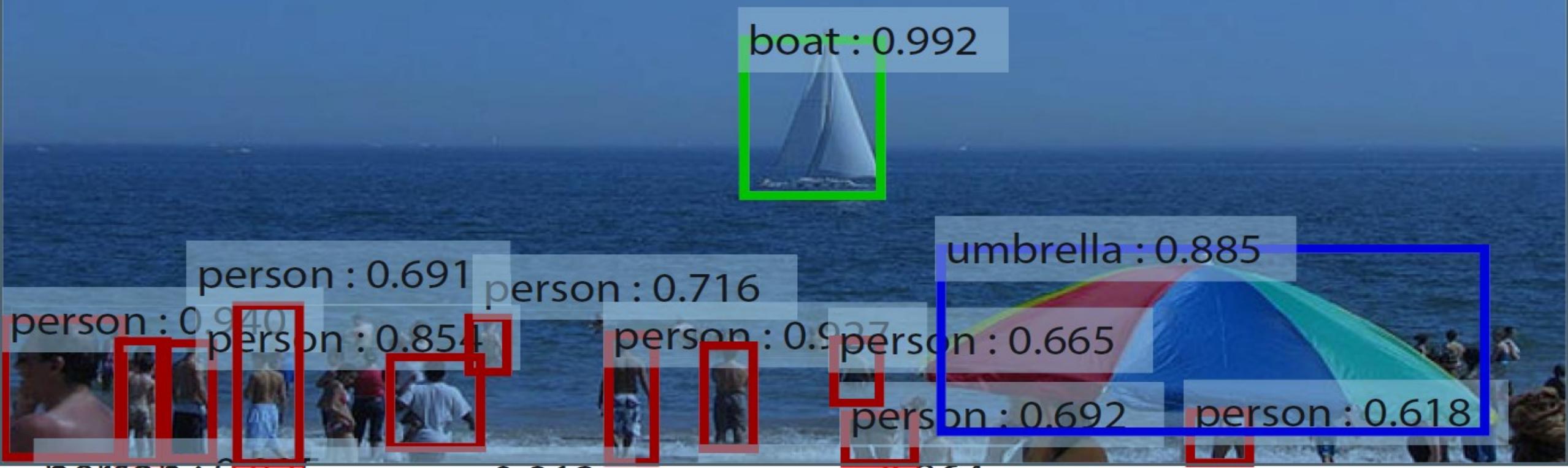


Object Detection



Computer Vision Task

Semantic Segmentation



GRASS, CAT,
TREE, SKY

No objects, just pixels

Classification + Localization



CAT

Single Object

Object Detection



DOG, DOG, CAT

Multiple Object

Instance Segmentation



DOG, DOG, CAT

[This image is CC0 public domain](#)

Classification + Localization

Classification: C classes

Input: Image

Output: Class label

Evaluation metric: Accuracy



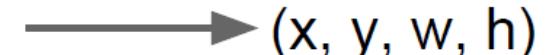
CAT

Localization:

Input: Image

Output: Box in the image (x, y, w, h)

Evaluation metric: Intersection over Union

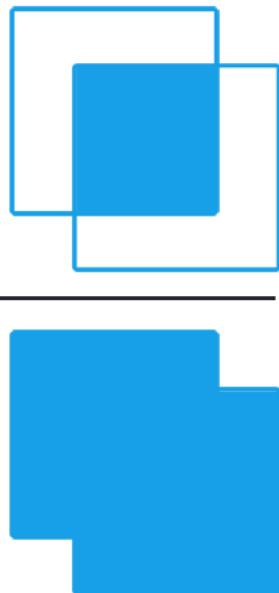


(x, y, w, h)

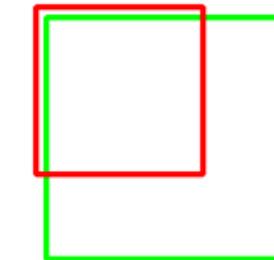
Classification + Localization: Do both

Intersection Over Union

$$\text{Jaccard Overlap} = \text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

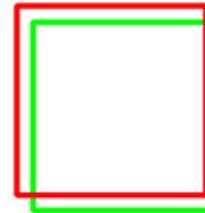


IoU: 0.4034



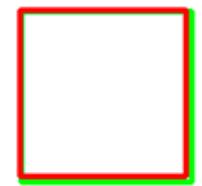
Poor

IoU: 0.7330



Good

IoU: 0.9264

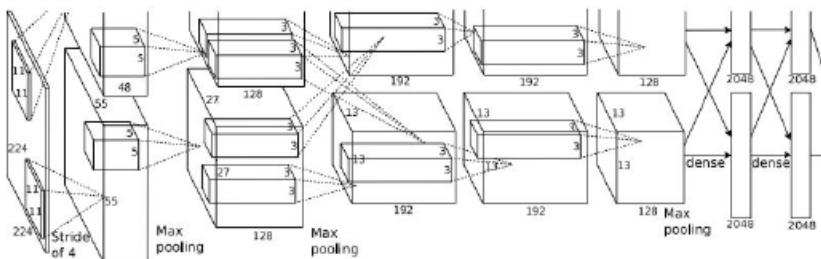


Excellent

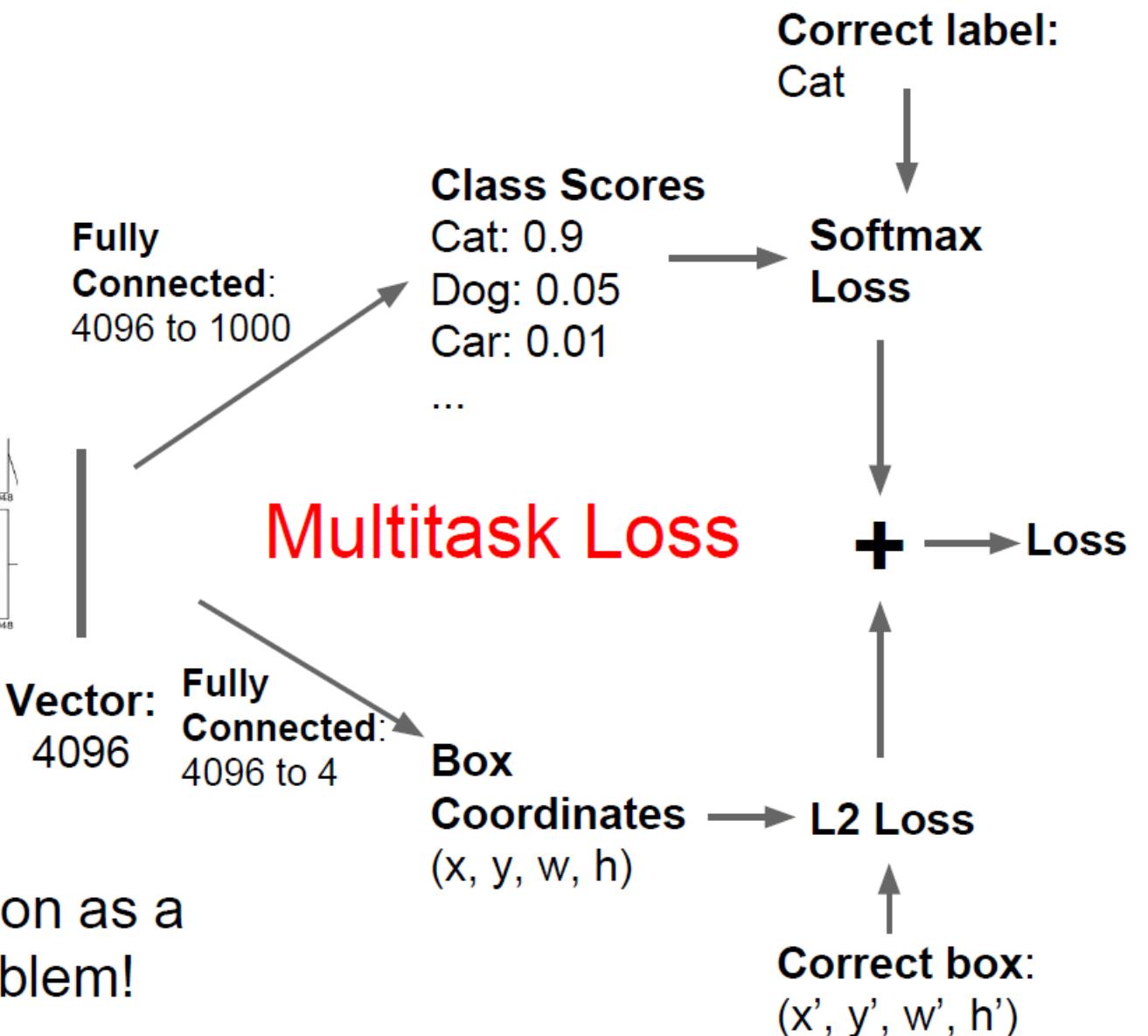
Classification + Localization



This image is CC0 public domain



Treat localization as a
regression problem!



Detection as Regression?



CAT, (x, y, w, h)

CAT, (x, y, w, h)

....

CAT (x, y, w, h)

= many numbers

Need variable sized outputs

R-CNN

1. Generating category independent region proposals
2. Extracting a fixed length feature vector from CNN
3. Class specific linear SVM

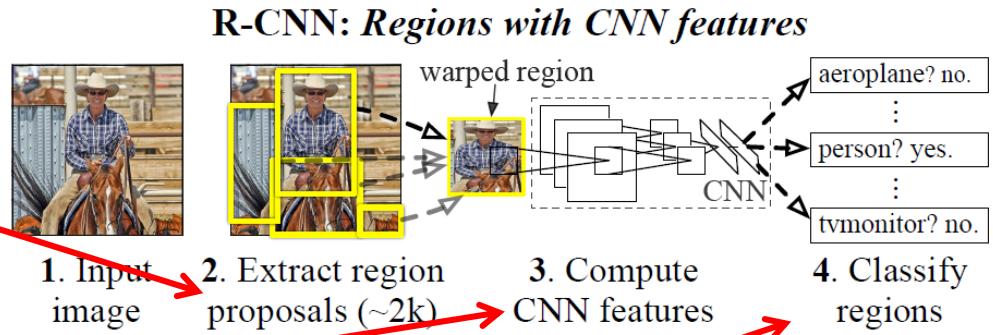


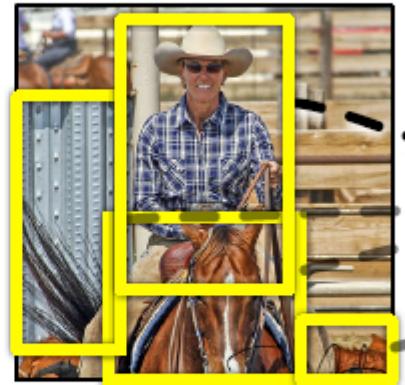
Figure 1: Object detection system overview. Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs. R-CNN achieves a mean average precision (mAP) of **53.7% on PASCAL VOC 2010**. For comparison, [39] reports 35.1% mAP using the same region proposals, but with a spatial pyramid and bag-of-visual-words approach. The popular deformable part models perform at 33.4%. On the 200-class **ILSVRC2013 detection dataset**, R-CNN's **mAP is 31.4%**, a large improvement over OverFeat [34], which had the previous best result at 24.3%.

R-CNN Architecture

R-CNN: *Regions with CNN features*

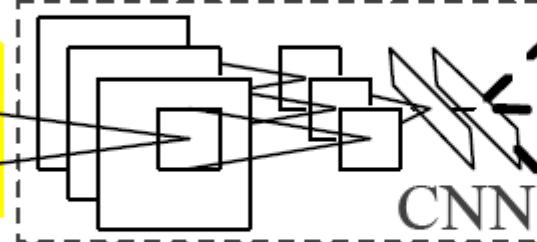


1. Input
image

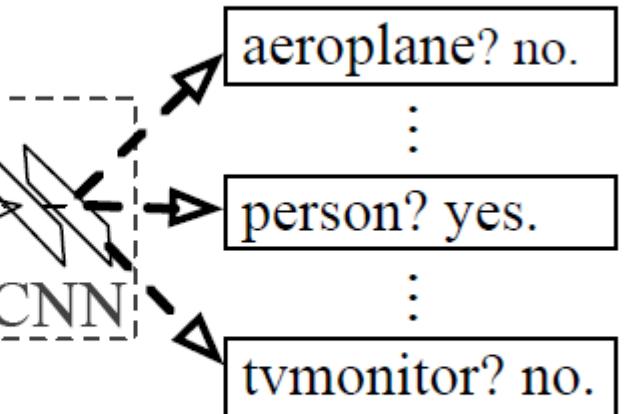


2. Extract region
proposals (~2k)

warped region



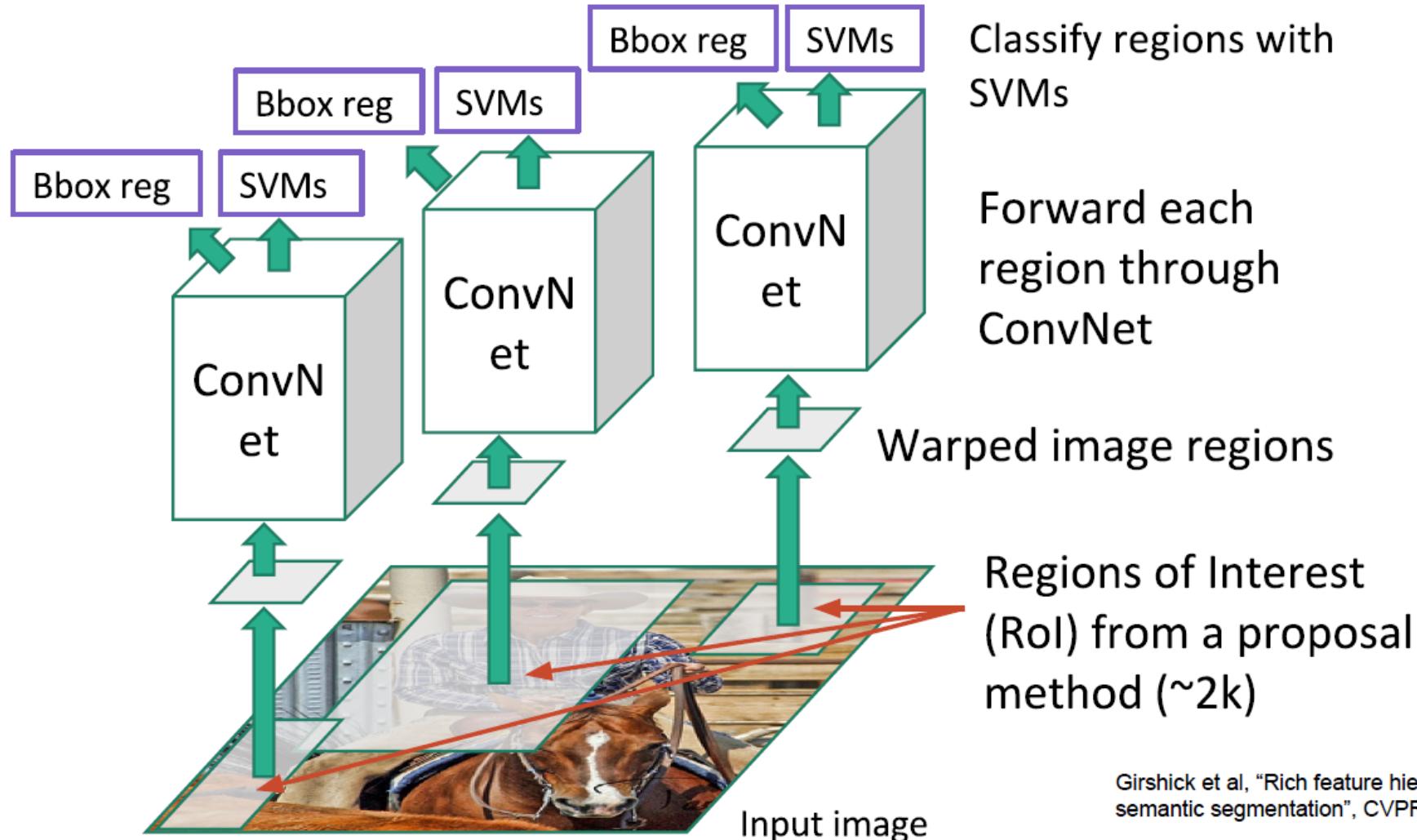
3. Compute
CNN features



4. Classify
regions

R-CNN

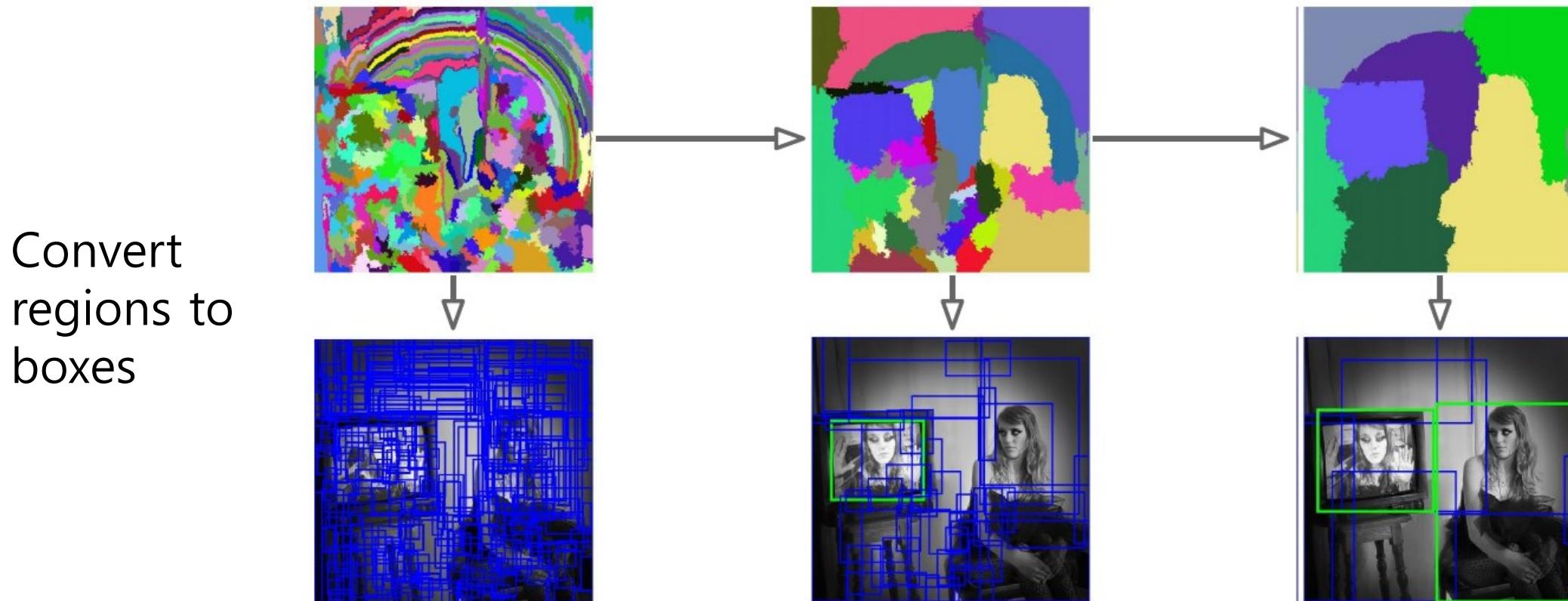
Linear Regression for bounding box offsets



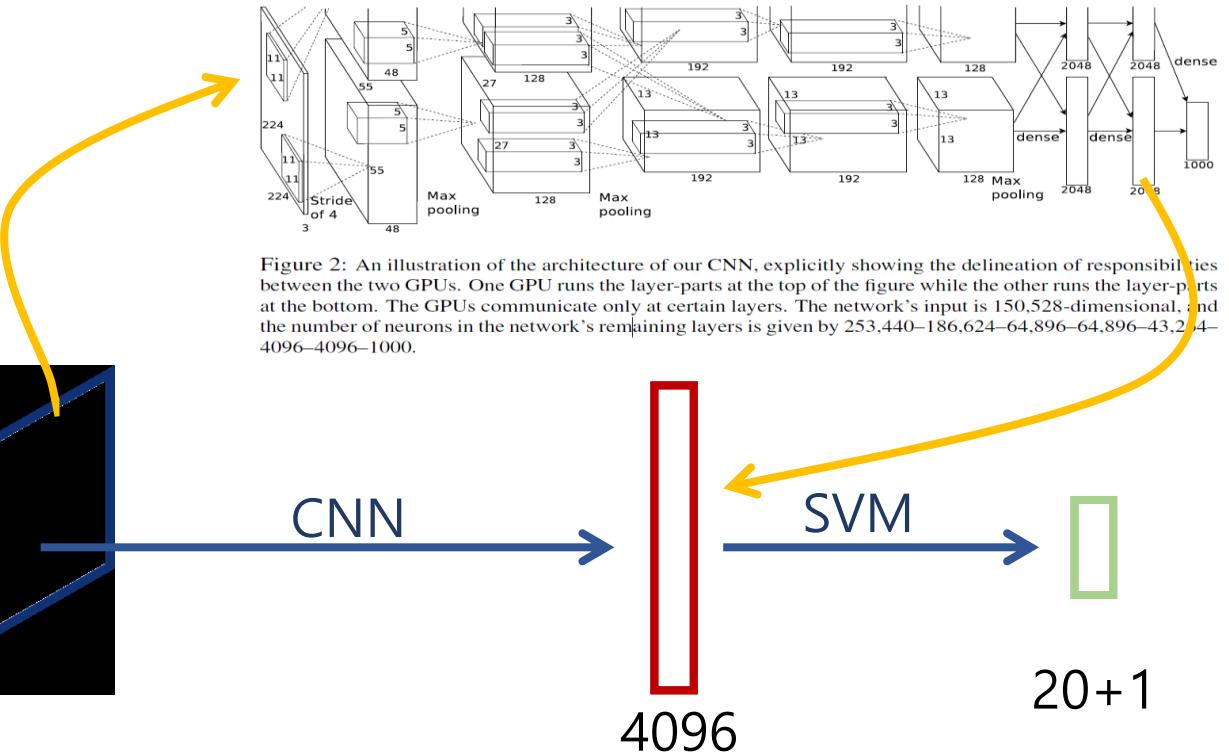
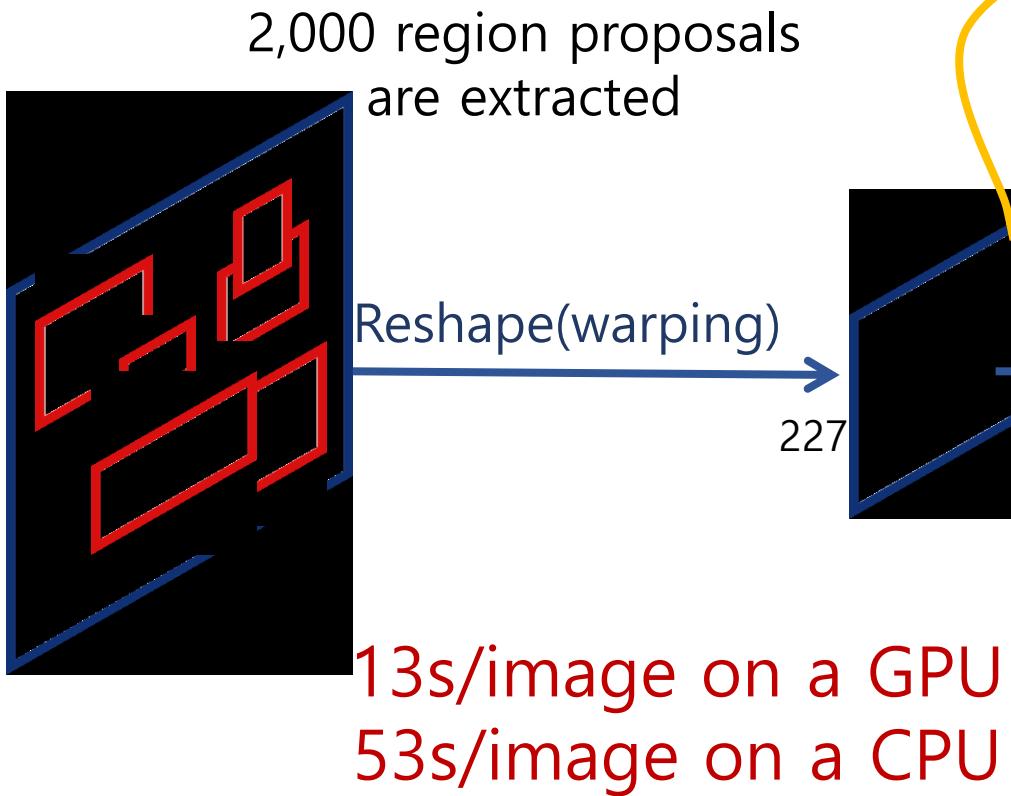
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

Region Proposals – Selective Search

- Bottom-up segmentation, merging regions at multiple scales



Test Time



Training(Region Proposal)

Positive : groundtruth

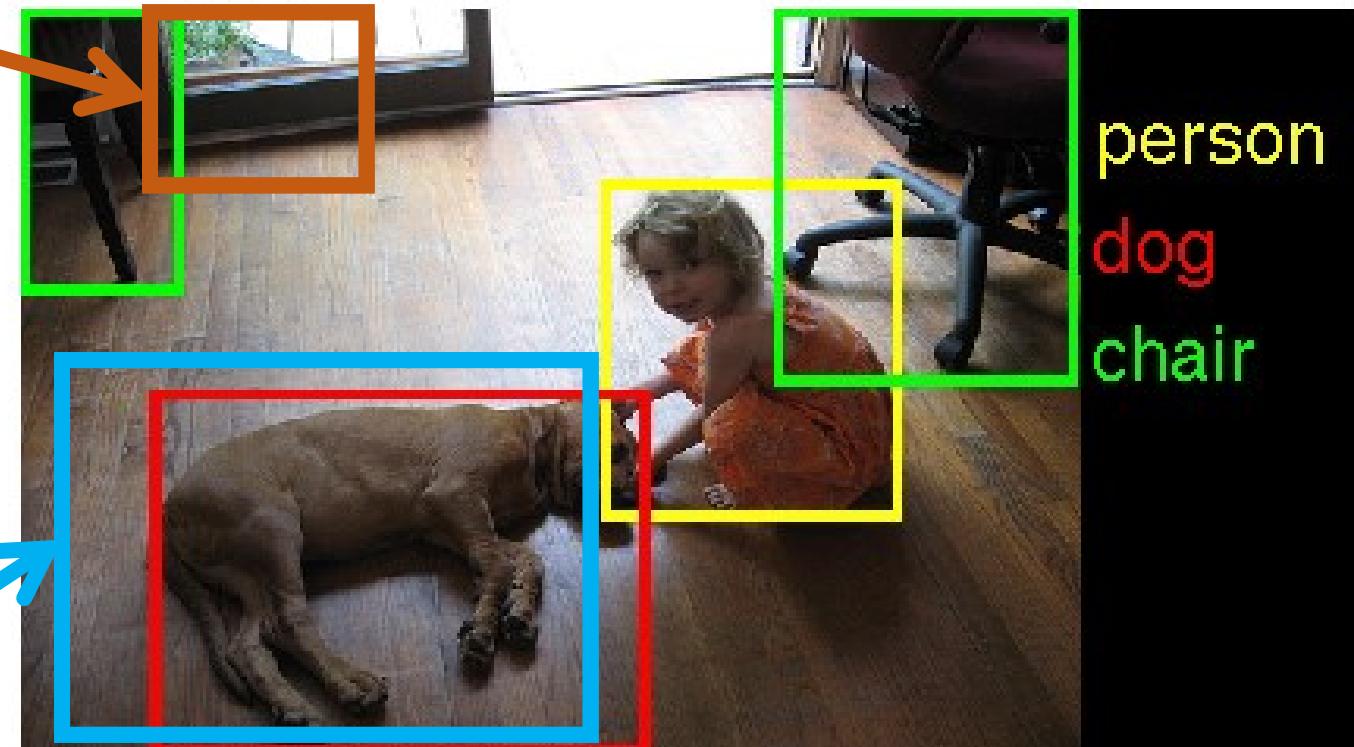
Negative (BG) : if $\text{IoU} \leq 0.3$

Training SVM

Fine-tuning

Positive (FG) : if $\text{IoU} \geq 0.5$

Negative (BG) : else



Bounding-Box Regression

$P^i = (P_x^i, P_y^i, P_w^i, P_h^i)$ specifies the pixel coordinates of the center of proposal P^i 's bounding box together with P^i 's width and height in pixels

$G = (G_x, G_y, G_w, G_h)$ means the ground-truth bounding box

$$\hat{G}_x = P_w d_x(P) + P_x \quad (1)$$

$$\hat{G}_y = P_h d_y(P) + P_y \quad (2)$$

$$\hat{G}_w = P_w \exp(d_w(P)) \quad (3)$$

$$\hat{G}_h = P_h \exp(d_h(P)). \quad (4)$$

Bounding-Box Regression

Each function $d_\star(P)$ (where \star is one of x, y, h, w) is modeled as a linear function of the pool₅ features of proposal P , denoted by $\phi_5(P)$. (The dependence of $\phi_5(P)$ on the image data is implicitly assumed.) Thus we have

$d_\star(P) = \mathbf{w}_\star^T \phi_5(P)$ where \mathbf{w}_\star is a vector of learnable model parameters. We learn \mathbf{w}_\star by optimizing the regularized least squares objective (ridge regression):

$$\mathbf{w}_\star = \underset{\hat{\mathbf{w}}_\star}{\operatorname{argmin}} \sum_i^N (t_\star^i - \hat{\mathbf{w}}_\star^T \phi_5(P^i))^2 + \lambda \|\hat{\mathbf{w}}_\star\|^2. \quad (5)$$

$$\hat{G}_x = P_w d_x(P) + P_x \quad (1)$$

$$\hat{G}_y = P_h d_y(P) + P_y \quad (2)$$

$$\hat{G}_w = P_w \exp(d_w(P)) \quad (3)$$

$$\hat{G}_h = P_h \exp(d_h(P)). \quad (4)$$

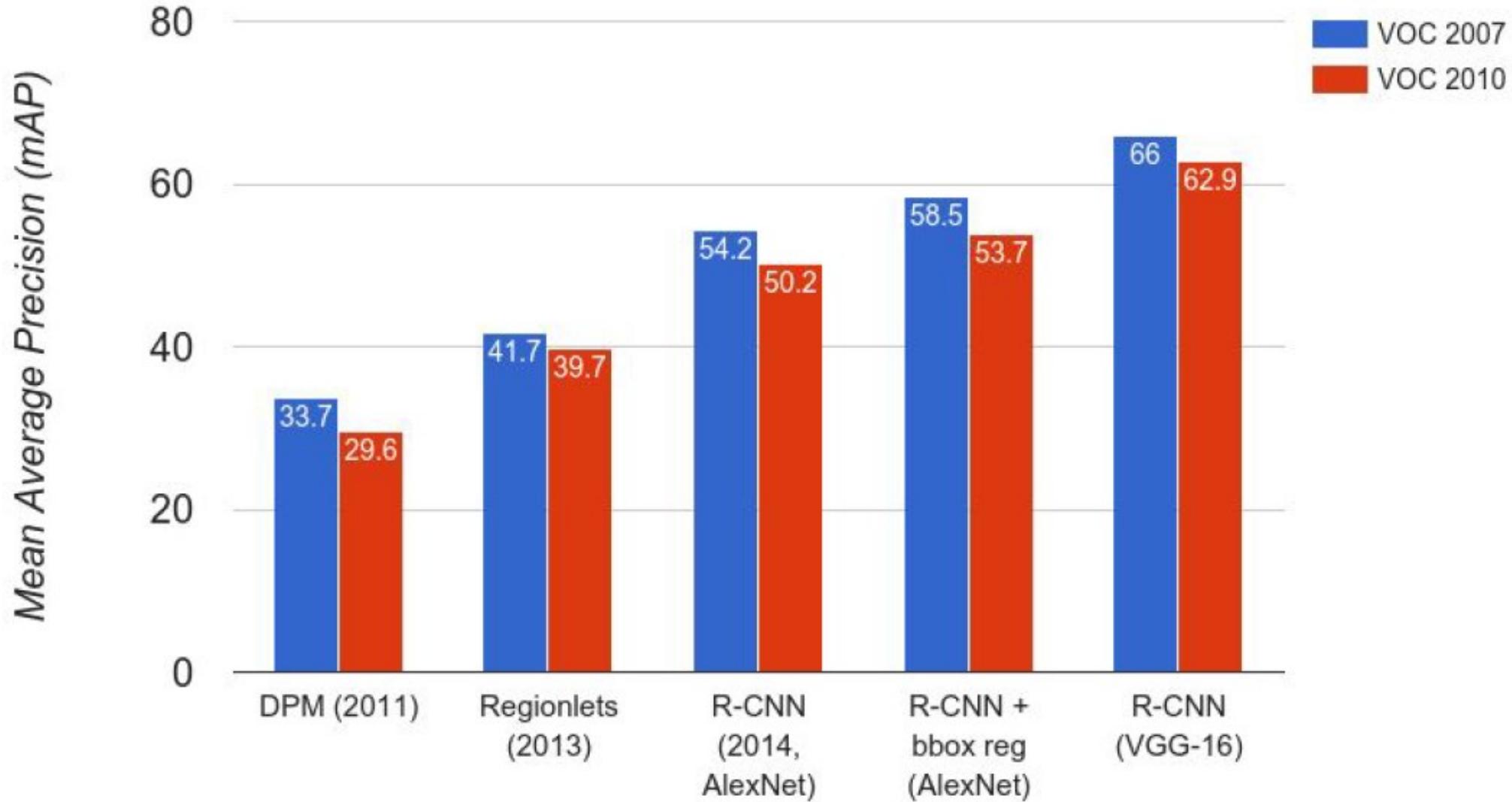
$$t_x = (G_x - P_x)/P_w \quad (6)$$

$$t_y = (G_y - P_y)/P_h \quad (7)$$

$$t_w = \log(G_w/P_w) \quad (8)$$

$$t_h = \log(G_h/P_h). \quad (9)$$

Results



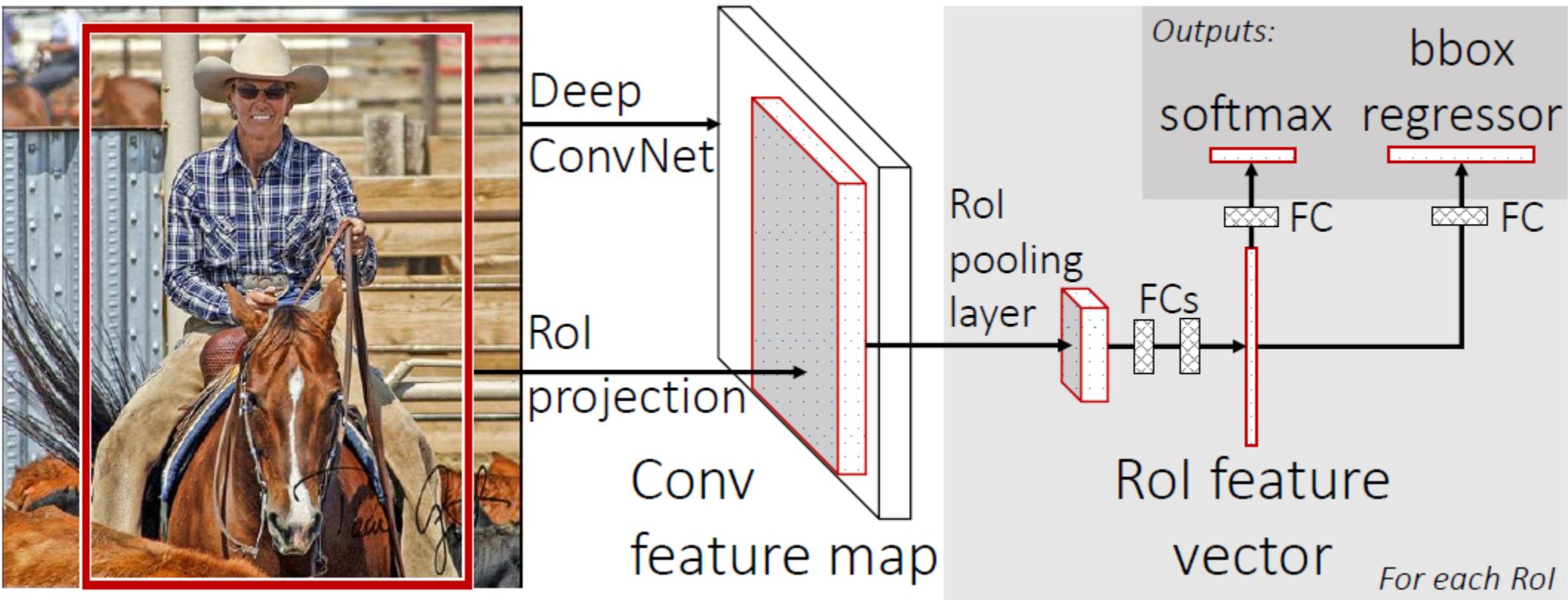
Problems of R-CNN

- **Slow at test-time:** need to run full forward path of CNN for each region proposal
 - 13s/image on a GPU(K40)
 - 53s/image on a CPU
- **SVM and regressors are post-hoc:** CNN features not updated in response to SVMs and regressors
- **Complex multistage training pipeline** (84 hours using K40 GPU)
 - Fine-tune network with softmax classifier(log loss)
 - Train post-hoc linear SVMs(hinge loss)
 - Train post-hoc bounding-box regressions(squared loss)

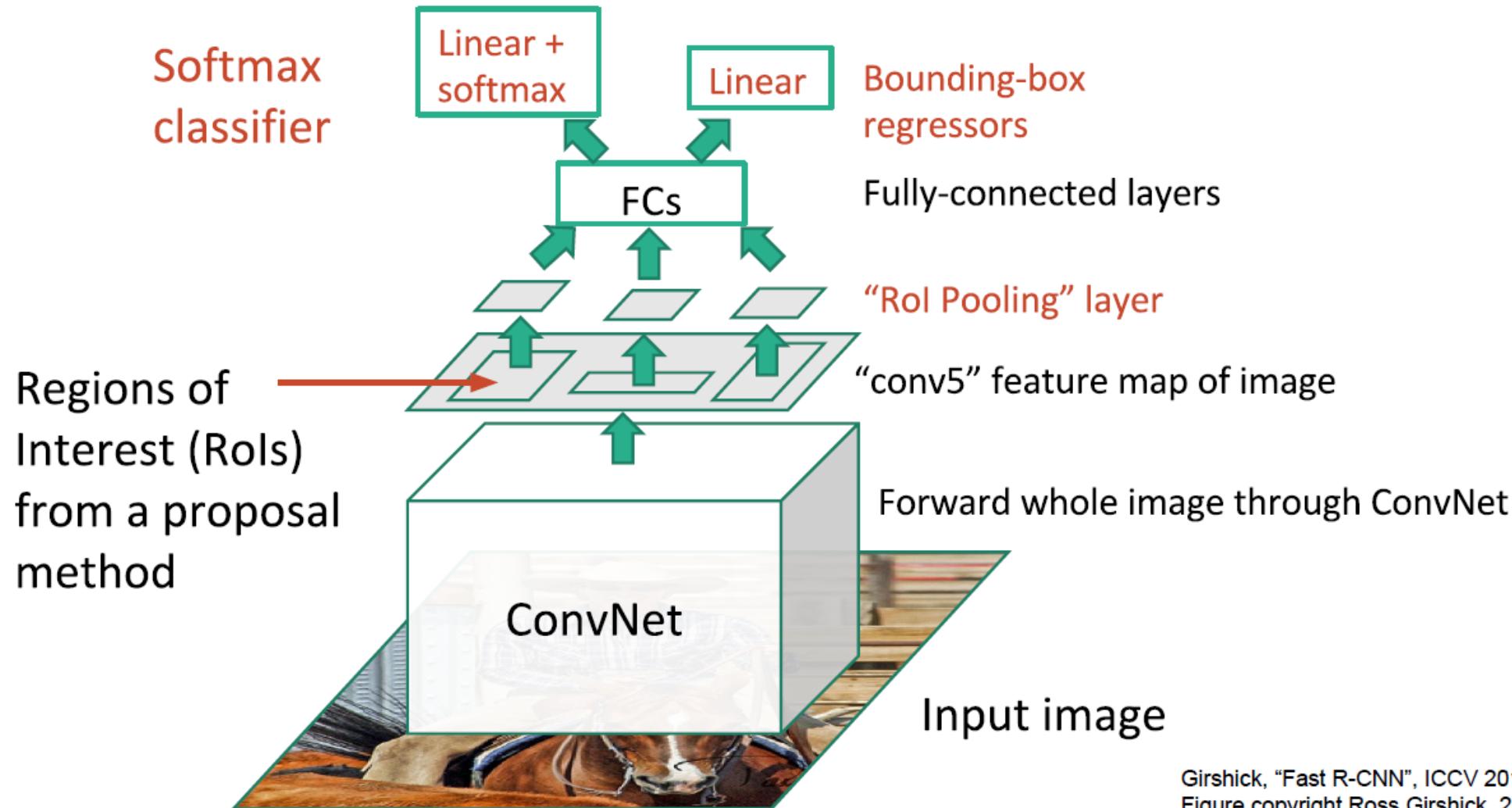
Fast R-CNN

- Fix most of what's wrong with R-CNN and SPP-net
- Train the detector in a **single stage, end-to-end**
 - No caching features to disk
 - No post hoc training steps
- Train **all layers** of the network

Fast R-CNN Architecture

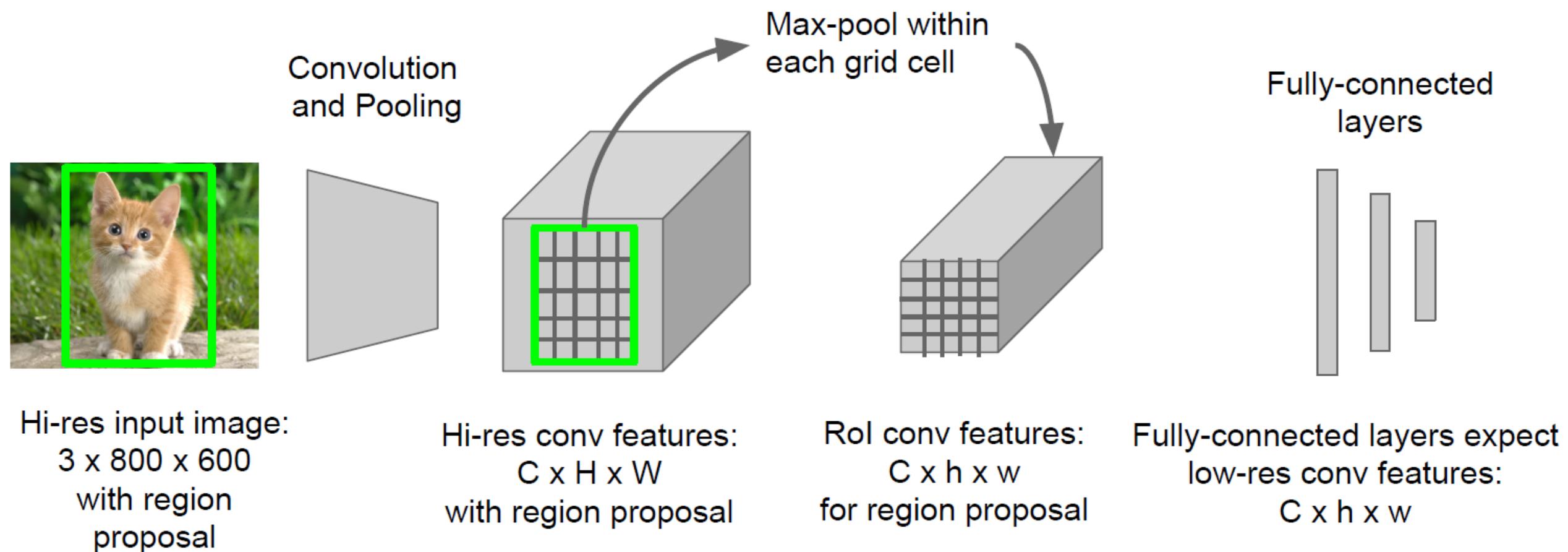


Fast R-CNN



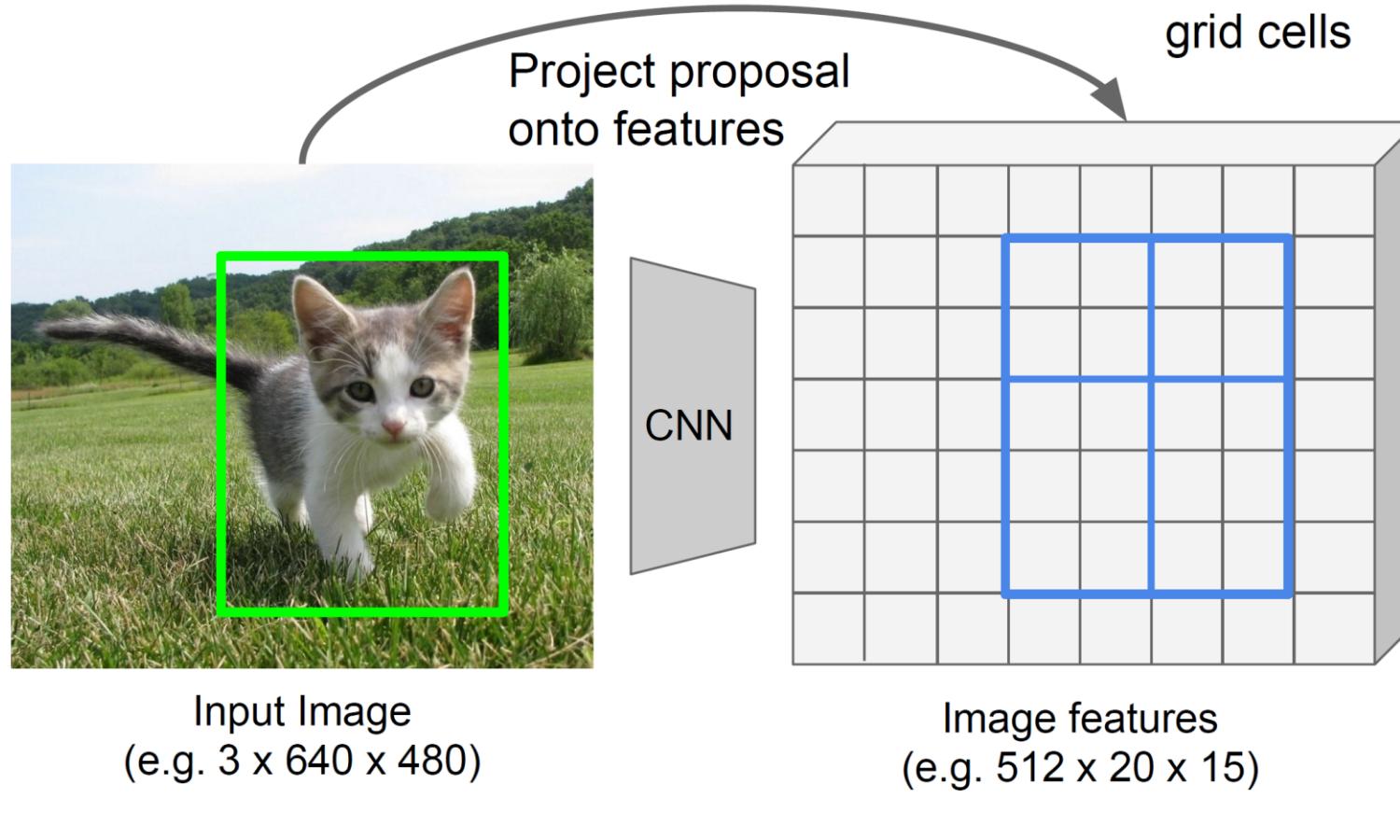
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015;

RoI Pooling



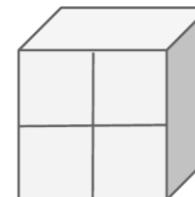
RoI Pooling

Cropping Features: RoI Pool



Divide into 2×2 grid of (roughly) equal subregions

Max-pool within each subregion



Region features
(here $512 \times 2 \times 2$;
In practice e.g $512 \times 7 \times 7$)

Region features always the same size even if input regions have different sizes!

Training & Testing

1. Takes an input and a set of bounding boxes
 2. Generate convolutional feature maps
 3. For each bbox, get a fixed-length feature vector from RoI pooling layer
 4. Outputs have two information
 - $K+1$ class labels
 - Bounding box locations
- Loss function

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v)$$

True box coordinates
Predicted box coordinates
True class scores
Predicted class scores
Log loss
Smooth L1 loss

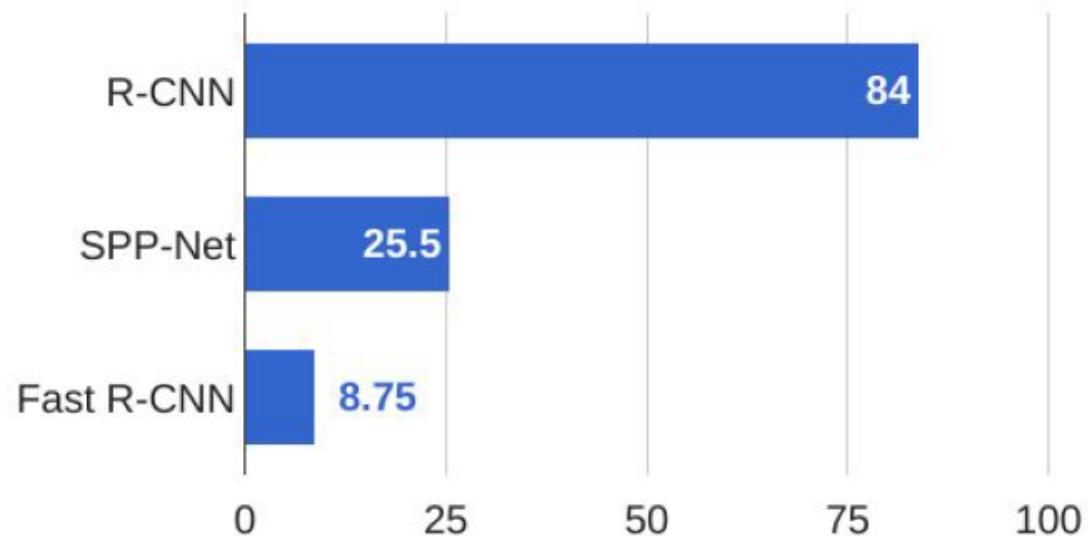
$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{x, y, w, h\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

in which

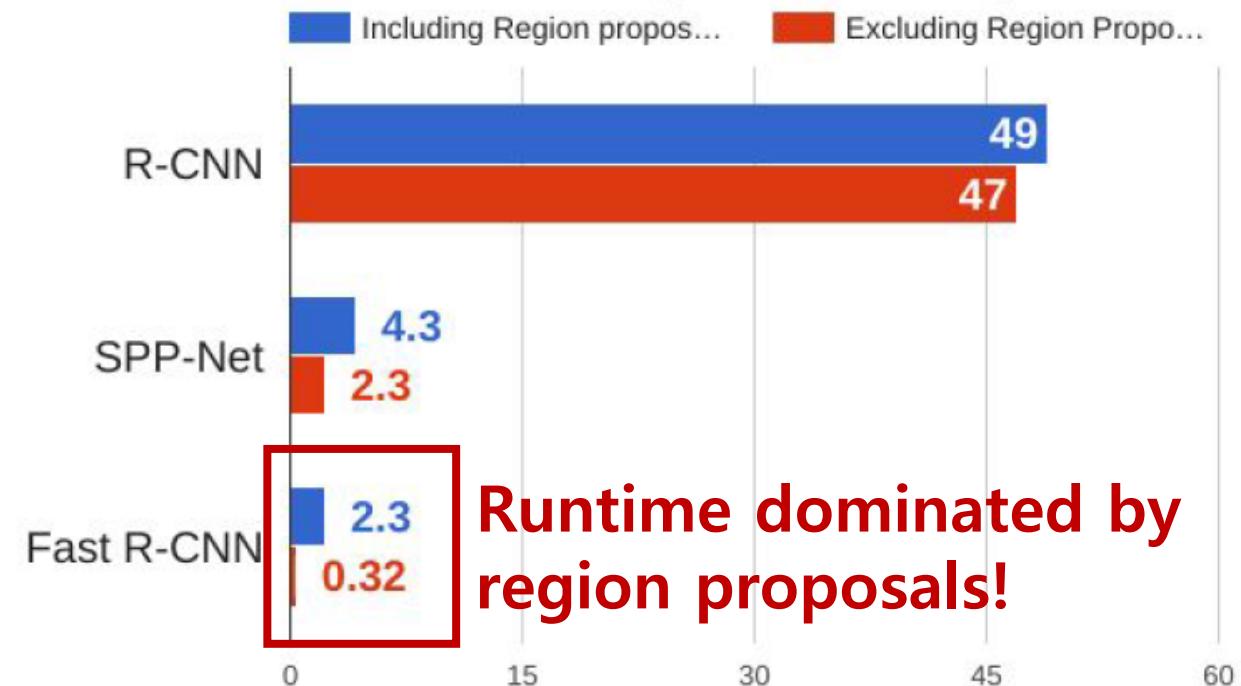
$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

R-CNN vs SPP-net vs Fast R-CNN

Training time (Hours)



Test time (seconds)

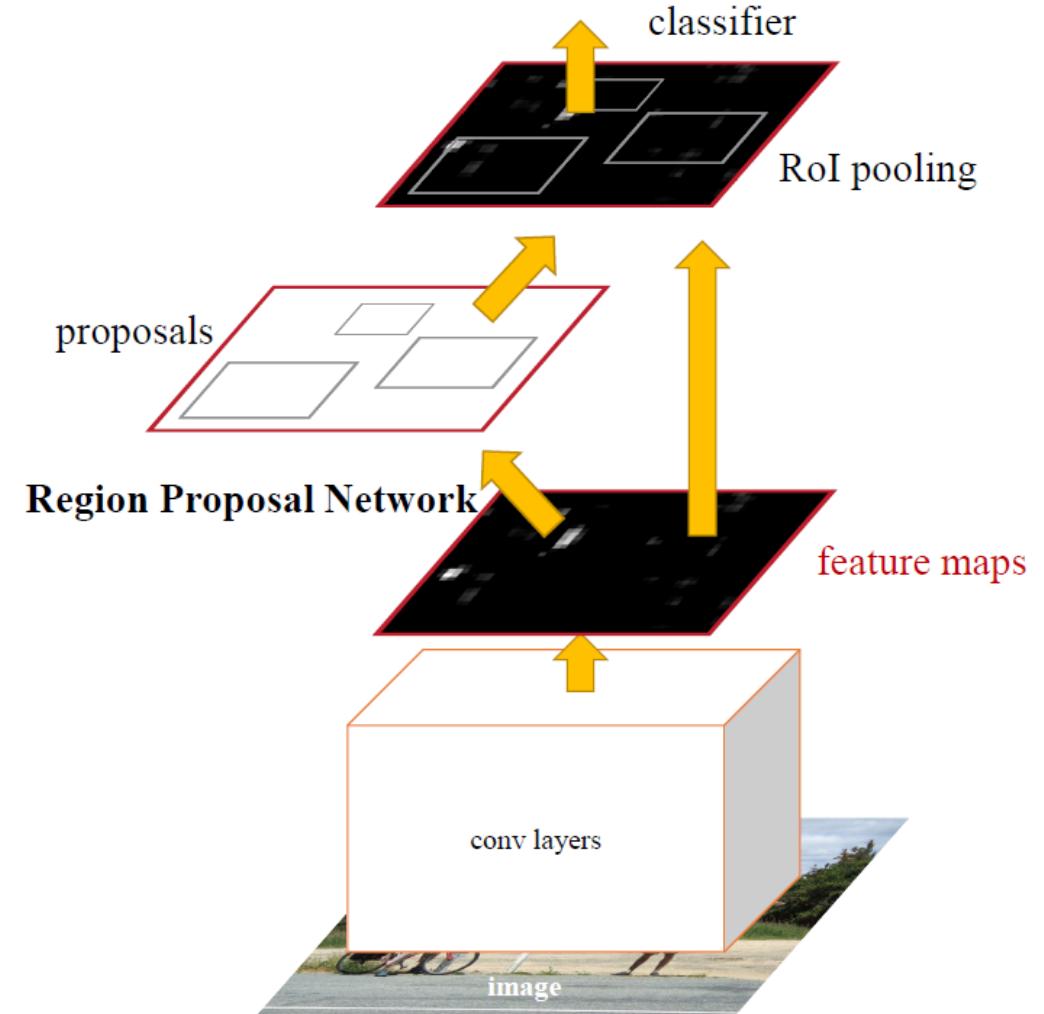


Problems of Fast R-CNN

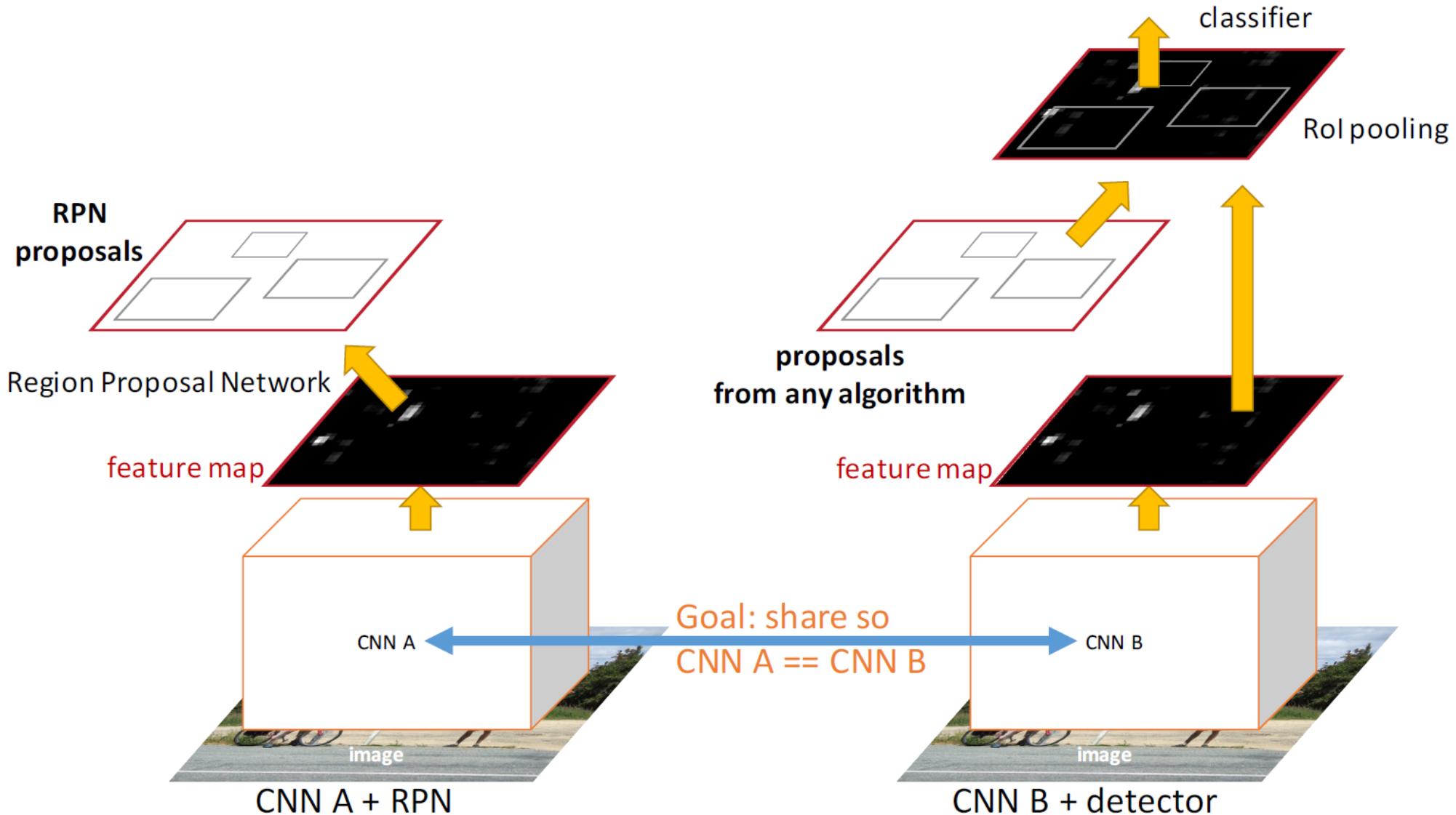
- Out-of-network region proposals are the test-time computational bottleneck
- Is it fast enough??

Faster R-CNN(RPN + Fast R-CNN)

- Insert a Region Proposal Network (RPN) after the last convolutional layer → using GPU!
- RPN trained to produce region proposals directly; no need for external region proposals
- After RPN, use RoI Pooling and an upstream classifier and bbox regressor just like Fast R-CNN

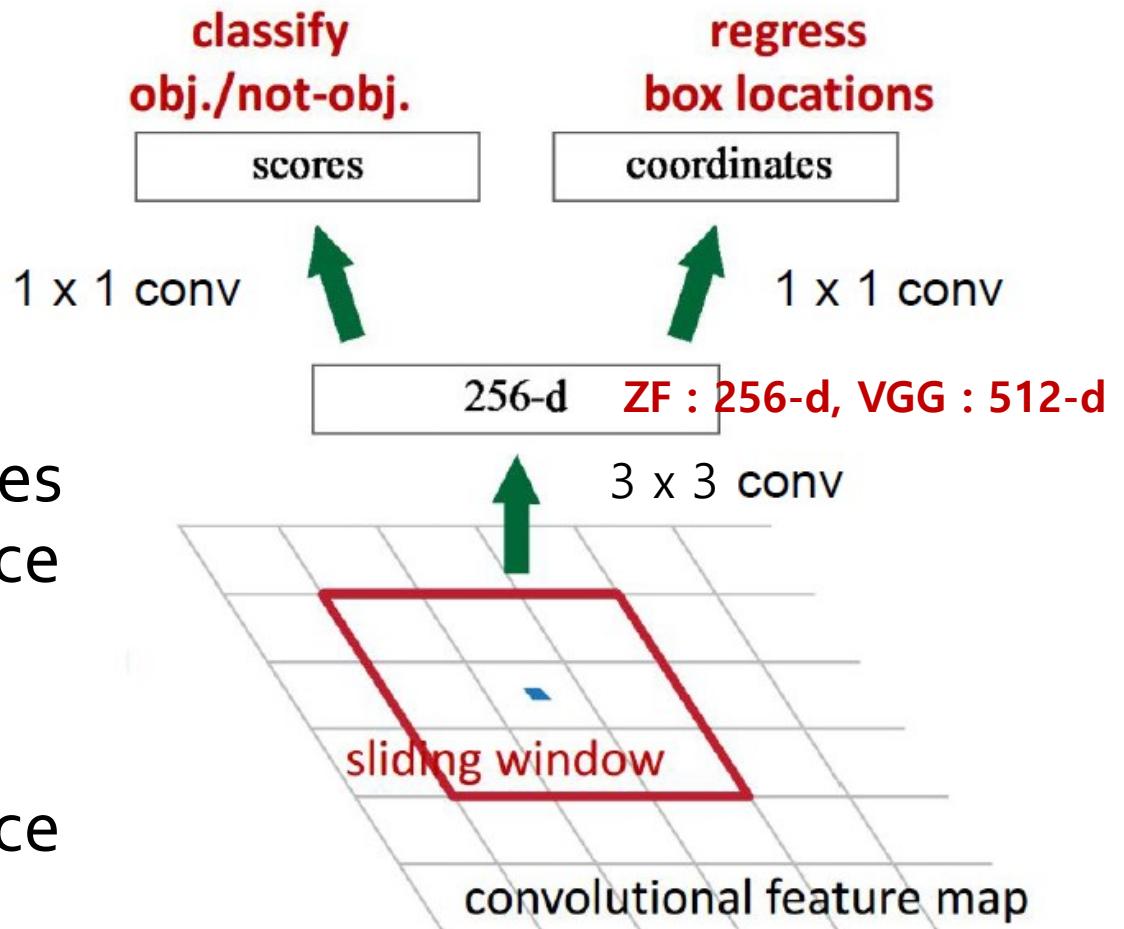


Training Goal : Share Features



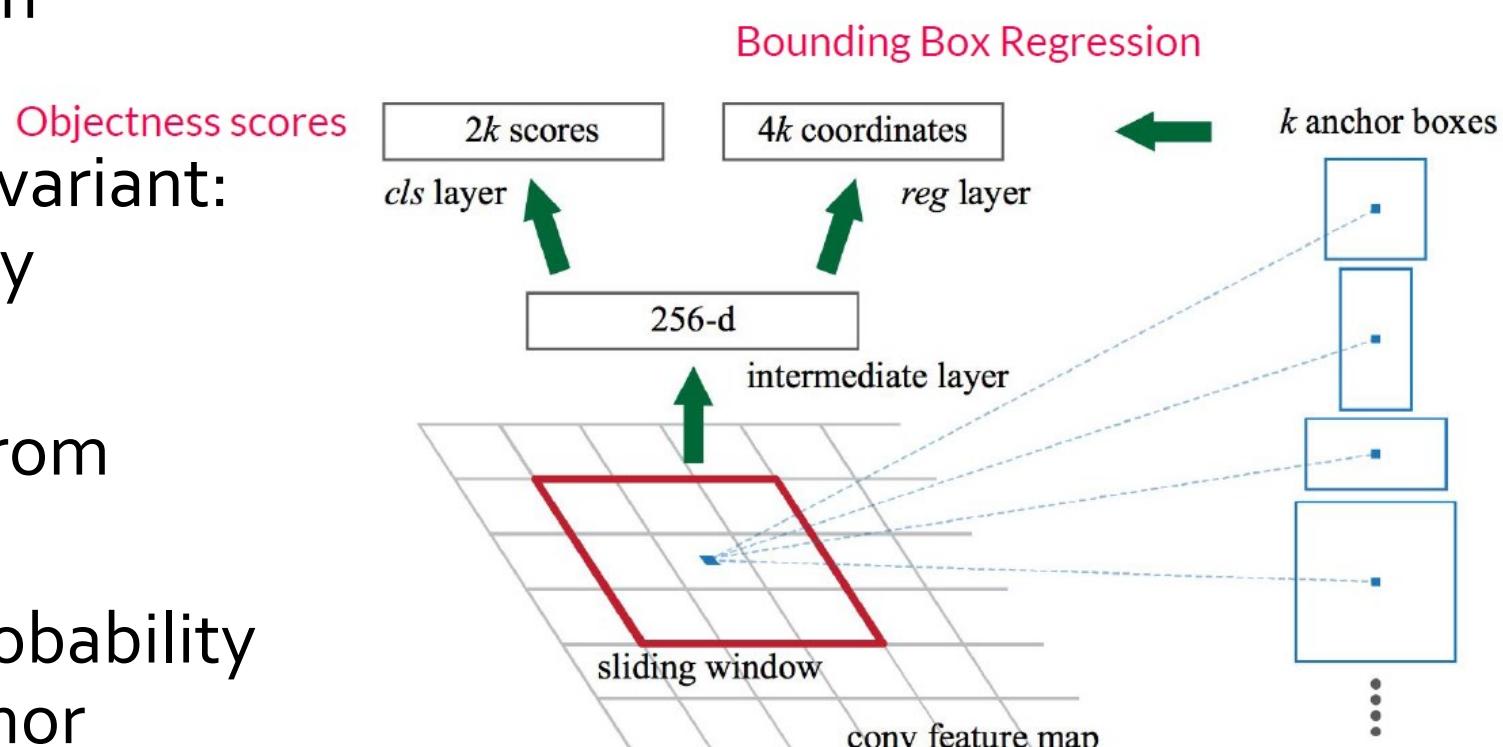
RPN

- Slide a small window on the feature map
- Build a small network for
 - Classifying object or not-object
 - Regressing bbox locations
- Position of the sliding window provides localization information with reference to the image
- Box regression provides finer localization information with reference to this sliding window



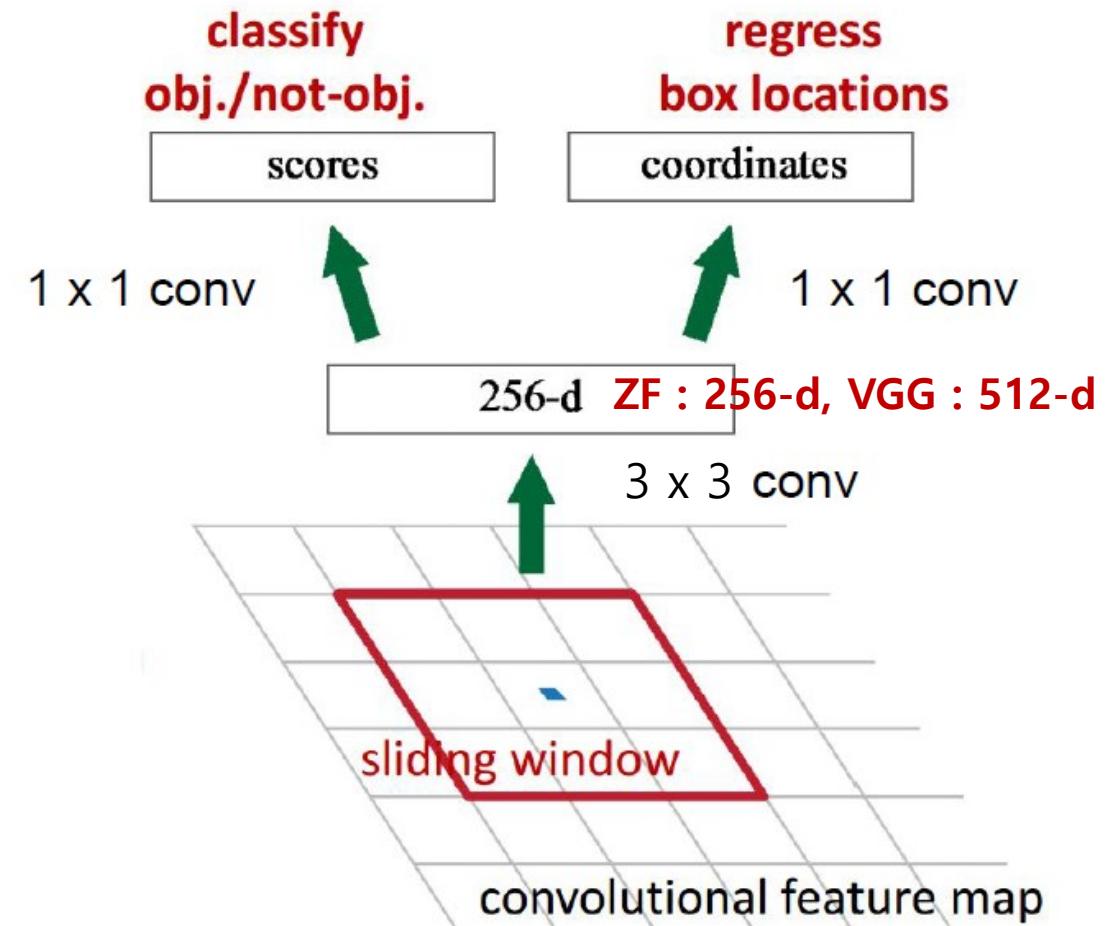
RPN

- Use k anchor boxes at each location
- Anchors are translation invariant: use the same ones at every location
- Regression gives offsets from anchor boxes
- Classification gives the probability that each (regressed) anchor shows an object



RPN(Fully Convolutional Network)

- Intermediate Layer – 256(or 512) 3×3 filter, stride 1, padding 1
- Cls layer – 18(9×2) 1×1 filter, stride 1, padding 0
- Reg layer – 36(9×4) 1×1 filter, stride 1, padding 0

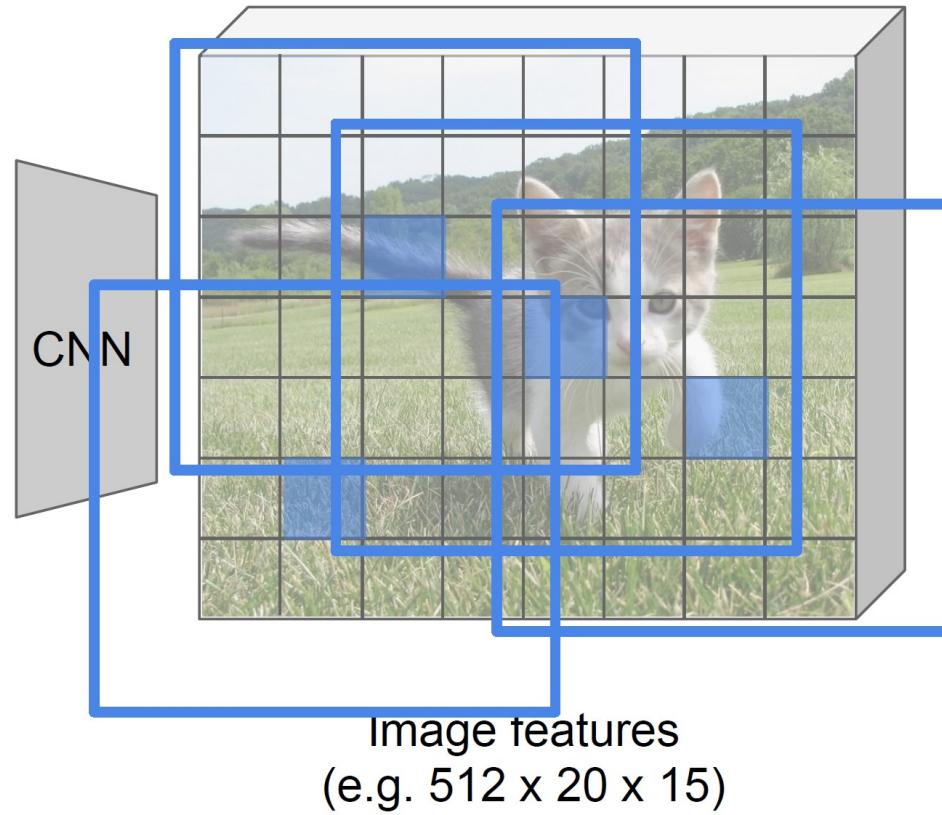


RPN

Region Proposal Network



Input Image
(e.g. $3 \times 640 \times 480$)



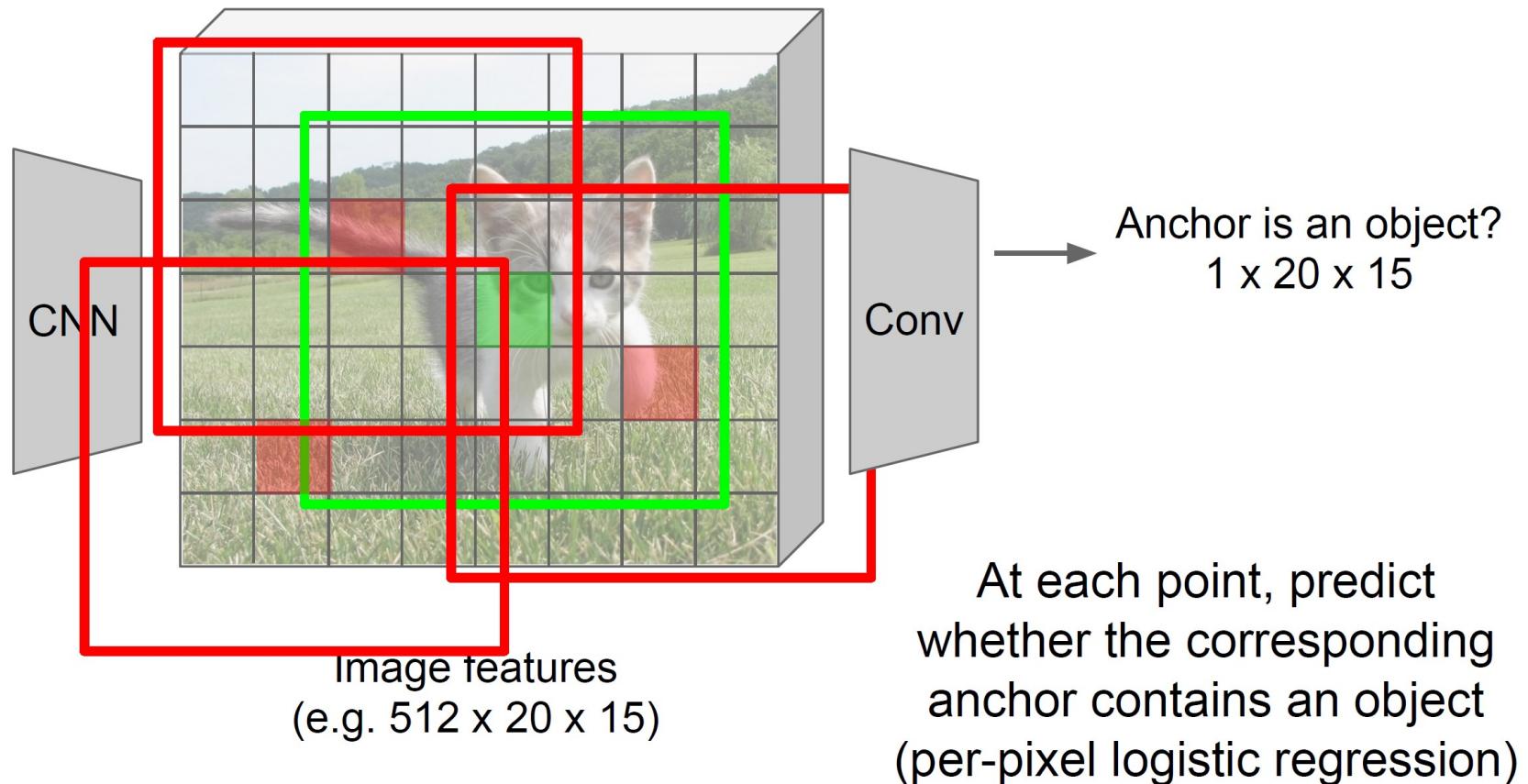
Imagine an **anchor box**
of fixed size at each
point in the feature map

RPN

Region Proposal Network



Input Image
(e.g. $3 \times 640 \times 480$)

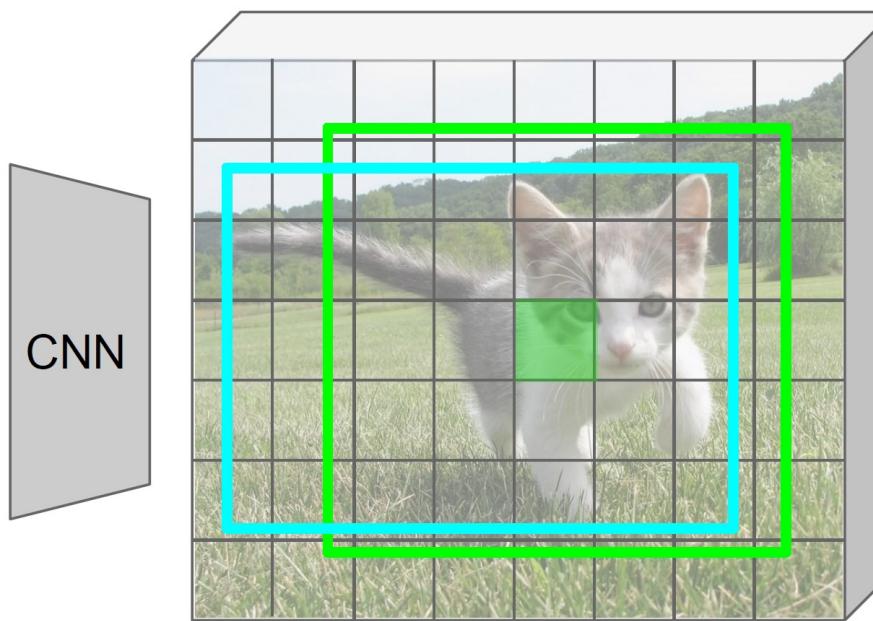


RPN

Region Proposal Network



Input Image
(e.g. $3 \times 640 \times 480$)



Imagine an **anchor box** of fixed size at each point in the feature map

Anchor is an object?
 $1 \times 20 \times 15$

Box transforms
 $4 \times 20 \times 15$

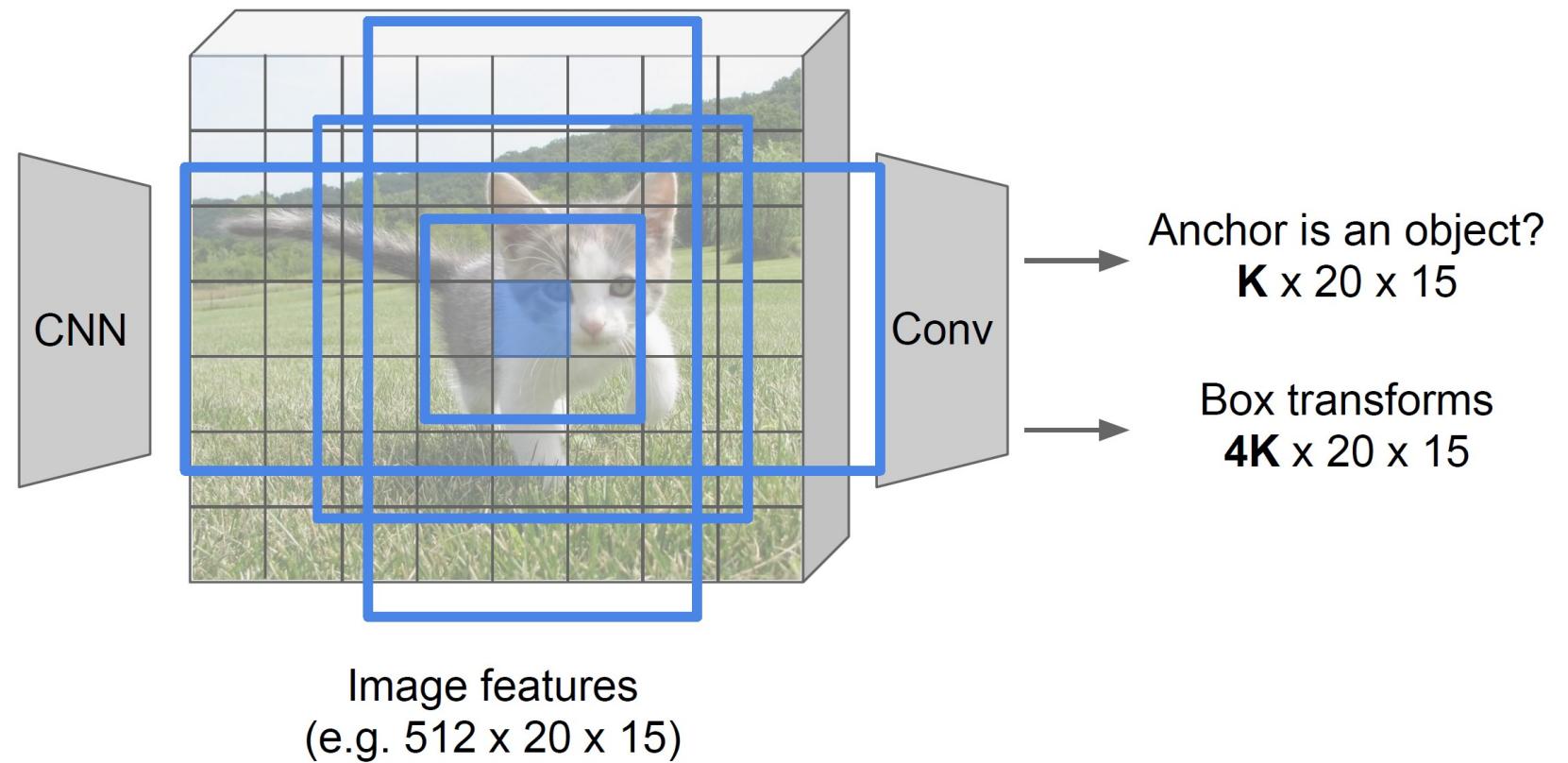
For positive boxes, also predict a transformation from the anchor to the ground-truth box (regress 4 numbers per pixel)

RPN

Region Proposal Network



Input Image
(e.g. $3 \times 640 \times 480$)



Anchors as references

- Anchors: pre-defined reference boxes
- Multi-scale/size anchors:
 - Multiple anchors are used at each position:
 - 3 scale(128x128, 256x256, 512x512) and 3 aspect ratios(2:1, 1:1, 1:2) yield 9 anchors
 - Each anchor has its own prediction function
 - Single-scale features, multi-scale predictions

Positive/Negative Samples

- An anchor is **labeled as positive** if
 - The anchor is the one with **highest IoU** overlap with a ground-truth box
 - The anchor has an IoU overlap with a ground-truth box **higher than 0.7**
- **Negative labels** are assigned to anchors with **IoU lower than 0.3** for all ground-truth boxes
- **50%/50%** ratio of positive/negative anchors in a minibatch

RPN Loss Function

i = anchor index in minibatch

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

Annotations:

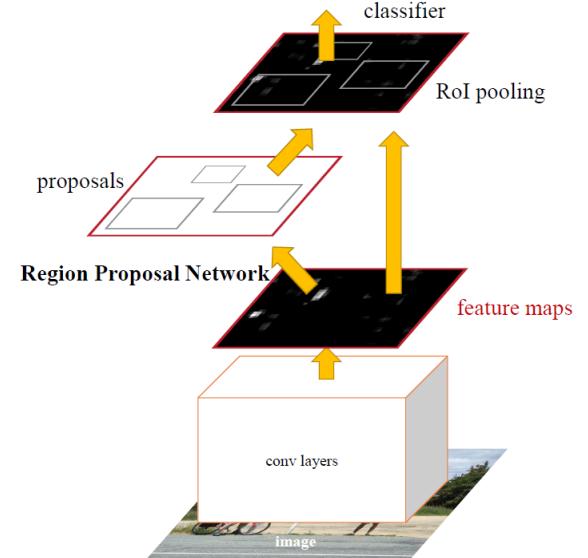
- Coordinates of the predicted bounding box for anchor i (blue double-headed arrow)
- Predicted probability of being an object for anchor i (blue double-headed arrow)
- Log loss (purple arrow pointing to the first term)
- Ground truth objectness label (red arrow pointing to p_i^*)
- Smooth L1 loss (purple arrow pointing to the second term)
- True box coordinates (red arrow pointing to t_i^*)
- λ (red circle)

N_{cls} = Number of anchors in minibatch (~ 256)

N_{reg} = Number of anchor locations (~ 2400)

In practice $\lambda = 10$, so that both terms are roughly equally balanced

4-Step Alternating Training



Let M0 be an ImageNet pre-trained network

1. train_rpn(**M0**) → M1 # Train an RPN initialized from M0, get M1
2. generate_proposals(M1) → P1 # Generate training proposals P1 using RPN M1
3. train_fast_rcnn(**M0**, P1) → M2 # Train Fast R-CNN M2 on P1 initialized from M0
4. train_rpn_frozen_conv(**M2**) → M3 # Train RPN M3 from M2 *without* changing conv layers
5. generate_proposals(M3) → P2
6. train_fast_rcnn_frozen_conv(M3, P2) → M4 # Conv layers are shared with RPN M3
7. return add_rpn_layers(M4, M3.RPN) # Add M3's RPN layers to Fast R-CNN M4

Results

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	0.2 seconds
(Speedup)	1x	25x	250x
mAP (VOC 2007)	66.0	66.9	69.9

Table 5: **Timing** (ms) on a K40 GPU, except SS proposal is evaluated in a CPU. “Region-wise” includes NMS, pooling, fully-connected, and softmax layers. See our released code for the profiling of running time.

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	10	47	198	5 fps
ZF	RPN + Fast R-CNN	31	3	25	59	17 fps

Experiments

Table 3: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: “07”: VOC 2007 trainval, “07+12”: union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. \dagger : this number was reported in [2]; using the repository provided by this paper, this result is higher (68.1).

method	# proposals	data	mAP (%)
SS	2000	07	66.9 \dagger
SS	2000	07+12	70.0
RPN+VGG, unshared	300	07	68.5
RPN+VGG, shared	300	07	69.9
RPN+VGG, shared	300	07+12	73.2
RPN+VGG, shared	300	COCO+07+12	78.8

Table 4: Detection results on **PASCAL VOC 2012 test set**. The detector is Fast R-CNN and VGG-16. Training data: “07”: VOC 2007 trainval, “07++12”: union set of VOC 2007 trainval+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. \dagger : <http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html>. \ddagger : <http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html>. \S : <http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html>.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared \dagger	300	12	67.0
RPN+VGG, shared \ddagger	300	07++12	70.4
RPN+VGG, shared \S	300	COCO+07++12	75.9

Is It Enough?

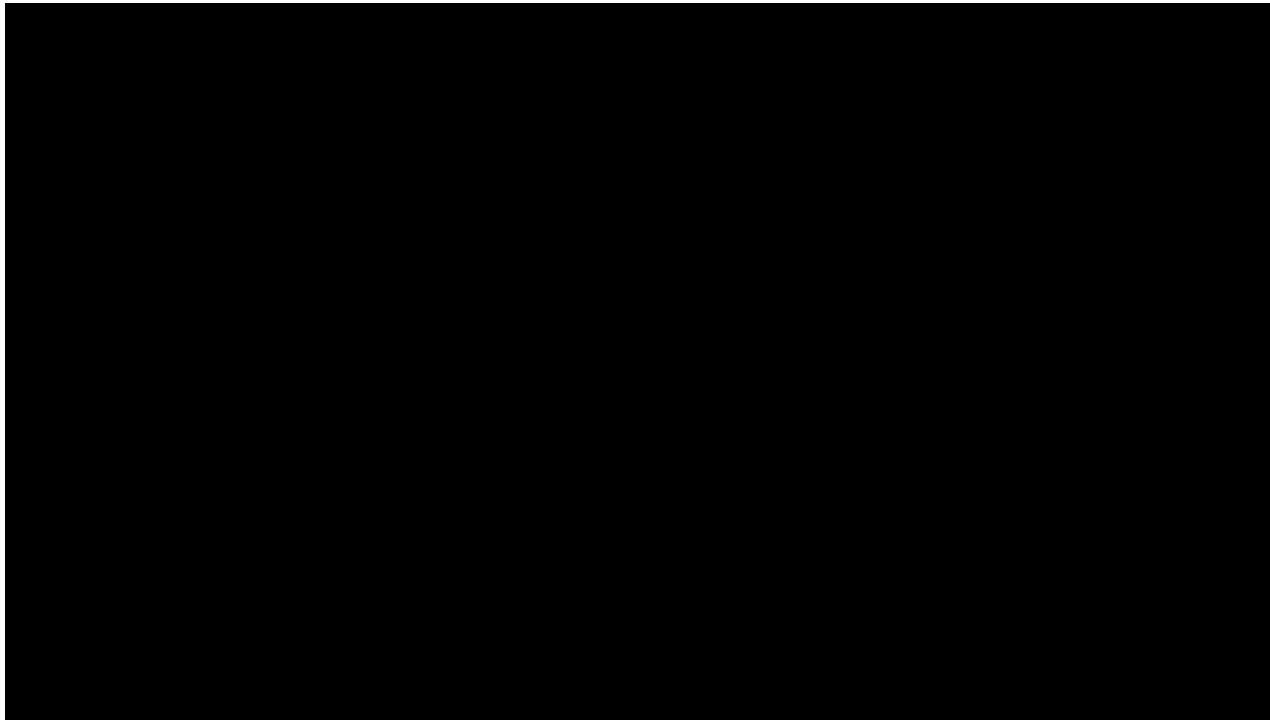
- RoI Pooling has some quantization operations
- These quantizations introduce misalignments between the RoI and the extracted features
- While this may not impact classification, it can make a negative effect on predicting bbox

YOLO

**YOU
ONLY
LIVE
ONCE**

YOLO

- You Only Look Once
- Quite similar with Faster R-CNN and very FAST!

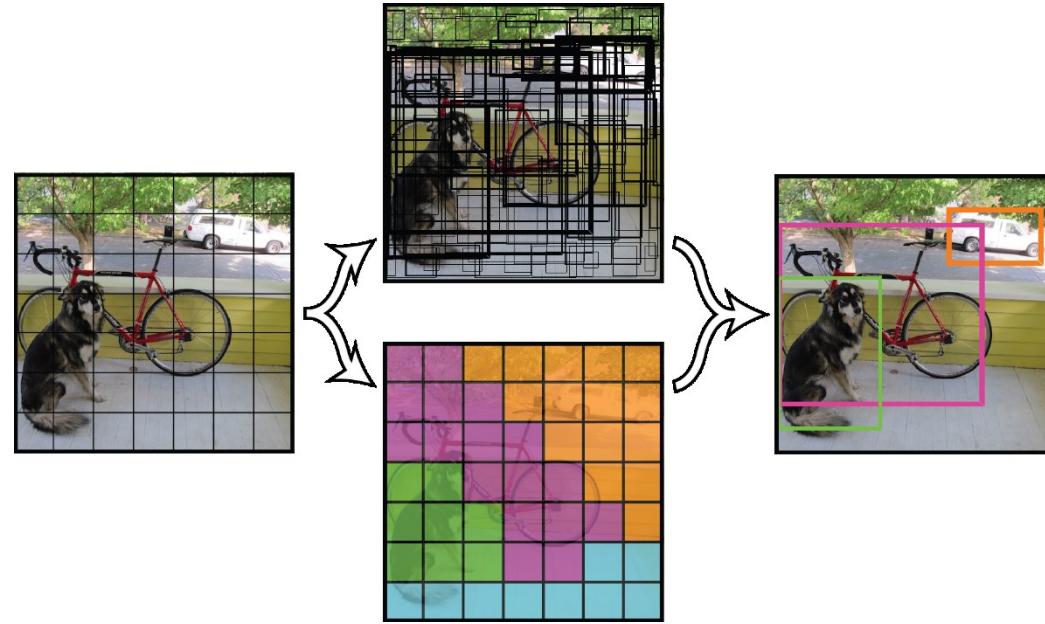


Concepts

- Detection as Single Regression Problem
- Developed as Single Convolutional Network
- Reason Globally on the Entire Image
- Learns Generalizable Representations

Unified Detection

- Given an image, divide it into an $S \times S$ grid
 - If the center of an object falls into the grid cell, that grid cell is responsible
- Each cell predicts B bounding boxes
 - Five predictions: $x, y, w, h, \text{confidence}$
- Each cell predicts C class probabilities
 - Cell → C class probabilities, $B \times 5$ bounding box informations
- One cell → One class probability
 - Low false positive!



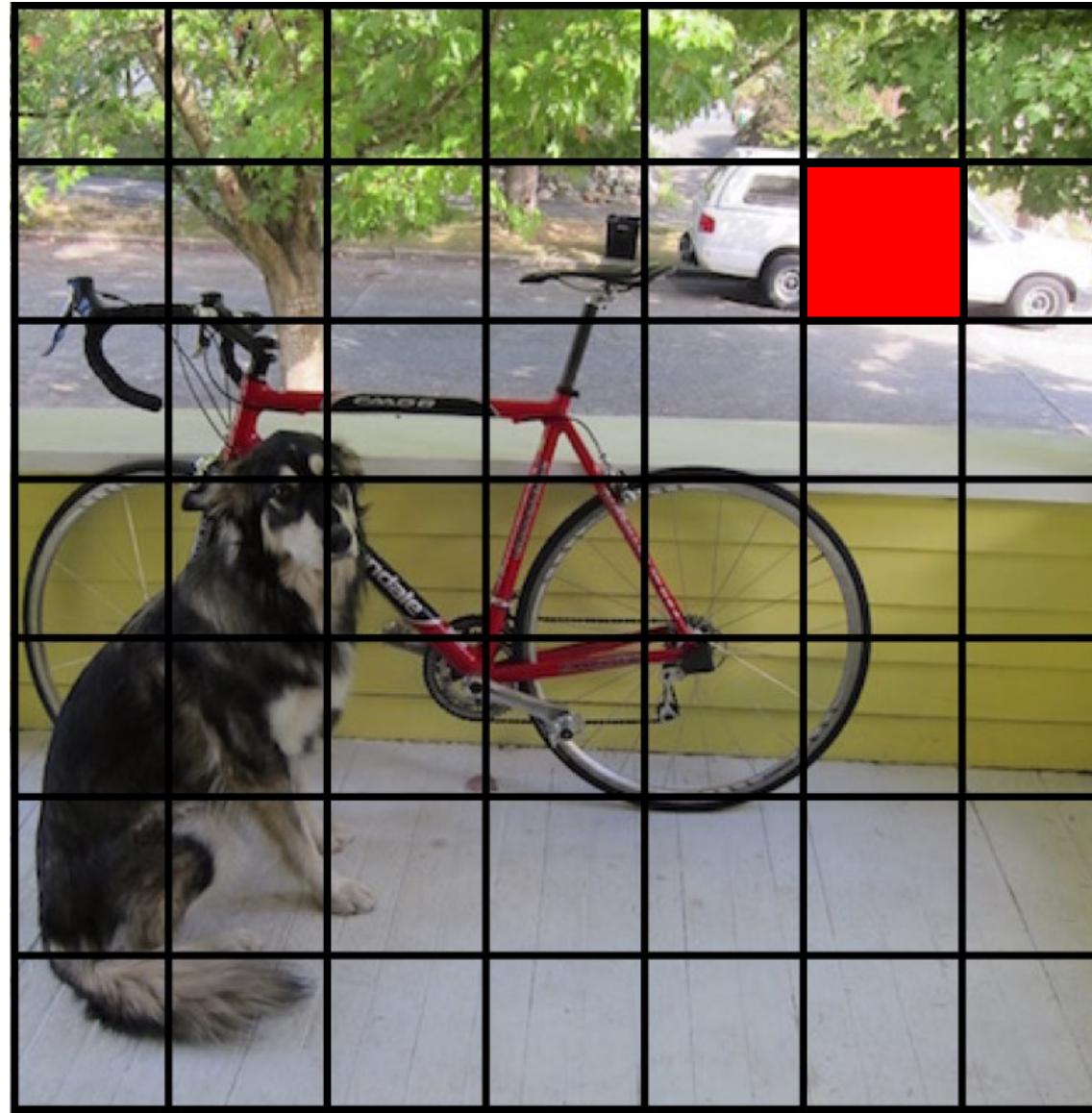
Detection Flow



Split the image into a grid(7x7)



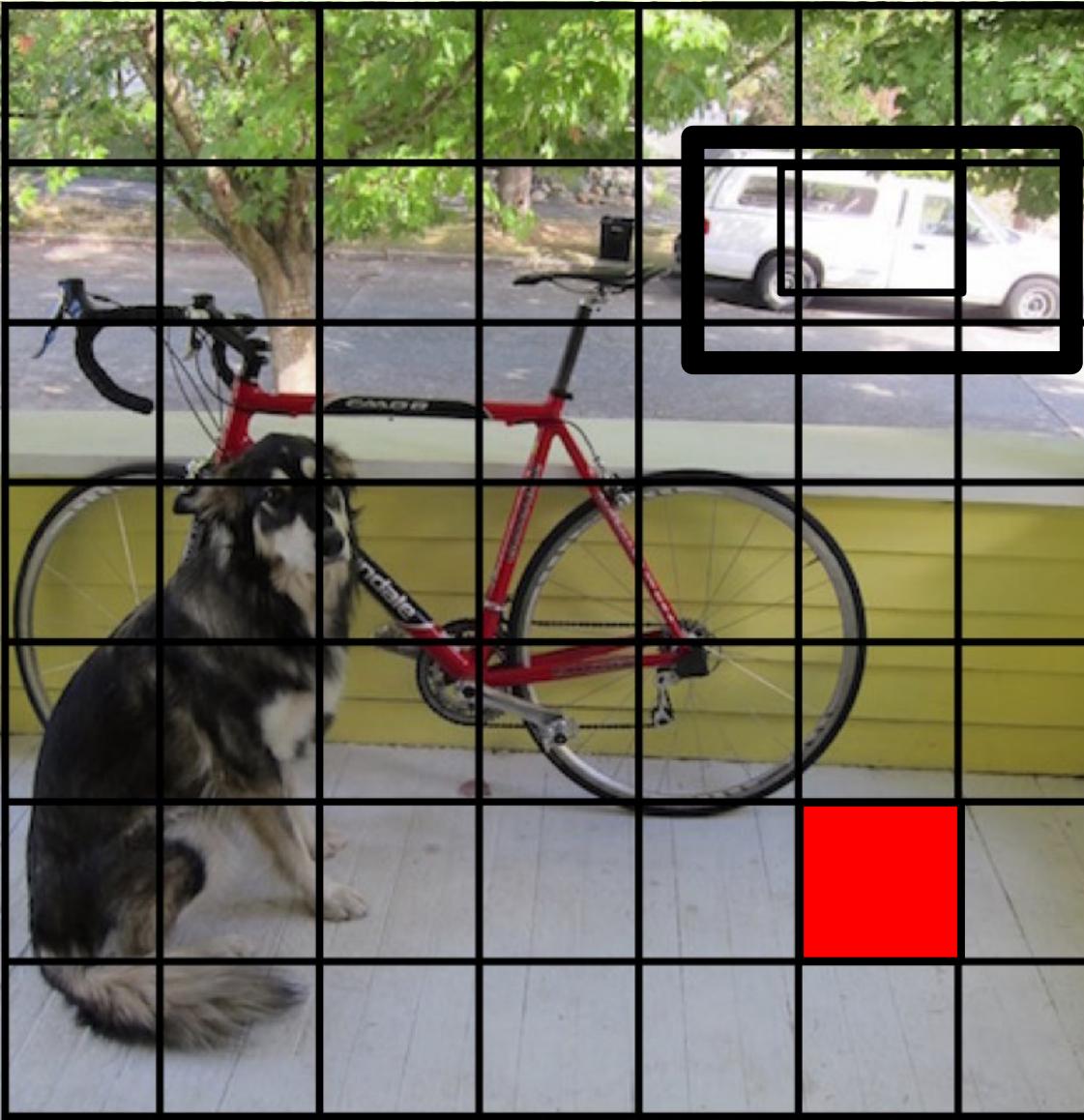
Each cell predicts boxes and confidences: $P(\text{Object})$



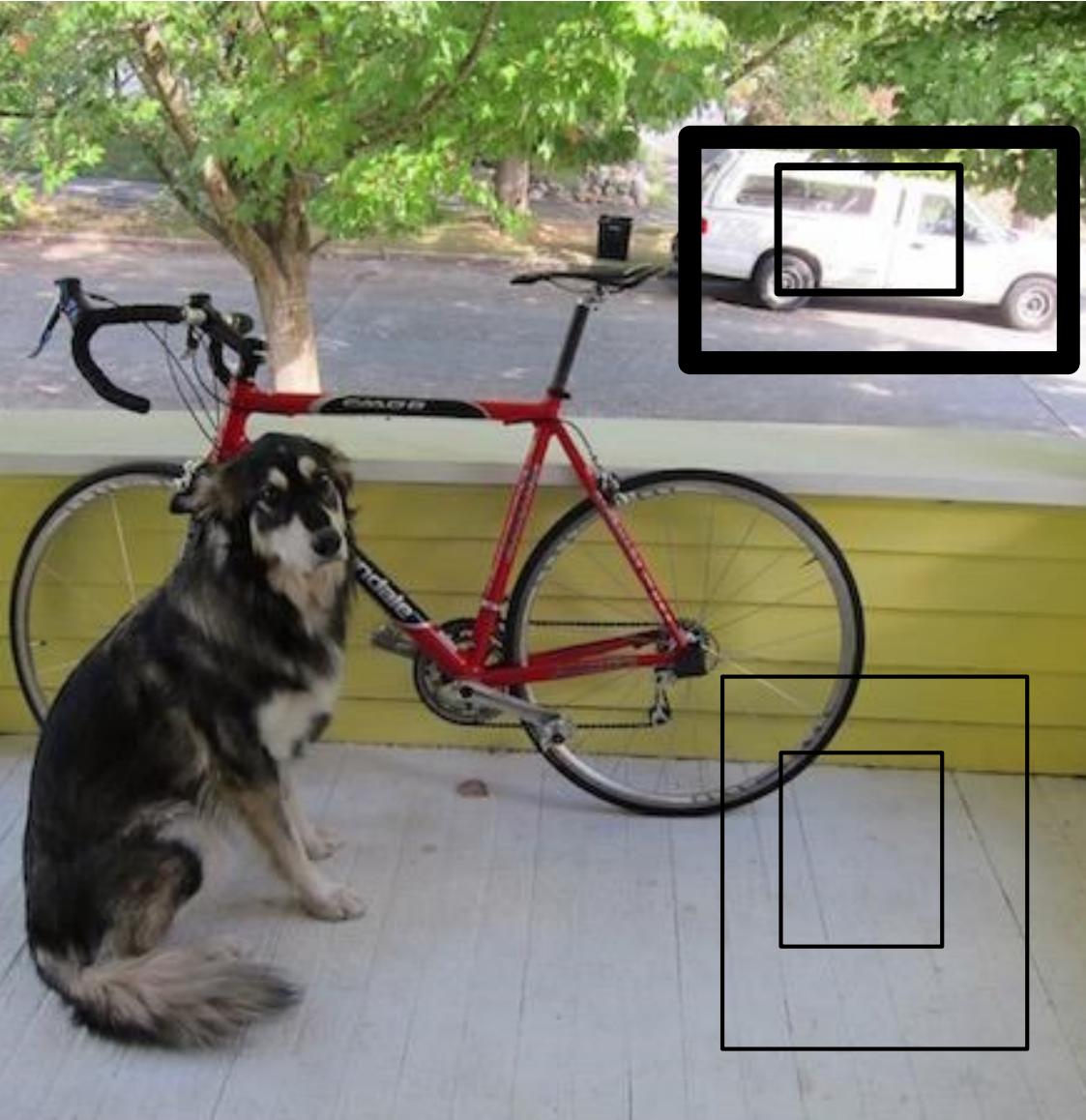
Each cell predicts boxes and confidences:
 $P(\text{Object})$



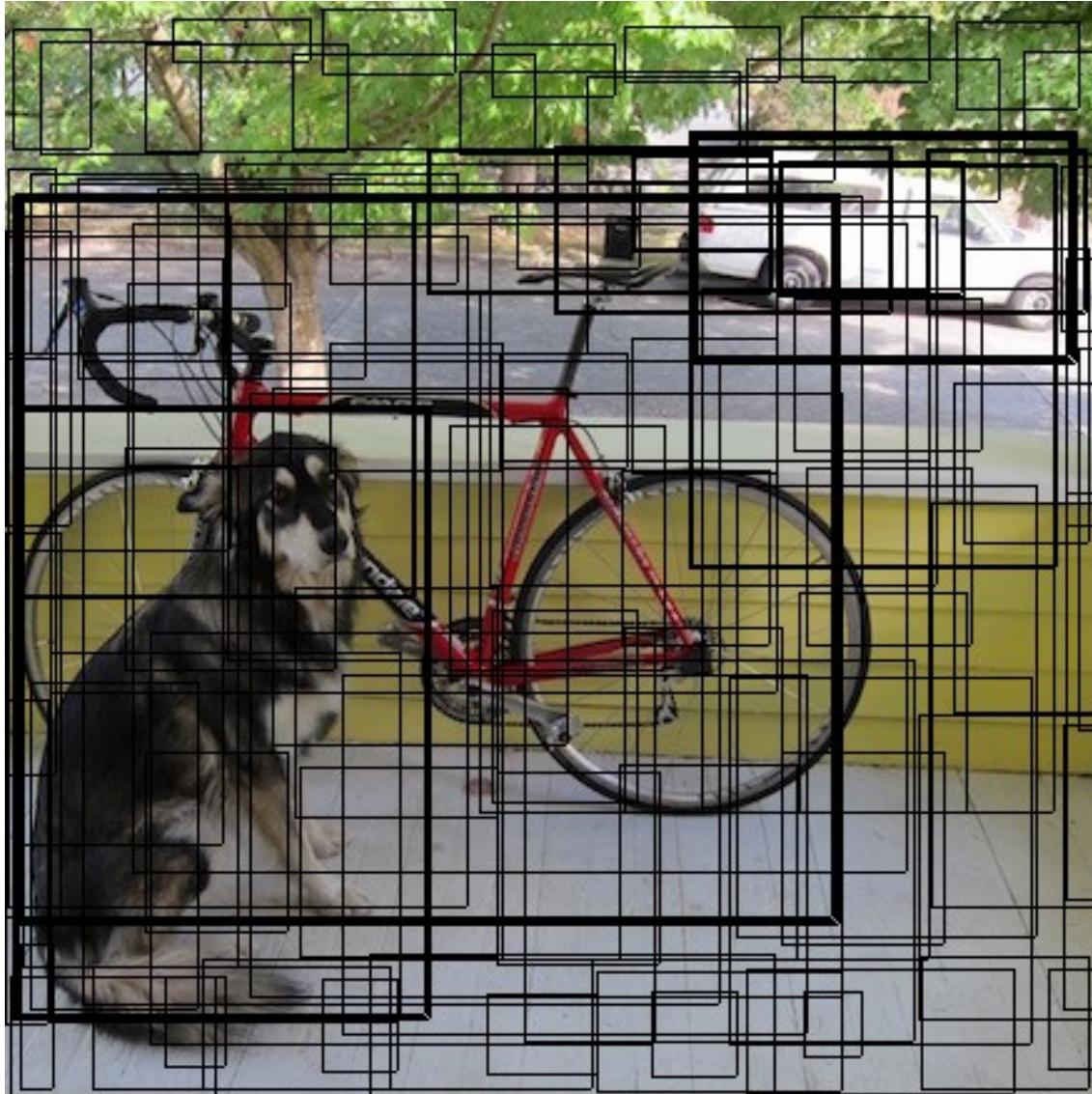
Each cell predicts boxes and confidences:
 $P(\text{Object})$



Each cell predicts boxes and confidences:
 $P(\text{Object})$



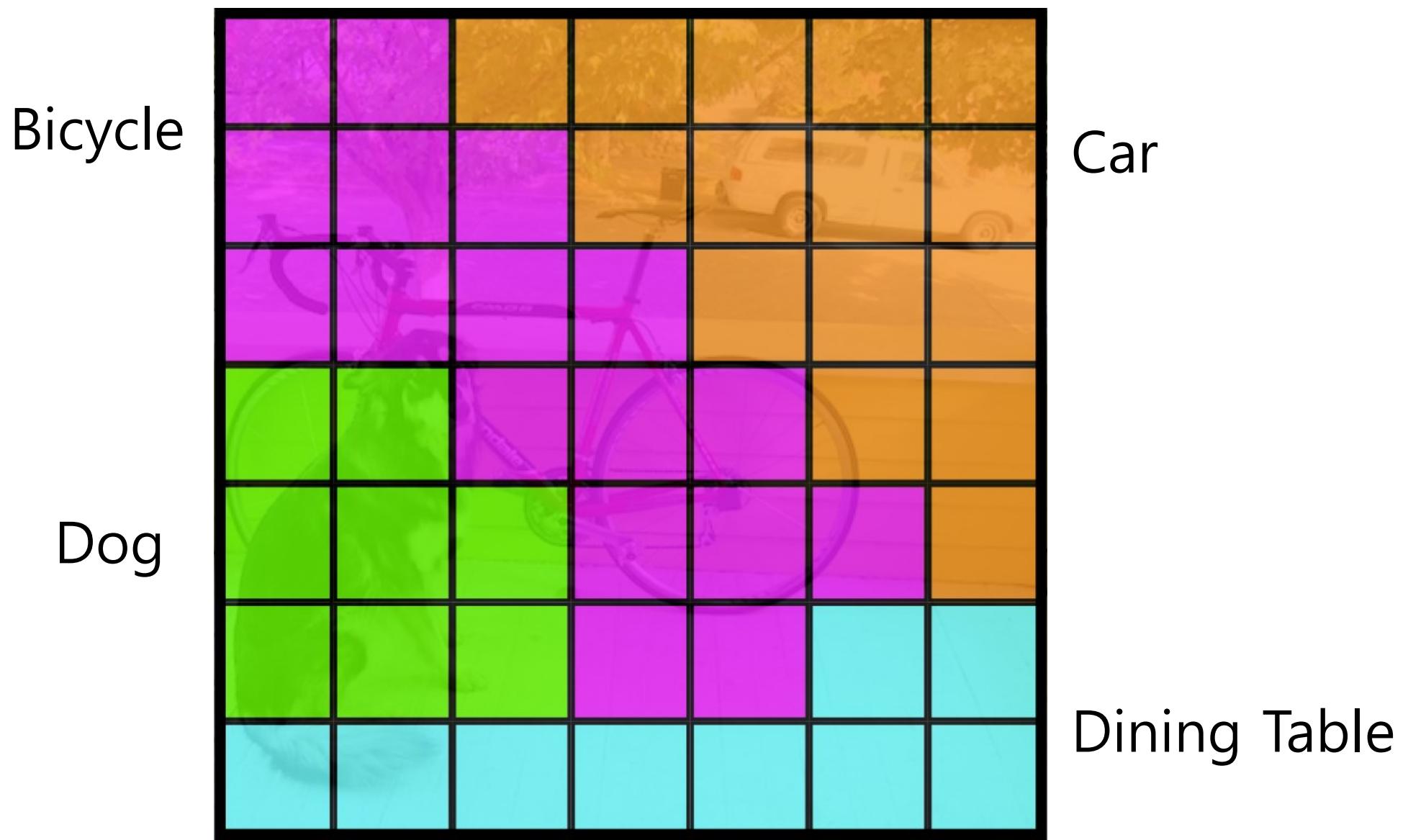
Each cell predicts boxes and confidences:
 $P(\text{Object})$



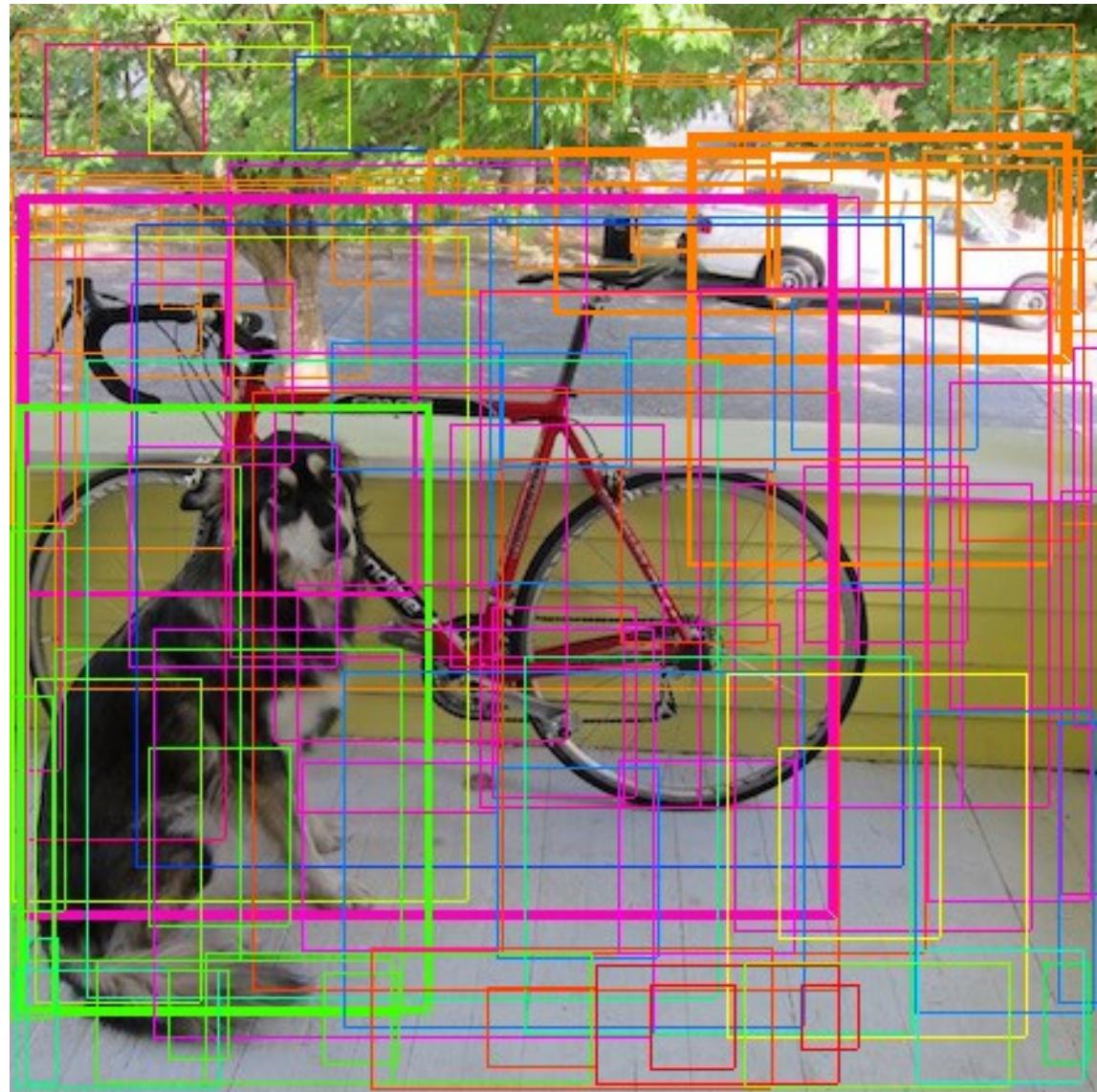
Each cell also predicts a class probability



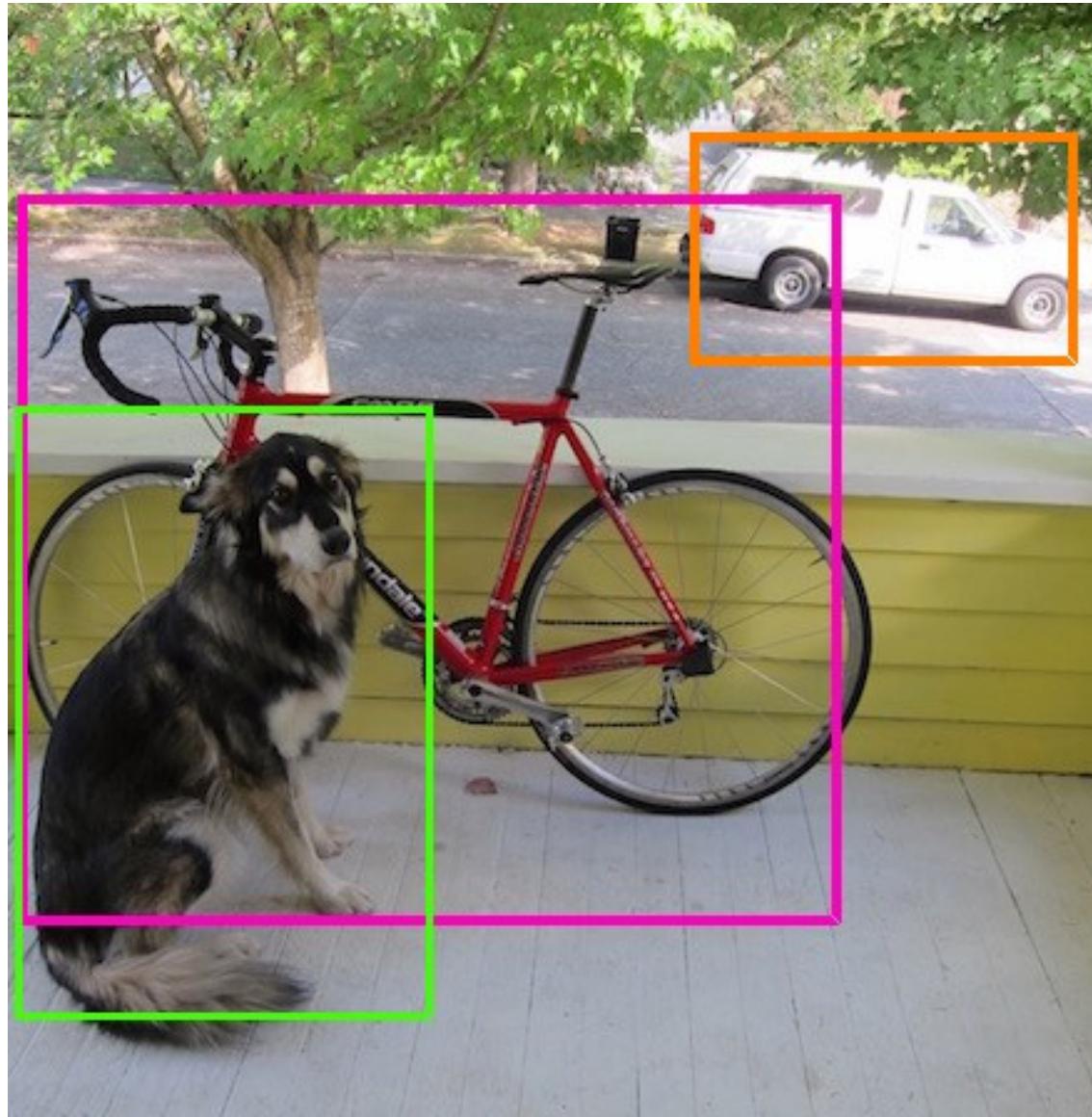
Each cell also predicts a class probability



Then we combine the box and class predictions.



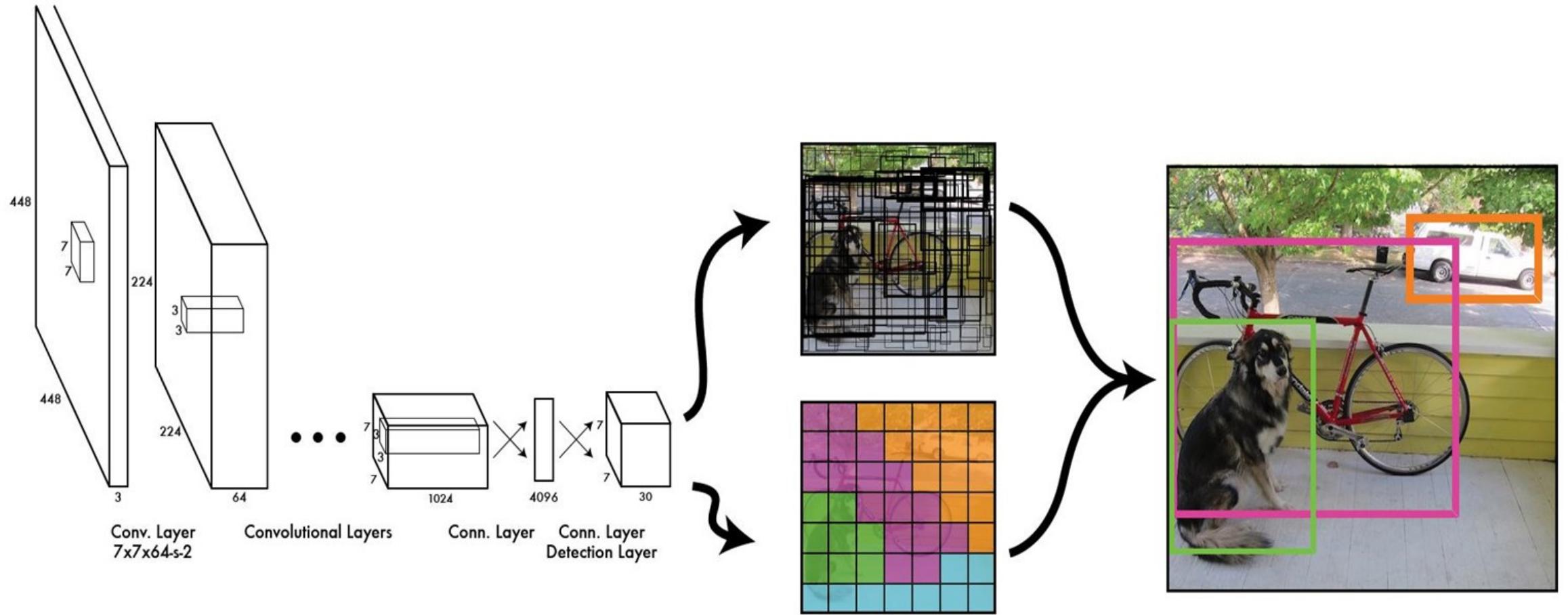
Finally we do NMS and threshold detections



Non-Maximum Supresion



YOLO



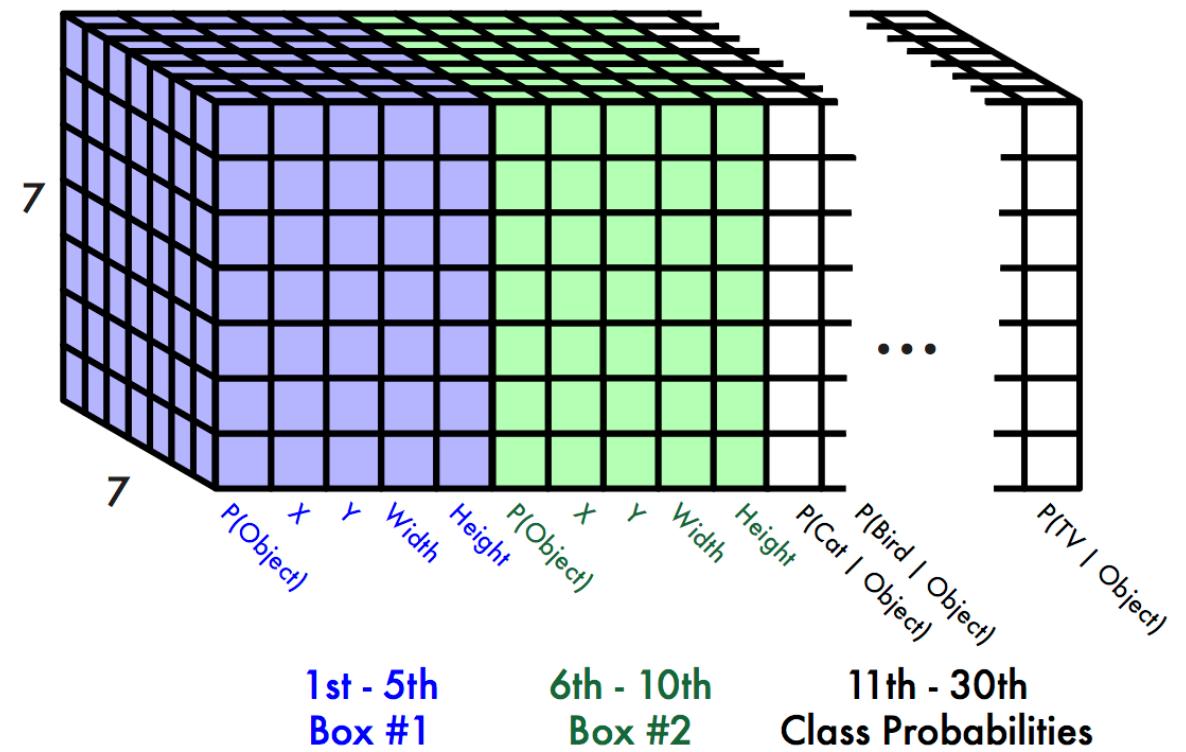
Outputs

Each cell predicts:

- For each bounding box:
 - 4 coordinates (x, y, w, h)
 - 1 confidence value
- Some number of class probabilities

For Pascal VOC:

- 7×7 grid
- 2 bounding boxes / cell
- 20 classes



Loss Function

- In training, one predictor which has the highest IoU with the ground truth is responsible

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3) \end{aligned}$$

$\mathbb{1}_{ij}^{\text{obj}}$

The **jth bbox predictor** in **cell i** is “responsible” for that prediction

$\mathbb{1}_{ij}^{\text{noobj}}$

$\mathbb{1}_i^{\text{obj}}$

If object appears in **cell i**

Note that the loss function only penalizes classification error if an object is present in that grid cell (hence the conditional class probability discussed earlier). It also only penalizes bounding box coordinate error if that predictor is “responsible” for the ground truth box (i.e. has the highest IOU of any predictor in that grid cell).

Problems

- Each grid cell can predict only $B(=2)$ bounding boxes and one class probability
 - Not good for small objects that flock together
- Uses relatively coarse features
 - Locations of bboxes are inaccurate
- Loss function treats errors in small bbox and big bbox equally
 - Not good for scoring

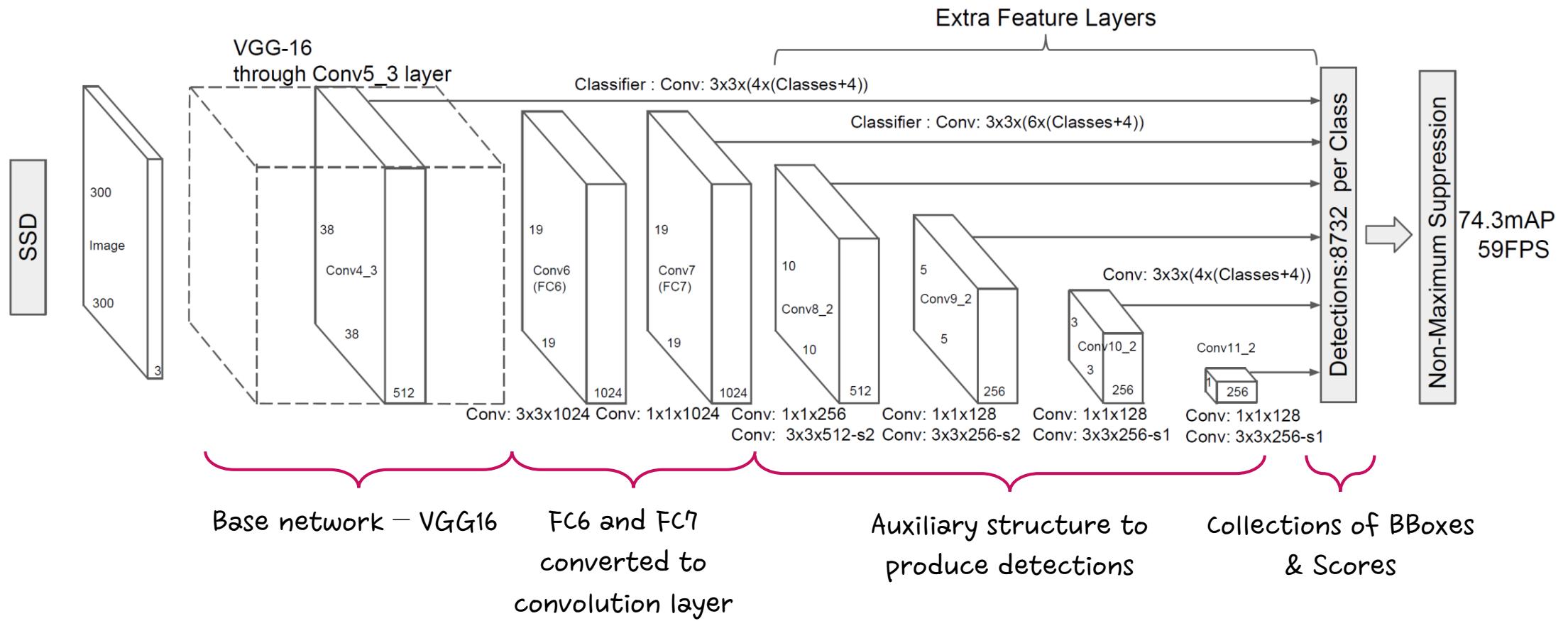
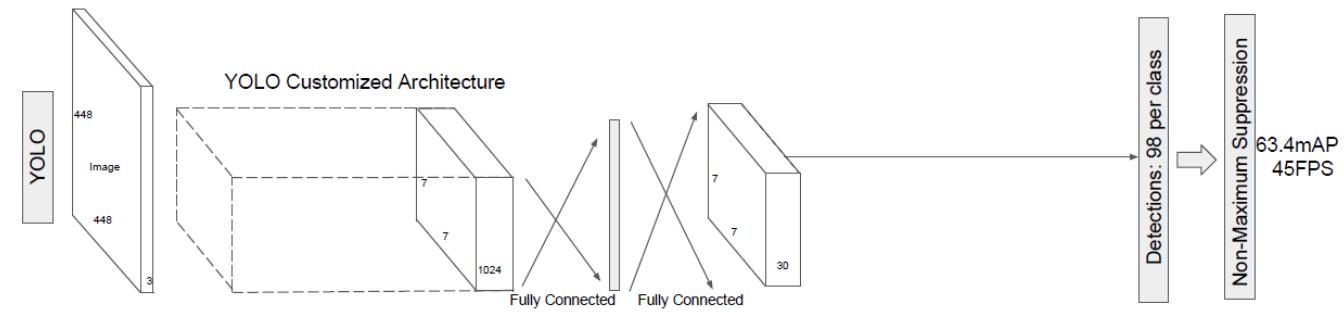
SSD – Single Shot MultiBox Detector

- Current state-of-the-art object detection systems are variants of the following approach:
 - Hypothesize bounding boxes, resample pixels or features for each box, and apply a high-quality classifier.
- Computationally too intensive and too slow for real-time applications.
 - Faster R-CNN operates at only 7 FPS with mAP 73.2%
- Significantly increased speed comes only at the cost of significantly decreased detection accuracy.
 - YOLO operates at 45FPS with mAP 63.4%

Single Shot MultiBox Detector

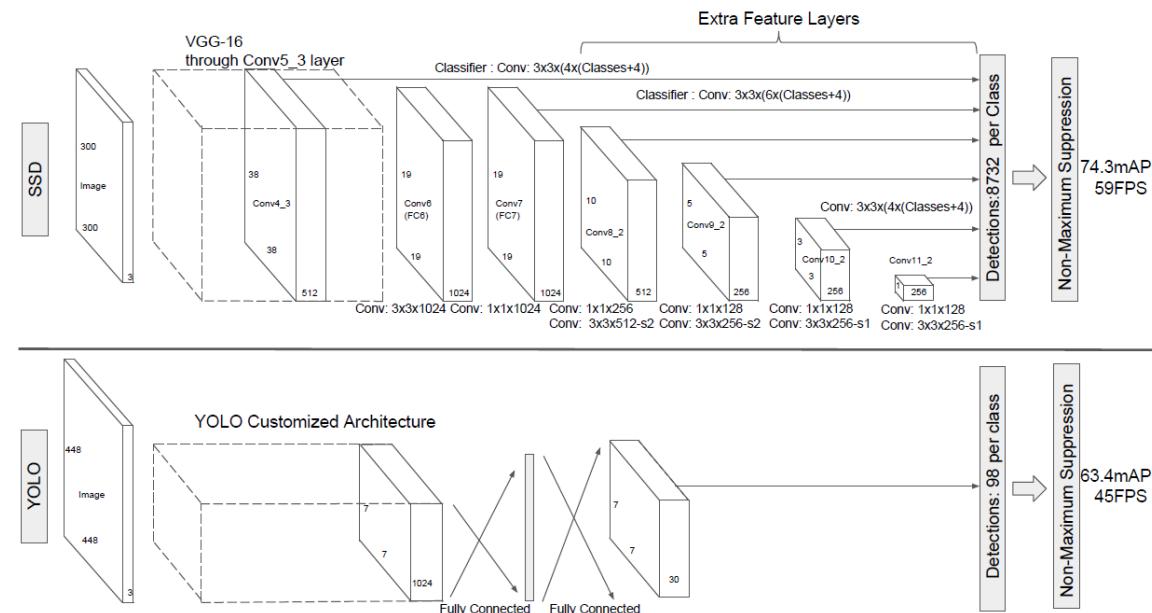
- First deep network based object detector that does not resample pixels or features for bounding box hypotheses(1-stage) and is as accurate as approaches that do.
 - 59 FPS with mAP 74.3% on VOC2007 test

Model



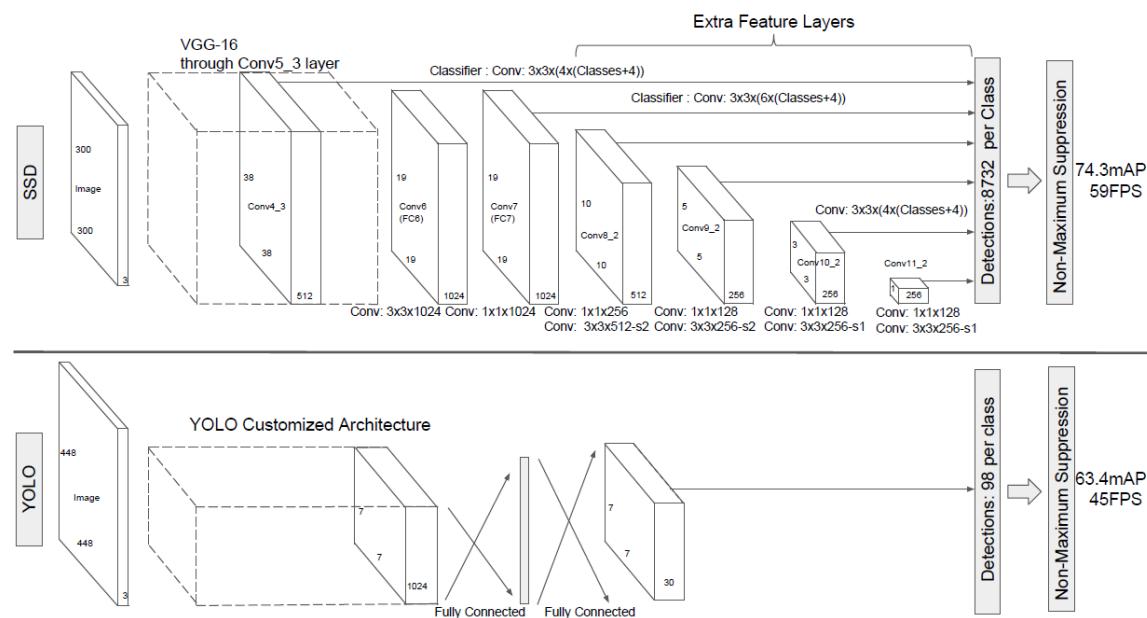
Multi-scale Feature Maps for Detection

- Adding convolutional feature layers to the end of the truncated base networks.
- These layers decrease in size progressively and allow predictions of detections at multiple scales.
 - YOLO use only single scale feature map.



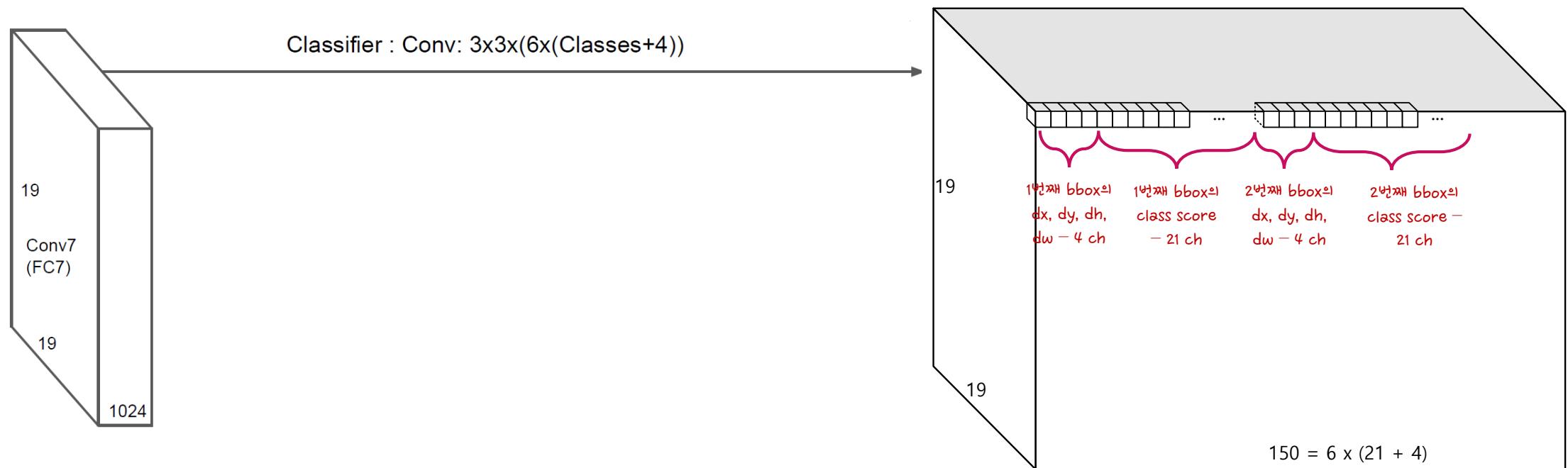
Convolutional Predictors for Detection

- For a feature layer of size $m \times n$ with p channels, the basic element for predicting parameters of a potential detection is a $3 \times 3 \times p$ small kernel that produces either a score for a category, or a shape offset relative to the default box coordinates.
 - YOLO has a fully connected layer for a score and a bbox coordinates.



Default Boxes and Aspect Ratios

- For each box out of k at a given location, we compute c class scores and the 4 offsets relative to the original default box shape.
- Total of $(c+4)k$ filters at each location in the feature map.
- Yielding $(c+4)kmn$ outputs for a $m \times n$ feature map.



Training – Matching Strategy

- Matching each ground truth box to **the default box with the best jaccard overlap**. Every ground truth box has at least 1 correspondence.
- Then, matching **default boxes to any ground truth with jaccard overlap higher than a threshold(0.5)**.

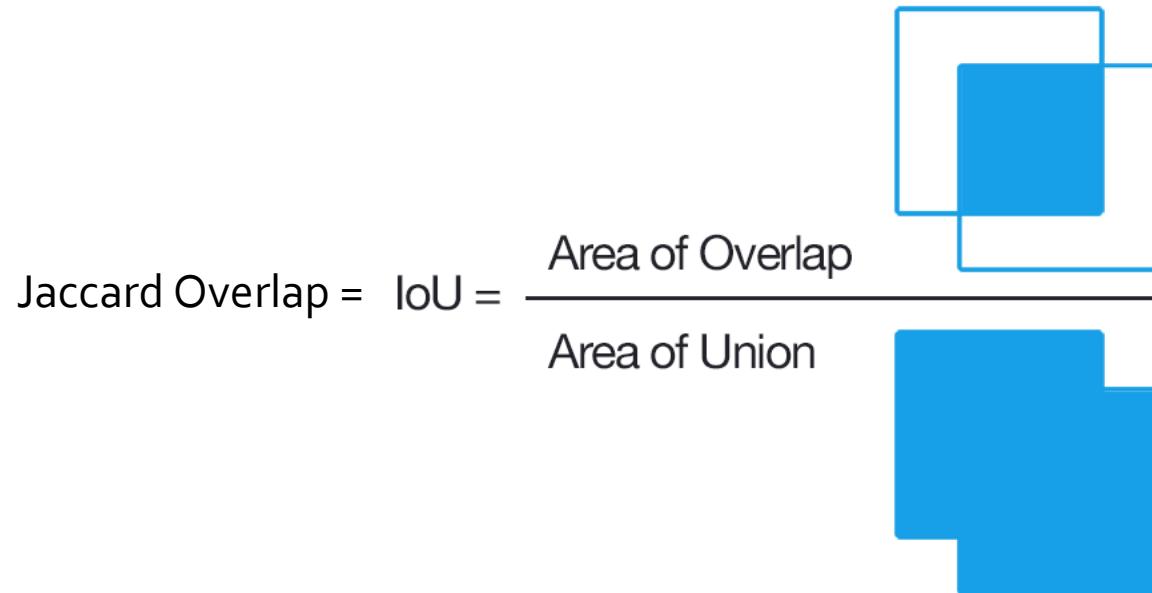


Figure from Wikipedia

Training Objective

- Similar to Faster R-CNN

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

set to 1 by cross validation

- N : number of default matched bboxes
- L_{conf} : classification loss → cross-entropy

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

- L_{loc} : localization loss → Smooth L1 loss

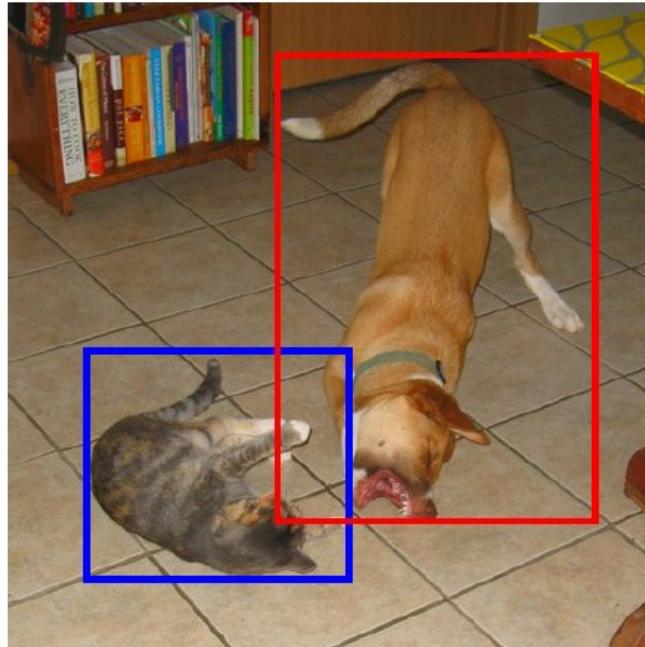
$$L_{loc}(x, l, g) = \sum_{i \in Pos}^N \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

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

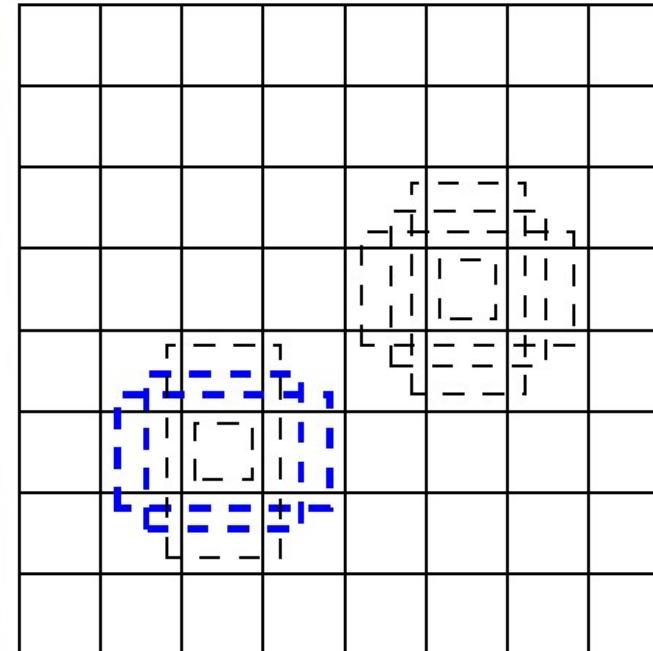
$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \quad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

Choosing Scales and Aspect Ratios for Default Boxes

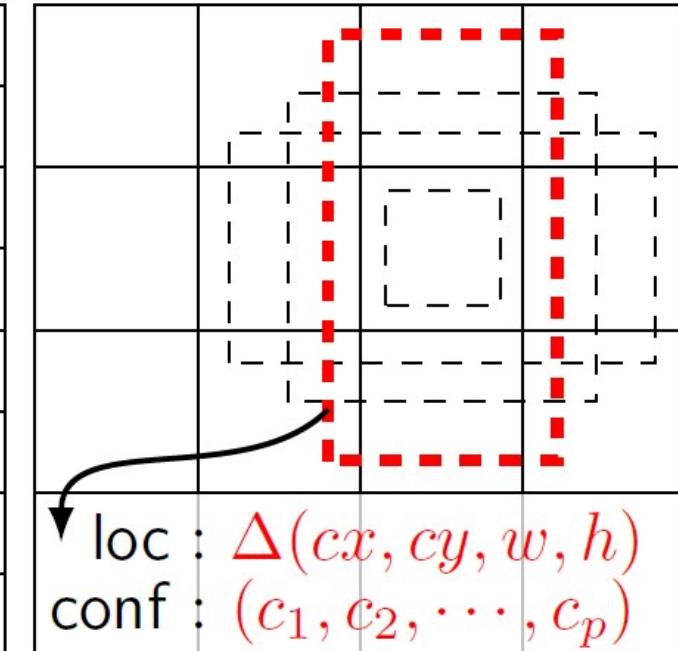
- Feature maps from different layers are used to handle scale variance.
- Specific feature map locations learn to be responsive to specific area of the image and particular scales of objects.



(a) Image with GT boxes

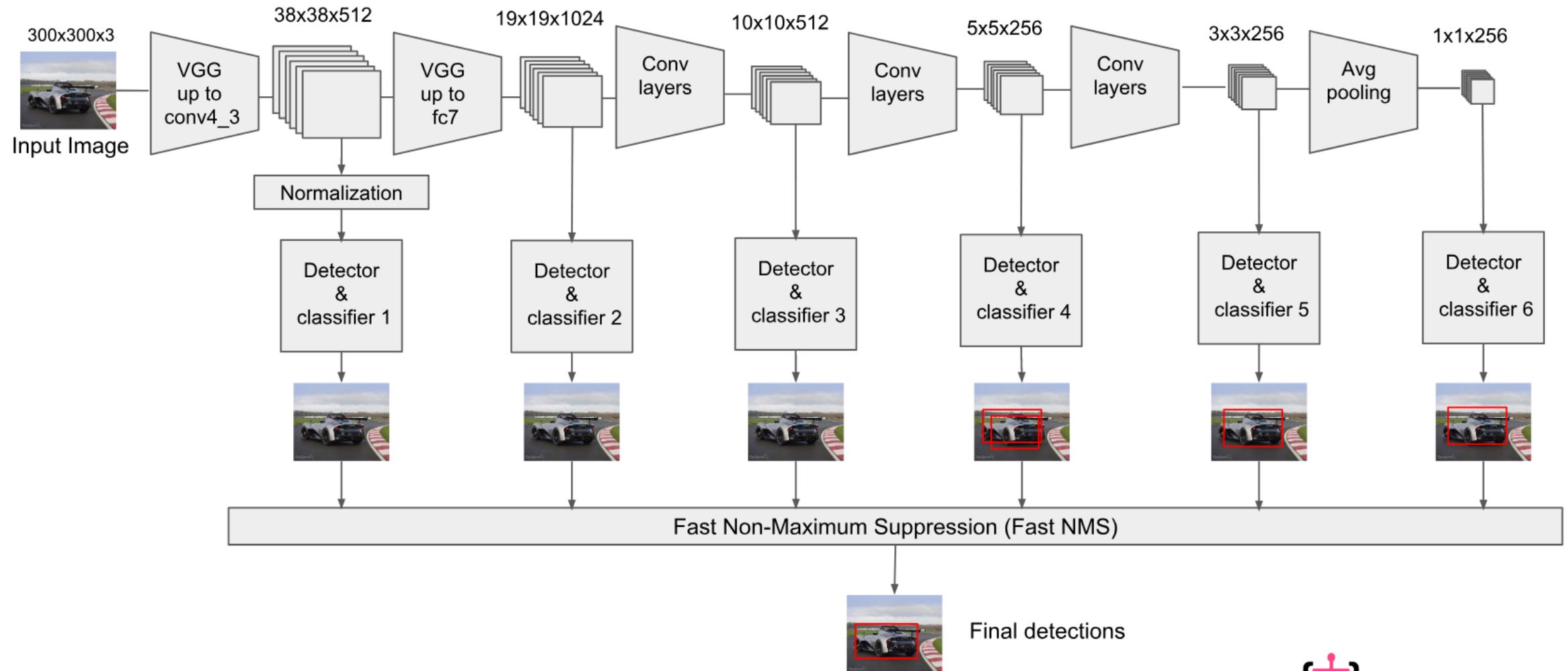


(b) 8×8 feature map



(c) 4×4 feature map

Choosing Scales and Aspect Ratios for Default Boxes



Hard Negative Mining

- Significant imbalance between positive and negative training examples.
 - After the matching step, most of the default boxes are negatives, especially when the number of possible default boxes is large.
- Sorting them using **the highest confidence loss** for each default box.
- Pick the top ones so that the ratio **between the negatives and positives is at most 3:1**

Class Imbalance Problem – RetinaNet

- Class 간의 data가 균형이 안맞는 경우

- Weighted cross-entropy

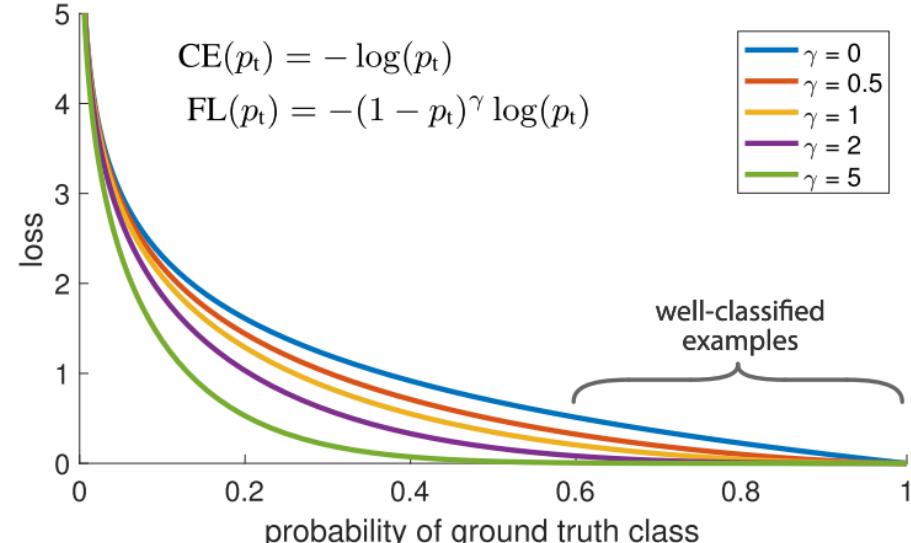
- Cross entropy 식에 class 별로 weight를 다르게 주어서 사용

$$L = - \sum_c w_c \cdot \log p$$

- Focal Loss

- Cross entropy에 비해 높은 확률의 경우에는 상대적으로 작은 loss를 나온 확률의 경우에는 높은 loss를 주도록 $(1-p)^\gamma$ 를 곱해줌

$$L = - \sum (1 - p)^\gamma \cdot \log p$$



Experimental Results

- Base Network
 - VGG16 with fc6 and fc7 converted to conv layers and pool5 from $2 \times 2 - s_2$ to $3 \times 3 - s_1$ using atrous algorithm, removed fc8 and dropout
 - Fine-tuned using SGD with momentum

PASCAL VOC2007 Detection Results

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

Inference Time

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