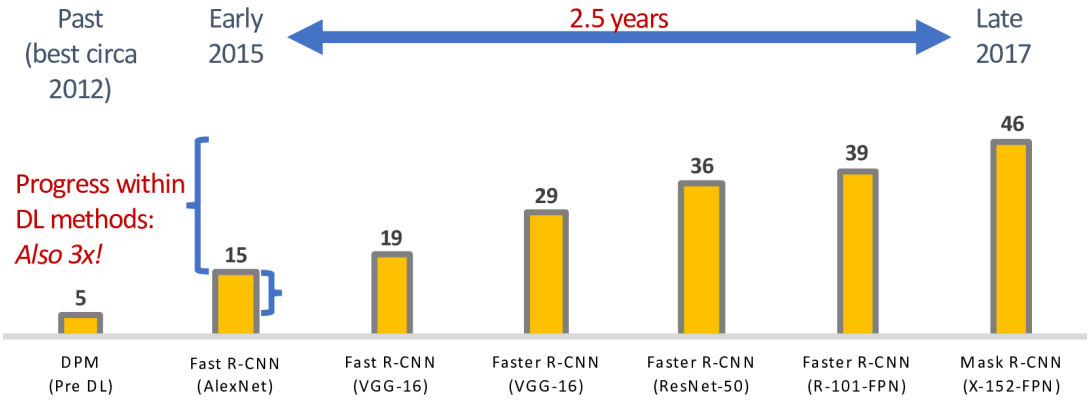
Car Detection and Orientation Estimation on KITTI

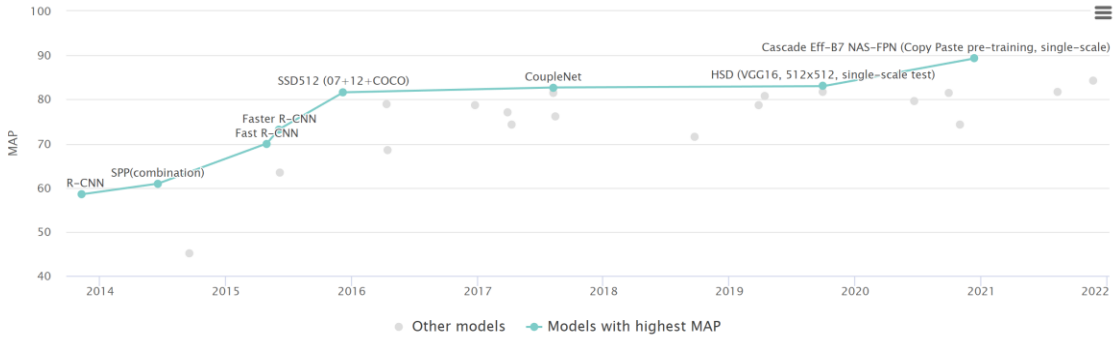
	Object Detection (AP)			Object Detection and Orientation estimation (AOS)		
Method	**Easy**	**Moderate**	**Hard**	**Easy**	**Moderate**	**Hard**
ACF [1]	55.89	54.77	42.98	N/A	N/A	N/A
DPM-VOC+VP [2]	74.95	64.71	48.76	72.28	61.84	46.54
OC-DPM [3]	74.94	65.95	53.86	73.50	64.42	52.40
SubCat [4]	84.14	75.46	59.71	83.41	74.42	58.83
Regionlets [5]	84.75	76.45	59.70	N/A	N/A	N/A
3DVP [6]	84.81	73.02	63.22	84.31	71.99	62.11
3DOP [7]	**93.04**	88.64	79.10	**91.44**	86.10	76.52
Mono3D [8]	92.33	88.66	78.96	91.01	86.62	76.84
SDP+RPN [9]	90.14	88.85	78.38	N/A	N/A	N/A
MS-CNN [10]	90.03	89.02	76.11	N/A	N/A	N/A
Ours SubCNN	90.81	**89.04**	**79.27**	90.67	**88.62**	**78.68**

[1] P. Dolla´r, R. Appel, S. Belongie, and P. Perona. Fast feature pyramids for object detection. TPAMI, 2014.
[2] B. Pepik, M. Stark, P. Gehler, and B. Schiele. Multi-view and 3d deformable part models. TPAMI, 2015.
[3] B. Pepikj, M. Stark, P. Gehler, and B. Schiele. Occlusion patterns for object class detection. In CVPR, 2013.
[4] E. Ohn-Bar and M. M. Trivedi. Learning to detect vehicles by clustering appearance patterns. T-ITS, 2015.
[5] X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In ICCV, 2013.
[6] Y. Xiang, W. Choi, Y. Lin, and S. Savarese. Data-driven 3d voxel patterns for object category recognition. In CVPR, 2015.
[7] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun. 3d object proposals for accurate object class detection. In NIPS, 2015.
[8] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, R. Urtasun. Monocular 3D Object Detection for Autonomous Driving, in CVPR, 2016.
[9] F. Yang, W. Choi, and Y. Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In CVPR, 2016.
[10] Z. Cai, Q. Fan, R. Feris, and N. Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In ECCV, 2016.

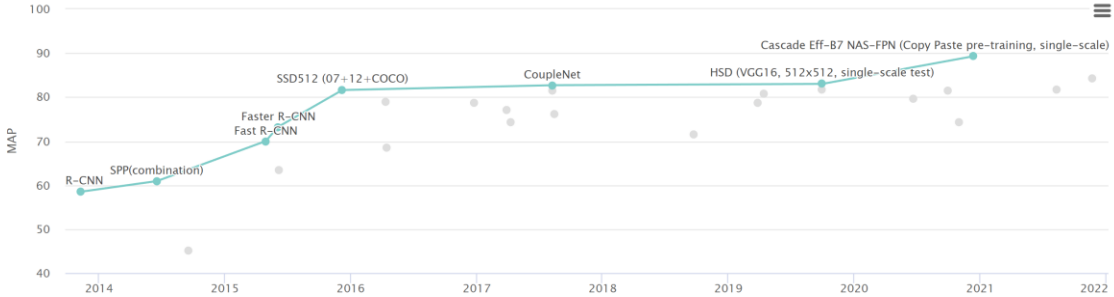
Detection: Rank 2
Pose : Rank 4

Hand-crafted Representations vs. Learned Representations





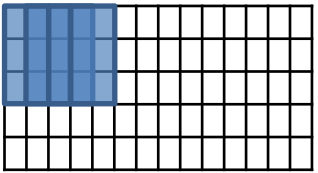
25	**R-CNN**	58.5%	Rich feature hierarchies for accurate object detection and semantic segmentation		2013
26	**Deformable Parts Model** (DeepPyramid)	45.2%	Deformable Part Models are Convolutional Neural Networks		2014



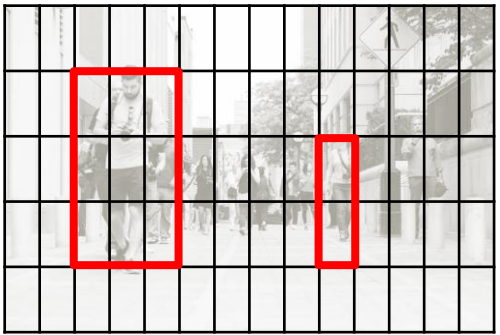
18	**FRCN**	74.2%	A-Fast-RCNN: Hard Positive Generation via Adversary for Object Detection	2017	
19	**Faster R-CNN**	73.2%	Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks	2015	
20	**VGG-16 + KL Loss + var voting + soft-NMS**	71.6%	Bounding Box Regression with Uncertainty for Accurate Object Detection	2018	
21	**Fast R-CNN**	70.0%	Fast R-CNN	2015	
22	**subCNN**	68.5%	Subcategory-aware Convolutional Neural Networks for Object Proposals and Detection	2016	
23	**YOLO**	63.4%	You Only Look Once: Unified, Real-Time Object Detection	2015	YOLO
24	**SPP** (combination)	60.9%	Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition	2014	
25	**R-CNN**	58.5%	Rich feature hierarchies for accurate object detection and semantic segmentation	2013	
26	**Deformable Parts Model** (DeepPyramid)	45.2%	Deformable Part Models are Convolutional Neural Networks	2014	

Idea 1: scanning window

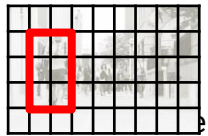
- Fix size
  - Can take a few different sizes
- Fixed stride
- Convolution with a filter
  - Classic: compute HOG/CNN features over entire image



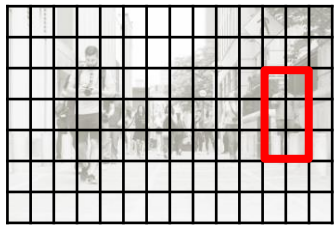
Dealing with scale



Dealing with scale



e window size, but run on *image pyramid*

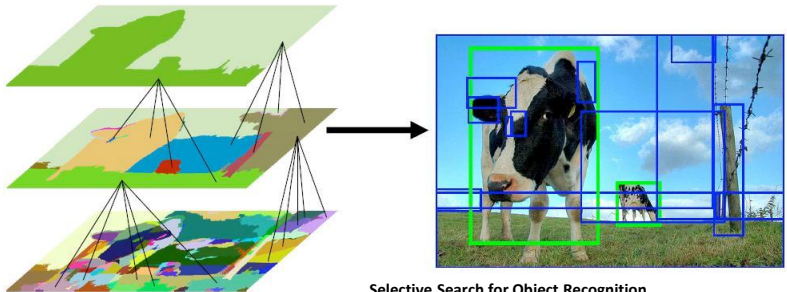


Issues – Large Sampling Space

Run through every possible box and classify – over 10^10 for 300x500 images

Idea 2: Object proposals

Use segmentation to produce ~5K candidates

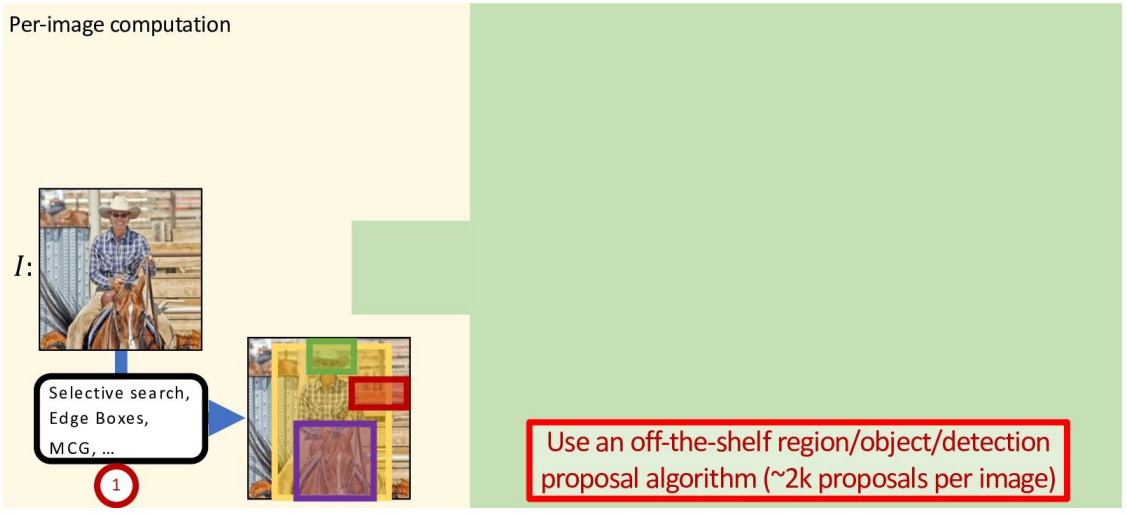


Selective Search for Object Recognition
J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, A. W. M. Smeulders
In International Journal of Computer Vision 2013.

What do we do with proposals?

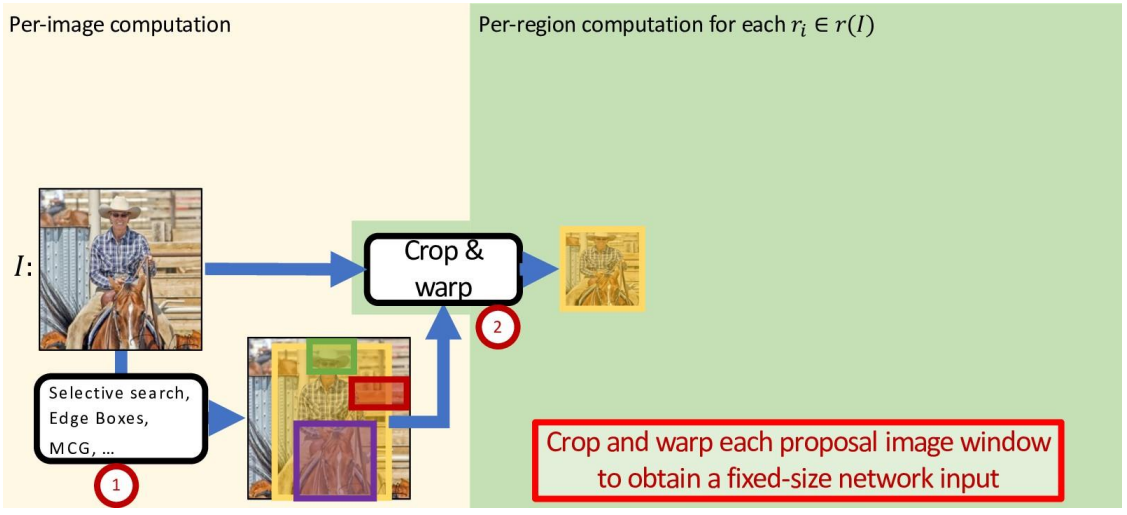
Each proposal is a group of pixels
Take tight fitting box and *classify it*
Can leverage any image classification approach

R-CNN: Region-based Convolutional Neural Network



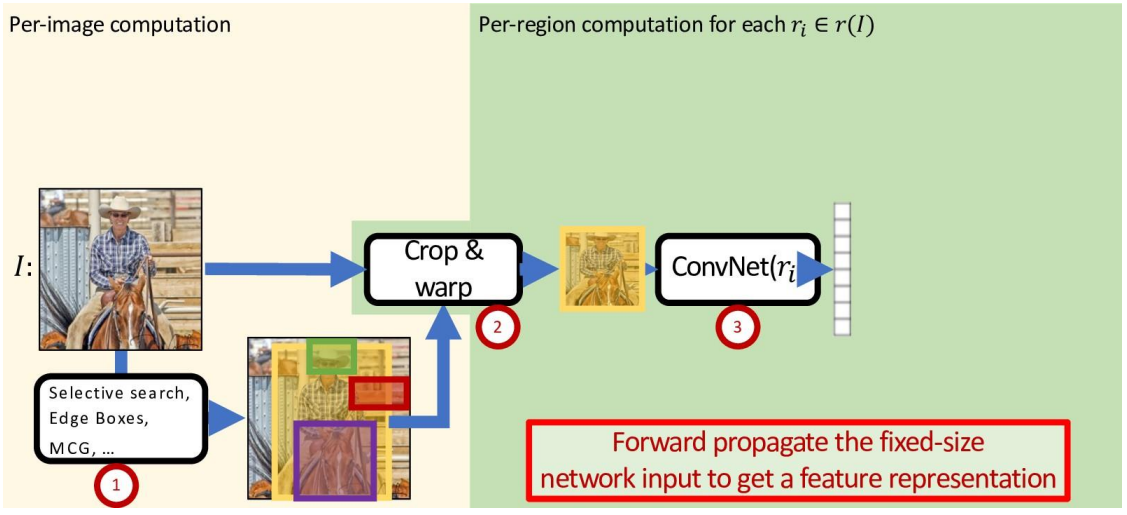
Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN: Region-based Convolutional Neural Network



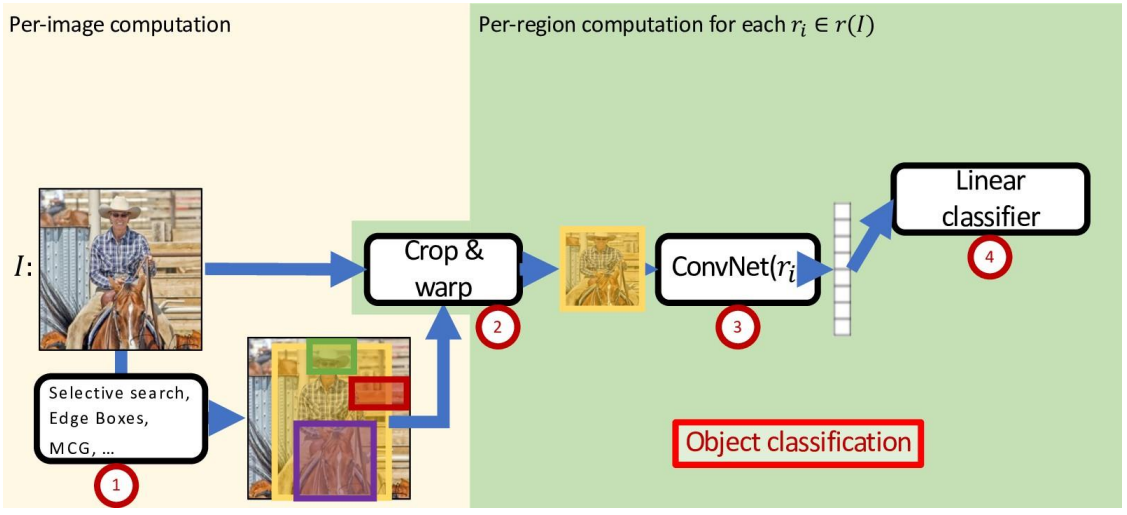
Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN: Region-based Convolutional Neural Network



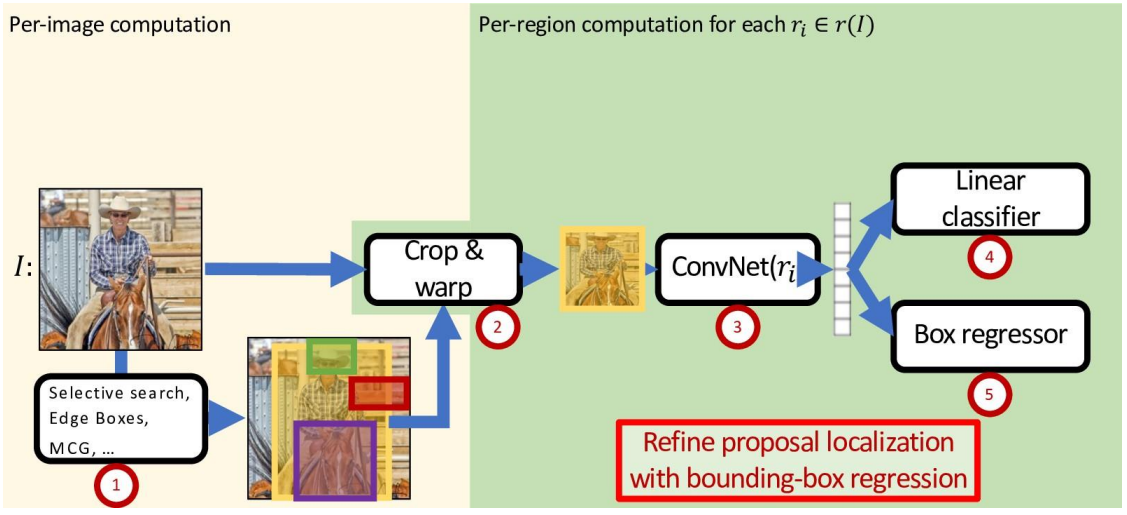
Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN: Region-based Convolutional Neural Network



Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN: Region-based Convolutional Neural Network



Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

Generalized Framework

Per-image computation

I:

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

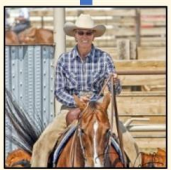
Input image
per-image operations | per-region operations

Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

Generalized Framework

Per-image computation

$f_I = f(I)$



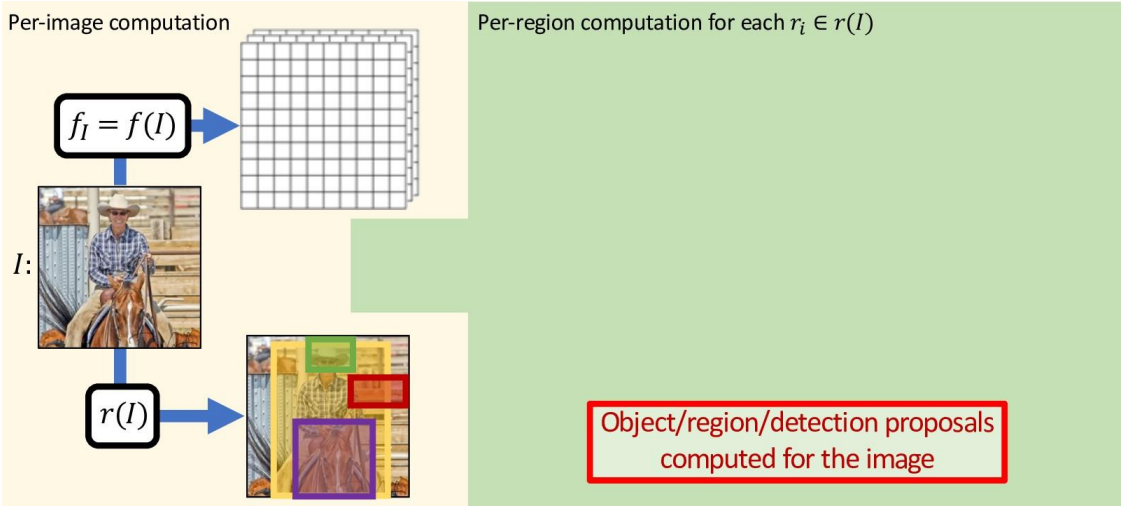
I:

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

Transformation of the input image into a featurized representation

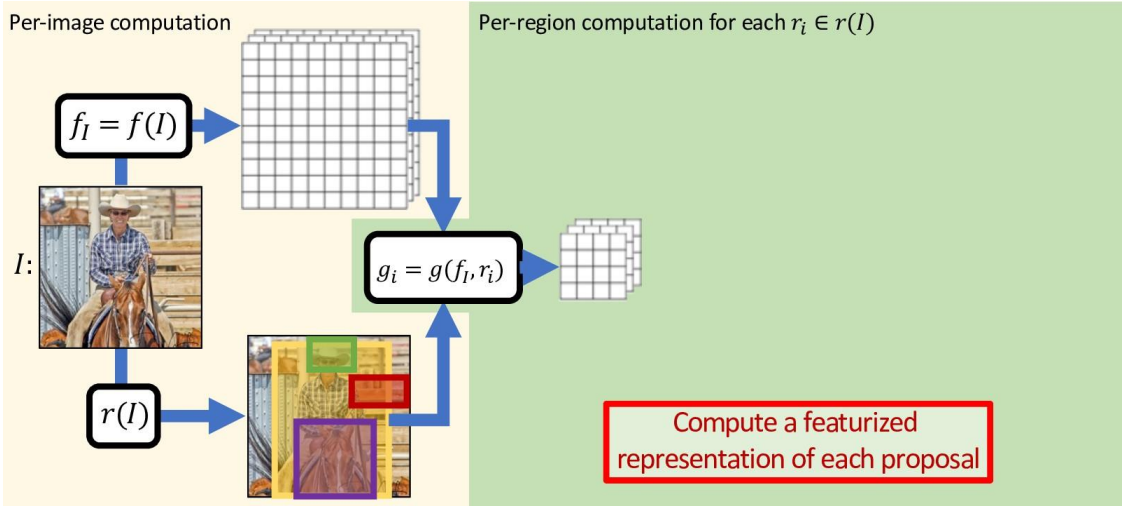
Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

Generalized Framework



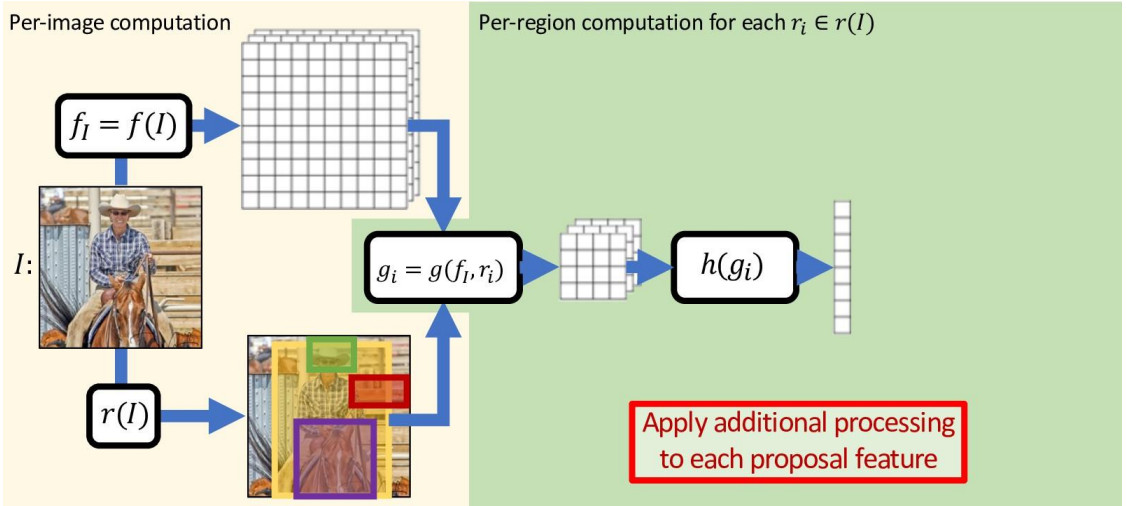
Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

Generalized Framework



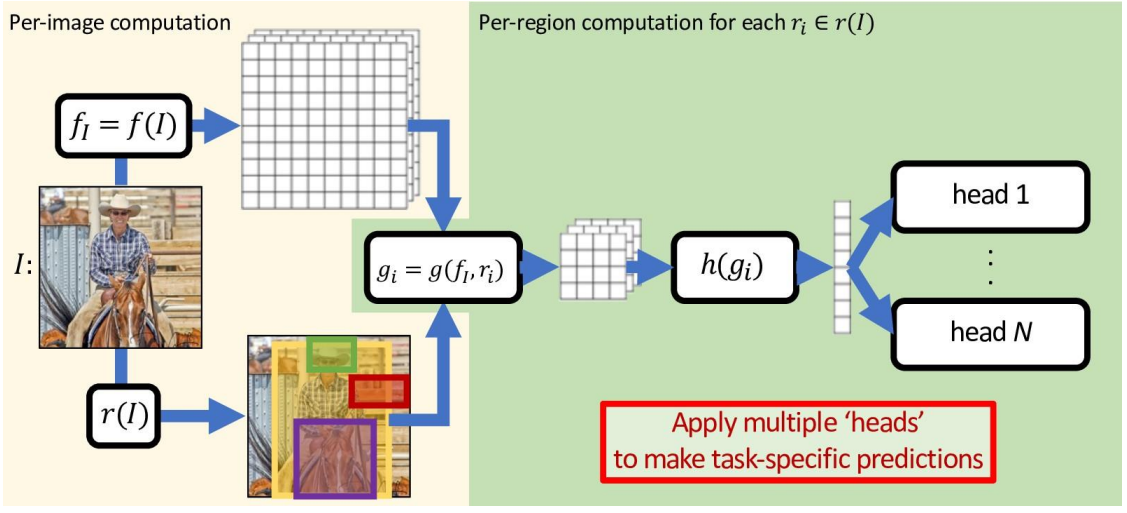
Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

Generalized Framework



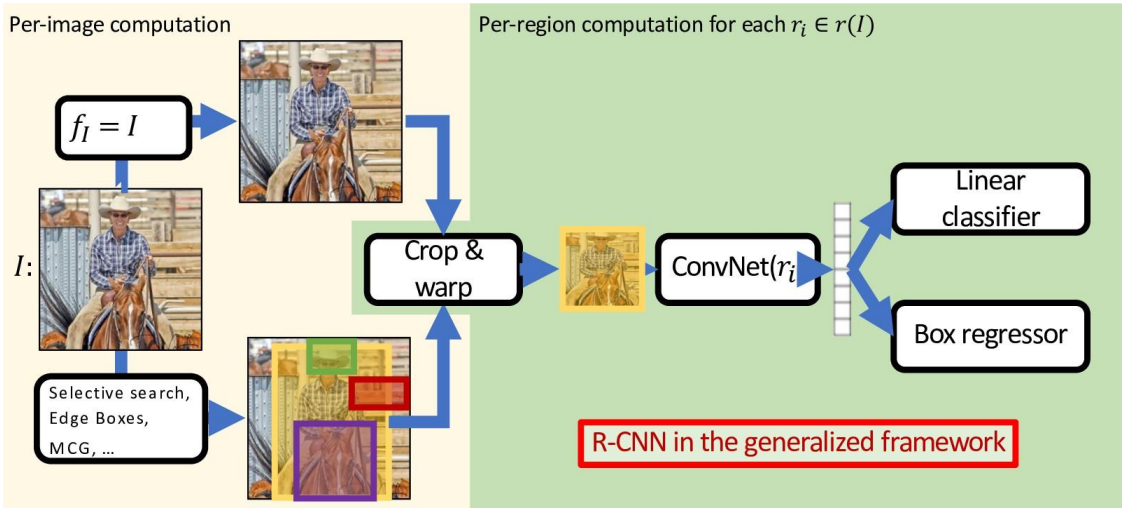
Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

Generalized Framework



Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

R-CNN in Generalized Framework



Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

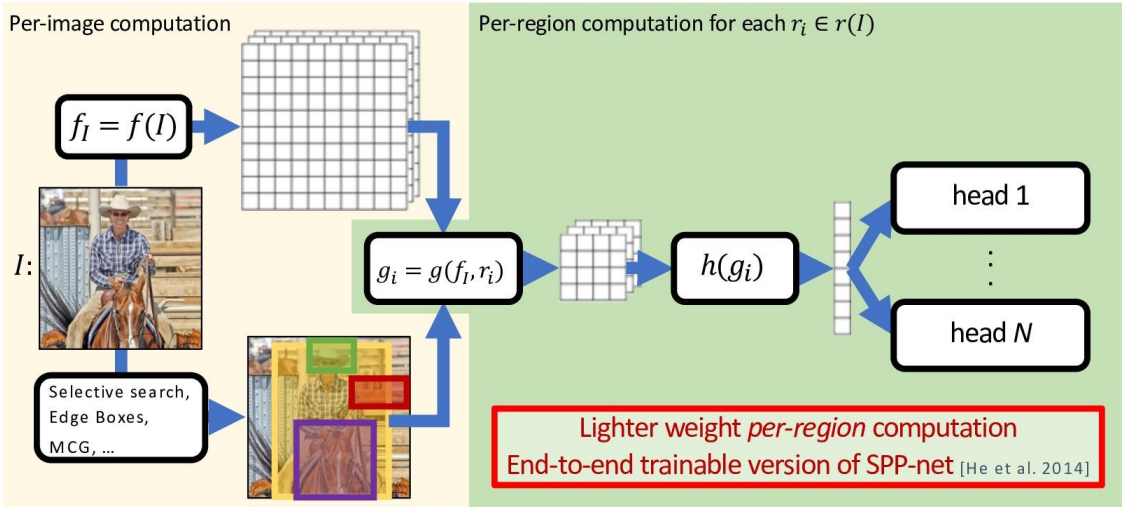
R-CNN in Generalized Framework

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

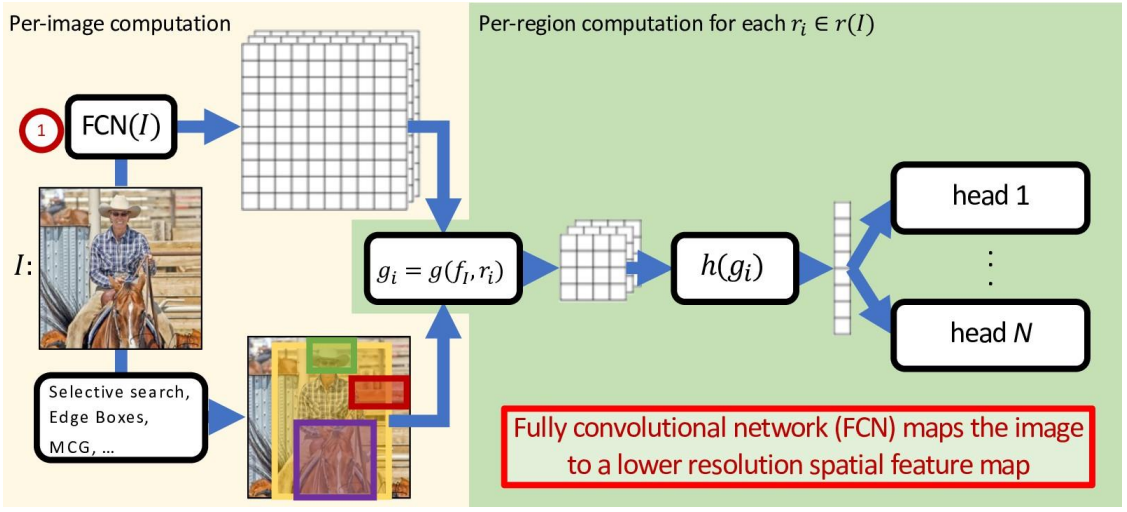
Girshick, Donahue, Darrell and Malik: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR, 2014.

Fast R-CNN



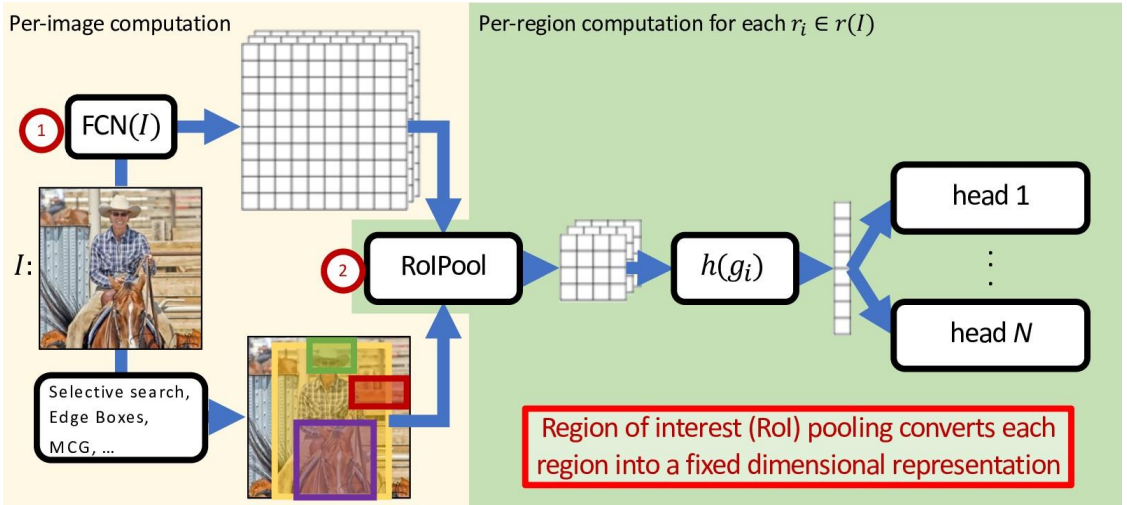
Girshick: Fast R-CNN. ICCV, 2015.

Fast R-CNN



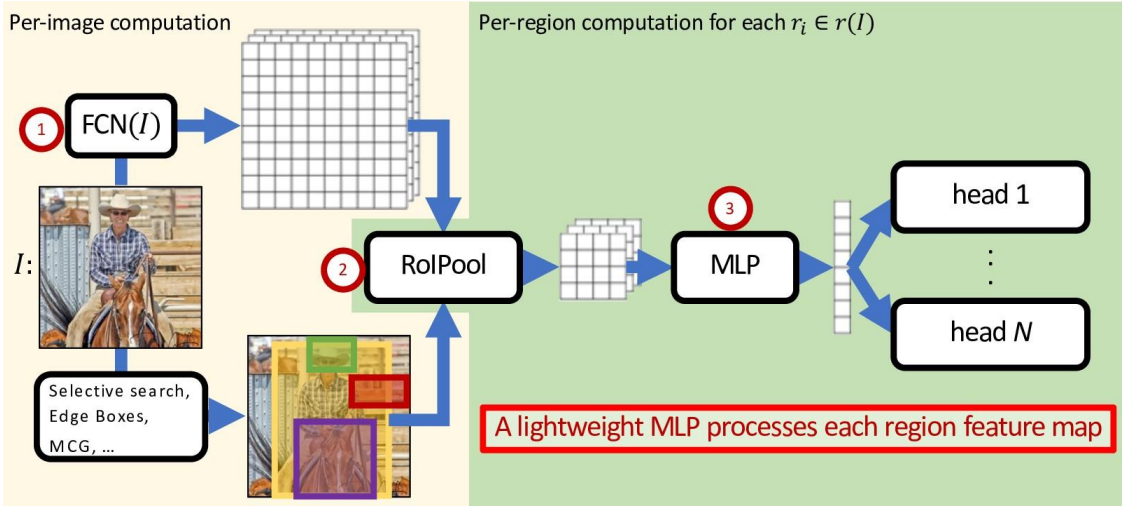
Girshick: Fast R-CNN. ICCV, 2015.

Fast R-CNN



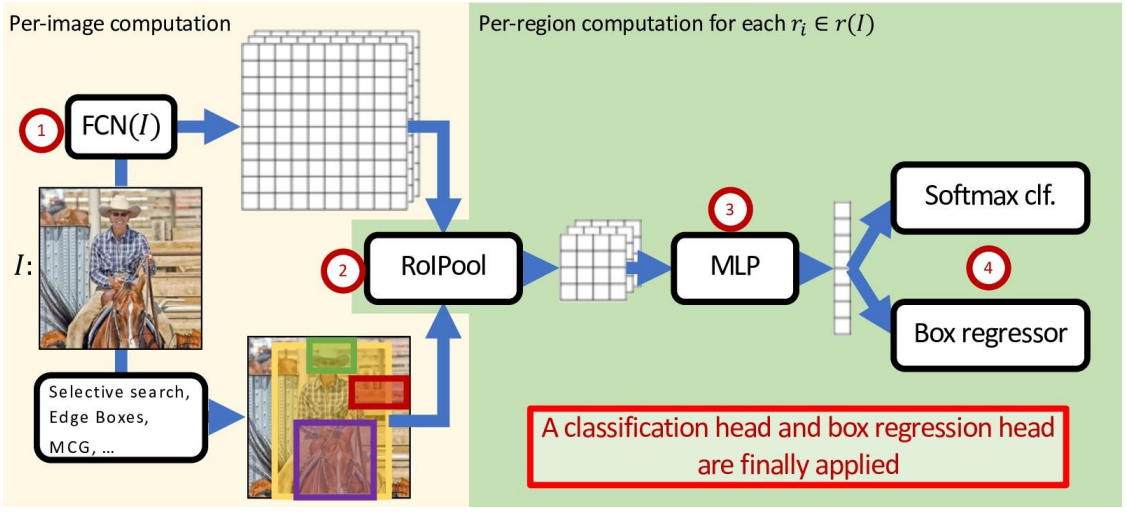
Girshick: Fast R-CNN. ICCV, 2015.

Fast R-CNN



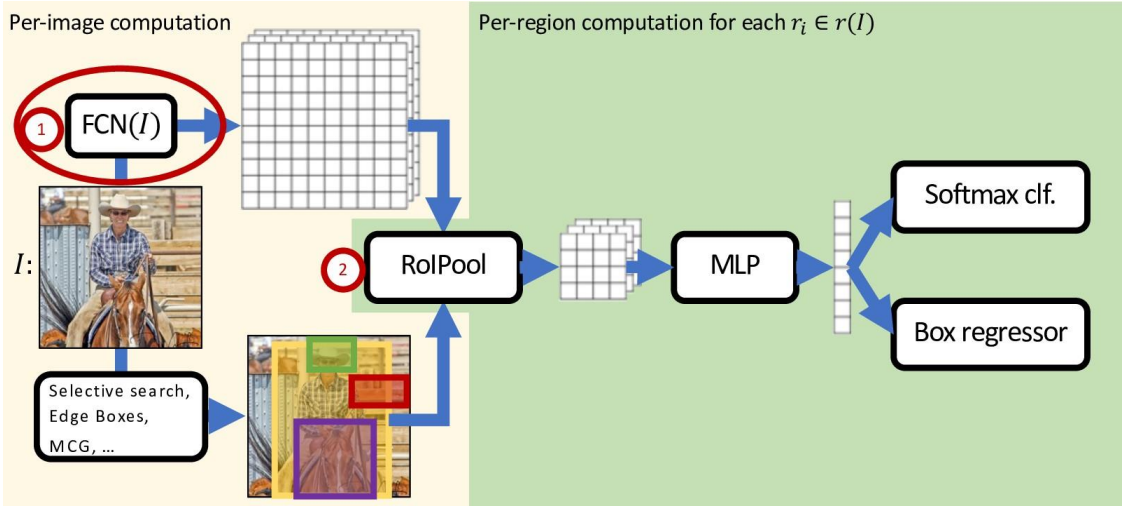
Girshick: Fast R-CNN. ICCV, 2015.

Fast R-CNN



Girshick: Fast R-CNN. ICCV, 2015.

Fast R-CNN

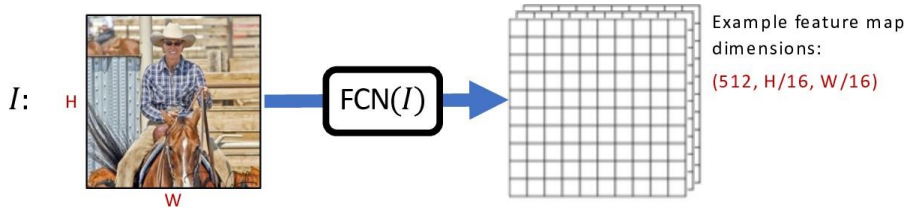


Girshick: Fast R-CNN. ICCV, 2015.

Fast R-CNN

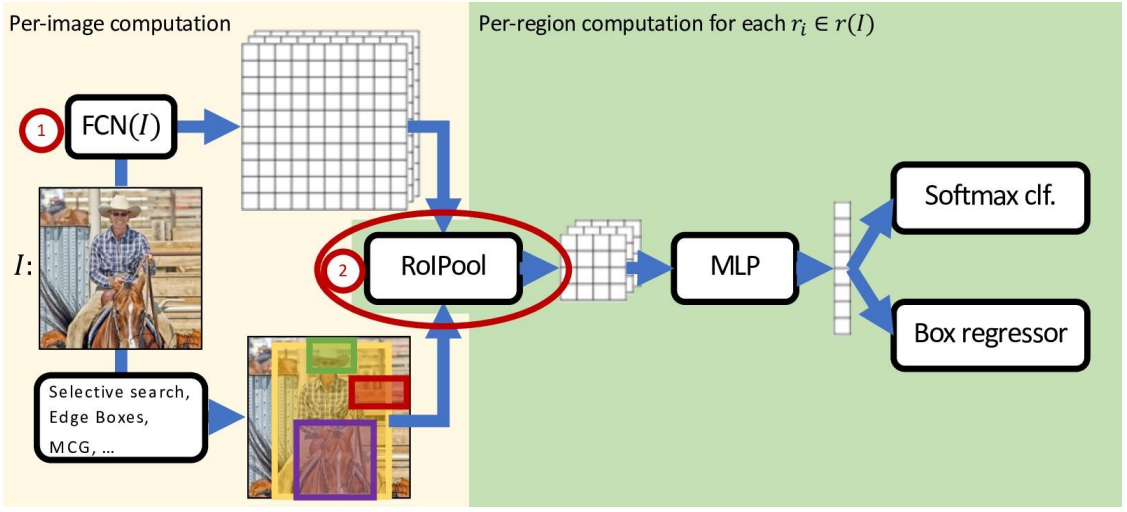
Backbone:

- 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!)



Girshick: Fast R-CNN. ICCV, 2015.

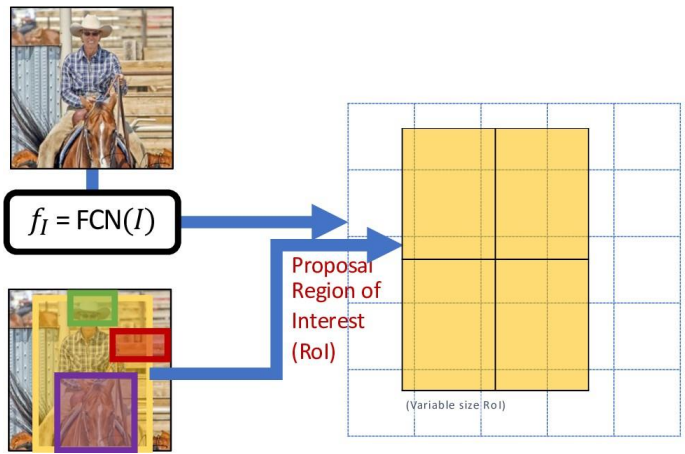
Fast R-CNN



Girshick: Fast R-CNN. ICCV, 2015.

Fast R-CNN

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

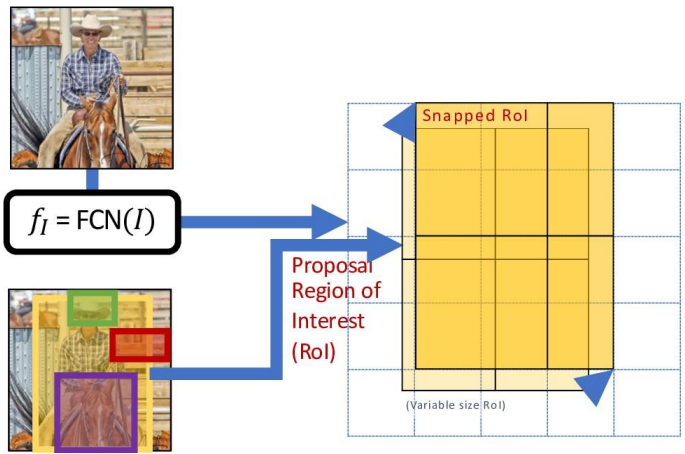


Key innovation in SPP-net
[He et al. 2014]

Girshick: Fast R-CNN. ICCV, 2015.

Fast R-CNN

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

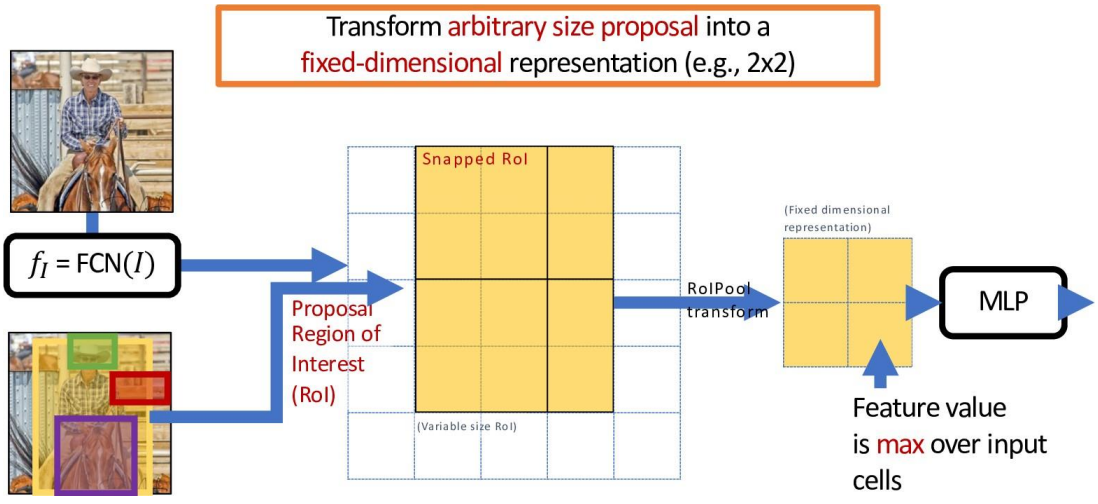


Key innovation in SPP-net
[He et al. 2014]

Girshick: Fast R-CNN. ICCV, 2015.

Fast R-CNN

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



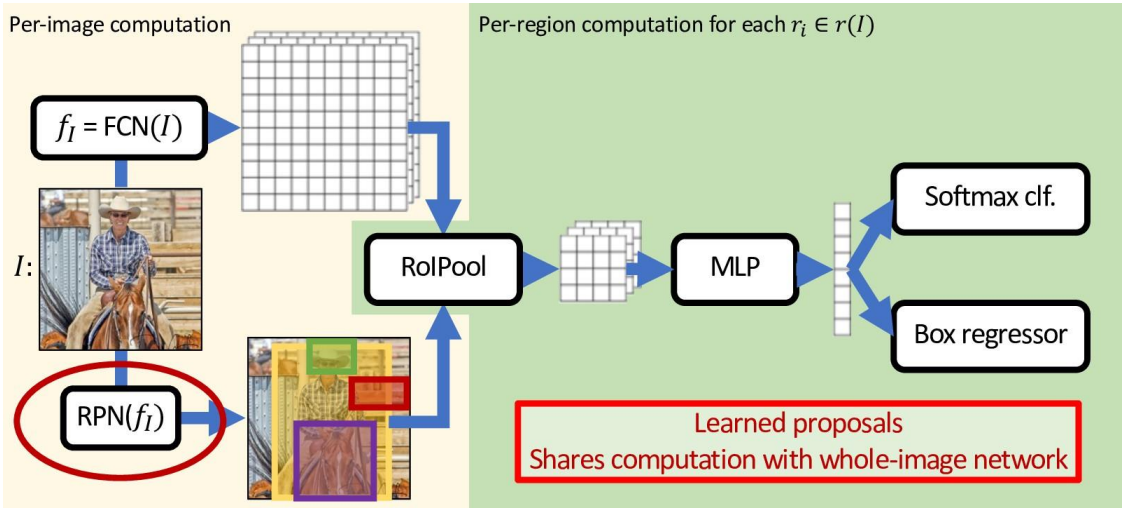
Girshick: Fast R-CNN. ICCV, 2015.

Fast R-CNN

What is the problem with Fast R-CNN?

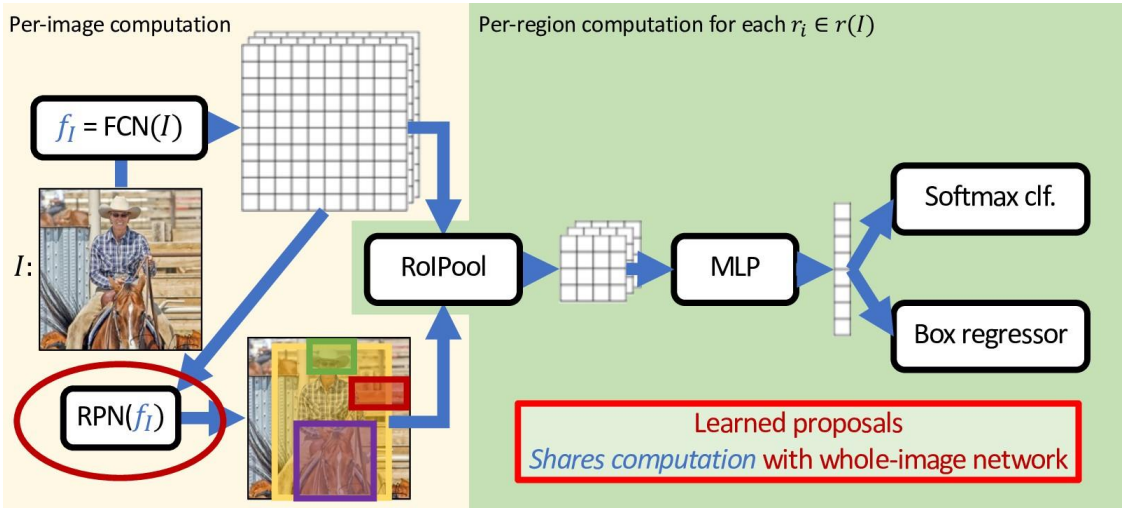
- ~~Heavy per-region computation (e.g., 2000 full network evaluations)~~
- ~~No computation/feature sharing~~
- Slow region proposal method adds to runtime
- Generic region proposal techniques have very low recall

Faster R-CNN



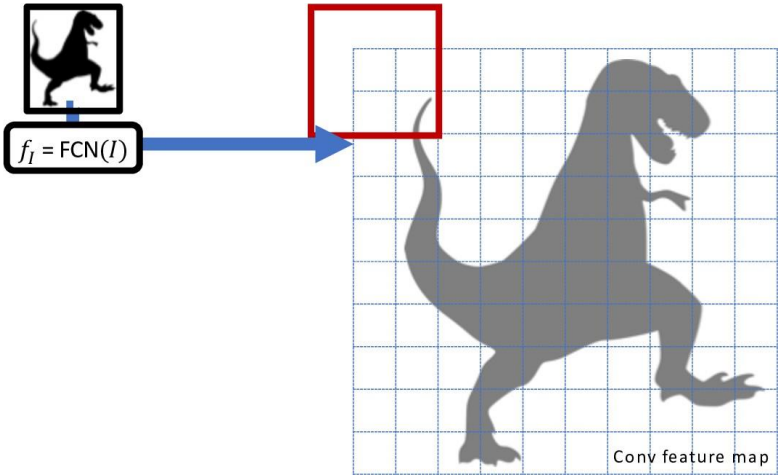
Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

Faster R-CNN



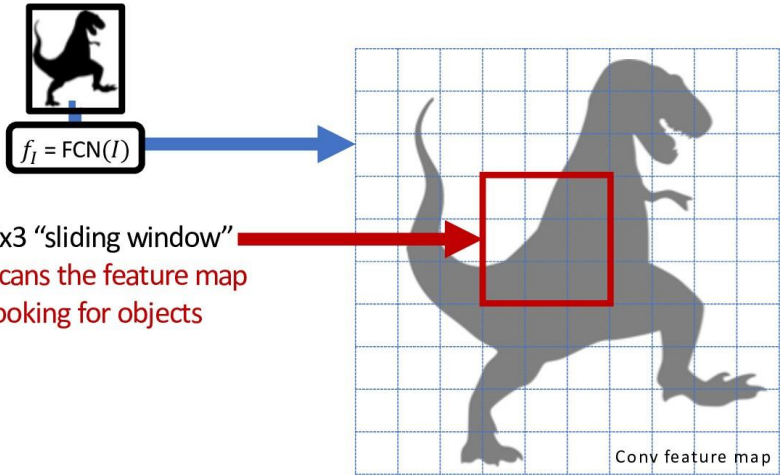
Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



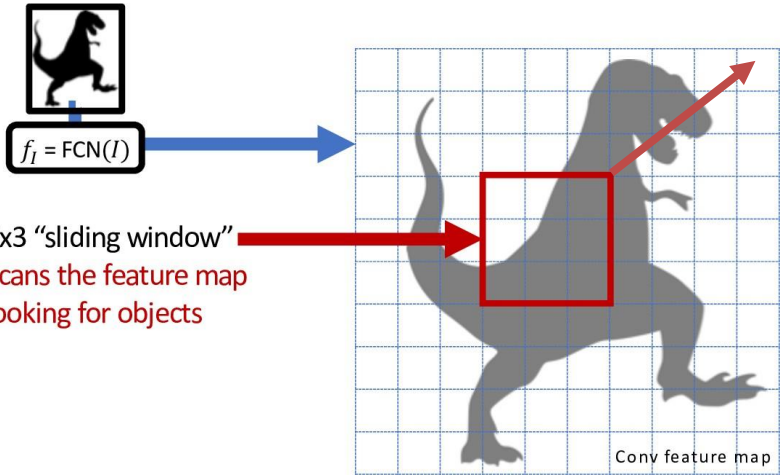
Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

Regression from Red Box to Full Object is Difficult!



Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)

f_I = FCN(I)

Anchor box: predictions are w.r.t. this box, *not the 3x3 sliding window*

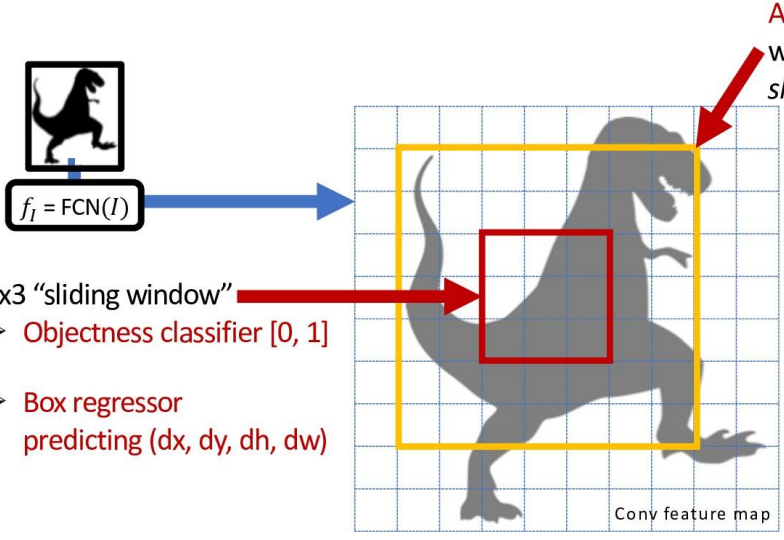
3x3 "sliding window"

Scans the feature map looking for objects

Conv feature map

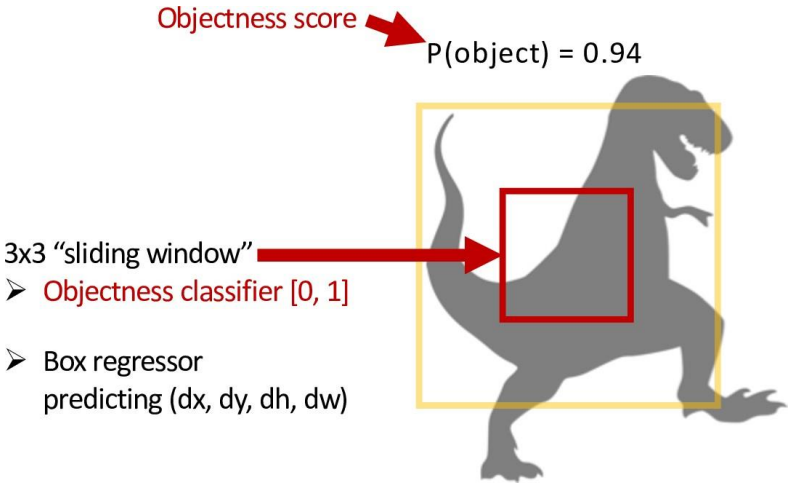
Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



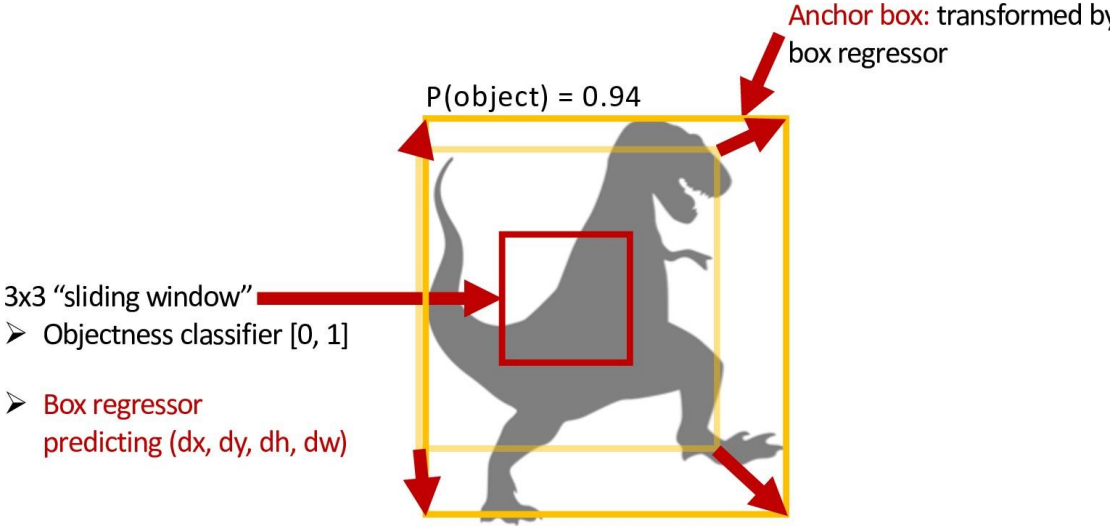
Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



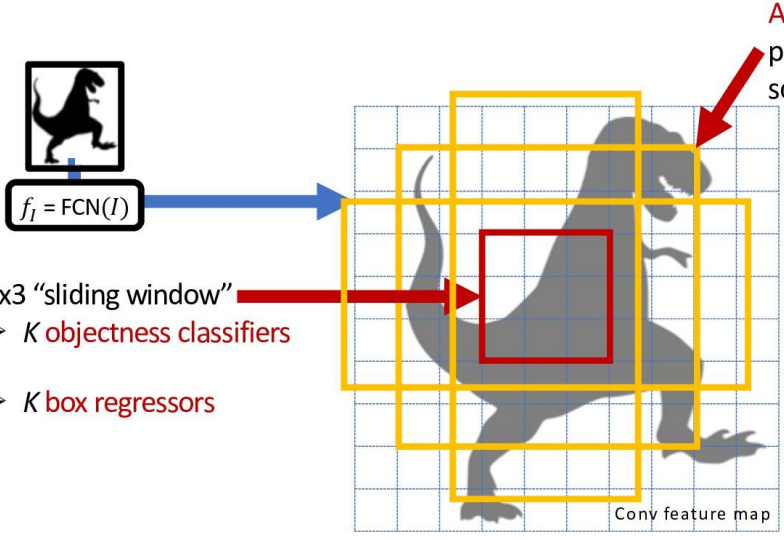
Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



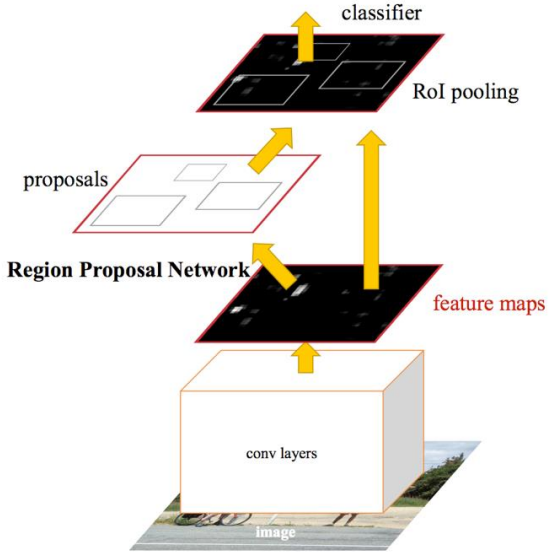
Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

Faster R-CNN: Region Proposal Network (RPN)



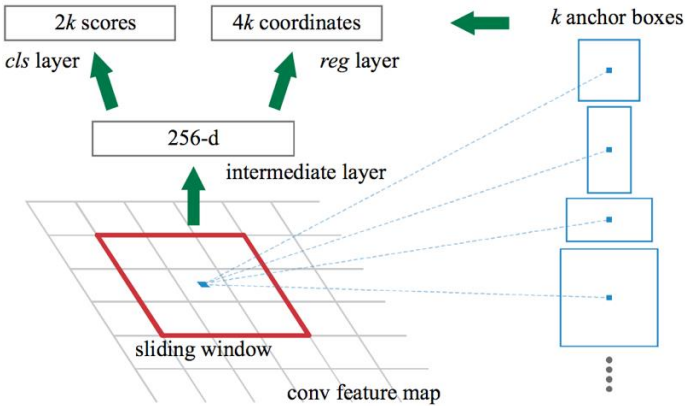
Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS, 2015.

Faster R-CNN

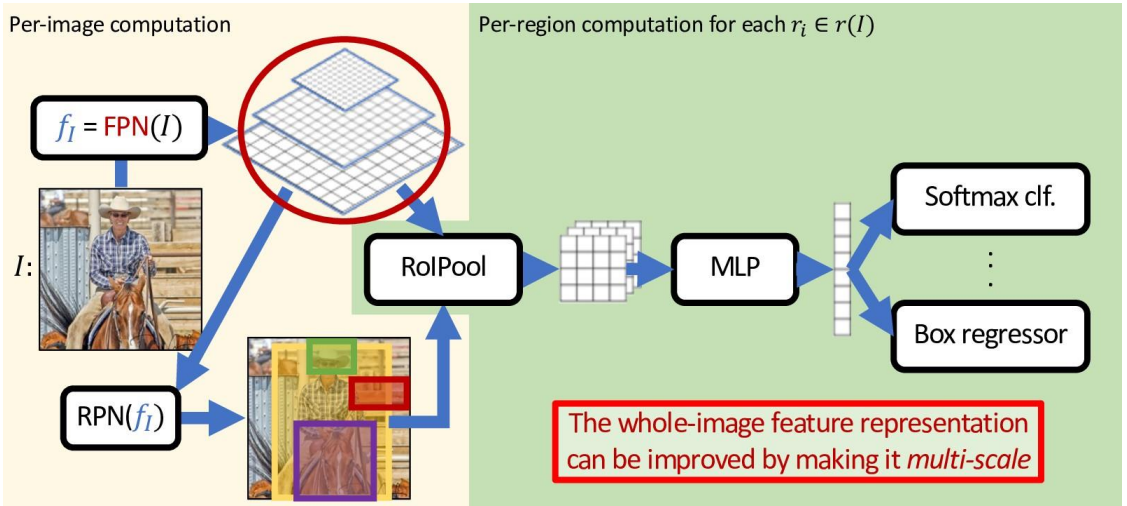

classifier
RoI pooling
proposals
Region Proposal Network
feature maps
conv layers
image

Faster R-CNN

At each location, consider boxes of many different sizes and aspect ratios



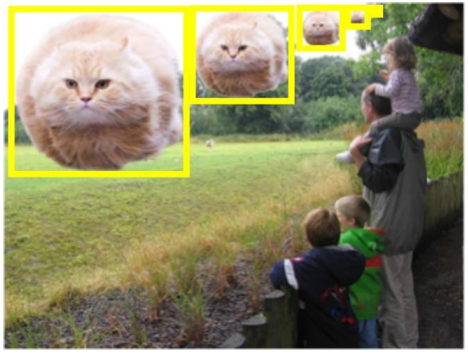
Feature Pyramid Network



Lin, Dolla´r, Girshick, He, Hariharan and Belongie: Feature Pyramid Networks for Object Detection. CVPR, 2017.

Feature Pyramid Network

Goal of Feature Pyramid Network: Improve Scale Equivariance



Detectors need to

1. classify and
2. localize

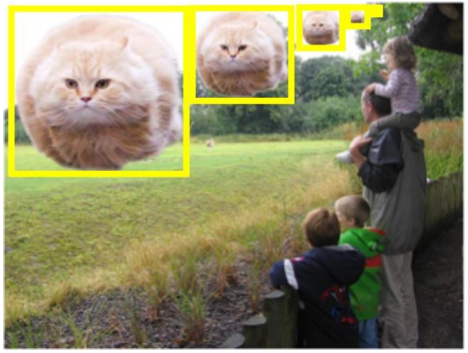
objects over a wide range of scales

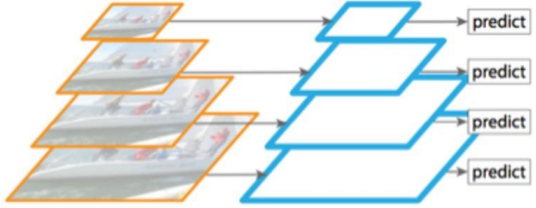
FPN improves this ability

Lin, Dolla´r, Girshick, He, Hariharan and Belongie: Feature Pyramid Networks for Object Detection. CVPR, 2017.

Feature Pyramid Network

Strategy 1:Image Pyramid





(a) Featurized image pyramid

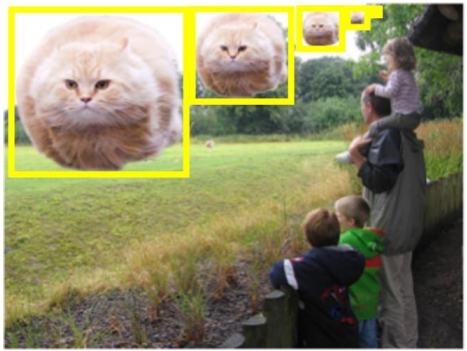
Standard solution – *slow!*

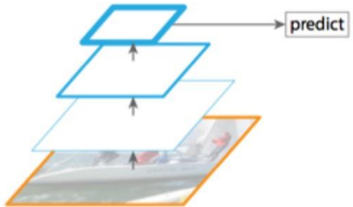
(E.g., Viola & Jones, HOG, DPM, SPP-net,

Lin, Dolla´r, Girshick, He, Hariharan and Belongie: Feature Pyramid Networks for Object Detection. CVPR, 2017.

Feature Pyramid Network

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





(b) Single feature map

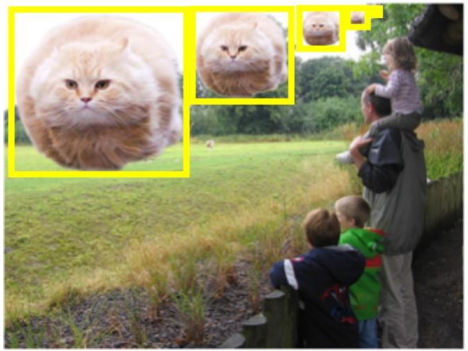
Leave it all to the features – *fast, suboptimal*

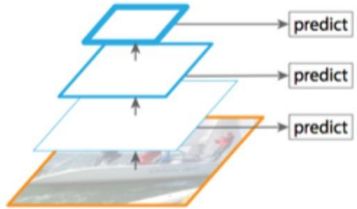
(E.g., Fast/er R-CNN, YOLO, …)

Lin, Dolla´r, Girshick, He, Hariharan and Belongie: Feature Pyramid Networks for Object Detection. CVPR, 2017.

Feature Pyramid Network

Strategy 3: Naive In-network Pyramid





(c) Pyramidal feature hierarchy

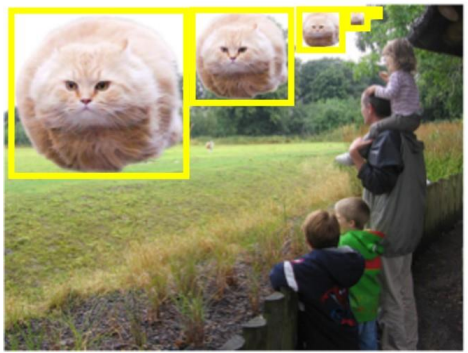
Use the internal pyramid – *fast, suboptimal*

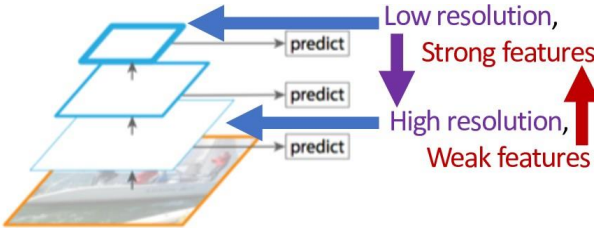
(E.g., ≈ SSD, …)

Lin, Dolla´r, Girshick, He, Hariharan and Belongie: Feature Pyramid Networks for Object Detection. CVPR, 2017.

Feature Pyramid Network

Strategy 3: Naive In-network Pyramid





(c) Pyramidal feature hierarchy

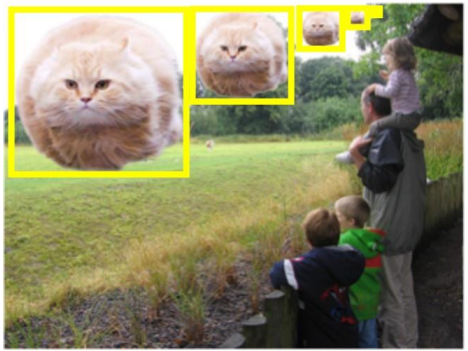
Use the internal pyramid – *fast, suboptimal*

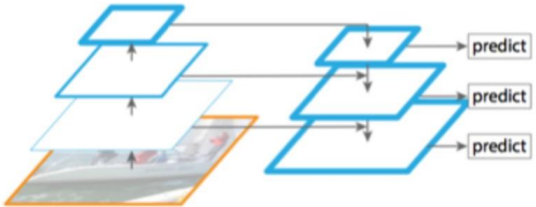
(E.g., ≈ SSD, …)

Lin, Dolla´r, Girshick, He, Hariharan and Belongie: Feature Pyramid Networks for Object Detection. CVPR, 2017.

Feature Pyramid Network

Strategy 4: Feature Pyramid Network





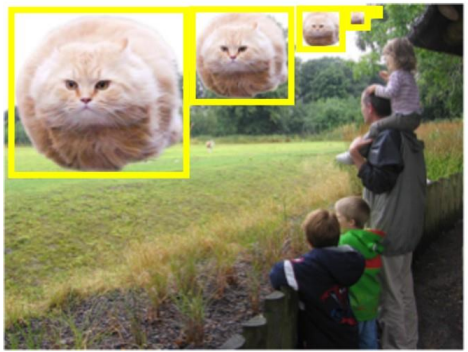
(d) Feature Pyramid Network

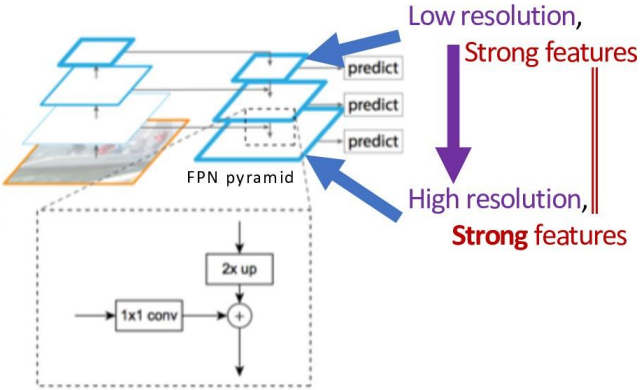
Top-down enrichment of high-res features – *fast, less suboptimal*

Lin, Dolla´r, Girshick, He, Hariharan and Belongie: Feature Pyramid Networks for Object Detection. CVPR, 2017.

Feature Pyramid Network

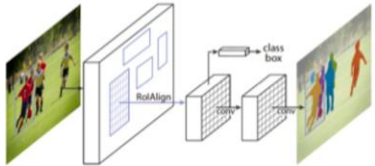
Strategy 4: Feature Pyramid Network



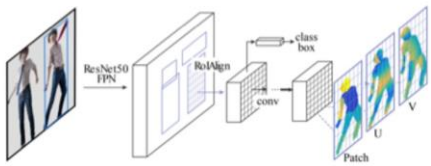


Lin, Dolla´r, Girshick, He, Hariharan and Belongie: Feature Pyramid Networks for Object Detection. CVPR, 2017.

Generalization to other Output Modalities



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



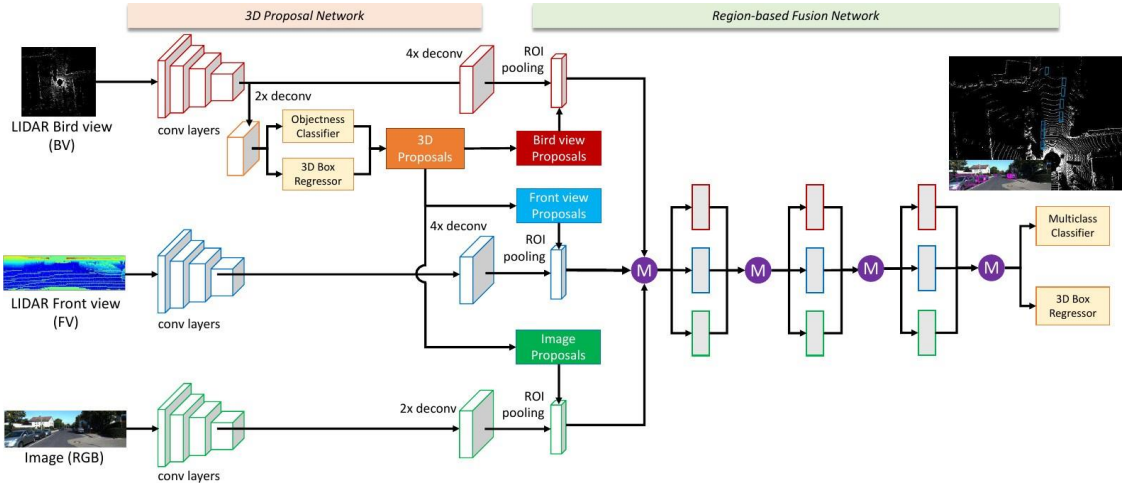
DensePose
[Güler, Neverova, Kokkinos]

- 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 (think: texture map coordinates)

https://github.com/facebookresearch/Detectron

He, Gkioxari, Dolla´r and Girshick: Mask R-CNN. ICCV, 2017.

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



Chen, Ma, Wan, Li and Xia: Multi-view 3D Object Detection Network for Autonomous Driving. CVPR, 2017.

Readings

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

► Object Detection with Discriminatively Trained Part Based Models. Felzenszwalb, Girshick, McAllester and Ramanan. PAMI, 2010.

► Ren, He, Girshick and Sun et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS, 2015.

► Lin, Dollar, Girshick, He, Hariharan and Belongie: Feature Pyramid Networks for Object Detection. CVPR, 2017.