Last time..

(Vanilla) RNN, LSTM, GRU

Today:

Object detection

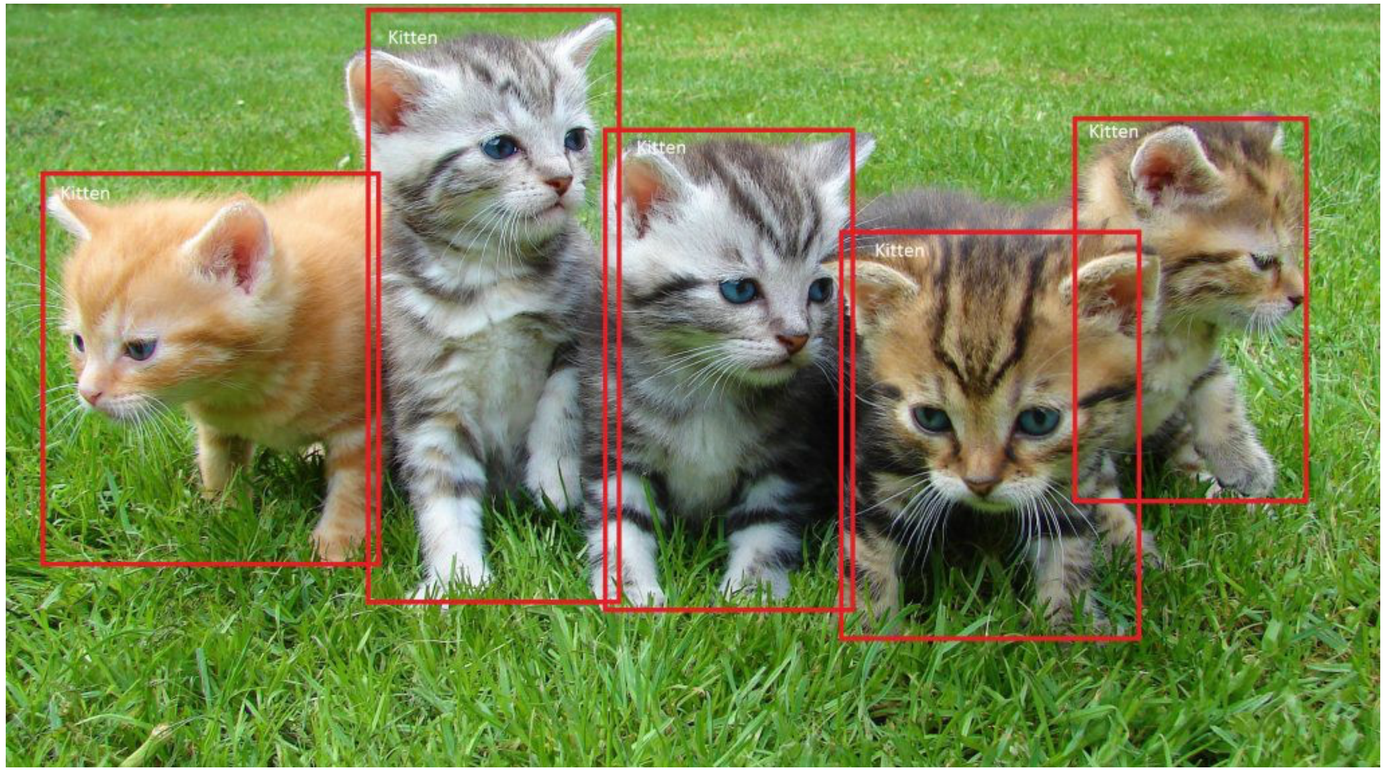
Links / References

- Stanford: CS231n: Detection and Segmentation (video)
  - Stanford: CS231n (course)
- MIT 6.S191: Deep Computer Vision (video)
  - MIT 6.S191 (course)
- CNN W3L01 : Object Localization++ (Deep Learning Course, Andrew Ng)
  - CNNs - CV (Deep Learning Course, Andrew Ng)

Agenda

- What do we mean by object detection?
- Datasets: **ImageNet, PASCAL VOC, COCO**
- Metrics: **IoU & mAP**
- Methods / Models
  - Early attempts: from classification to detection: **Regression (fixed), Sliding window**
  - Region-based methods: two stage methods (find ROIs, classify each): **R-CNN, Fast R-CNN, Faster R-CNN, R-FCN, FPN**
  - Single-shot methods: **YOLO v1/v2/v3, SSD, RetinaNet**

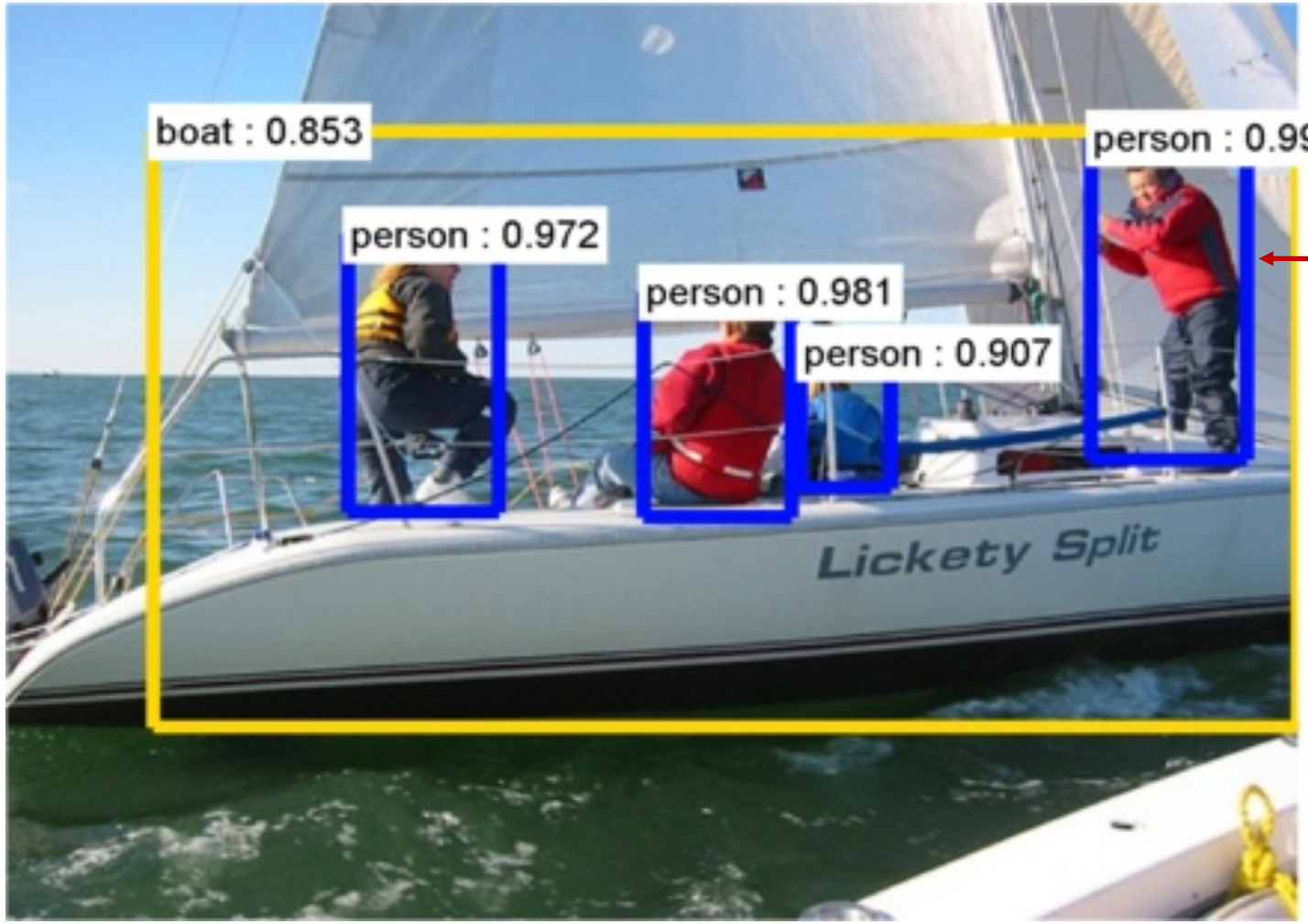
What is **Object detection?**



Real-time object detection with YOLO v2

Real-time object detection with YOLO v3

Bounding-Box Object Detection

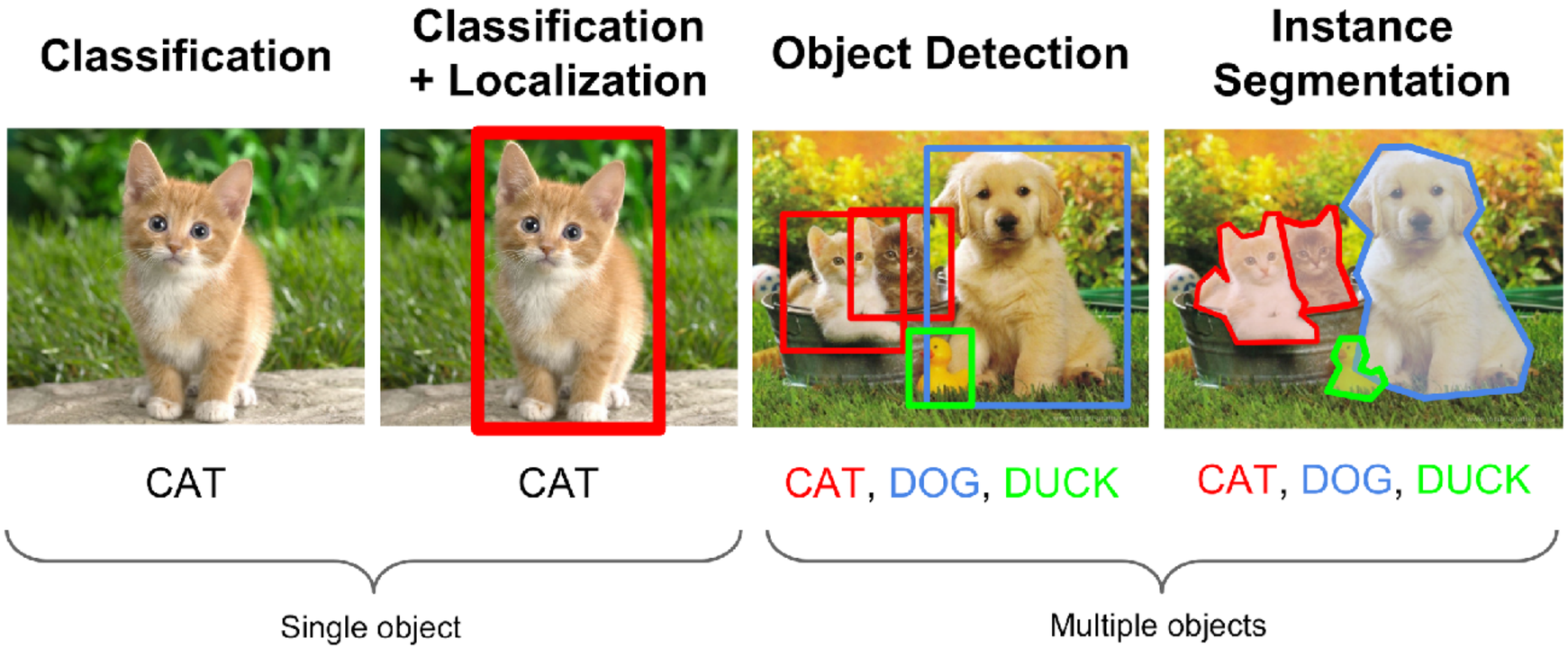


What?

Where?

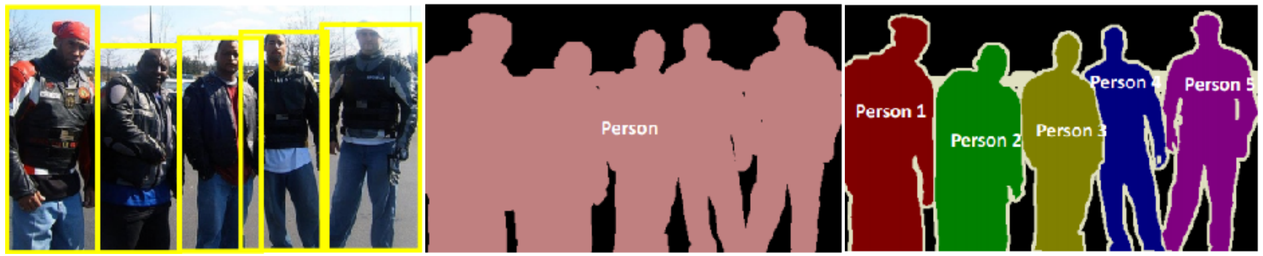
Object detection

Object detection



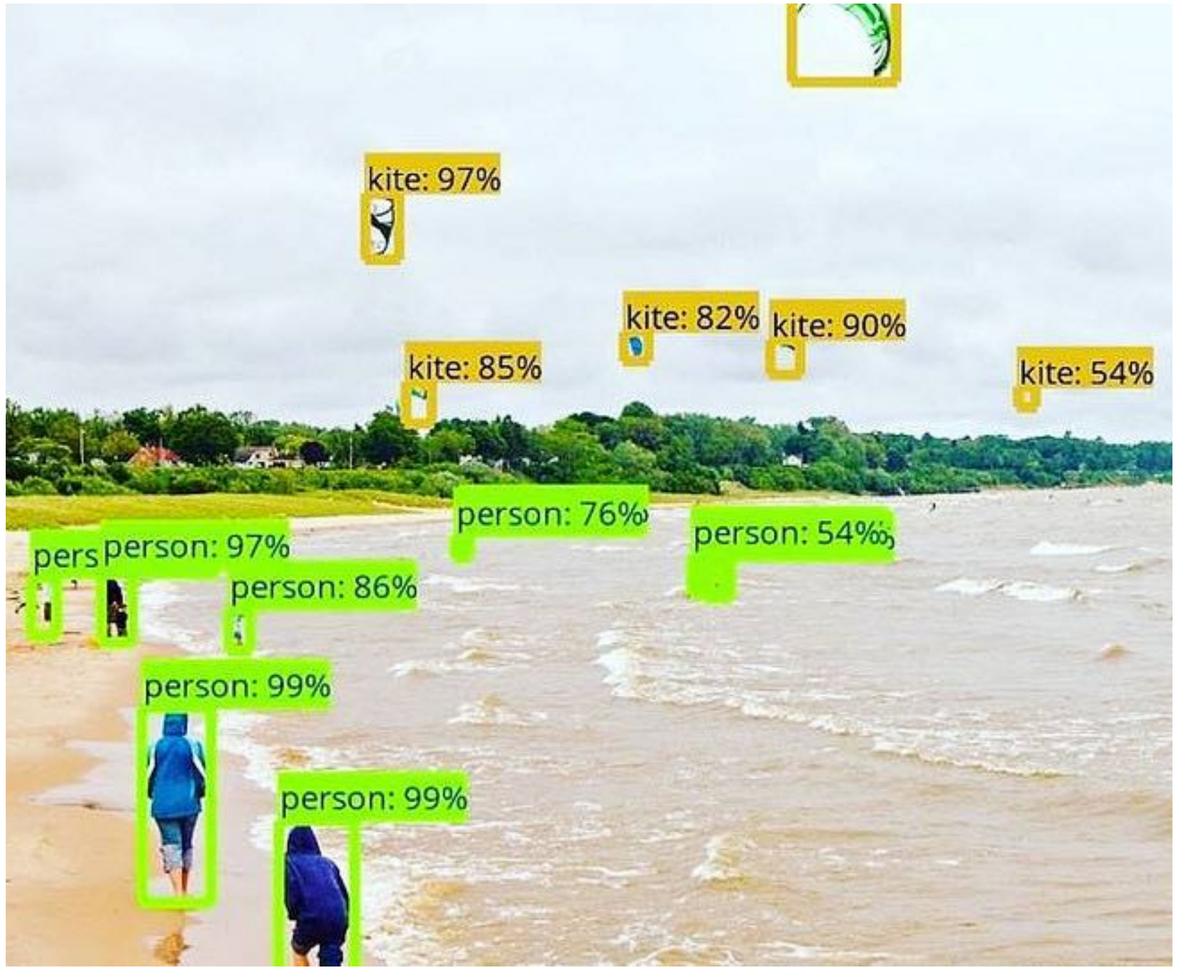
Fixed number

Semantic vs. Instance segmentation



Object detection

Find which objects are in the image, and where they are



CV: Tasks

- **Classification (recognition):** Given an image, output what object is in it (one object per image)
- **Localization:** Given an image, output the object and the location of the object: a box that covers the object
- **<u>Detection</u>: :** Given an image, find and localize all objects (more than one object per image)

- **Segmentation++**

Datasets / Challenges

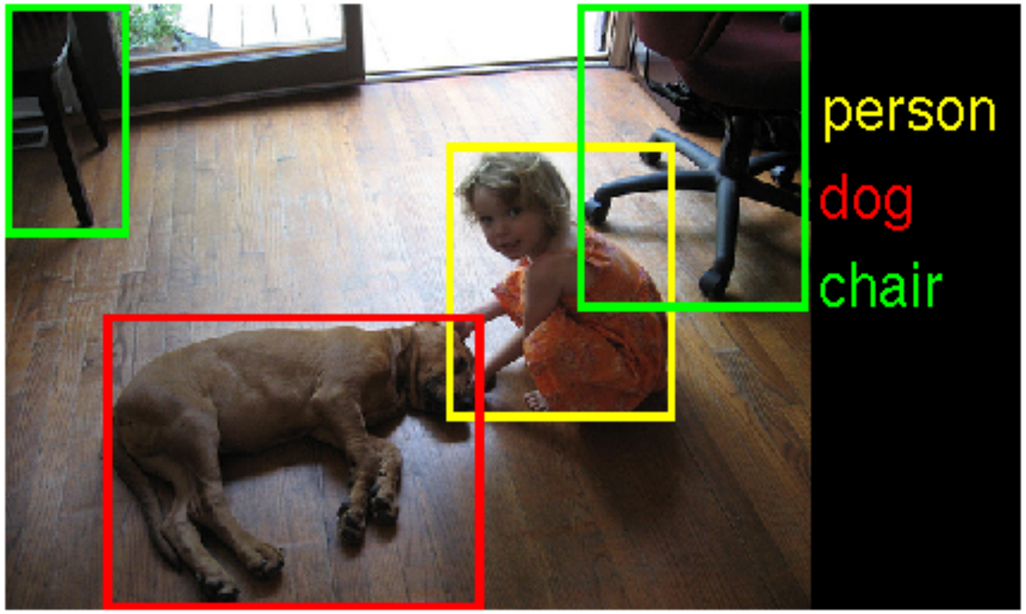
- ImageNet
- PASCAL VOC
- COCO object detection

ImageNet challenge

- Data for evaluating classification, **localization, and detection** tasks

- Similar to classification data, there are **1,000** classes for **localization** tasks. The accuracy is calculated based on the **top five detections**. There will be at least **one** bounding box in all the images.

- There are **200** objects for **detection** problems with **470,000** images, with an average of **1.1** objects per image.



Object / Bounding Boxes

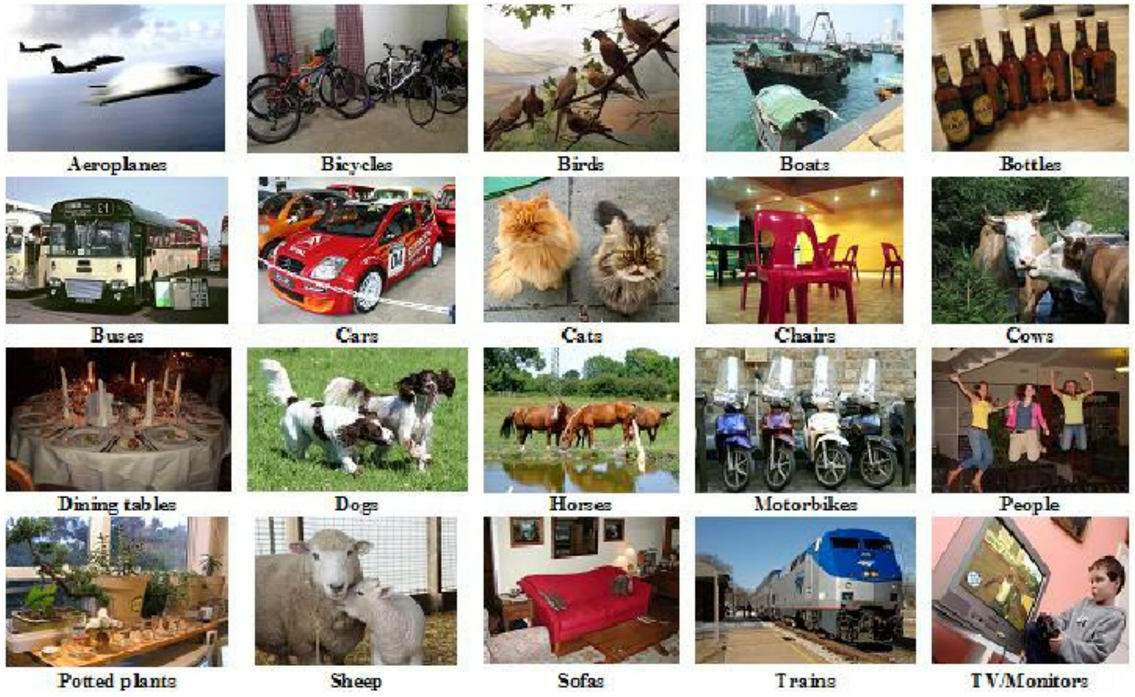
PASCAL VOC challenge

- Pascal Visual Object Classes (VOC), ran from 2005 to 2012

- Considered the **benchmark** for object detection techniques.

- There are **20** classes in the dataset. The dataset has **11,530** images for training and validations with **27,450** annotations for regions of interest. There is an average of **2.4** objects per image.

- The following are the twenty classes present in the dataset: **Person** (Person), **Animal** (Bird, cat, cow, dog, horse, sheep), **Vehicle** (Airplane, bicycle, boat, bus, car, motorbike, train), **Indoor** (Bottle, chair, dining table, potted plant, sofa, tv/monitor)



PASCAL VOC

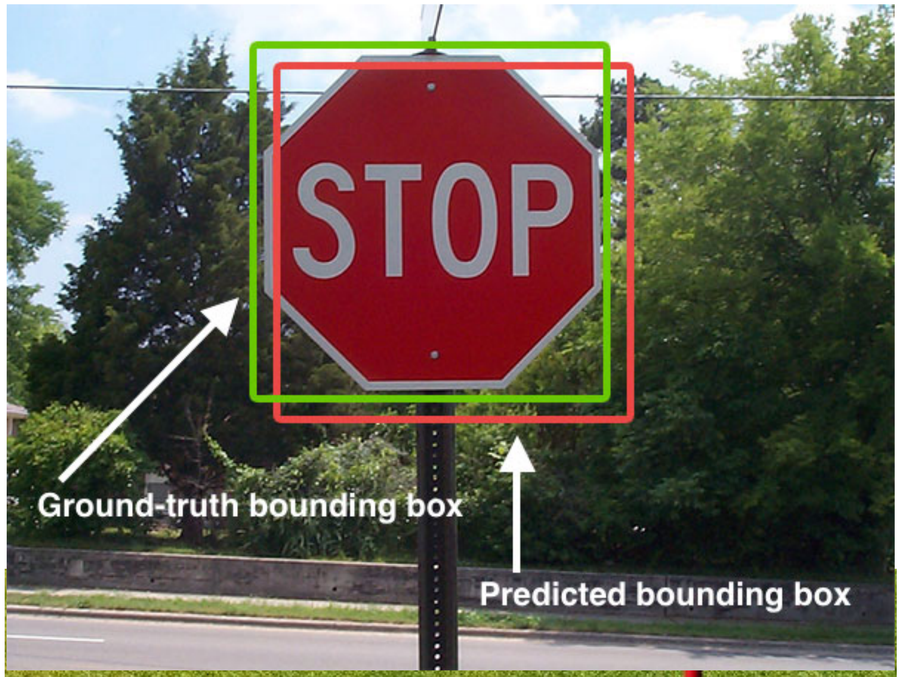
COCO object detection challenge

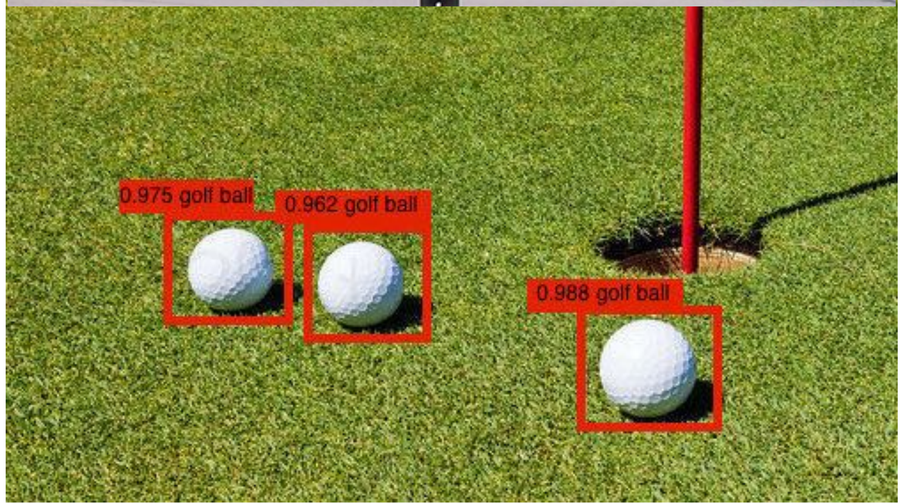
- Common Objects in Context (COCO) dataset
- **200,000** images with more than **500,000** object annotations in **80** categories
- most extensive publicly available object detection database.
- The average number of objects is **7.2** per image.

COCO

Metrics

- **How** to measure performance?
- Measuring performance is more **tricky** for object detection than classification
  - **Multiple** targets - what if some are correct and some are not?
  - **Variable** number of targets
  - How good is a bounding box (**BB**), is there a limit for when we judge it as correct?
  - What about class label and **confidence**?



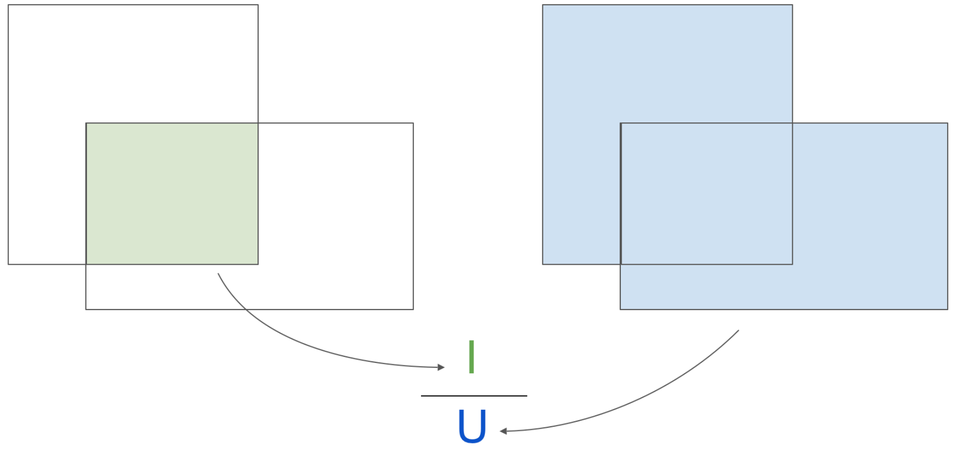


IoU - mAP

IoU, Confidence -> TP, FP, FN, TN -> Precision, recall -> AP -> mAP

For detection, a common way to determine if one object proposal was right is *Intersection over Union* (IoU, IU). This takes the set A of proposed object pixels and the set of true object pixels B and calculates:

$$IoU(A, B) = \frac{A \cap B}{A \cup B}$$

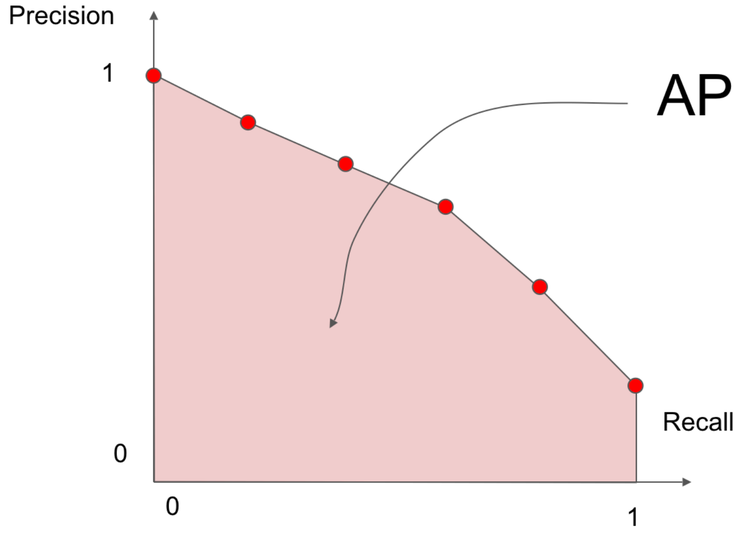


Commonly, IoU > 0.5 means that it was a hit, otherwise it was a fail. For each class, one can calculate the

- True Positive ($TP(c)$): a proposal was made for class c and there actually was an object of class c
- False Positive ($FP(c)$): a proposal was made for class c, but there is no object of class c
- Average Precision for class c: $\frac{\#TP(c)}{\#TP(c)+\#FP(c)}$

The mAP (mean average precision) = $\frac{1}{|classes|} \sum_{c \in classes} \frac{\#TP(c)}{\#TP(c)+\#FP(c)}$

If one wants better proposals, one does increase the IoU from 0.5 to a higher value (up to 1.0 which would be perfect). One can denote this with mAP@p, where $p \in (0, 1)$ is the IoU.

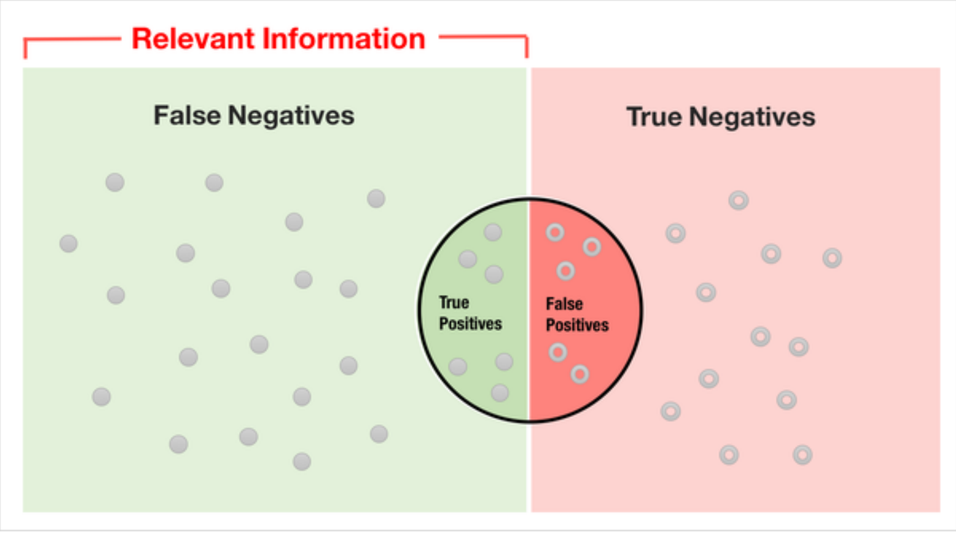


TP, FP, FN, TN

- **Medicine:**
  - TP (True Positive): When the **test** says you have it and you actually have it.
  - FP (False Positive): When the test says you have it but you actually don't have it.
  - FN (False Negative): When the test says you don't have it but you actually have it.
  - TN (True Negative): When the test says you don't have it and you actually don't have it.

		Disease	
		+	-
Test	+	True Positive (TP)	False Positive (FP)
	-	False Negative (FN)	True Negative (TN)
		All with disease= TP + FN	All without disease= FP + TN

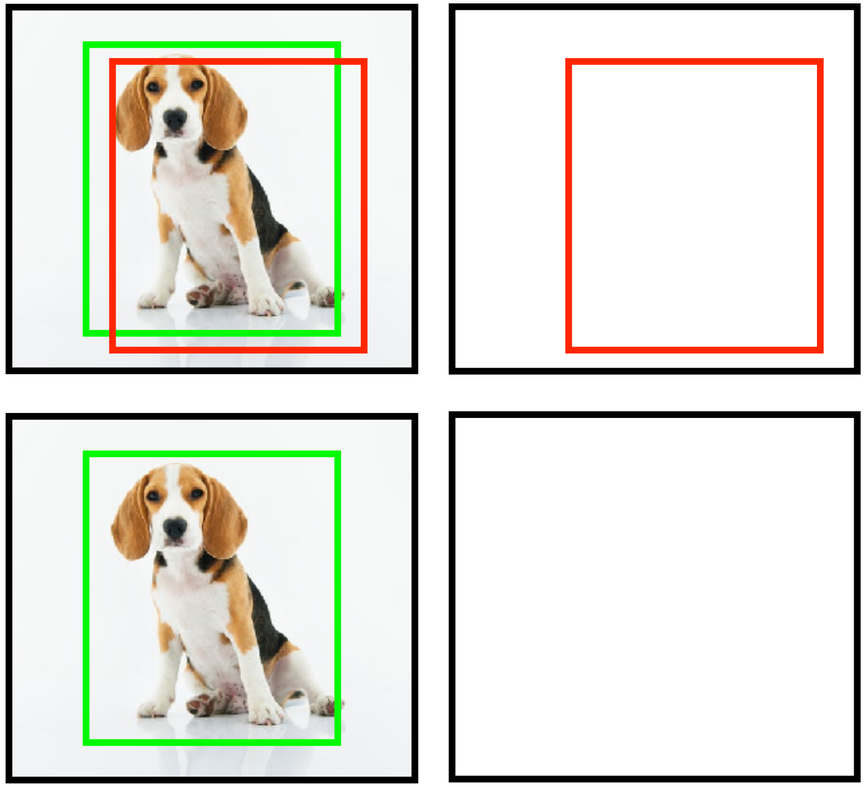
Relevant vs. Selected items (positives)



TP, FP, FN, TN (2)

		Actual	
		Positive	Negative
Predicted	Positive	**True Positive**	**False Positive**
	Negative	**False Negative**	**True Negative**

- **Object detection (P_BB, GT_BB):**
  - TP: P_BB & GT_BB) (red & green BB (object))
  - FP: P_BB & NO GT_BB (red (background))
  - FN: NO P_BB & GT_BB (green (object))
  - TN: NO P_BB & NO GT_BB (background)



Relevant vs. Selected items (positives)

Precision & recall

TP = True positive
TN = True negative
FP = False positive
FN = False negative

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

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

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

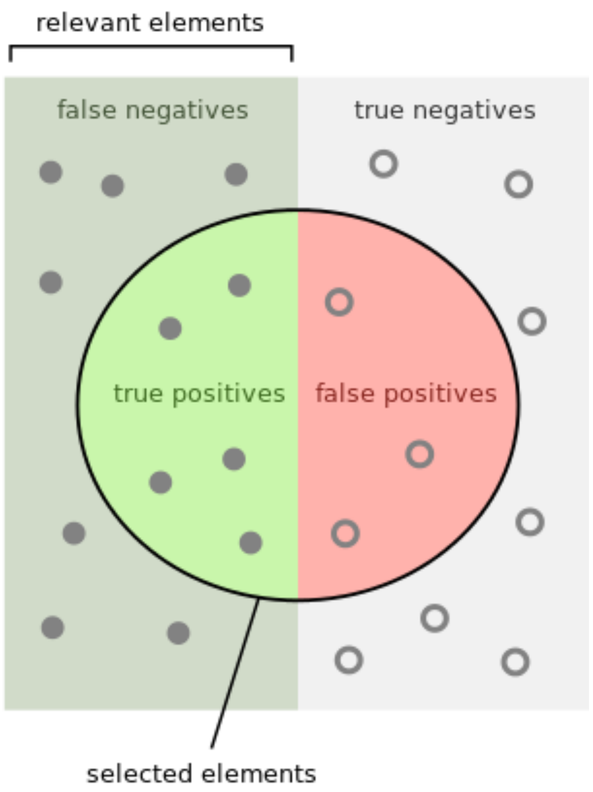
$$Precision = \frac{TP}{\text{total positive results}}$$

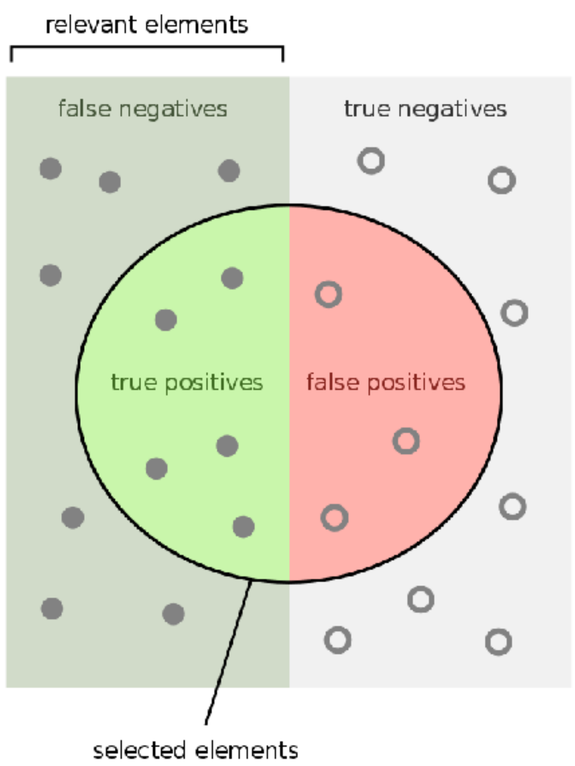
$$Recall = \frac{TP}{\text{total cancer cases}}$$

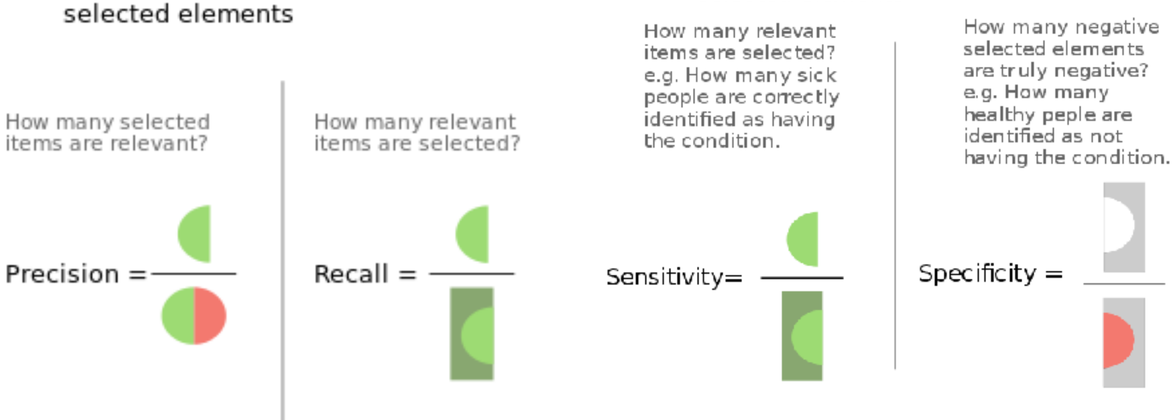
Relevant vs. Selected items (positives)

All with disease vs. all tests (i.e. health)

All GT-BB vs. P-BB





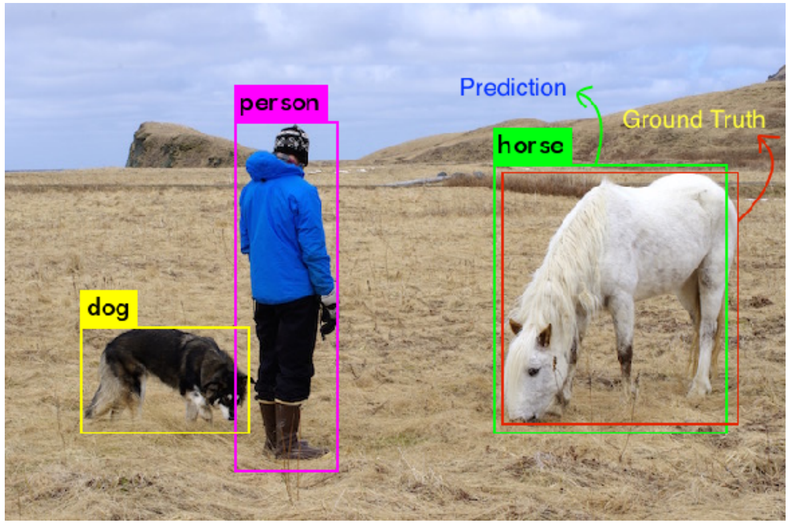


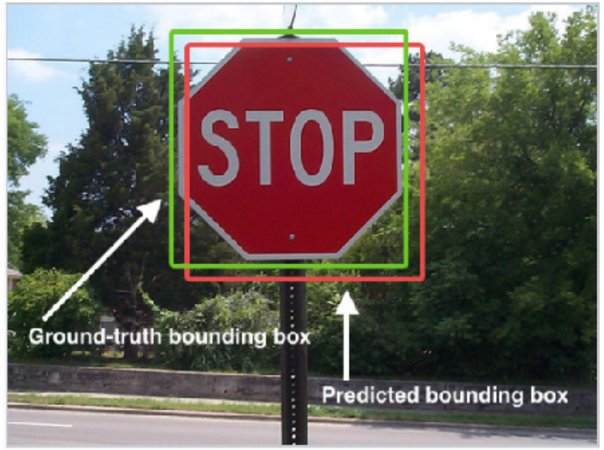
IoU (Intersection over Union)

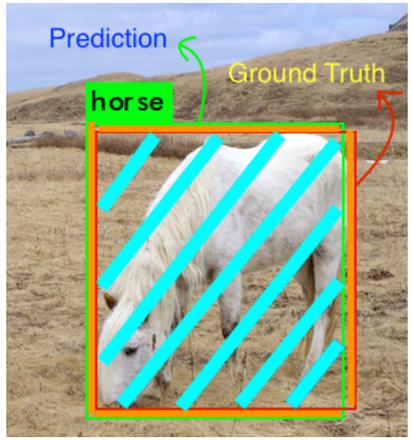
- **IoU** is a **ratio** between the **intersection** and the **union** of the **predicted** boxes and the **ground truth** boxes (Jaccard Index).

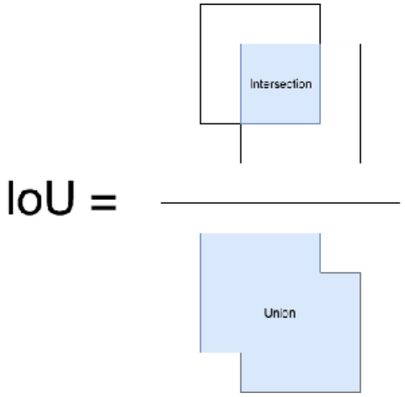
- The area **overlapping** the prediction box and ground truth box is the intersection area and the **total area** spanned is the union.

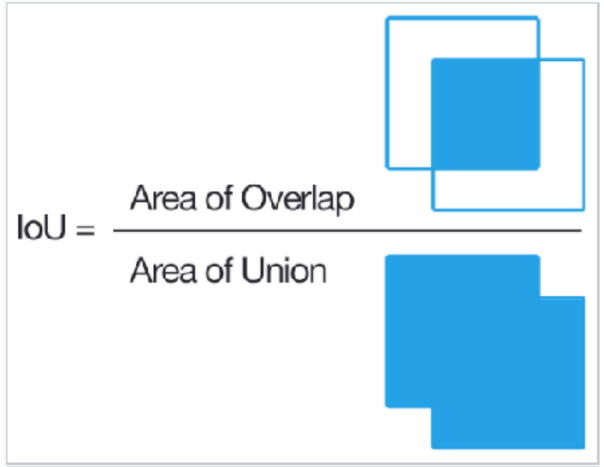
- (Confidence for TP)











AP / mAP

- AP = Average Precision / mAP = mean AP

- **IoU** and the **Confidence** thresholds determines True Positives, False Positives and “False” Negatives (Missed Detections) which determines Precision and Recall.

  - **IoU** > 0.5 is considered a True Positive, else it is considered a False Positive. “False” Negatives are the objects that our model misses.

  - By varying our **confidence** threshold we can change whether a predicted box is a **Positive** or **Negative**. The confidence factor varies across models, which would vary the **precision recall curve** shape as well.

- The **AP** summarizes the **shape** of the precision/recall curve, and is de-fined as the **average precision** at a set of **eleven** equally spaced **recall** levels [0,0.1,…,1]. Chose **11** different **confidence** thresholds (which determine the “rank”). The thresholds should be such that the **Recall** at those confidence values is 0, 0.1, 0.2, 0.3, … , 0.9 and 1.0. The AP is now defined as the average of the Precision values at these chosen 11 Recall values. (old method).

- mAP (mean average precision) is the **mean** of all the average precisions (APs) of our **classes** in the dataset.

AP / mAP (2)

- mAP:
  - is always calculated over a **fixed** dataset.
  - not easy to **interpret**, pretty good **relative** metric, popular public datasets, compare old and new approaches.
  - AP values might vary from very high for some classes(which had good training data) to very low(for classes with less/bad data). Model might be really good for certain classes and really bad for certain classes. These values might also serve as an indicator to add more training samples.

Metrics for object detection

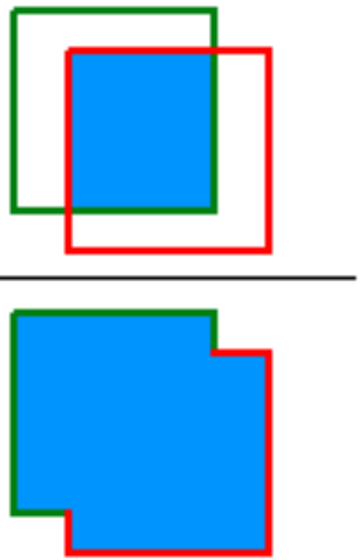
theory

Intersection Over Union (IOU)

- **ground truth** bounding box
- **predicted** bounding box
- By applying the IOU we can tell if a detection is valid (True Positive) or not (False Positive).
- The figure illustrates the IOU between a ground truth bounding box (in **green**) and a detected bounding box (in **red**).

$$\mathrm{IOU} = \frac{\mathrm{area}\left(B_p \cap B_{gt}\right)}{\mathrm{area}\left(B_p \cup B_{gt}\right)}$$

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



True Positive, False Positive, False Negative and True Negative

- True Positive (**TP**): A **correct** detection. Detection with IOU ≥ threshold
- False Positive (**FP**): A **wrong** detection. Detection with IOU < threshold
- False Negative (**FN**): A ground truth **not detected**
- True Negative (**TN**): Does not apply. All possible bounding boxes that were correctly not detected (that would apply to many possible boxes within an image).
- T**hreshold**: depends, usually set to 50%, 75% or 95%.

Precision & Recall

- **Precision** is the ability of a model to identify only the relevant objects. It is the percentage of correct positive predictions. Of all the detections predicted, how many where true.

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}}$$

- **Recall** is the ability of a model to find all the **relevant** cases (all **ground truth** bounding boxes). It is the percentage of true positive detected among all relevant ground truths. Of all the objects present, how many were our model able to detect.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}}$$

Metric: Precision x Recall curve

- An object detector of a particular class is considered good if its **precision** stays high as recall **increases**, which means that if you vary the **confidence** threshold, the precision and recall will still be high.

- Another way to identify a good object detector is to look for a detector that can **identify only relevant objects** (0 False Positives = high precision) and **find all ground truth objects** (0 False Negatives = high recall).

- A poor object detector needs to increase the number of detected objects (increasing False Positives = lower precision) in order to retrieve all ground truth objects (high recall). That's why the Precision x Recall curve usually starts with high precision values, decreasing as recall increases.

Metric: Average Precision

- Another way to **compare** the **performance** of object detectors is to calculate the **area** under the curve of the Precision x Recall **curve**. As AP curves are often zigzag curves going up and down, comparing different curves (different detectors) in the same plot usually is not an easy task - because the curves tend to cross each other much frequently. That's why **Average Precision (AP)**, a numerical metric, can also help us compare different detectors. In practice AP is the precision averaged across all recall values between 0 and 1.

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} \rho_{\text{interp}}(r)$$

- **<u>11-point interpolation (before 2010):</u>**

  - tries to summarize the shape of the Precision x Recall curve by averaging the precision at a set of eleven equally spaced recall levels [0, 0.1, 0.2, ... , 1]

$$\rho_{\text{interp}}(r) = \max_{\tilde{r}:\tilde{r} \geq r} \rho(\tilde{r})$$

  - Instead of using the precision observed at each point, the AP is obtained by interpolating the precision only at the 11 levels r taking the maximum precision whose recall value is greater than r

Metric: Average Precision (2)

$$\sum_{r=0}^{1} (r_{n+1} - r_n)\, \rho_{interp}(r_{n+1})$$

- **<u>Interpolating all points:</u>**
  - Instead of interpolating only in the 11 equally spaced points, you could interpolate through all points

$$\rho_{interp}(r_{n+1}) = \max_{\tilde{r}:\tilde{r} \geq r_{n+1}} \rho(\tilde{r})$$

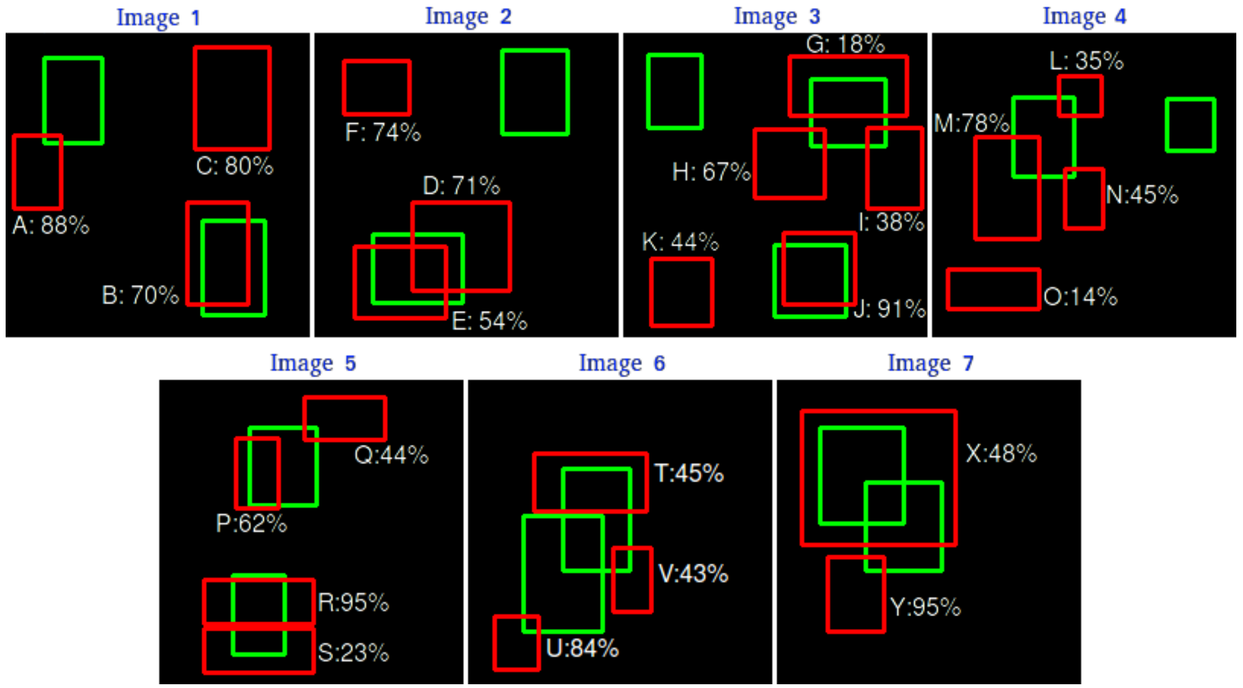
  - In this case, instead of using the precision observed at only few points, the AP is now obtained by interpolating the precision at each level r taking the maximum precision whose recall value is greater or equal than r+1.
  - This way we calculate the estimated area under the curve.

Metrics for object detection

Example

Images, GT & predictions

- An illustrated example to help us understand the concept of interpolated average precision
  - **7** images with **15** ground truth objects represented by the **green** bounding boxes and **24** detected objects represented by the **red** bounding boxes.
  - Each detected object has a **confidence** level and is **identified** by a letter (A,B,...,Y).



Tables and curve

- The following table shows the **bounding boxes** with their corresponding **confidences**. The last column identifies the **detections as TP or FP**. In this example a TP is considered if IOU 30%, otherwise it is a FP. By looking at the images above we can roughly tell if the detections are TP or FP.

- In some images there are more than one detection overlapping a ground truth (Images 2, 3, 4, 5, 6 and 7). For those cases the **detection with the highest IOU is taken**, discarding the other detections. This rule is applied by the PASCAL VOC 2012 metric: "e.g. 5 detections (TP) of a single object is counted as 1 correct detection and 4 false detections”.

- The Precision x Recall curve is plotted by calculating the **precision and recall** values of the **accumulated TP or FP** detections. For this, first we need to **order** the detections by their **confidences**, then we calculate the precision and recall for each accumulated detection as shown in the table

- As mentioned before, there are **two** different ways to measure the interpolated average precision: **11**-point interpolation and interpolating **all** points.

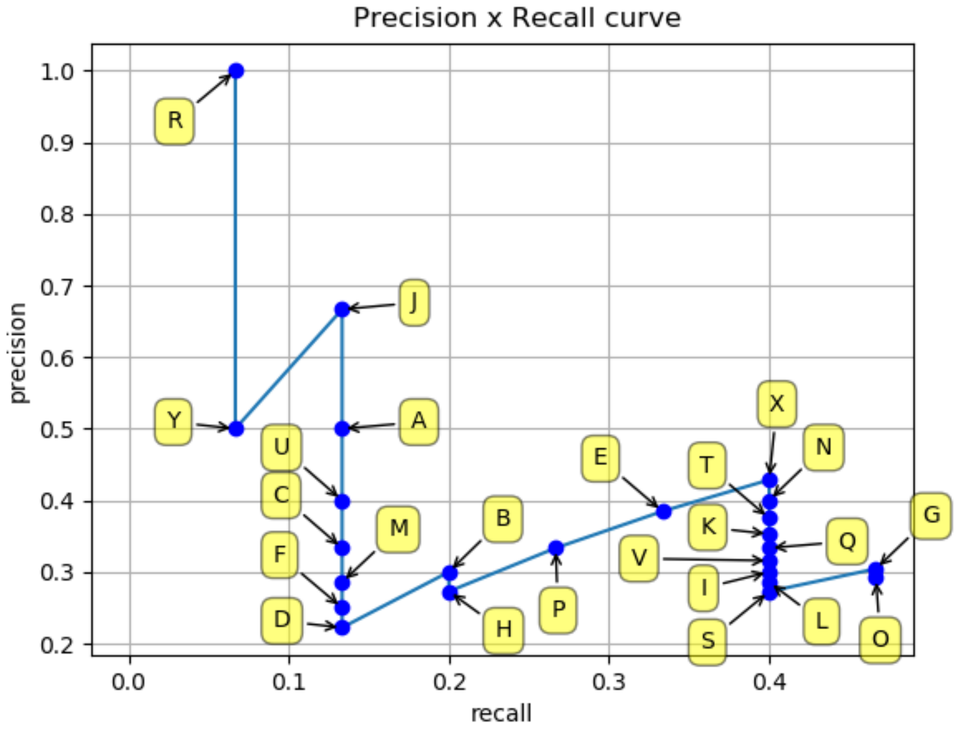
Tables and curve (2)

Images	Detections	Confidences	TP or FP
Image 1	A	88%	FP
Image 1	B	70%	TP
Image 1	C	80%	FP
Image 2	D	71%	FP
Image 2	E	54%	TP
Image 2	F	74%	FP
Image 3	G	18%	TP
Image 3	H	67%	FP
Image 3	I	38%	FP
Image 3	J	91%	TP
Image 3	K	44%	FP
Image 4	L	35%	FP
Image 4	M	78%	FP
Image 4	N	45%	FP
Image 4	O	14%	FP
Image 5	P	62%	TP
Image 5	Q	44%	FP
Image 5	R	95%	TP
Image 5	S	23%	FP
Image 6	T	45%	FP
Image 6	U	84%	FP
Image 6	V	43%	FP
Image 7	X	48%	TP
Image 7	Y	95%	FP

Images	Detections	Confidences	TP	FP	Acc TP	Acc FP	Precision	Recall
Image 5	R	95%	1	0	1	0	1	0.0666
Image 7	Y	95%	0	1	1	1	0.5	0.0666
Image 3	J	91%	1	0	2	1	0.6666	0.1333
Image 1	A	88%	0	1	2	2	0.5	0.1333
Image 6	U	84%	0	1	2	3	0.4	0.1333
Image 1	C	80%	0	1	2	4	0.3333	0.1333
Image 4	M	78%	0	1	2	5	0.2857	0.1333
Image 2	F	74%	0	1	2	6	0.25	0.1333
Image 2	D	71%	0	1	2	7	0.2222	0.1333
Image 1	B	70%	1	0	3	7	0.3	0.2
Image 3	H	67%	0	1	3	8	0.2727	0.2
Image 5	P	62%	1	0	4	8	0.3333	0.2666
Image 2	E	54%	1	0	5	8	0.3846	0.3333
Image 7	X	48%	1	0	6	8	0.4285	0.4
Image 4	N	45%	0	1	6	9	0.4	0.4
Image 6	T	45%	0	1	6	10	0.375	0.4
Image 3	K	44%	0	1	6	11	0.3529	0.4
Image 5	Q	44%	0	1	6	12	0.3333	0.4
Image 6	V	43%	0	1	6	13	0.3157	0.4
Image 3	I	38%	0	1	6	14	0.3	0.4
Image 4	L	35%	0	1	6	15	0.2857	0.4
Image 5	S	23%	0	1	6	16	0.2727	0.4
Image 3	G	18%	1	0	7	16	0.3043	0.4666
Image 4	O	14%	0	1	7	17	0.2916	0.4666

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}}$$

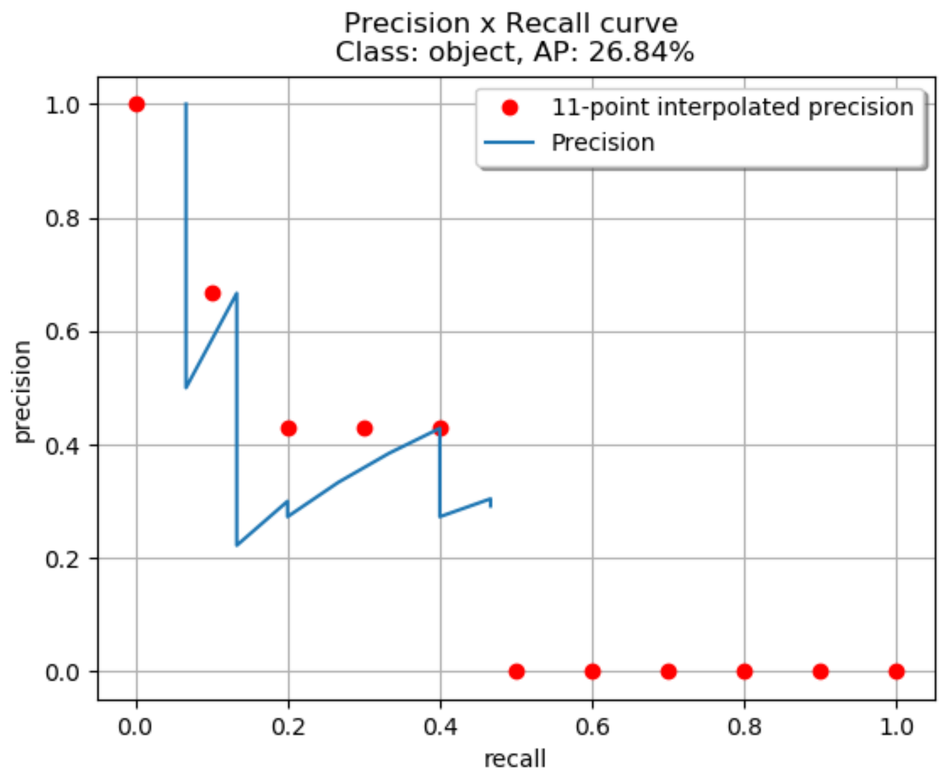
$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}}$$



Calculating the 11-point interpolation

- The idea of the 11-point interpolated average precision is to average the precisions at a set of 11 recall levels (0,0.1, …,1).

- The interpolated precision values are obtained by taking the maximum precision whose recall value is greater than its current recall value



$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} \rho_{\text{interp}}(r)$$

$$AP = \frac{1}{11} (1 + 0.6666 + 0.4285 + 0.4285 + 0.4285 + 0 + 0 + 0 + 0 + 0 + 0)$$

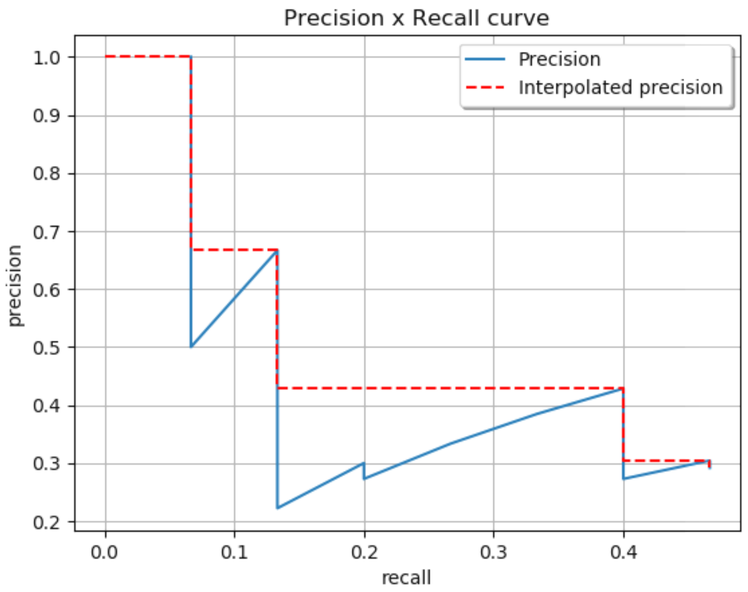
$$AP = 26.84\%$$

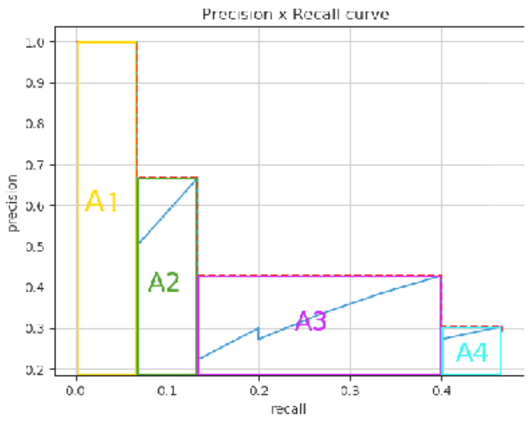
Calculating the all points interpolation

- By interpolating all points, the Average Precision (AP) can be interpreted as an approximated area under the Precision x Recall curve. The intention is to reduce the impact of the wiggles in the curve.

- Looking at the plot we can divide it into 4 areas (A1, A2, A3 and A4):

- The results between the two different interpolation methods are a little different: 24.56% and 26.84% by the every point interpolation and the 11-point interpolation respectively.





$$AP = A1 + A2 + A3 + A4$$

with:

$$A1 = (0.0666 - 0) \times 1 = \mathbf{0.0666}$$
$$A2 = (0.1333 - 0.0666) \times 0.6666 = \mathbf{0.04446222}$$
$$A3 = (0.4 - 0.1333) \times 0.4285 = \mathbf{0.11428095}$$
$$A4 = (0.4666 - 0.4) \times 0.3043 = \mathbf{0.02026638}$$

$$AP = 0.0666 + 0.04446222 + 0.11428095 + 0.02026638$$
$$AP = 0.24560955$$
$$AP = \mathbf{24.56}\%$$

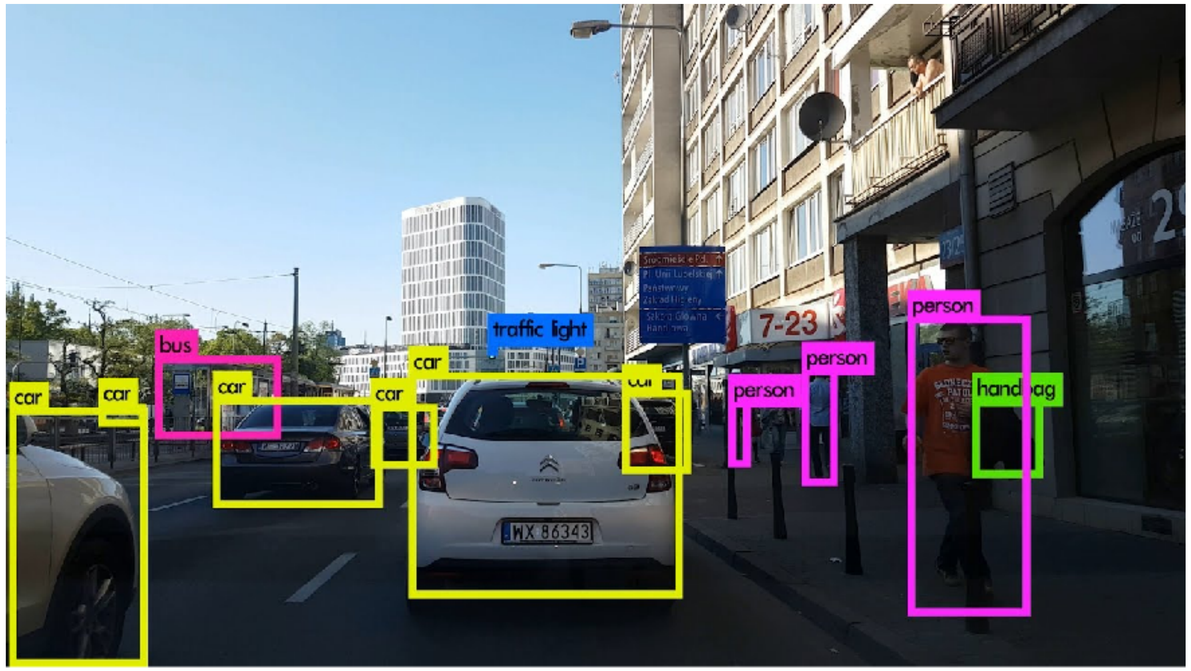
Models / Methods

- Early attempts
- Region-based models (two stages)
- Single-shot methods (one stage)

From Classification to Detection

Early attempts

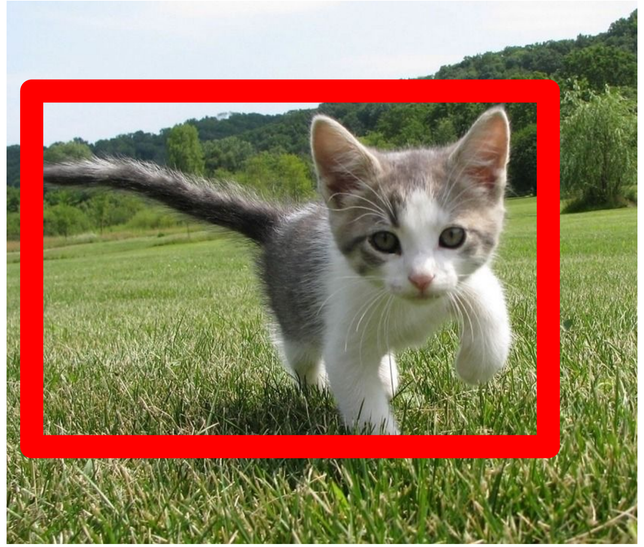
- CNNs work well for classifying an image
- How can we detect objects in an image?
  - (Localization)
  - Regression (fixed)
  - Sliding window
    - Run a classifier by sliding it over the image
    - Detect objects based on output from classifier
  - Region based methods:
    - First part of network finds regions of interest
    - Classifier is run on each region of interest
  - Single-shot detection
    - Split image into a set of regions
    - Constant number of detections per region
    - Thresholding



From Classification to Detection

Early attempts

Classification + Localization



CAT

So far: Image Classification

This image is CC0 public domain

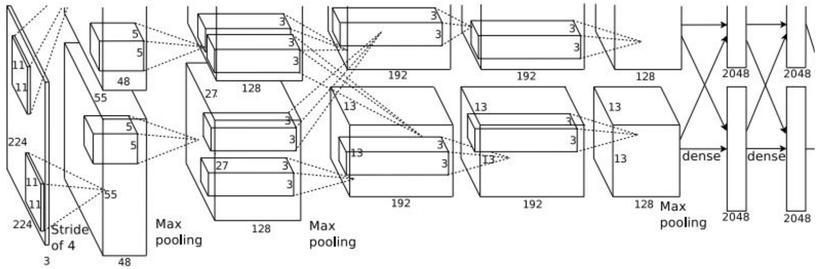


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

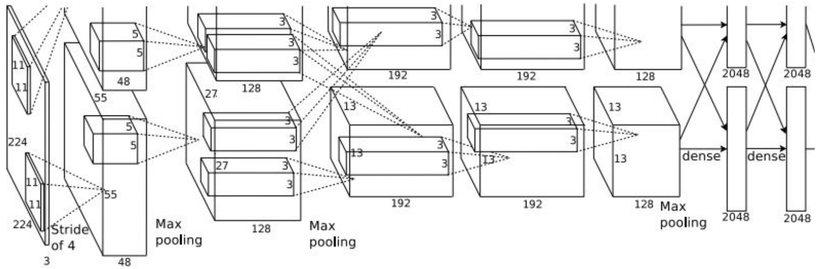
Vector:
4096

Fully-Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Classification + Localization

This image is CC0 public domain



Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

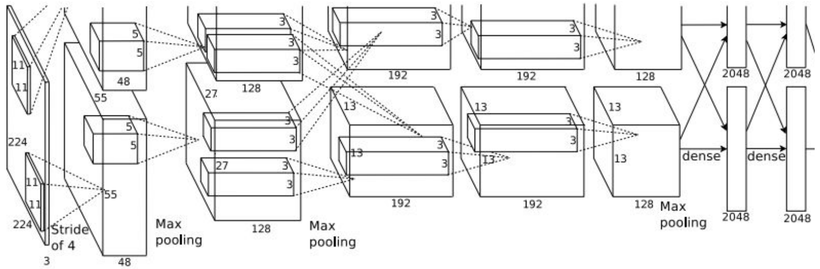
Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

Treat localization as a regression problem!

Classification + Localization

This image is CC0 public domain



Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

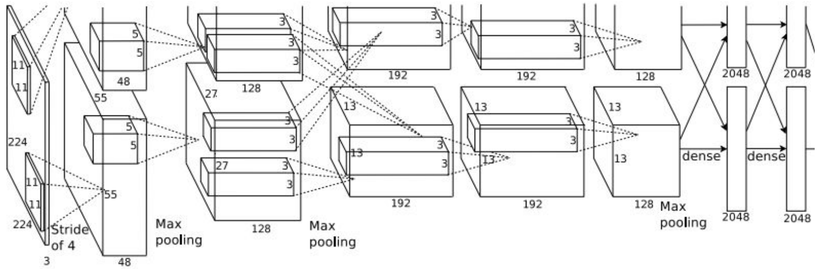
L2 Loss

Correct box: (x', y', w', h')

Treat localization as a regression problem!

Classification + Localization

This image is CC0 public domain



Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

Correct box: (x', y', w', h')

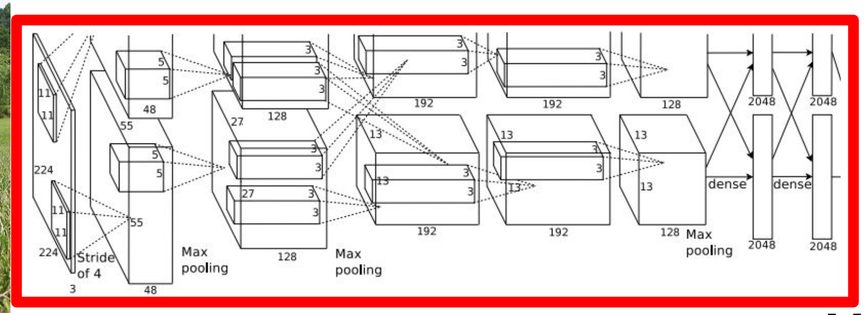
Multitask Loss

+ → **Loss**

Treat localization as a regression problem!

Classification + Localization

This image is CC0 public domain



Often pretrained on ImageNet
(Transfer learning)

Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Correct label: Cat

Softmax Loss

+ → **Loss**

Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

L2 Loss

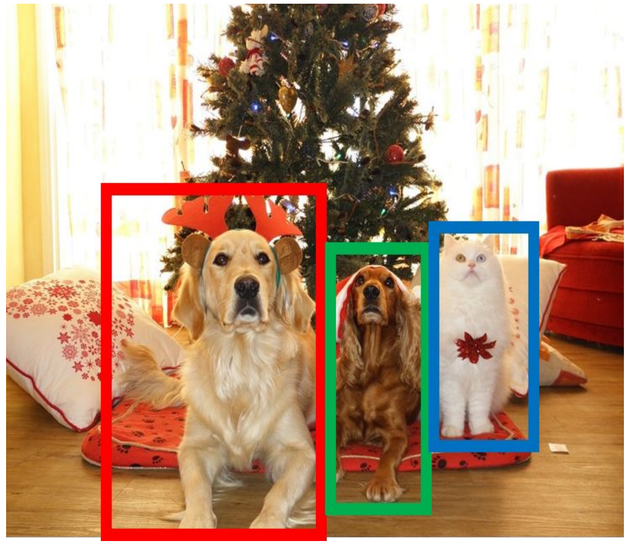
Correct box:
(x', y', w', h')

Treat localization as a regression problem!

From Classification to Detection

Early attempts

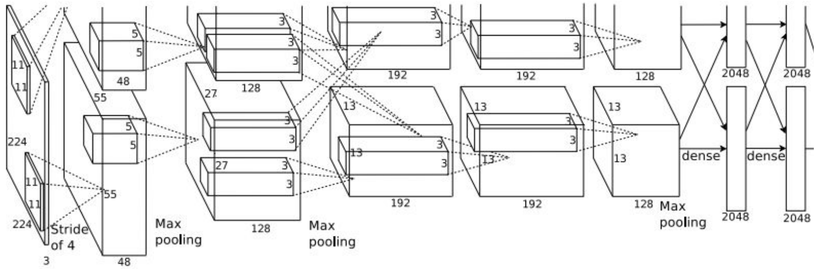
Object Detection as Regression



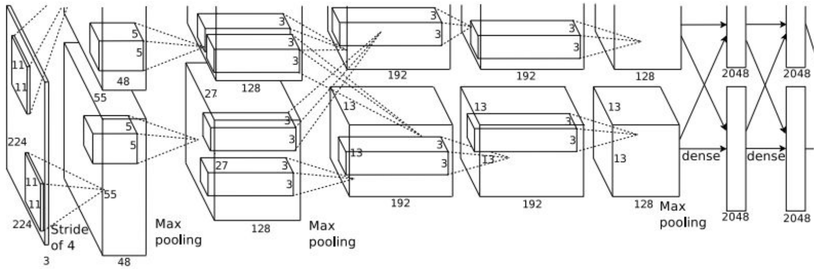
Fixed #

DOG, DOG, CAT

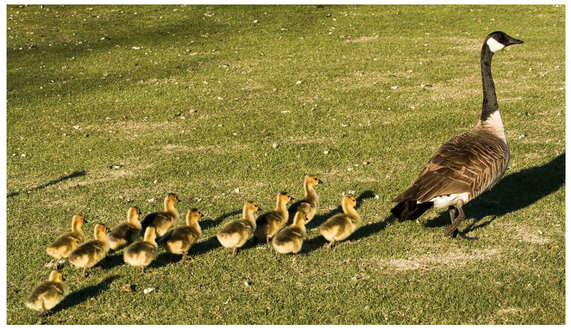
Object Detection as Regression?

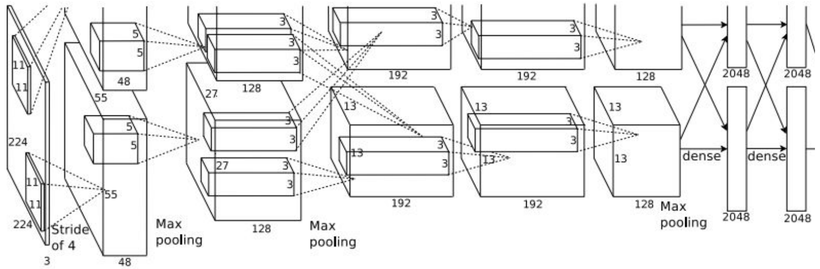


CAT: (x, y, w, h)



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

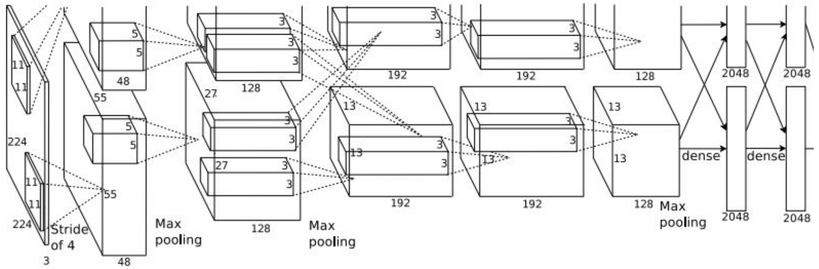




DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

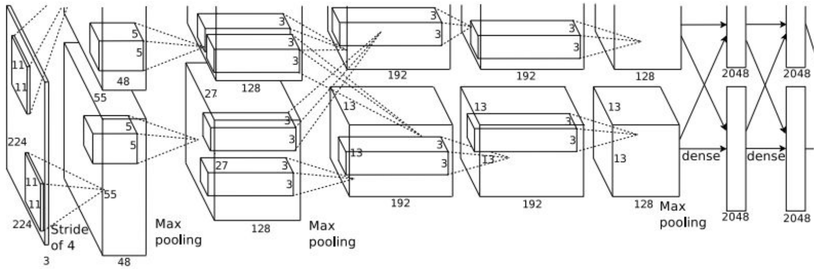
Object Detection as Regression?

Each image needs a different number of outputs!



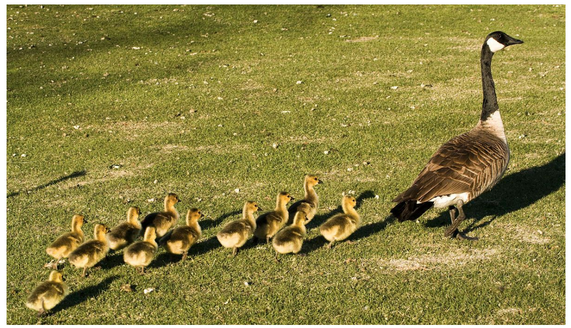
CAT: (x, y, w, h)

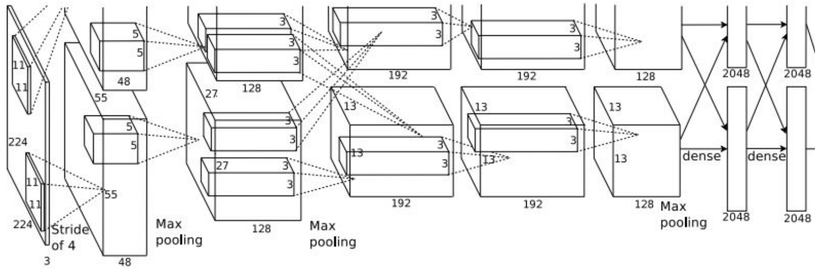
4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

16 numbers

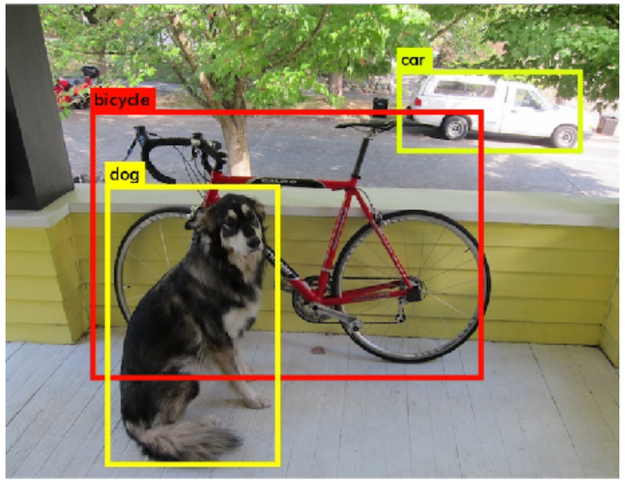




DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

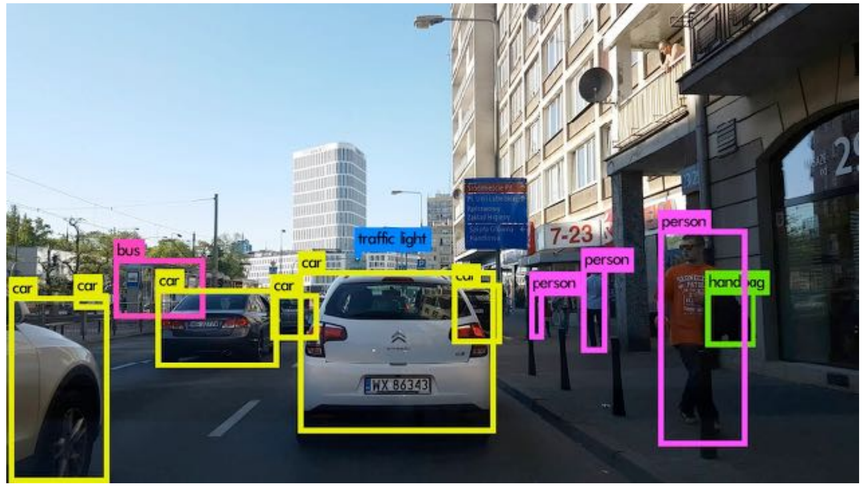
Many numbers!

Object detection as regression?



DOG, (x, y, w, h)
Bicycle, (x, y, w, h)
Car, (x, y, w, h)

Output = 12 numbers



Car, (x, y, w, h)
Car, (x, y, w, h)
Car, (x, y, w, h)
....
Car, (x, y, w, h)

Output = many numbers

Ok if you look at a fixed number of objects in each image..
But we need variable sized outputs

Aside: Human Pose Estimation

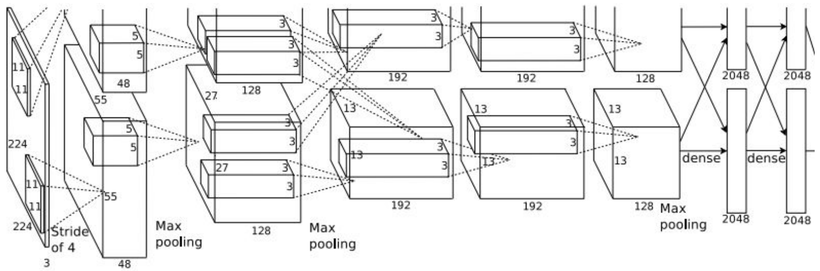
This image is licensed under CC-BY 2.0.

Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

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

Aside: Human Pose Estimation



Vector:
4096

Left foot: (x, y)

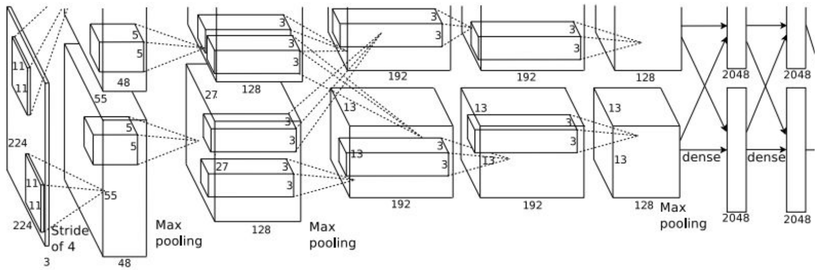
Right foot: (x, y)

…

Head top: (x, y)

Toshev and Szegedy, “DeepPose: Human Pose Estimation via Deep Neural Networks”, CVPR 2014

Aside: Human Pose Estimation



Vector:
4096

Left foot: (x, y) → L2 loss ← **Correct left foot**: (x', y')

Right foot: (x, y) → L2 loss

…

Head top: (x, y) → L2 loss ← **Correct head top**: (x', y')

...

+ → **Loss**

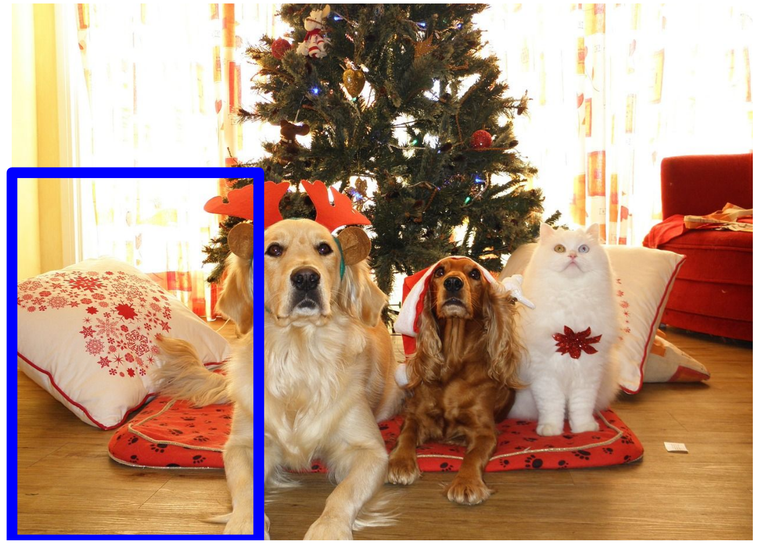
Toshev and Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks", CVPR 2014

From Classification to Detection

Early attempts

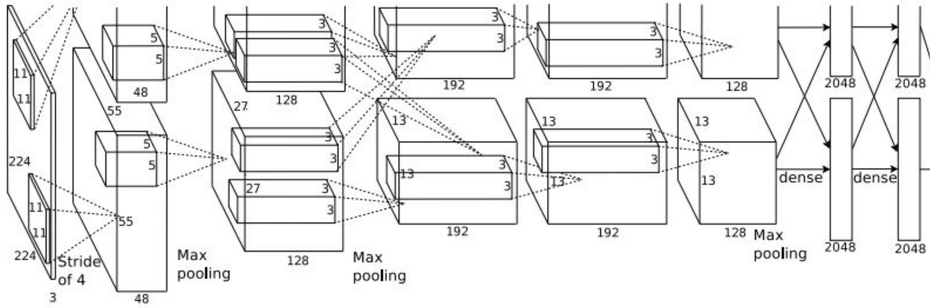
Object Detection as **Classification**

Sliding Window



Object Detection as Classification: Sliding Window

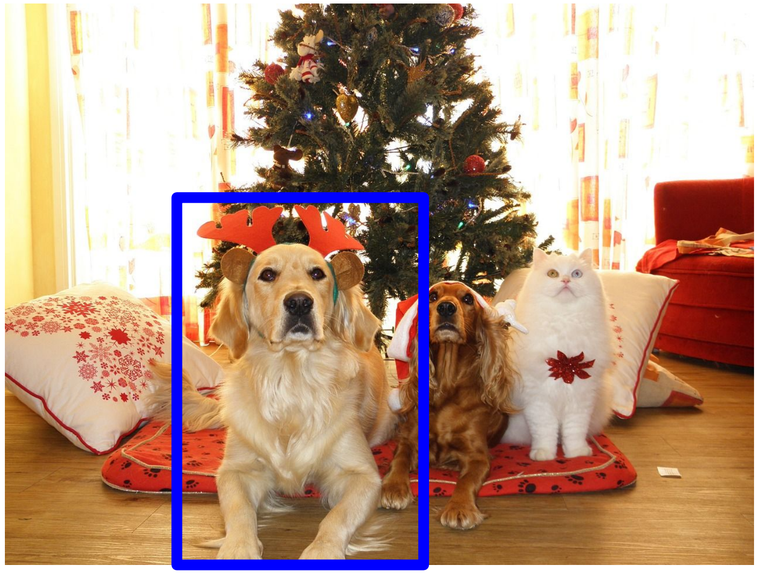
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

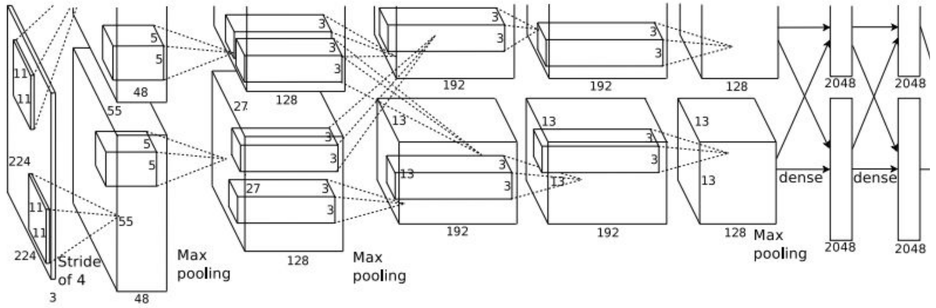


Dog? NO
Cat? NO
Background? YES

Object Detection as Classification: Sliding Window

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

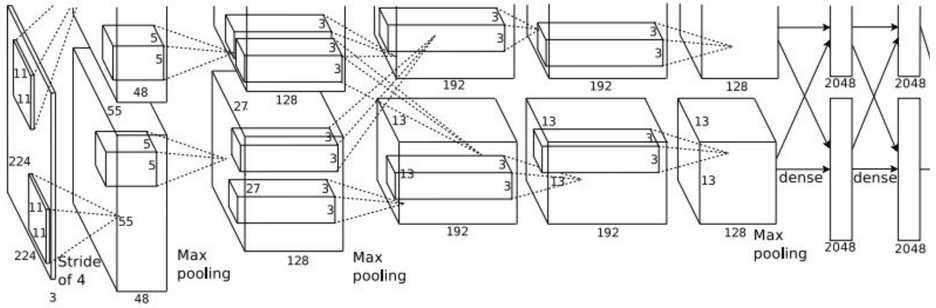




Dog? YES
Cat? NO
Background? NO

Object Detection as Classification: Sliding Window

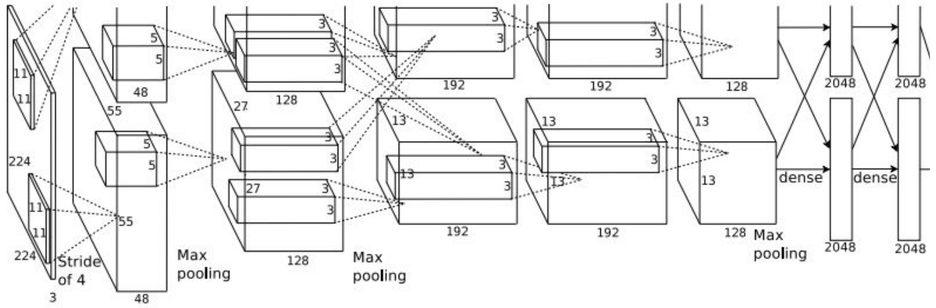
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

Object Detection as Classification: Sliding Window

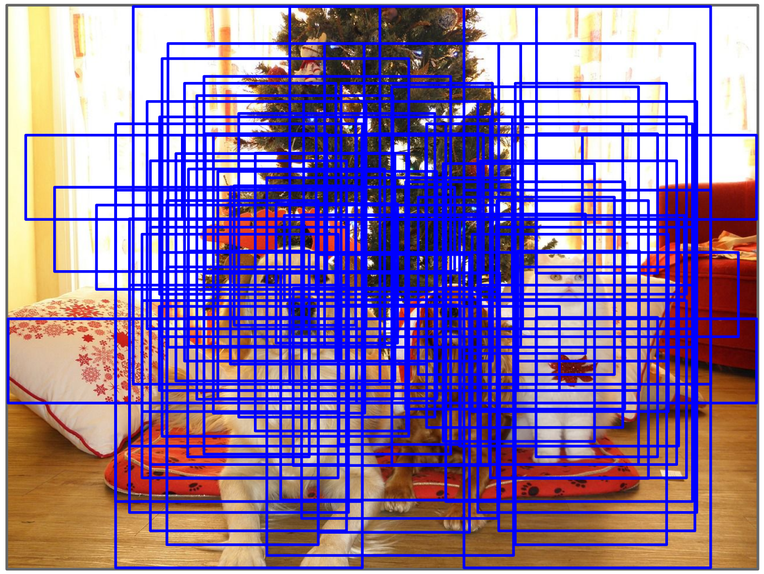
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

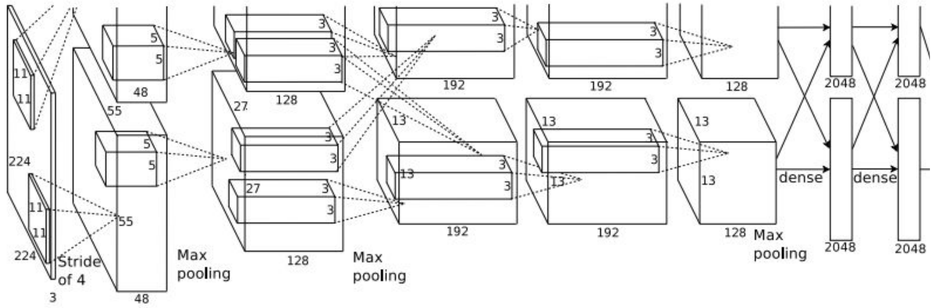


Dog? NO
Cat? YES
Background? NO

Object Detection as Classification: Sliding Window

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



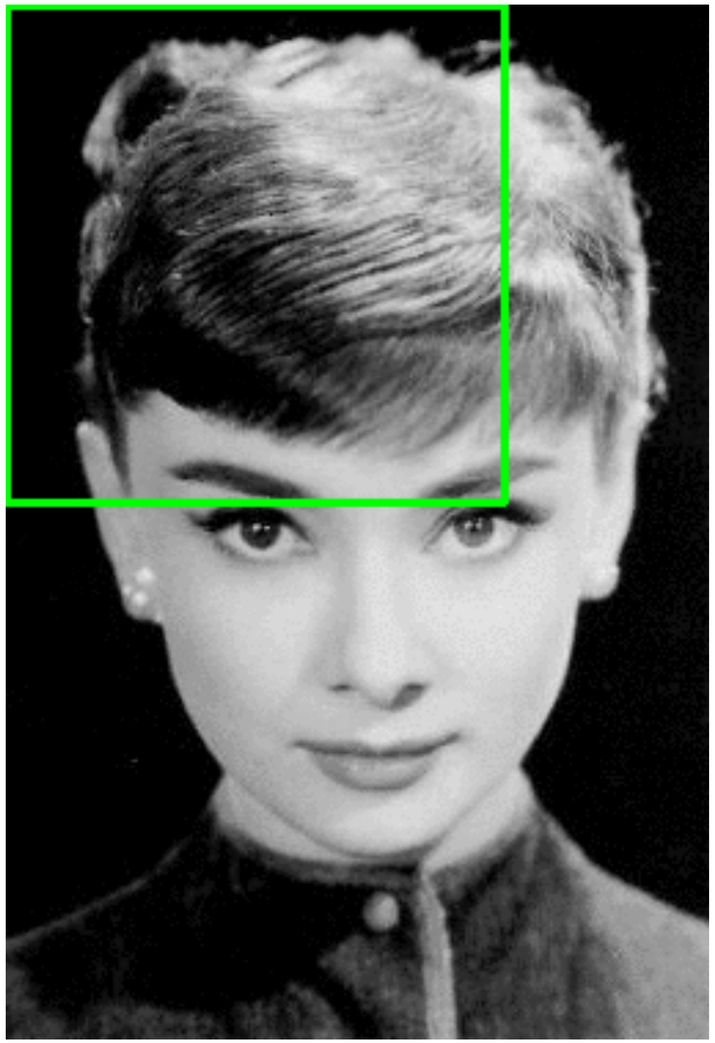


Dog? NO
Cat? YES
Background? NO

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

Sliding Window

- Extending the classification **idea**:
  - Object Detection -> multiple instances of Classification
- **Classify** each window:
  - (C classes + background)
- **Problem**: Many positions, scales (and aspect ratios) to try:
  - -> expensive
- **Solution**:
  - Only look at a tiny **subset** of possible positions

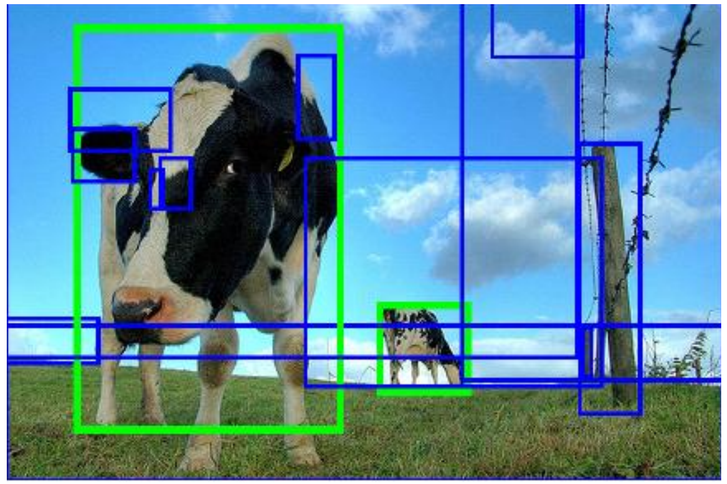


From Classification to Detection

Early attempts

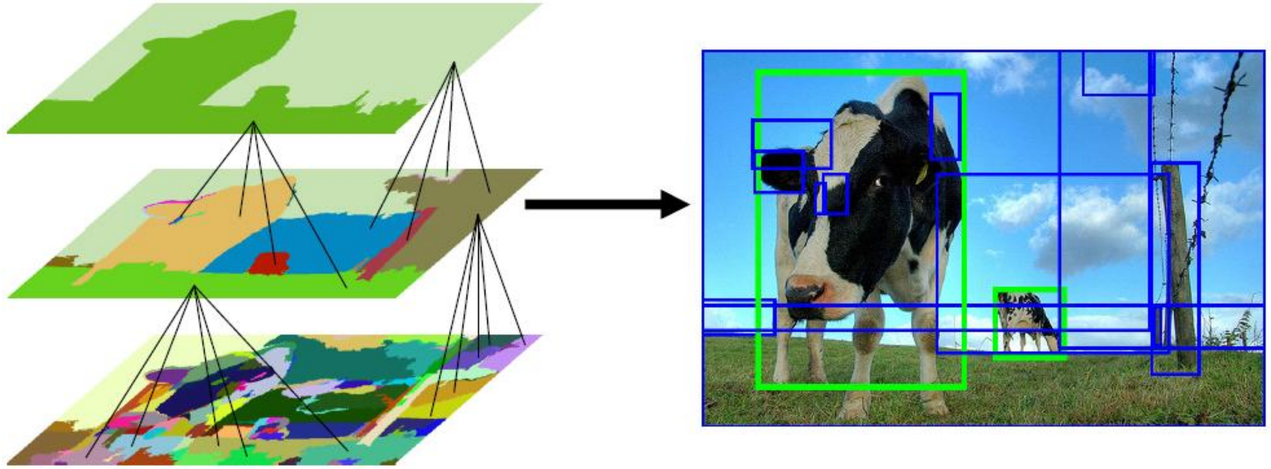
Object Detection as **Classification**

Region proposals



Region Proposals

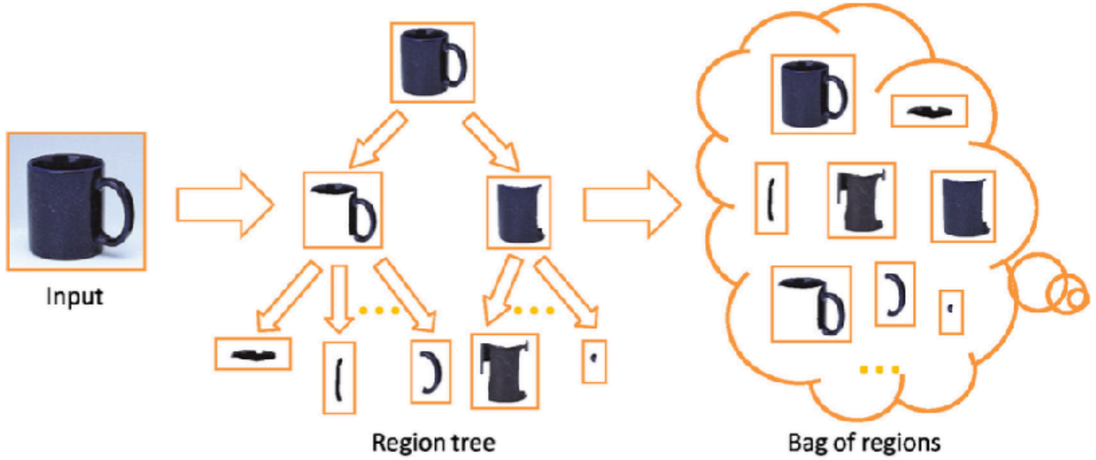
- **Reduce** the number of regions
- Many category independent region **proposal** methods:
  - Objectness
  - **Selective search**
- **Classify** each proposal



Region Proposals (2)

Bottom-up segmentation, merging regions at multiple scales

Hierarchical grouping



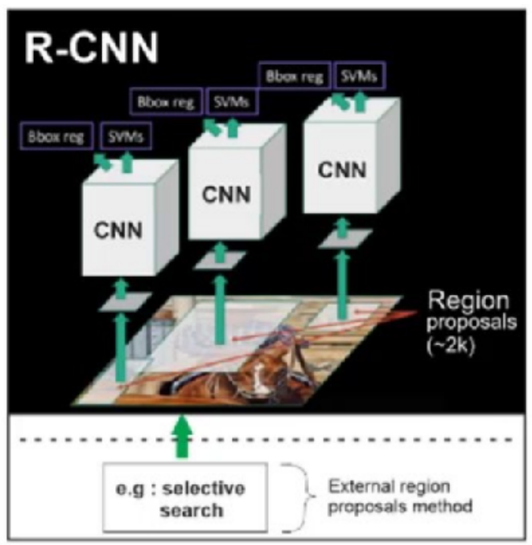
Based on color, texture, contours/shapes

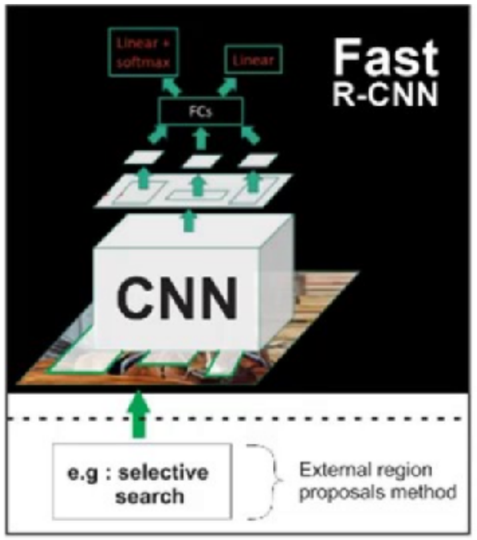
Recognition using regions paper

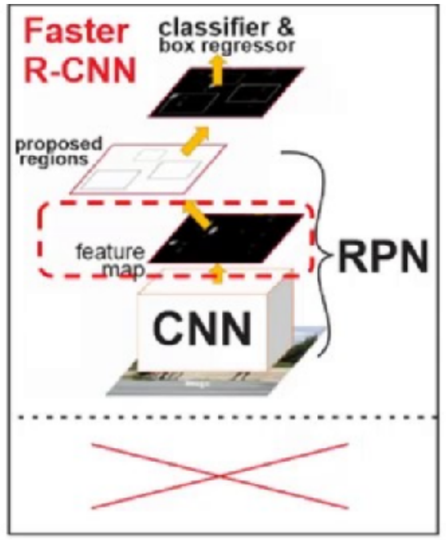
Region Proposals + image classification (CNN (feature extraction)+ SVM (classification)), similar to Sliding Window

Object Detection as Classification
«Two-steps methods»
Region-based CNNs

R-CNN, Fast R-CNN, Faster R-CNN







Region-based CNNs

- Introduced in 2013 (R-CNN) as a way of solving the object detection task
- Main steps:
  - 1. Generate a set of region proposals from an image
  - 2. Process the region proposals using a CNN (feature extraction / learning)
  - 3. Compute task-specific output (classification and bounding boxes)
- R-CNN, Fast R-CNN and Faster R-CNN are all region-based CNNs

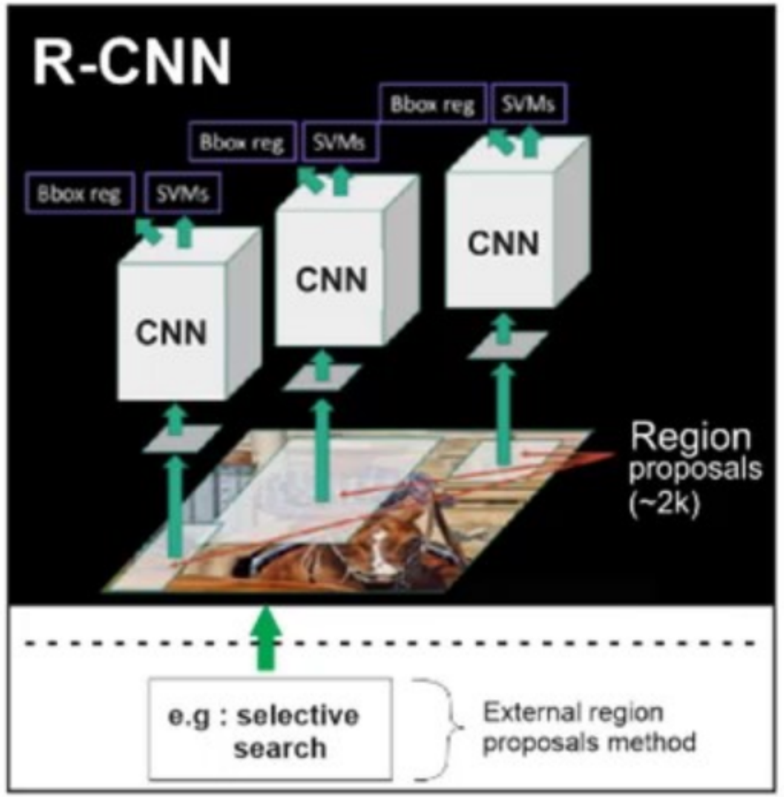
Region-based CNNs

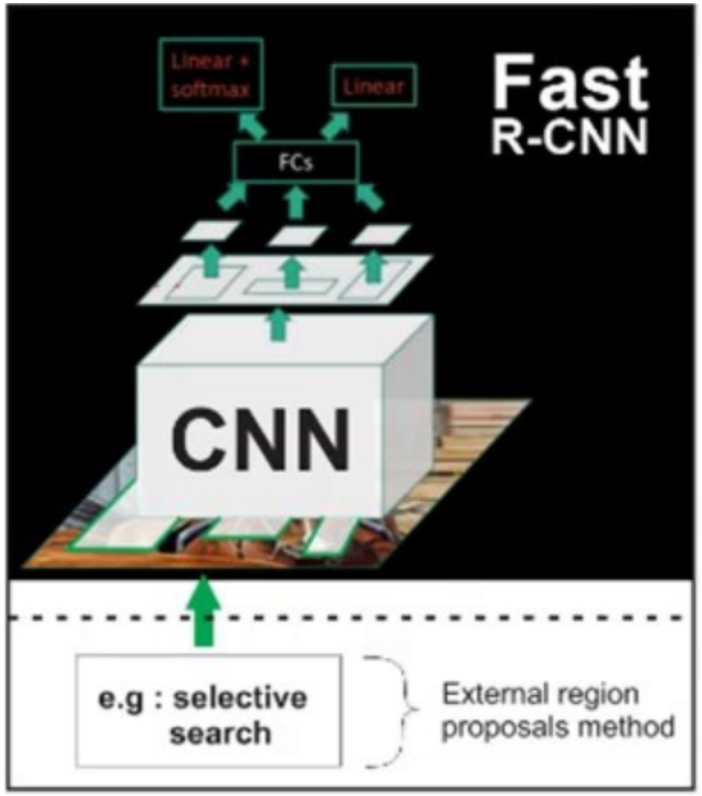
- R-CNN, Fast R-CNN and Faster R-CNN represent incremental improvements in the degree of using CNNs in the different steps

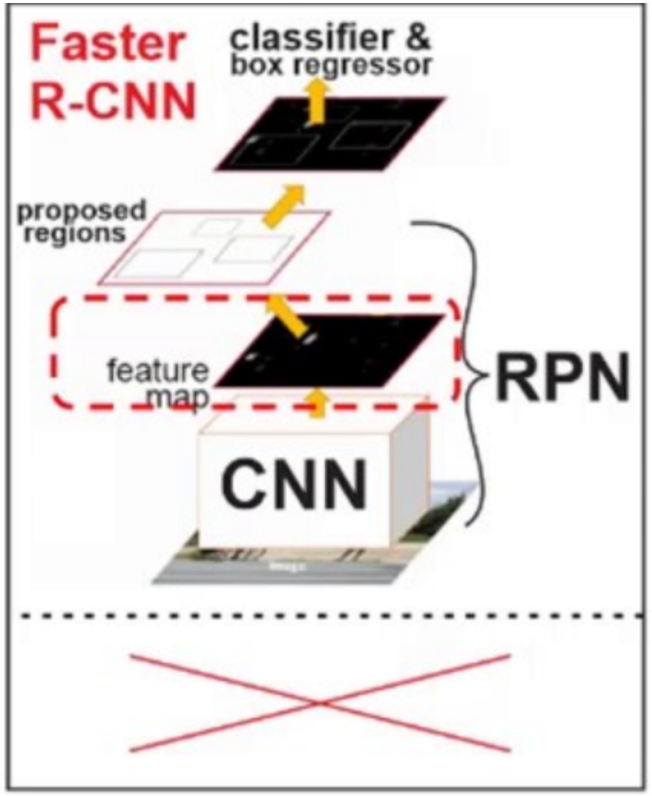
	1. Generate region proposals	2. Process region proposals	3. Compute output
R-CNN	Selective Search	**CNN**	SVM and linear regression
Fast R-CNN	Selective Search	**CNN**	**CNN**
Faster R-CNN	**CNN**	**CNN**	**CNN**

Region-based CNNs

- R-CNN, Fast R-CNN and Faster R-CNN represent incremental improvements in the degree of using CNNs in the different steps







Region-based CNNs

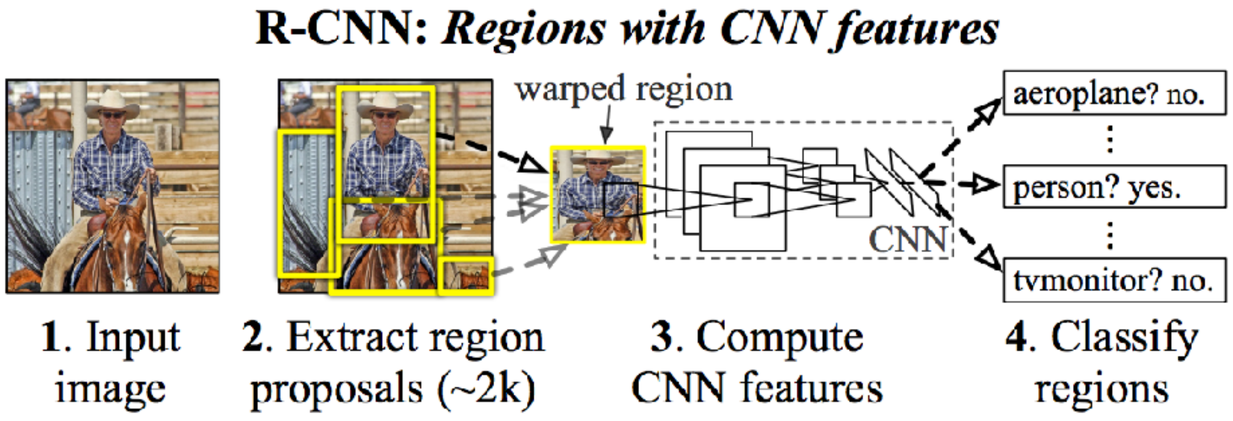
- The use of CNNs in a more unified pipeline has resulted in a massive speed-up and improved accuracy

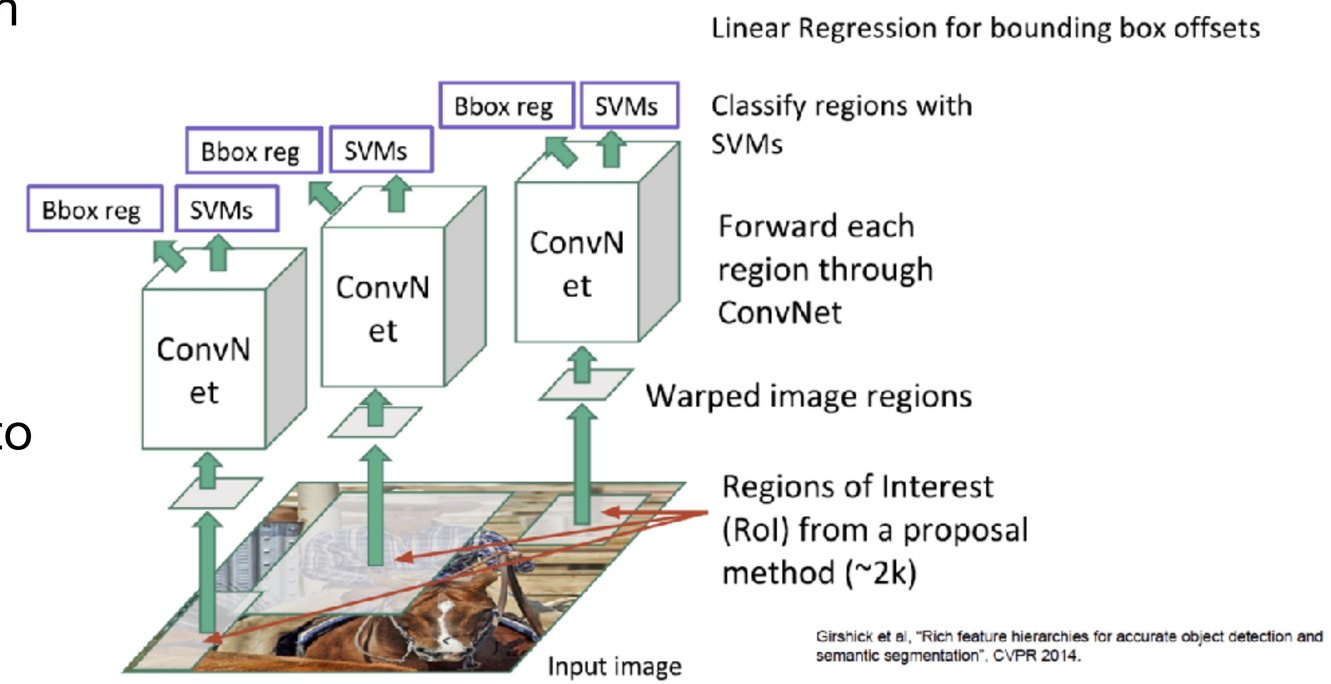
	Test time per image	Speed-up	Accuracy (mAP on VOC2007)
R-CNN	50 s	1x	66.0%
Fast R-CNN	2 s	25x	66.9%
Faster R-CNN	0.2 s	250x	69,9%

R-CNN

R-CNN

- Purpose: Object detection
  - Two-part, bounding box around object and then classification
- Bounding box **proposals** with Selective Search
  - looks at the image through windows of different sizes, and for each size tries to **group** together adjacent pixels by texture, color, or intensity to identify objects.
- Proposals are warped to squares and passed to a convnet, e.g. AlexNet



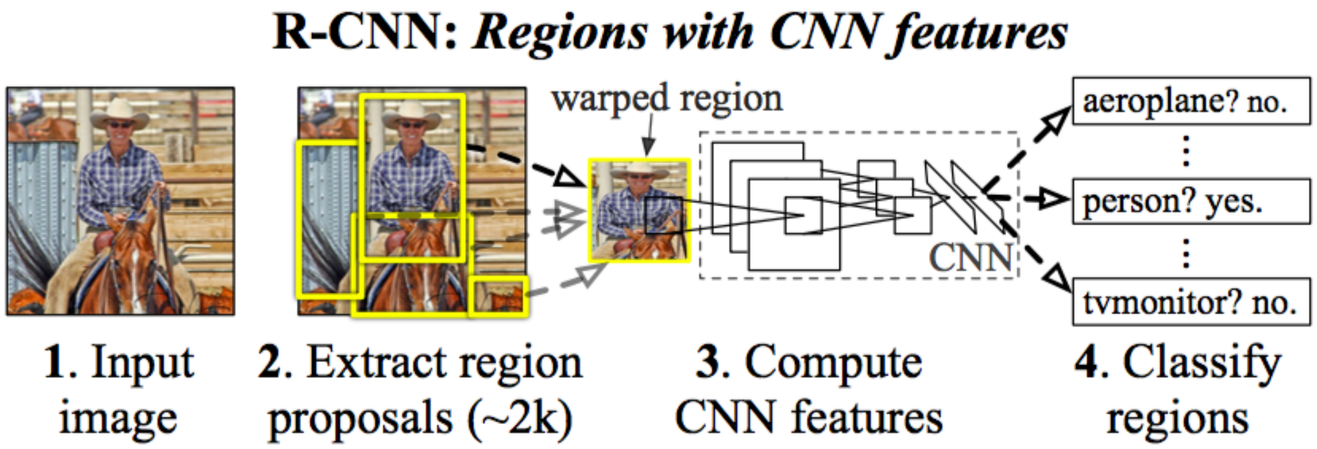


Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

R-CNN (2)

- At test time, generate **2000 category-independent** regions of interest
- System is agnostic to region proposal method, but paper uses '**Selective Search**' to compare with earlier papers.
- Regardless of region dimensions (size), it is **converted** to a mean-subtracted 227x227 RGB image. Conversion is done by affine image warping
- **Warped images** are passed to a convolutional net, feed forwarded, and a **fixed length feature vector** is extracted (length 4096). Class specific SVMs classify the region
- Class specific computation is dot product between features and SVM weights (2000x4096 and 4096xN (N = number of classes))
- During detection, **reject** regions with **IoU** with a higher-scoring region above a certain threshold.

R-CNN (3)



- R-CNN
  - Method consists of three modules
    - Region proposals using selective search
    - Feature extraction using CNN
    - Classification using SVM
  - Region proposal method can be replaced by virtually any method for extracting region proposals
  - The extracted regions (~2k) are resampled to a fixed 227x227 RGB image
  - Each warped region is passed through a 5-layer CNN in order to extract features
  - For each class, a score is given to the feature vectore by the corresponding SVM
  - Lastly, for each class, greedy non-maximum supression is used to reject regions with an IoU overlap with a higher scoring region based on a learned threshold

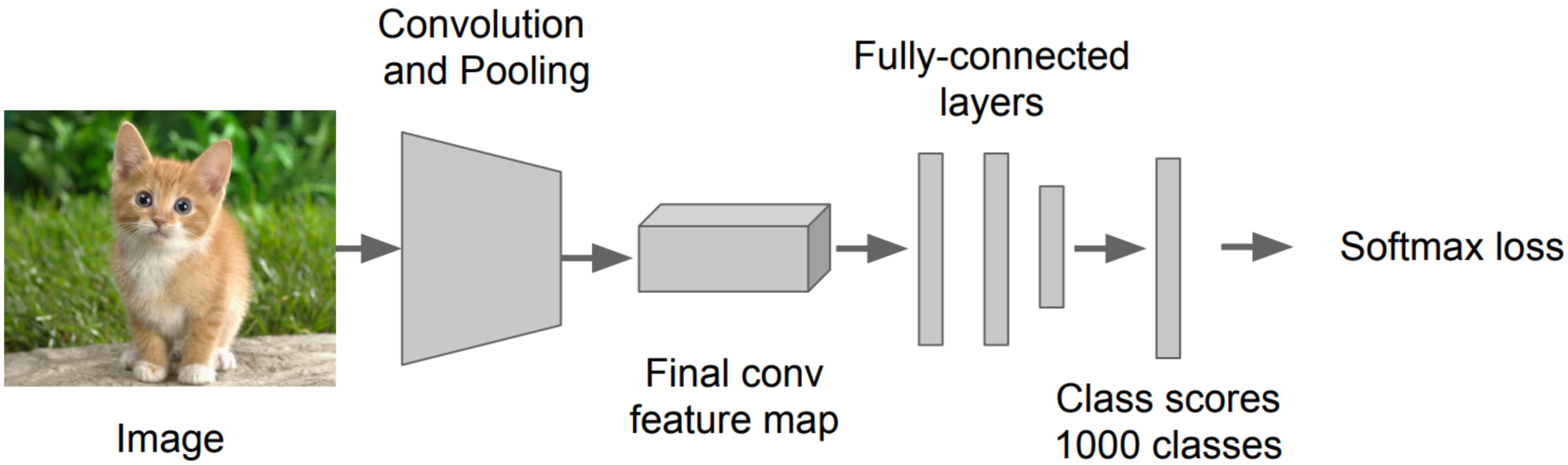
R-CNN (4)

- R-CNN
  - Training
    - Training of the CNN was done in two steps
    - Pre-trained on the ILSVRC 2012 dataset
      - Image-level annotations only
      - No bounding boxes
    - Domain-specific fine-tuning
      - Continue training of CNN parameters on warped region proposals from PASCAL VOC
      - Learning rate of 0.001

R-CNN Training

Step 1: Train or download a classification model for ImageNet (AlexNet)

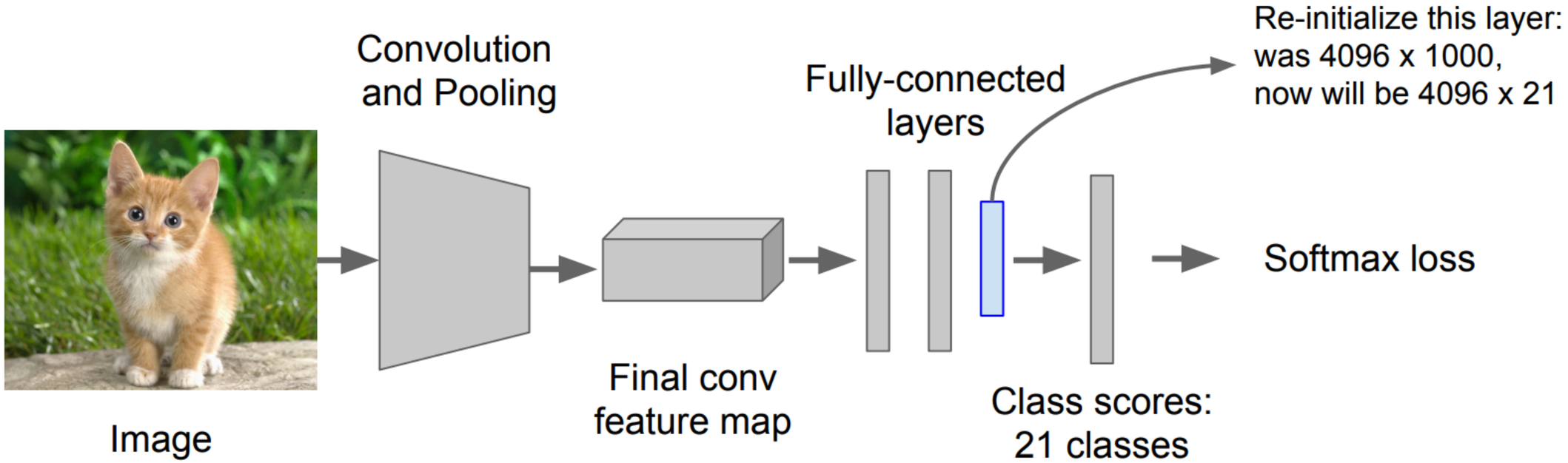
- Image net has 1000 classes



R-CNN Training (2)

Step 2: Fine-tune model for detection

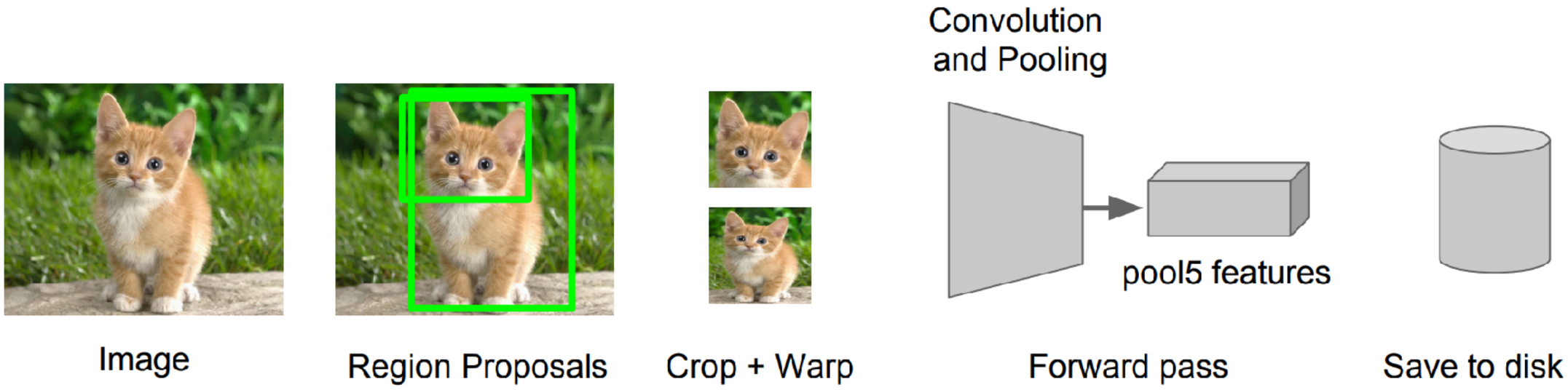
- Instead of 1000 classes choose X classes you want to detect (e.g. 20 + background)
- Throw away final fully-connected layer, reinitialize from scratch
- Keep training



R-CNN Training (3)

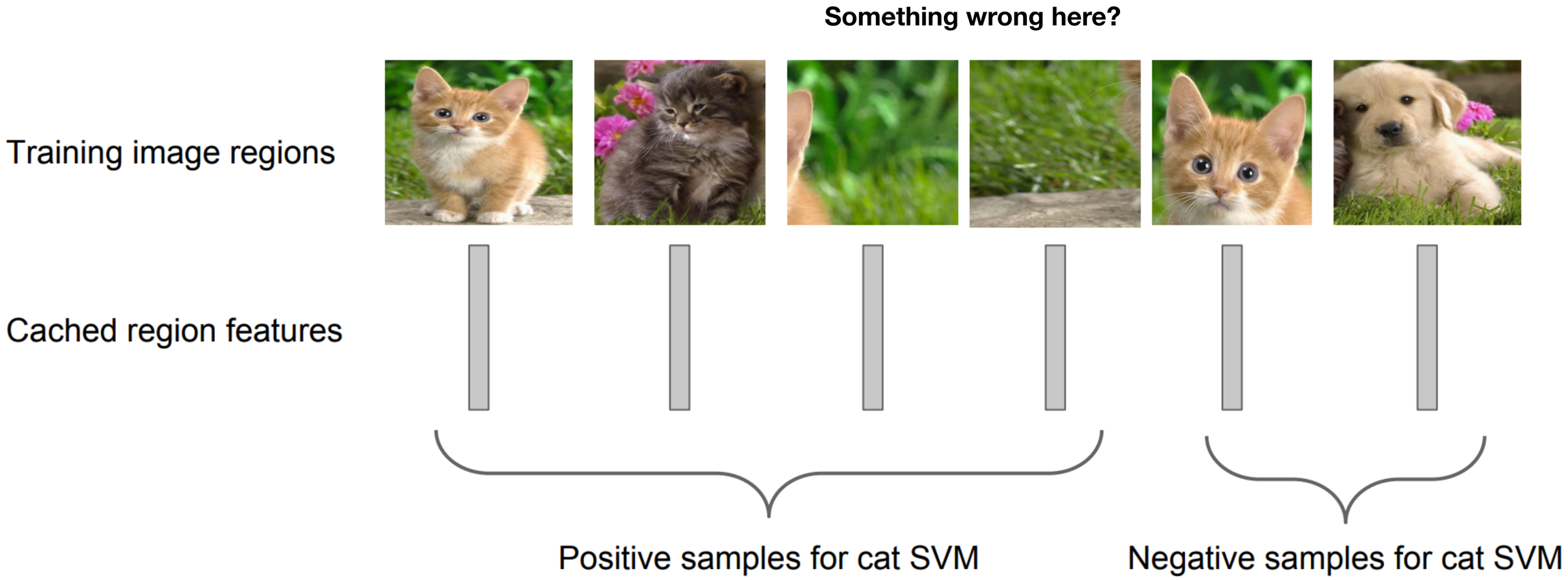
Step 3: Extract features

- Save features to disk (~200GB for PASCAL dataset)



R-CNN Training (4)

Step 4: Train one binary SVM per class to classify region features



R-CNN Training (5)

Step 5(bbox regression): For each class, train a linear regression model to map from cached features to offsets

Training image regions

Cached region features

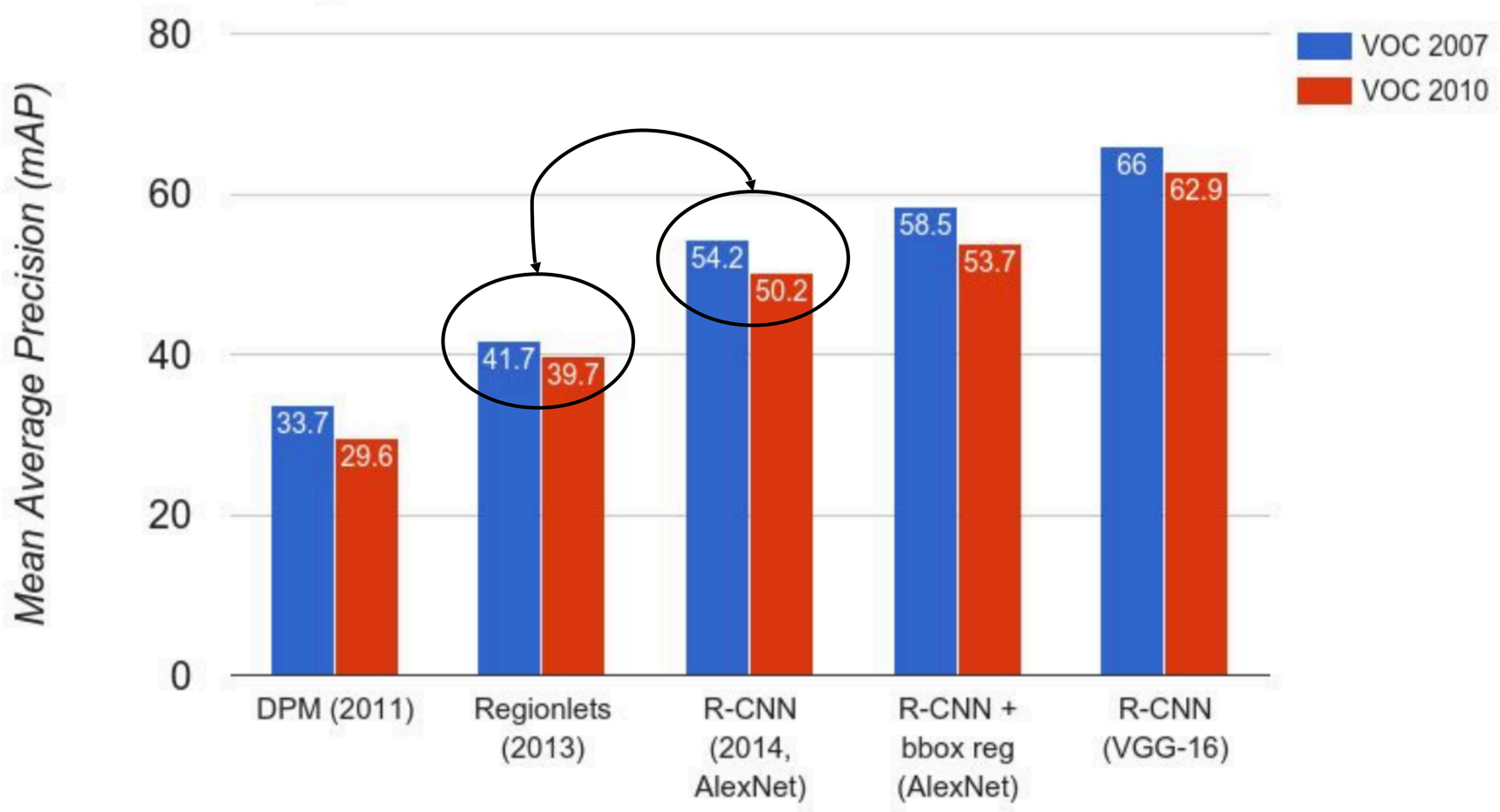
Regression targets
(dx, dy, dw, dh)
Normalized coordinates

(0, 0, 0, 0)
Proposal is good

(.25, 0, 0, 0)
Proposal too
far to left

(0, 0, -0.125, 0)
Proposal too
wide

R-CNN Results



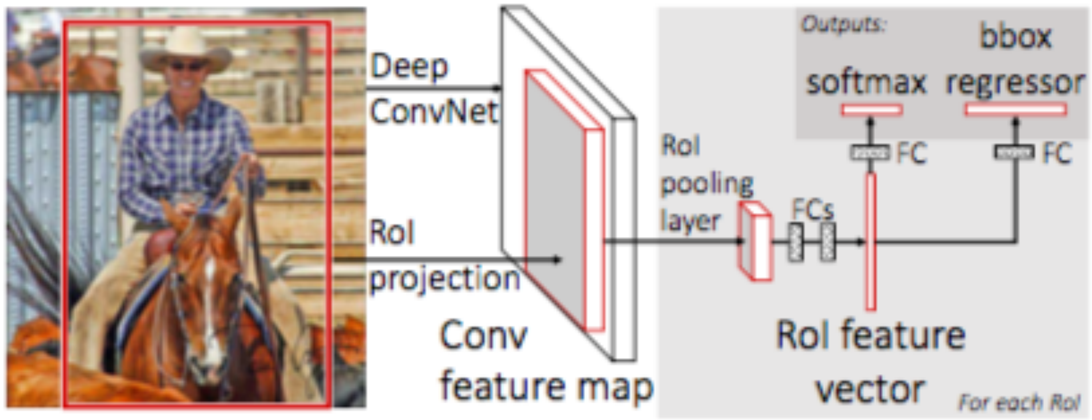
R-CNN Problems

- **Slow** at **test** time: need to run full forward passes of CNN for each region proposal (~2000)
- SVMs and regressors are trained **separately**: CNN weights can't update in response to their objective
- Complex **multistage** training pipeline (saving to disk)

Fast R-CNN

Fast R-CNN

- Fast R-CNN
  - Seeks to improve on some of the issues with R-CNN
    - Multi-stage pipeline training
    - Expensive training
    - Slow object detection
  - Takes an entire image and a set of object proposals as input
  - Features extracted using VGG16
  - **RoI pooling** layer extracts fixed-length feature vector from the feature map
  - A softmax output layer outputs object classification **scores**
  - Another output layer outputs 4 values that **refine** the bounding-box of the object



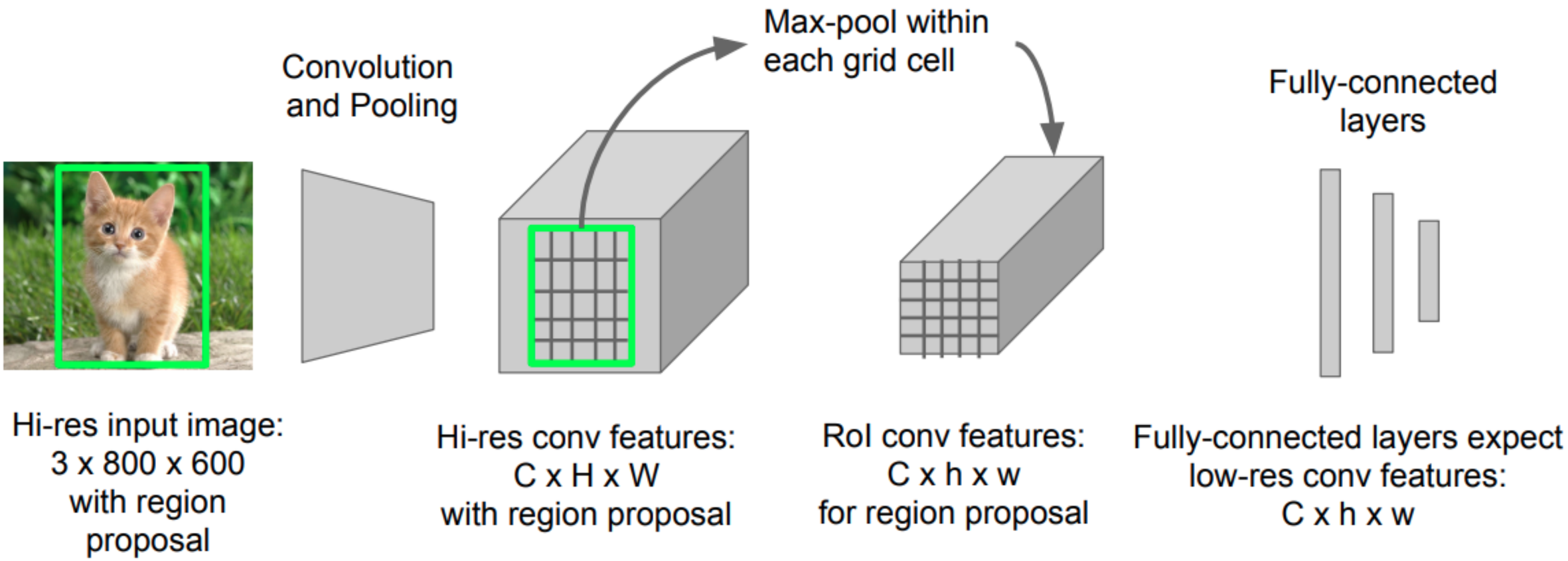
Fast R-CNN (2)

- Fast R-CNN
  - RoI pooling
    - Uses max-pooling to convert features inside a region of interest into a small, fixed-size feature map of size *H*x*W*, where *H* and *W* are hyper-parameters
    - A region of interest is defined by the tuple (*r*, *c*, *h*, *w*)
      - Top-left corner (*r*, *c*)
      - Height and width (*h*, *w*)
    - A region of interest of arbitrary size is divided into subwindows of approx. size *h/H x w/W*
      - Each subwindow is max-pooled into the corresponding cell of the *HxW* feature map
    - Applied independently to each feature map channel

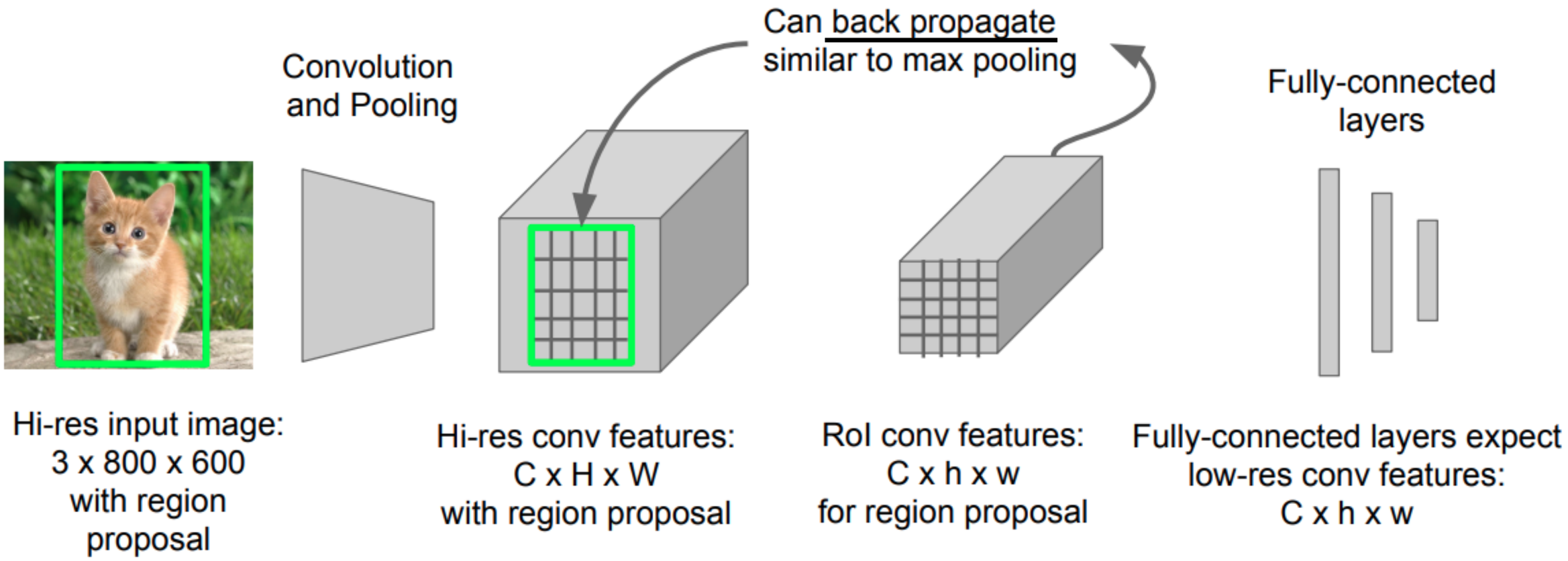
Fast R-CNN (3)

- Fast R-CNN
  - Able to use pre-trained CNNs for feature extraction with few modifications
    - Replace last max-pooling layer with RoI-pooling
    - Replace output-layer with two sibling layers (softmax classifier and bbox regressor)
    - Modify network to accept two inputs, image and RoI proposals
  - Training is made faster, as the entire network can be trained at once
    - Multi-task loss
      - Trains for classification and bounding box regression at the same time
    - Backpropagation through RoI-pooling layers
      - Utilizes argmax switches to determine if a specific neuron should be updated
  - Reduced test-time speed due to shared computation
    - Feature map is computed once for entire image
  - Still has some issues due to the fact that it uses pre-computed region proposals

Fast R-CNN improvement: ROI-Pooling



Fast R-CNN improvement: ROI-Pooling



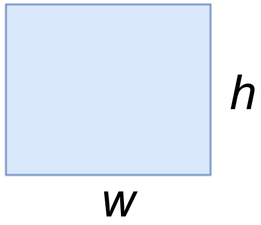
Numerical Example

ROI-Pooling

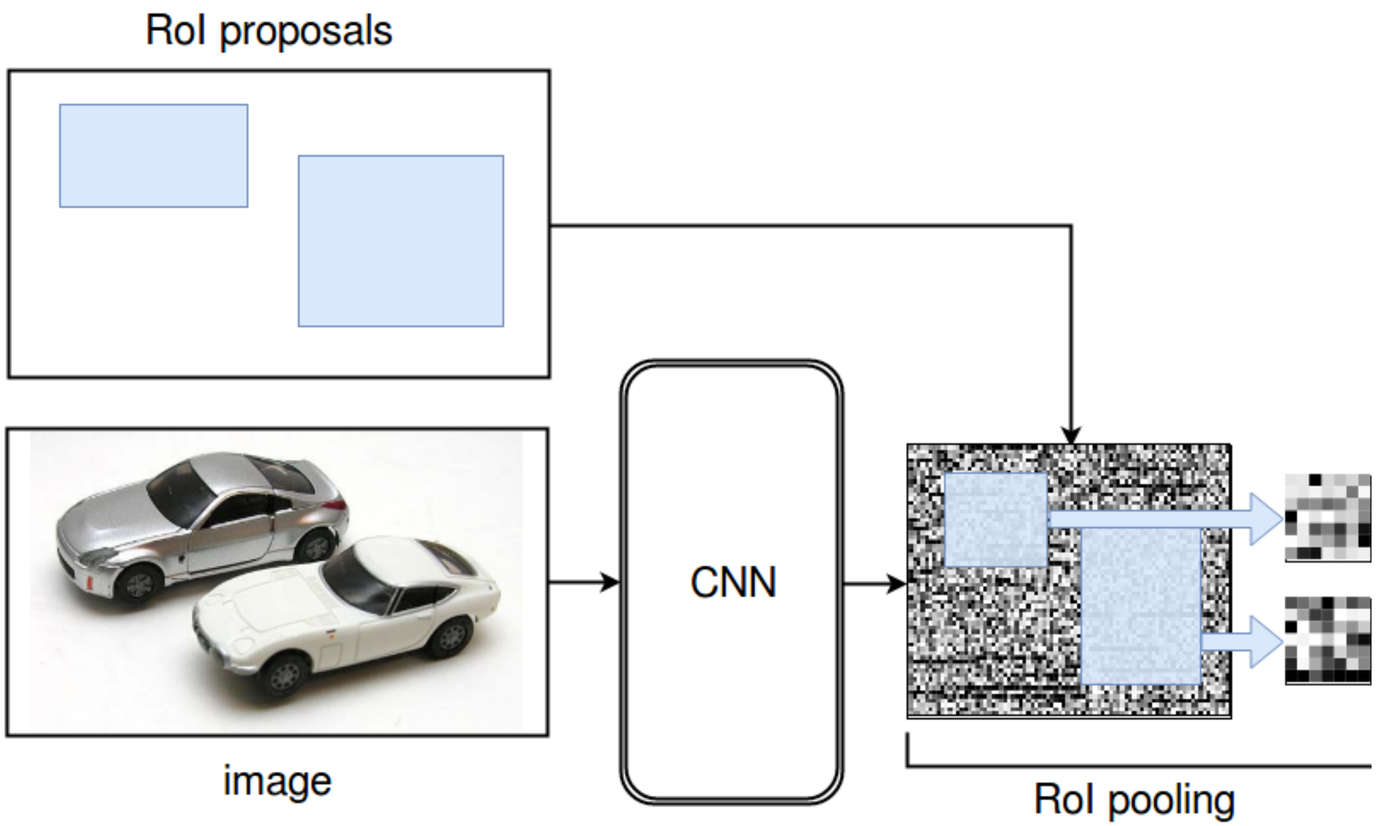
RoI Pooling

- RoI pooling is responsible for extracting a **fixed** size feature map from an arbitrary sized region proposal

Region proposal:
- Arbitrary size (*h* x w)



Output feature:
- Fixed size (H x W)



RoI Pooling

- We have:

Input feature map:

0.88	0.44	0.14	0.16	0.37	0.77	0.96	0.27
0.19	0.45	0.57	0.16	0.63	0.29	0.71	0.70
0.66	0.26	0.82	0.64	0.54	0.73	0.59	0.26
0.85	0.34	0.76	0.84	0.29	0.75	0.62	0.25
0.32	0.74	0.21	0.39	0.34	0.03	0.33	0.48
0.20	0.14	0.16	0.13	0.73	0.65	0.96	0.32
0.19	0.69	0.09	0.86	0.88	0.07	0.01	0.48
0.83	0.24	0.97	0.04	0.24	0.35	0.50	0.91

RoI coordinates: ($\text{top_left}_x = 0$,
$\text{top_left}_y = 3$,
$h = 5$
$w = 7$)

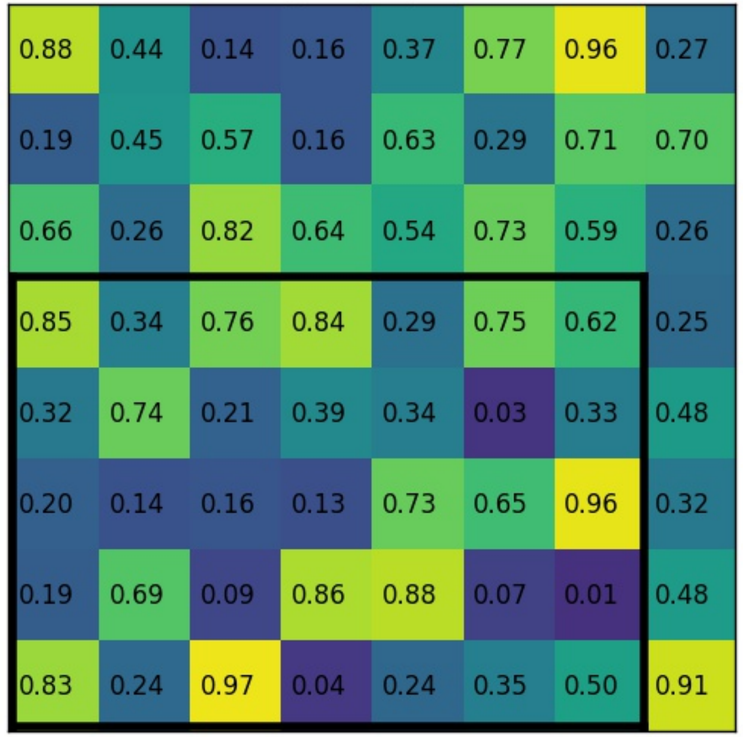
Output feature size: 2 x 2 ($H = 2$, $W = 2$)

RoI Pooling

- We have:

RoI coordinates: (top_left_x = 0, top_left_y = 3, height = 5, width = 7)

1. Obtain RoI from feature map



RoI Pooling

- We have:

RoI size: 5 x 7 (h = 5, w = 7) Output feature size: 2 x 2 (H = 2, W = 2)

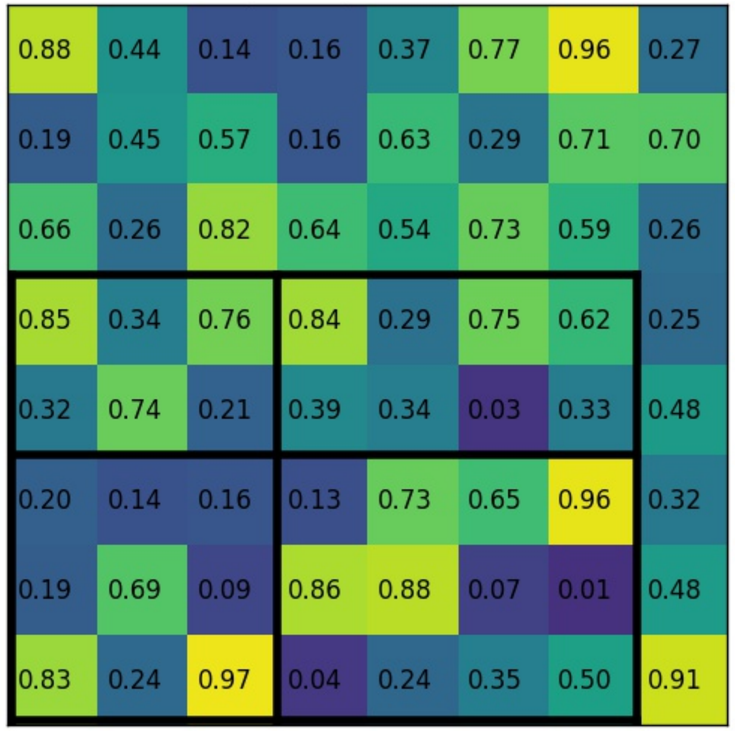
2. Divide RoI into pooling sections corresponding with output feature size

   a) Calculate size of pooling sections (quantization):

$$\left\lfloor \frac{h}{H} \right\rfloor = \left\lfloor \frac{5}{2} \right\rfloor = \lfloor 2.5 \rfloor = 2$$

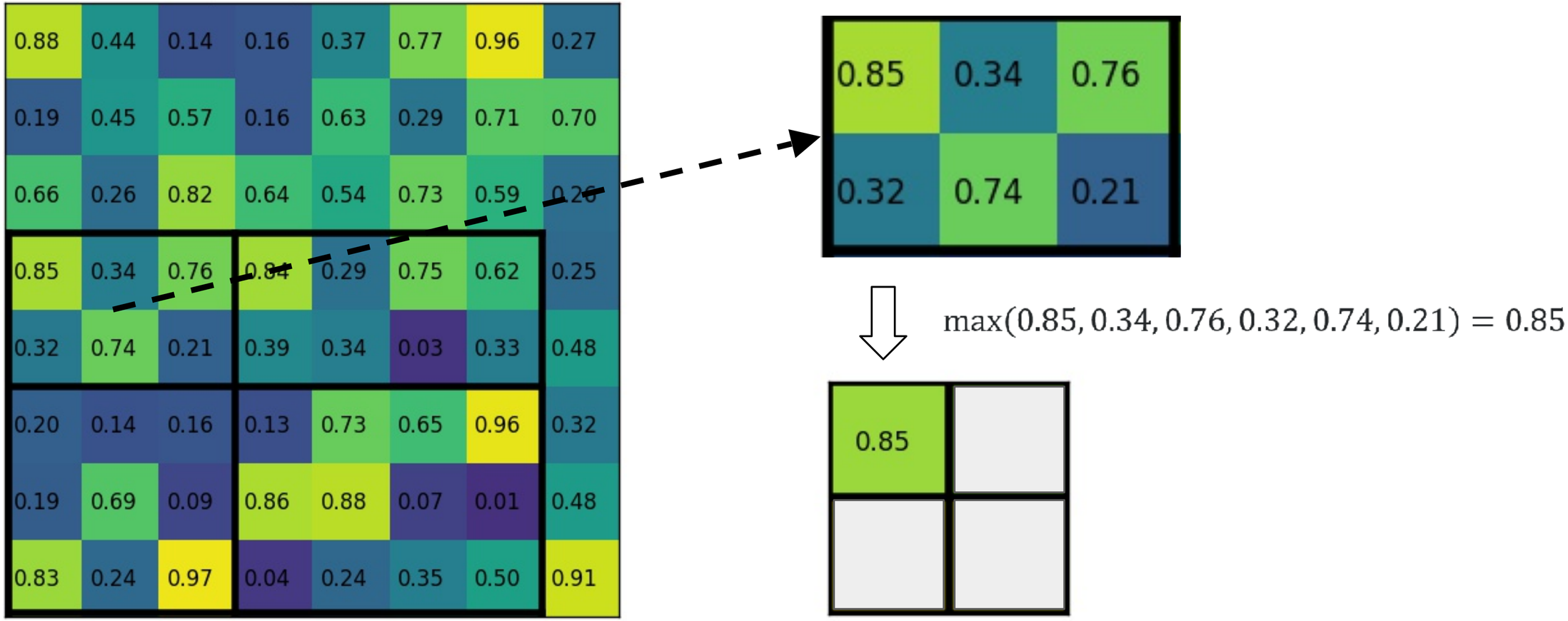
$$\left\lfloor \frac{w}{W} \right\rfloor = \left\lfloor \frac{7}{2} \right\rfloor = \lfloor 3.5 \rfloor = 3$$

   a) Top-left section assigned the calculated size while the other sections are larger because of the section size not perfectly matching the RoI size



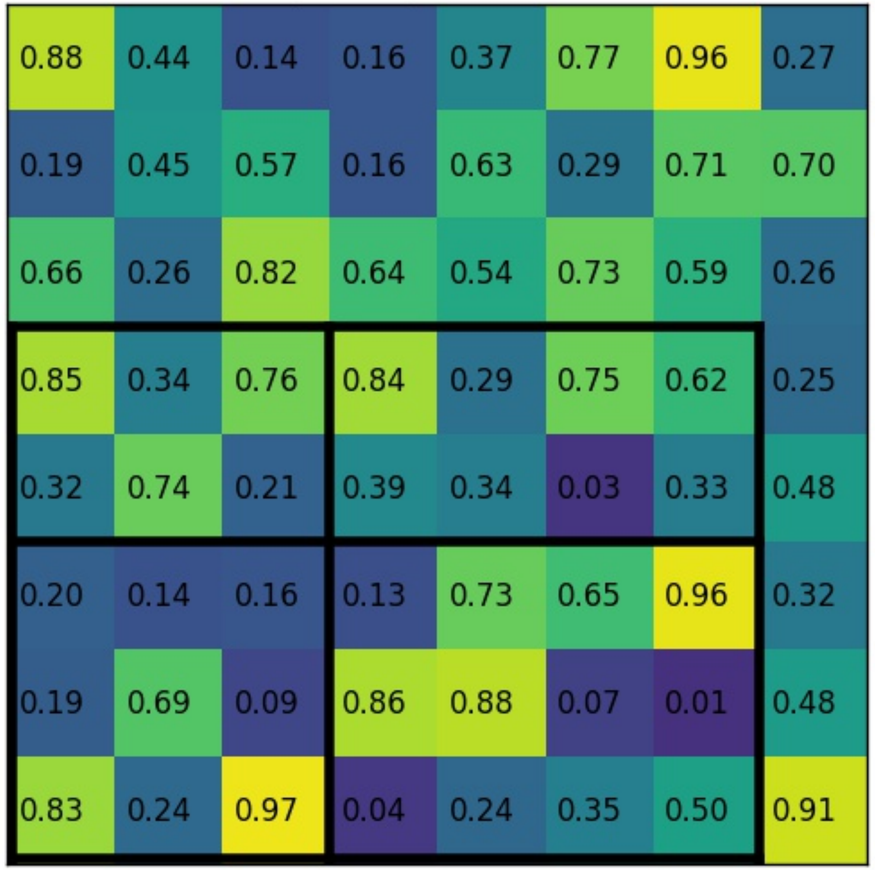
RoI Pooling

3. Apply max pooling on the assigned sections to obtain output feature representing the region proposal (RoI)

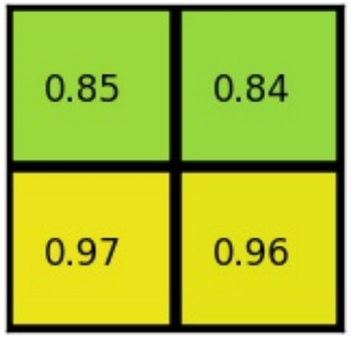


RoI Pooling

3. Apply max pooling on the assigned sections to obtain output feature representing the region proposal (RoI)



Output feature



Faster R-CNN

Faster R-CNN: Motivation

- Before Faster R-CNN **region proposal** computation was regarded the bottleneck
  - Fast R-CNN: CNN efficiently computing detections given region proposals
  - External algorithm (selective search) still used for generating proposals
    - **10x** slower than the detection network

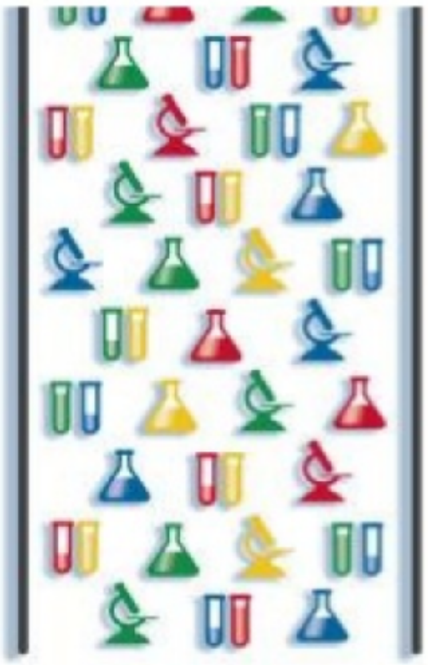
Detection network
(Fast R-CNN)

+

Selective Search

Faster R-CNN: Motivation (2)

- Main contribution of Faster R-CNN: Nearly **cost-free** region proposals
  - Region Proposal Network (**RPN**) used to generate region proposals
  - RPN share convolutional layers with the detection network
  - Only a cost of 10 ms per image

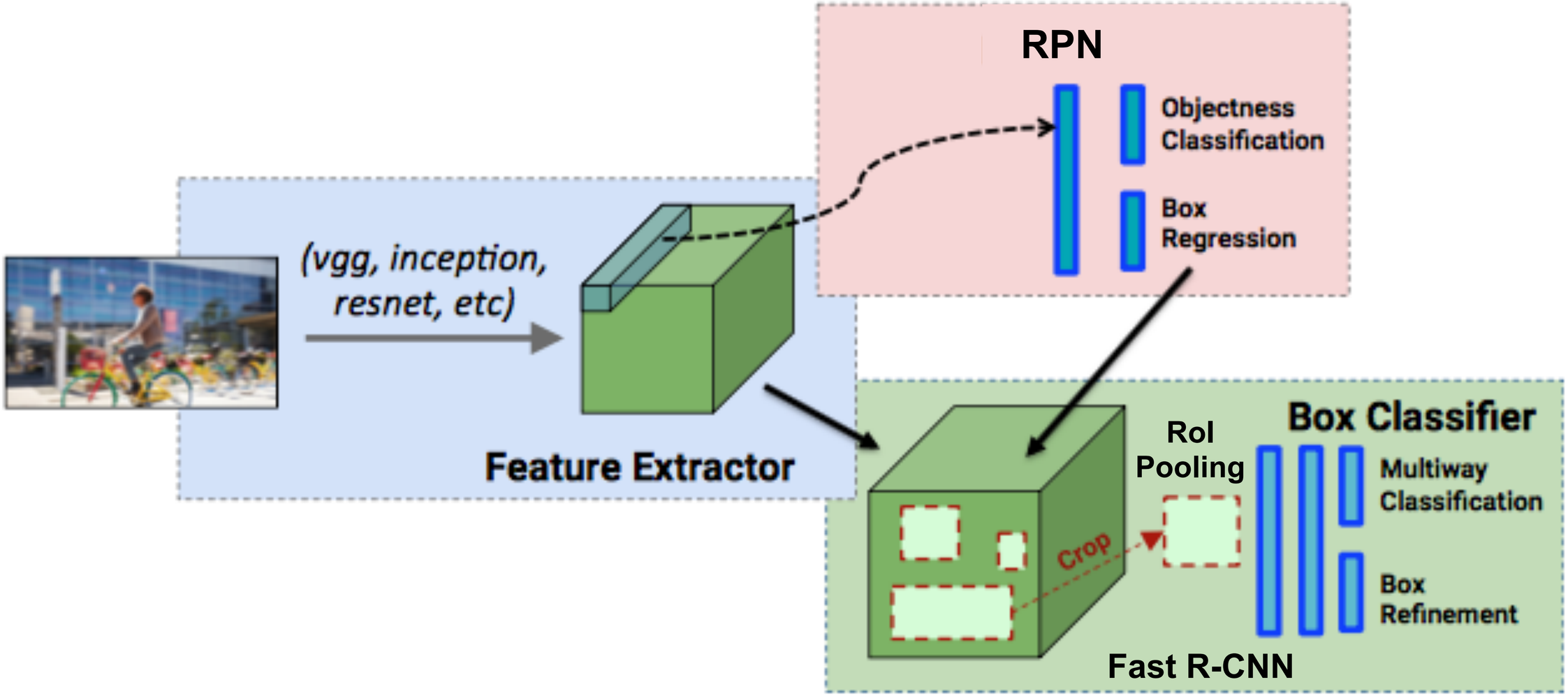


Detection network
(Fast R-CNN)

+

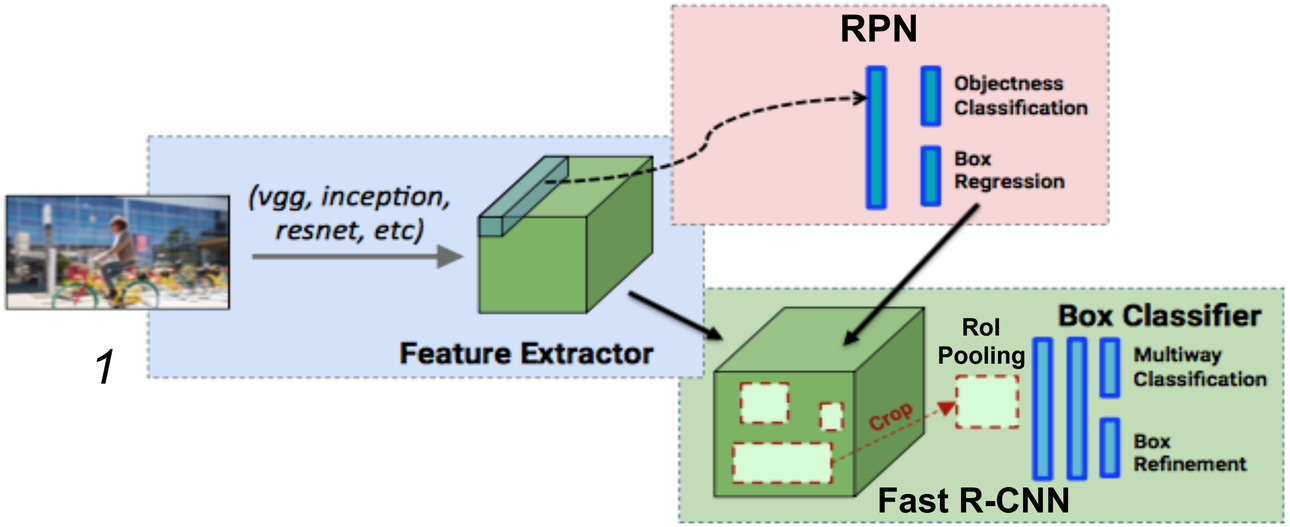
RPN

Faster R-CNN: Architecture



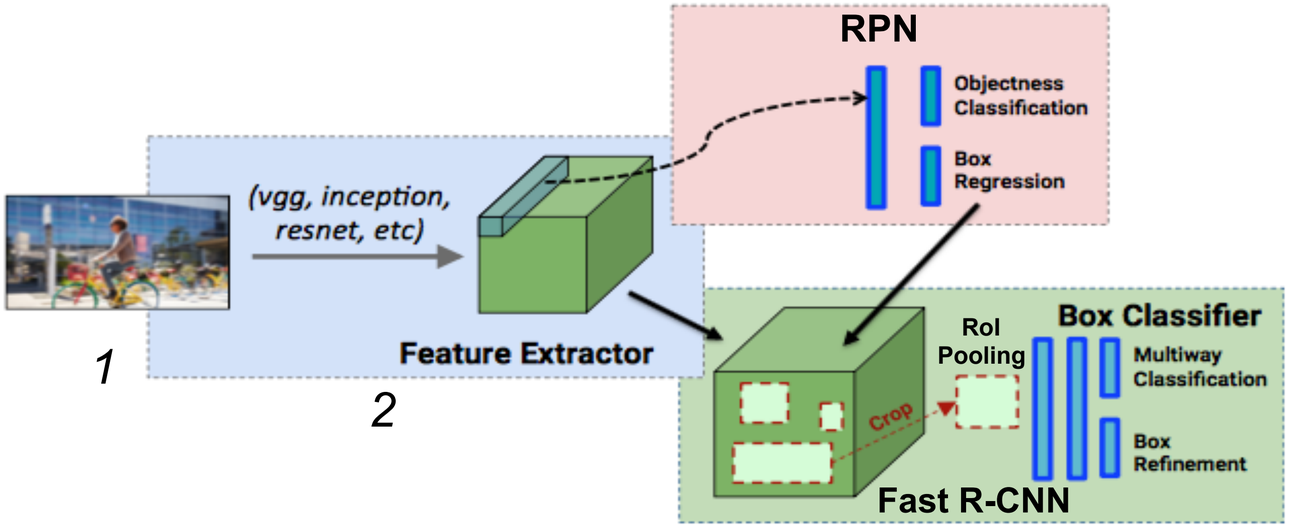
Faster R-CNN: Pipeline

1. Input image of any size



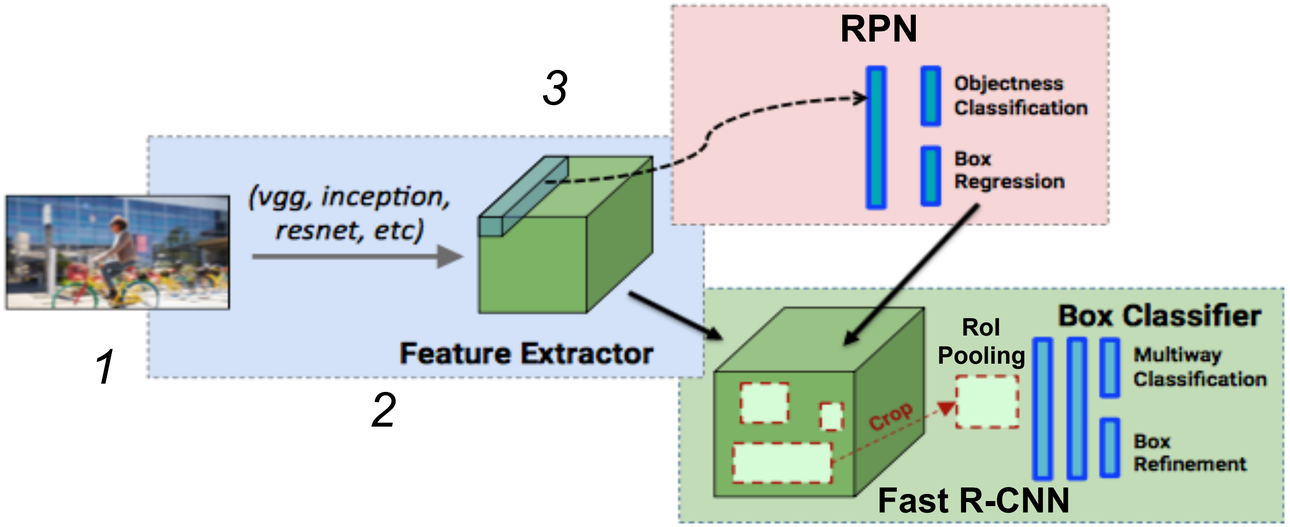
Faster R-CNN: Pipeline

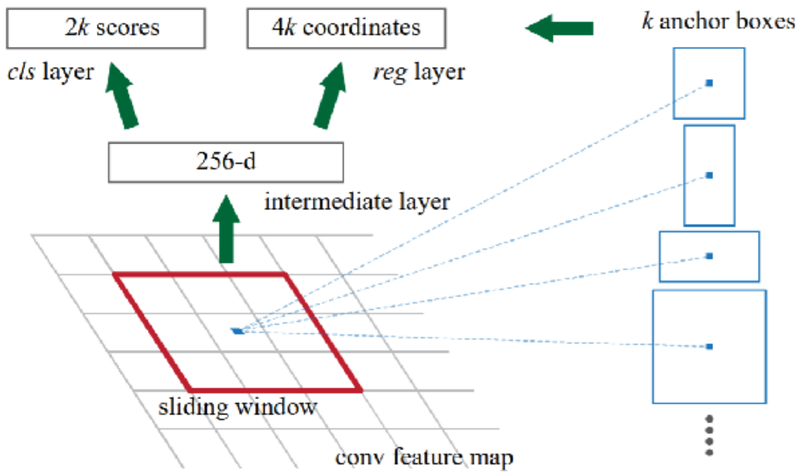
1. Input image of any size
2. Extract features using VGG-16



Faster R-CNN: Pipeline

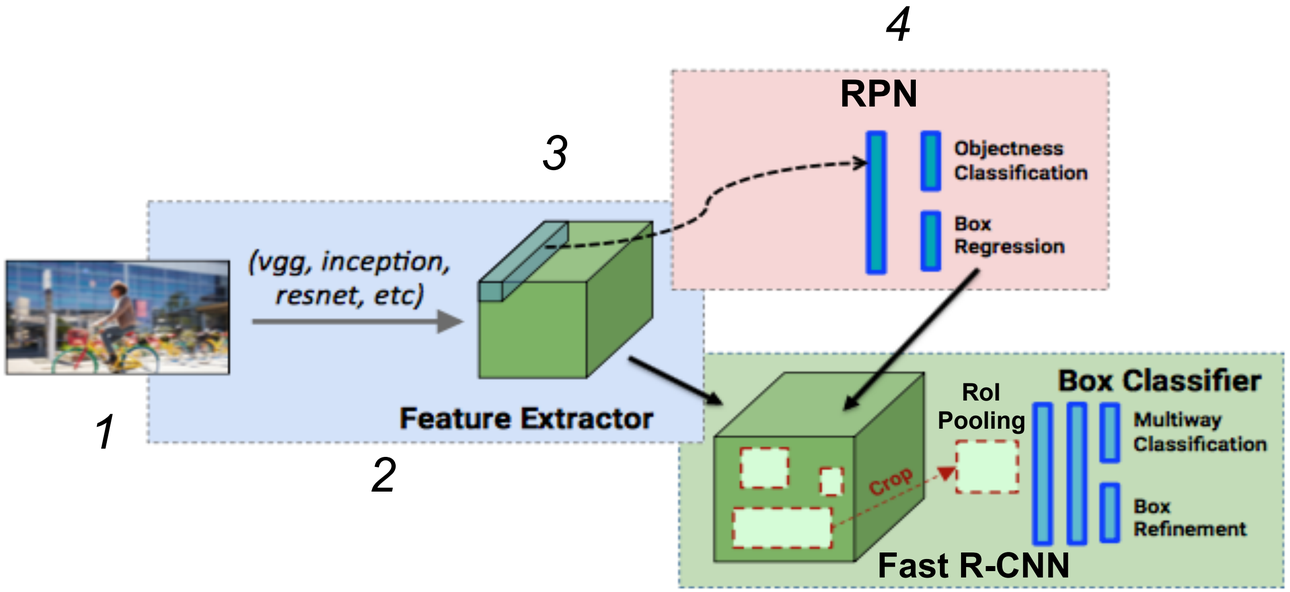
1. Input image of any size
2. Extract features using VGG-16
3. Each *n* x *n* spatial window inputted to RPN

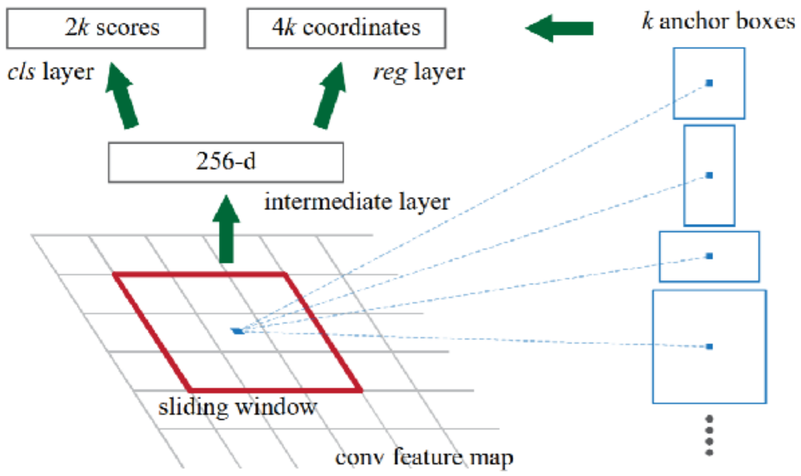




Faster R-CNN: Pipeline

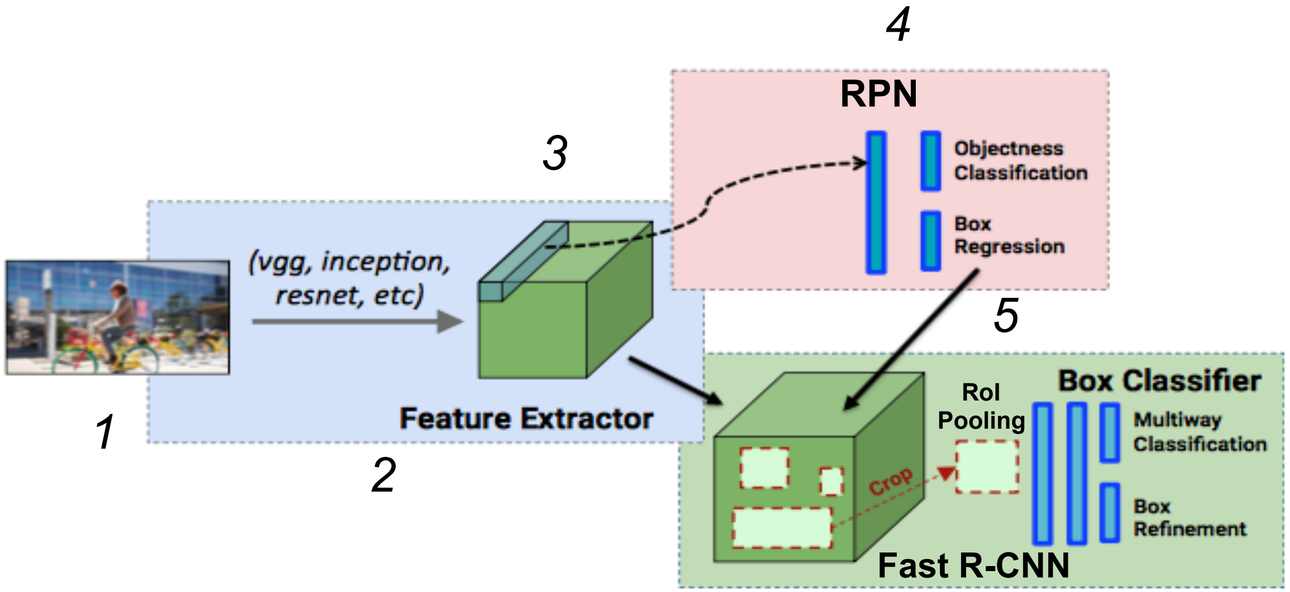
1. Input image of any size
2. Extract features using VGG-16
3. Each *n* x *n* spatial window inputted to RPN
4. RPN predict objectness score and box coordinates for *k* anchor boxes





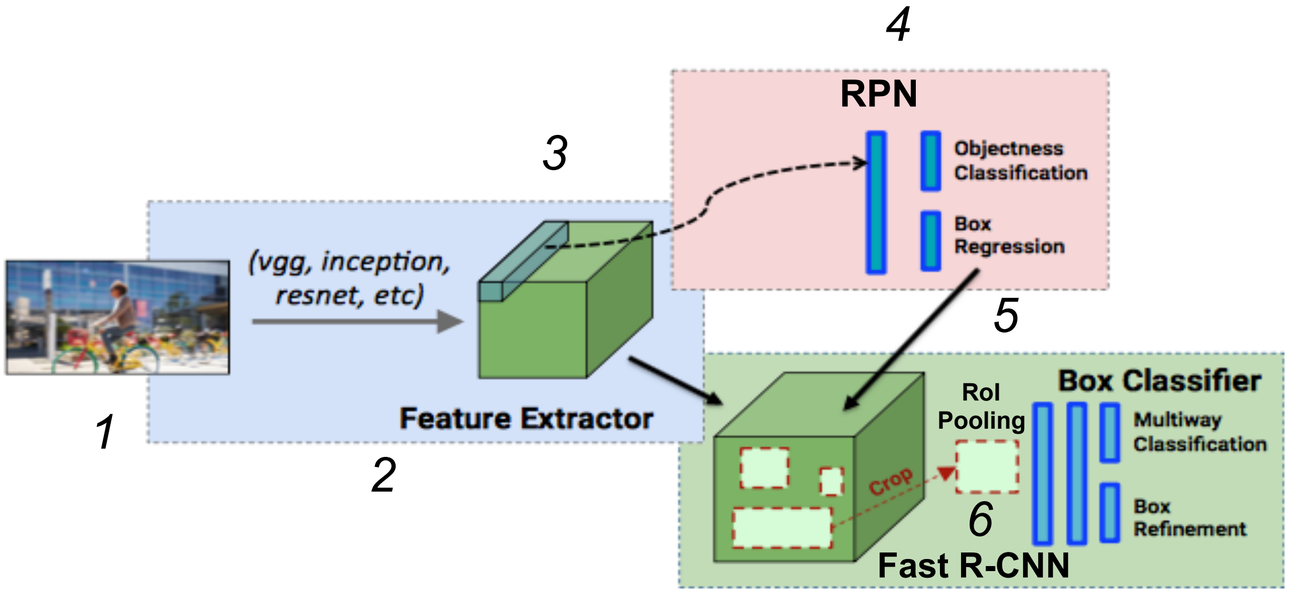
Faster R-CNN: Pipeline

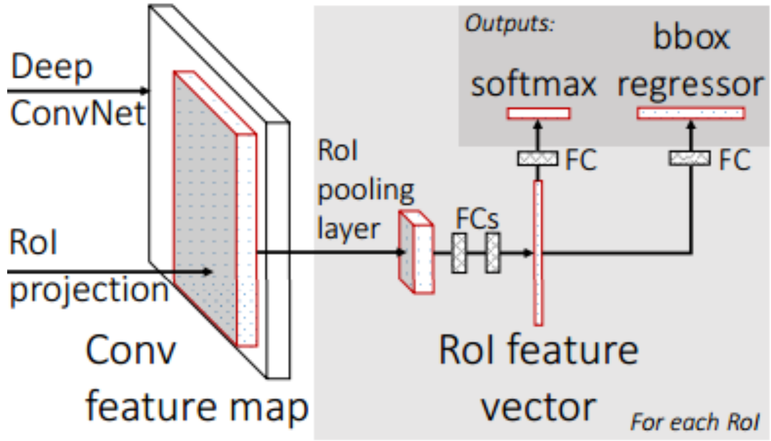
1. Input image of any size
2. Extract features using VGG-16
3. Each *n* x *n* spatial window inputted to RPN
4. RPN predict objectness score and box coordinates for *k* anchor boxes
5. NMS to reduce number of proposals



Faster R-CNN: Pipeline

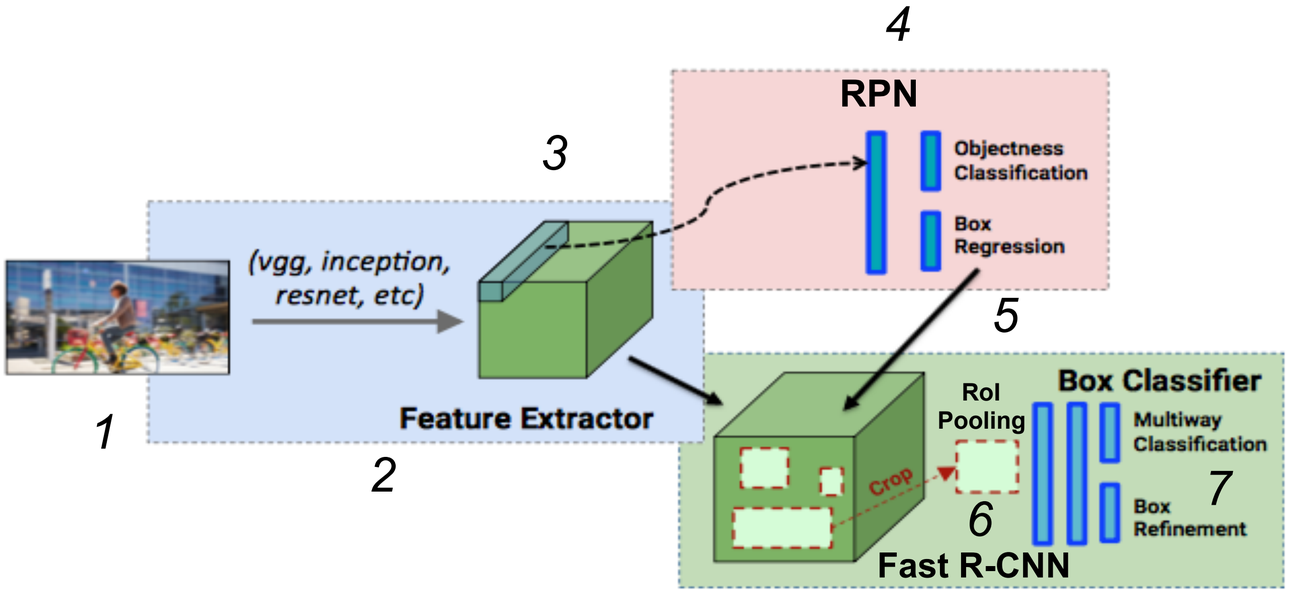
1. Input image of any size
2. Extract features using VGG-16
3. Each *n* x *n* spatial window inputted to RPN
4. RPN predict objectness score and box coordinates for *k* anchor boxes
5. NMS to reduce number of proposals
6. For each remaining region proposal perform RoI pooling

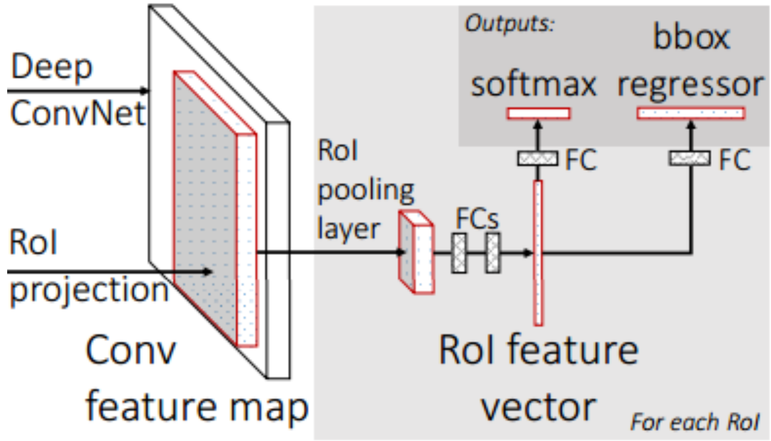




Faster R-CNN: Pipeline

1. Input image of any size
2. Extract features using VGG-16
3. Each *n* x *n* spatial window inputted to RPN
4. RPN predict objectness score and box coordinates for *k* anchor boxes
5. NMS to reduce number of proposals
6. For each remaining region proposal perform RoI pooling
7. Final predictions (softmax probability over classes and bounding box coordinates for each class)





Faster R-CNN: Training

- **4-step alternating training:**
  - 1. Initialize with pre-trained model and fine-tune **RPN** for the region proposal task
  - 2. Train a separate **Fast R-CNN** detection network (also pre-trained) using proposals generated in step 1
  - 3. RPN training with the shared convolutional layers of the detection network fixed
  - 4. Fine-tune detection network by keeping shared convolutional layers fixed

- **Losses specific to RPN and Fast R-CNN:**
  - Multi-task loss of classification and regression
    - Classification loss (log loss):

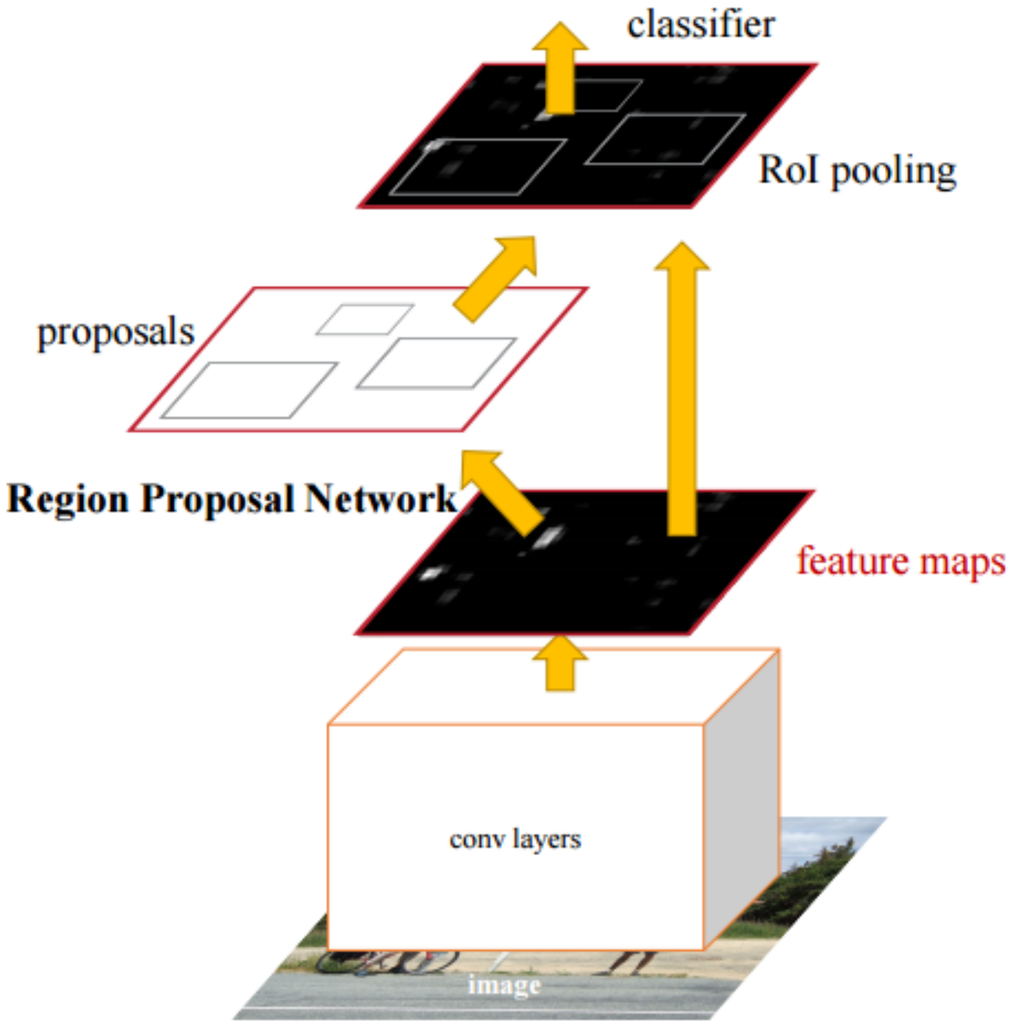
$$L_{\text{cls}}(p, u) = -\log p_u$$

    - Regression loss (smooth L1):

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i), \qquad \text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

Faster R-CNN: Region Proposal Network (RPN)

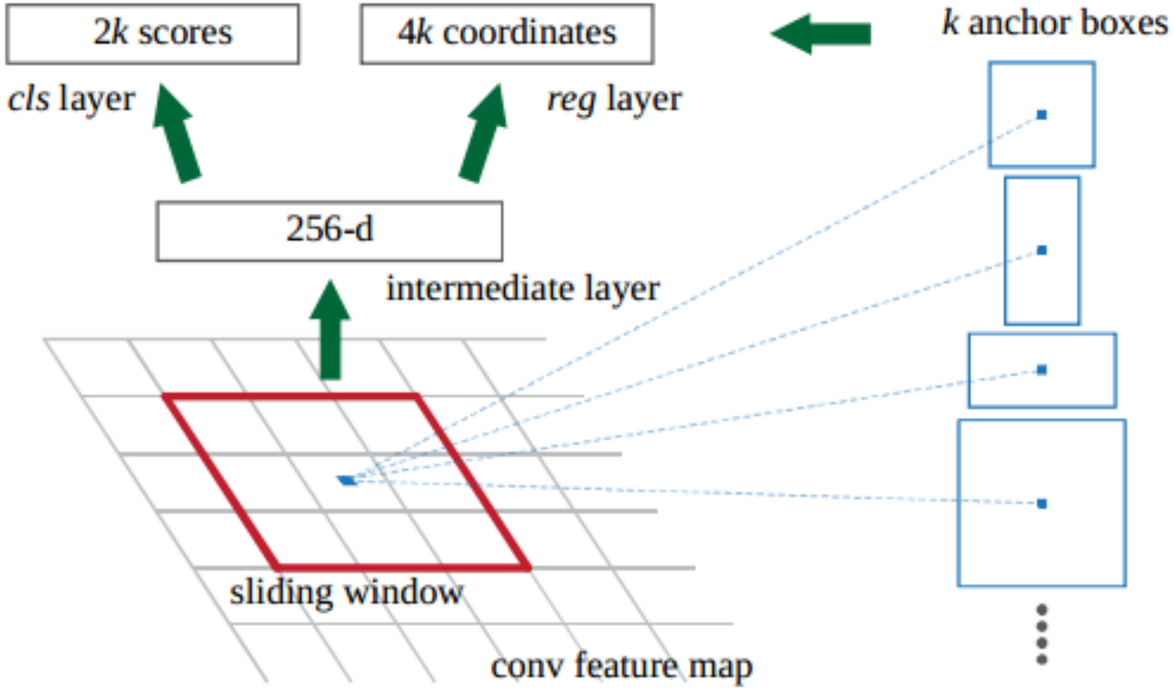
1. RoI generation of R-CNN is slow
   a. Has to be done separately
   b. Does not take advantage from feature extraction part
2. → **Region proposal network!**
   a. Use extracted features from CNN
   b. Trained «jointly» with classifier and BBox regressor
3. RoI Pooling



Faster R-CNN: **RPN** (2)

- Sliding window over output feature map
- For each point, extract *k* anchor boxes
- Run the extracted part through CNN to extract feature map
- Run feature map through two fully connected layers to predict objectless / certainties and region proposals.

Input: Conv. feature map
Output: Region proposal + certainty scores

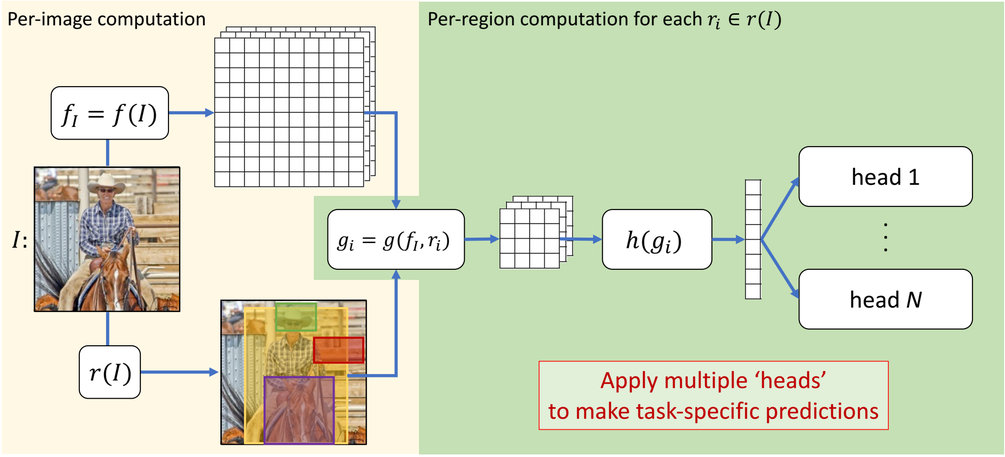


Alternative look at Region-based CNNs:
Per Image vs. per Region Computations

The Generalized R-CNN Framework for Object Detection

ICCV 2017 Tutorial - Ross Girshick, FAIR

Generalized R-CNN Framework



R-CNN (Region-based Convolutional Neural Net)

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

Per-image computation

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

I:

Selective search,
Edge Boxes,
MCG, …

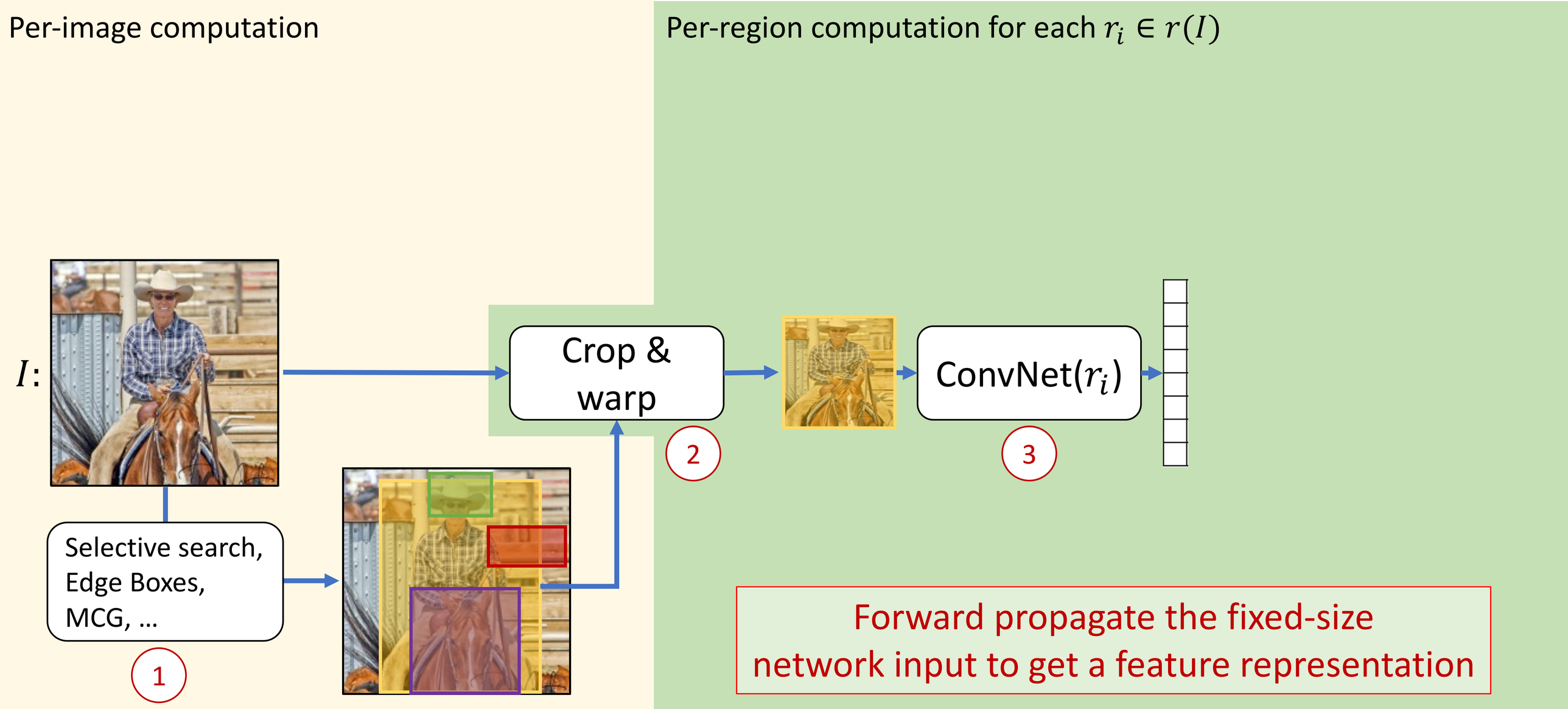
1

Crop &
warp

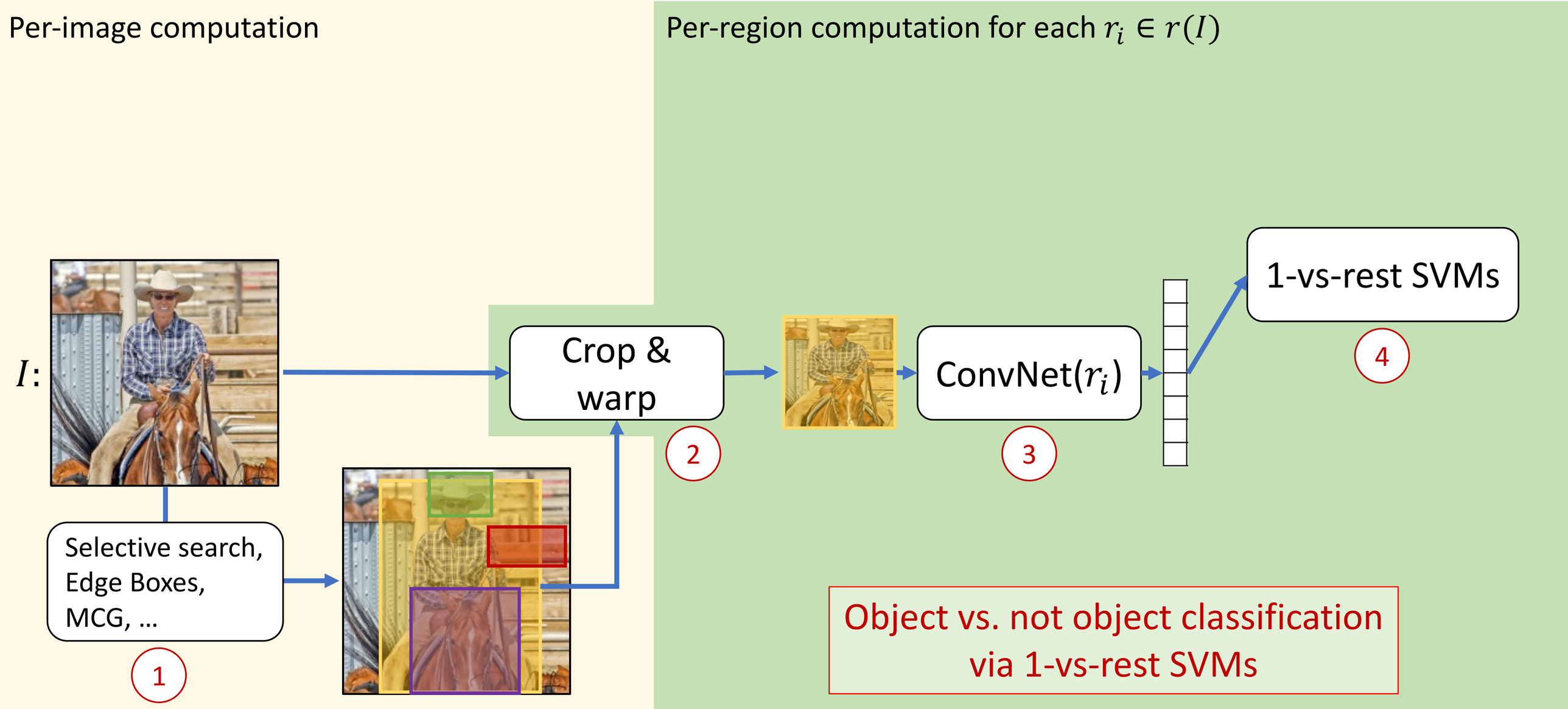
2

Crop and warp each proposal image window
to obtain a fixed-size network input

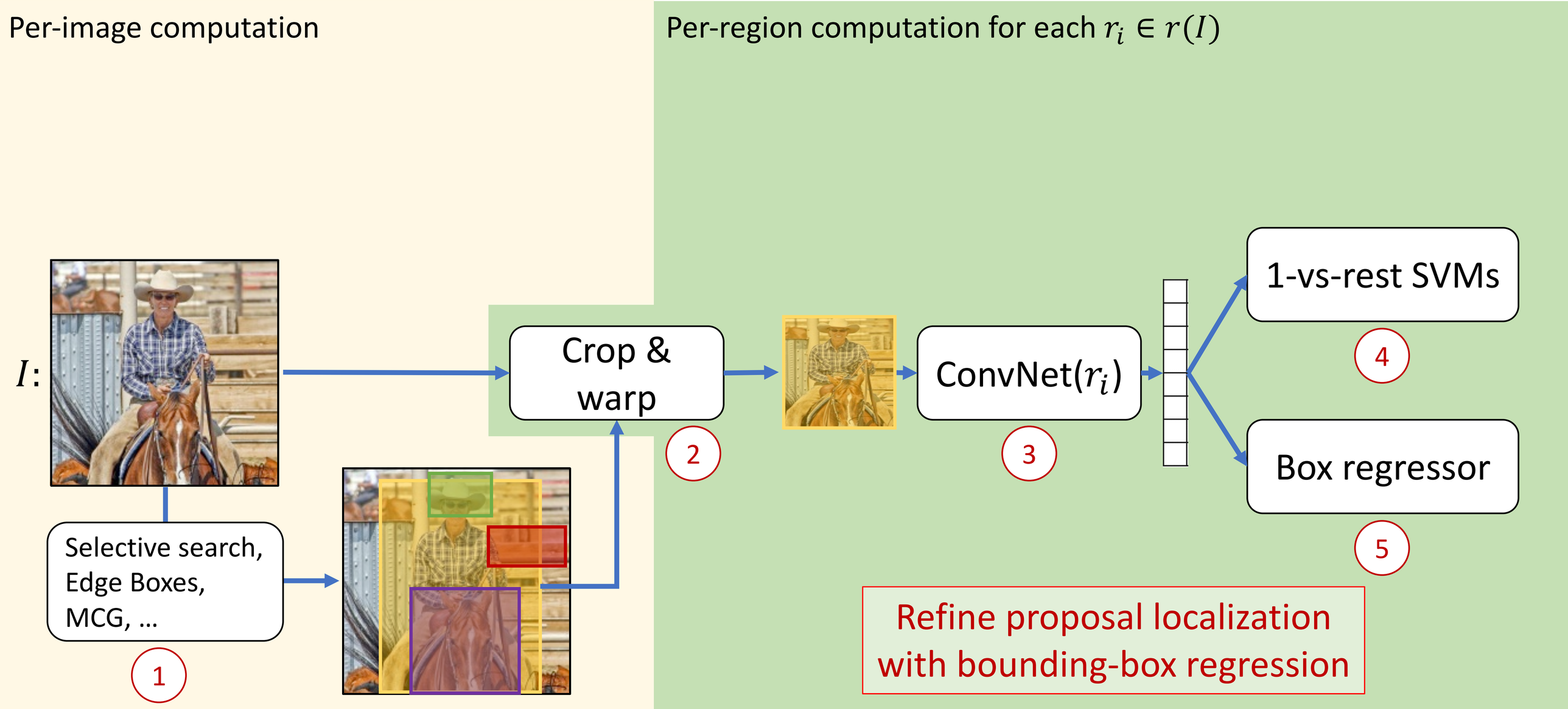
R-CNN


Per-image computation
Per-region computation for each $r_i \in r(I)$
I:
Selective search, Edge Boxes, MCG, …
1
Crop & warp
2
ConvNet(r_i)
3
Forward propagate the fixed-size network input to get a feature representation

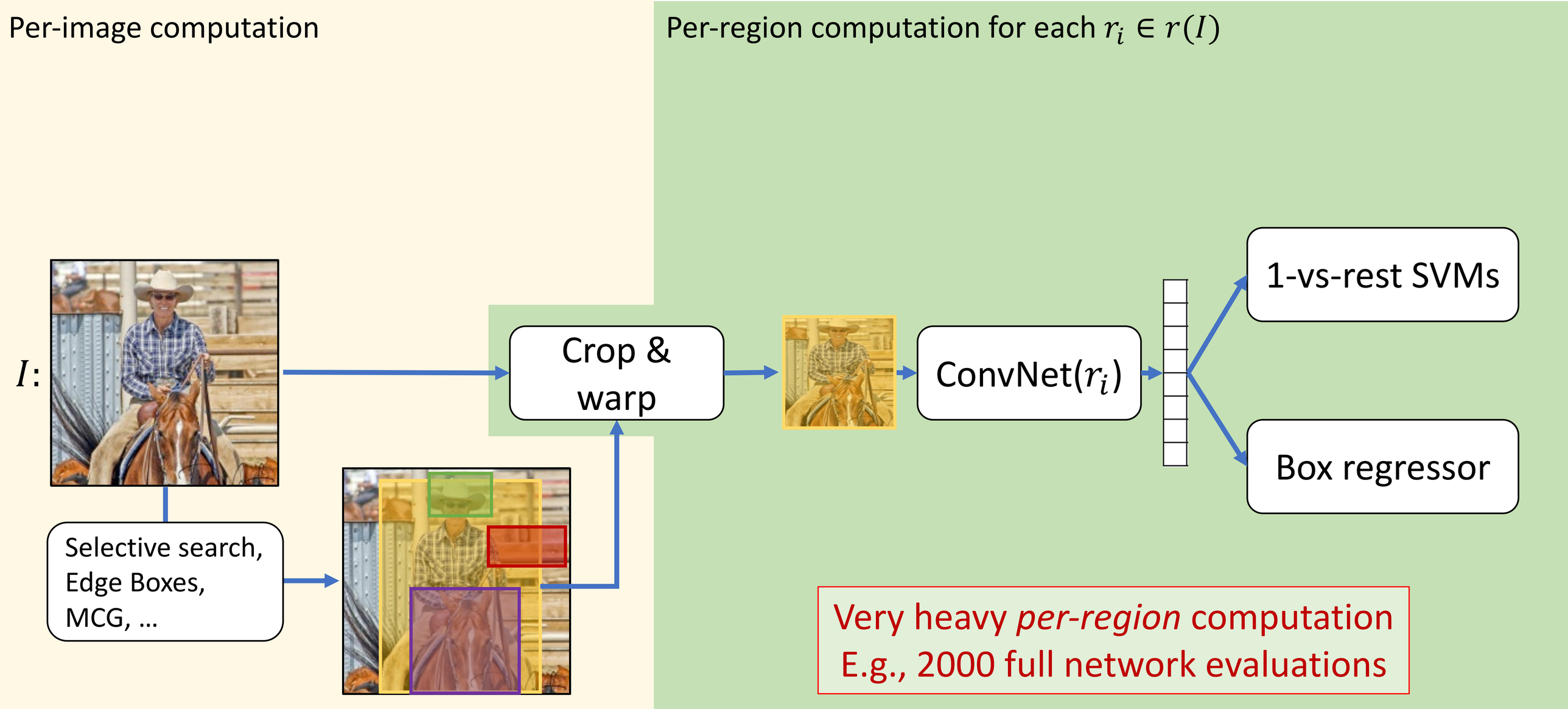
R-CNN


Per-image computation
Per-region computation for each $r_i \in r(I)$
I:
Selective search, Edge Boxes, MCG, …
1
Crop & warp
2
ConvNet(r_i)
3
1-vs-rest SVMs
4
Object vs. not object classification via 1-vs-rest SVMs

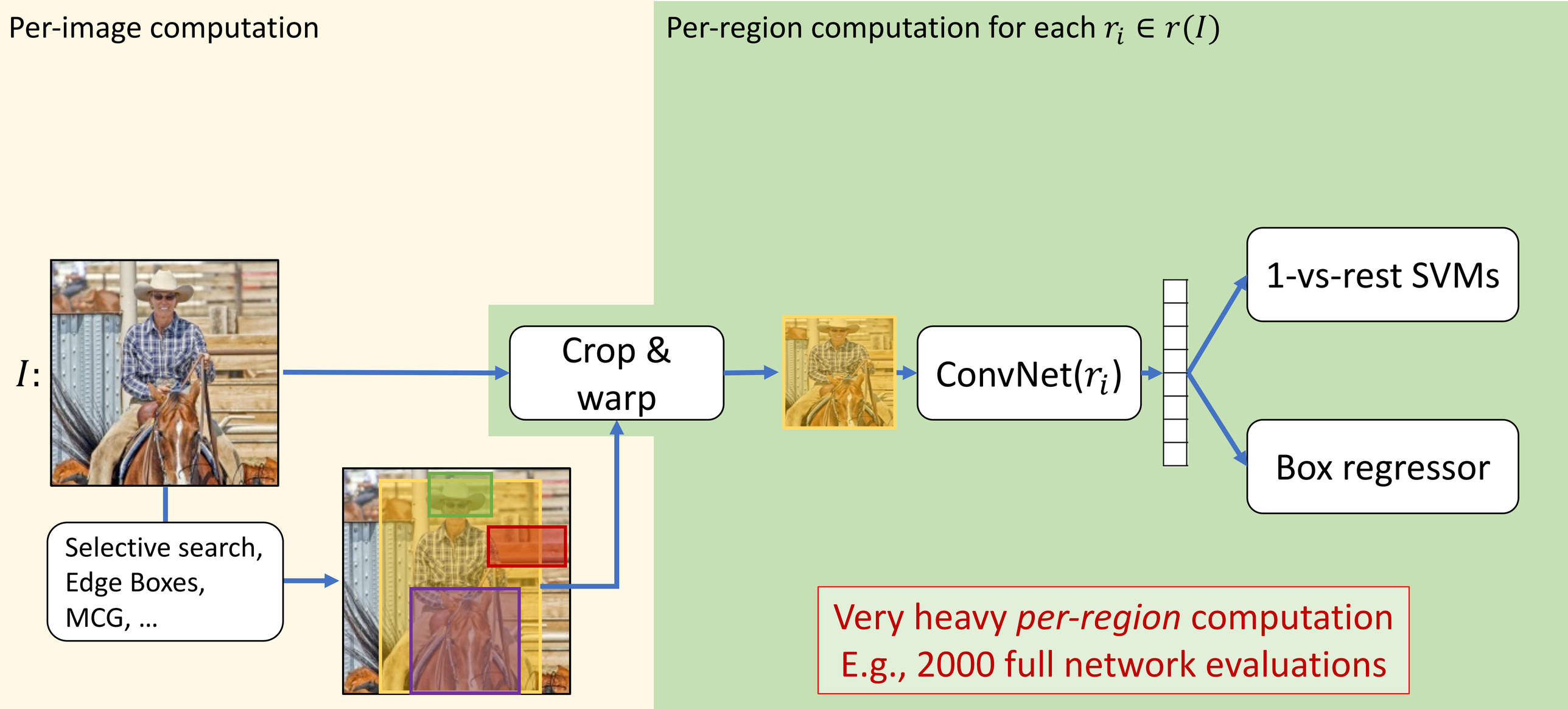
R-CNN


Per-image computation
Per-region computation for each $r_i \in r(I)$
I:
Selective search, Edge Boxes, MCG, …
1
Crop & warp
2
ConvNet(r_i)
3
1-vs-rest SVMs
4
Box regressor
5
Refine proposal localization with bounding-box regression

The Problem with R-CNN



"Slow" R-CNN


Per-image computation
Per-region computation for each $r_i \in r(I)$
I:
Selective search, Edge Boxes, MCG, …
Crop & warp
ConvNet(r_i)
1-vs-rest SVMs
Box regressor
Very heavy *per-region* computation
E.g., 2000 full network evaluations

R-CNN

References

- B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. TPAMI, 2012.
- I. Endres and D. Hoiem. Category independent object proposals. In ECCV, 2010.
- J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013.
- A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In NIPS, 2012.
- X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In ICCV, 2013.
- R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In CVPR, 2014.

Generalized R-CNN Framework

Per-image computation

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

I:

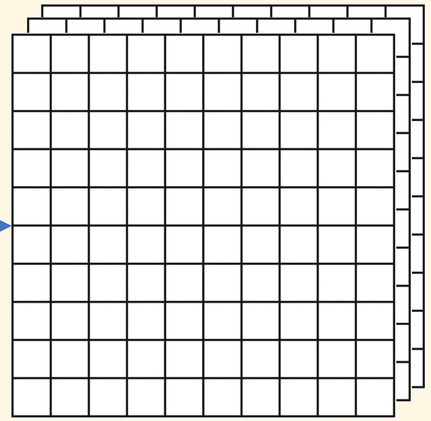
Input image

per-image operations | per-region operations

Generalized R-CNN Framework

Per-image computation

$f_I = f(I)$



I:

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

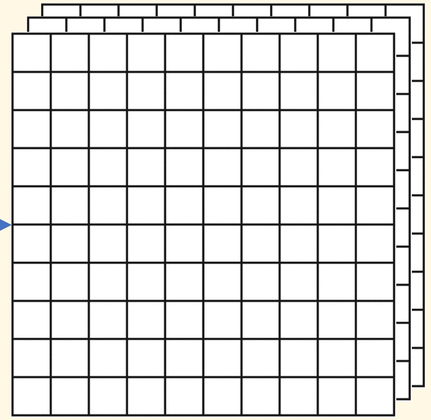
Transformation of the input image into a featurized representation

Generalized R-CNN Framework

Per-image computation

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

$f_I = f(I)$

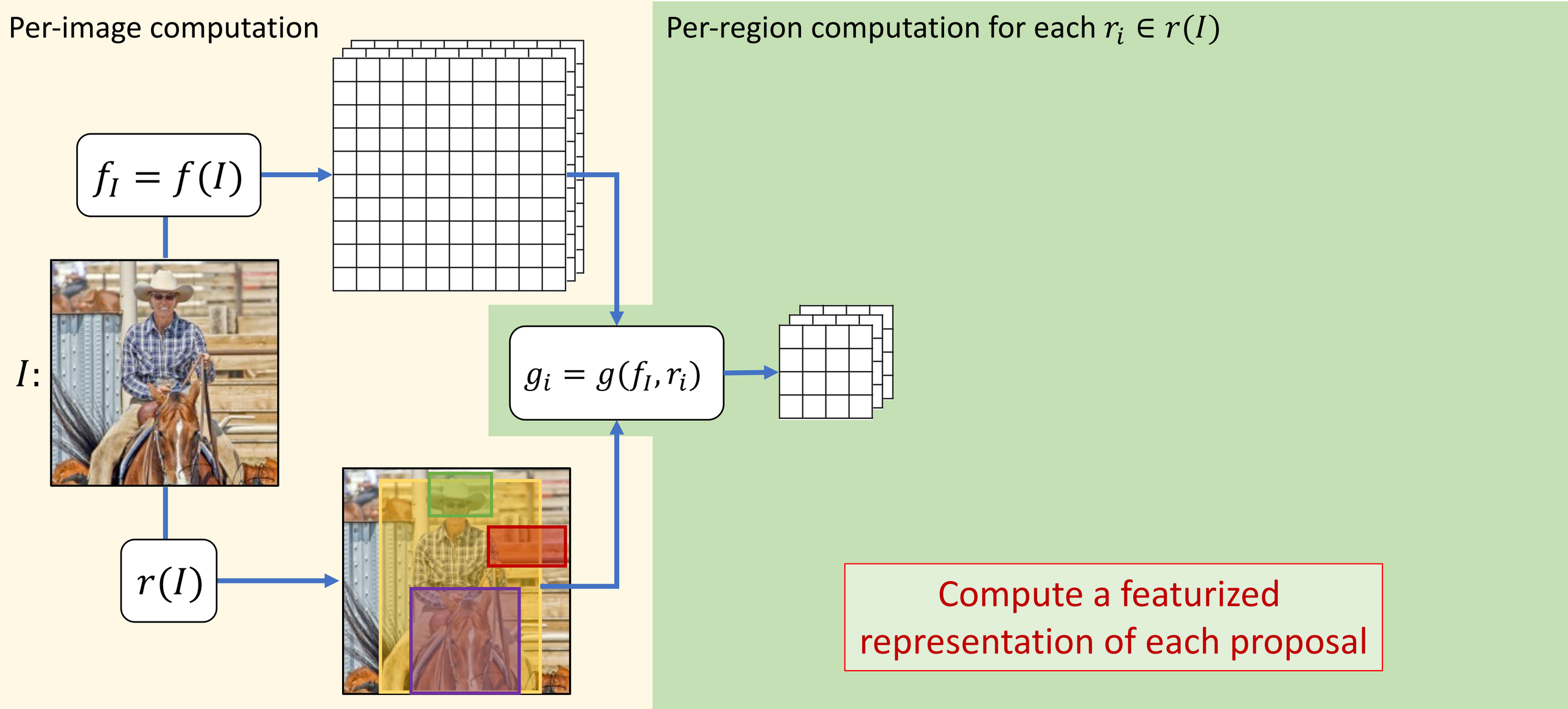


I:

$r(I)$

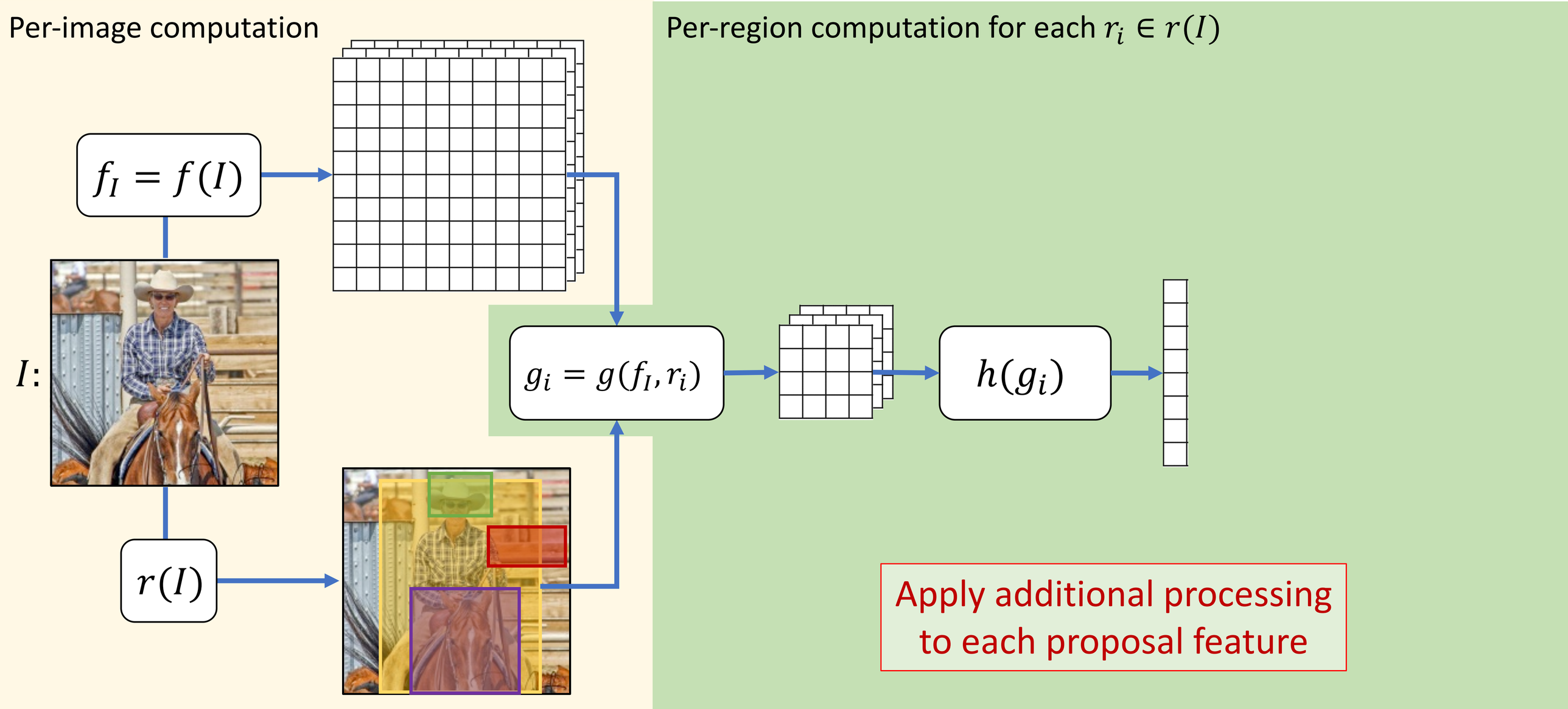
Object/region/detection proposals computed for the image

Generalized R-CNN Framework

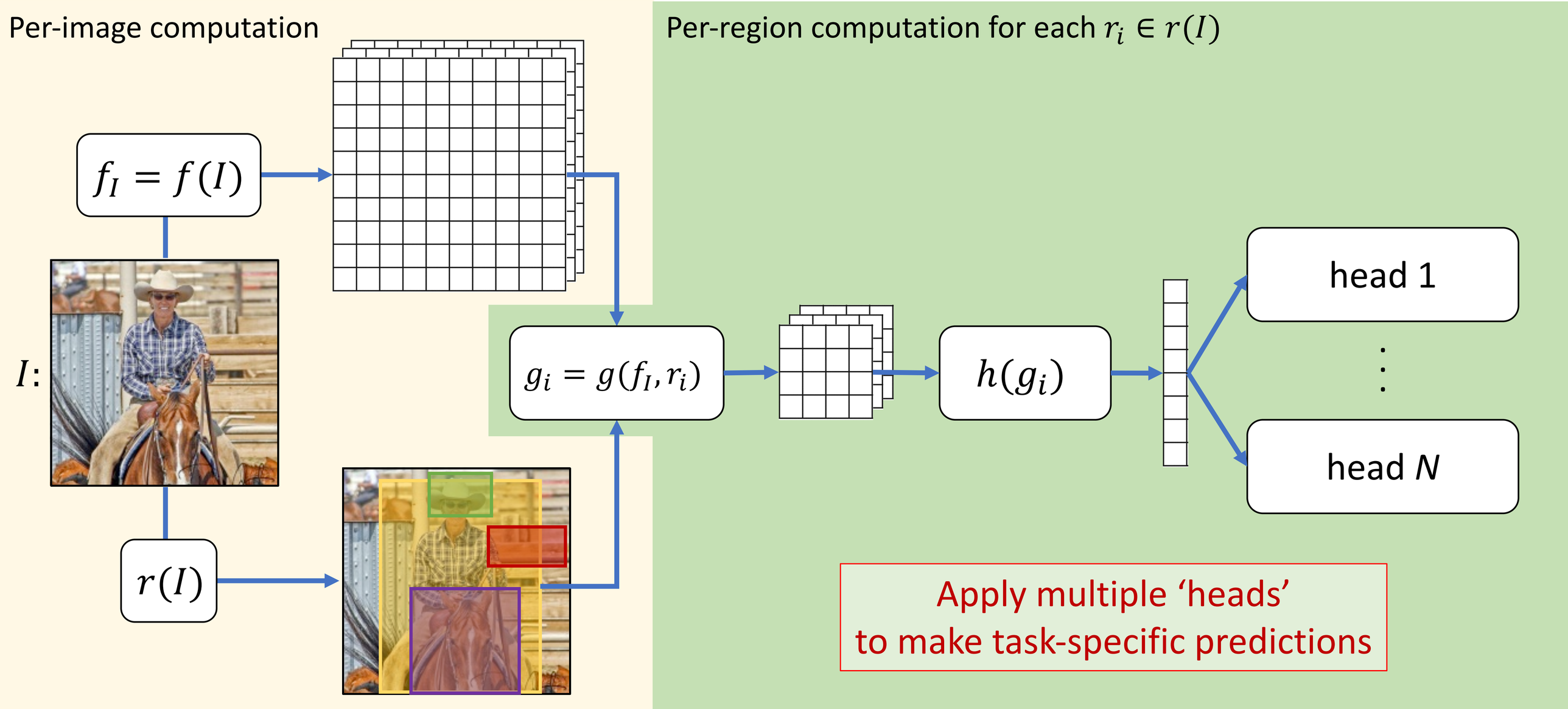


Compute a featurized representation of each proposal

Generalized R-CNN Framework



Generalized R-CNN Framework



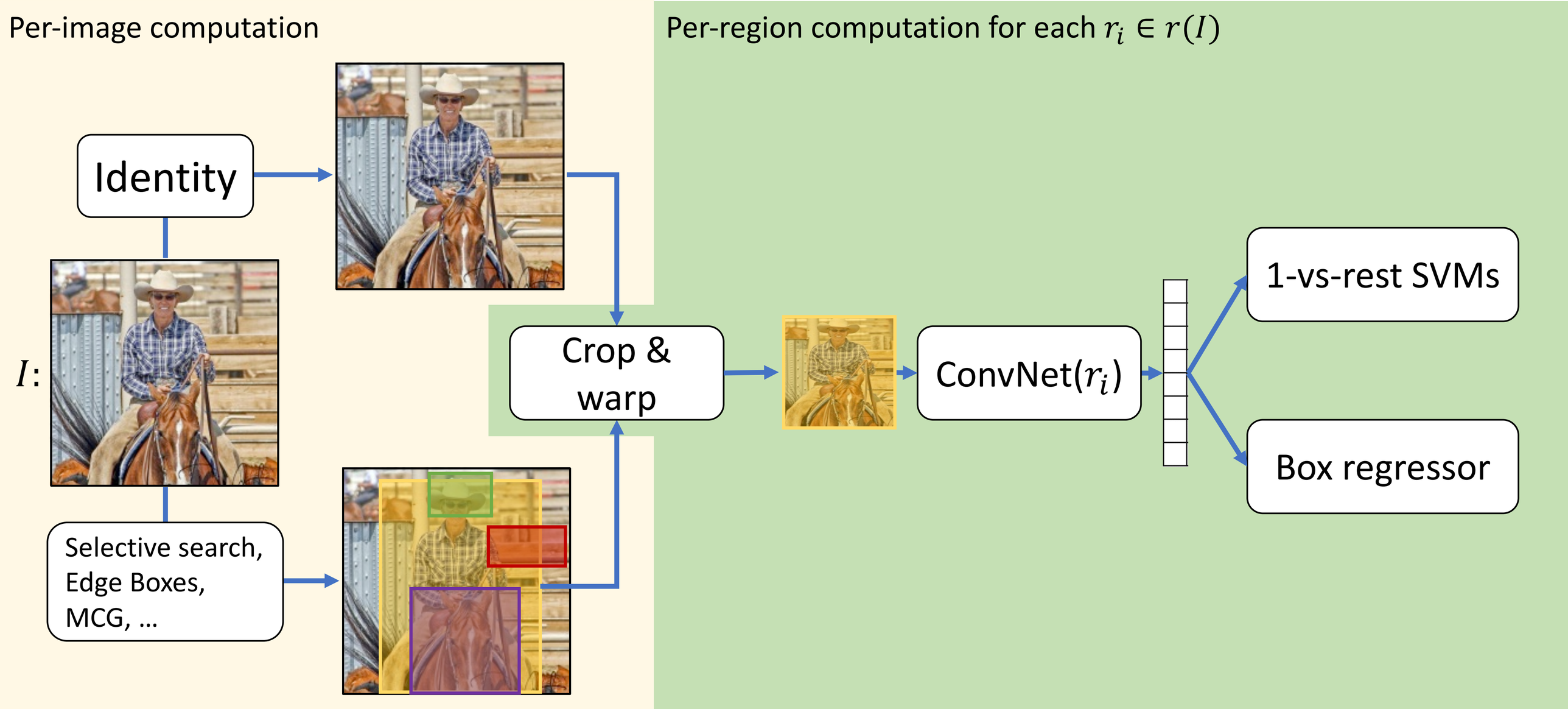
Generalized R-CNN Framework

A common framework for understanding

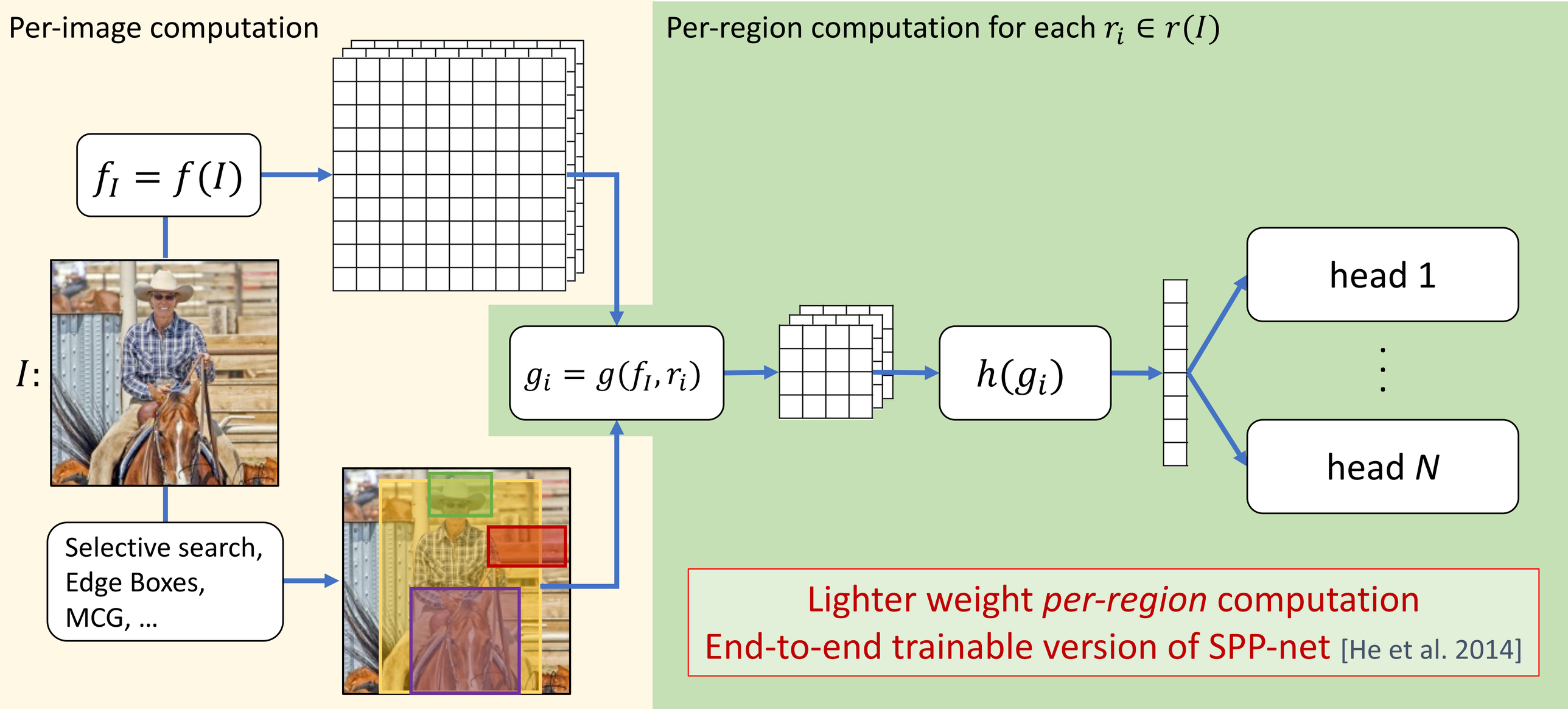
- (slow) R-CNN
- Fast R-CNN
- Faster R-CNN
- Feature Pyramid Networks (FPN) + Faster R-CNN

- Mask R-CNN (next tutorial)

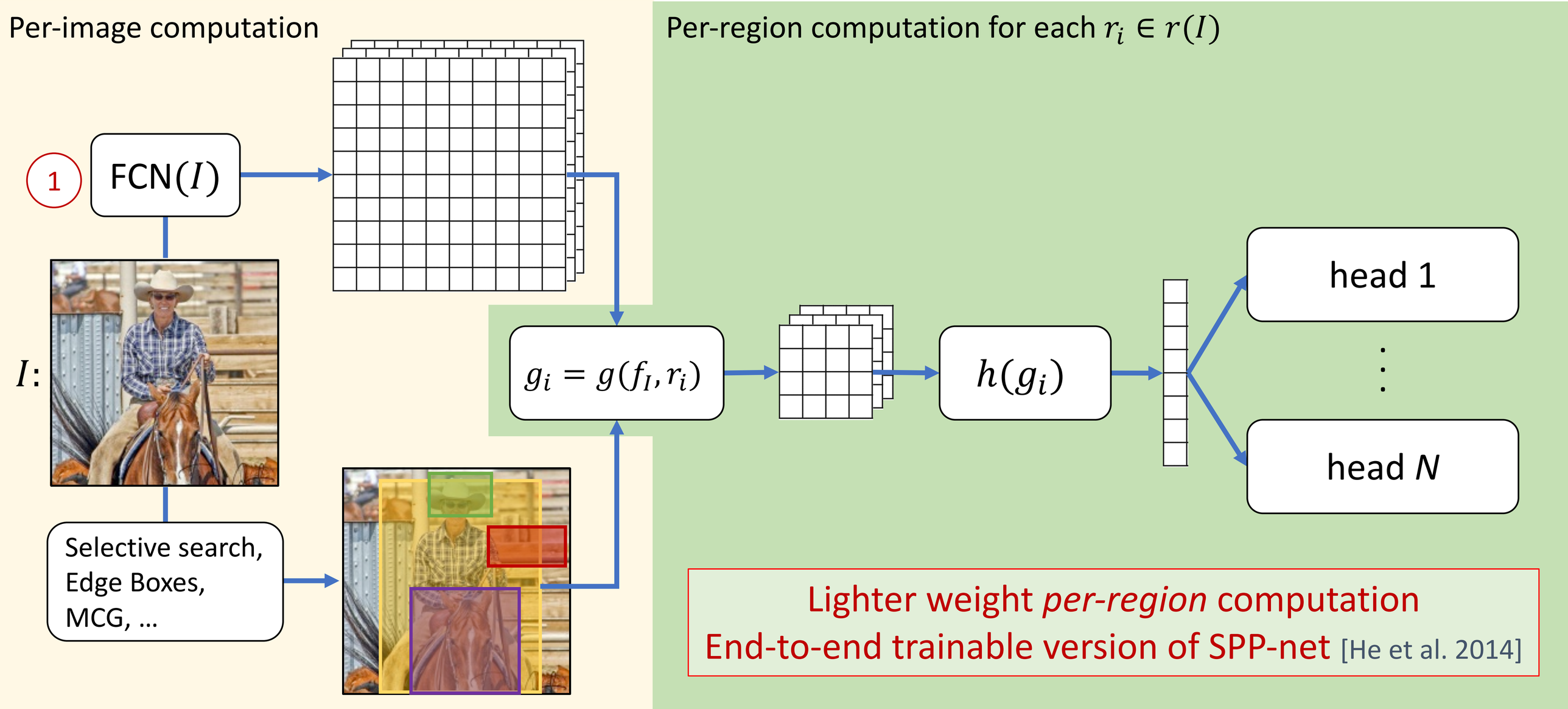
Slow R-CNN in the Generalized Framework



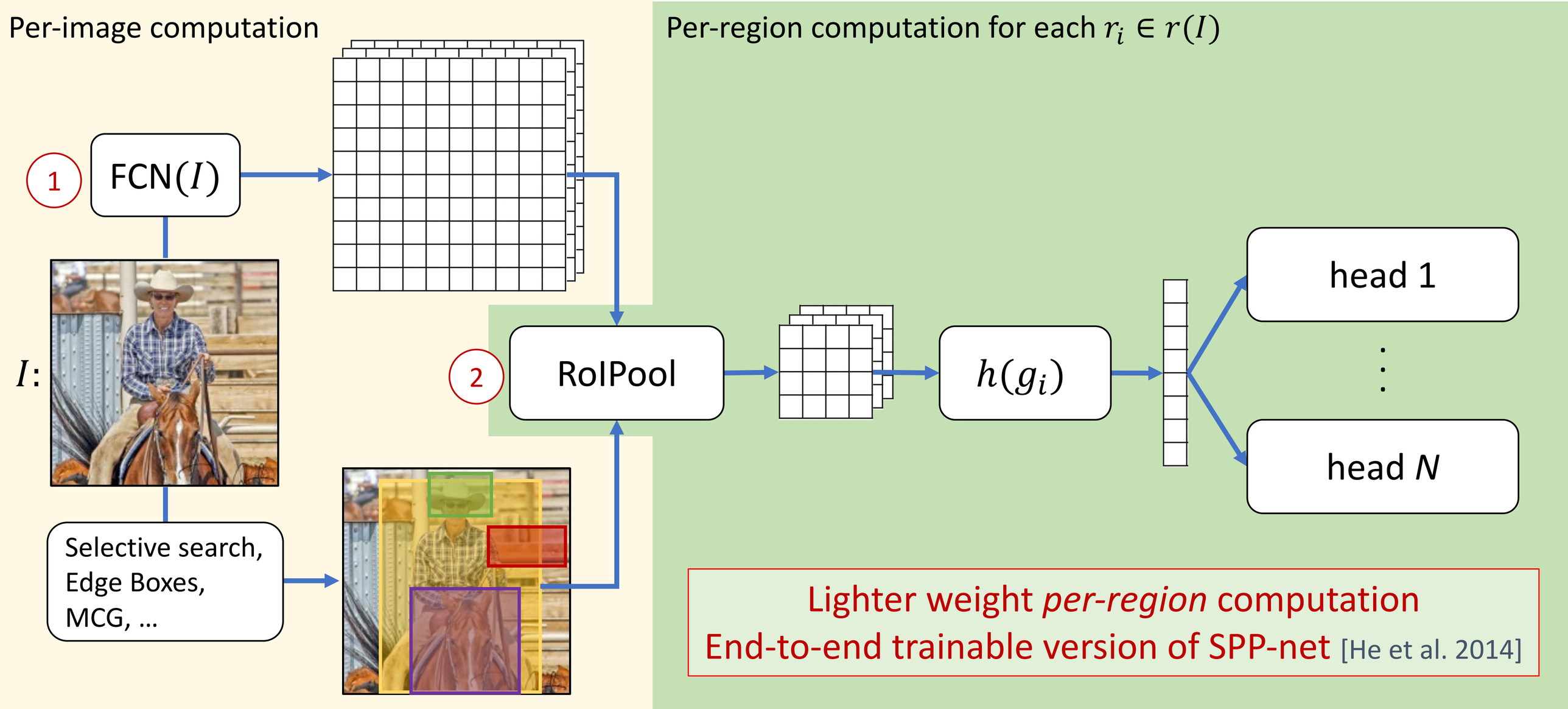
Generalized R-CNN → Fast R-CNN



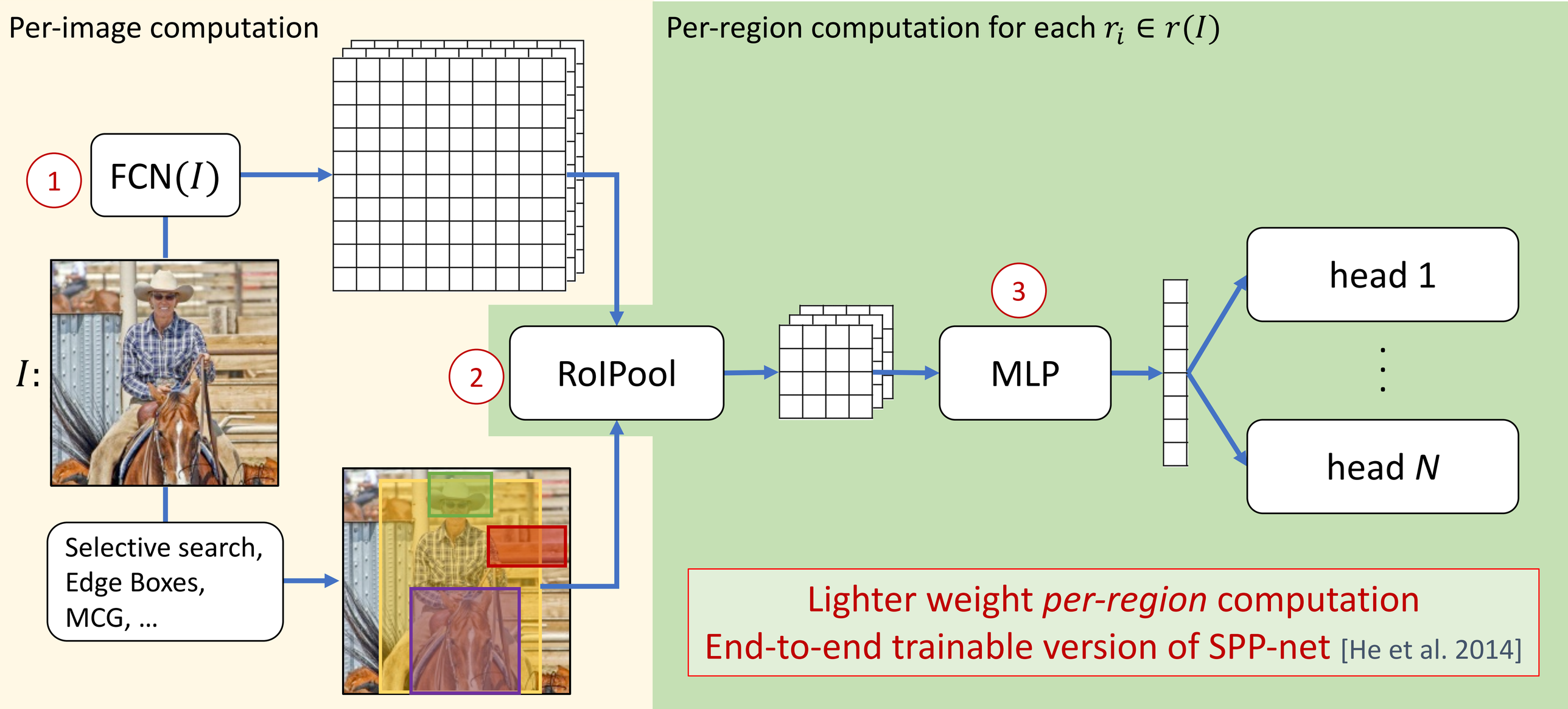
Generalized R-CNN → Fast R-CNN



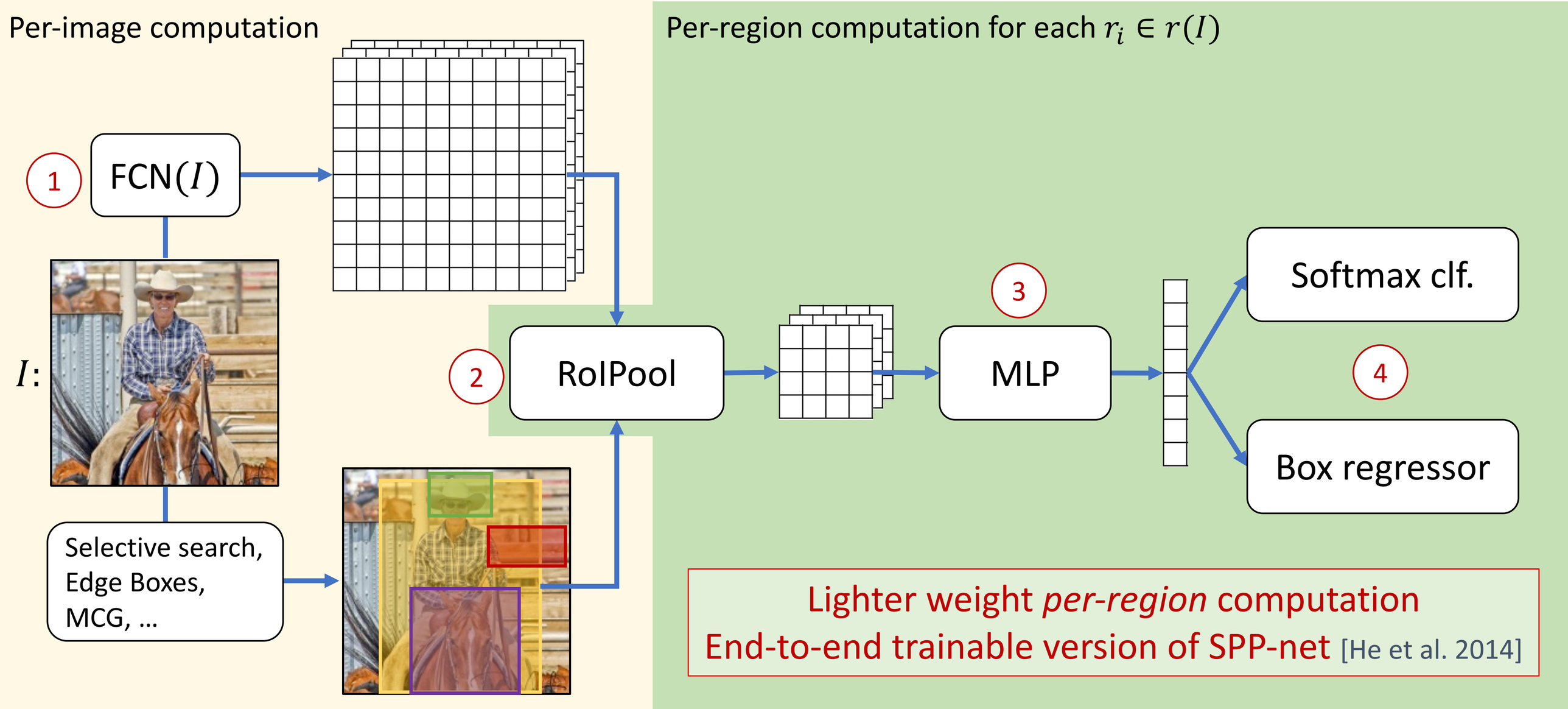
Generalized R-CNN → Fast R-CNN



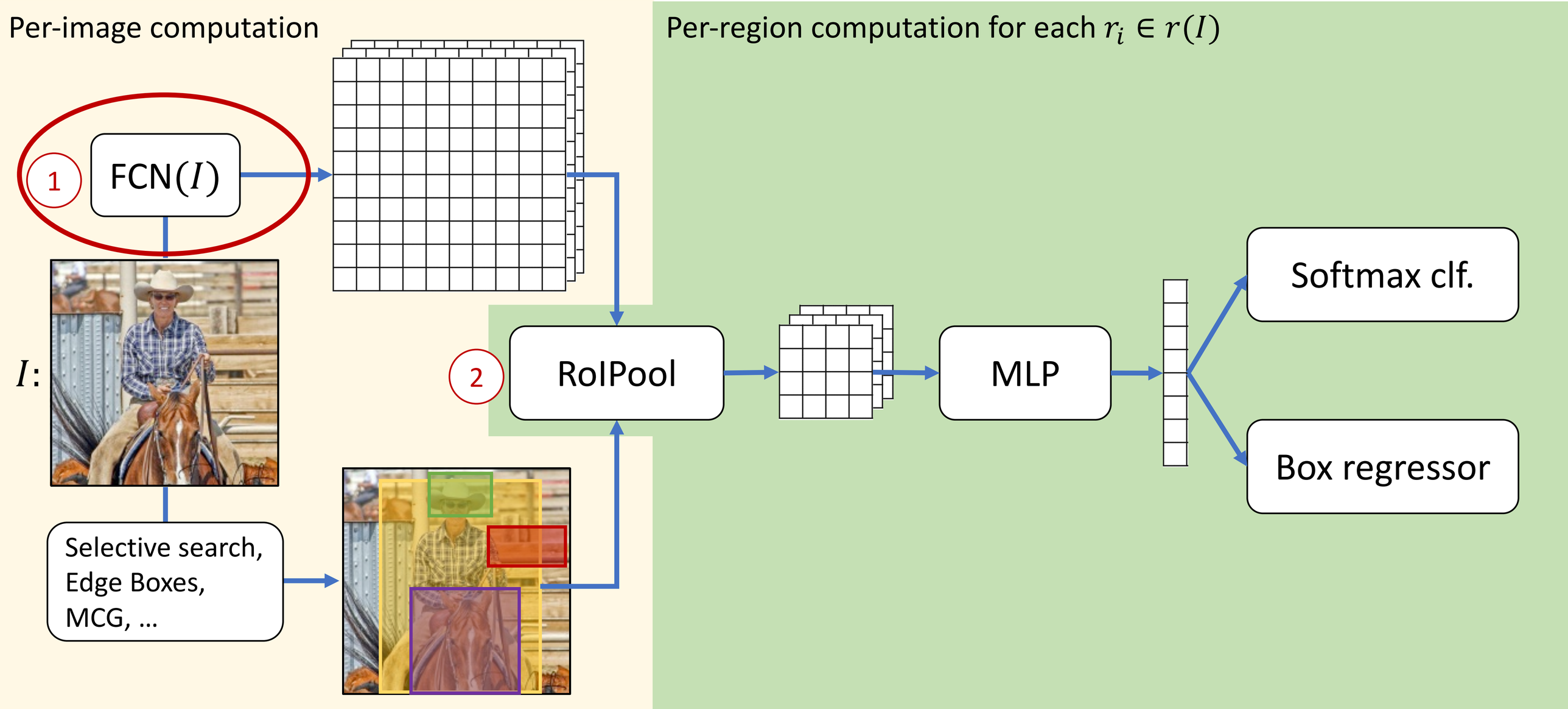
Generalized R-CNN → Fast R-CNN



Generalized R-CNN → Fast R-CNN

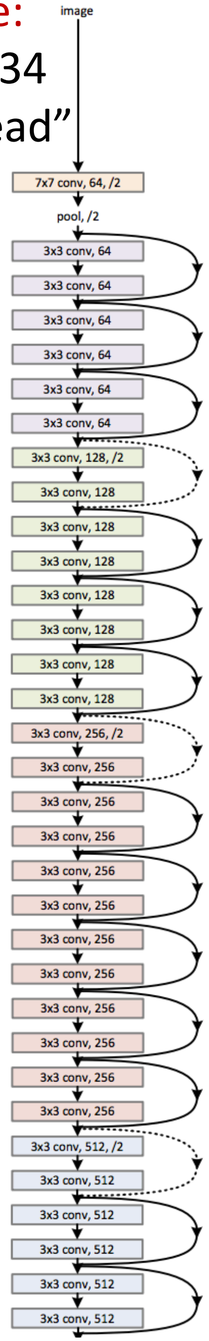


Fast R-CNN


Per-image computation
Per-region computation for each $r_i \in r(I)$
1
FCN(I)
I:
2
RoIPool
MLP
Softmax clf.
Box regressor
Selective search,
Edge Boxes,
MCG, …

Whole-image, Fully-conv. Network (FCN)

Example:
ResNet-34
w/o "head"

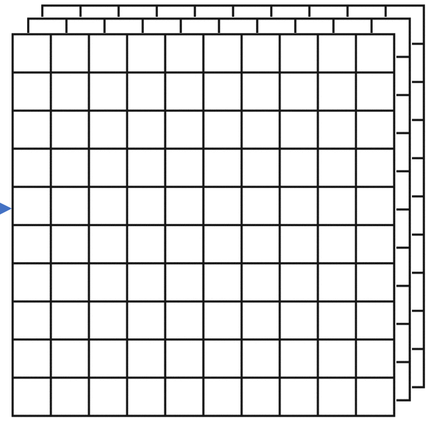


Use any standard ConvNet as the "backbone architecture"

- AlexNet, VGG, ResNet, Inception, Inception-ResNet, ResNeXt, DenseNet, …
- Remove global pooling / FC
- Output spatial dims are proportional to input spatial dims

I:

FCN(I)



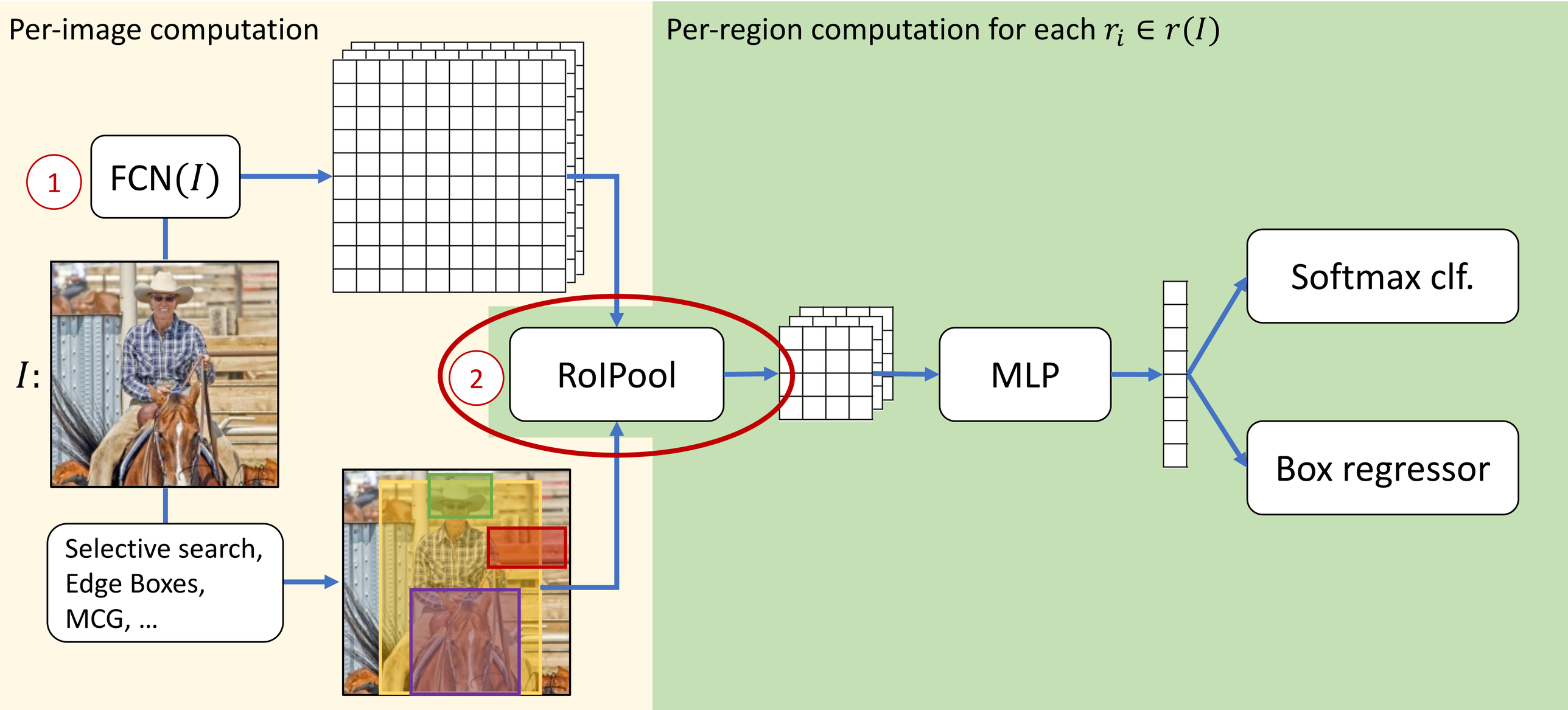
Example feature map dimensions:
(512, H/16, W/16)

The Engine of Recognition

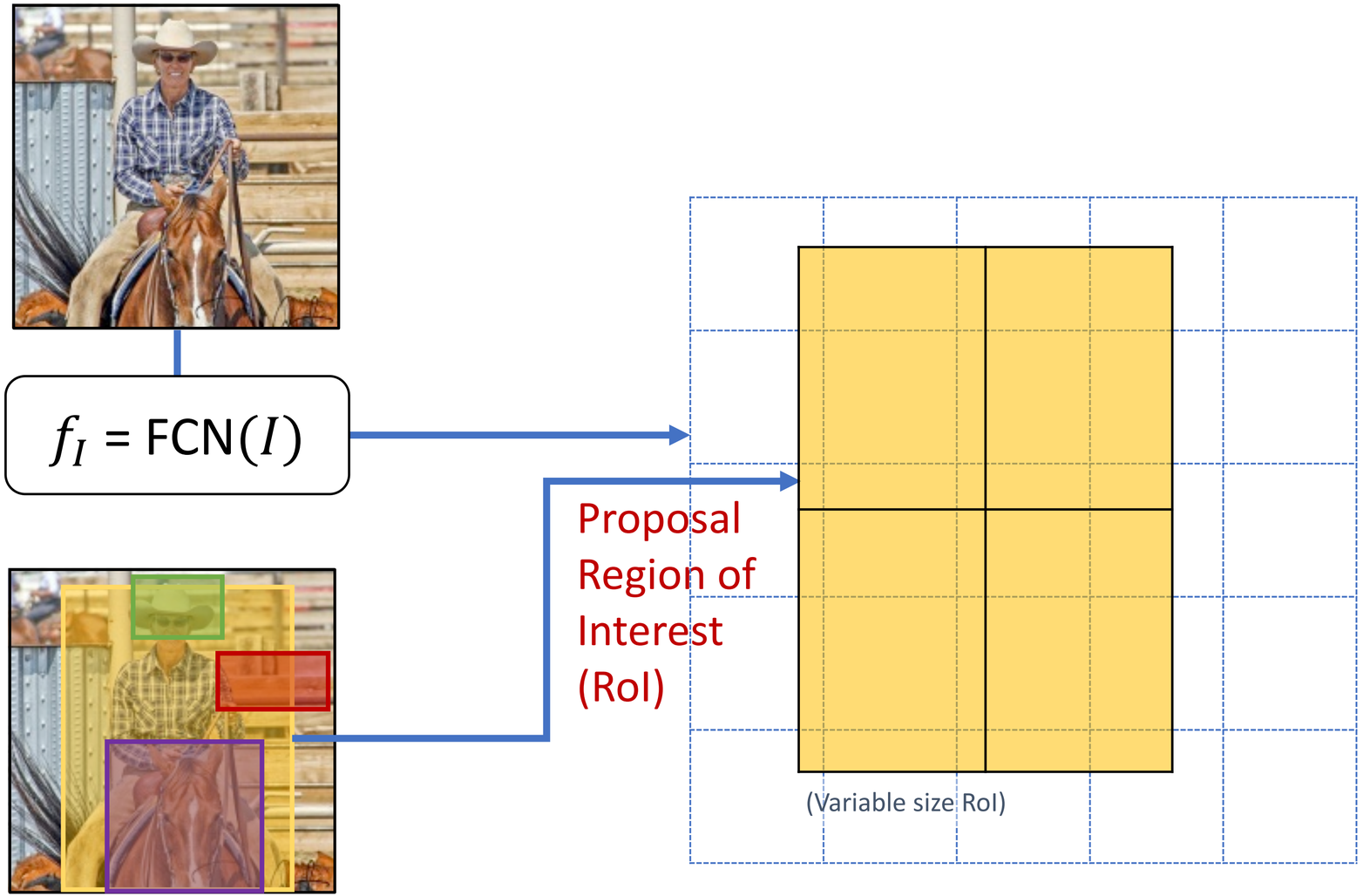
A good network lift all boats (“features matter”)

- AlexNet
- VGG
- GoogleNet / Inception
- ResNet
- ResNeXt
- …

Fast R-CNN

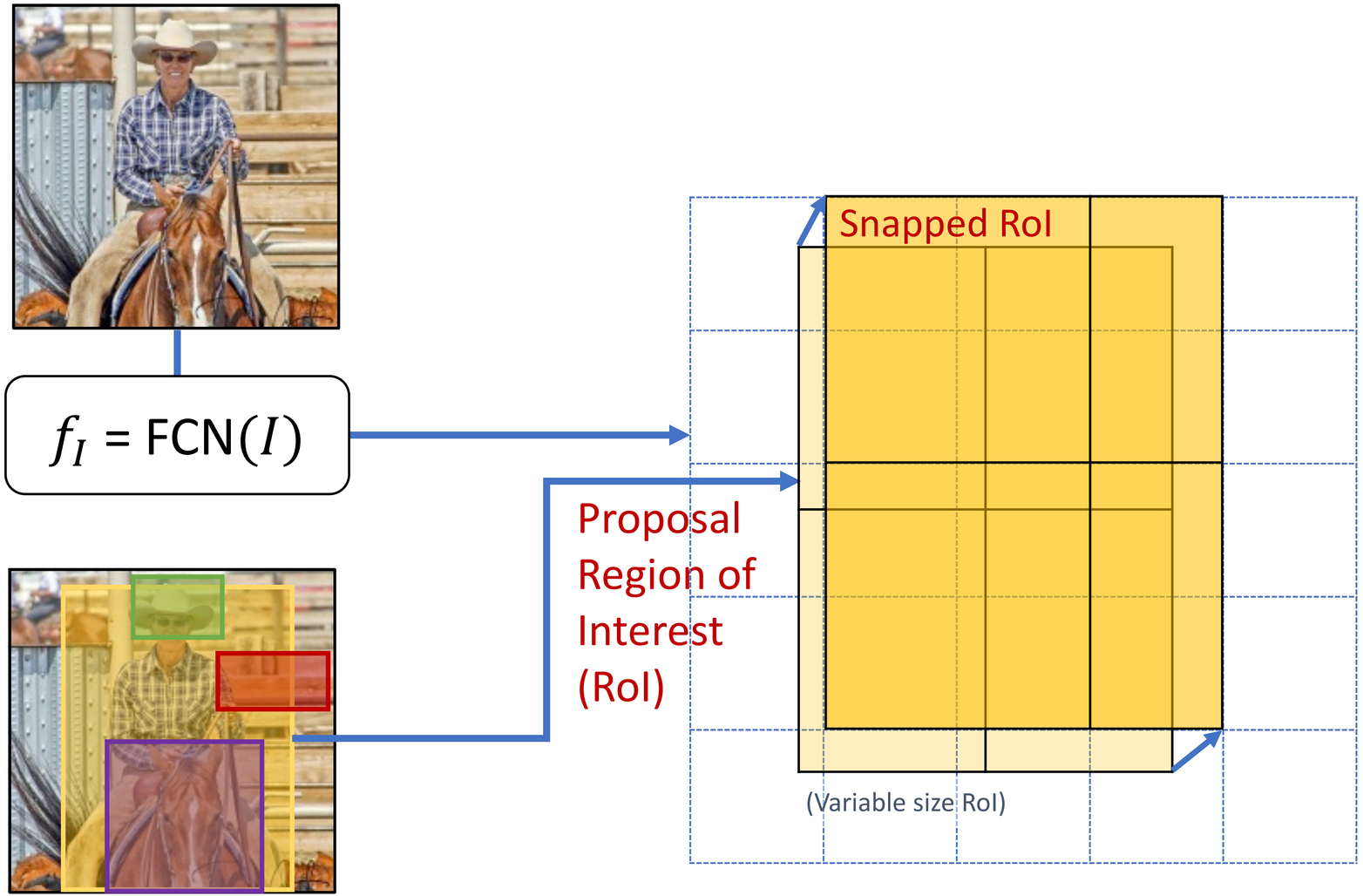

Per-image computation
Per-region computation for each $r_i \in r(I)$
1
FCN(I)
I:
2
RoIPool
MLP
Softmax clf.
Box regressor
Selective search,
Edge Boxes,
MCG, …

RoIPool (on each Proposal)



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

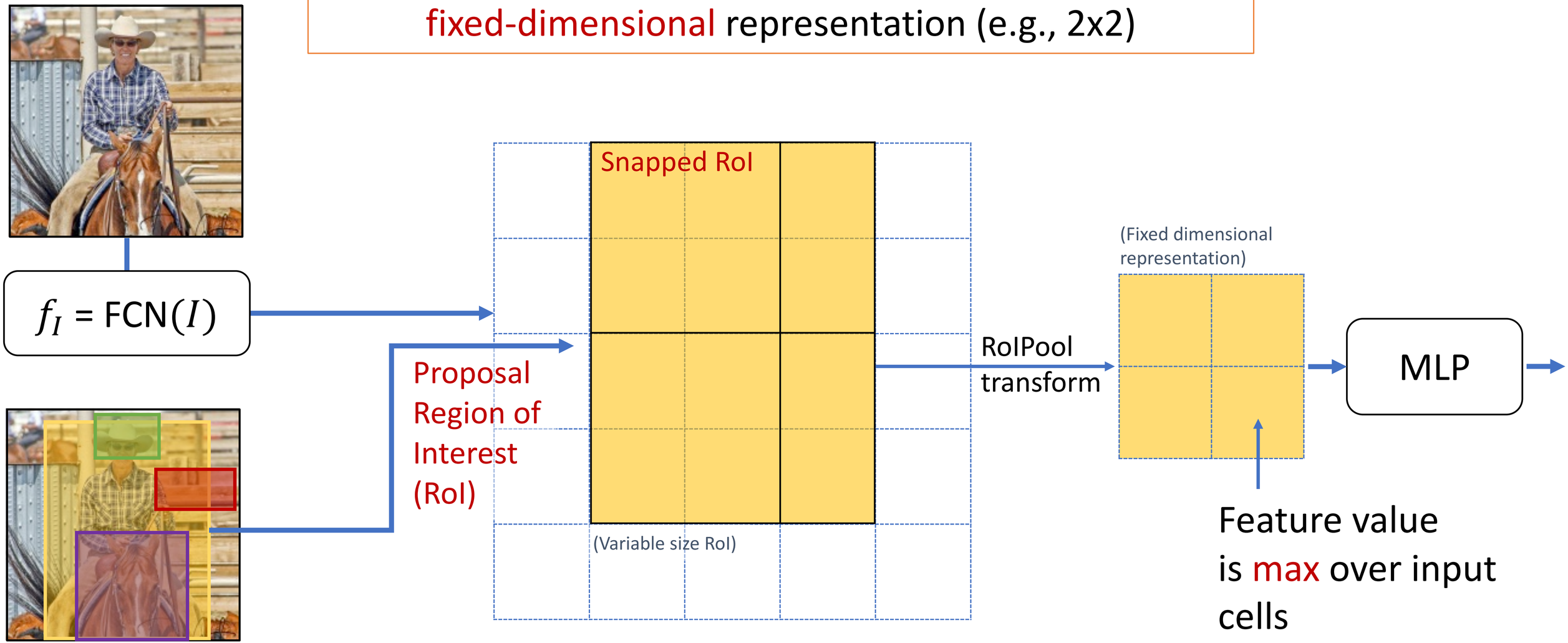
RoIPool (on each Proposal)



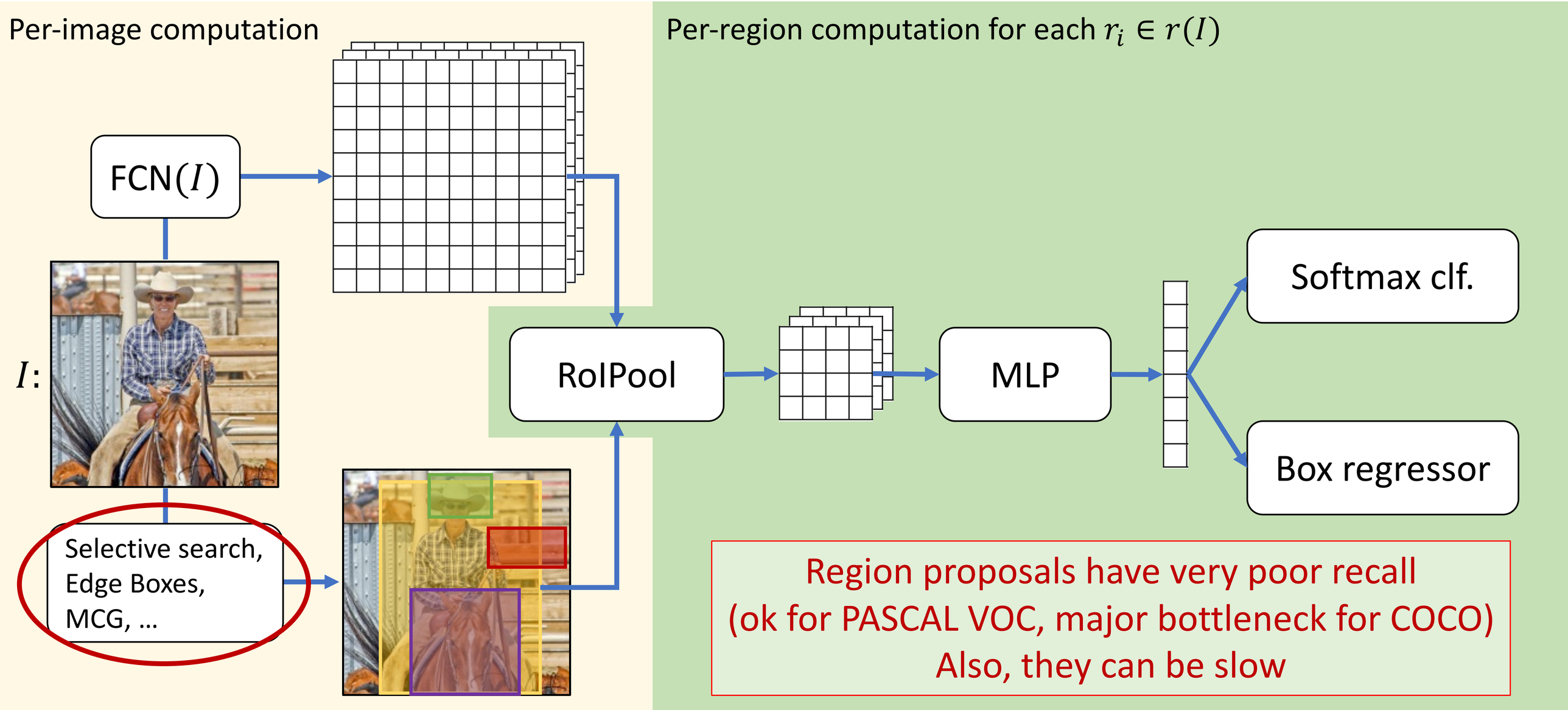
Key innovation in SPP-net
[He et al. 2014]

RoIPool (on each Proposal)

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



The Problem with Fast R-CNN



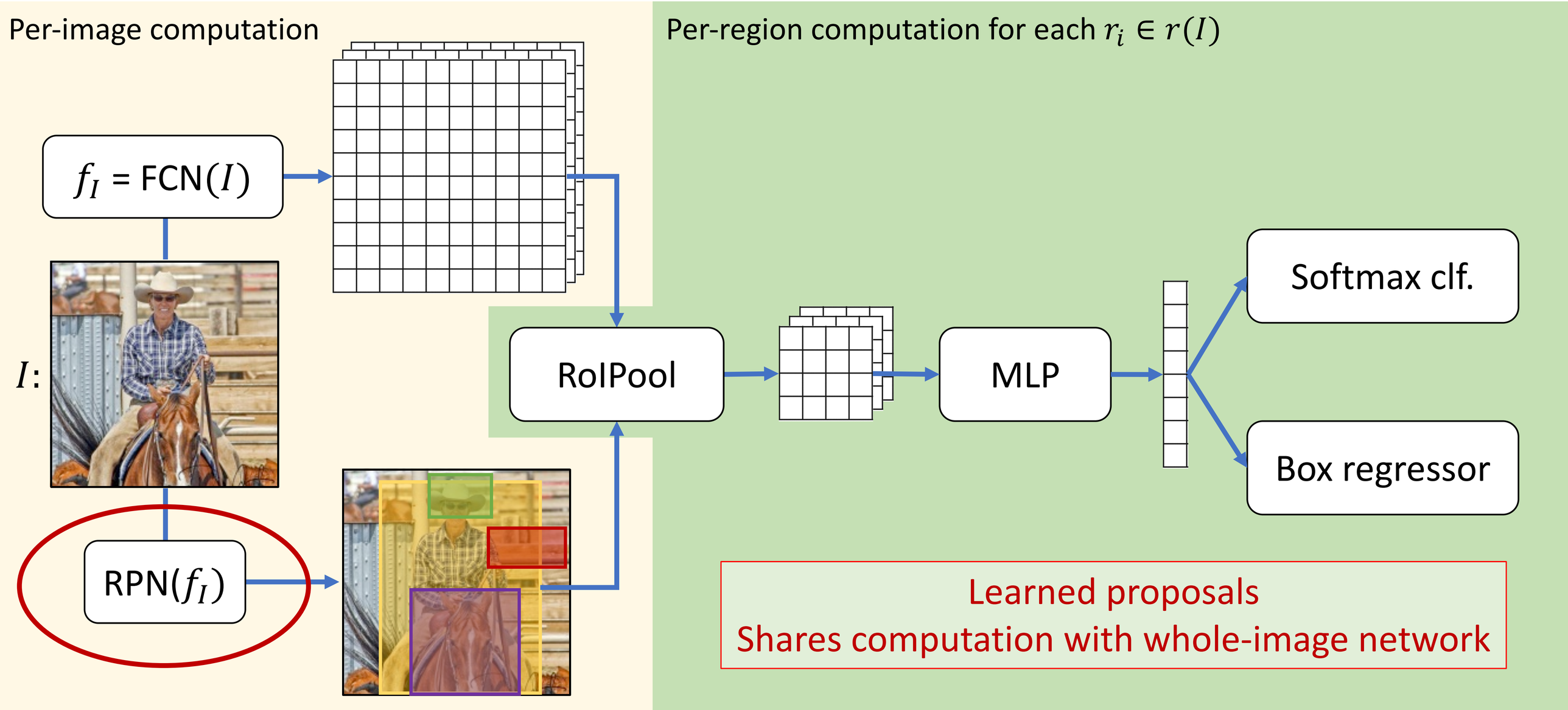
Region proposals have very poor recall
(ok for PASCAL VOC, major bottleneck for COCO)
Also, they can be slow

Fast R-CNN

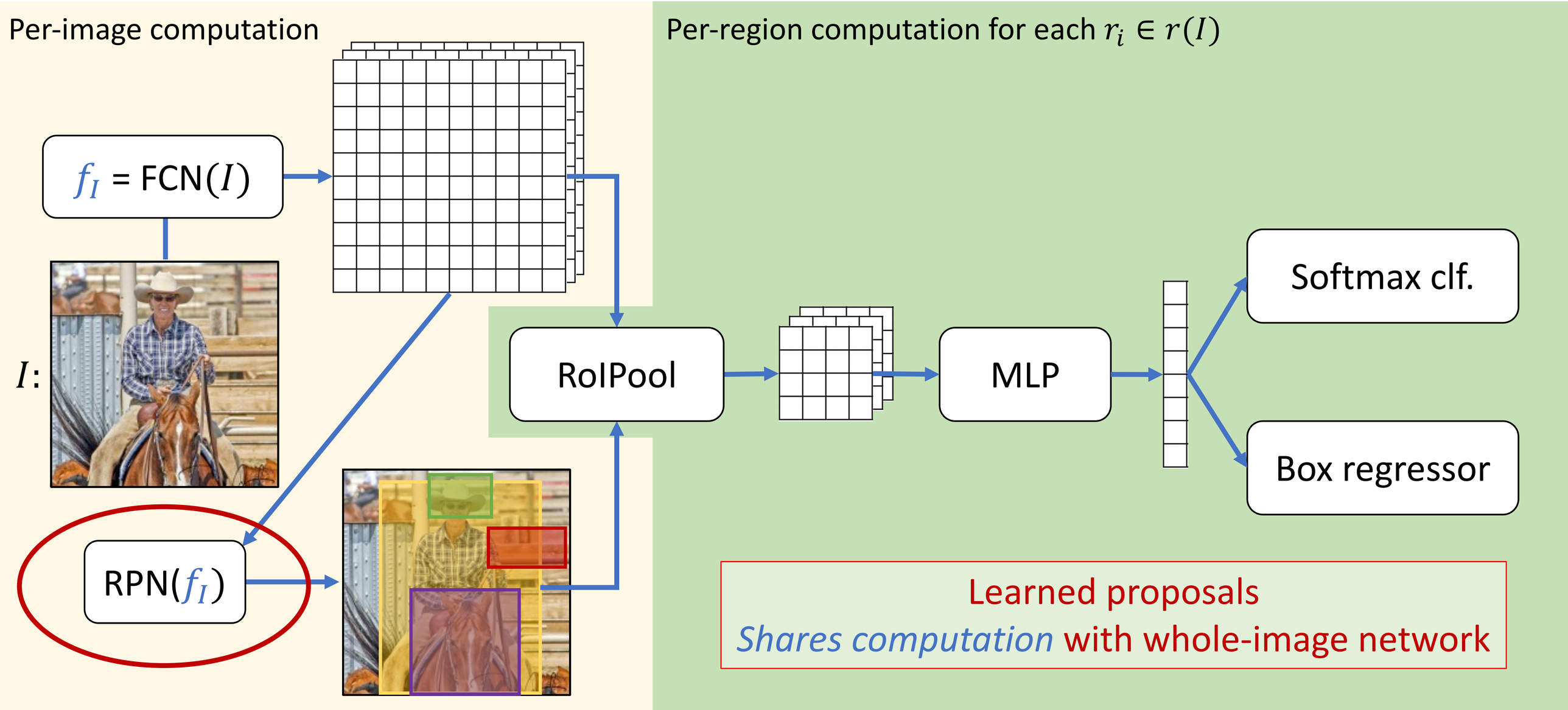
References

- K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In ECCV, 2014.
- R. Girshick. Fast R-CNN. In ICCV, 2015.

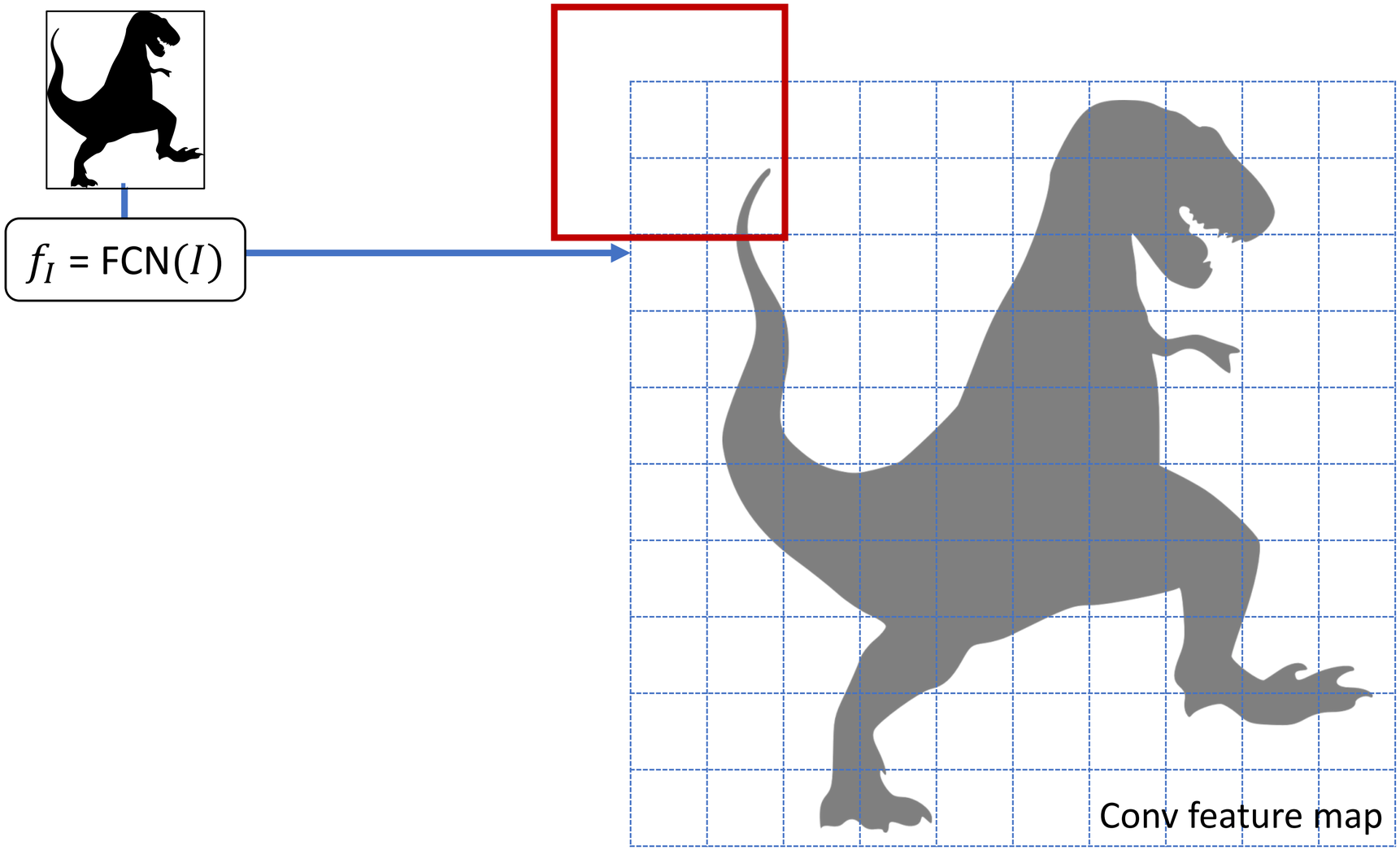
Faster R-CNN


Per-image computation
Per-region computation for each $r_i \in r(I)$
f_I = FCN(I)
I:
RPN(f_I)
RoIPool
MLP
Softmax clf.
Box regressor
Learned proposals
Shares computation with whole-image network

Faster R-CNN


Per-image computation
Per-region computation for each $r_i \in r(I)$
f_I = FCN(I)
I:
RPN(f_I)
RoIPool
MLP
Softmax clf.
Box regressor
Learned proposals
Shares computation with whole-image network

RPN: Region Proposal Network

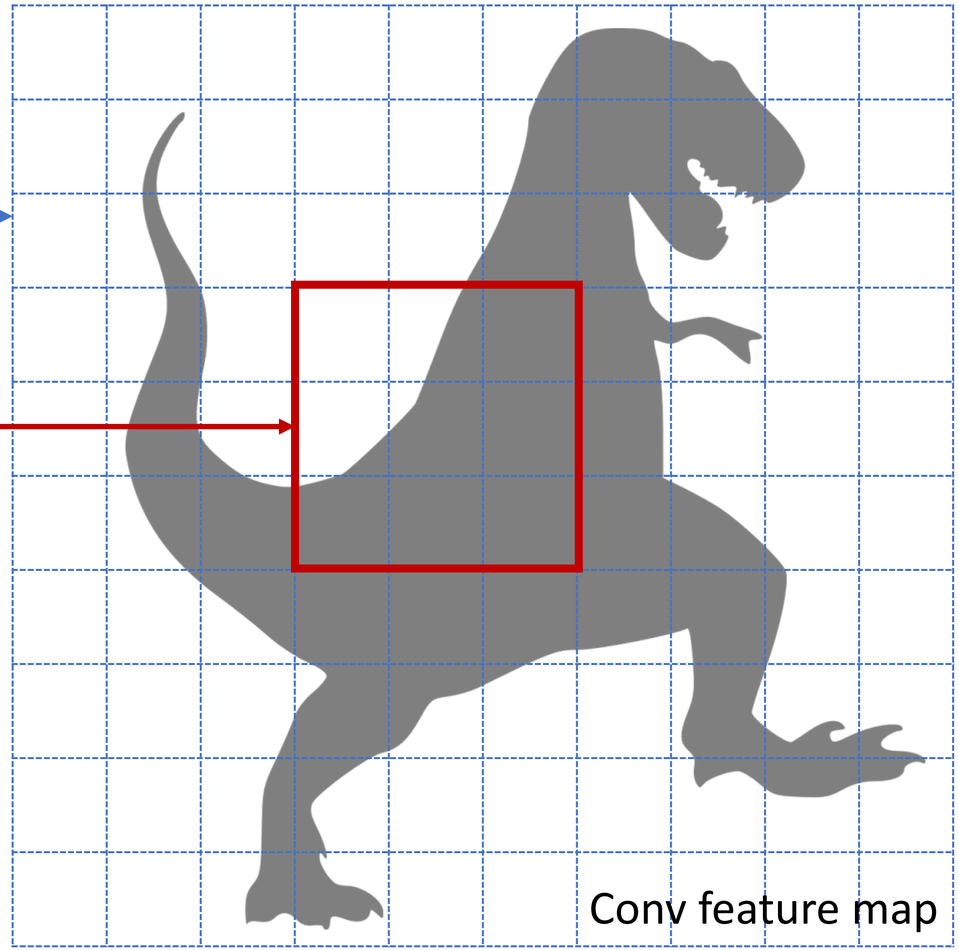


RPN: Region Proposal Network

f_I = FCN(I)

3x3 “sliding window”

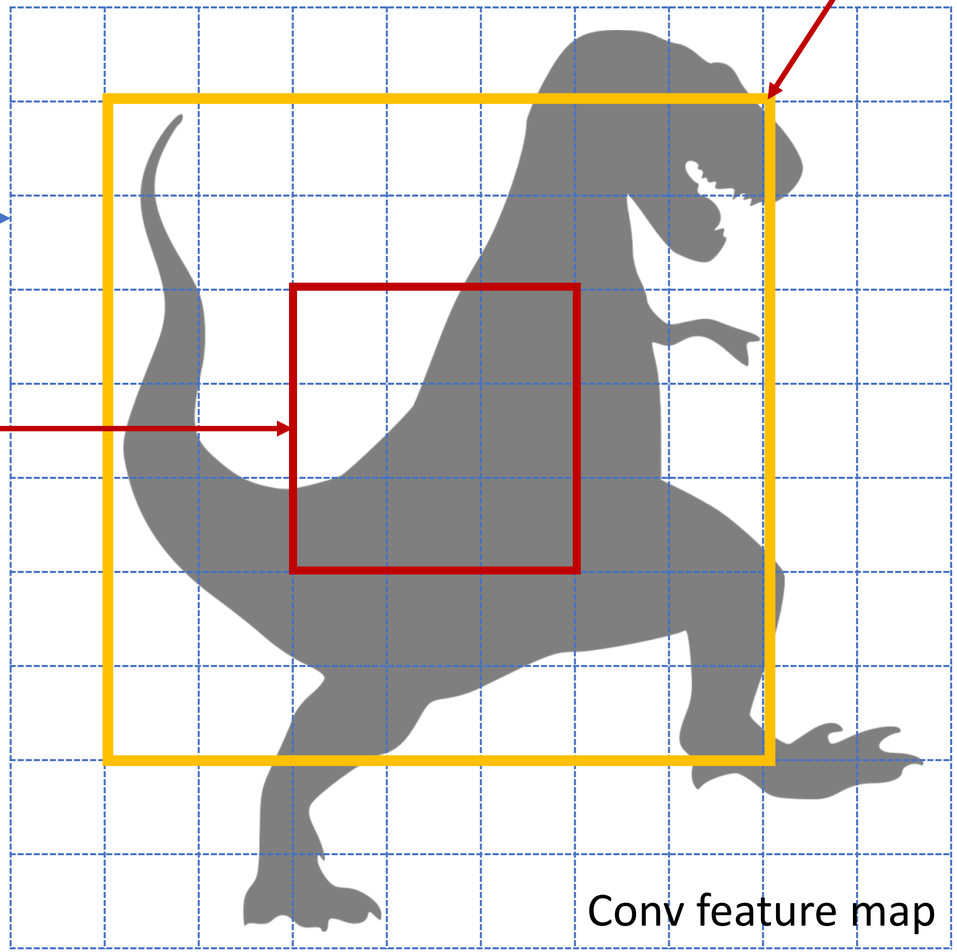
Scans the feature map looking objects



RPN: Anchor Box

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

f_I = FCN(I)



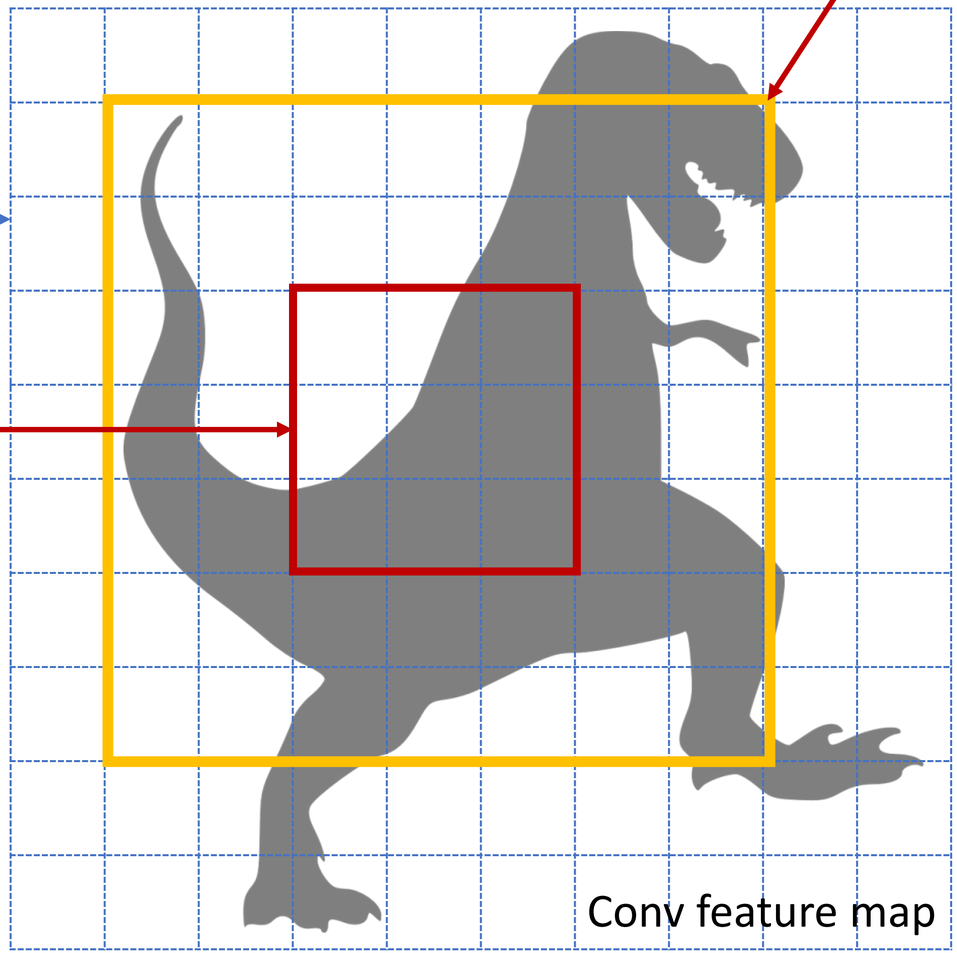
3x3 “sliding window”

Scans the feature map looking objects

RPN: Anchor Box

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

f_I = FCN(I)



3x3 “sliding window”

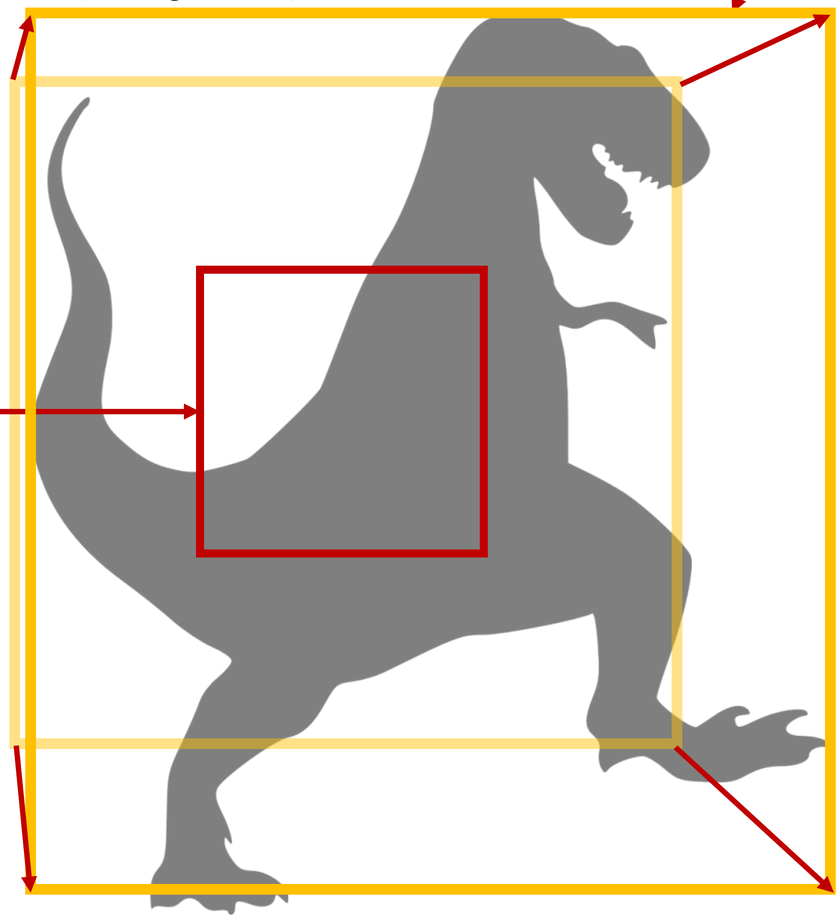
- Objectness classifier
- Box regressor predicting (dx, dy, dh, dw)

RPN: Prediction (on object)

Anchor box: transformed by box regressor

Objectness score

P(object) = 0.94



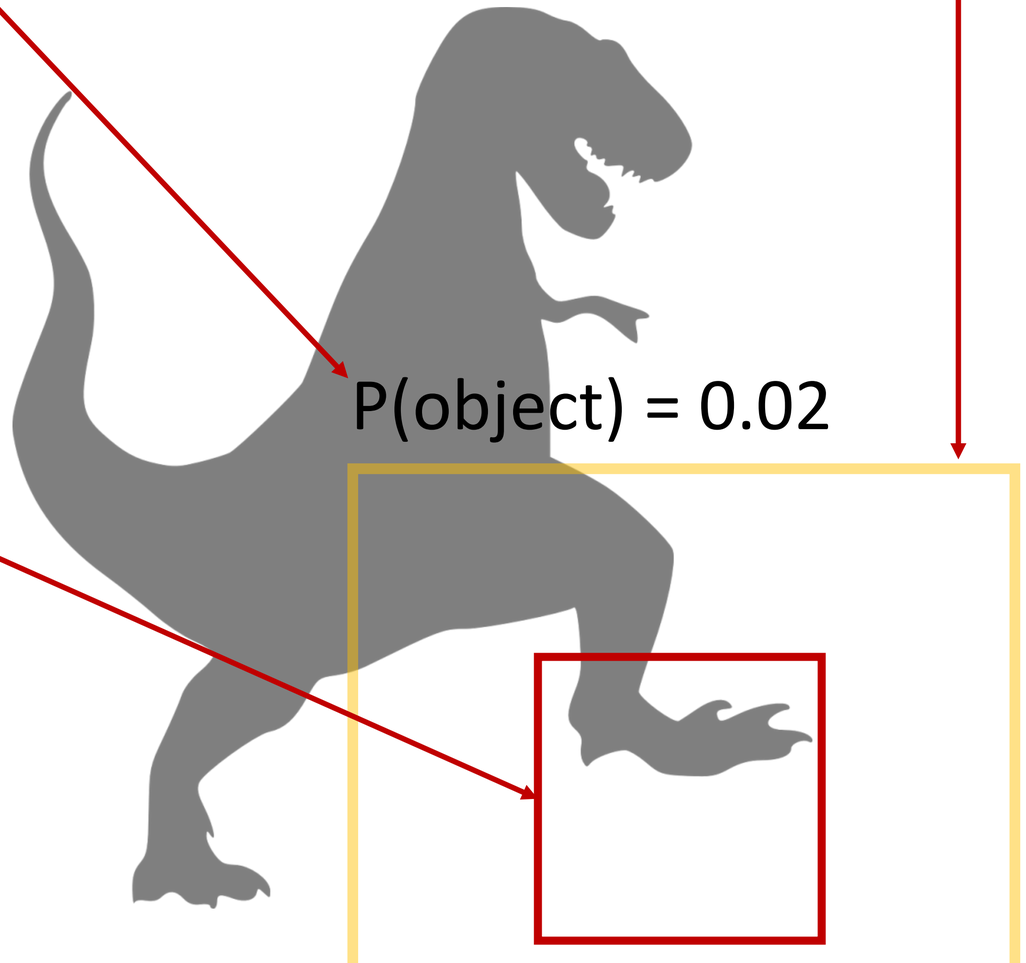
3x3 “sliding window”

- Objectness classifier
- Box regressor predicting (dx, dy, dh, dw)

RPN: Prediction (off object)

Objectness score

Anchor box: transformed by box regressor



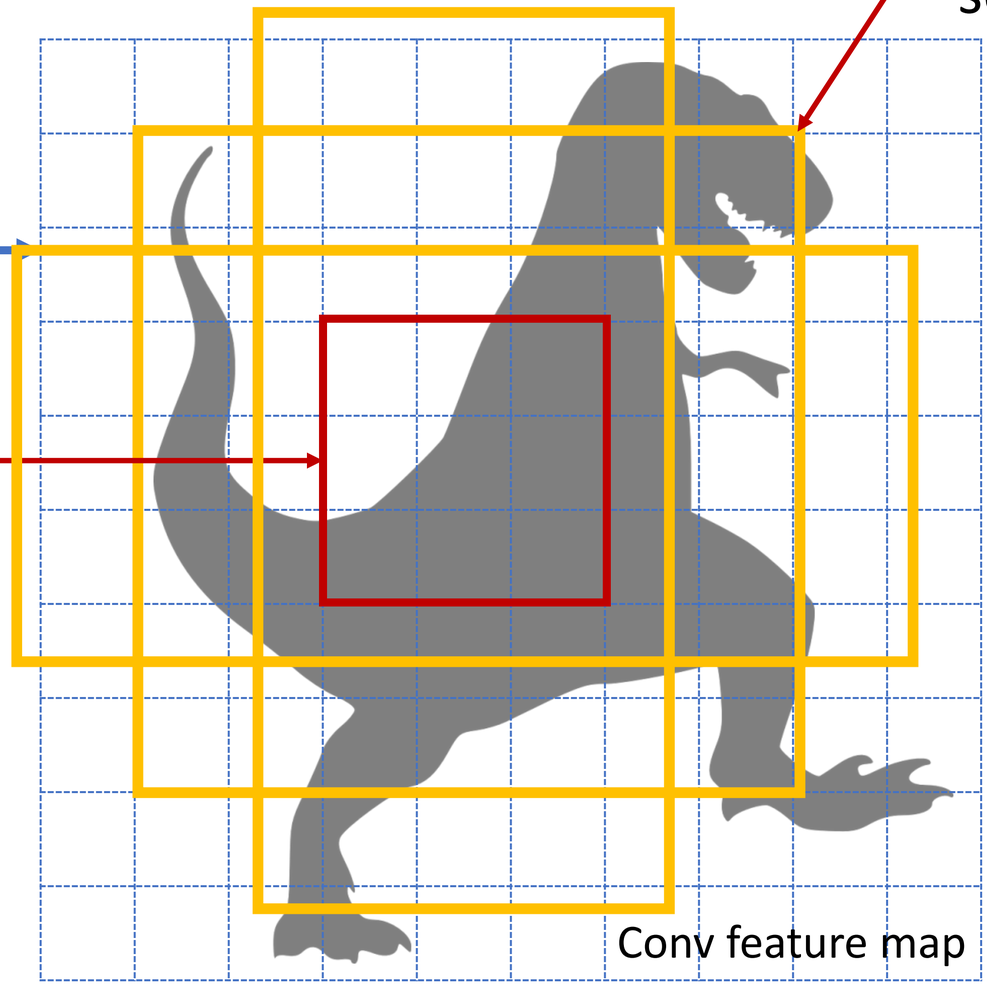
3x3 “sliding window”

- Objectness classifier
- Box regressor predicting (dx, dy, dh, dw)

RPN: Multiple Anchors

Anchor boxes: *K* anchors per location with different scales and aspect ratios

f_I = FCN(I)



3x3 “sliding window”

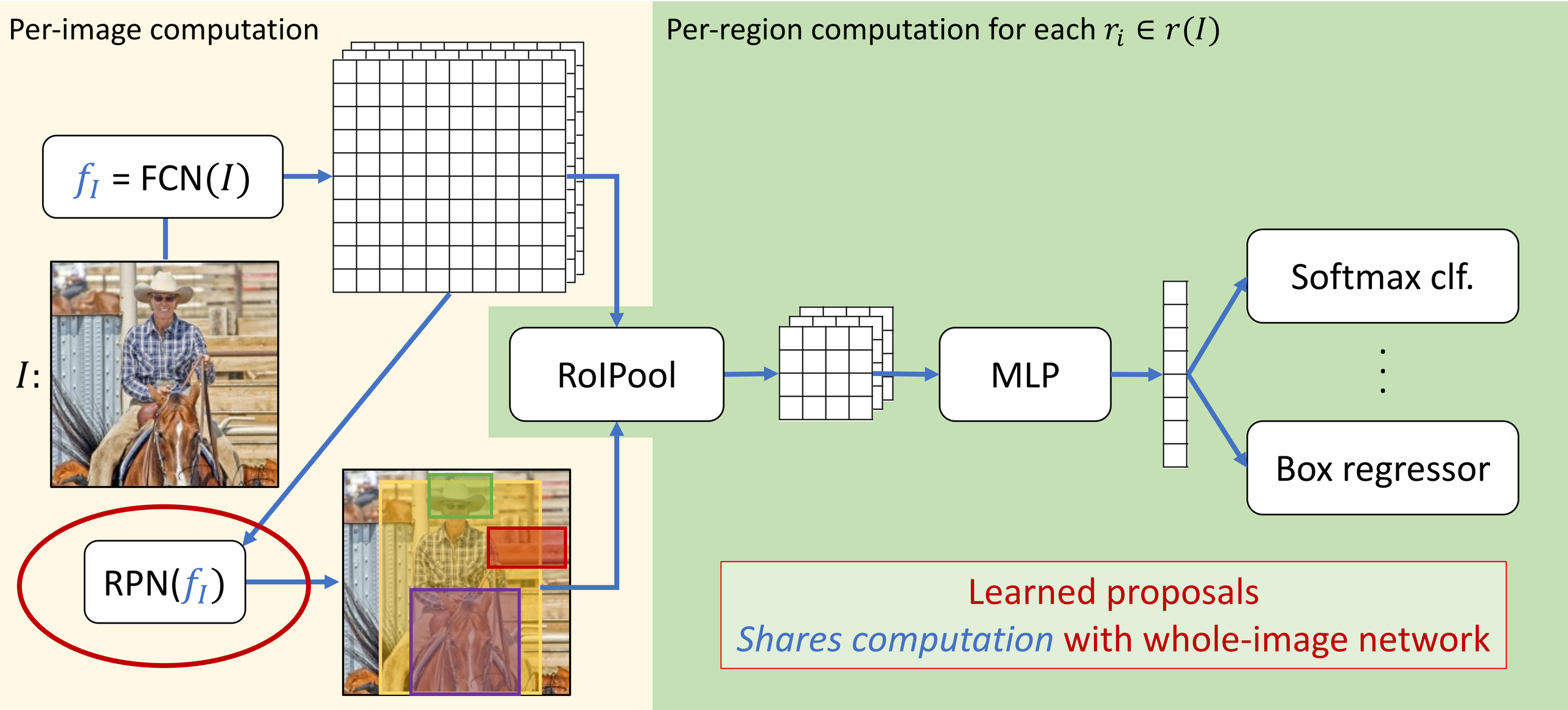
- *K* objectness classifiers
- *K* box regressors

Faster R-CNN

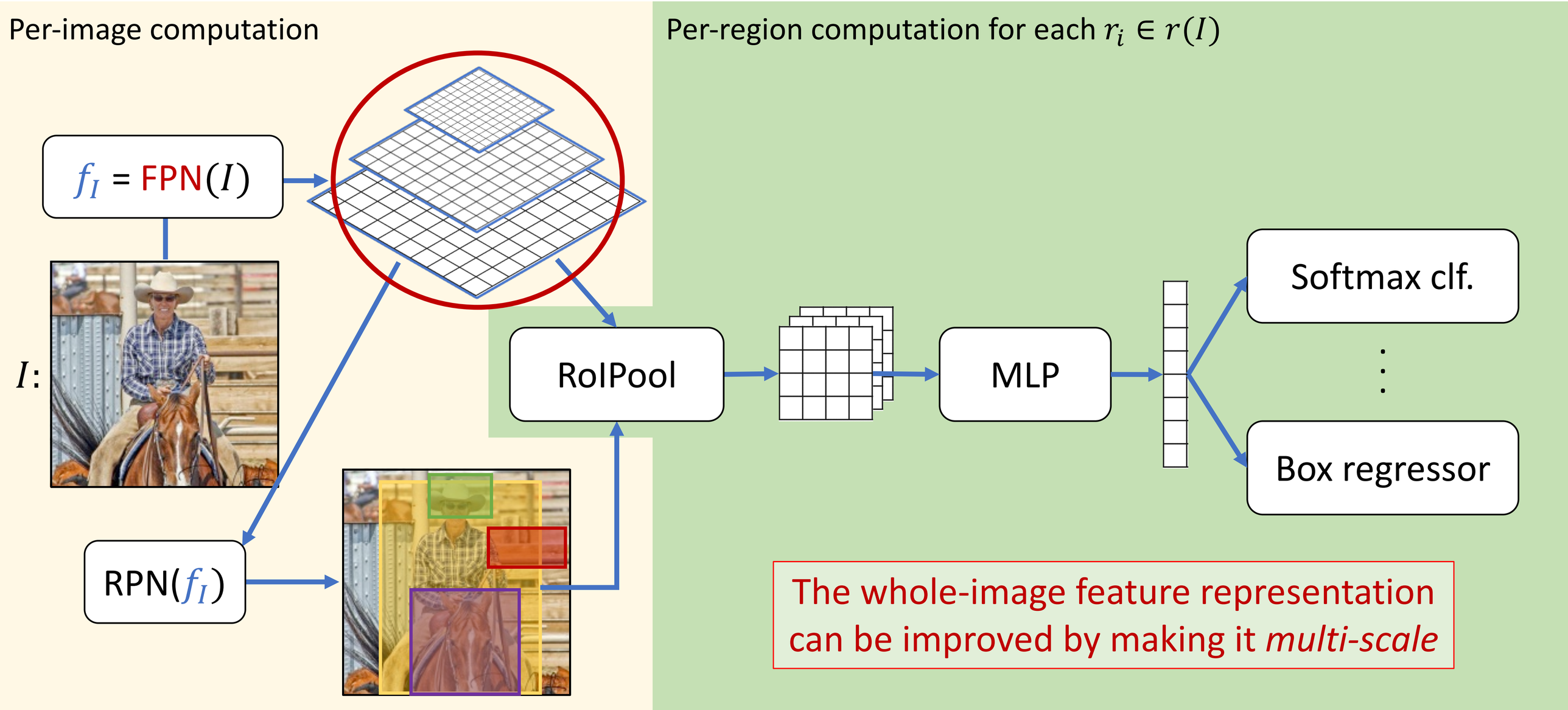
References

- D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In CVPR, 2014.
- P. O. Pinheiro, R. Collobert, and P. Dollar. Learning to segment object candidates. In NIPS, 2015.
- S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In NIPS, 2015.

Faster R-CNN

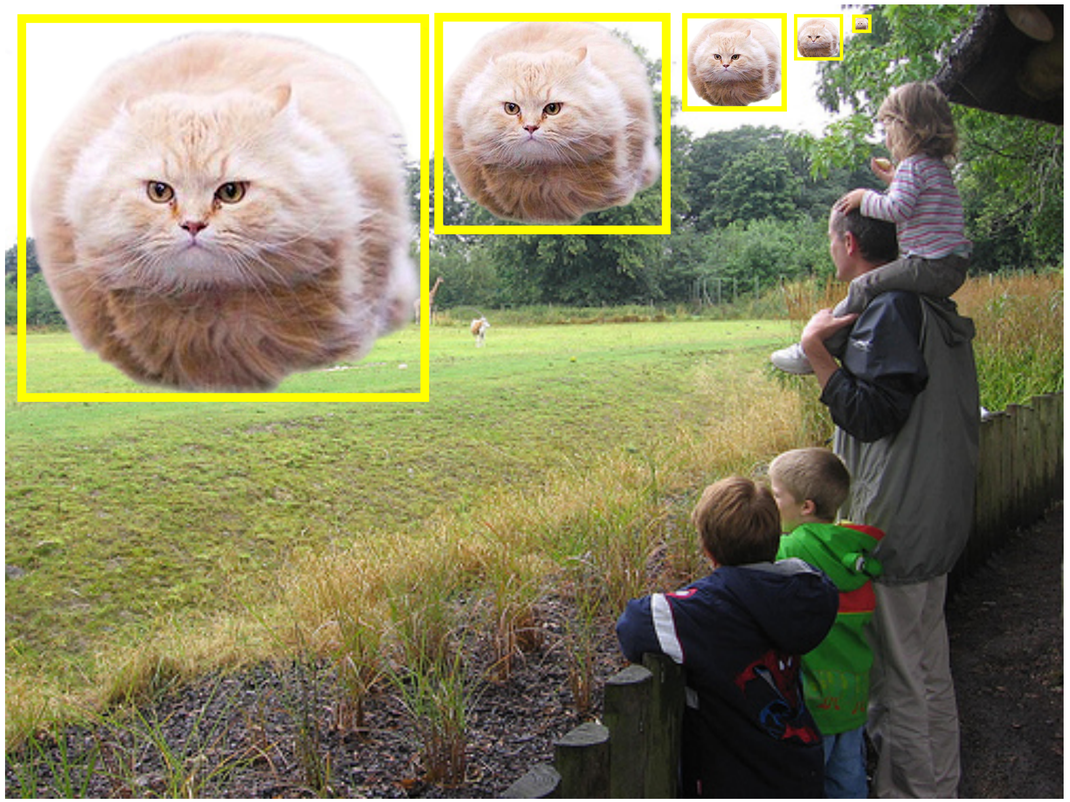

Per-image computation
Per-region computation for each $r_i \in r(I)$
f_I = FCN(I)
I:
RPN(f_I)
RoIPool
MLP
Softmax clf.
Box regressor
Learned proposals
Shares computation with whole-image network

Faster R-CNN with a Feature Pyramid Network



The whole-image feature representation can be improved by making it *multi-scale*

FPN: Improving Scale Invariance and Equivariance



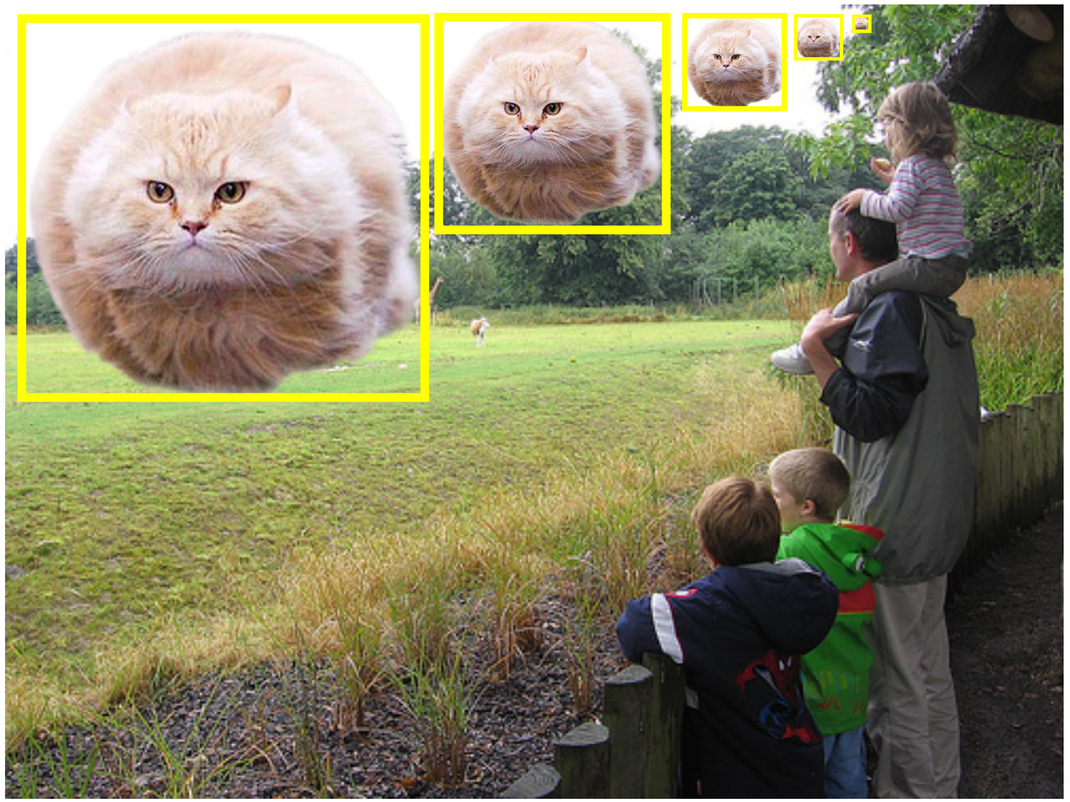
Detectors need to

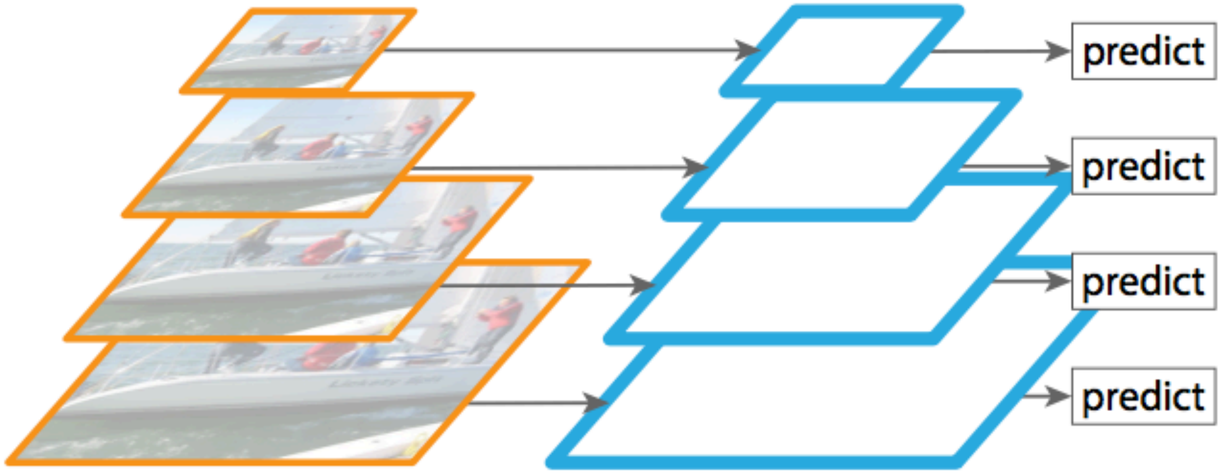
1. classify and
2. localize

objects over a wide range of scales

FPN improves this ability

Strategy 1: Image Pyramid



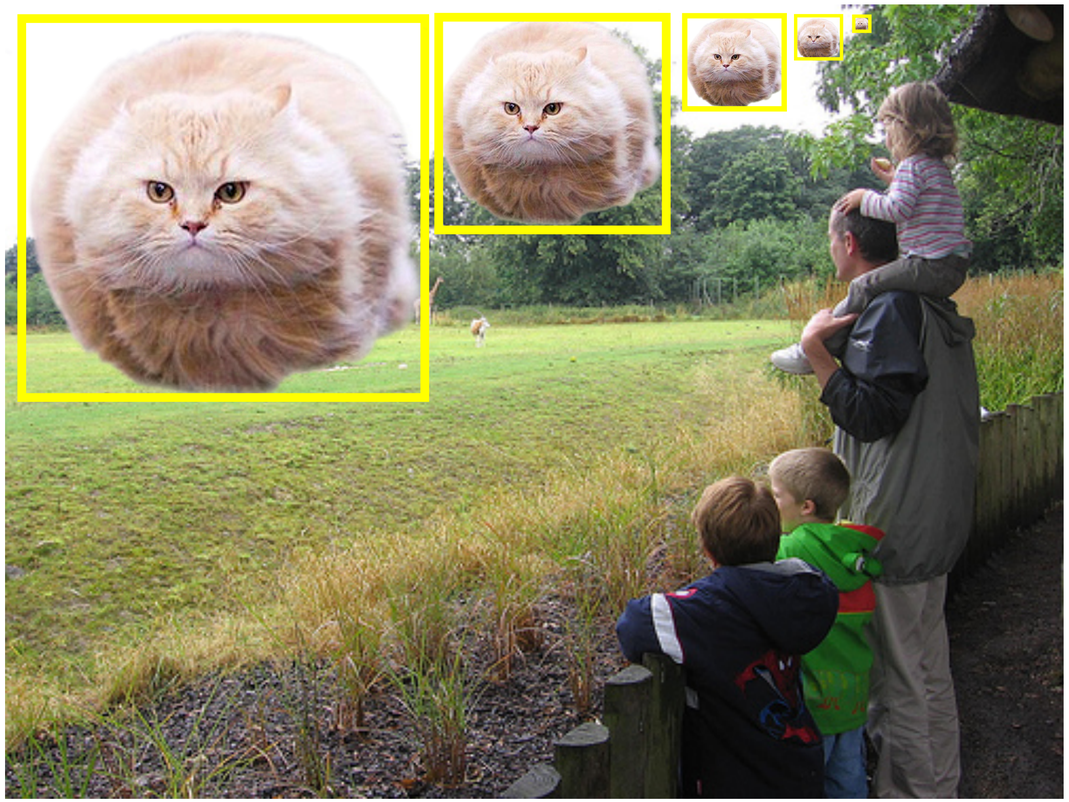


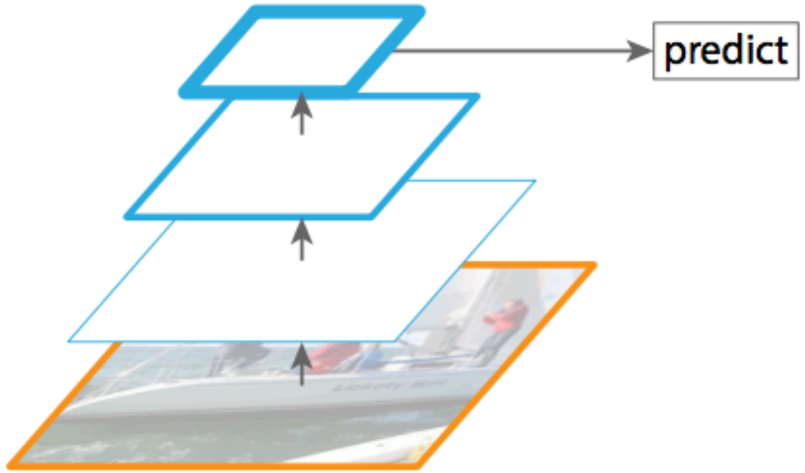
(a) Featurized image pyramid

Standard solution – *slow!*

(E.g., Viola & Jones, HOG, DPM, SPP-net, multi-scale Fast R-CNN, …)

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



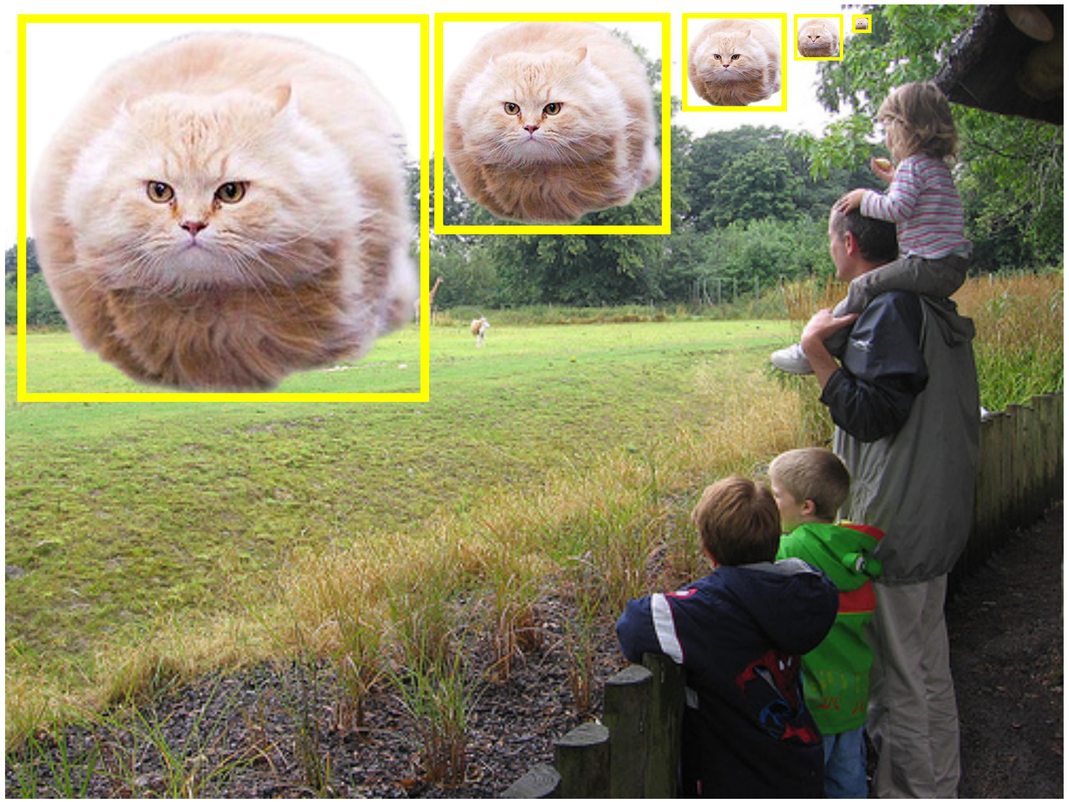


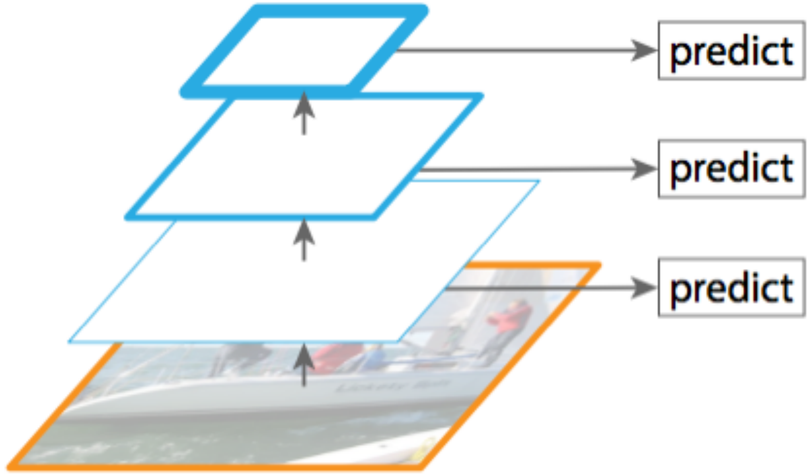
(b) Single feature map

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

(E.g., Fast R-CNN, YOLO, …)

Strategy 3: Naïve In-network Pyramid



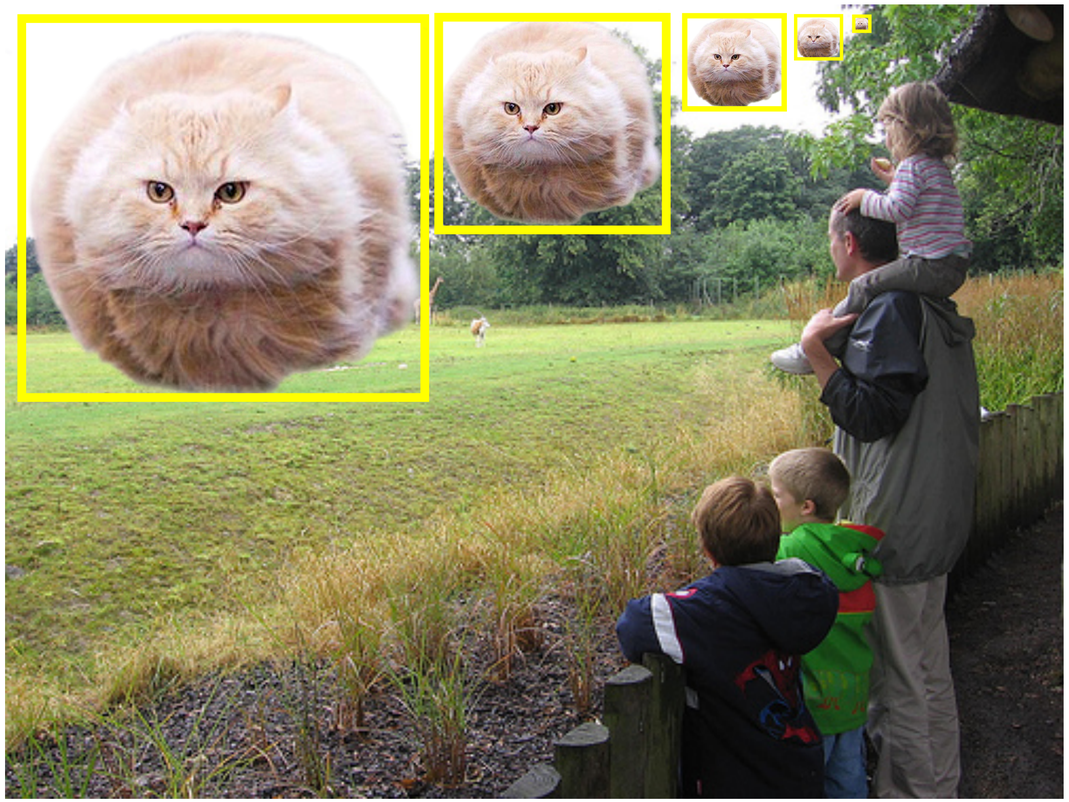


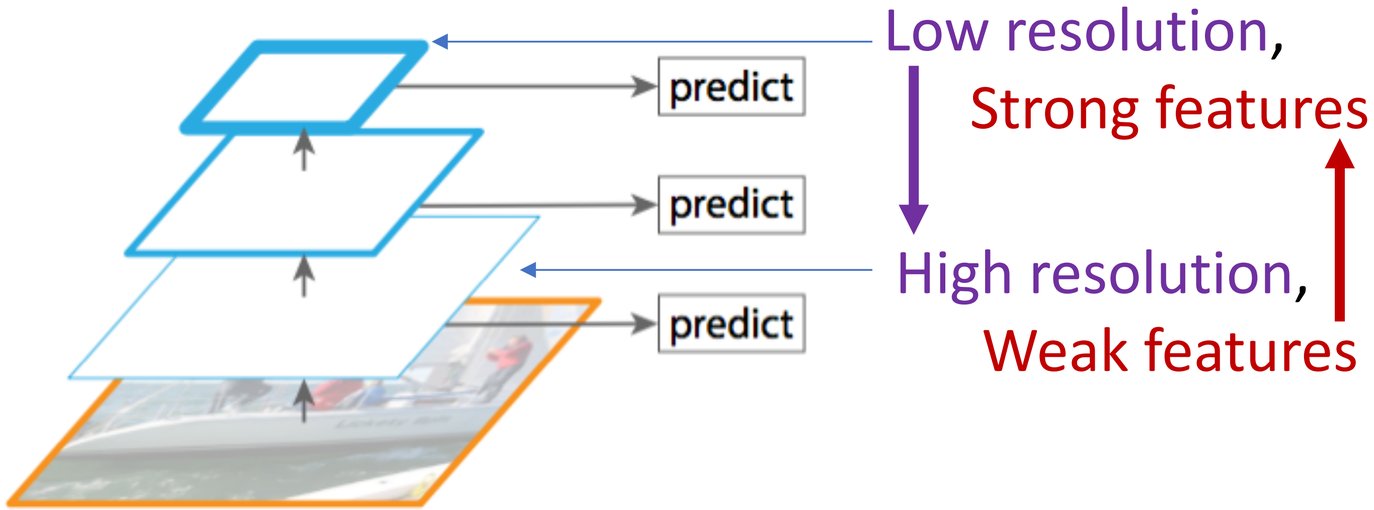
(c) Pyramidal feature hierarchy

Use the internal pyramid – *fast, suboptimal*

(E.g., ≈ SSD, …)

Strategy 3: Naïve In-network Pyramid



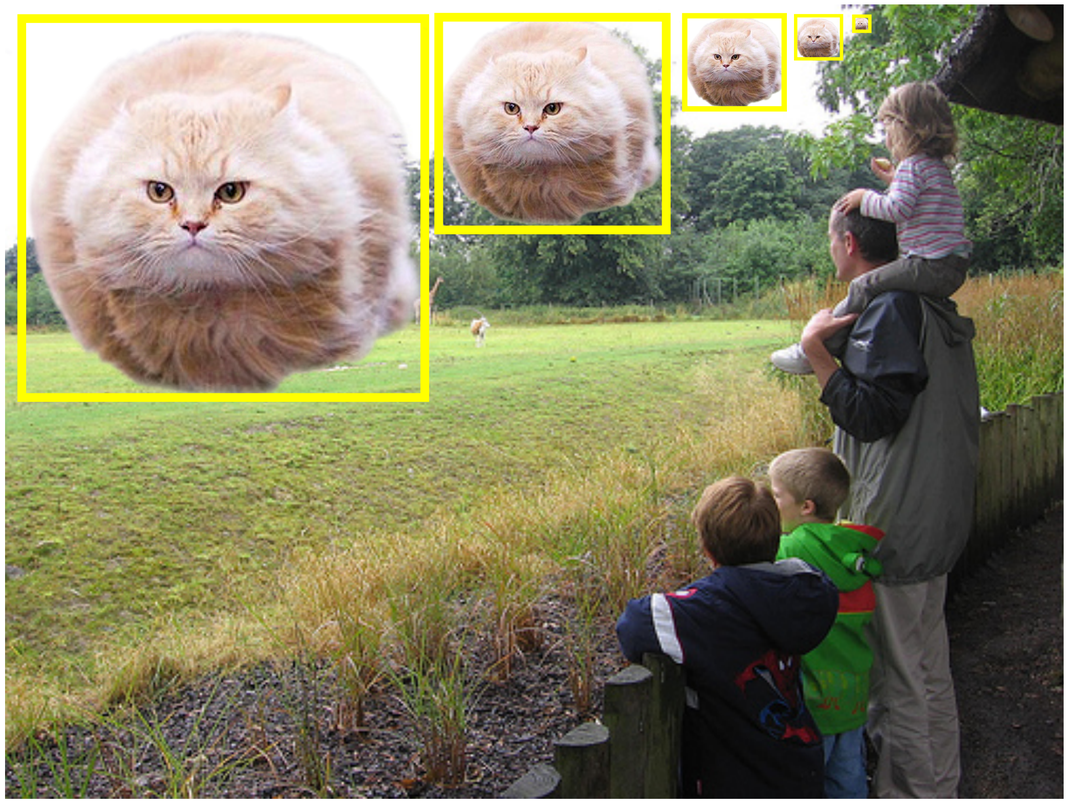


(c) Pyramidal feature hierarchy

Use the internal pyramid – *fast, suboptimal*

(E.g., ≈ SSD, …)

Strategy 4: Feature Pyramid Network

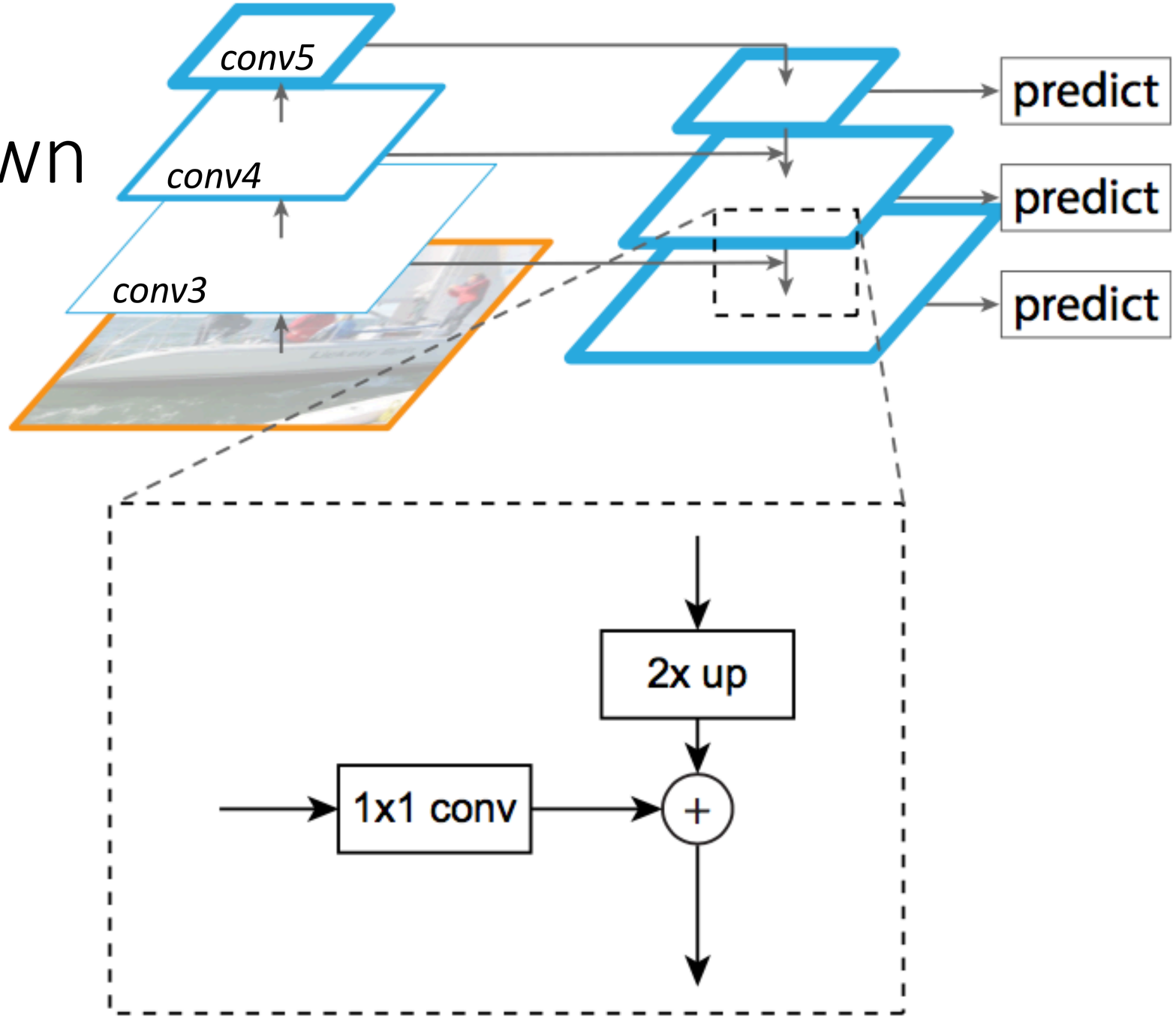


(d) Feature Pyramid Network

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

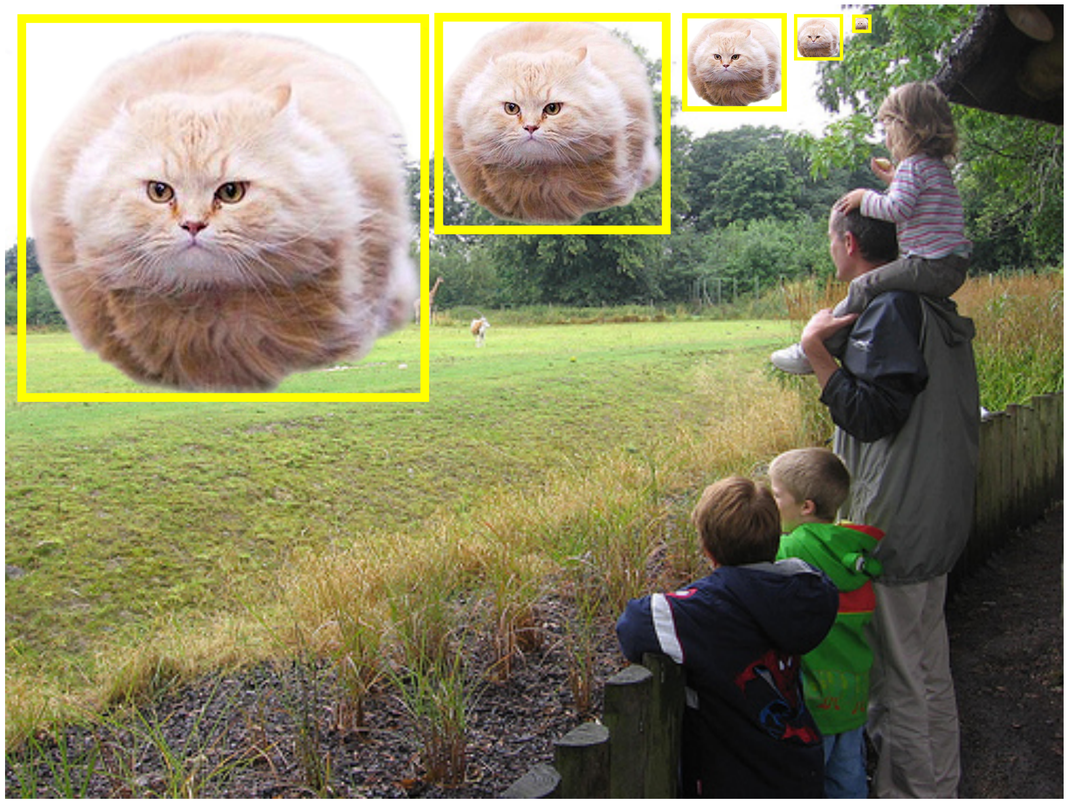
Lin et al. Feature Pyramid Networks for Object Detection. CVPR 2017.

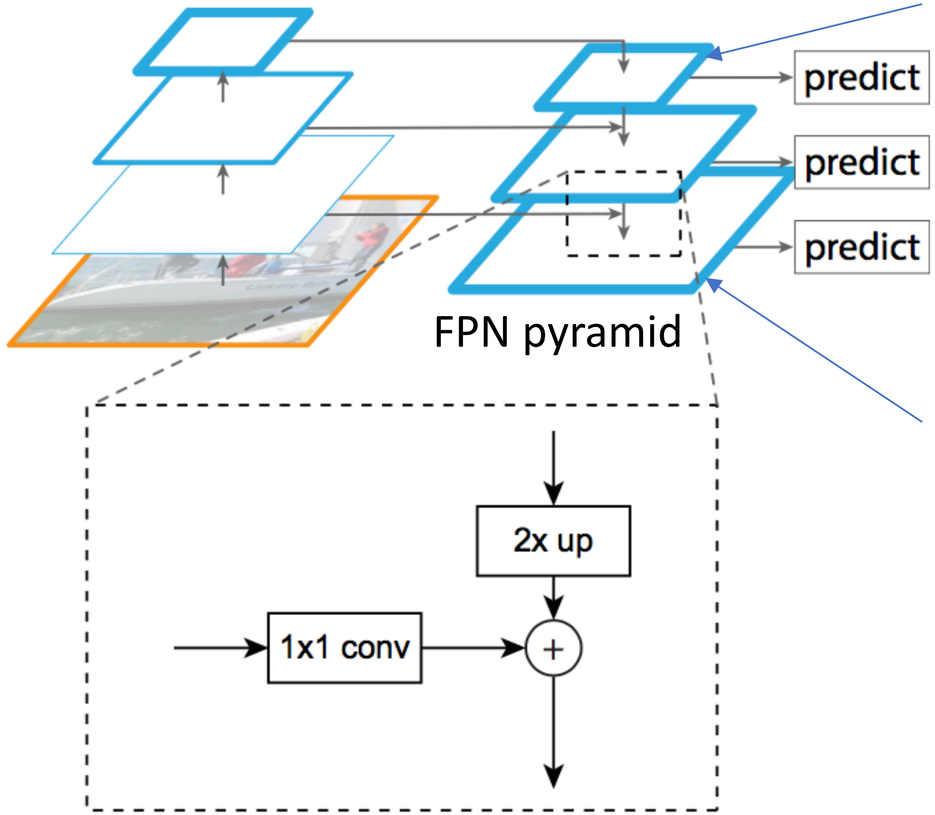
FPN Top-down Refinement Module



Lin et al. Feature Pyramid Networks for Object Detection. CVPR 2017.

No Compromise on Feature Quality, still Fast





Low resolution,
Strong features

High resolution,
Strong features

Lin et al. Feature Pyramid Networks for Object Detection. CVPR 2017. See also: Shrivastava's TDM.

FPN – A Generic Backbone Modification

Generates a feature pyramid—*useful in many applications!*

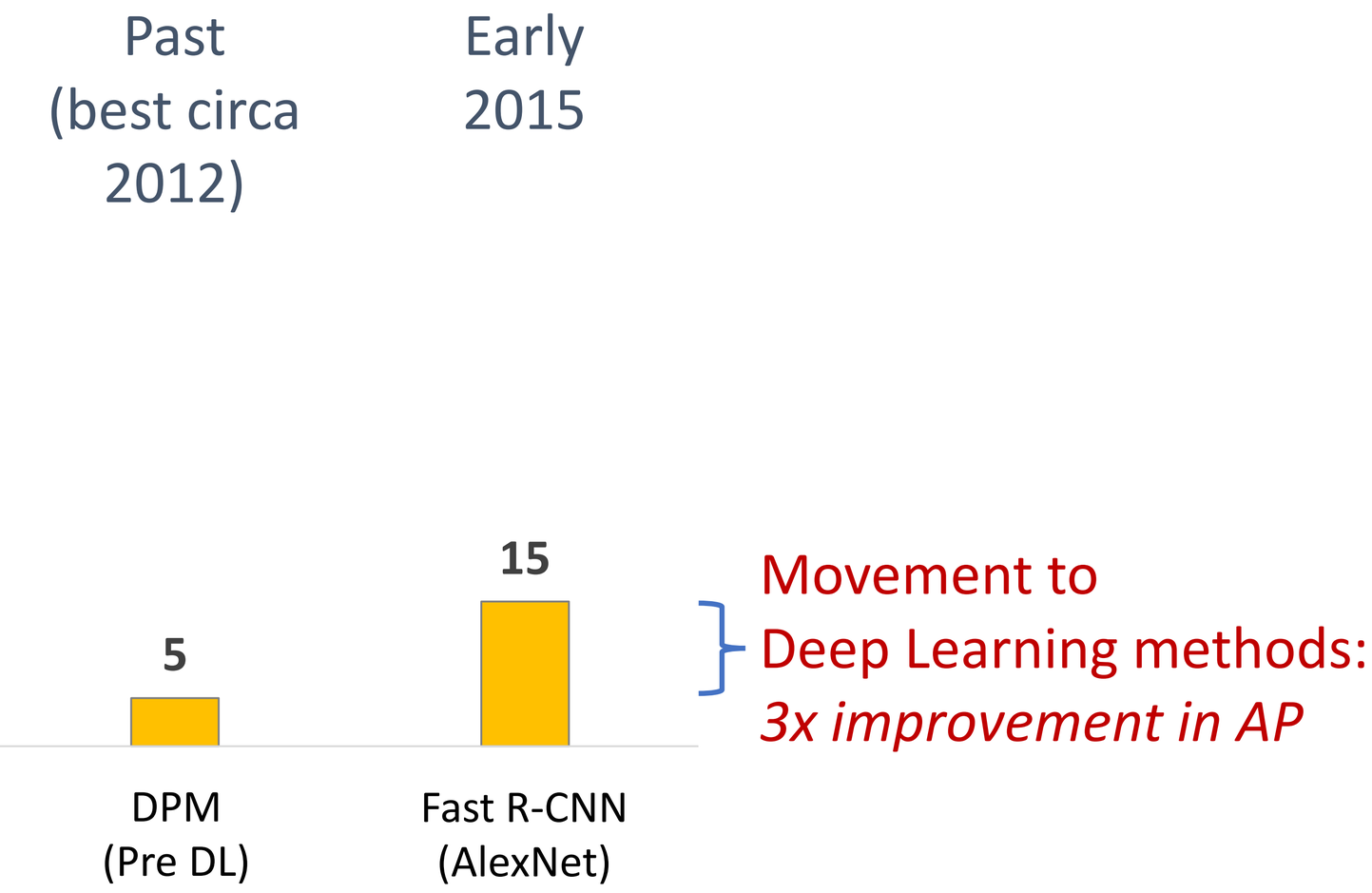
- RPN
- Fast/er R-CNN
- Mask R-CNN
- RetinaNet
- Semantic segmentation

Feature Pyramid Network (FPN)

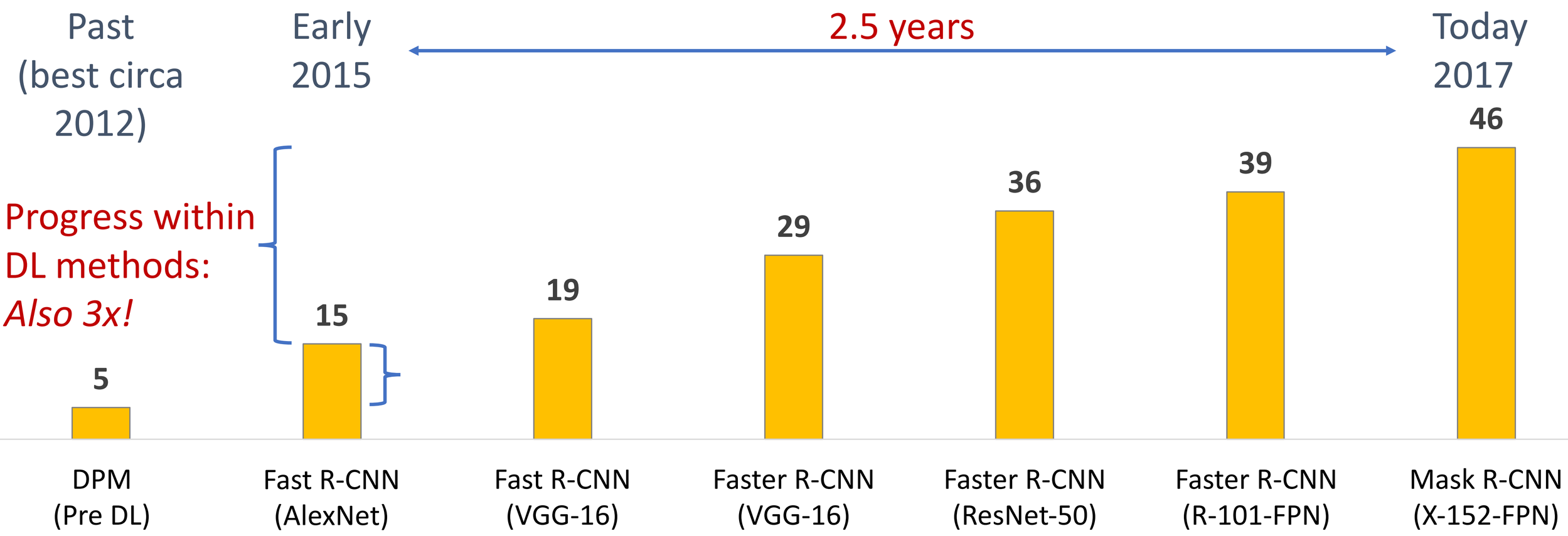
References

- O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In MIC- CAI, 2015.
- P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. Learning to refine object segments. In ECCV, 2016.
- W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. SSD: Single shot multibox detector. In ECCV, 2016.
- A. Shrivastava, R. Sukthankar, J. Malik, and A. Gupta. Beyond skip connections: Top-down modulation for object detection. arXiv:1612.06851, 2016.
- T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, S. Belongie. Feature Pyramid Networks for Object Detection. In CVPR, 2017.

COCO Object Detection Average Precision (%)

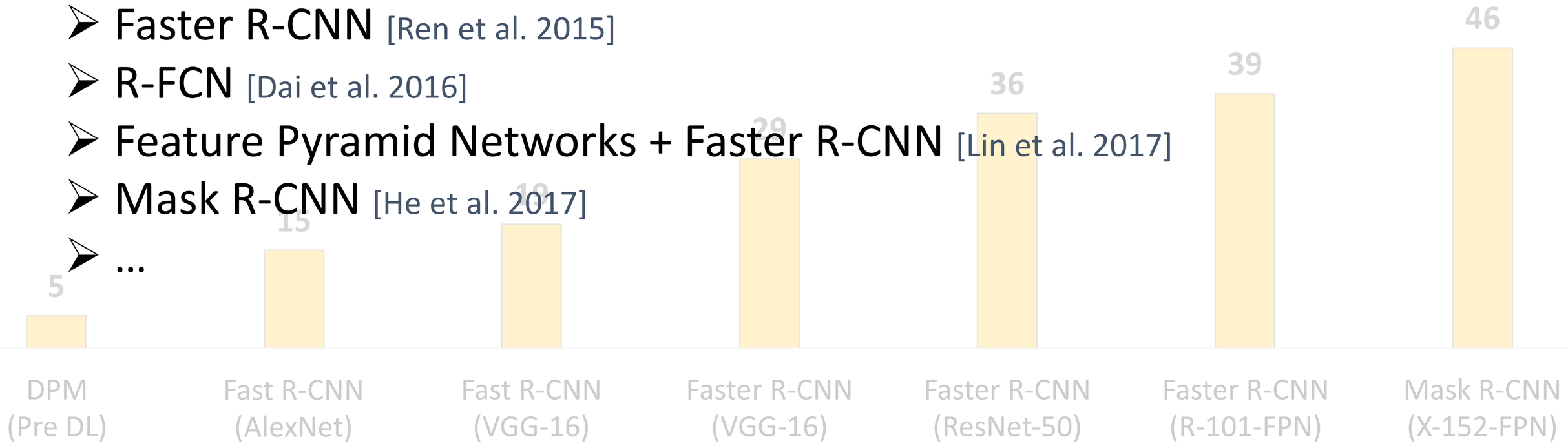


Summary: Progress through Better Detector Design and Better Backbones



Steady Progress

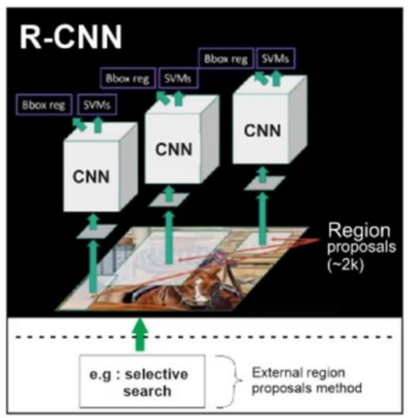
- R-CNN [Girshick et al. 2014]
- SPP-net [He et al. 2014]
- Fast R-CNN [Girshick. 2015]
- Faster R-CNN [Ren et al. 2015]
- R-FCN [Dai et al. 2016]
- Feature Pyramid Networks + Faster R-CNN [Lin et al. 2017]
- Mask R-CNN [He et al. 2017]
- …

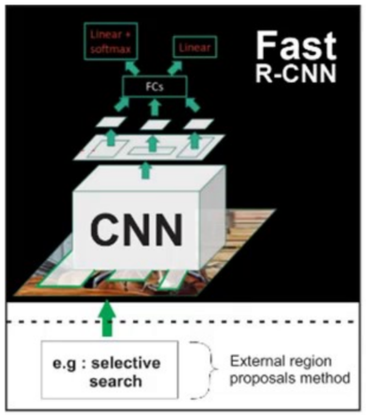


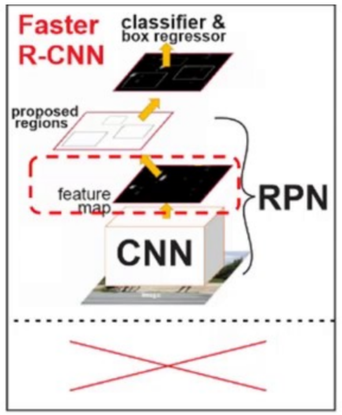
Alternative look at Region-based CNNs:
Per Image vs. per Region Computations

Object Detection Region-based CNNs

Stanford: CS231n (video)

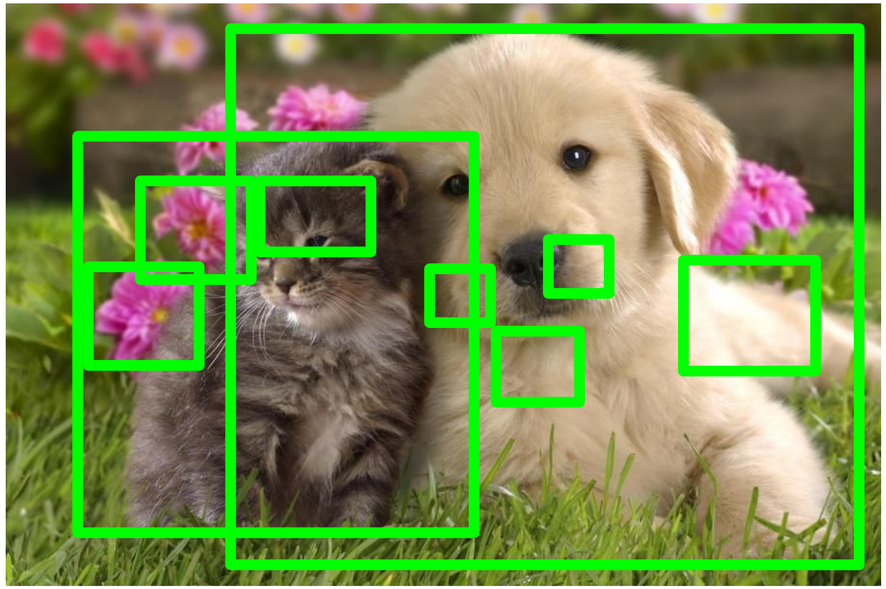






Region Proposals

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



Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

R-CNN

Input image

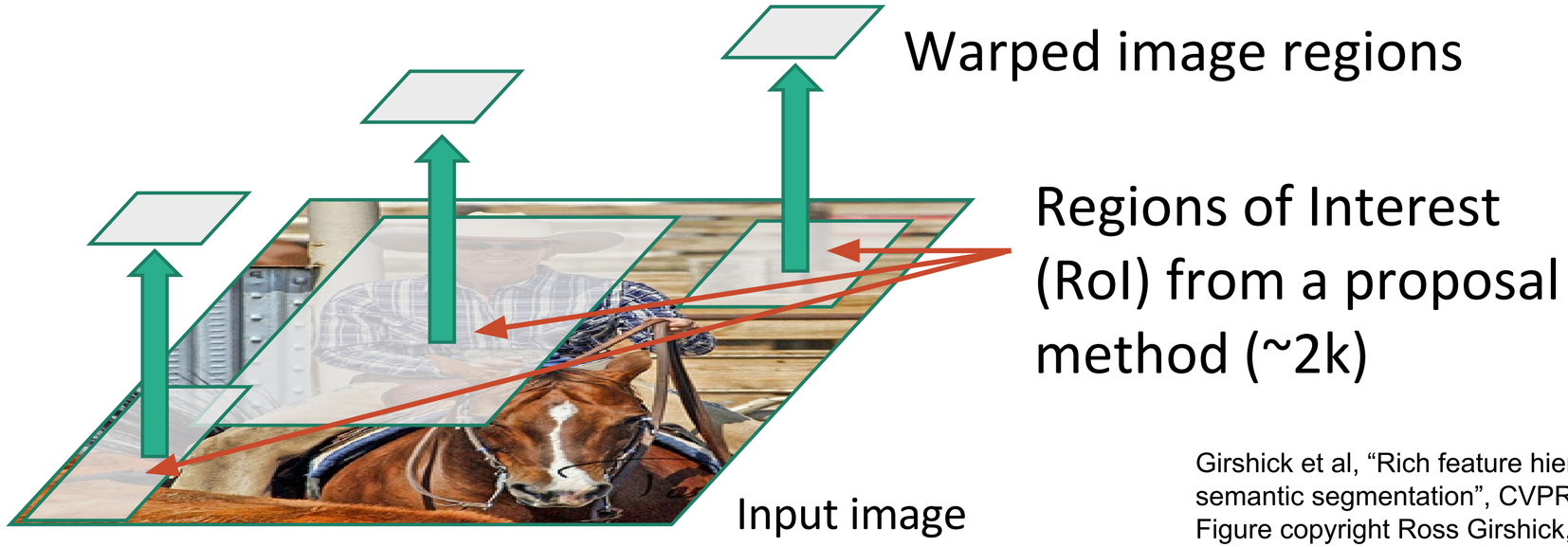
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN

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

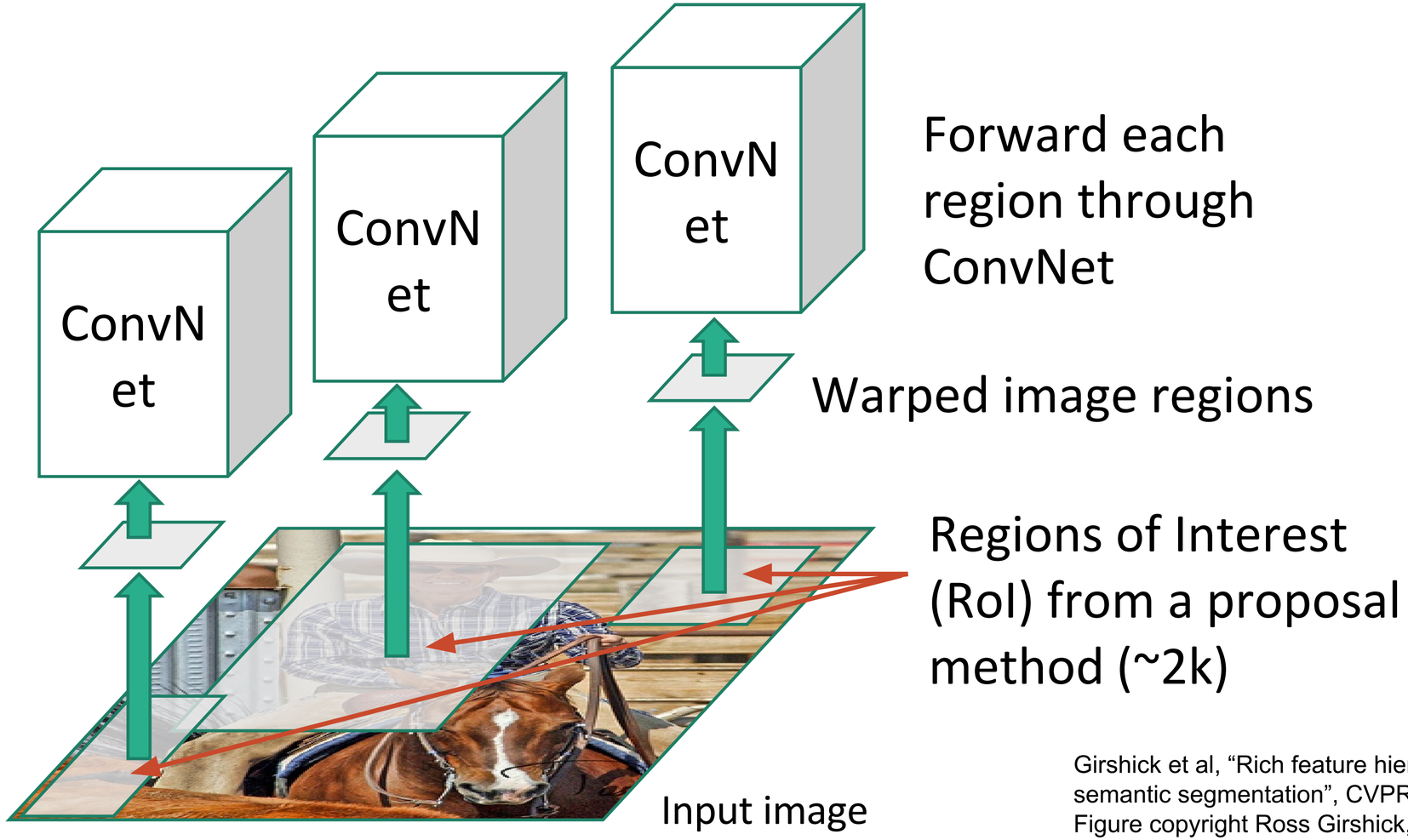
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN



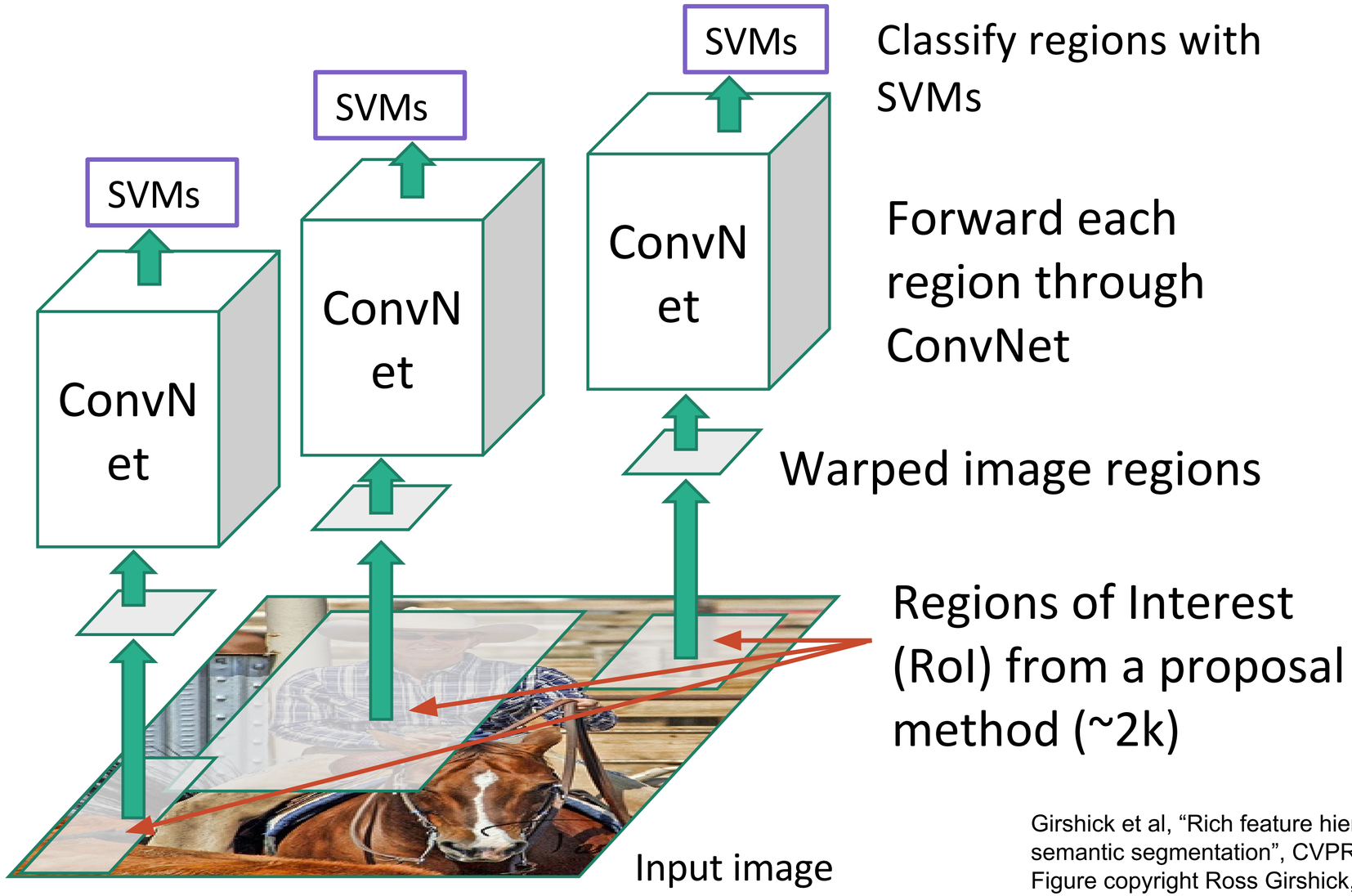
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN



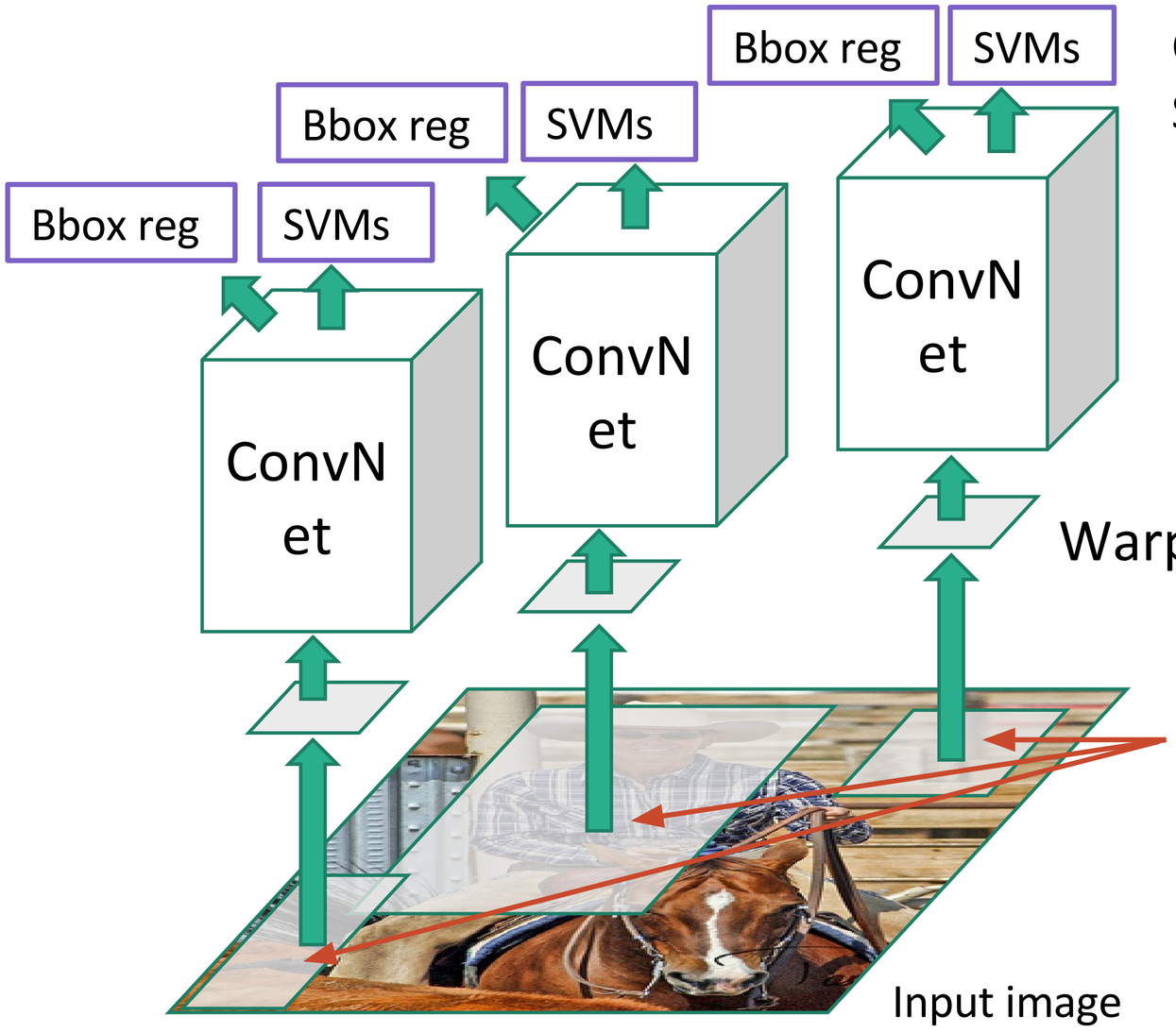
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN



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

R-CNN



Linear Regression for bounding box offsets

Classify regions with SVMs

Forward each region through ConvNet

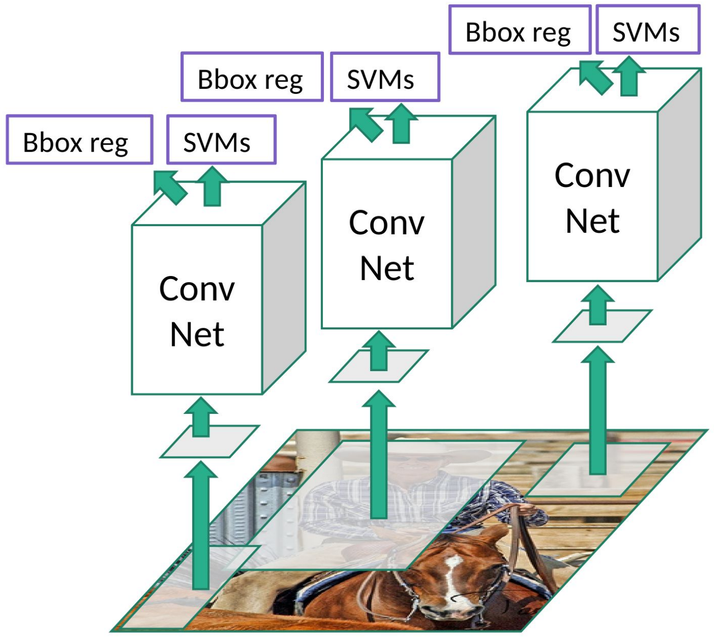
Warped image regions

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

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

R-CNN: Problems

- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressions (least squares)
- Training is slow (84h), takes a lot of disk space
- Inference (detection) is slow
  - 47s / image with VGG16 [Simonyan & Zisserman. ICLR15]
  - Fixed by SPP-net [He et al. ECCV14]



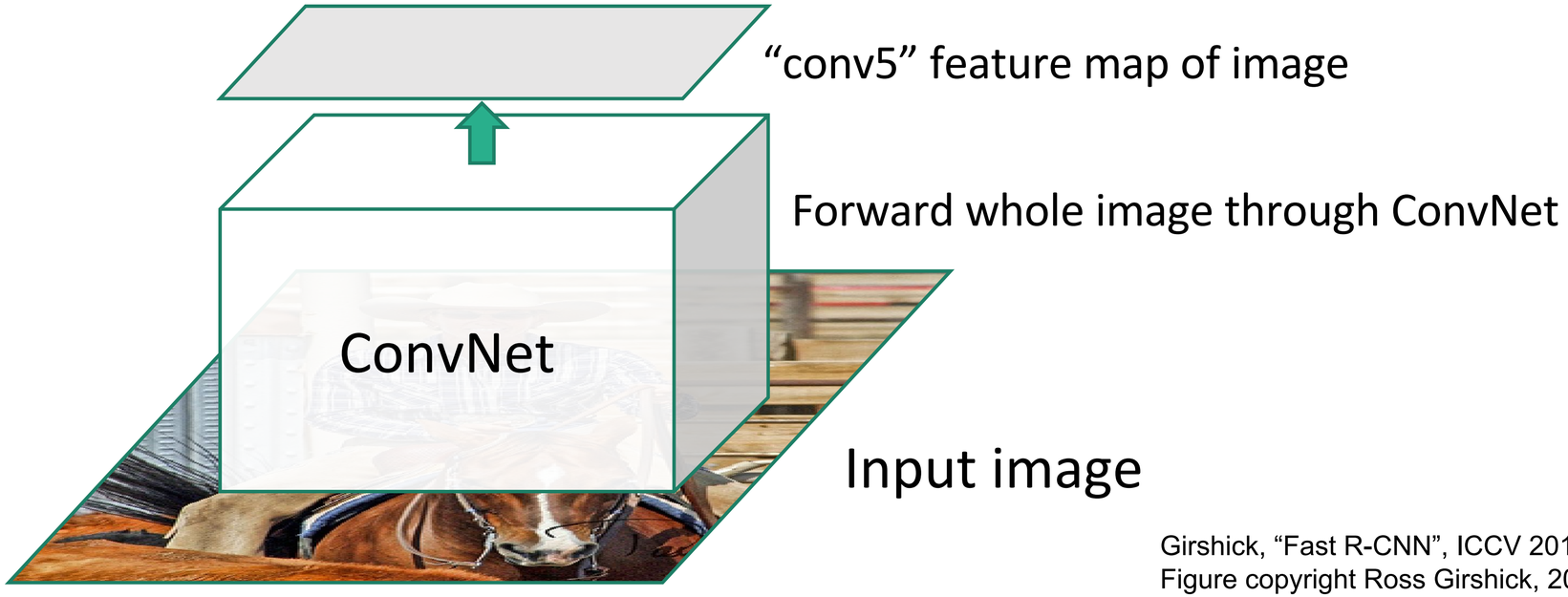
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Slide copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN

Input image

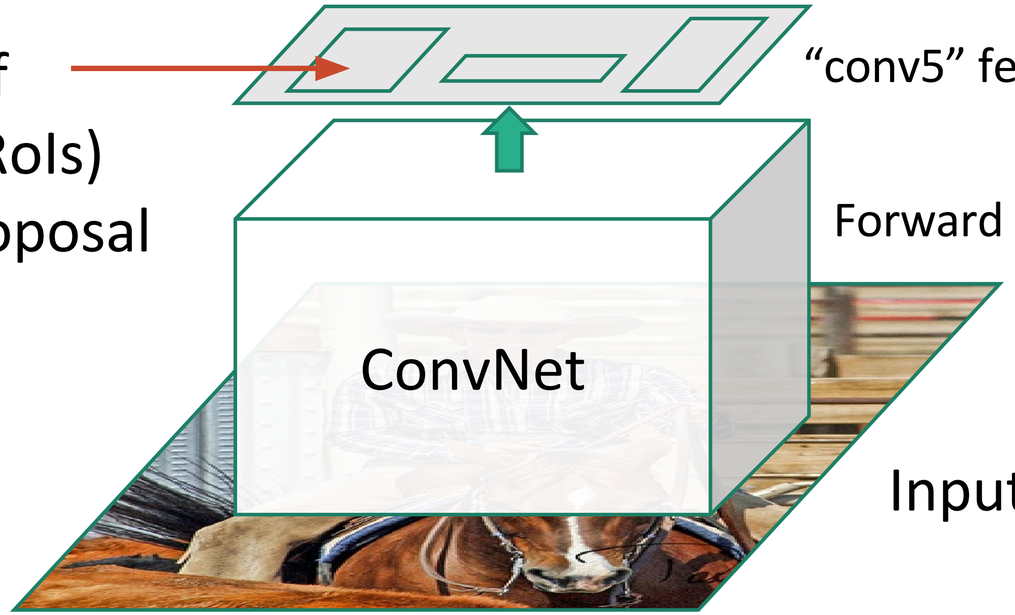
Girshick, “Fast R-CNN”, ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN



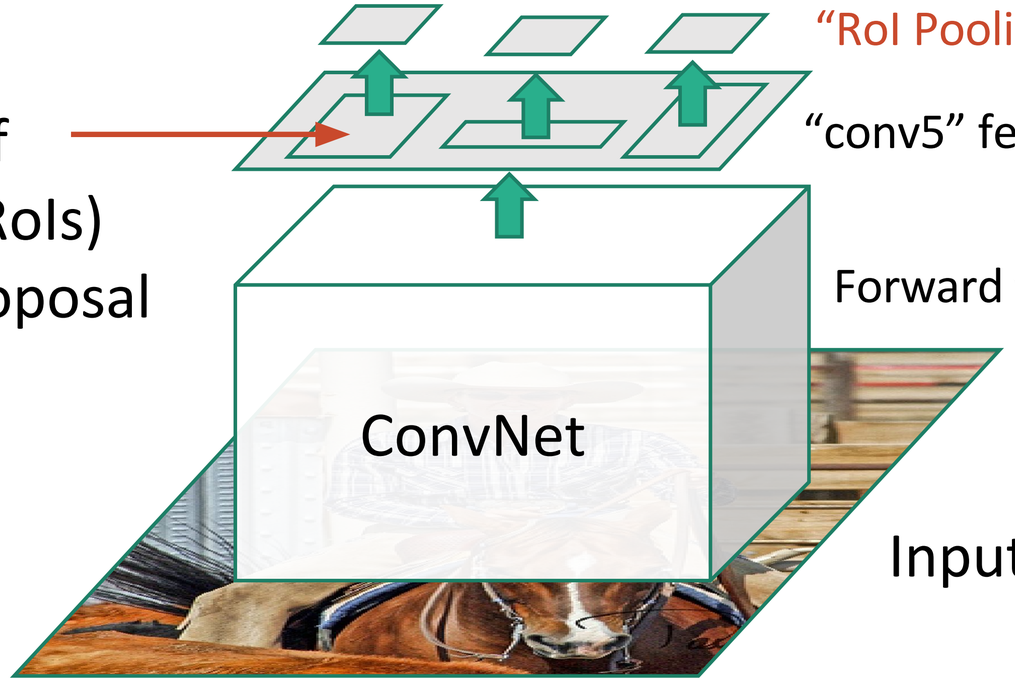
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN



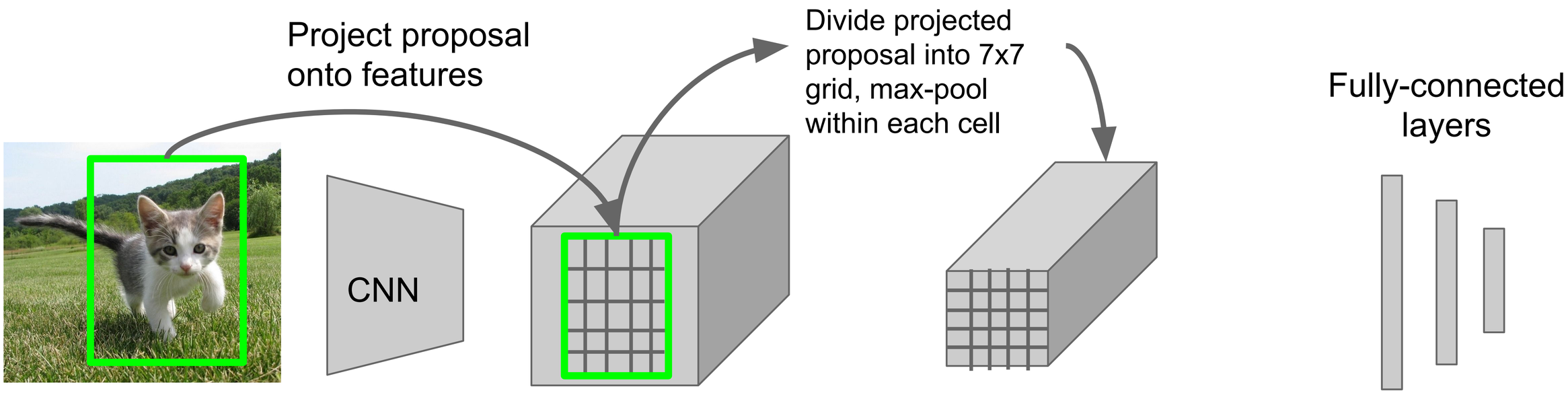
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN



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

Fast R-CNN: RoI Pooling



Hi-res input image:
3 x 640 x 480
with region
proposal

Hi-res conv features:
512 x 20 x 15;

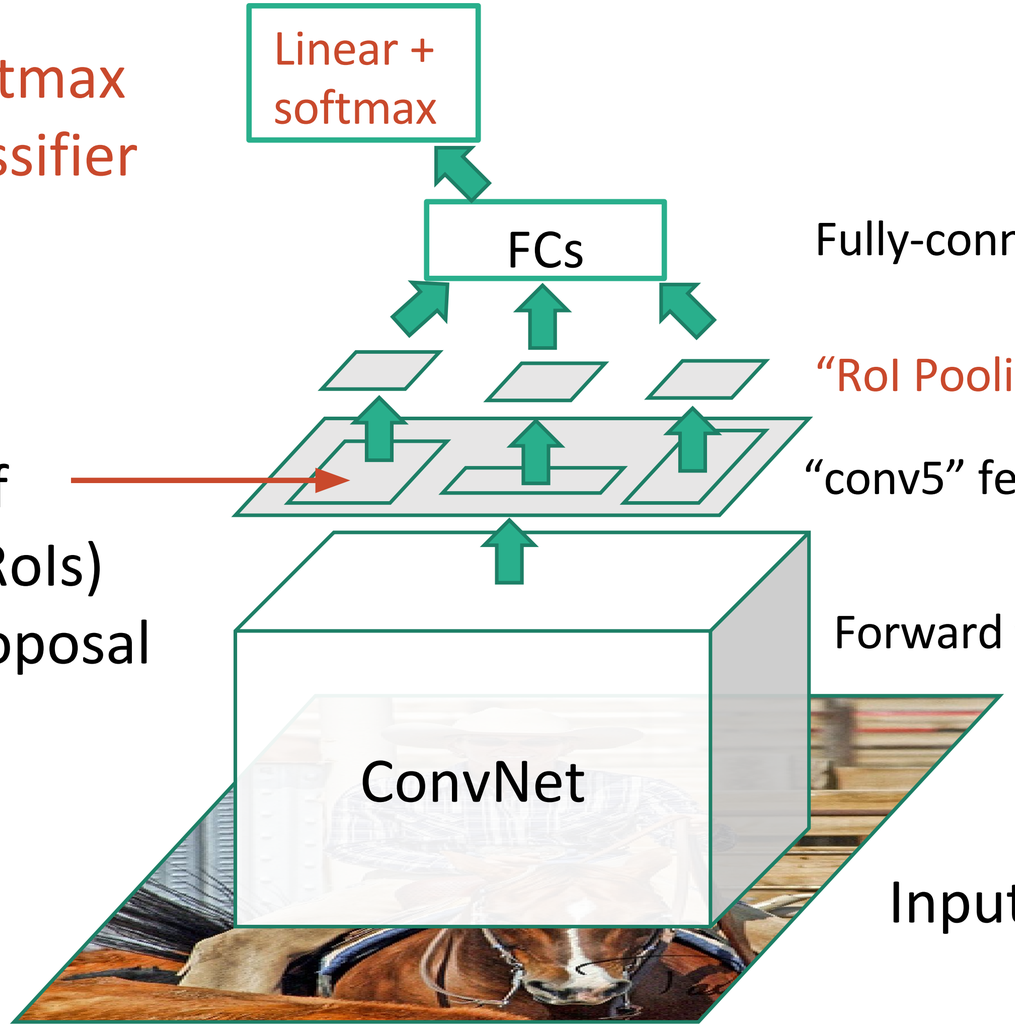
Projected region
proposal is e.g.
512 x 18 x 8
(varies per proposal)

RoI conv features:
512 x 7 x 7
for region proposal

Fully-connected layers expect
low-res conv features:
512 x 7 x 7

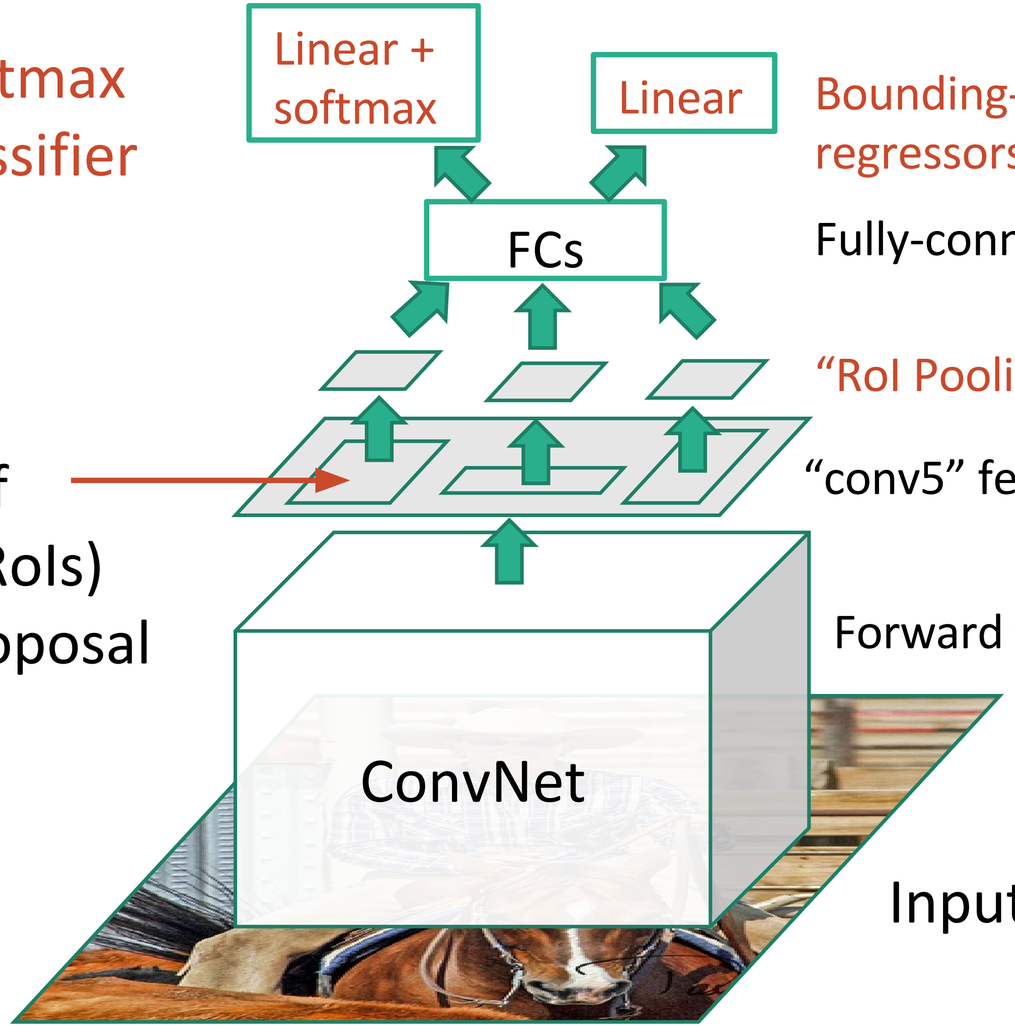
Girshick, “Fast R-CNN”, ICCV 2015.

Fast R-CNN



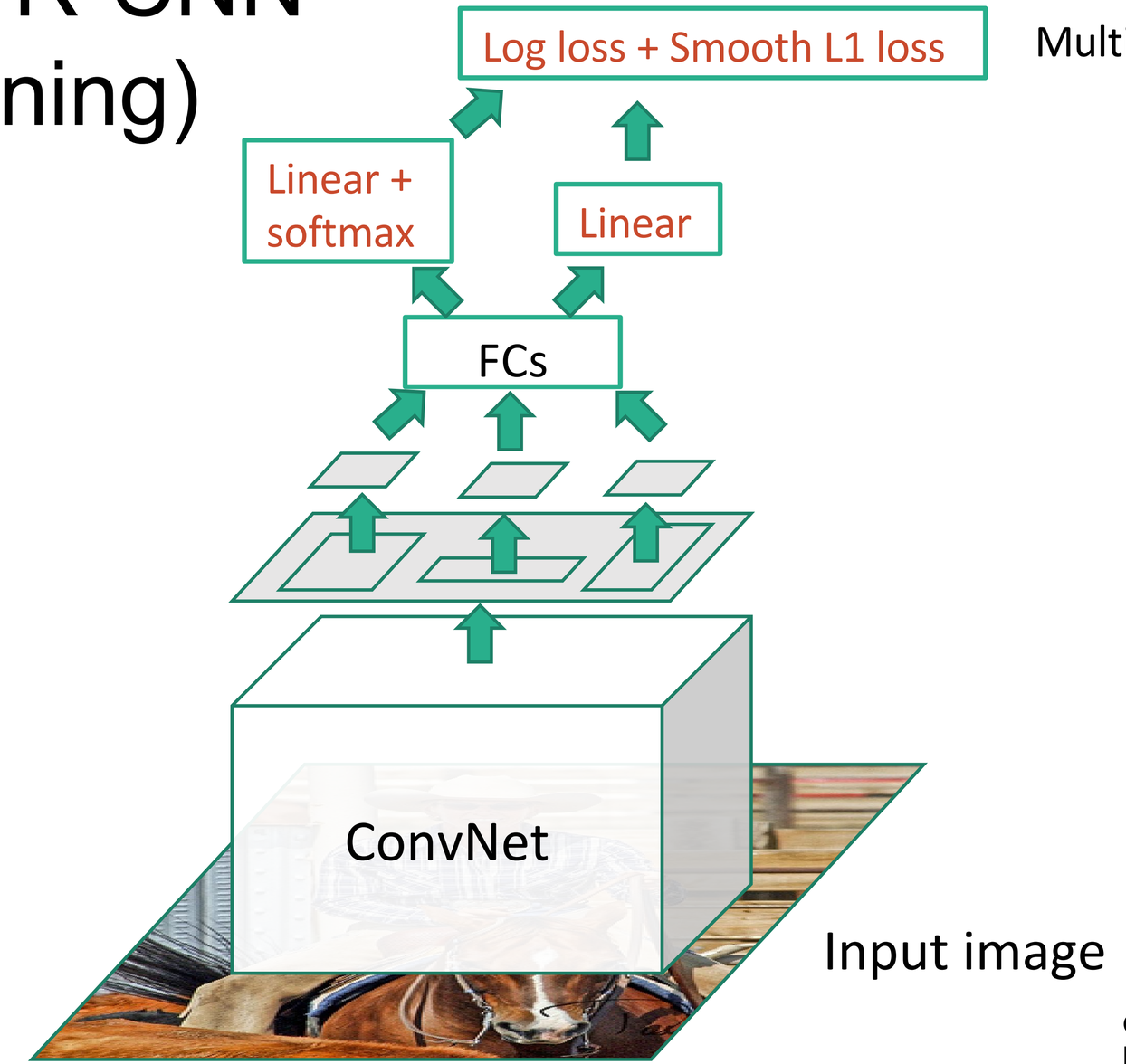
Girshick, “Fast R-CNN”, ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN



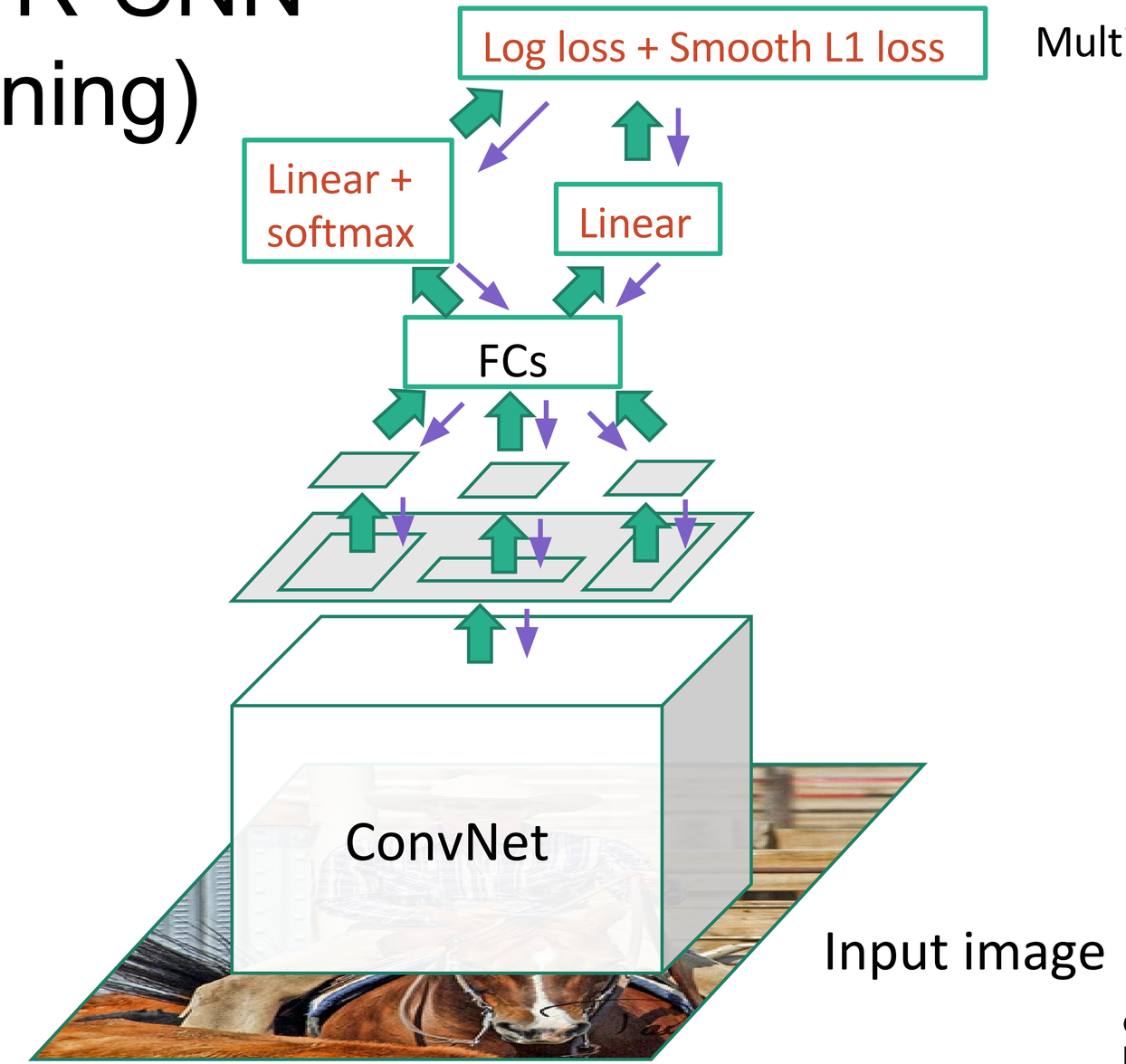
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN (Training)



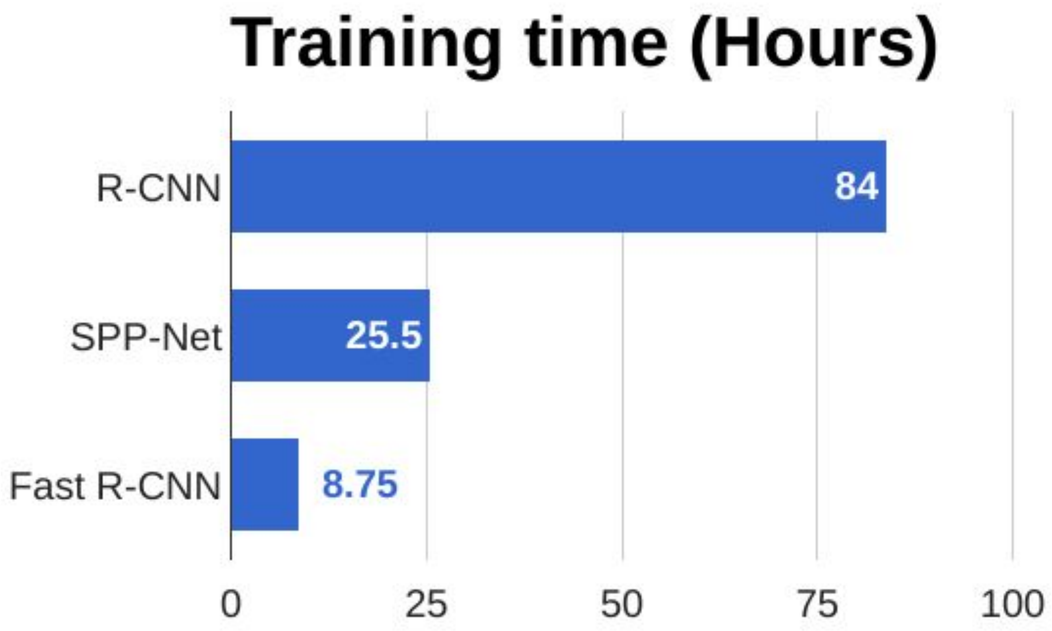
Girshick, “Fast R-CNN”, ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

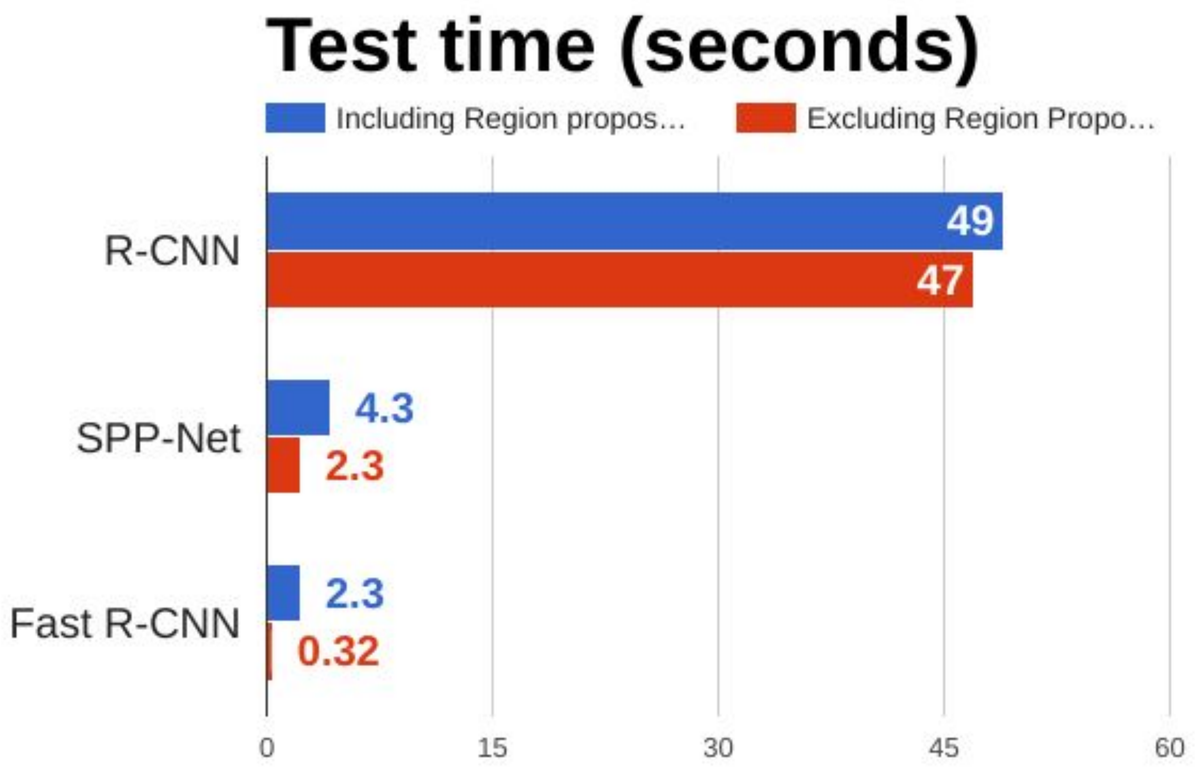
Fast R-CNN (Training)



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

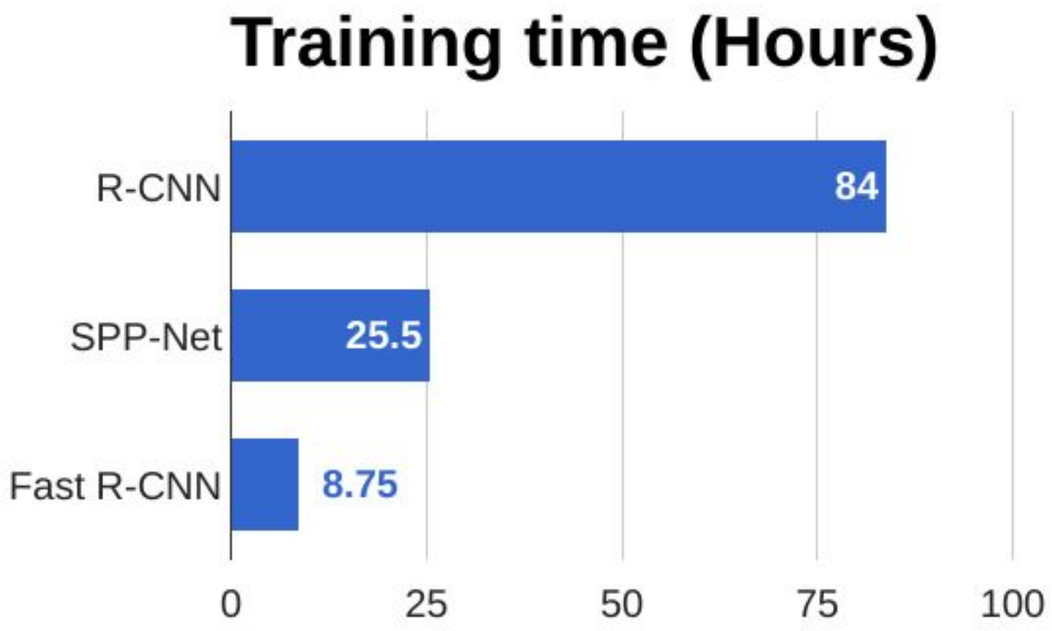
R-CNN vs SPP vs Fast R-CNN

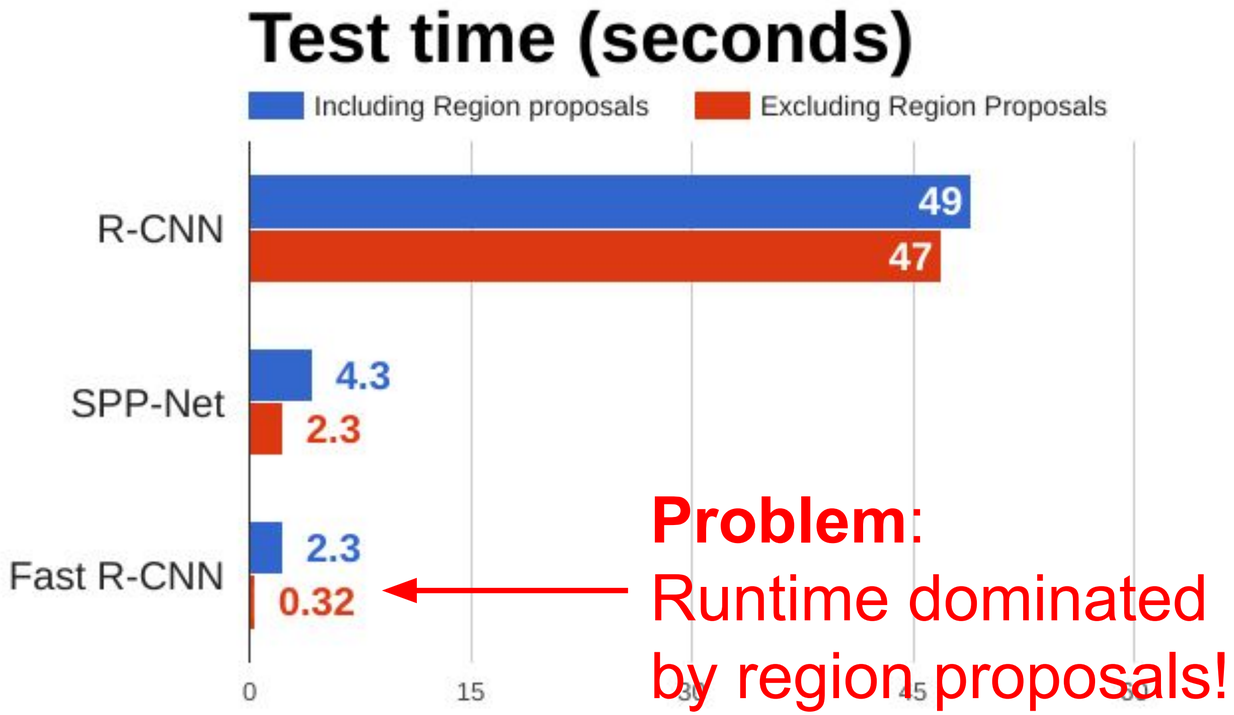




Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014
Girshick, “Fast R-CNN”, ICCV 2015

R-CNN vs SPP vs Fast R-CNN





Problem:
Runtime dominated by region proposals!

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

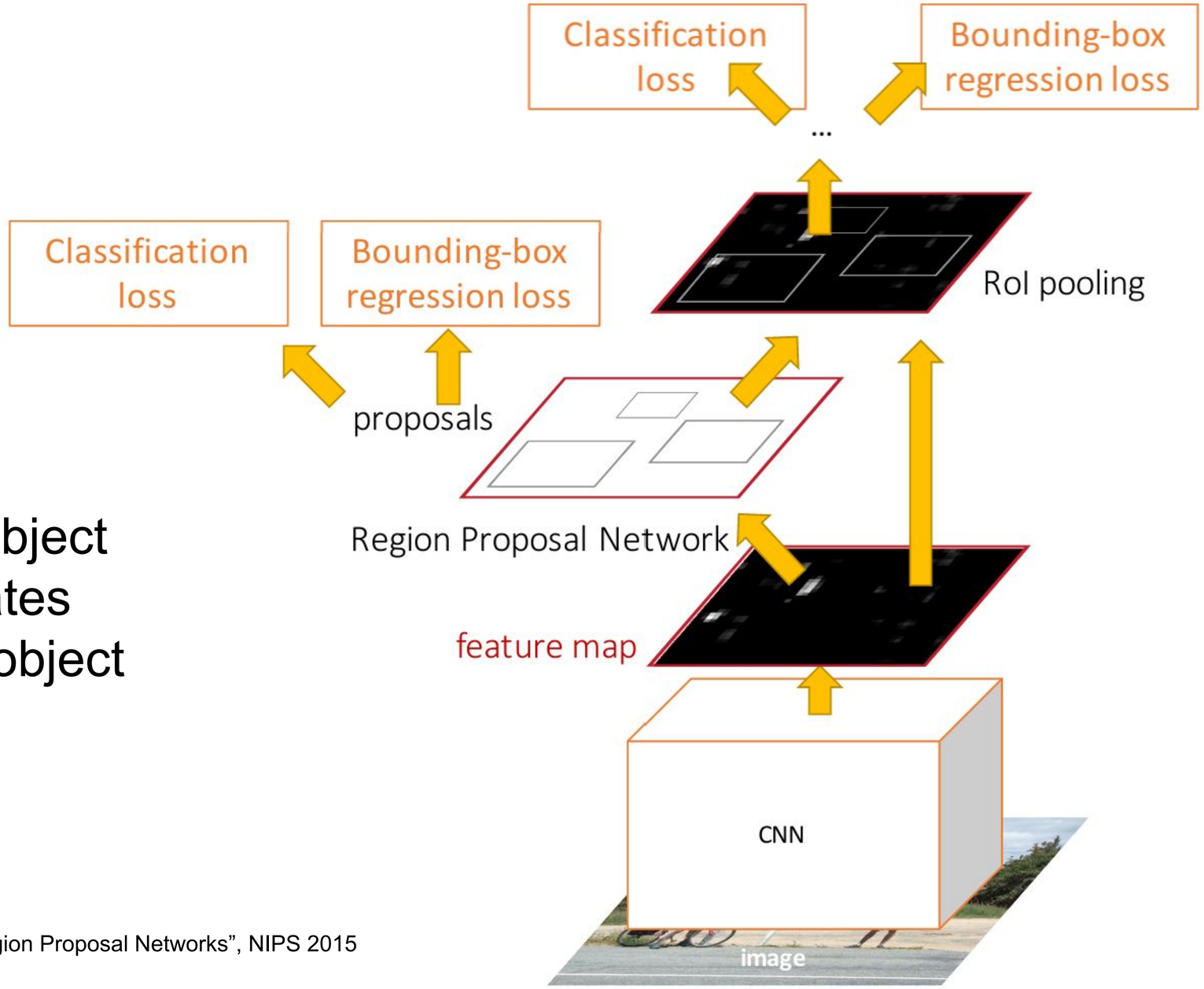
Faster R-CNN:

Make CNN do proposals!

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

Jointly train with 4 losses:

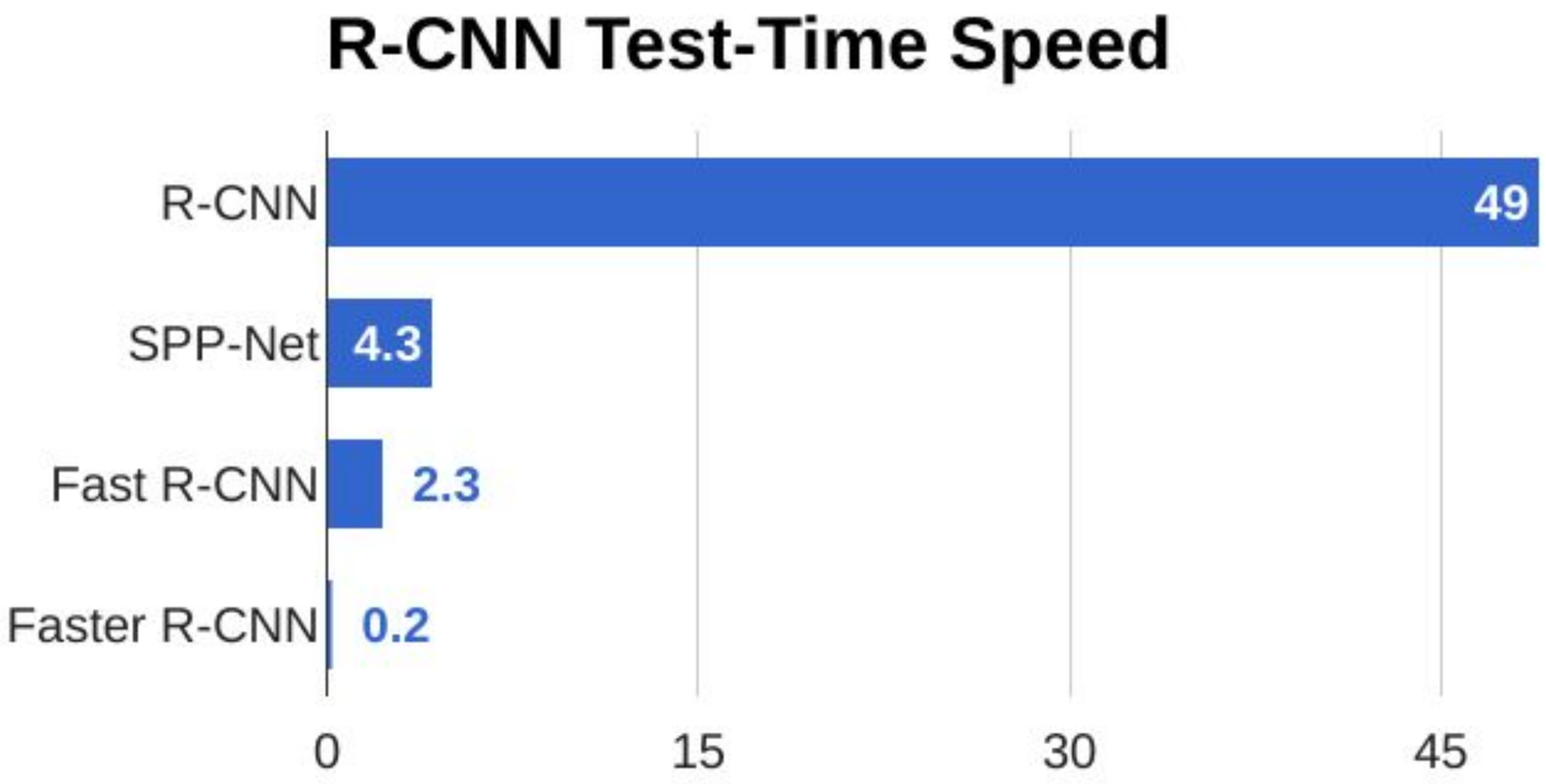
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



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

Fast**er** R-CNN:

Make CNN do proposals!



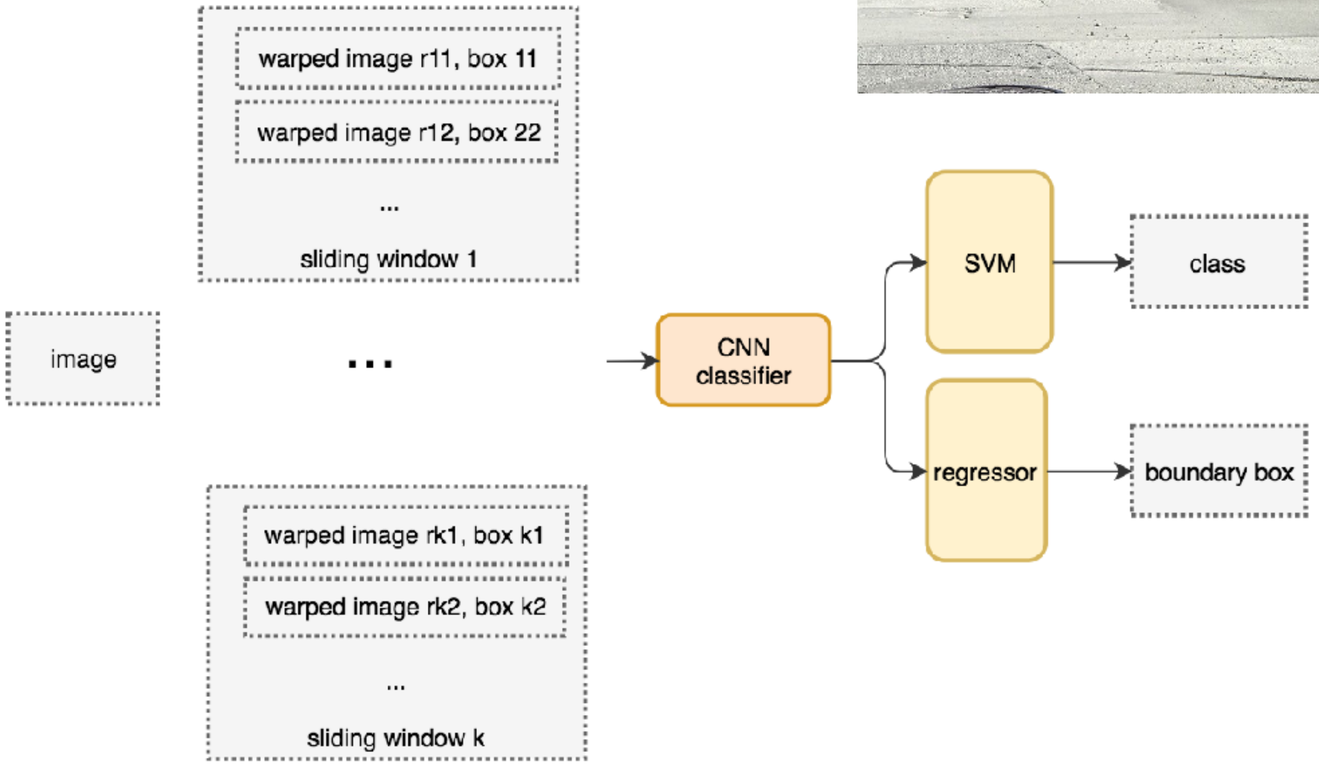
Alternative look 2 at Region-based CNNs:

Object Detection Region-based CNNs

Summary: R-CNN, Fast, Faster

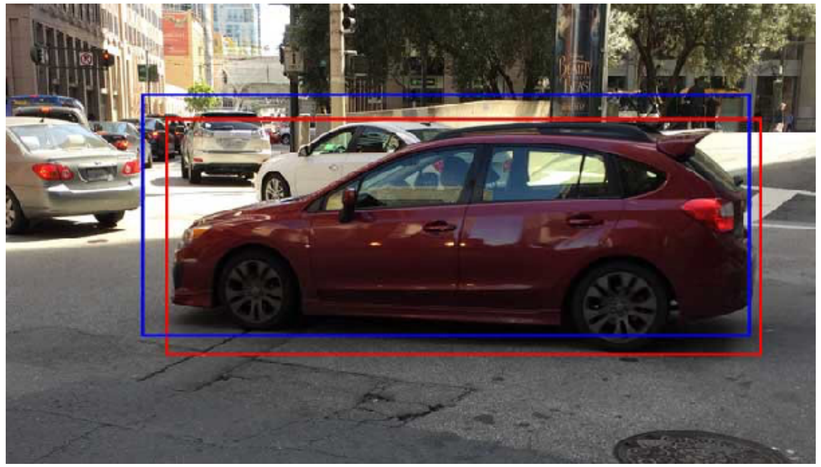
R-FCN

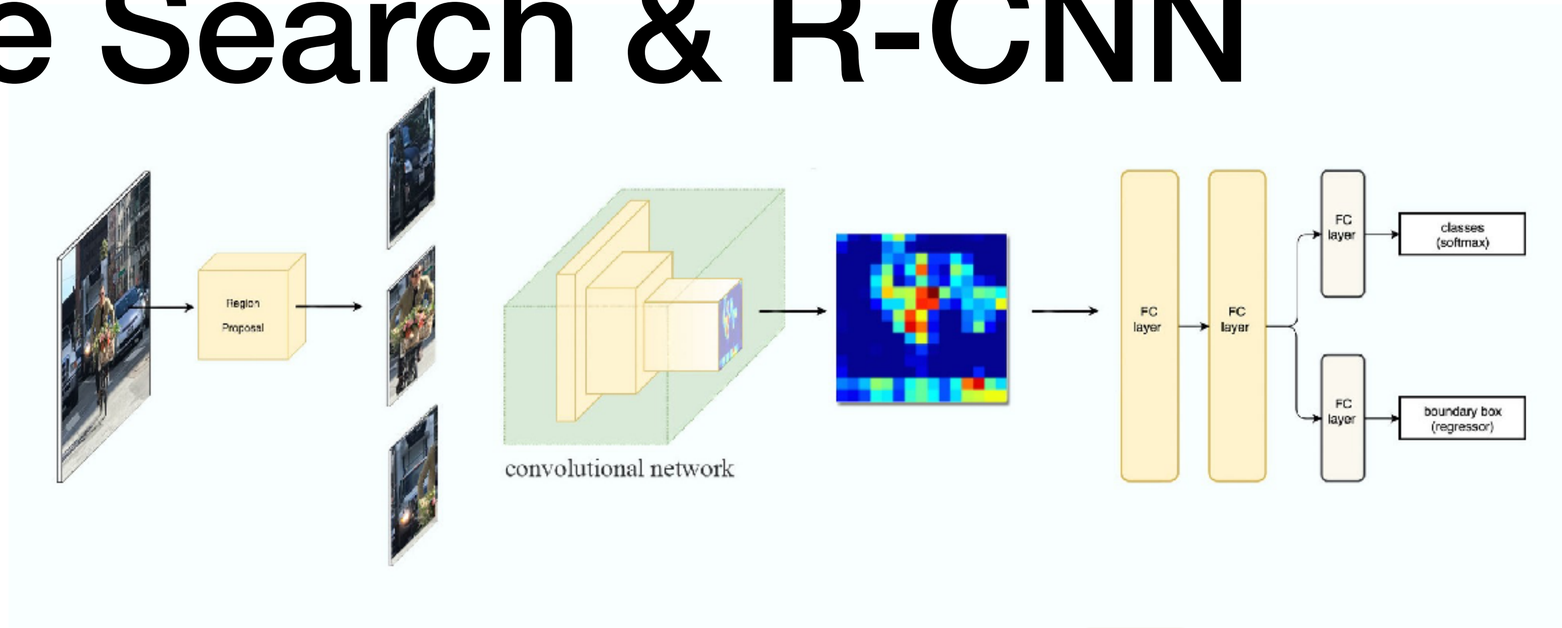
Sliding-window detectors

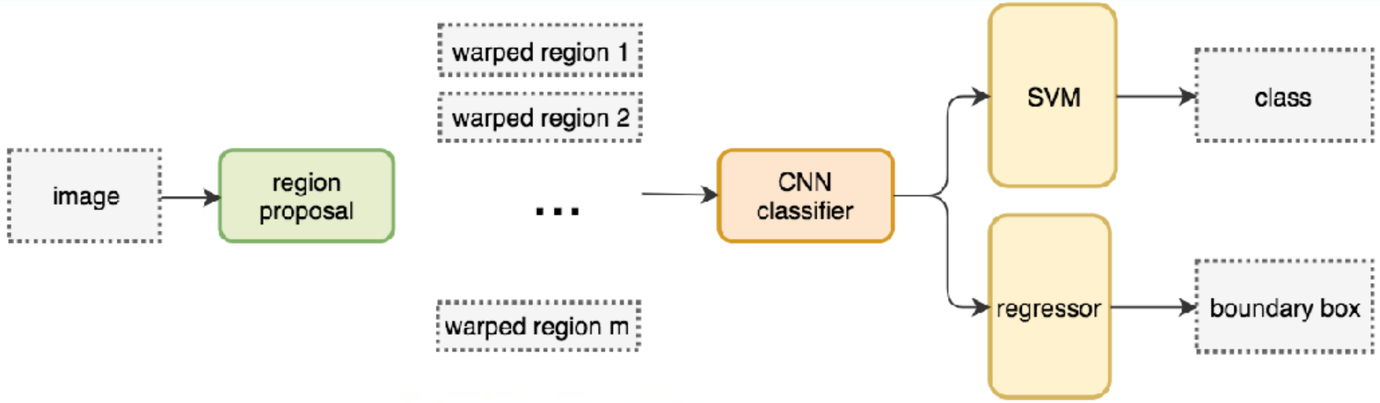


```
for window in windows
    patch = get_patch(image, window)
    results = detector(patch)
```

Selective Search & R-CNN

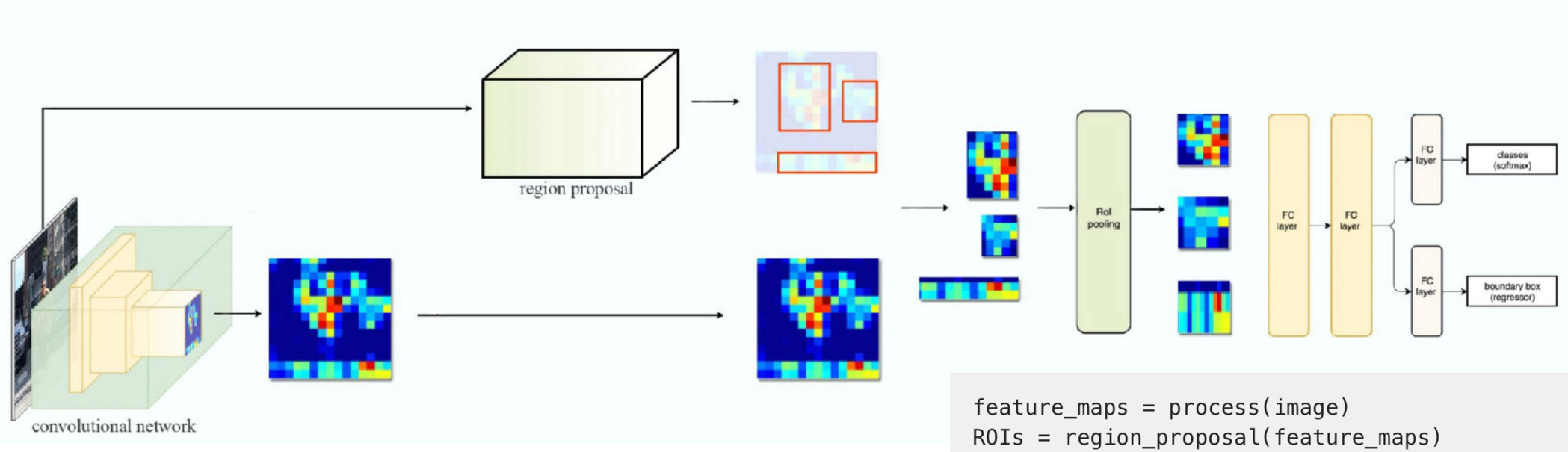




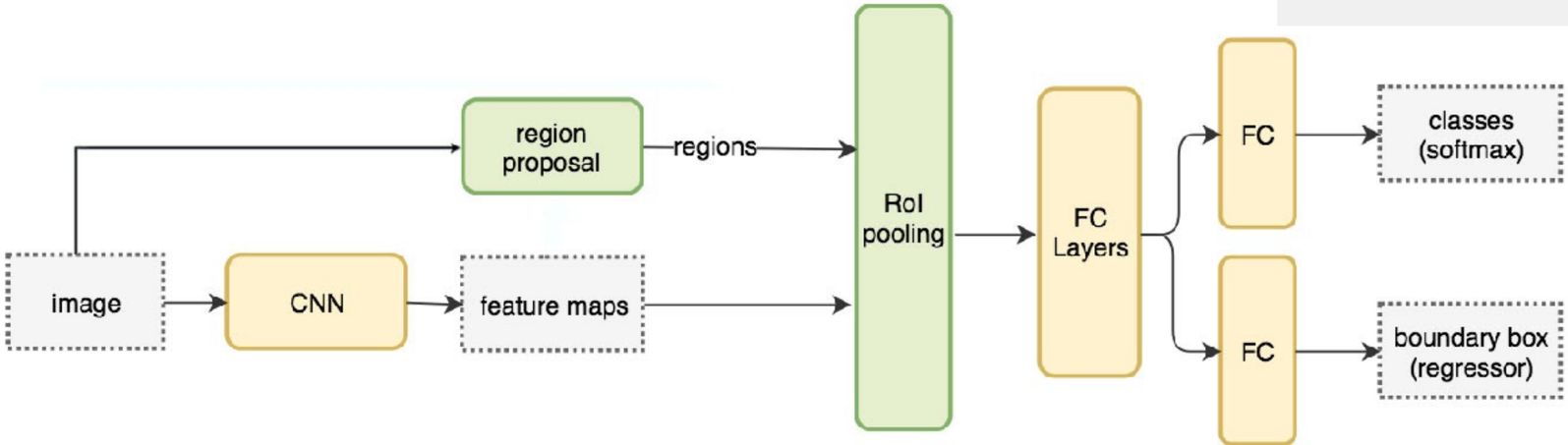


```
ROIs = region_proposal(image)
for ROI in ROIs
    patch = get_patch(image, ROI)
    results = detector(patch)
```

Fast R-CNN




```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
for ROI in ROIs
    patch = roi_align(feature_maps, ROI)
    results = detector2(patch)    # Reduce the amount of work here!
```



Fast R-CNN: ROI Pooling

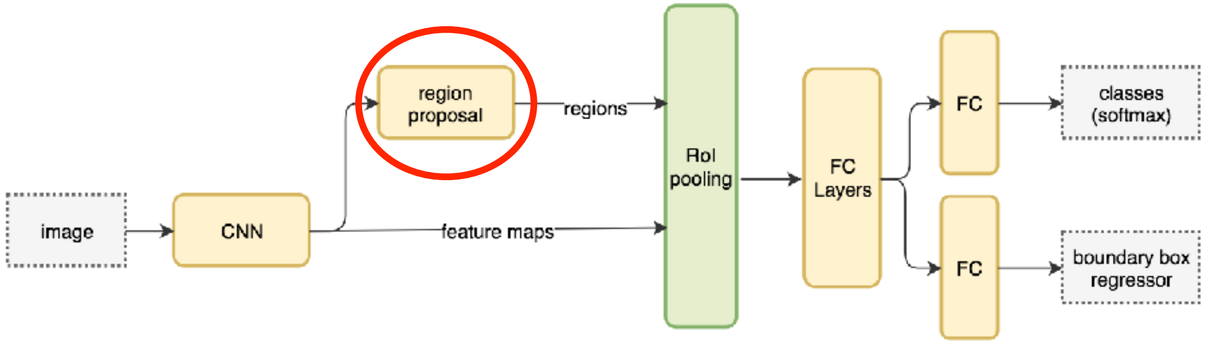
0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

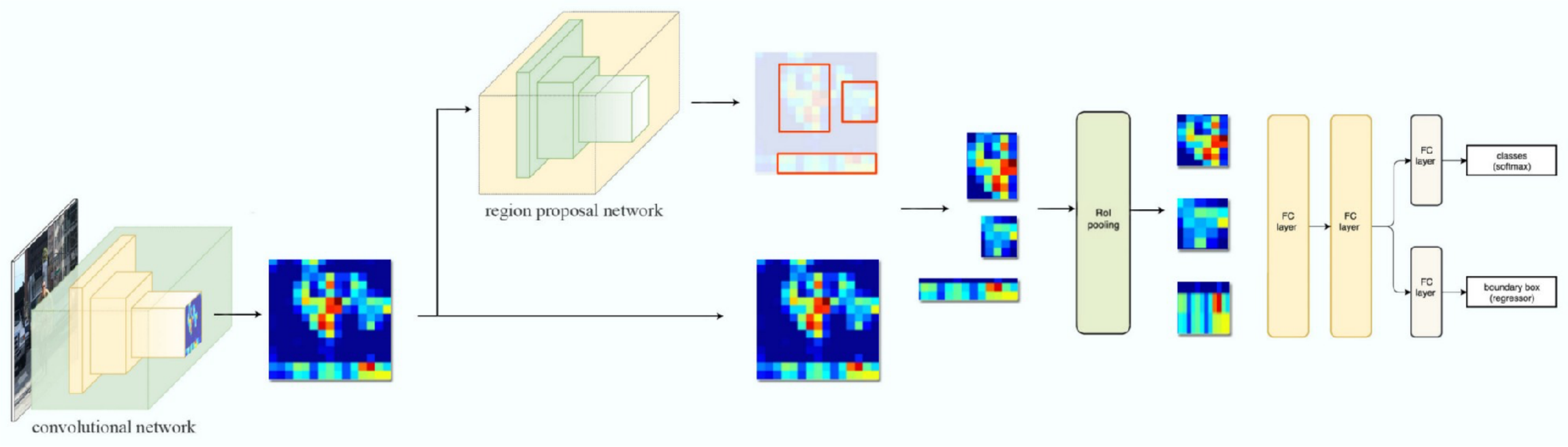
0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

0.8	0.6
0.9	0.6

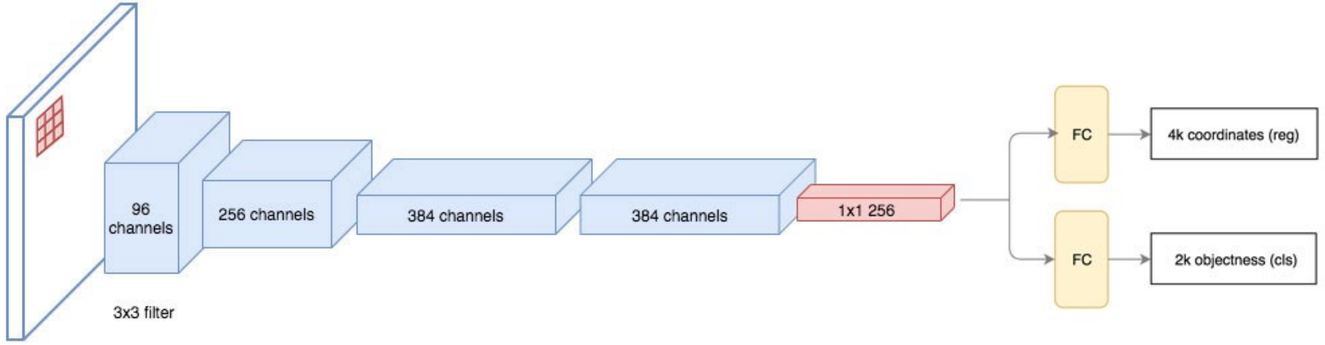
Faster R-CNN

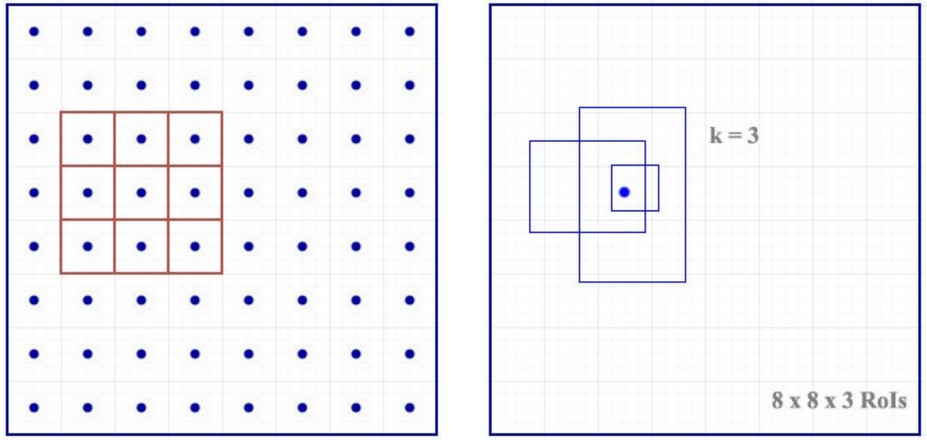



```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
for ROI in ROIs
    patch = roi_align(feature_maps, ROI)
    results = detector2(patch)    # Reduce the amount of work here!
```



Faster R-CNN: RPN

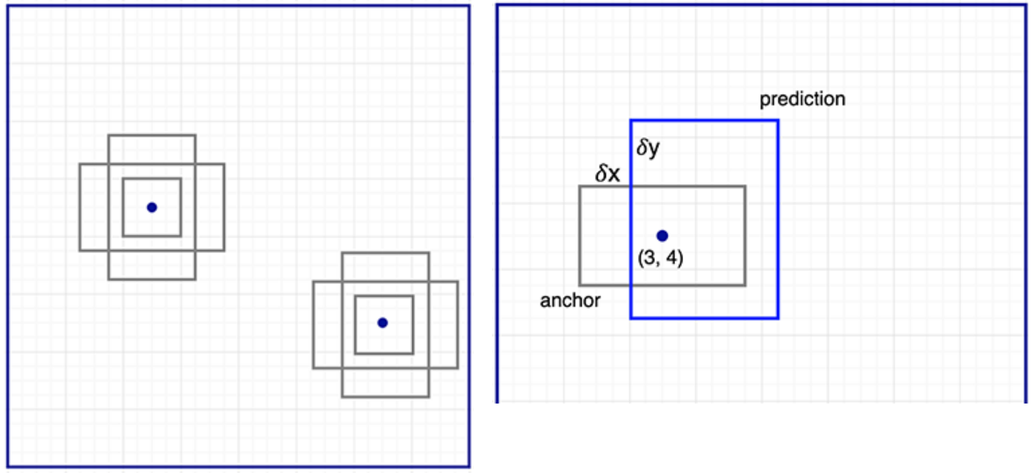


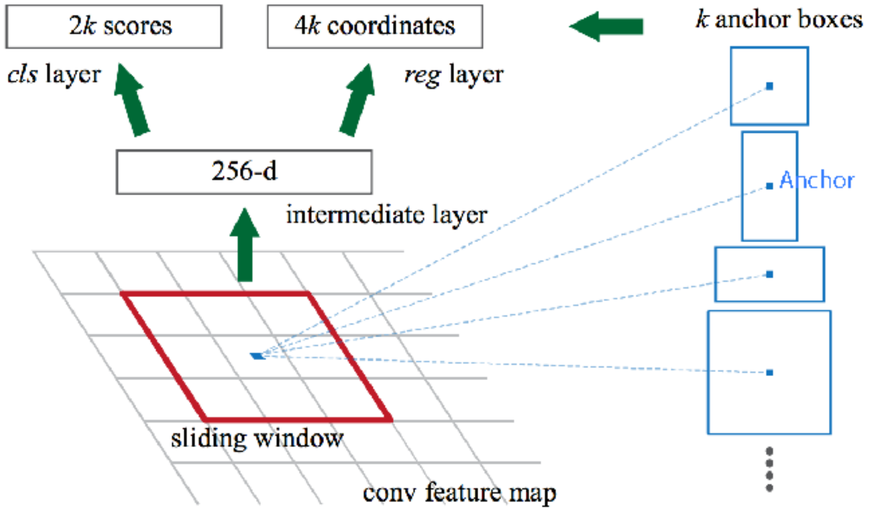


8 x 8 feature maps

3 proposals for each location

9 anchor (3 scales, 3 aspect ratios) ->
generates 2×9 objectness scores and 4×9 coordinates per location



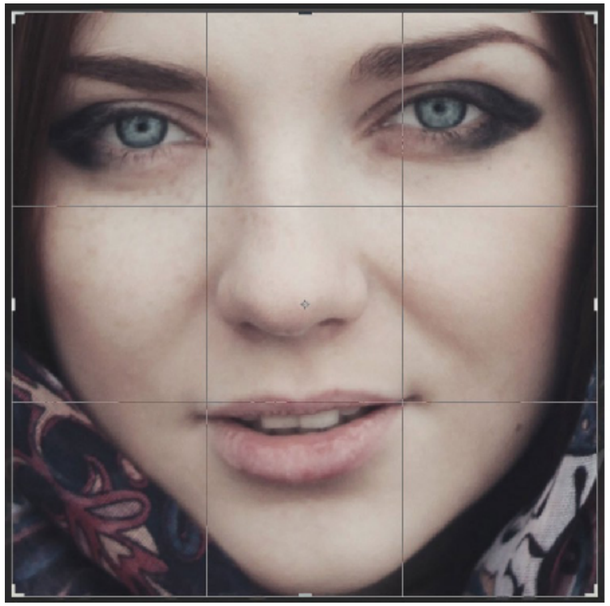


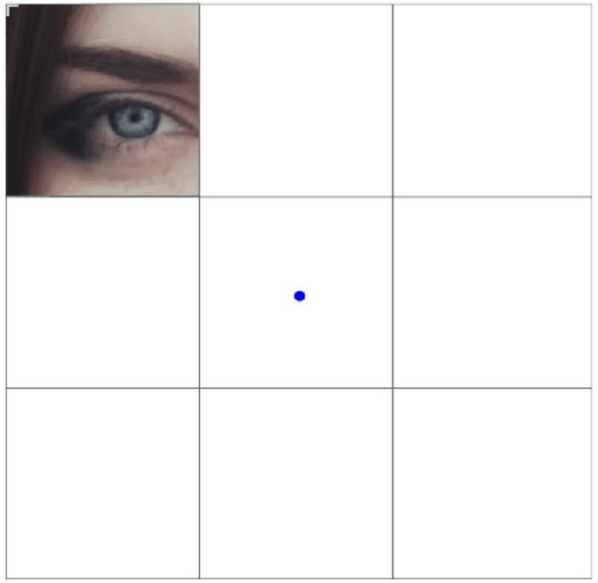
R-FCN

Region-based Fully Convolutional Networks

- **Intuition**: assume we have a feature map detecting the **right eye** of a face. Can we use it to locate a **face**?

- If we have other feature maps specialized in detecting the **left eye**, the **nose** or the **mouth**, we can **combine** the results to locate the face better.





R-FCN (2)

- In Faster R-CNN, the detector applies multiple **fully connected** layers to make predictions. With many (2,000) ROIs, it can be **expensive**.

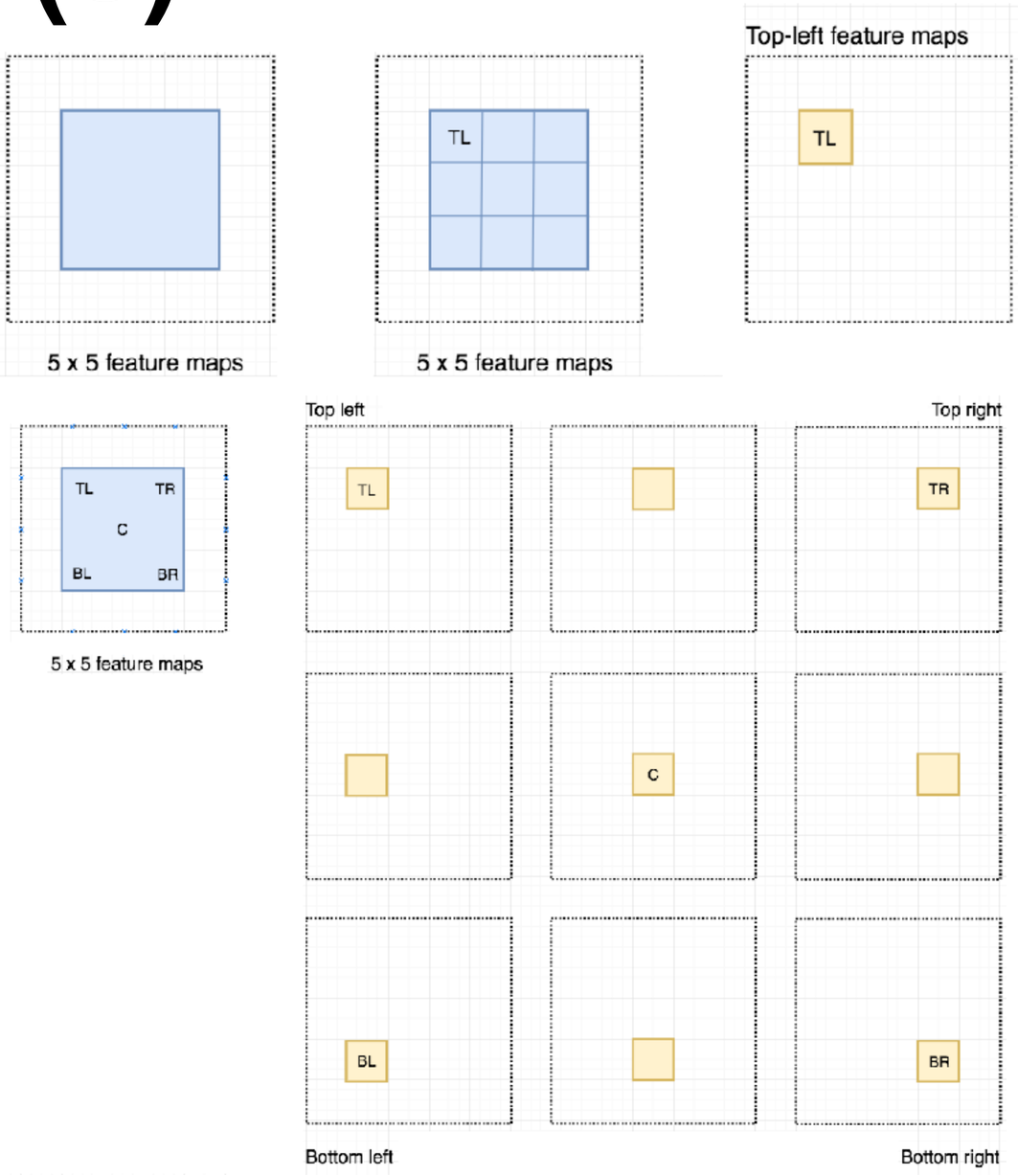
- R-FCN improves speed by **reducing** the amount of work **needed** for each **ROI**. The region-based feature maps (right eye) are **independent** of ROIs and can be computed outside each ROI. The remaining **work** is much **simpler** and therefore R-FCN is **faster** than Faster R-CNN.

```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
for ROI in ROIs
    patch = roi_pooling(feature_maps, ROI)
    class_scores, box = detector(patch)         # Expensive!
    class_probabilities = softmax(class_scores)
```

```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
score_maps = compute_score_map(feature_maps)
for ROI in ROIs
    V = region_roi_pool(score_maps, ROI)
    class_scores, box = average(V)                  # Much simpler!
    class_probabilities = softmax(class_scores)
```

R-FCN (3)

- Consider a 5×5 feature map with a blue square object inside.
- Divide the square object equally into 3×3 regions.
- Create a new feature map from the first to detect the top left (TL) corner of the square only.
- Since we divide the square into 9 parts, we can **create 9 feature maps each detecting the corresponding region of the object.** These feature maps are called **position-sensitive score maps** because each map detects (scores) a sub-region of the object. By combing the **votes** from these feature maps, we determine the class and the location of the objects.



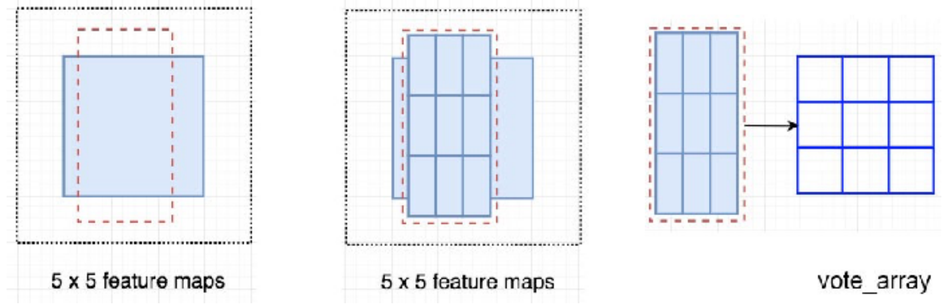
R-FCN (4)

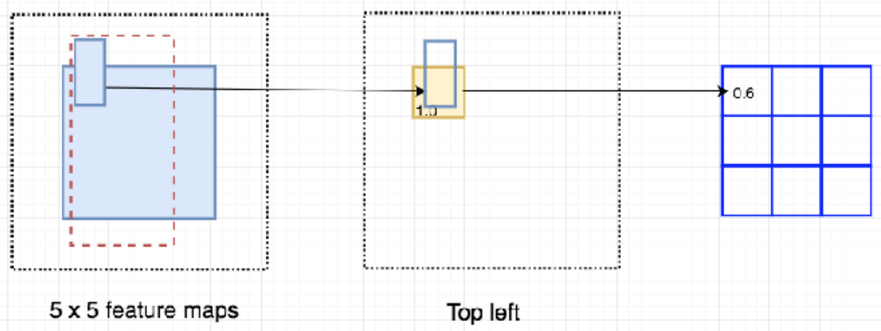
- Assume that the dotted **red** rectangle below is the **ROI proposed.**

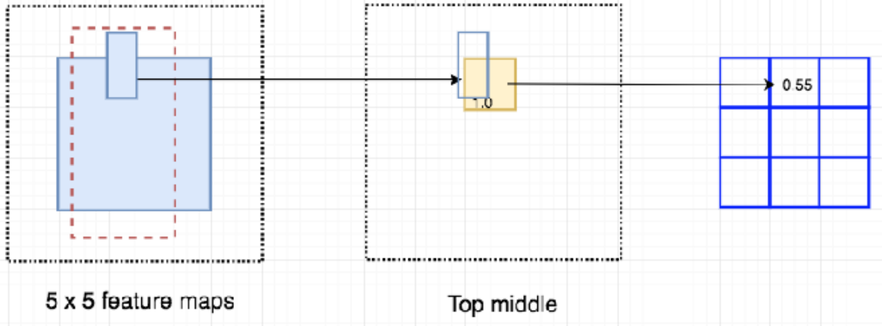
- We divide it into 3×3 **regions** and ask how likely each region **contains the corresponding part of the object.** For example, how likely the top-left ROI region contains the right eye.

- We store the results into a 3×3 **vote array.** For example, vote_array[0][0] contains the score on whether we find the top-left region of the square object.

- **Position-sensitive ROI-pool: Overlay** a portion of the ROI onto the corresponding score map to generate the vote array.

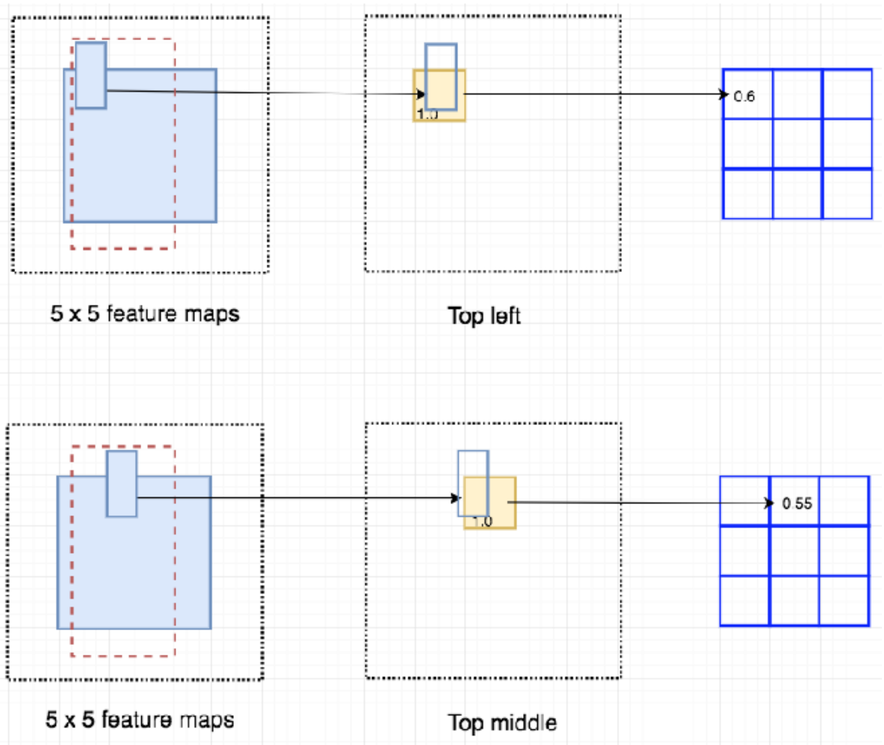




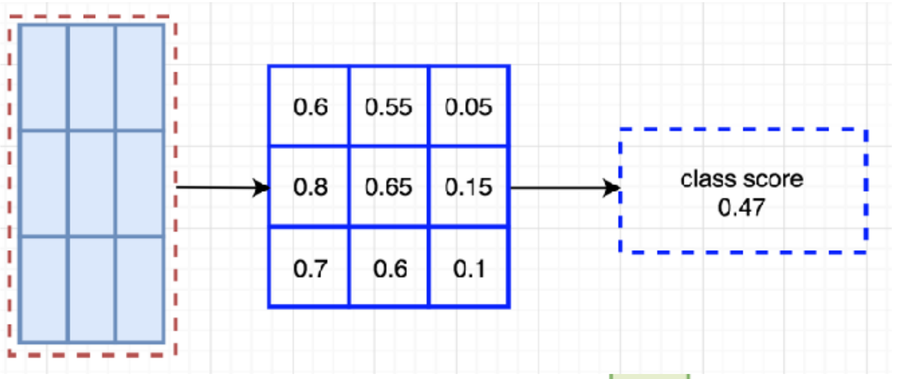


R-FCN (5)

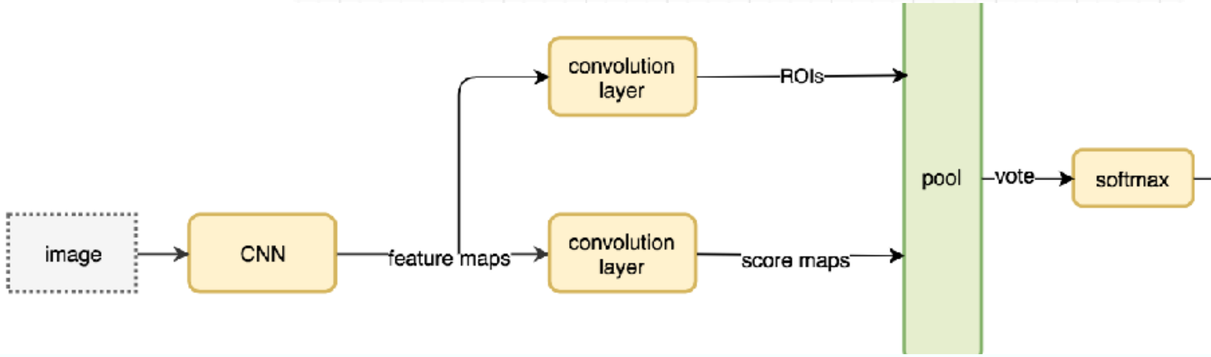
- **Position-sensitive ROI-pool** (some more details): map **ROIs** to (position-sensitive) **score maps** and generate the **vote array**
  - **take** top-left ROI region, **map to** top-left score map, **compute** the average score of the top-left score map bounded by the top-left ROI (blue rectangle): About 40% of the area inside the blue rectangle has 0 activation and 60% have 100% activation, i.e. 0.6 in average. So the **likelihood** that we have detected the top-left object is 0.6.
  - We store the result (0.6) into the vote array



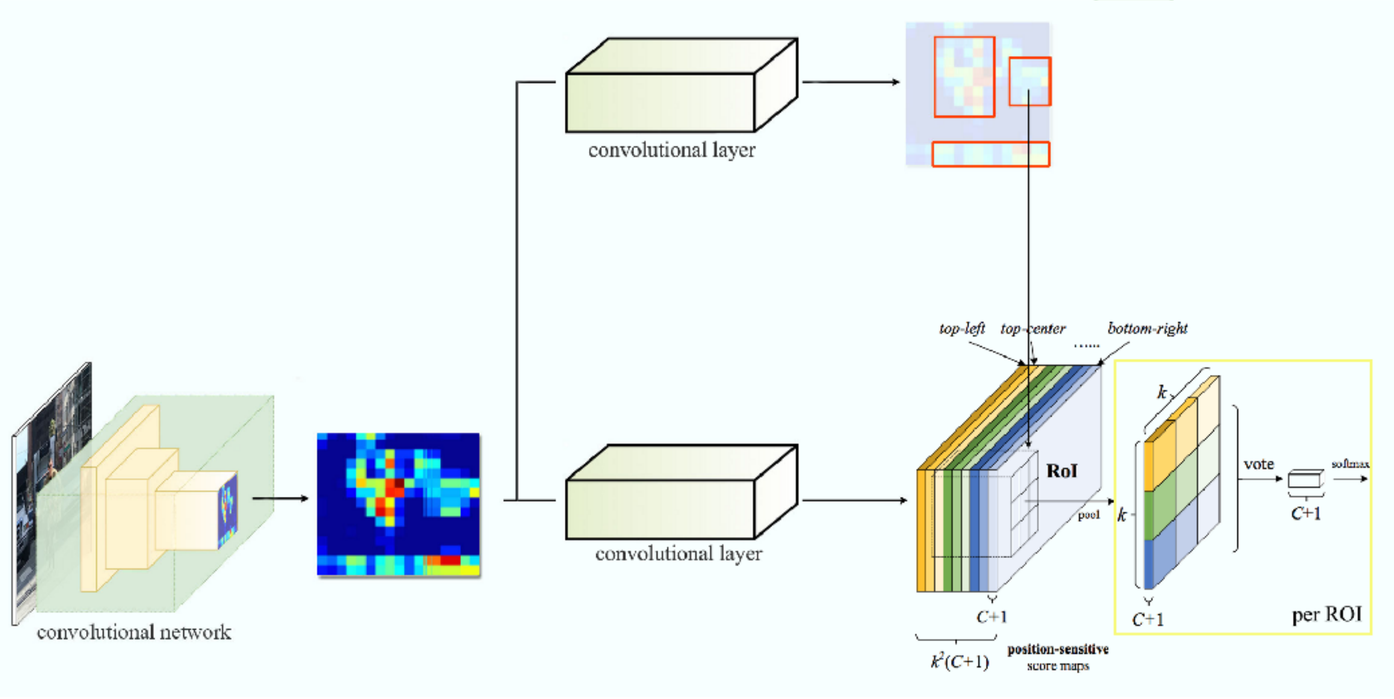
R-FCN (6)



- After calculating all the values for the position-sensitive ROI pool, the **class score** is the **average** of all its elements.

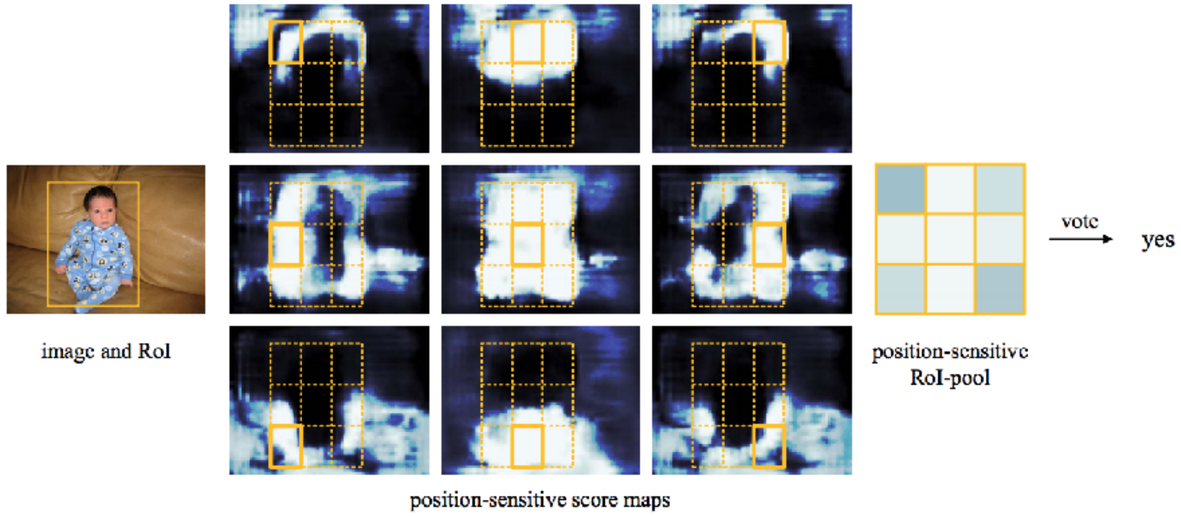


- Want to detect C classes (+1 for background or no class). Each class will have its own 3×3 score maps and therefore a total of 3×3 x (C+1) score maps. Using its own set of score maps, we predict a class score for each class. Then we apply a softmax on those scores to compute the probability for each class (k=3 here)

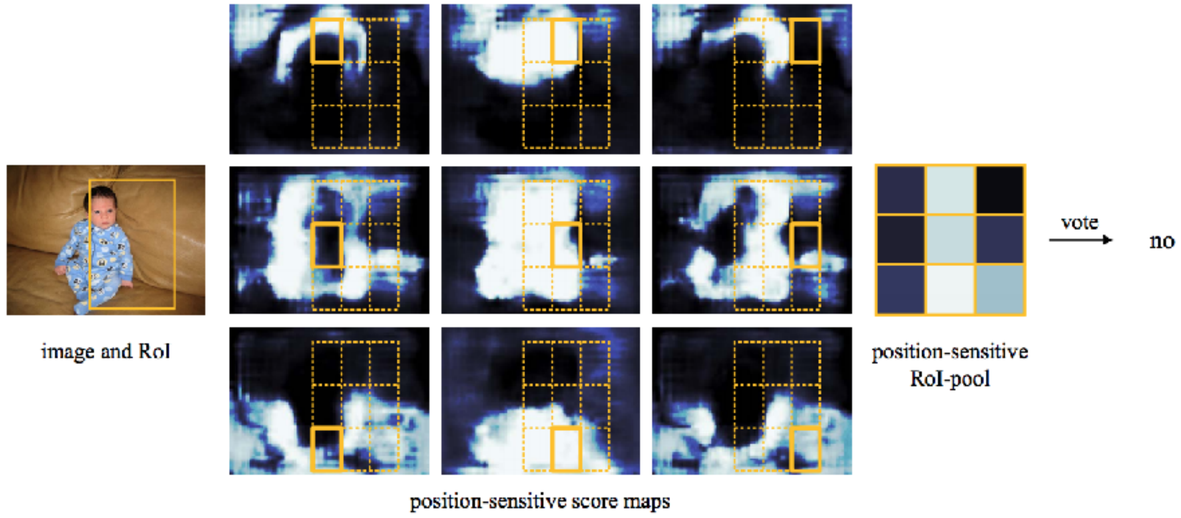


R-FCN (7)

- **Example:**
  - We have **9 score maps** to detect the top-left to the bottom-right region of a baby.
  - **Top** diagram: the ROI **aligns** well with the ground truth. The solid **yellow** rectangle in the middle column indicates the ROI sub-region corresponding to the specific score map. **Activations** are high inside the solid yellow box for every score maps. Therefore the scores in the vote array are **high** and a baby is **detected**.
  - **Second** diagram: ROI is **misaligned**. The score maps are the **same** but the corresponding locations for the ROI sub-regions (solid yellow) are **shifted**. The overall **activations** are **low** and we will **not classify** this ROI contains a baby.



Visualization of R-FCN ($k \times k = 3 \times 3$) for the *person* category.



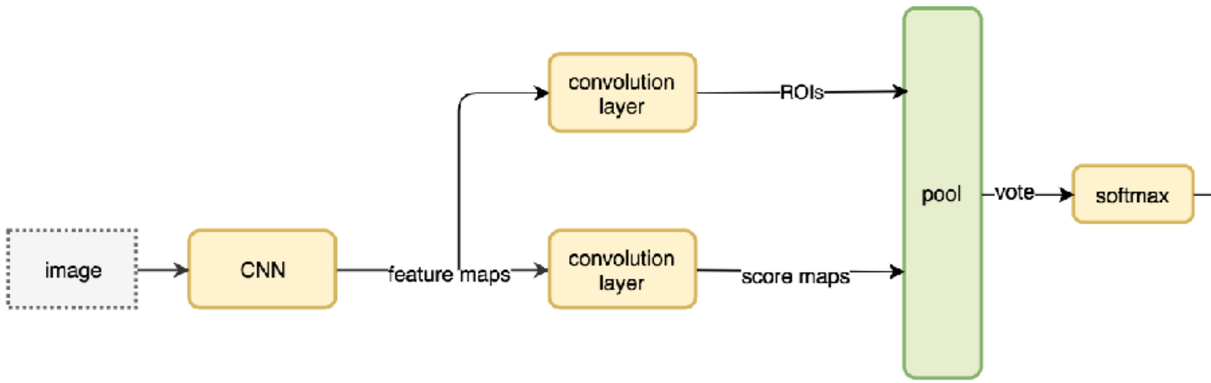
Visualization when an RoI does not correctly overlap the object.

R-FCN (8)

- **Classification**: convolution filters to create the k×k × (C+1) score maps from feature map

- **BB regression**: use additional convolution filters to create a k×k x 4 score map from the same feature maps. Apply the position-based ROI-pool to compute a k×k array with each element containing a boundary box. The final prediction is the average of those elements.

- **Results**: R-FCN 20x faster than the Faster R-CNN.



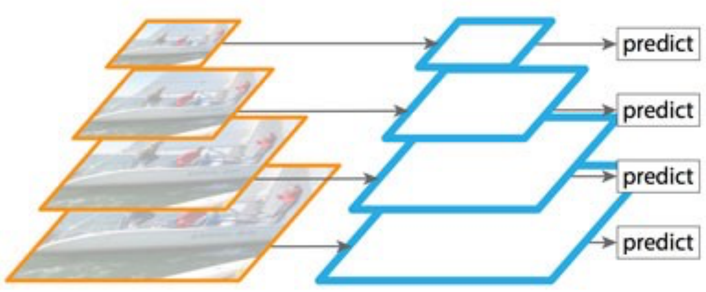
Comparisons on PASCAL VOC 2012 *test* set using **ResNet-101**. "07++12" [6] denotes the union set of 07 *trainval+test* and 12 *trainval*. †: `http://host.robots.ox.ac.uk:8080/anonymous/44L5HI.html` ‡: `http://host.robots.ox.ac.uk:8080/anonymous/MVCM2L.html`

	training data	mAP (%)	test time (sec/img)
Faster R-CNN [9]	07++12	73.8	0.42
Faster R-CNN +++ [9]	07++12+COCO	**83.8**	3.36
R-FCN multi-sc train	07++12	77.6†	0.17
R-FCN multi-sc train	07++12+COCO	**82.0**‡	0.17

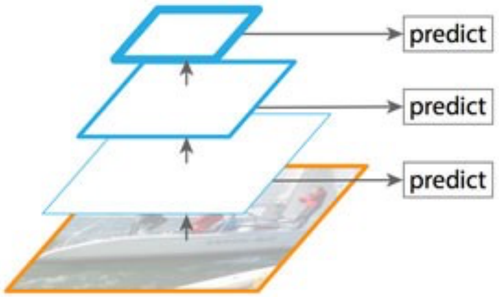
FPN

Feature Pyramid Networks

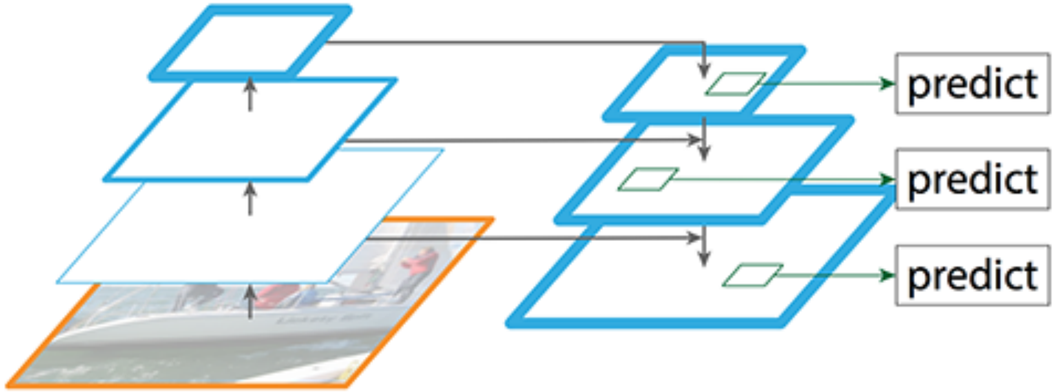
- Detecting objects at different **scales** is challenging in particular for **small** objects.
- Pyramid of images: pyramid of the **same image at different scale** to detect objects: computational demanding training: time and memory, just inference.
- Pyramid of feature maps: feature maps **closer** to the image layer consist of **low**-level structures that are not effective for accurate object detection.
- FPN: designed with accuracy and speed in mind. Generates **multi-scale feature maps** with better **quality** information that can **replace** the regular feature pyramid for object detection.



Pyramid of images



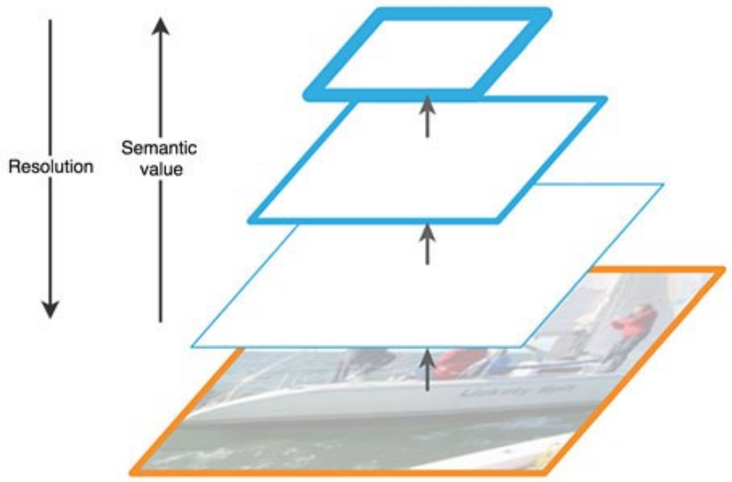
Pyramid of feature maps

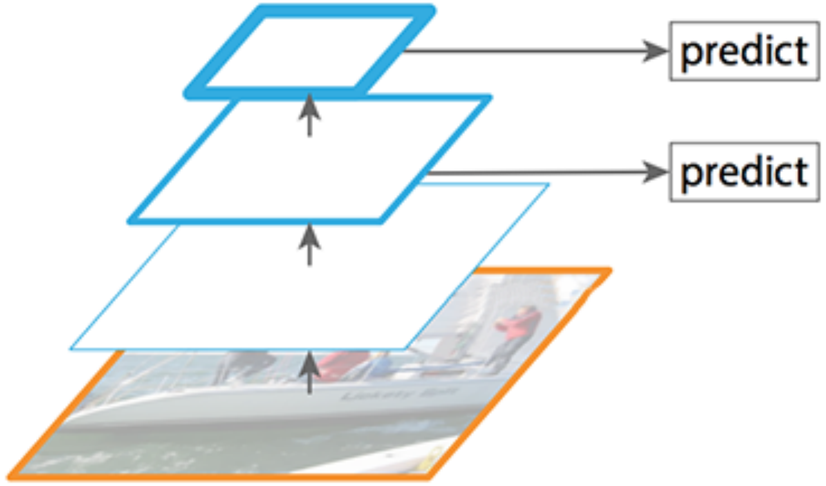


FPN

FPN (2)

- FPN consists of a **bottom-up** and a **top-down** pathway.
- Bottom-up pathway: usual convolutional network for feature extraction.
- As we go up, the spatial resolution decreases. With more high-level structures detected, the semantic value for each layer increases.
- Bottom layers are not used for object detection. They are in high **spatial** resolution but the **semantic** value is not high enough to **justify** its use as the speed slow-down is significant.



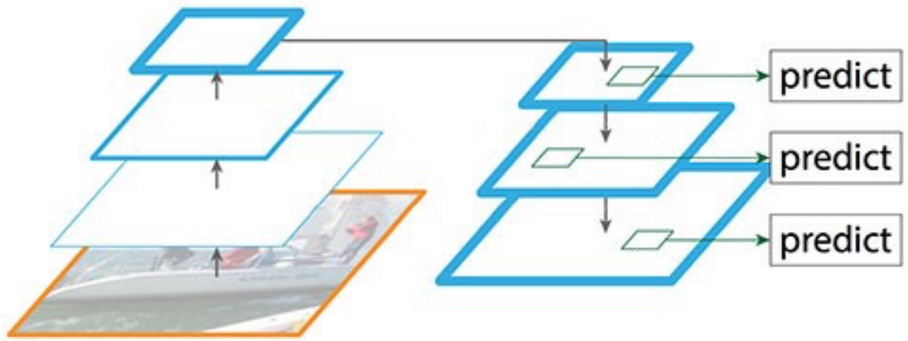


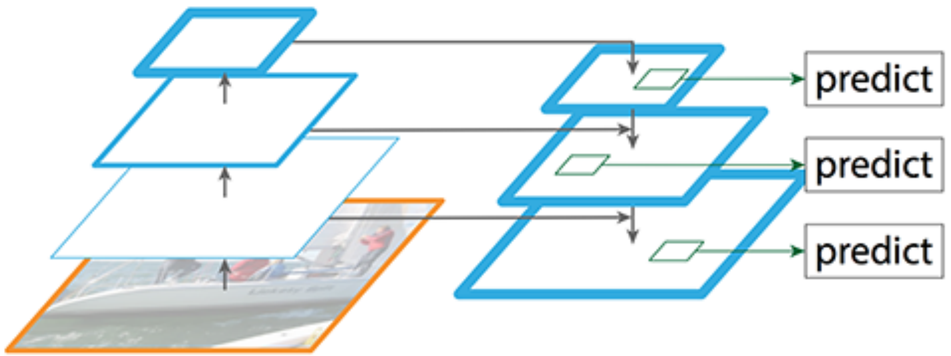
FPN (3)

- FPN provides a **top-down pathway** to construct higher resolution layers from a semantic rich layer.

- While the reconstructed layers are **semantic** strong the **locations** of objects are not precise after all the downsampling and upsampling.

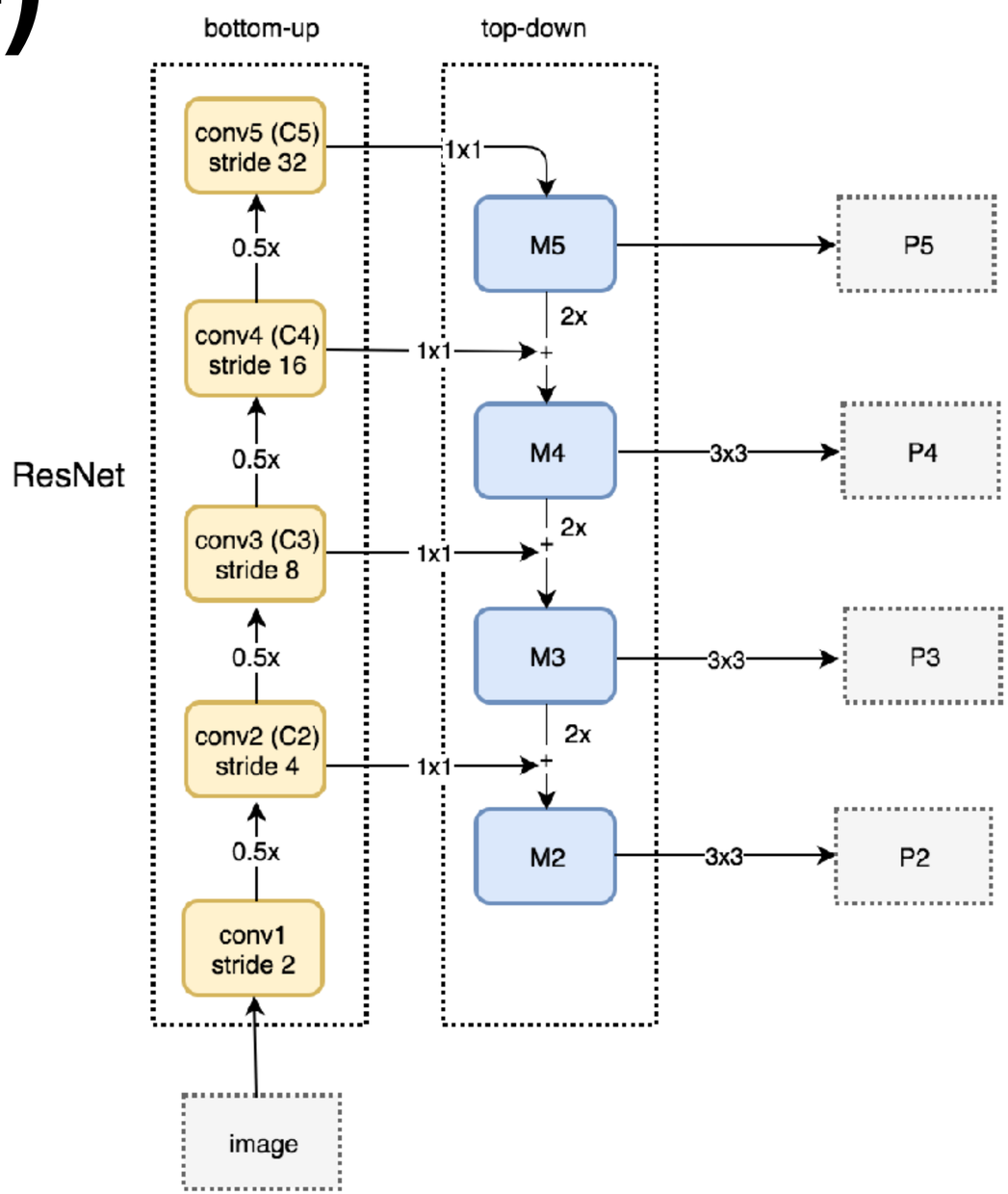
- We add **lateral connections** between reconstructed layers and the corresponding feature maps to help the detector to predict the **location** betters. It also acts as **skip** connections to make **training easier** (similar to what ResNet does).





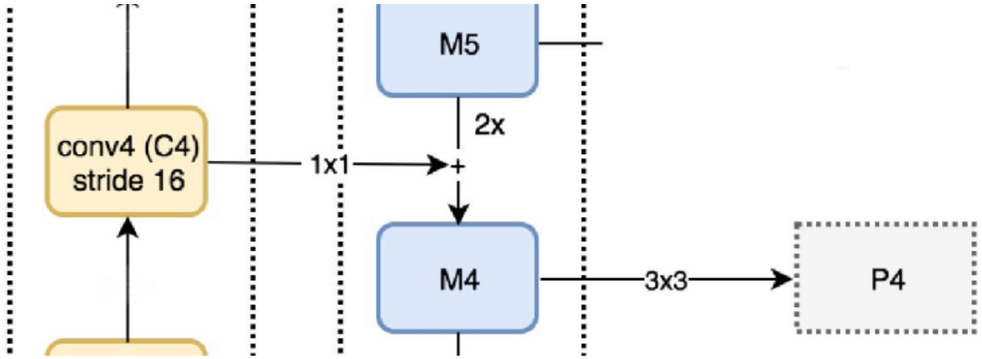
FPN (4)

- **Bottom-up pathway:**
  - Use **ResNet** to construct the bottom-up pathway.
  - It consists of many convolution **modules** (convi for i equals 1 to 5), each with many convolution **layers**.
  - As we move up, the **spatial** dimension is **reduced** by 1/2 (i.e. double the **stride**).
  - The output of each convolution module is labeled as Ci and later used in the top-down pathway.



FPN (5)

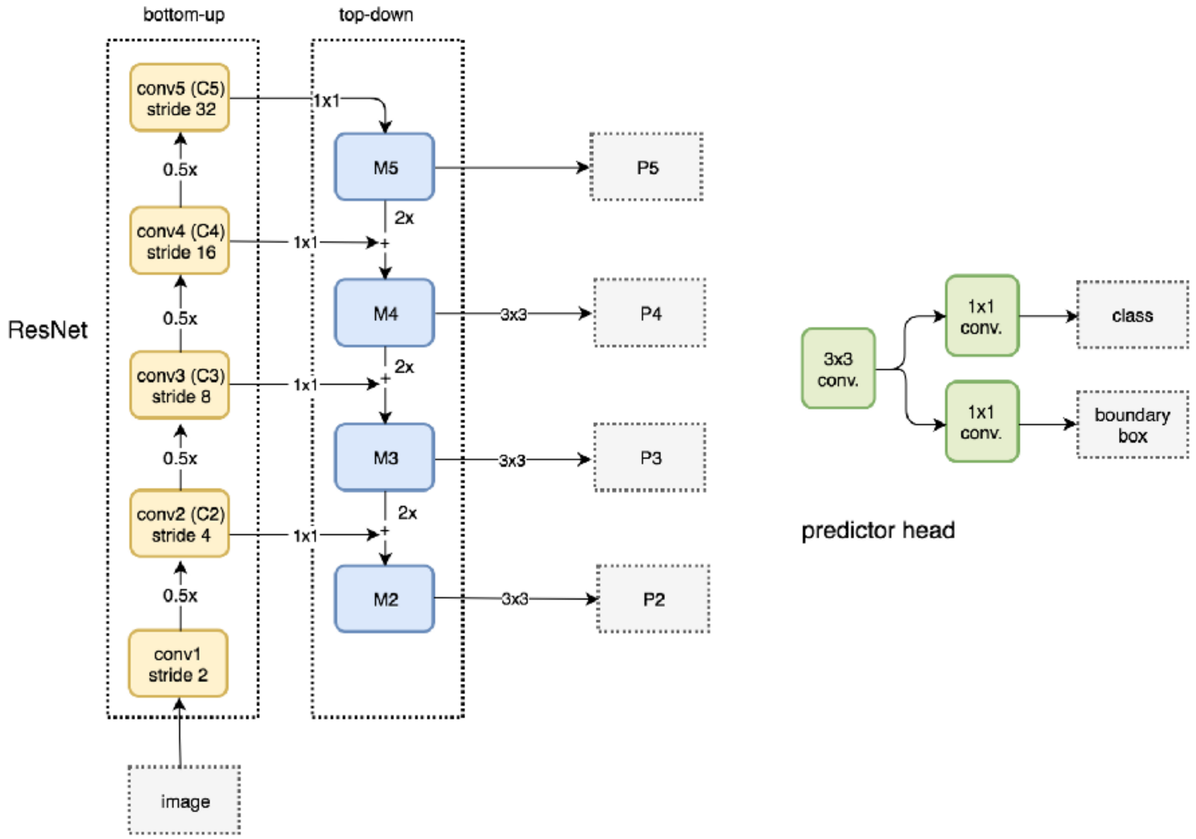
- **Top-down pathway:**
  - We apply a 1×1 convolution filter to reduce C5 channel depth to 256-d to create M5. This becomes the first feature map layer used for object prediction.
  - As we move in the top-down path, we **upsample** the previous layer by 2 using nearest neighbors upsampling.
  - We again apply a 1 × 1 convolution to the corresponding feature maps in the bottom-up pathway.
  - Then we add them element-wise.
  - We apply a 3×3 convolution to all merged layers. This filter reduces the aliasing effect when merged with the upsampled layer.
  - We repeat the same process for P3 and P2. However, we stop at P2 because the spatial dimension of C1 is too large. Otherwise, it will slow down the process too much. Because we **share** the same classifier and box regressor of every output feature maps, all pyramid feature maps (P5, P4, P3 and P2) have 256-d output channels.



FPN with RPN

- FPN is not an object detector by itself. It is a feature extractor that works with object detectors.

- For example, we extract multiple feature map layers with FPN and feed them into an RPN (a network that use convolutions and anchors) to detecting objects. RPN applies 3×3 convolutions over the feature maps followed by separate 1×1 convolution for class predictions and boundary box regression. These 3×3 and 1×1 convolutional layers are called the RPN head. The same head is applied to all feature maps.



FPN with RPN (2)

- Placing FPN in RPN improves AR (average recall: the ability to capture objects) to **56.3**, an **8.0** points improvement over the RPN baseline. The performance on **small** objects is increased by a large margin of **12.9** points.

- FPN is very competitive with state-of-the-art detectors. In fact, it beats the winners for the COCO 2016 and 2015 challenge.

RPN	feature	# anchors	lateral?	top-down?	AR^{100}	AR^{1k}	AR^{1k}_s	AR^{1k}_m	AR^{1k}_l
(a) baseline on conv4	C_4	47k			36.1	48.3	32.0	58.7	62.2
(b) baseline on conv5	C_5	12k			36.3	44.9	25.3	55.5	64.2
(c) **FPN**	$\{P_k\}$	200k	✓	✓	**44.0**	**56.3**	**44.9**	**63.4**	66.2
Ablation experiments follow:									
(d) bottom-up pyramid	$\{P_k\}$	200k	✓		37.4	49.5	30.5	59.9	**68.0**
(e) top-down pyramid, w/o lateral	$\{P_k\}$	200k		✓	34.5	46.1	26.5	57.4	64.7
(f) only finest level	P_2	750k	✓	✓	38.4	51.3	35.1	59.7	67.6

Faster R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP_s	AP_m	AP_l
(*) baseline from He *et al.* [16]†	RPN, C_4	C_4	conv5			47.3	26.3	-	-	-
(a) baseline on conv4	RPN, C_4	C_4	conv5			53.1	31.6	13.2	35.6	**47.1**
(b) baseline on conv5	RPN, C_5	C_5	2*fc*			51.7	28.0	9.6	31.9	43.1
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	2*fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	45.8

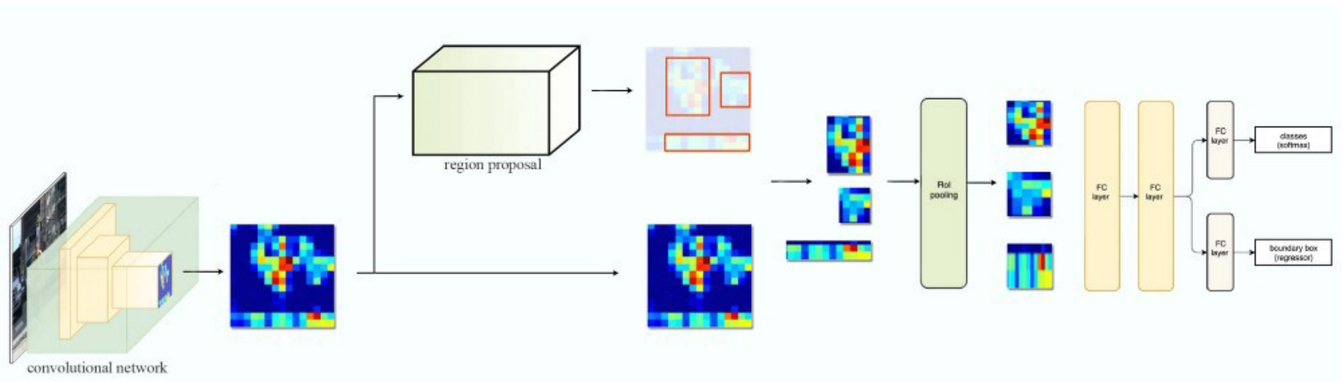
method	backbone	competition	image pyramid	test-dev $AP_{@.5}$	test-dev AP	test-dev AP_s	test-dev AP_m	test-dev AP_l	test-std $AP_{@.5}$	test-std AP	test-std AP_s	test-std AP_m	test-std AP_l
ours, Faster R-CNN on **FPN**	ResNet-101	-		**59.1**	**36.2**	**18.2**	**39.0**	48.2	**58.5**	**35.8**	**17.5**	**38.7**	47.8
*Competition-winning **single-model** results follow:*													
G-RMI†	Inception-ResNet	2016		-	34.7	-	-	-	-	-	-	-	-
AttractioNet‡ [10]	VGG16 + Wide ResNet§	2016	✓	53.4	35.7	15.6	38.0	**52.7**	52.9	35.3	14.7	37.6	**51.9**
Faster R-CNN +++ [16]	ResNet-101	2015	✓	55.7	34.9	15.6	38.7	50.9	-	-	-	-	-
Multipath [40] (on minival)	VGG-16	2015		49.6	31.5	-	-	-	-	-	-	-	-
ION‡ [2]	VGG-16	2015		53.4	31.2	12.8	32.9	45.2	52.9	30.7	11.8	32.8	44.8

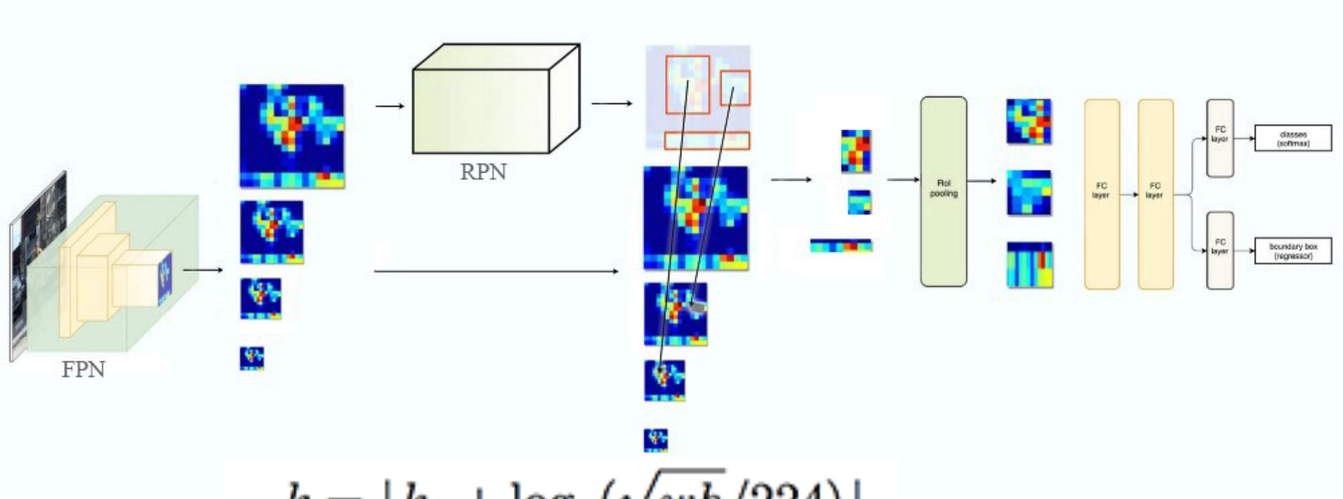
FPN with Fast R-CNN or Faster R-CNN

- Fast R-CNN and Faster R-CNN: Use one feature map layer to create ROIs. use the ROIs and the feature map layer to create feature patches to be fed into the ROI pooling.

- Using a FPN, we generate a **pyramid** of feature maps. We apply the RPN (described above) to generate ROIs. Based on the size of the ROI, we select the feature map layer in the most proper scale to extract the feature patches.

- The formula to pick the feature maps is based on the width w and height h of the ROI. So if k=3, we select P3 as our feature map. We apply the ROI pooling and feed the result to the Fast R-CNN head (Fast R-CNN and Faster R-CNN have the same head) to finish the prediction.





$$k = \lfloor k_0 + \log_2(\sqrt{wh}/224) \rfloor.$$

$k_0 = 4$

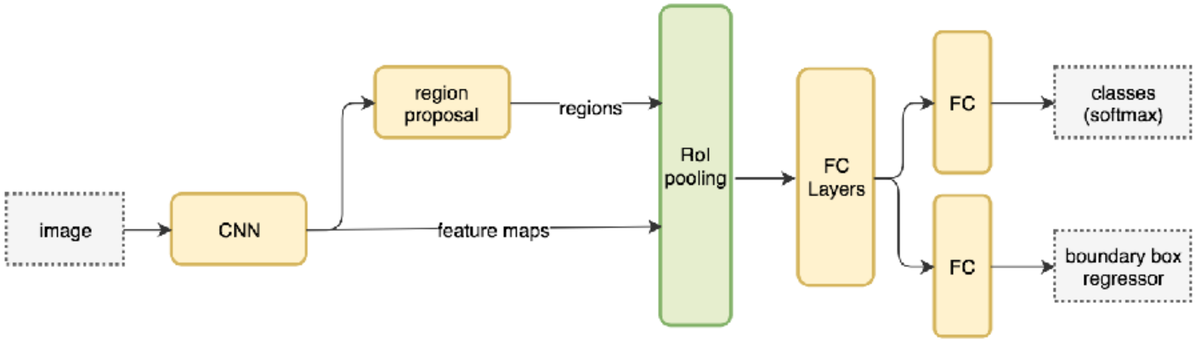
k is the P_k layer in the FPN used to generate the feature patch.

Single-shot methods

- YOLO v1/v2/v3
- SSD
- RetinaNet

From Region-based Detectors
to
Single Shot Detectors

Single Shot Detectors




```
for window in windows
    patch = get_patch(image, window)
    results = detector(patch)
```

```
ROIs = region_proposal(image)
for ROI in ROIs
    patch = get_patch(image, ROI)
    results = detector(patch)
```

```
feature_maps = process(image)
ROIs = region_proposal(feature_maps)
for ROI in ROIs
    patch = roi_align(feature_maps, ROI)
    results = detector2(patch)    # Reduce the amount of work here!
```

Faster R-CNN

- **Faster R-CNN**: dedicated RPN followed by a classifier, accurate but not fast (7 FPS), need to reduce the amount of work done for each ROI

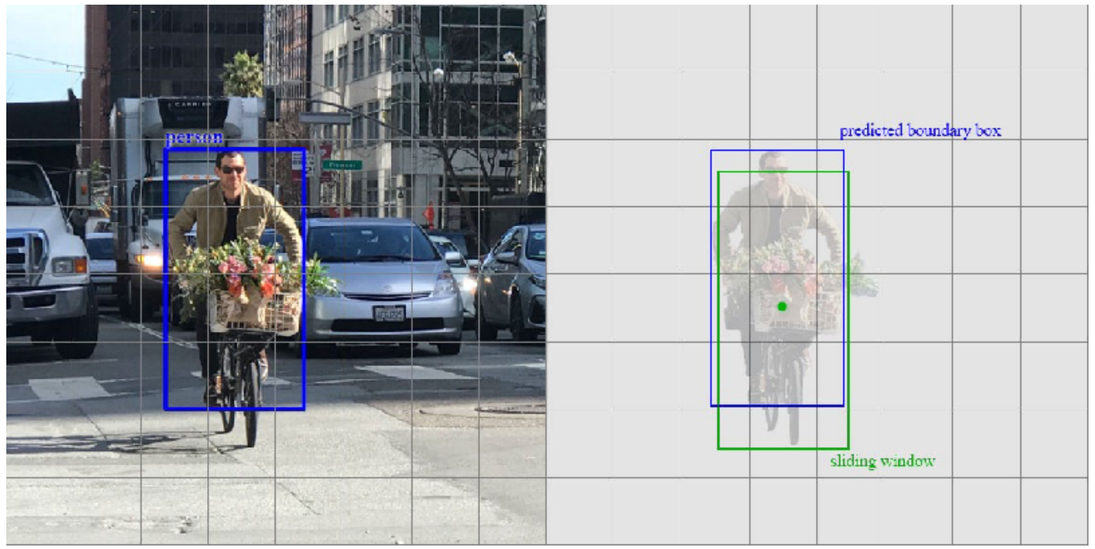
- As an alternative, do we need a separate region proposal step? Can we derive both boundary boxes and classes directly from feature maps in **one step**?

```
feature_maps = process(image)
results = detector3(feature_maps) # No more separate step for ROIs
```

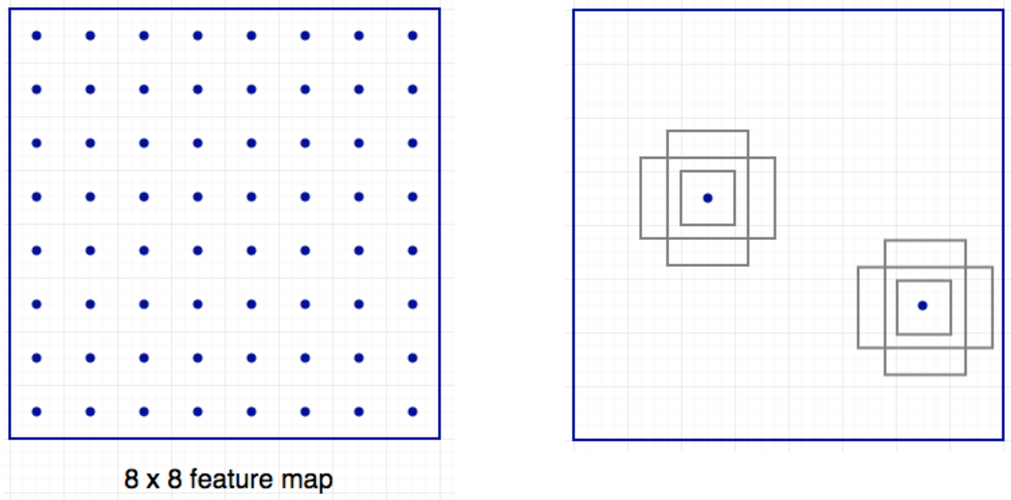
Single Shot Detectors

Single Shot Detectors (2)

- **Sliding-window detector**:
  - Use the windows as the **final** BB, need too many shapes to cover most objects. Instead, treat the window as an initial **guess** and let detector predict class and final BB from sliding window simultaneously.
  - We can also **slide** windows over **feature maps** (instead of images) to detect objects. This concept is very similar to the **anchors** in RPN and Faster R-CNN. However, single shot detectors predict **both** the BB and the class at the same time.

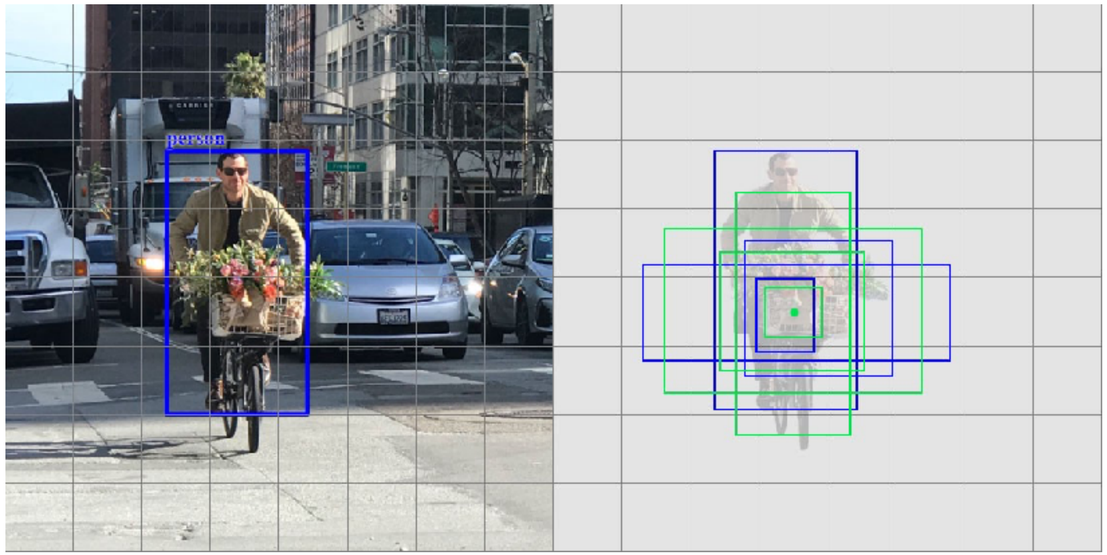


Single Shot Detectors (3)



- **Recap**: 8×8 feature **map**, k **anchors** at each location -> 8×8xk **predictions** in total (anchors: fixed initial BB **guesses**, one prediction for each anchor, carefully selected, same shapes at each location).

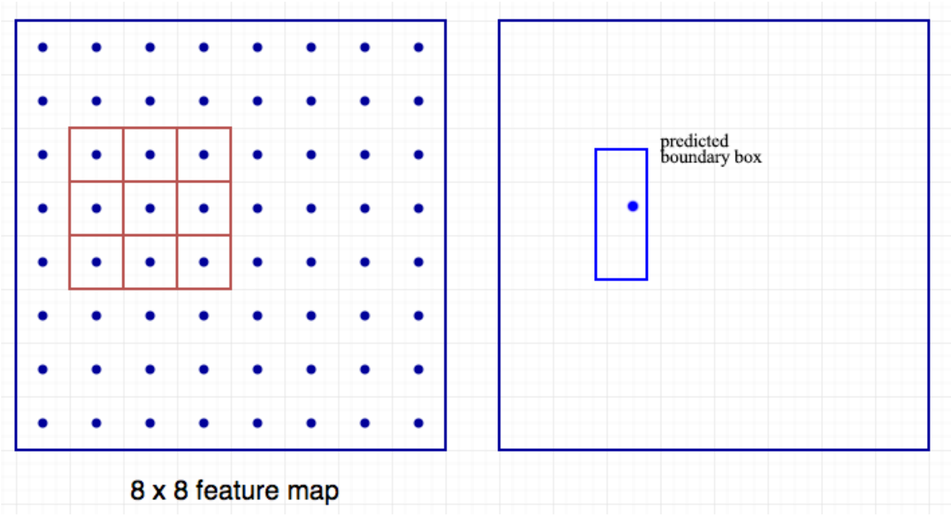
- Figure to right: **4 anchors** (green) and **4 corresponding predictions** (blue), each related to one specific anchor.

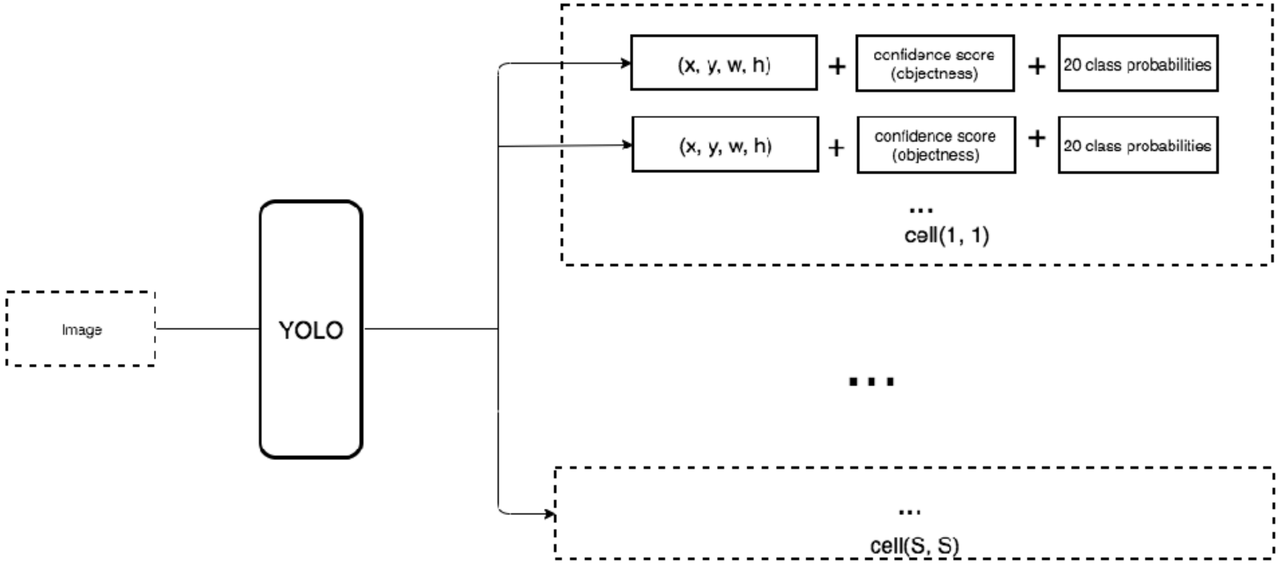


Single Shot Detectors (4)

- In **Faster R-CNN (FPN)**: One CNN to make a 5(6) parameter **prediction**: 4 for BB relative to anchor, 1(2) for objectness score. 5(6) 3×3×D filters transforms the feature map from 8×8×D to 8×8×5(6) - for each anchor.

- In a single shot detector, the convolution filters also predicts **C class** probabilities for classification (one per class). So we apply 25 3×3×D convolution filter to transform the feature map from 8×8×D to 8×8×25 for C=20.





Single Shot Detectors (5)

- Single shot detector often trades **accuracy** with real-time processing **speed**.
- They also tend to have issues in detecting objects that are **too close** or **too small**.
- Picture: 9 Santas in the lower left corner but only 5 detected.

person
umbrella
person
person
person

YOLO v1/v2/v3

You only look once

More info: Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3

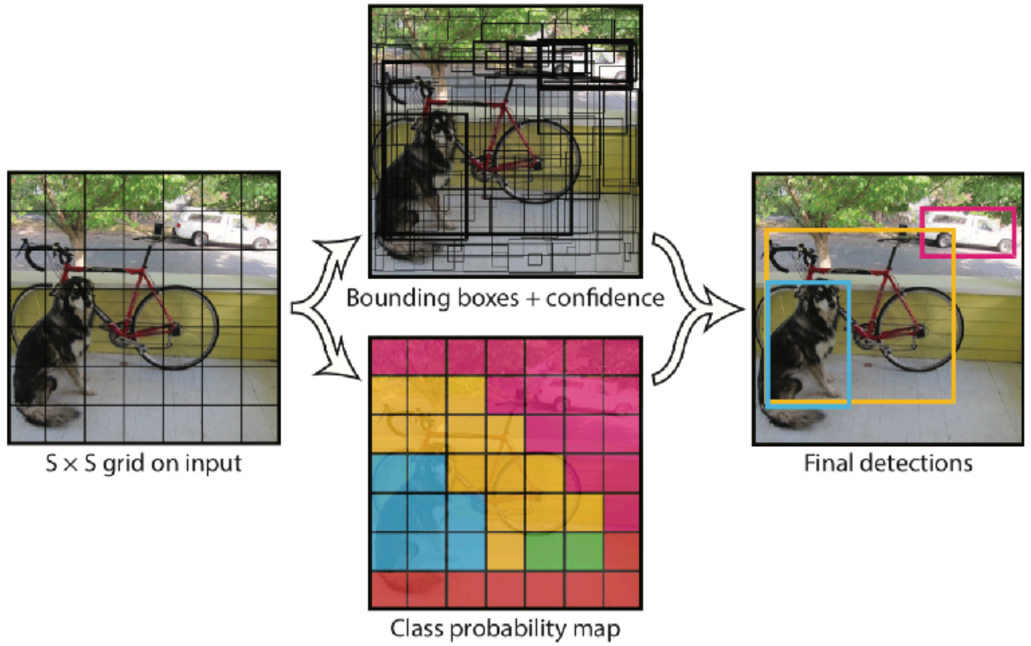
Paper: YOLO v1: You Only Look Once: Unified, Real-Time Object Detection

Paper: YOLO v2: YOLO9000: Better, Faster, Stronger

Paper: YOLO v3: YOLOv3: An Incremental Improvement

YOLO

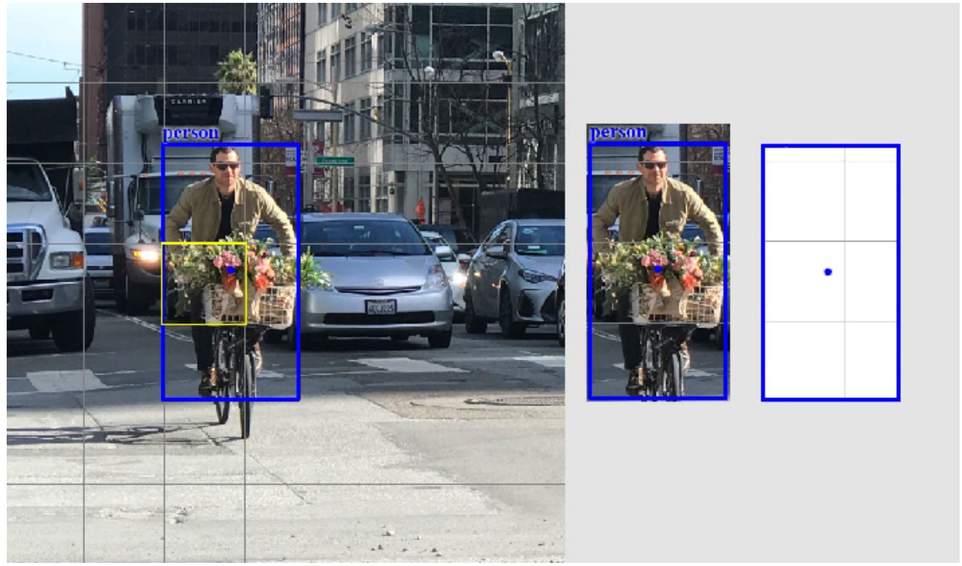
- Object detection system targeted for real-time processing.
- Incremental improvements: YOLOv1, YOLOv2, YOLOv3
- YOLOv3: Fast and Accurate

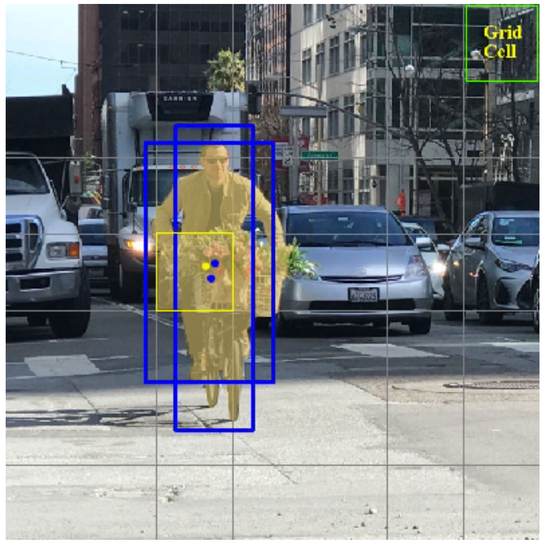


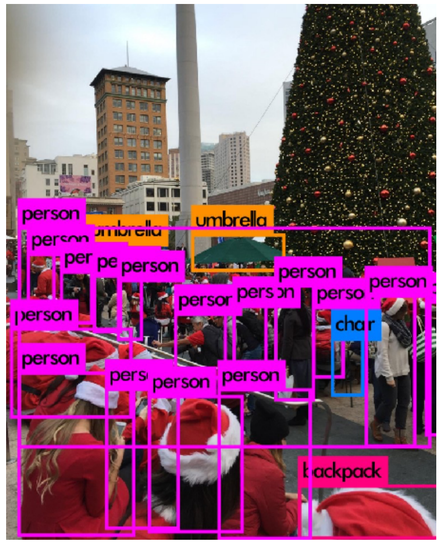
Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	-	-	7.07 Bn	200
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-416	COCO trainval	test-dev	57.9	140.69 Bn	20

YOLO v1

- YOLO divides the input image into an **S×S grid**
- Each grid cell predicts only **one** object, e.g. the yellow grid cell tries to predict the “person” object whose center (the blue dot) falls inside the grid cell.
- Each grid cell predicts a **fixed** number of bounding boxes (BBs).
- The one-object-per-grid-cell rule **limits** how **close** detected objects can be, e.g. only 5 of 9 Santas detected.

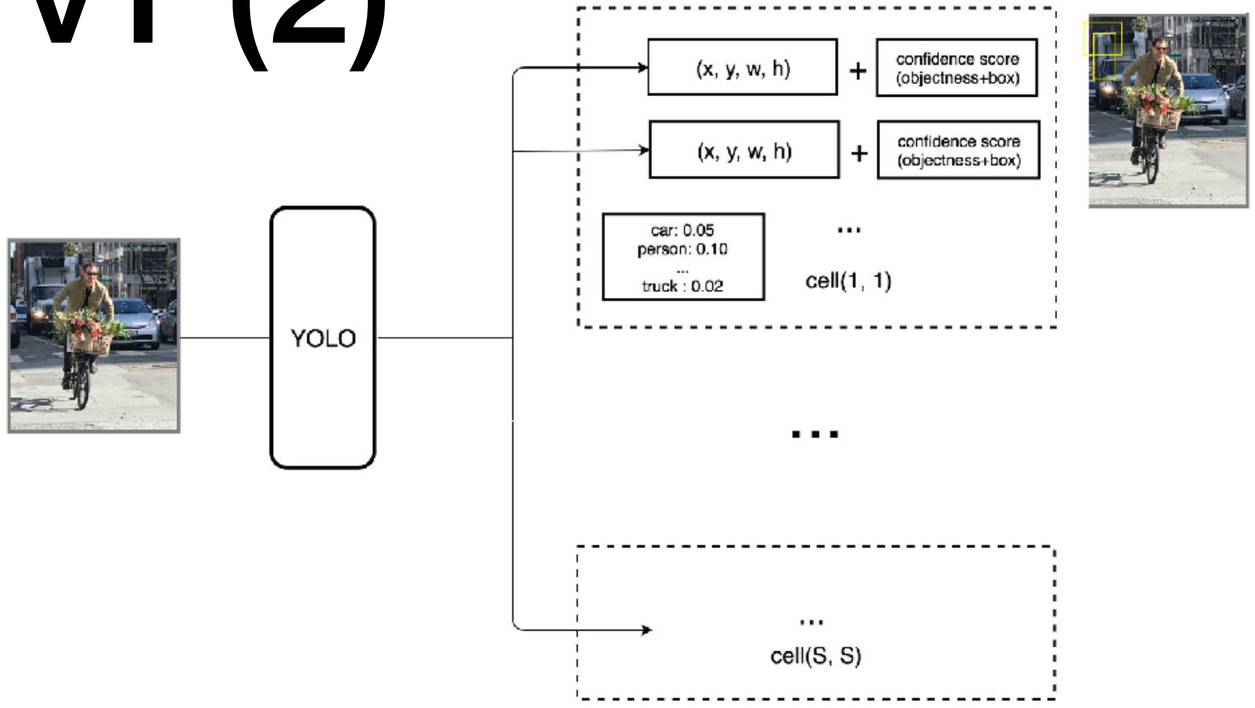


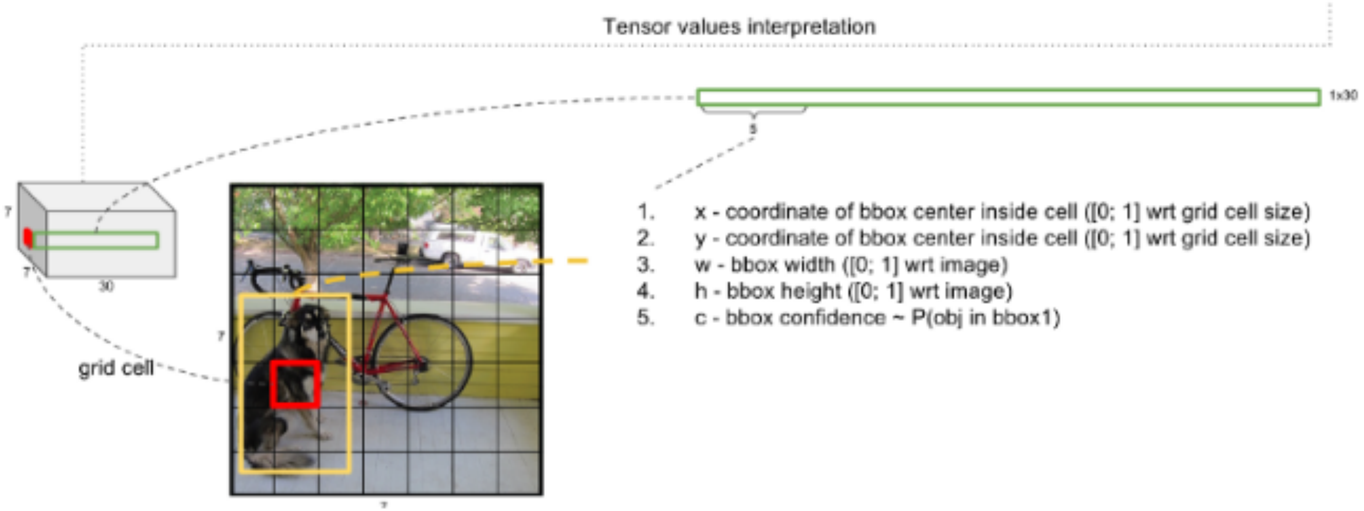



person
umbrella
person
person
person

YOLO v1 (2)

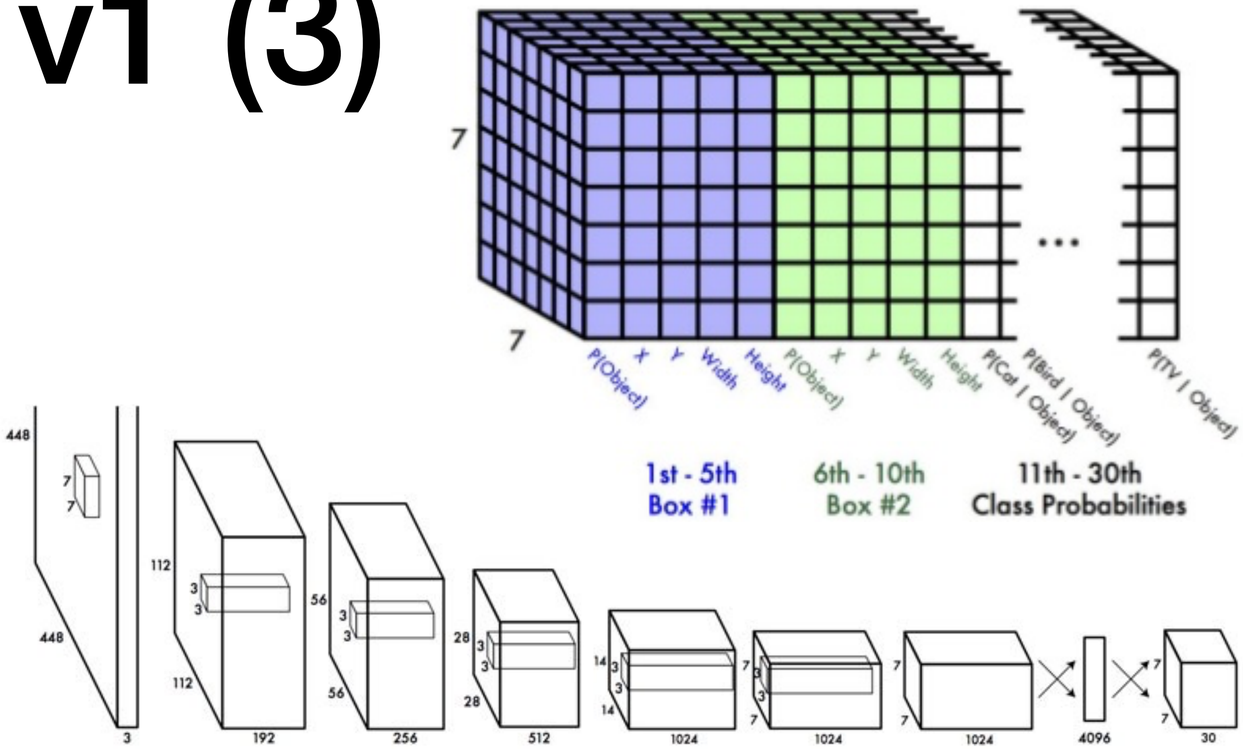
- For **each** grid **cell**:
  - predict **B** bounding boxes (BB) and a **box confidence score** for each BB
  - detect **one** object only regardless of the number of boxes B
  - predicts C **conditional class probabilitie** (one per class)
- YOLO (PASCAL VOC): 7×7 grids (S×S), 2 boundary boxes (B) and 20 classes (C).





YOLO v1 (3)

- Each BB contains 5 elements: (x, y, w, h) and a **box confidence score**.
    - The confidence score reflects how **likely** the box contains an object (objectness) and how **accurate** the BB is.
    - We **normalize** the bounding box width w and height h by the image width and height. x and y are offsets to the corresponding cell. Hence, x, y, w and h are all between 0 and 1.
- Each cell has 20 **conditional class probabilities**, the probability that the detected object belongs to a particular class given there is an object
- The major concept of YOLO is to build a CNN network to predict a **(7, 7, 30) tensor.**
-



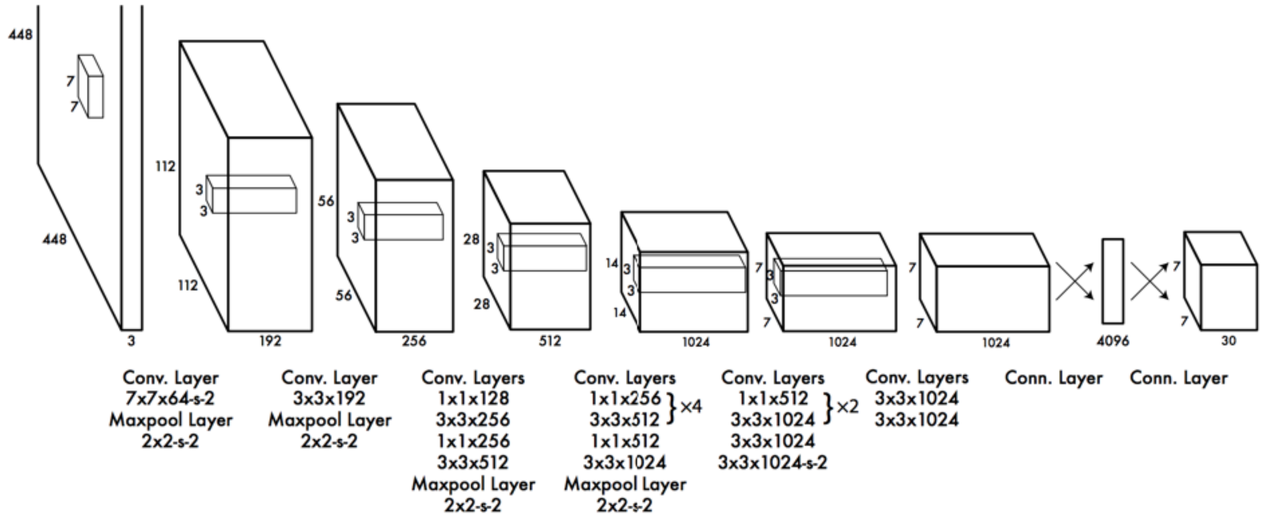
$$\text{box confidence score} \equiv P_r(object) \cdot IoU$$
$$\text{conditional class probability} \equiv P_r(class_i|object)$$
$$\text{class confidence score} \equiv P_r(class_i) \cdot IoU$$
$$= \text{box confidence score} \times \text{conditional class probability}$$

where

$P_r(object)$ is the probability the box contains an object.
IoU is the IoU (intersection over union) between the predicted box and the ground truth.
$P_r(class_i|object)$ is the probability the object belongs to $class_i$ given an object is presence.
$P_r(class_i)$ is the probability the object belongs to $class_i$

YOLO v1 (4)

- Network design:
  - YOLO has 24 convolutional layers followed by 2 fully connected layers (FC).
  - Some convolution layers use 1×1 reduction layers to reduce the depth of the features maps.
  - For the last convolution layer, it outputs a tensor with shape (7, 7, 1024). The tensor is then flattened.
  - Using 2 fully connected layers as a form of linear regression, it outputs 7×7×30 parameters and then reshapes it to (7, 7, 30), i.e. 2 boundary box predictions per location.



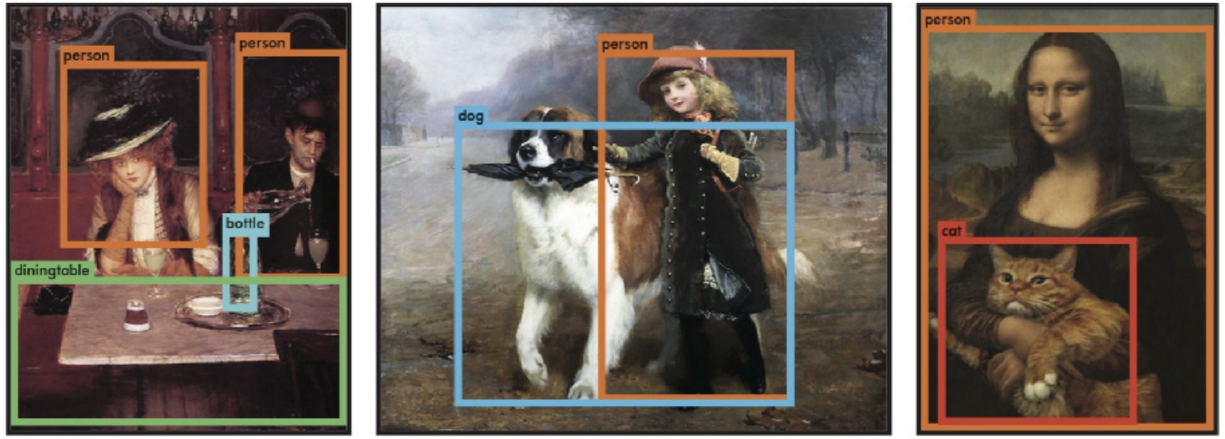
YOLO v1 (5)

- Loss function:
  - YOLO uses **sum-squared error** between the **predictions** and the **ground** truth to calculate loss. Predicts multiple bounding boxes, only use the one with the highest IoU for calc. TP, strategy leads to specialization (scale and aspect ratios)
  - The loss function composes of:
    - the **localization** loss (errors between the predicted boundary box and the ground truth). Object detected, small vs. large boxes: square root of w&h, more emphasis on the BB accuracy: λcoord=5
    - the **confidence** loss (the objectness of the box). Object detected vs. not detected, imbalance, background more frequently, λnoobj=0.5
    - the **classification** loss. Object detected, squared error of the class conditional probabilities

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
$$

YOLO v1 (6)

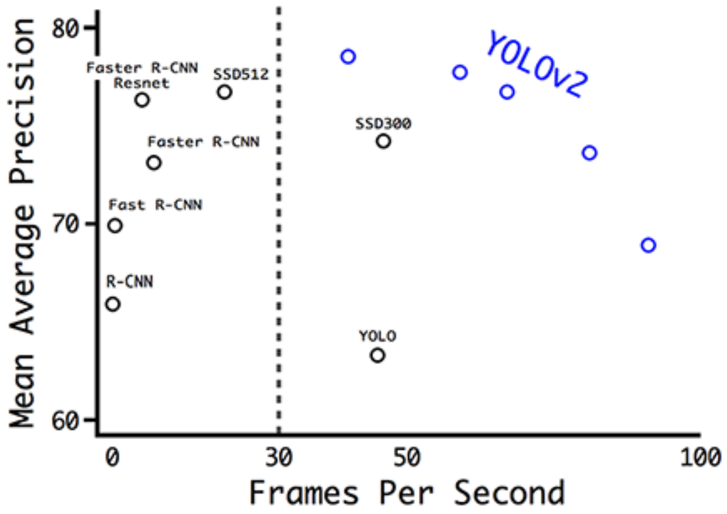
- Non-maximal suppression (**NMS**):
  - YOLO can make duplicate detections for the same object.
  - Apply NMS to remove duplications with lower confidence.
- **Benefits** of YOLO:
  - Fast. Good for real-time processing.
  - Trained end-to-end to improve accuracy.
  - Generalizes well from natural images to other domains like artwork
  - YOLO accesses to the whole image, demonstrates fewer false positives in background areas.
  - YOLO detects one object per grid cell. It enforces spatial diversity in making predictions.



YOLO v2

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

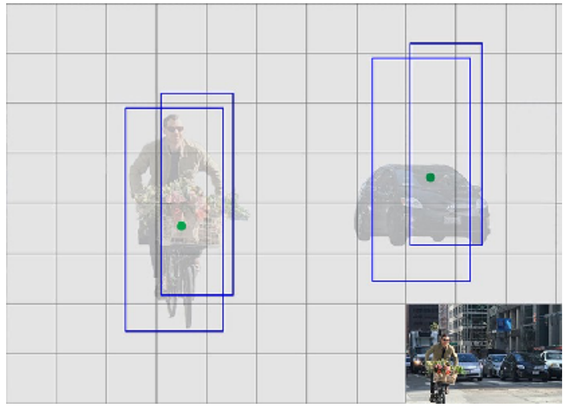
- Background: behind SSD and region-based methods in terms of accuracy.
- Goal: improve the accuracy significantly while making it faster.
- Several improvements made

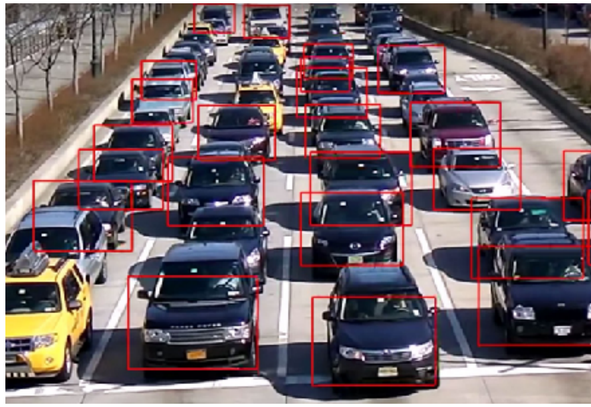


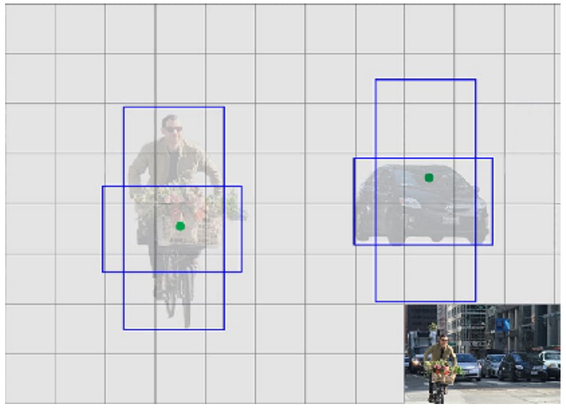
YOLO v2 (2)

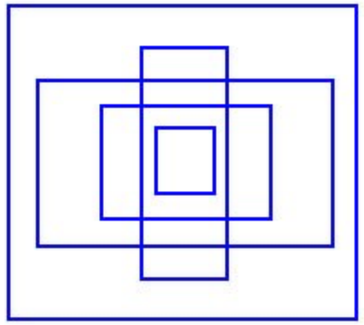
Accuracy improvements:

- **Batch normalization** (mAP up 2%): Conv layers, no dropout needed.
- **High-resolution classifier** (mAP up 4%): v1 training in 2 phases: Class: 224×224 -> 448×448
- Convolutional with **Anchor Boxes**:
  - v1 uses arbitrary guesses on the boundary boxes, unstable gradients, in early training predictions are fighting with each other on what shapes to specialize on, real-world BB are not arbitrary (cars pedestrians).
  - Initial training will be more stable if we start with diverse guesses that are common for real-life objects, e.g. 5 **anchor** boxes with the shapes shown. Instead of predicting 5 arbitrary BB, we predict **offsets** to each of the anchor boxes. If we **constrain** the offset values, we can maintain the diversity of the predictions and have each prediction focuses on a specific shape. So the initial training will be more **stable**.
  - Anchor boxes decrease mAP slightly from 69.5 to 69.2 but the recall improves from 81% to 88%. i.e. even the accuracy is slightly decreased but it increases the chances of detecting all the ground truth objects.

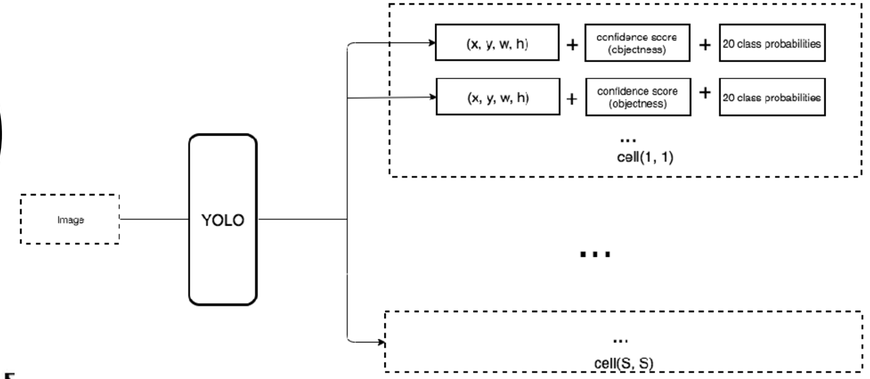


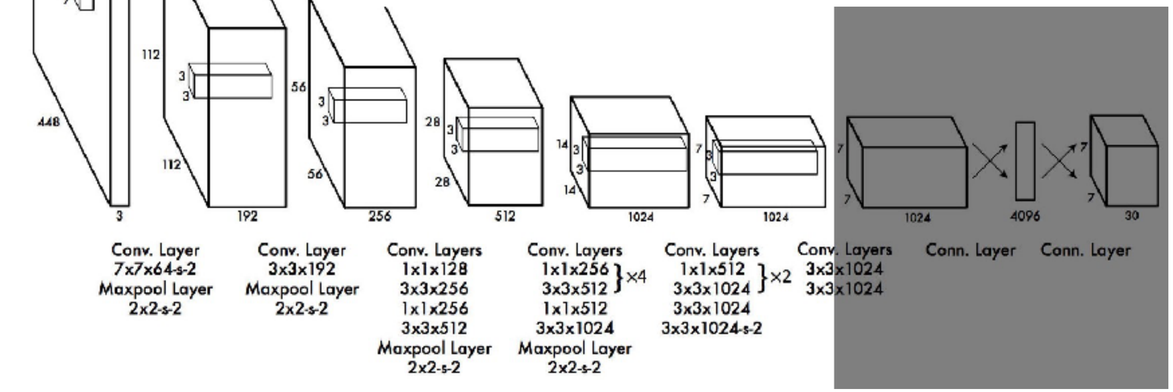






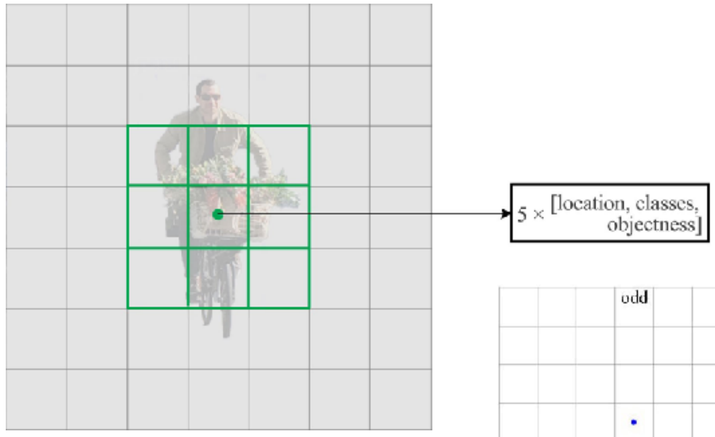
YOLO v2 (3)

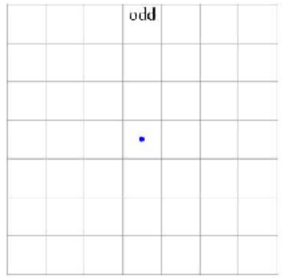


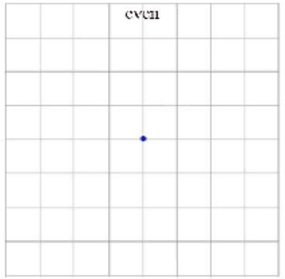


Accuracy improvements:

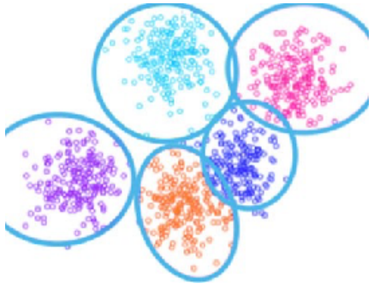
- Convolutional with **Anchor Boxes**: Changes to the network:
  - Remove the fully connected layers responsible for predicting the boundary box.
  - We move the class prediction from the cell level to the boundary box level. 5 boundary boxes with 25 parameters: 125 parameters per grid cell (vs. 30)
  - To generate predictions with a shape of 7×7×125, we replace the last convolution layer with three 3×3 convolutional layers each outputting 1024 output channels. Then we apply a final 1×1 convolutional layer to convert the 7×7×1024 output into 7×7× 125.
  - Change the input image size from 448 × 448 to 416 × 416 (7×7 v.s. 8×8 grid cell)
  - Remove one pooling layer to make the spatial output of the network to 13×13 (instead of 7×7).





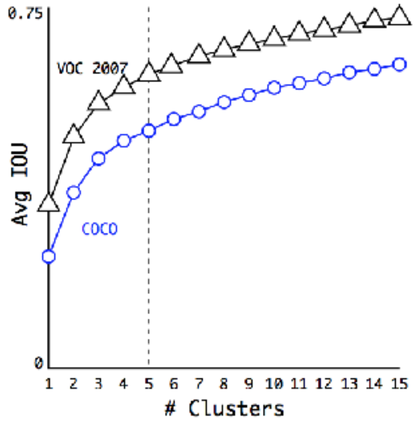


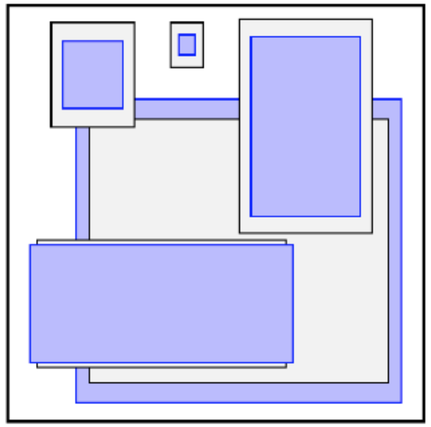
YOLO v2 (4)



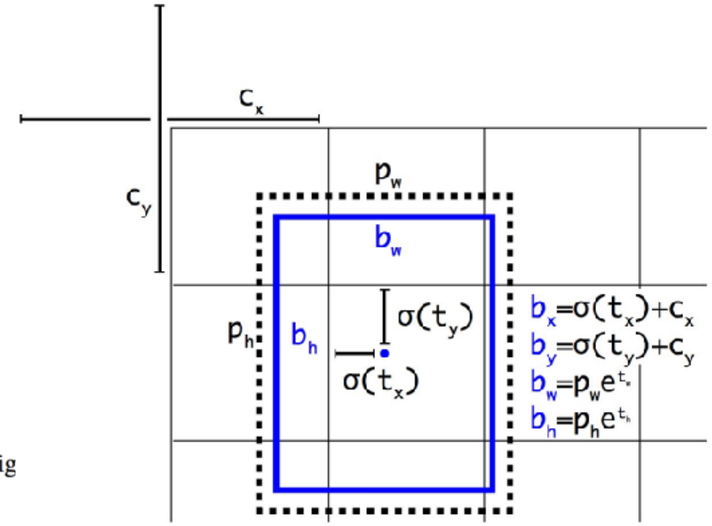
Accuracy improvements:

- **Dimension Clusters:**
  - Identify the top-K anchors that have the best coverage for the training data
  - IoU between the anchors and the ground truth boxes using different numbers of clusters (anchors). COCO vs. VOC2007 datasets.
- **Direct location prediction**
  - We make predictions on the offsets to the anchors. Constrained





$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$



t_x, t_y, t_w, t_h are predictions made by YOLO.

c_x, c_y is the top left corner of the anchor.

c_w, c_h are the width and height of the anchor.

c_x, c_y, c_w, c_h are normalized by the image width and heig

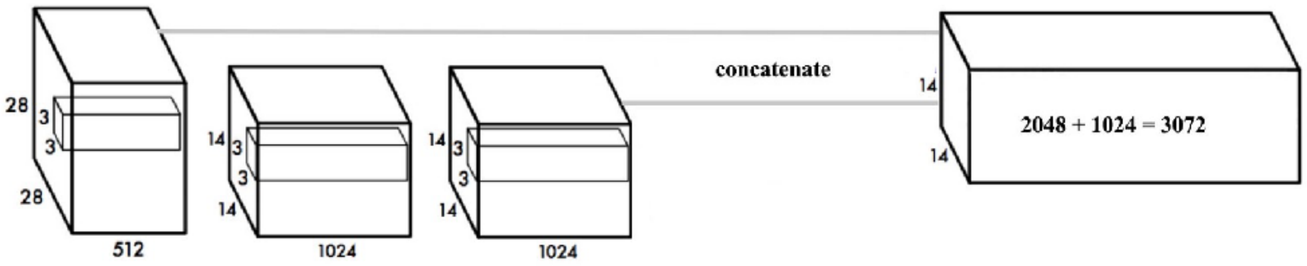
b_x, b_y, b_w, b_h are the predicted boundary box.

$\sigma(t_o)$ is the box confidence score.

YOLO v2 (5)

Accuracy improvements:

- **Fine-Grained Features:**
  - Convolution layers decrease the spatial dimension gradually. As the corresponding resolution decreases, it is harder to detect small objects.
  - One approach: Locate objects from different layers of feature maps so each layer specializes at a different scale (SSD). YOLO adopts a different approach called **passthrough**
- **Multi-Scale Training:**
  - After removing the fully connected layers, YOLO can take images of different sizes. Since the YOLO network downsamples the input by 32, we just need to make sure the width and height is a multiple of 32.
  - OLOv2 randomly selects another image size to train the model, this acts as data augmentation.



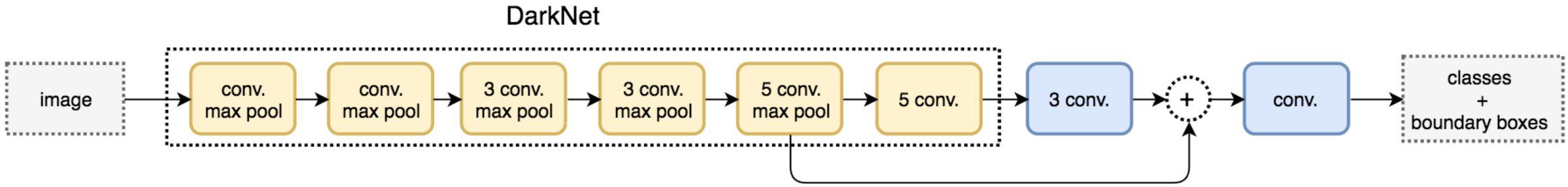
	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

YOLO v2 (6)

Speed improvement

- Simplify the backbone: less operations:
  - VGG16 -> customized GoogLeNet -> Darknet
- Darknet uses mostly 3 × 3 filters to extract features and 1 × 1 filters to reduce output channels. It also uses global average pooling to make predictions.
- We replace the last convolution layer (the cross-out section) with three 3×3 convolutional layers each outputting 1024 output channels. Then we apply a final 1×1 convolutional layer to convert the 7×7×1024 output into 7×7×125. (5 boundary boxes each with 4 parameters for the box, 1 objectness score and 20 conditional class probabilities)

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
~~Convolutional~~	~~1000~~	~~1×1~~	~~7×7~~
~~Avgpool~~		~~Global~~	~~1000~~
~~Softmax~~			



YOLO v3

- **Class Prediction**:
  - mutually **exclusive** object classes (e.g. “pedestrian” and “child”), uses multi-label classification, replaces the softmax function with independent logistic classifiers to calculate the likeliness of the input belongs to a specific label.
- **Bounding box prediction & cost function calculation**:
  - Predicts an objectness score for each bounding box using logistic regression. and changes the way the cost function is calculated.
- **Feature Pyramid Networks (FPN) like Feature Pyramid:**
  - YOLOv3 makes 3 predictions per location. Each prediction consists of a BB, a objectness and 80 class scores, i.e. $N \times N \times [3 \times (4 + 1 + 80)]$ predictions.
  - YOLOv3 makes predictions at 3 different scales (similar to the FPN)

YOLO v3 (2)

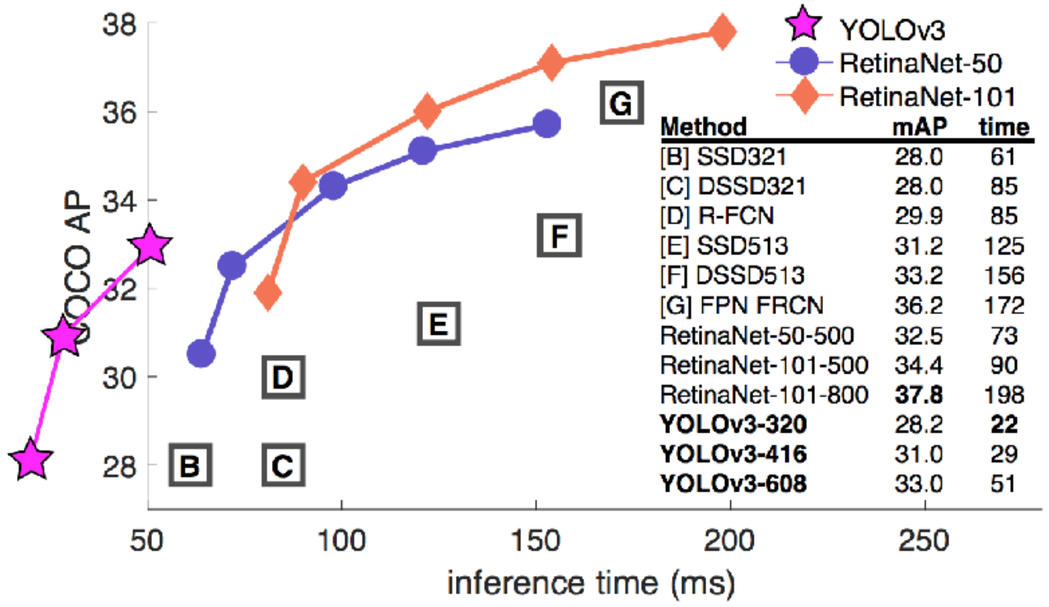
- **Feature extractor:**
  - A new 53-layer Darknet-53 is used to replace the Darknet-19 as the feature extractor.
  - Darknet-53 mainly compose of 3×3 and 1×1 filters with **skip** connections like the residual network in **ResNet**. Darknet-53 has less BFLOP (billion floating point operations) than ResNet-152, but achieves the same classification **accuracy** at 2x faster.

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

YOLO v3 (2)

- **YOLOv3 performance:**
  - YOLOv3's COCO AP metric is on par with SSD but 3x faster (still behind RetinaNet)
  - YOLOv3 also shows significant improvement in detecting small objects.
  - YOLOv3 performs very well in the fast detector category when speed is important.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [7]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

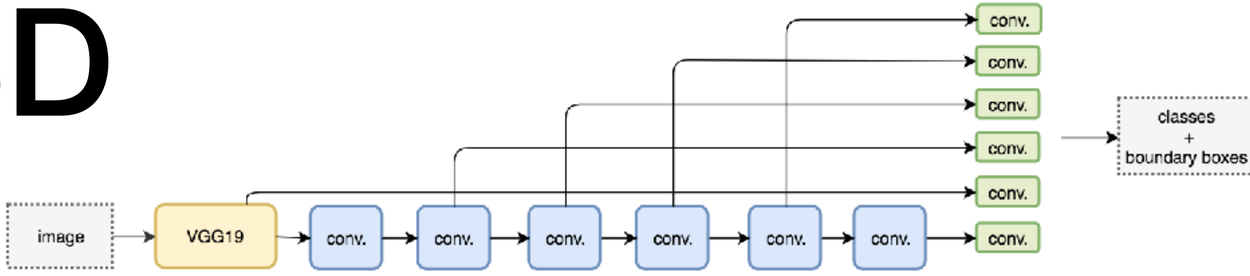
SSD

Single Shot MultiBox Detector

More info: SSD object detection: Single Shot MultiBox Detector for real-time processing

Paper: SSD: Single Shot MultiBox Detector

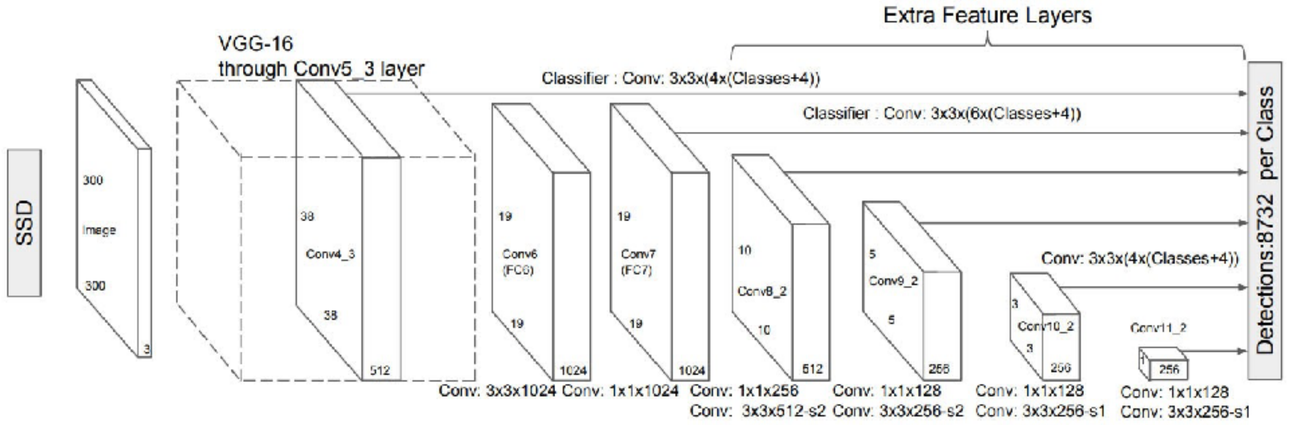
SSD

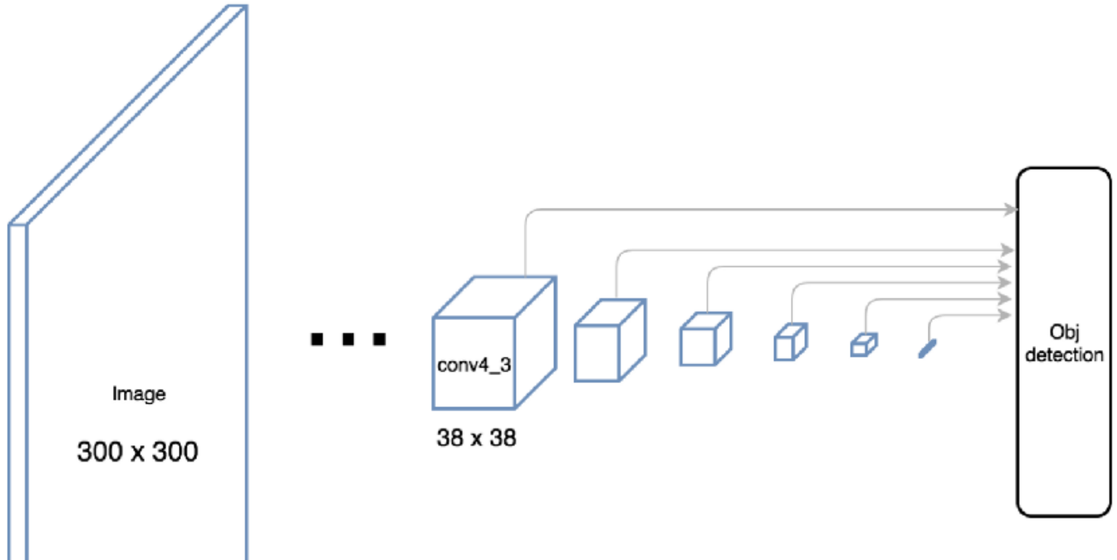


- **SSD** use the VGG16(19) network as the feature **extractor**. Then we add **custom** convolution layers (blue) afterward (**multi-scale**) and use convolution **filters** (green) to make **predictions**.

- Convolution layers **reduce** spatial dimension and resolution so we make **independent** object detections from **multiple** feature maps.

- If we redraw the diagram closer to **scale**, we should realize the **spatial** resolution has **dropped** significantly and may already miss the opportunity in locating **small** objects that are too hard to detect in low resolution.



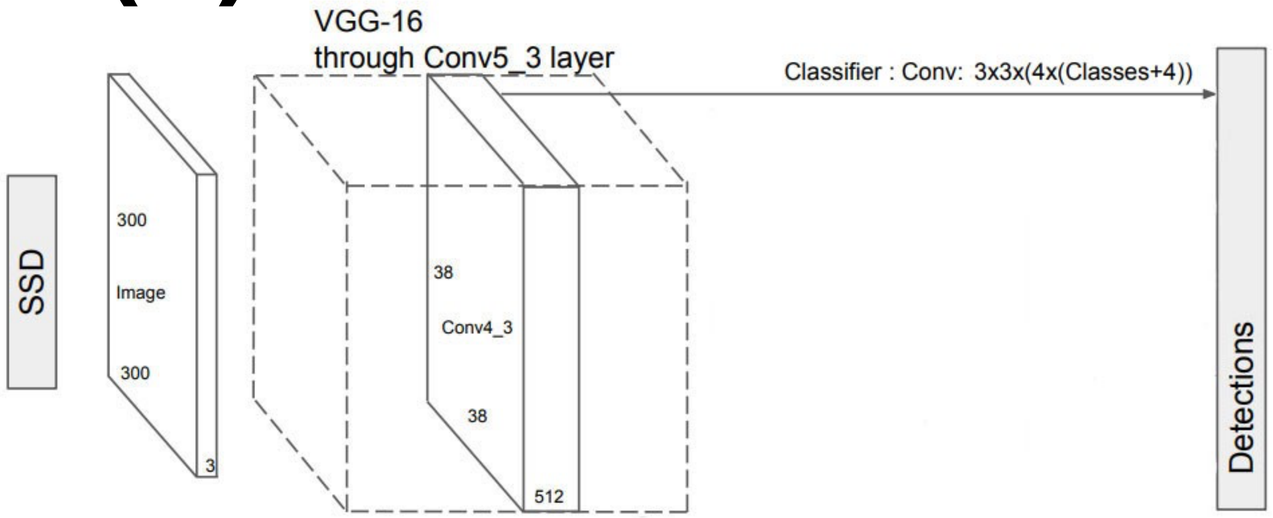


SSD (2)

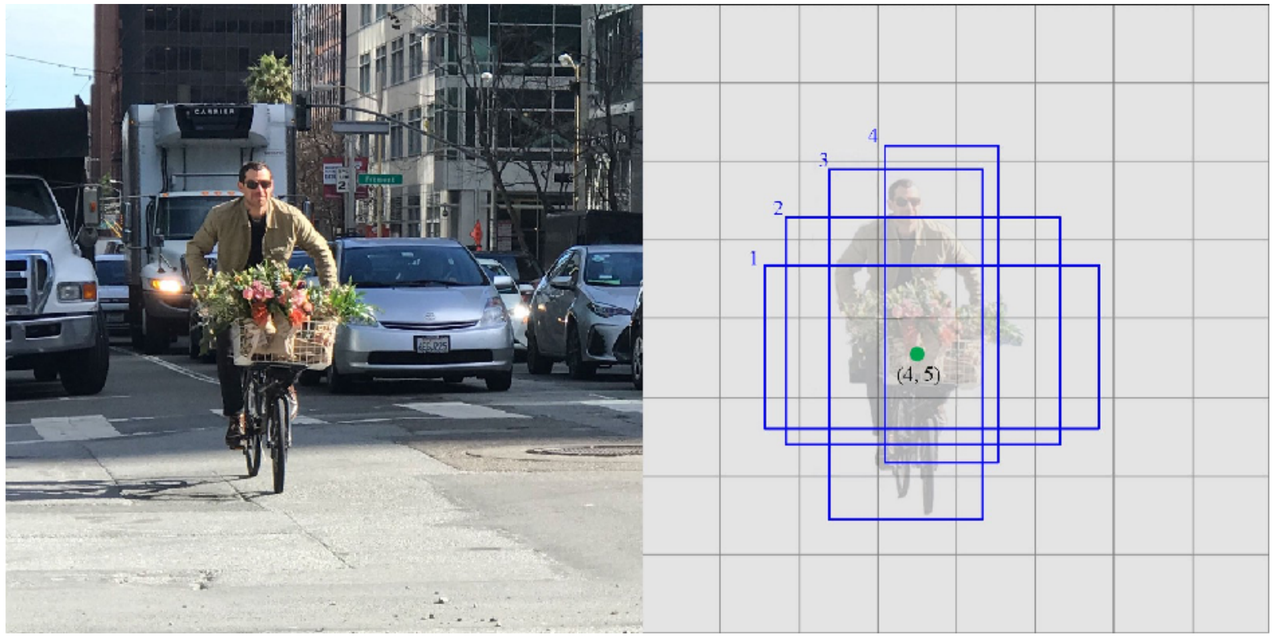
- **Background**:
  - SSD is designed for object detection in **real-time**.
  - Faster R-CNN is considered the start-of-the-art in **accuracy** at the time, but the speed is just **7 FPS**.
  - SSD **speeds** up the process (to real-time) by **eliminating** the need of the **RPN** and applies several **improvements**, including **multi-scale features** and **default boxes** to **match** the accuracy.
- In fact, the improvements allow SSD to match the Faster R-CNN's accuracy using **lower resolution images**, which further pushes the **speed** higher.

System	VOC2007 test *mAP*	FPS (Titan X)	Number of Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	~6000	~1000 x 600
YOLO (customized)	63.4	45	98	448 x 448
SSD300* (VGG16)	77.2	46	8732	300 x 300
SSD512* (VGG16)	79.8	19	24564	512 x 512

SSD (3)



- Overall, the SSD method consist of **two** parts:
  - Extract feature **maps**
  - Apply convolution **filters** to detect objects
- SSD uses **VGG16** to extract feature maps.
- Then it detects objects using the map from the **Conv4_3** layer (38×38 spatially, 8×8 here for illustration)
- For each **cell** (also called location), it makes 4 object **predictions**.

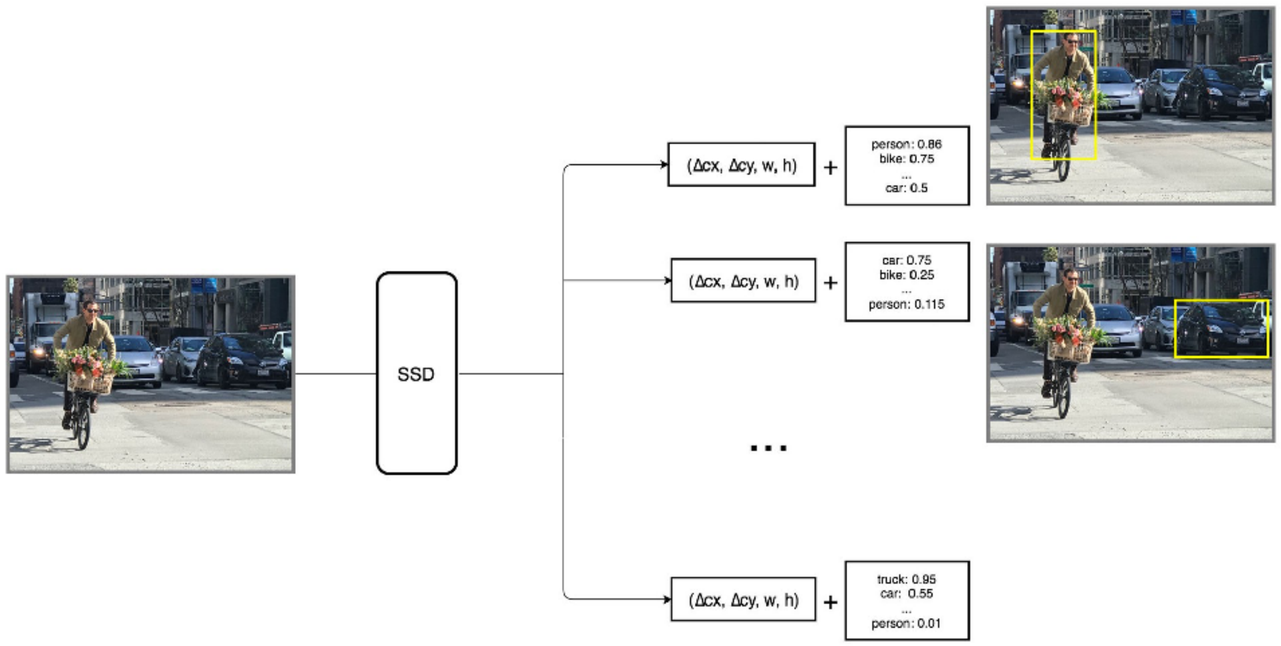


SSD (4)

- Each **prediction** consists of a **BB** and **21 scores**, **20** classes and one extra class for **no object**, and we pick the **highest** score as the class for the BB.

- Conv4_3 makes a total of 38×38 × 4 predictions: **four** predictions **per cell** regardless of the depth of the feature maps. As expected, many predictions contain **no object**. SSD reserves a class “0” to indicate it has no objects.

- Making **multiple** predictions containing BBs and confidence scores is called **multibox**

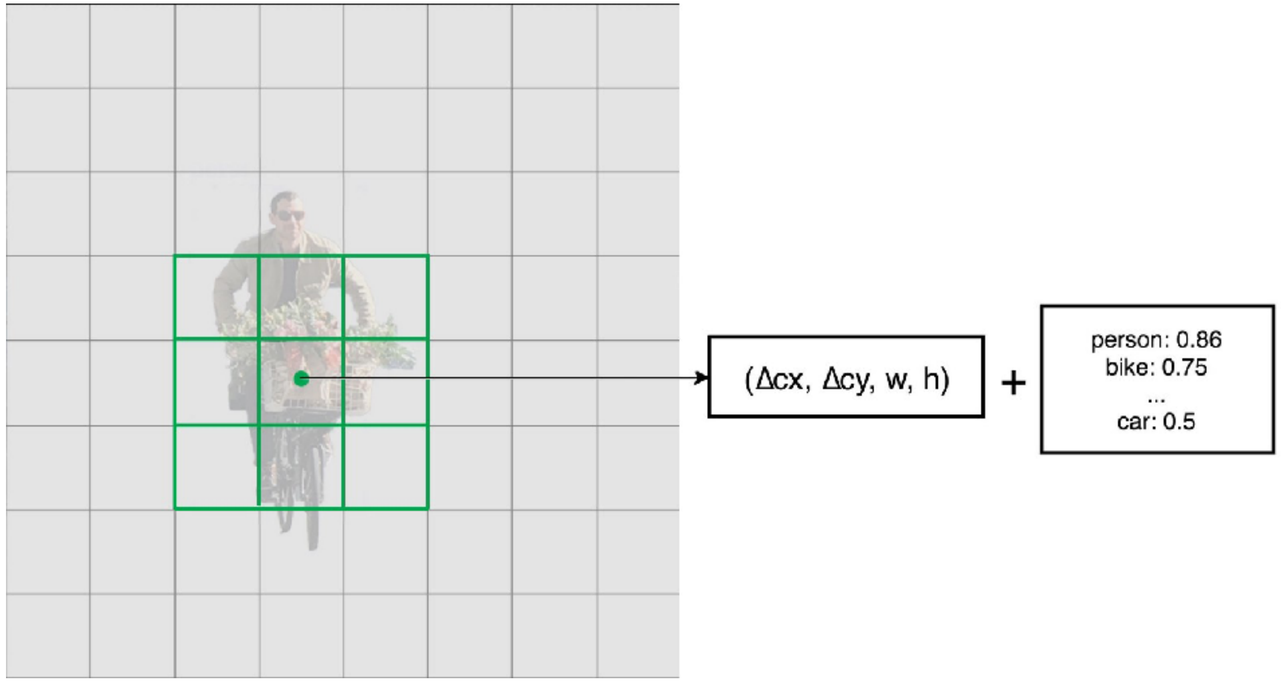


SSD (5)

- After extracting the **feature** maps, SSD predicts both the location and class scores using **small 3×3 convolution filters** for each **cell**.

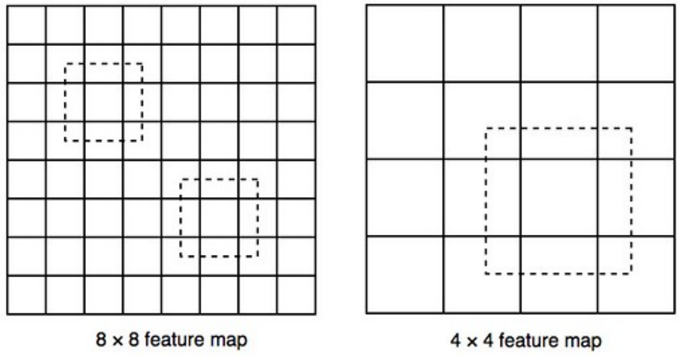
- Each filter outputs 25 **channels**: one BB pluss 21 scores for each class.

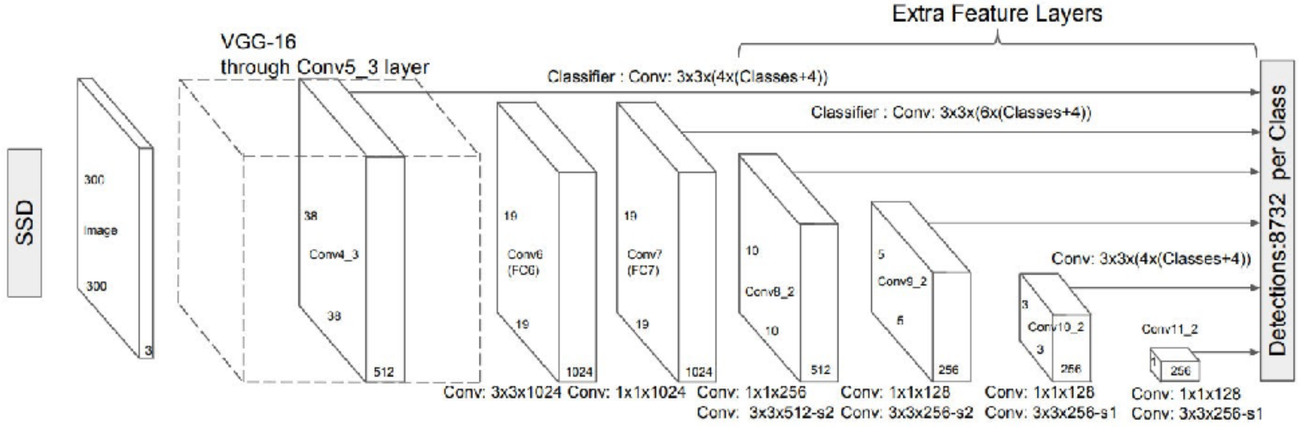
- For example, in Conv4_3, we apply **four** 3×3 filters to map 512 input channels to 25 output channels.



SSD: Multi-scale feature maps

- SSD uses **multi-scale feature maps** to detect objects independently.
- As CNNs **reduce** the **spatial dimension** gradually, the resolution of the **feature maps** also decrease.
- SSD **uses lower resolution layers** to detect **larger scale objects** (e.g. the 4x4 feature map to the right)
- SSD adds **6** more **auxiliary** convolution layers after the VGG16. **Five** of them will be used for object detection (**6** with Conv4_3). In **three** of those layers, we make **6 predictions instead of 4**. In total, SSD makes **8732** predictions using 6 layers.
- Multi-scale feature maps **improve accuracy** significantly.





Prediction source layers from:						mAP use boundary boxes?		# Boxes
38 × 38	19 × 19	10 × 10	5 × 5	3 × 3	1 × 1	Yes	No	
✔	✔	✔	✔	✔	✔	74.3	63.4	8732
✔	✔	✔				70.7	69.2	9864
	✔					62.4	64.0	8664

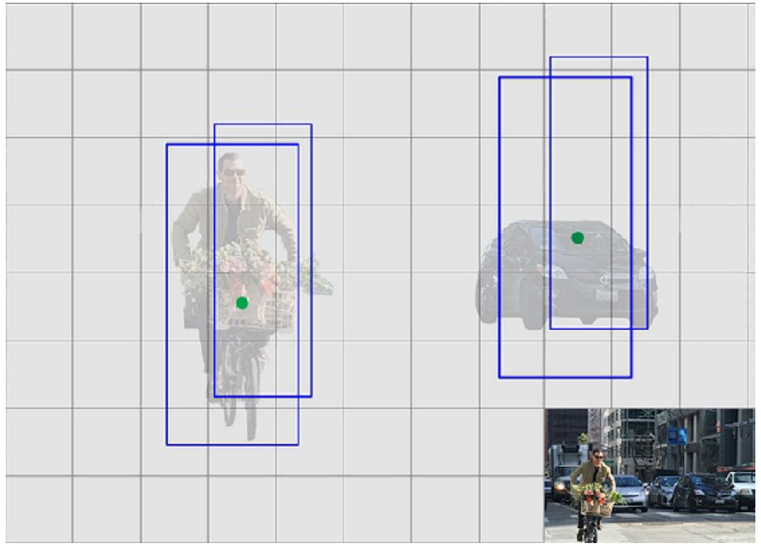
SSD: Default boxes

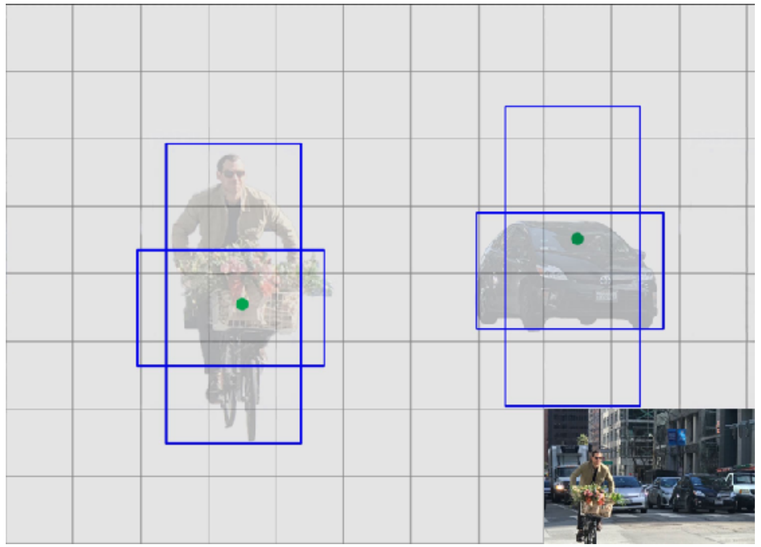
- To predict BB: we can start with **random** predictions (one alt.) and use **gradient** descent to optimize the model.

- Initial training: fight to determine what **shapes** (pedestrians or cars) to be optimized for which predictions. Empirical results indicate early training can be very **unstable**.

- Some BBs work well for **one** category but not for others.

- We want our initial predictions to be **diverse** and not looking similar. If our BBs cover **more shapes** our model can detect **more object types**. This kind of head start makes training much **easier** and more **stable**.





SSD: Default boxes (2)

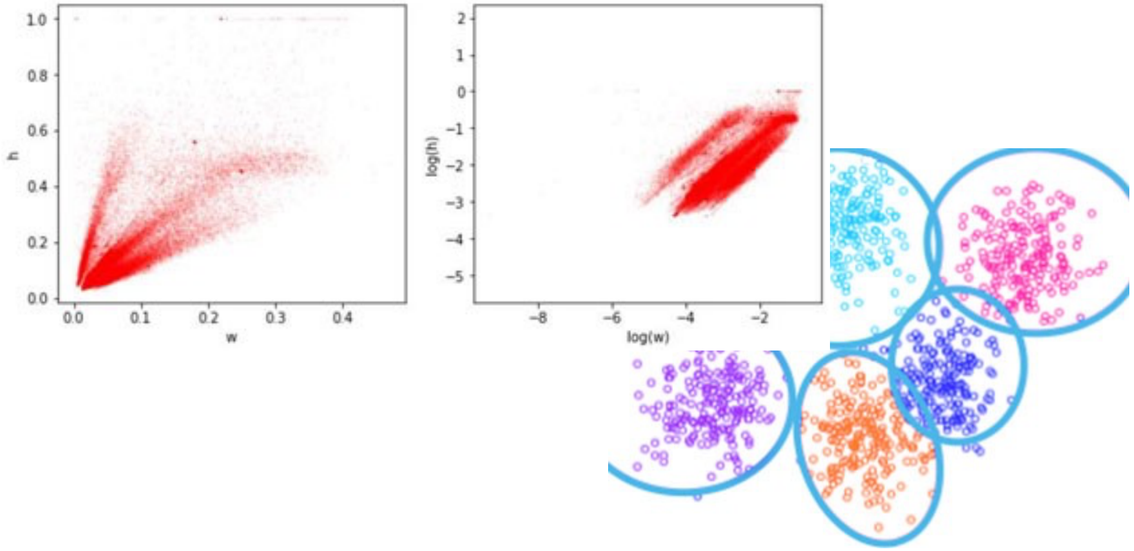
- **Real-world BBs** do not have arbitrary **shape** and size, e.g. pedestrians have an approximate aspect ratio of 0.41

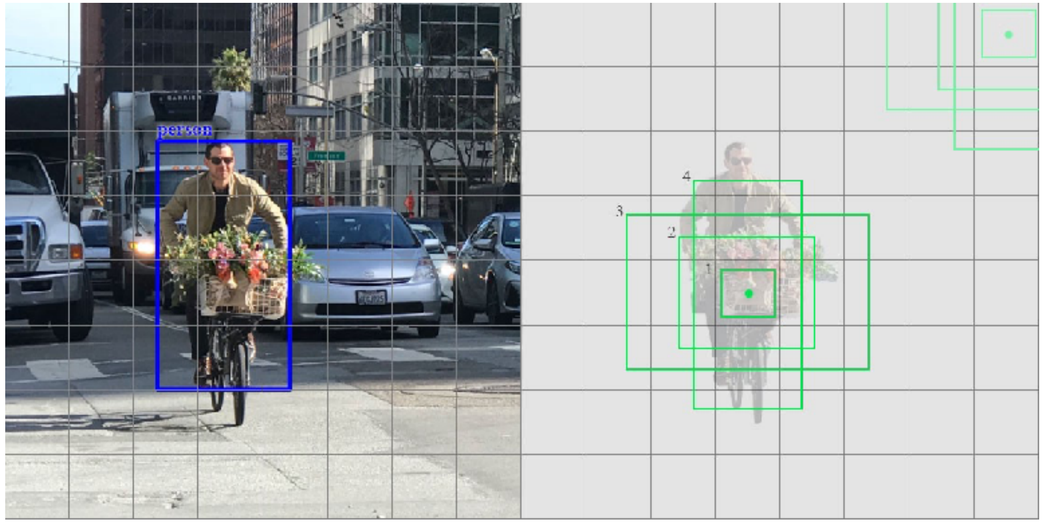
- In the KITTI dataset used for autonomous driving, the **width and height** distributions for the BBs are highly **clustered**.

- Conceptually, the **ground truth** BBs can be partitioned into **clusters** with each cluster represented by a **default** BB (the centroid of the cluster). So, instead of making random **guesses**, we can start the guesses based on those **default boxes**.

- In practice, default boxes are carefully **pre-selected** to **cover** a **wide** spectrum of real-life objects and the **number** of boxes are kept to a **minimum** (4 or 6 for SSD) with one prediction per default box. Instead of predicting **global** box coordination the BB predictions are **relative** to the default BB at each cell, i.e. the **offsets** to the default box at each cell.

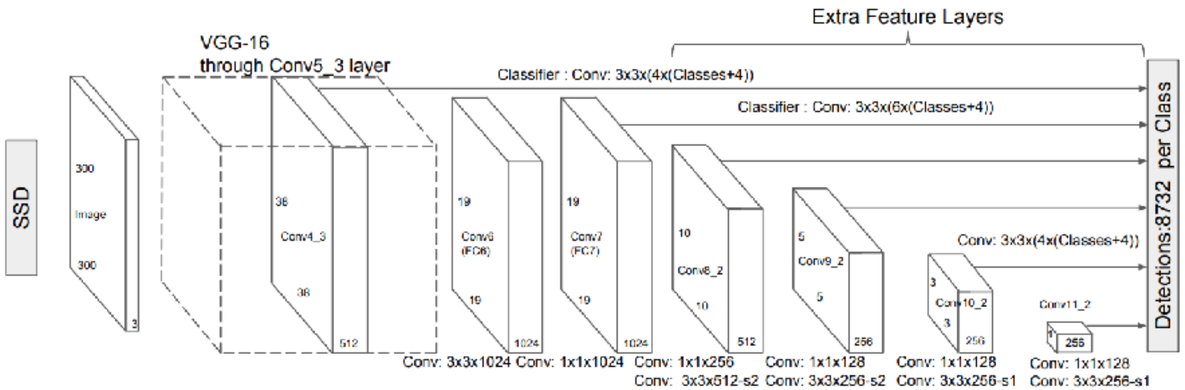
- Each **feature** map layers **shares** the same set of default boxes centered at the corresponding cell. But different layers use different sets of default boxes to customize object detections at different **resolutions**. Four default boxes (green)are illustrated to the right.





SSD: Default boxes (3)

- **Choosing** default boundary boxes:
  - SSD defines a **scale value** for each feature map layer. Starting from the left, Conv4_3 detects objects at the smallest scale 0.2 (or 0.1 sometimes) and then increases linearly to the rightmost layer at a scale of 0.9.
  - Combining the scale value with the **target aspect ratios**, we compute the width and the height of the default boxes. For layers making 6 predictions, SSD starts with 5 target aspect ratios: 1, 2, 3, 1/2 and 1/3.
  - (YOLO uses k-means clustering on the training dataset to determine the default boxes)



$$w = scale \cdot \sqrt{\text{aspect ratio}}$$

$$h = \frac{scale}{\sqrt{\text{aspect ratio}}}$$

Then SSD adds an extra default box with scale:

$$scale = \sqrt{\text{scale} \cdot \text{scale at next level}}$$

and aspect ratio = 1.

SSD: Matching strategy

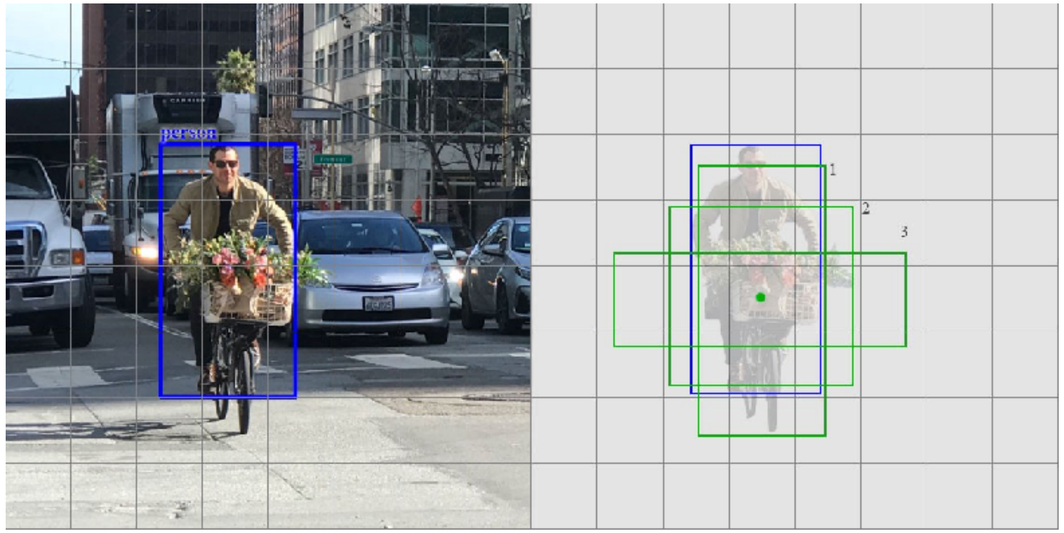
- SSD **predictions** are classified as **positive** matches or negative matches. SSD only uses positive matches in calculating the localization **cost**.

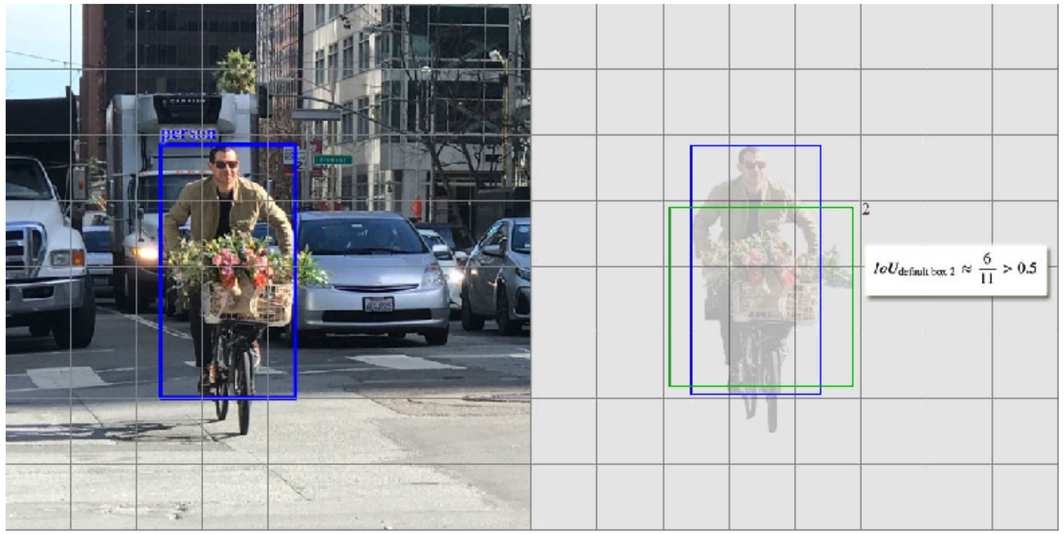
- If the corresponding default BB (not the predicted BB) has an **IoU** greater than 0.5 with the ground truth BB, the match is positive. Otherwise, it is negative.

- Let's say we have 3 default boxes (green). Only default box 1 and 2 (but not 3) have an IoU greater than 0.5 with the ground truth box in figure to the right (blue box). So only box 1 and 2 are positive matches.

- Once we identify the positive matches, we use the corresponding **predicted** BB to calculate the cost.

- This matching strategy encourages each prediction to predict **shapes** closer to the **corresponding** default box. Therefore the predictions are more **diverse** and more **stable** during training.



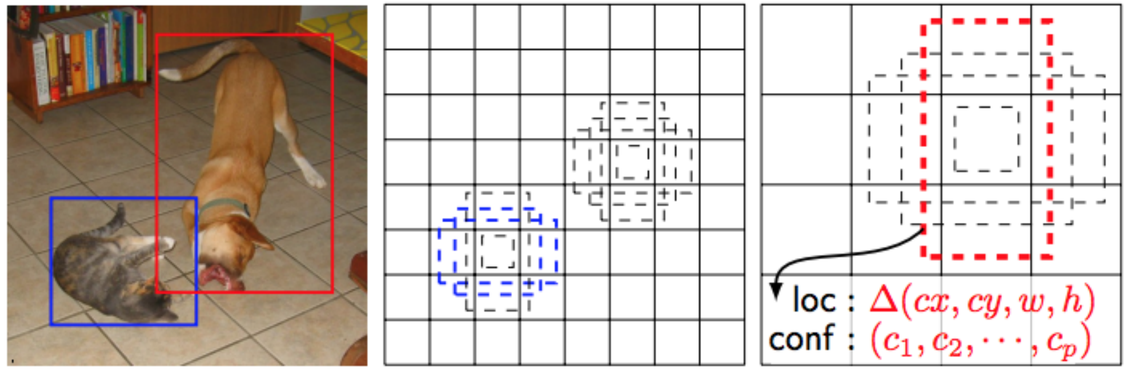


SSD:
Multi-scale feature maps & default boundary boxes

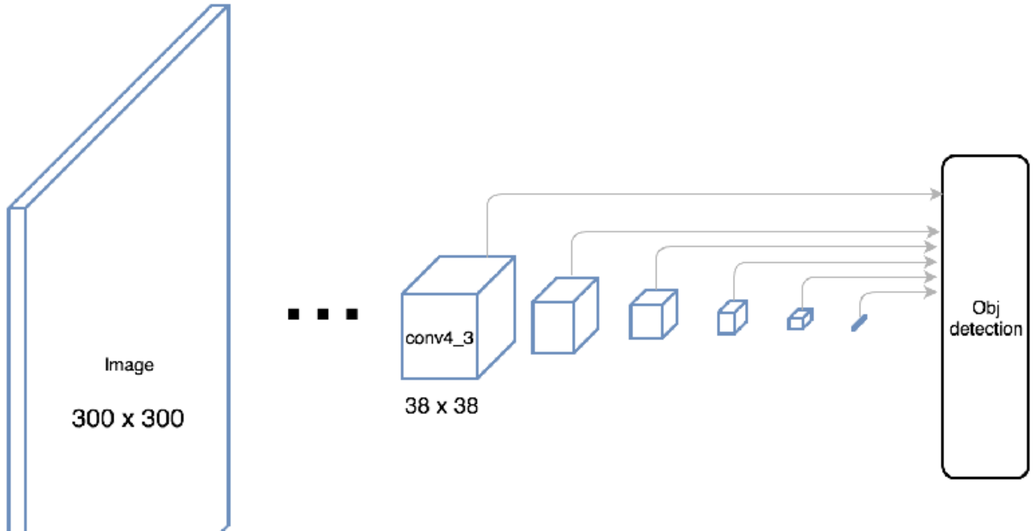
- Example showing how SSD combines **multi-scale** feature maps and **default** boxes to detect objects at different **scale**.

  - The **dog** matches one default box (in red) in the 4×4 feature map layer, but not any default boxes in the higher resolution 8×8 feature map. The **cat** which is smaller is detected only by the 8×8 feature map layer in 2 default boxes (in blue).

  - **Higher resolution** feature maps are responsible for detecting **small** objects. The first layer for object detection conv4_3 has a **spatial** dimension of 38×38, a pretty **large reduction** from the input image. Hence, SSD usually performs badly for small objects compared with other detection methods. If it is a problem, we can mitigate it by using images with higher resolution.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map



SSD: Loss function

- **Localization** loss: mismatch between predicted and GT BB.
  - SSD only penalizes predictions from **positive** matches.
- **Confidence** loss: loss in making a class prediction
  - For every **positive** match prediction, we penalize the loss according to the confidence score of the corresponding class.
  - For **negative** match predictions, we penalize the loss according to the confidence score of the class “0”: class “0” classifies **no object** is detected.
- The final loss function is computes as the **sum** (N is the **number** of positive match and α is the **weight** for the localization loss)

The localization loss between the predicted box l and the ground truth box g is defined as the smooth L1 loss with cx, cy as the offset to the default bounding box d of width w and height h.

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^{w} = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^{h} = \log\left(\frac{g_j^h}{d_i^h}\right)$$

$$x_{ij}^p = \begin{cases} 1 & \text{if } IoU > 0.5 \text{ between default box } i \text{ and ground true box } j \text{ on class } p \\ 0 & \text{otherwise} \end{cases}$$

It is calculated as the softmax loss over multiple classes confidences c (class score).

$$L_{conf}(x,c) = -\sum_{i \in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

where N is the number of matched default boxes.

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g))$$

SSD: Hard negative mining

- We make far **more predictions** than the number of **objects** present in an image, i.e. more negative matches than positive matches.

- This creates a class **imbalance** which hurts training. We are training the model to **learn background** space rather than detecting objects.

- However, SSD still requires negative sampling so it can learn what constitutes a bad prediction. So, instead of using **all** the negatives, we **sort** those negatives by their calculated confidence loss. SSD picks the negatives with the **top loss** and makes sure the **ratio** between the picked negatives and positives is at most 3:1

- This leads to a **faster** and more **stable** training.

SSD: Data augmentation

- Data augmentation is **important** in improving accuracy.
- Augment data with flipping, cropping and color distortion.
- To handle **various object sizes and shapes**, each **training** image is randomly **sampled** by one of the following options: 1) Use the original, 2) Sample a patch with IoU of 0.1, 0.3, 0.5, 0.7 or 0.9, 3) Randomly sample a patch.
- The **sampled patch** will have an aspect ratio between 1/2 and 2. Then it is **resized** to a fixed size and we flip one-half of the training data. In addition, we can apply photo distortions.
- The performance **improvement** after data augmentation is **significant**.

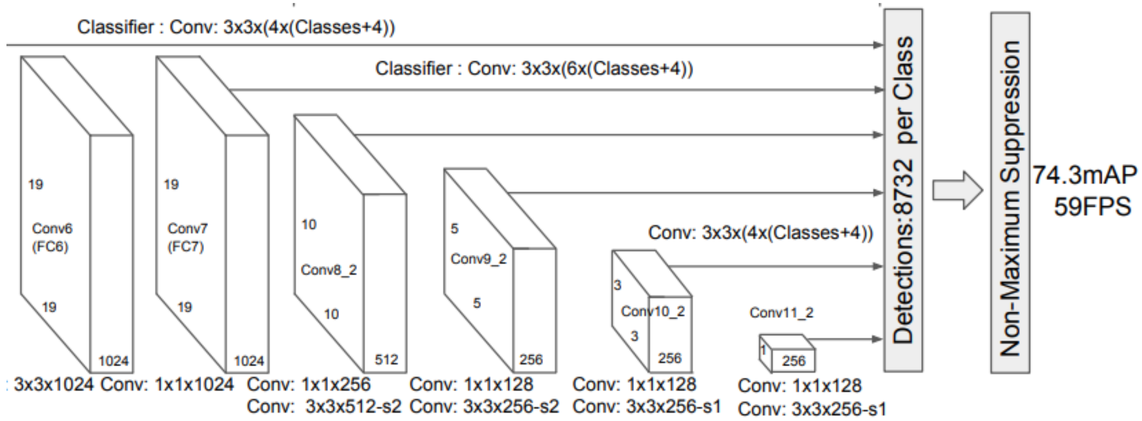
data augmentation	SSD300		
horizontal flip	✔	✔	✔
random crop & color distortion		✔	✔
random expansion			✔
VOC2007 test mAP	65.5	74.3	**77.2**

SSD: Inference time & NMS

- SSD makes **many** predictions (8732) for a better **coverage** of location, scale and aspect ratios (more than many other detection methods). Many predictions contain **no object**. Therefore, any predictions with class **confidence** scores lower than 0.01 will be **eliminated**.

- SSD uses **non-maximum suppression** (NMS) to **remove duplicate** predictions pointing to the same object. SSD **sorts** the predictions by the confidence scores. Start from the **top** confidence predictions, SSD evaluates whether any other predicted BBs have an **IoU** higher than **0.45** for the same class, if so - remove the other predictions. At most, we keep the top 200 predictions per image.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512



SSD: Result

- The model is trained using SGD with initial learning rate 0.001 , 0.9 momentum, 0.0005 weight decay, and batch size 32.
- Using a Nvidia Titan X on VOC2007: **SSD** achieves 59 FPS with mAP 74.3%, vs. **Faster** R-CNN 7 FPS with mAP 73.2% or **YOLO** 45 FPS with mAP 63.4%.
- For SSD, it uses image size of 300×300 or 512×512
- A comparison of the performance, including FPS can be found in the table to the left.

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	07	66.9	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8
Fast [6]	07+12	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4
Faster [2]	07	69.9	70.0	80.6	70.1	57.3	49.9	78.2	80.4	82.0	52.2	75.3	67.2	80.3	79.8	75.0	76.3	39.1	68.3	67.3	81.1	67.6
Faster [2]	07+12	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
Faster [2]	07+12+COCO	78.8	84.3	82.0	77.7	68.9	65.7	88.1	88.4	88.9	63.6	86.3	70.8	85.9	87.6	80.1	82.3	53.6	80.4	75.8	86.6	78.9
SSD300	07	68.0	73.4	77.5	64.1	59.0	38.9	75.2	80.8	78.5	46.0	67.8	69.2	76.6	82.1	77.0	72.5	41.2	64.2	69.1	78.0	68.5
SSD300	07+12	74.3	75.5	80.2	72.3	66.3	47.6	83.0	84.2	86.1	54.7	78.3	73.9	84.5	85.3	82.6	76.2	48.6	73.9	76.0	83.4	74.0
SSD300	07+12+COCO	79.6	80.9	86.3	79.0	**76.2**	57.6	87.3	88.2	88.6	60.5	85.4	**76.7**	**87.5**	**89.2**	84.5	81.4	55.0	81.9	**81.5**	85.9	78.9
SSD512	07	71.6	75.1	81.4	69.8	60.8	46.3	82.6	84.7	84.1	48.5	75.0	67.4	82.3	83.9	79.4	76.6	44.9	69.9	69.1	78.1	71.8
SSD512	07+12	76.8	82.4	84.7	78.4	73.8	53.2	86.2	87.5	86.0	57.8	83.1	70.2	84.9	85.2	83.9	79.7	50.3	77.9	73.9	82.5	75.3
SSD512	07+12+COCO	**81.6**	**86.6**	**88.3**	**82.4**	76.0	**66.3**	**88.6**	**88.9**	**89.1**	**65.1**	**88.4**	73.6	86.5	88.9	**85.3**	**84.6**	**59.1**	**85.0**	80.4	**87.4**	**81.2**

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

SSD: Observations

- SSD performs worse than Faster R-CNN for **small**-scale objects. In SSD, small objects can only be detected in **higher** resolution layers (leftmost layers). But those layers contains **low-level features**, like edges or color patches, that are **less informative** for classification.

- **Accuracy** increases with the **number of default** boxes at the cost of **speed**.

- Multi-scale feature maps improve the detection of objects at different **scale**.

- Designing **better default** boxes will help accuracy.

- The **COCO** dataset has **smaller** objects. To improve accuracy, use smaller default boxes.

- SSD has **lower localization** error compared with R-CNN but **more classification** error dealing with similar categories. The higher classification errors are likely because we use the same BB to make multiple class predictions.

- SSD512 has better accuracy (2.5%) than SSD300 but run at 22 FPS instead of 59.

SSD: Conclusion

- SSD is a single-shot detector. It has no dedicated region proposal network and predicts the boundary boxes and the classes directly from feature maps in one single pass.

- To improve **accuracy**, SSD **introduces**:

  - **multi-scale** feature maps for object detection.

  - small convolutional filters to predict object classes and offsets to **default** boxes.

- SSD can be trained end-to-end for better accuracy. SSD makes more predictions and has a better coverage on location, scale and aspect ratios. With the improvements above, SSD can lower the input image resolution to 300×300 with a comparative accuracy. By removing the dedicated region proposal step and using lower resolution images, the model can run at real-time speed and still beats the accuracy of the state-of-the-art Faster R-CNN.

RetinaNet

Focal loss

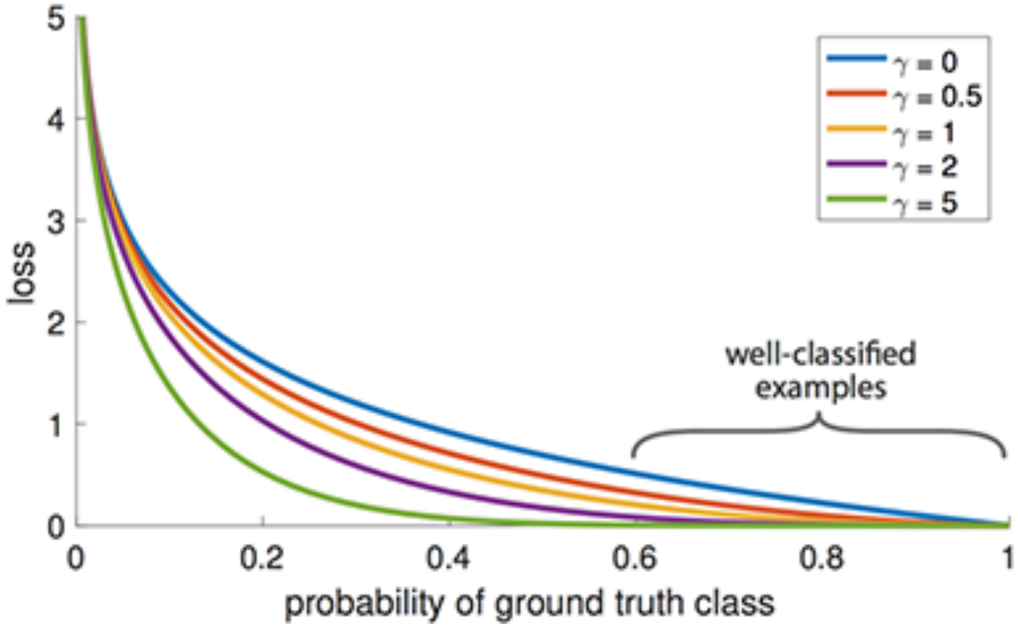
Paper: Focal Loss for Dense Object Detection

RetinaNet

- **Class imbalance** hurts performance.
- **SSD** resamples the **ratio** of the object class and background class during training so it will not be overwhelmed by image background.
- **Focal loss** (FL) adopts another approach to **reduce loss for well-trained classes**. So whenever the model is good at detecting background, it will reduce its loss and reemphasize the training on the object class.
- We start with the **cross-entropy** loss CE and we add a **weight** to de-emphasize the CE for high confidence class. For example, for γ = 0.5, the focal loss for well-classified examples will be pushed toward 0.

$$CE(p_t) = -\log(p_t)$$
$$FL(p_t) = -(1-p_t)^{\gamma}\log(p_t)$$

p_t is the predicted class probability for ground truth.
$\gamma > 0$



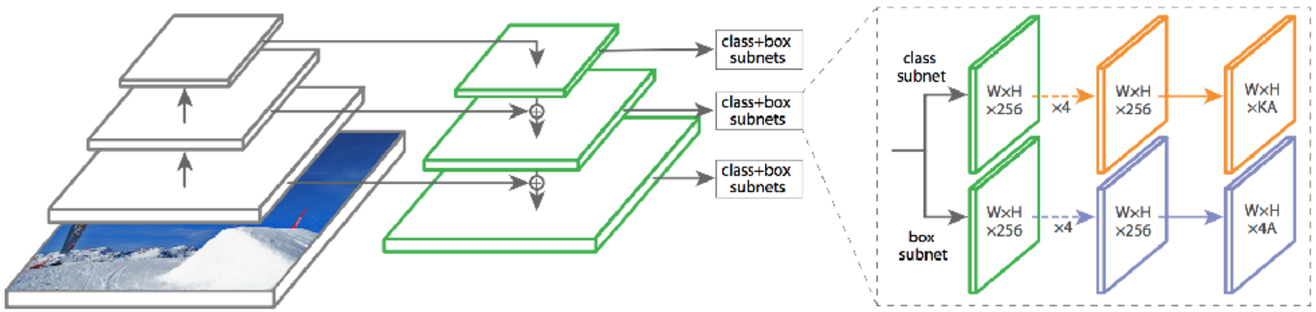
RetinaNet (2)

- The one-stage **RetinaNet** architecture build on ResNet, FPN (Feature Pyramid Network) and uses Focal loss.

- The FPN backbone on top of a the feedforward ResNet architecture generates a rich, multi-scale convolutional feature pyramid.

- To this backbone RetinaNet attaches two **subnetworks**, one for **classifying** anchor boxes and one for **regressing** from anchor boxes to ground-truth object boxes.

- The network **design** is intentionally **simple**, which enables this work to focus on a novel **focal loss** function that eliminates the **accuracy gap** between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN while running at faster speeds.



(a) ResNet (b) feature pyramid net (c) class subnet (top) (d) box subnet (bottom)

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2

Next time..

Segmentation