Object Detection

PIRL, POSTECH

Hanul Roh

Computer Vision Tasks

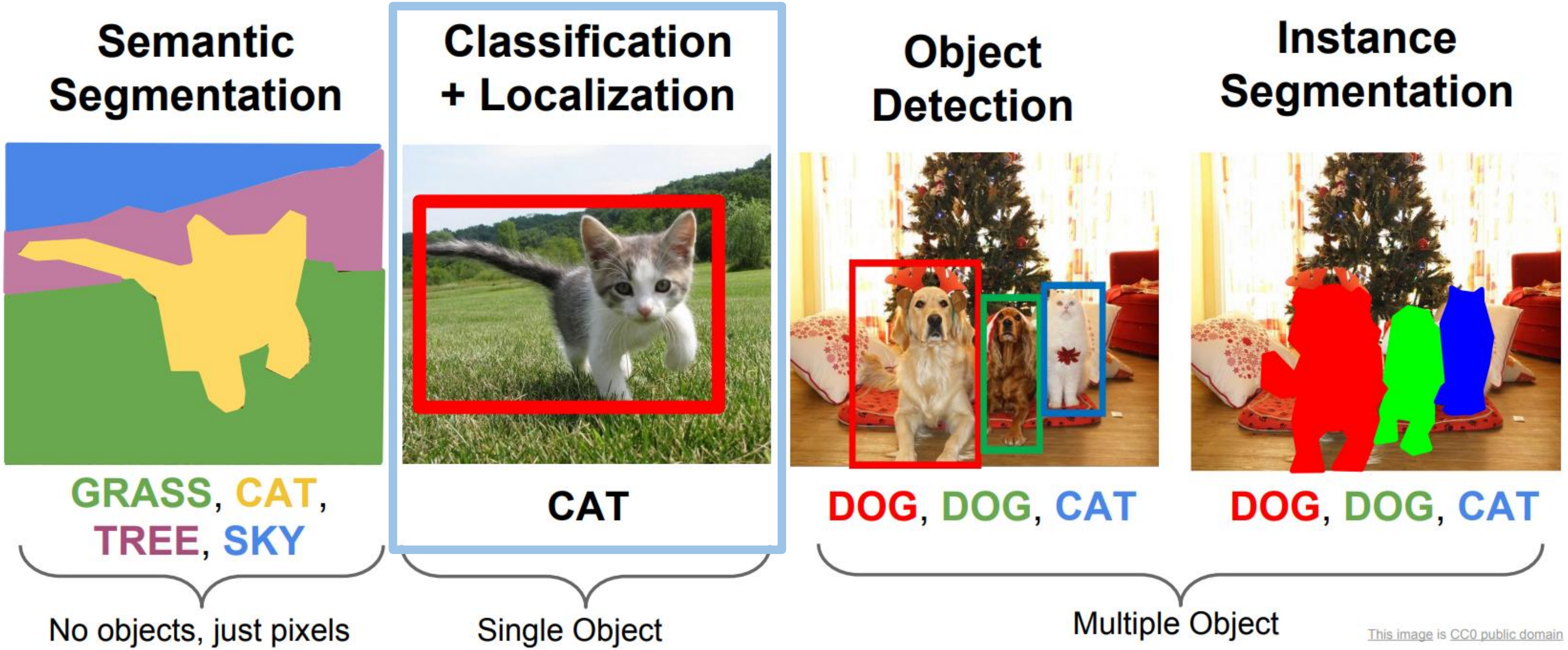


Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Computer Vision Tasks

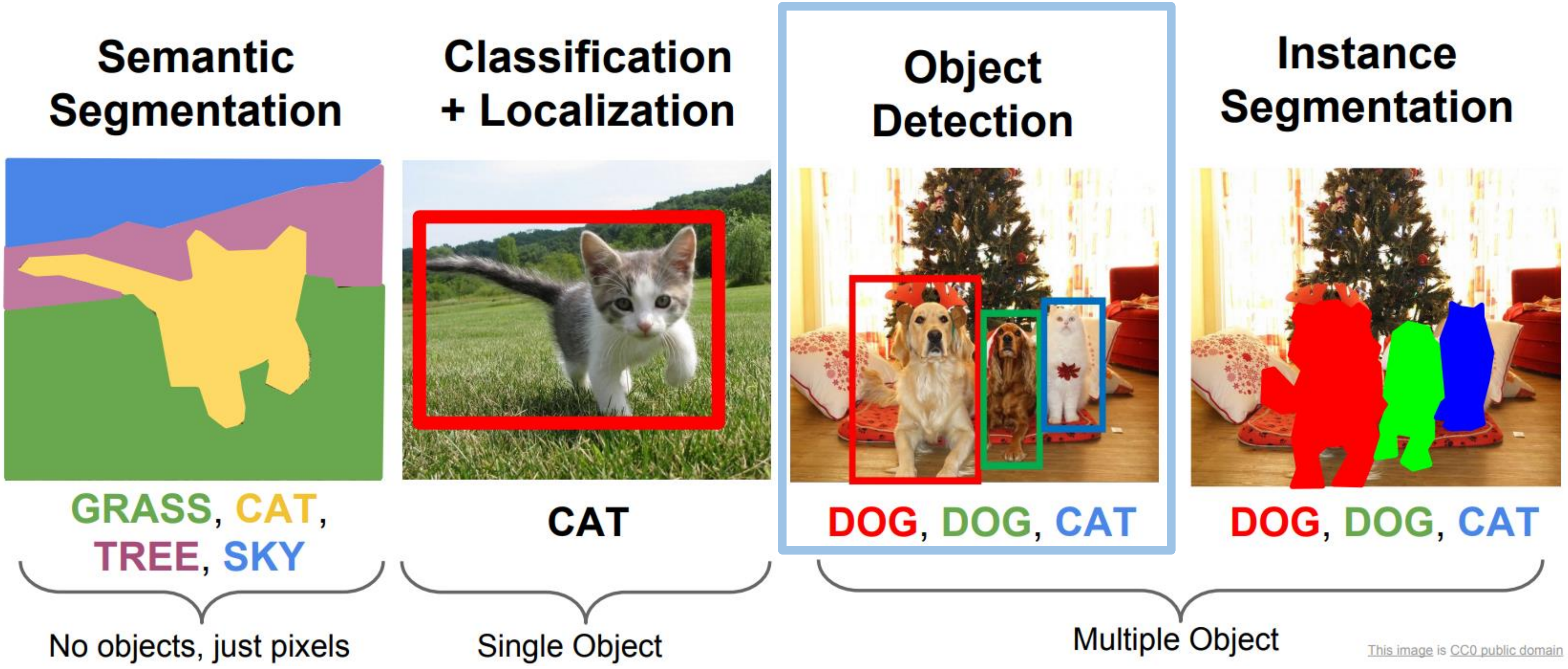


Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Classification + Localization

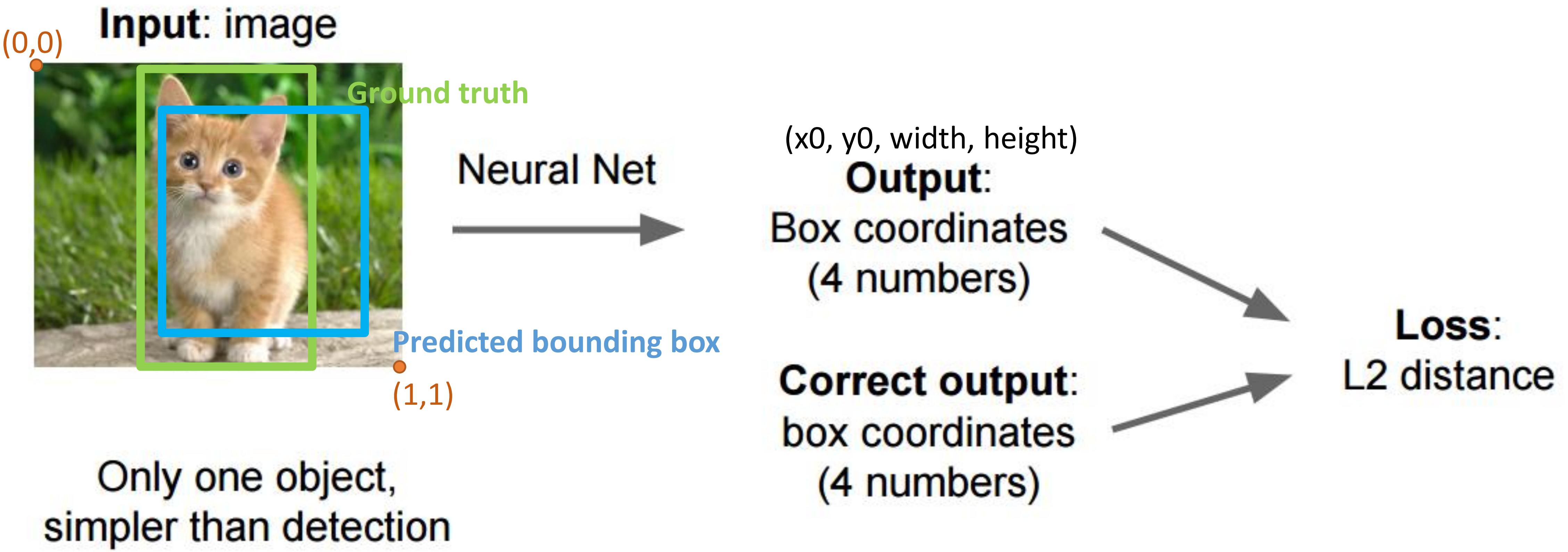


Figure from https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html

Classification + Localization

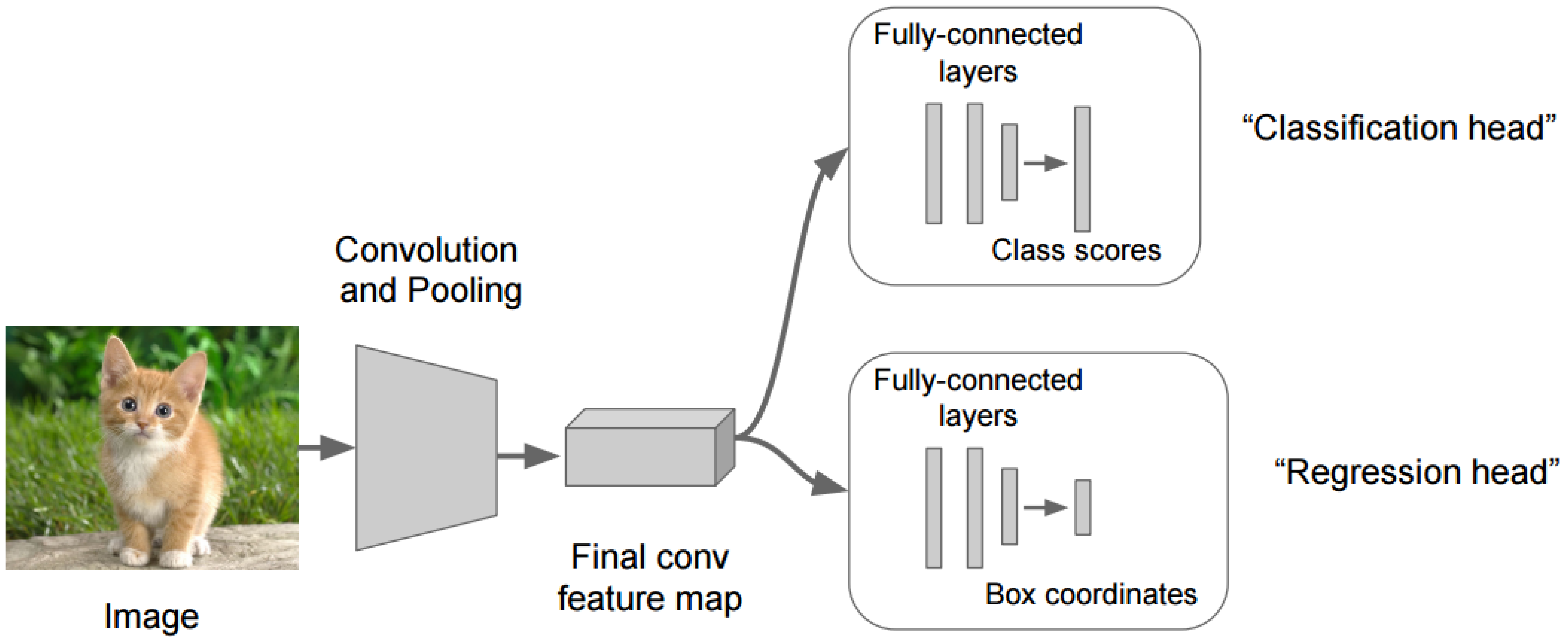
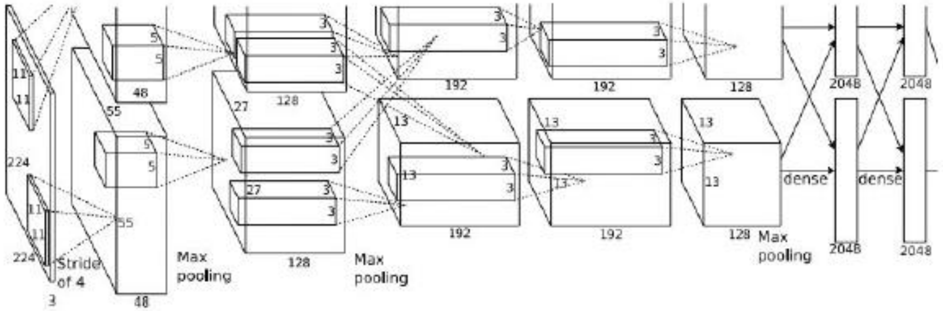


Figure from https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html

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 and scales, very computationally expensive!

Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Region Proposals

- Find image regions that are likely to contain objects
- To find regions, Selective Search algorithm gives 1K~2K region proposals per image

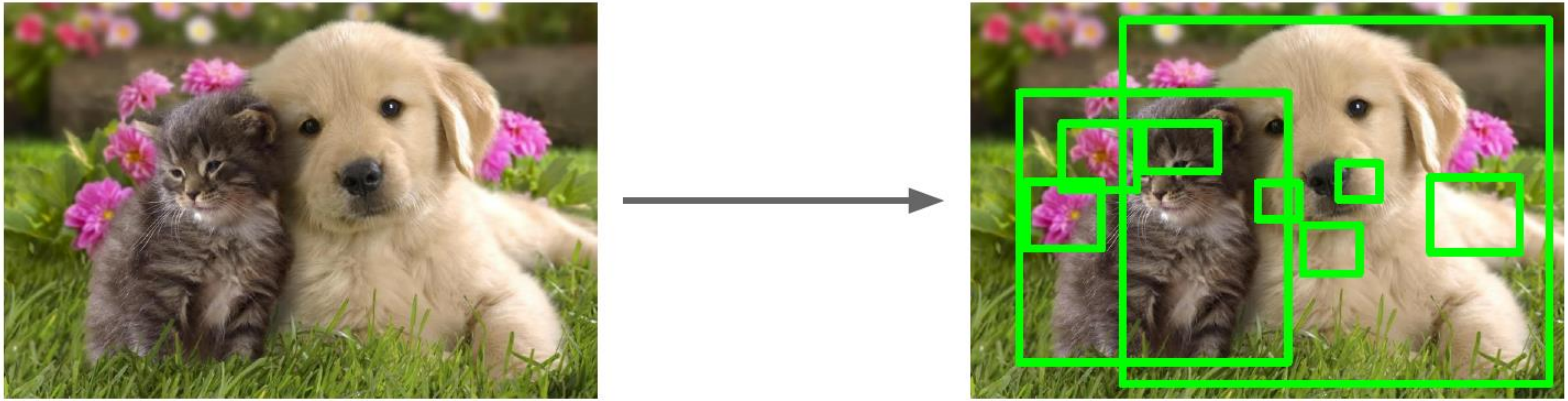
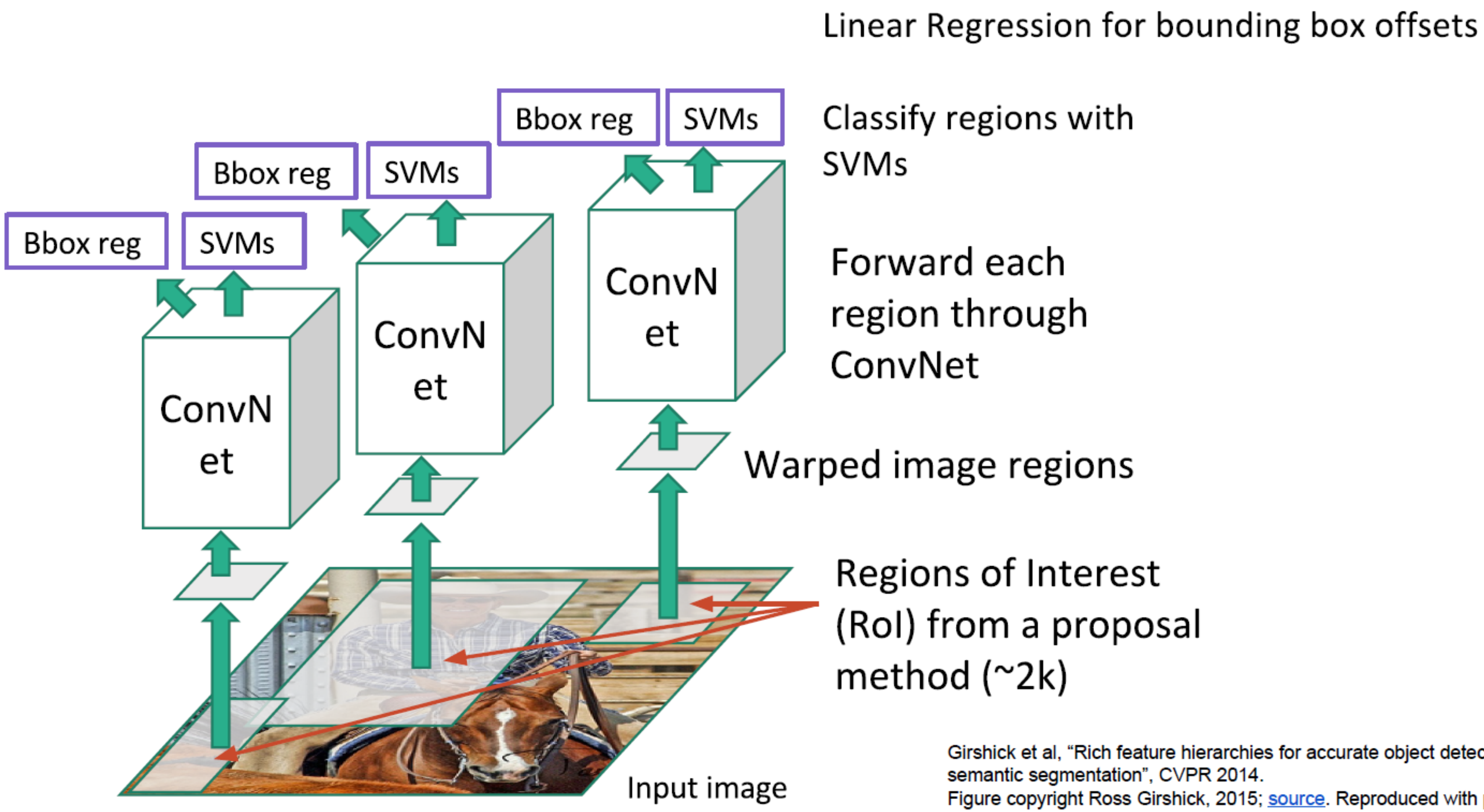


Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

R-CNN: Regions with CNN features



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

Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Region Proposal – Selective Search

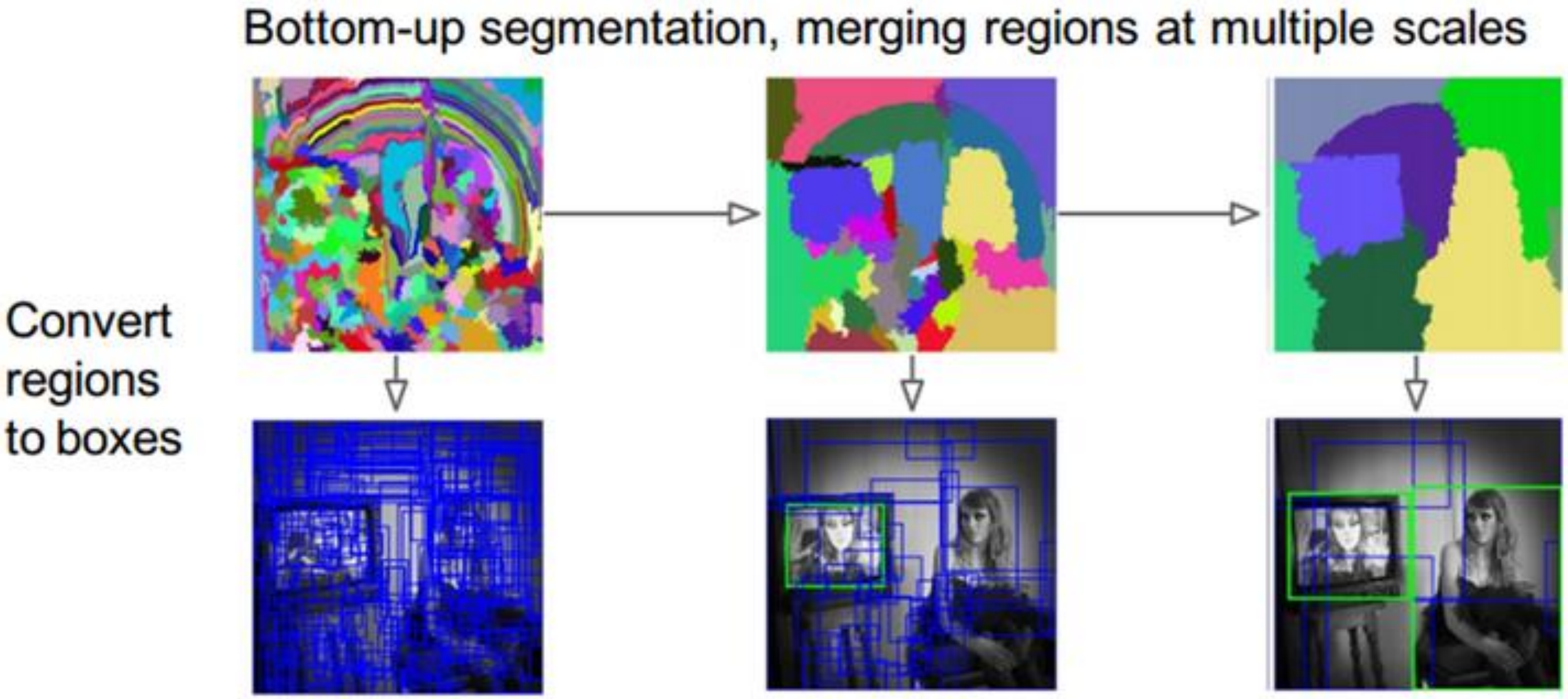


Figure from https://m.blog.naver.com/PostView.nhn?blogId=laonple&logNo=220918802749&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F

R-CNN Training

1. M ← **Pre-train a ConvNet** for ImageNet classification dataset
2. M' ← **Fine-tune** for object detection(softmax + log loss)
3. R ← Selective Search for region proposals
4. F ← **Cache feature vectors** to disk using M' in R
5. Train post hoc linear SVMs on F for object classification
6. Train post hoc linear bounding-box regressors on F

* Post hoc means the parameters are learned after the ConvNet is fixed

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

$G^i = (G_x^i, G_y^i, G_w^i, G_h^i)$ The ground-truth bounding box

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

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

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

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

From https://arxiv.org/pdf/1311.2524.pdf

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 pool$_5$ 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 $\boxed{d_\star(P) = \mathbf{w}_\star^{\mathrm{T}}\phi_5(P)}$, 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):

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

$$\hat{G}_x = P_w d_x(P) + P_x \quad (1)$$
$$\hat{G}_y = P_h d_y(P) + P_y \quad (2)$$
$$\hat{G}_w = P_w \exp(d_w(P)) \quad (3)$$
$$\hat{G}_h = P_h \exp(d_h(P)). \quad (4)$$

The regression targets $t_\star$ for the training pair (P, G) are defined as

$$t_x = (G_x - P_x)/P_w \quad (6)$$
$$t_y = (G_y - P_y)/P_h \quad (7)$$
$$t_w = \log(G_w/P_w) \quad (8)$$
$$t_h = \log(G_h/P_h). \quad (9)$$

From https://arxiv.org/pdf/1311.2524.pdf

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
  - 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
  - Fine-tune network with softmax classifier
  - Train post-hoc linear SVMs
  - Train post-hoc bounding-box regressions

Fast R-CNN

- 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

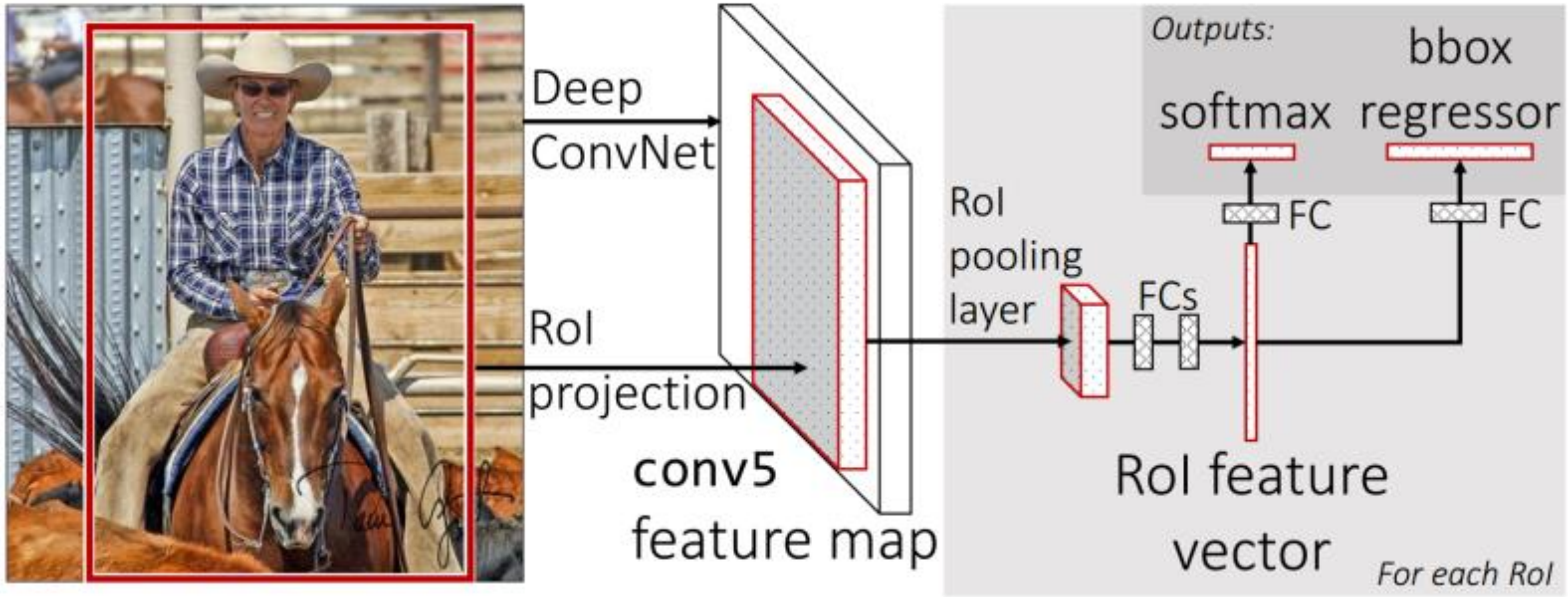
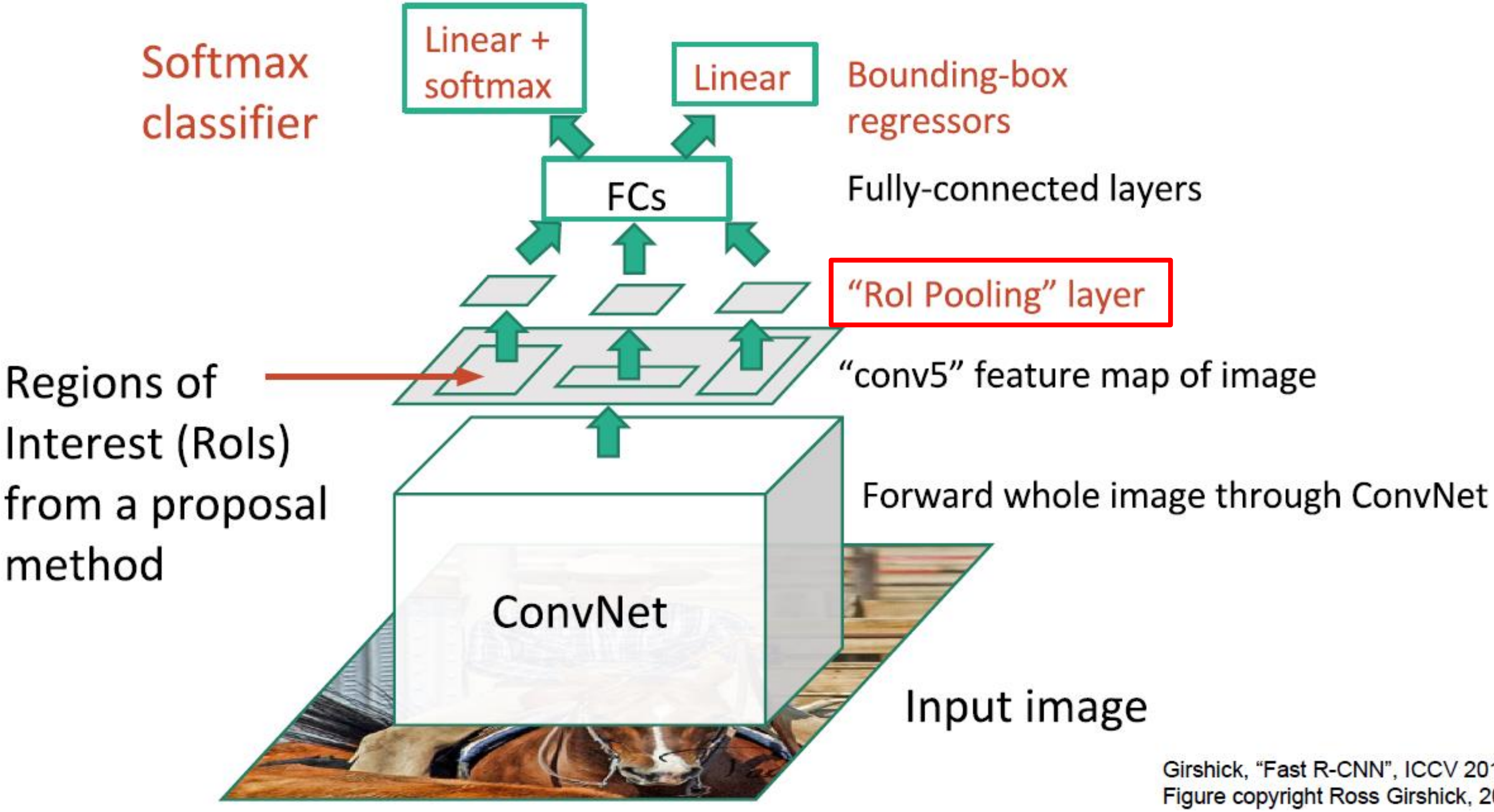


Figure from https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html

Fast R-CNN

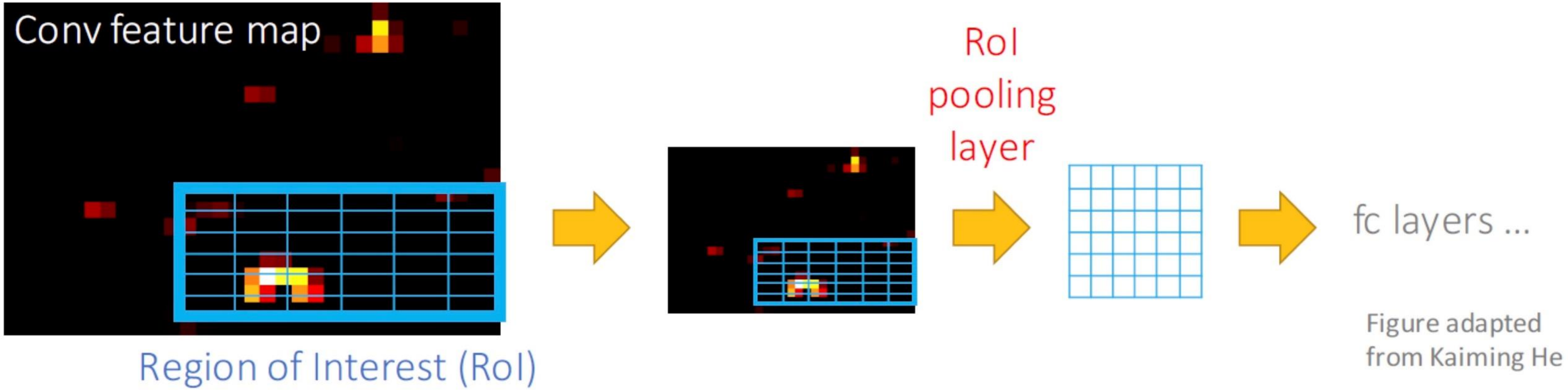


Girshick, "Fast R-CNN", ICCV 2015.

Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

ROI Pooling

- Its purpose is to perform max pooling on inputs of non uniform sizes to obtain fixed-size feature maps (e.g. 7×7 for VGG-net fc layer).
- The ROI pooling is done by:



ROI in Conv feature map : 21x14 -> 3x2 max pooling with stride (3,2) -> output: 7x7
ROI in Conv feature map : 35x42 -> 5x6 max pooling with stride (5,6) -> output: 7x7

Figure from https://jamiekang.github.io/2017/05/28/faster-r-cnn/

Fast R-CNN Training

1. Take 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
    1. K+1 class labels
    2. Bounding box locations

- Loss function

True box coordinates

Predicted box coordinates

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

True class score

Log loss

Smooth L1 loss

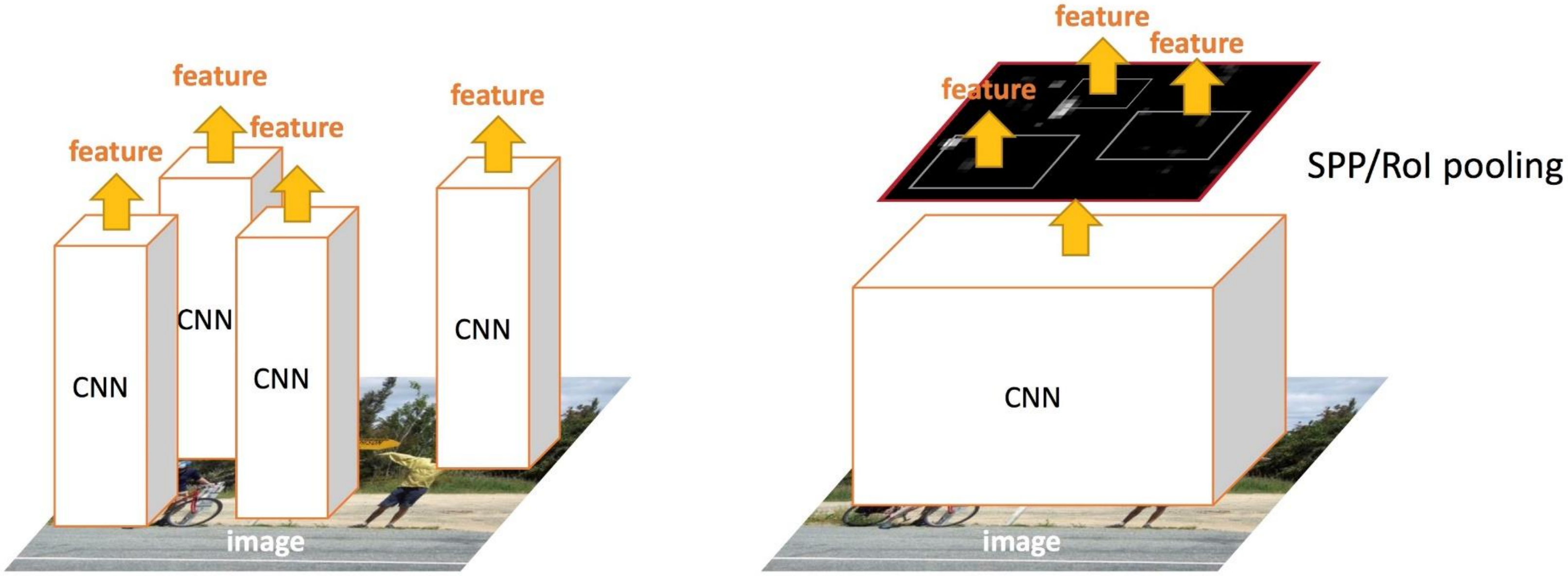
Predicted class score

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i), \qquad (2)$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases} \qquad (3)$$

R-CNN vs Fast R-CNN



R-CNN

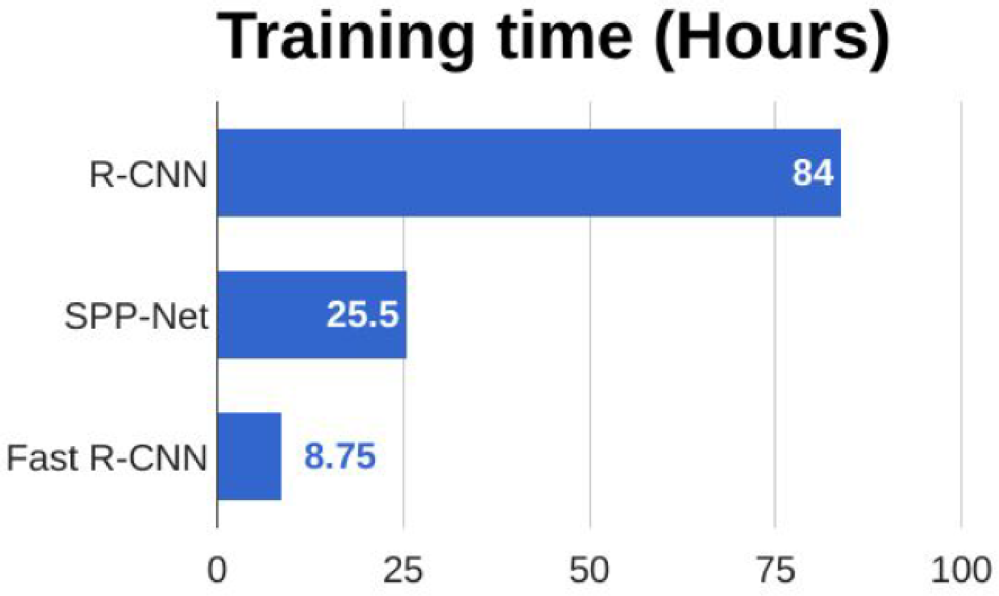
- Extract image regions
- 1 CNN per region (2000 CNNs)
- Classify region-based features

SPP-net & Fast R-CNN (the same forward pipeline)

- 1 CNN on the entire image
- Extract features from feature map regions
- Classify region-based features

Figure from http://kaiminghe.com/iccv15tutorial/iccv2015_tutorial_convolutional_feature_maps_kaiminghe.pdf

R-CNN vs Fast R-CNN



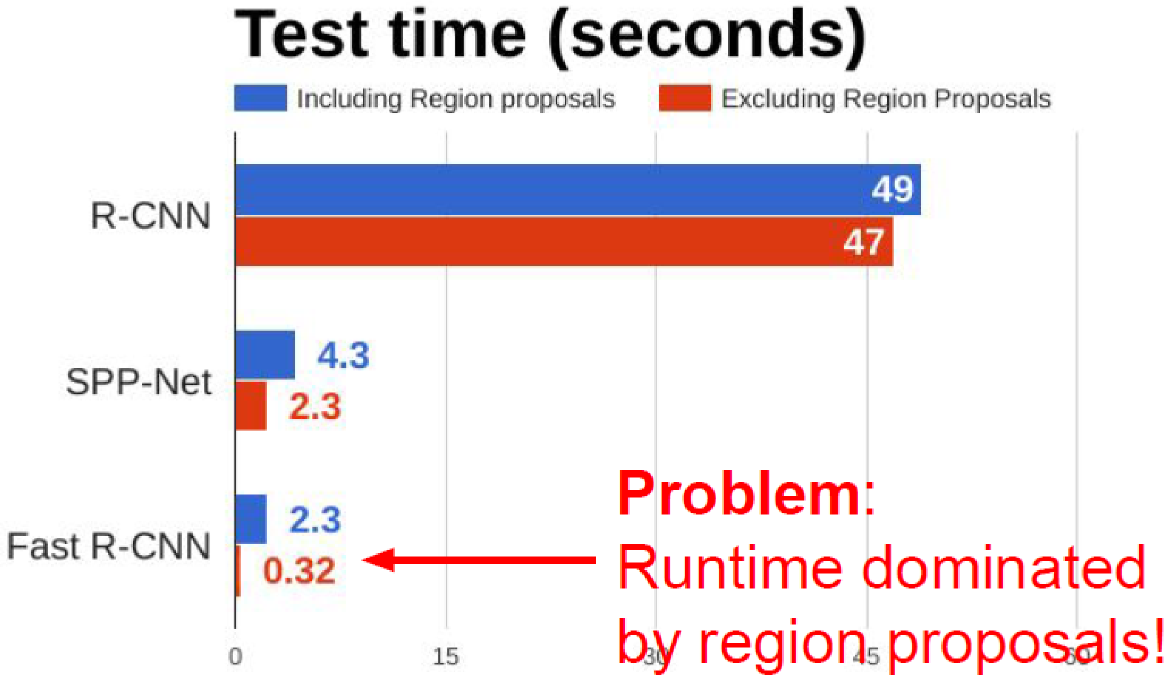


Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Problems of Fast R-CNN

- Out-of-network region proposals are the test-time computational bottleneck

- Object detection networks are fast but,
  - Region proposal
    - Selective Search [Uijlings et al. ICCV 2011]: 2s per image
    - EdgeBoxes [Zitnick & Dollar. ECCV 2014]: 0.2s per image

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

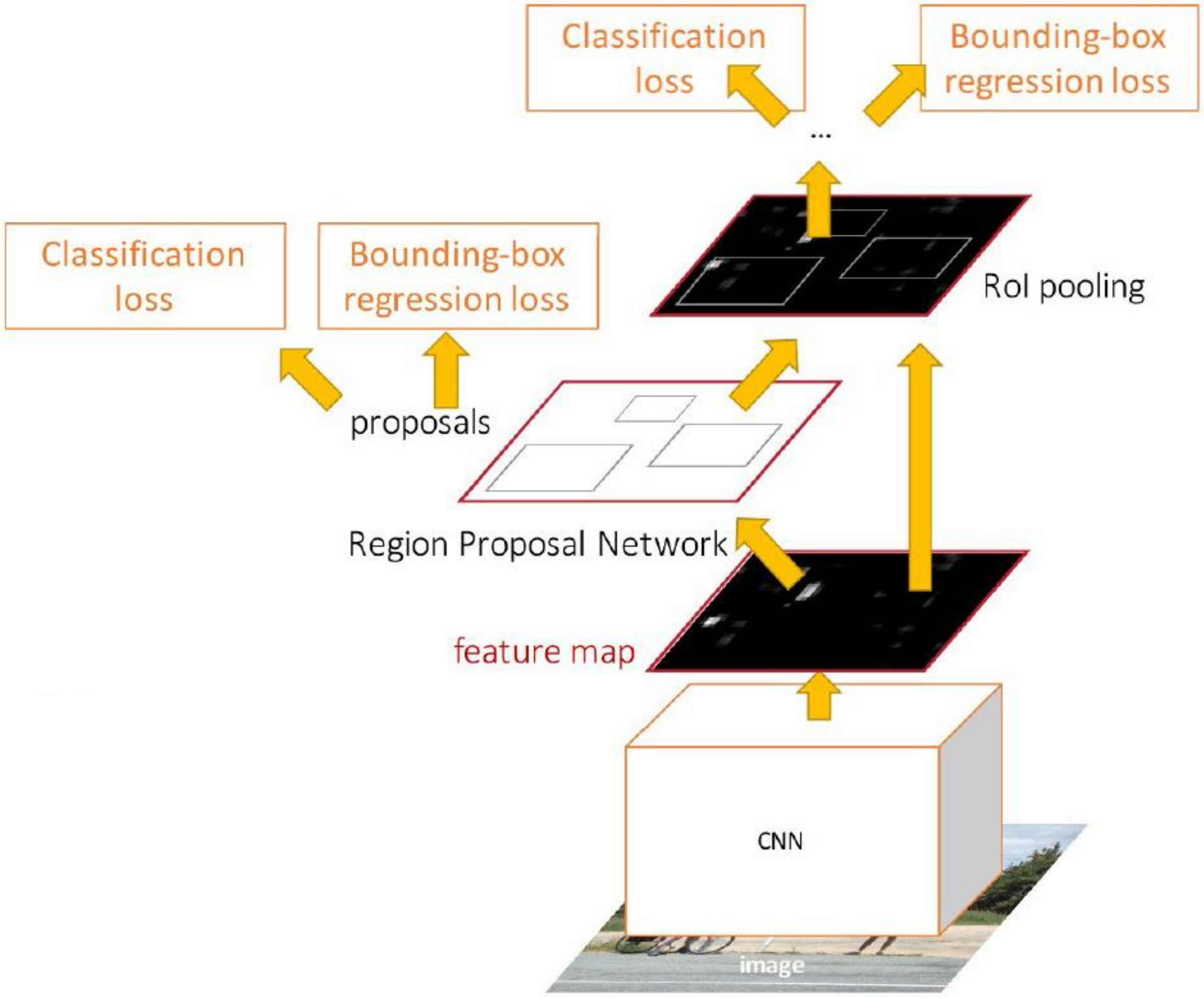


Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Training Goal: Share Feature Map

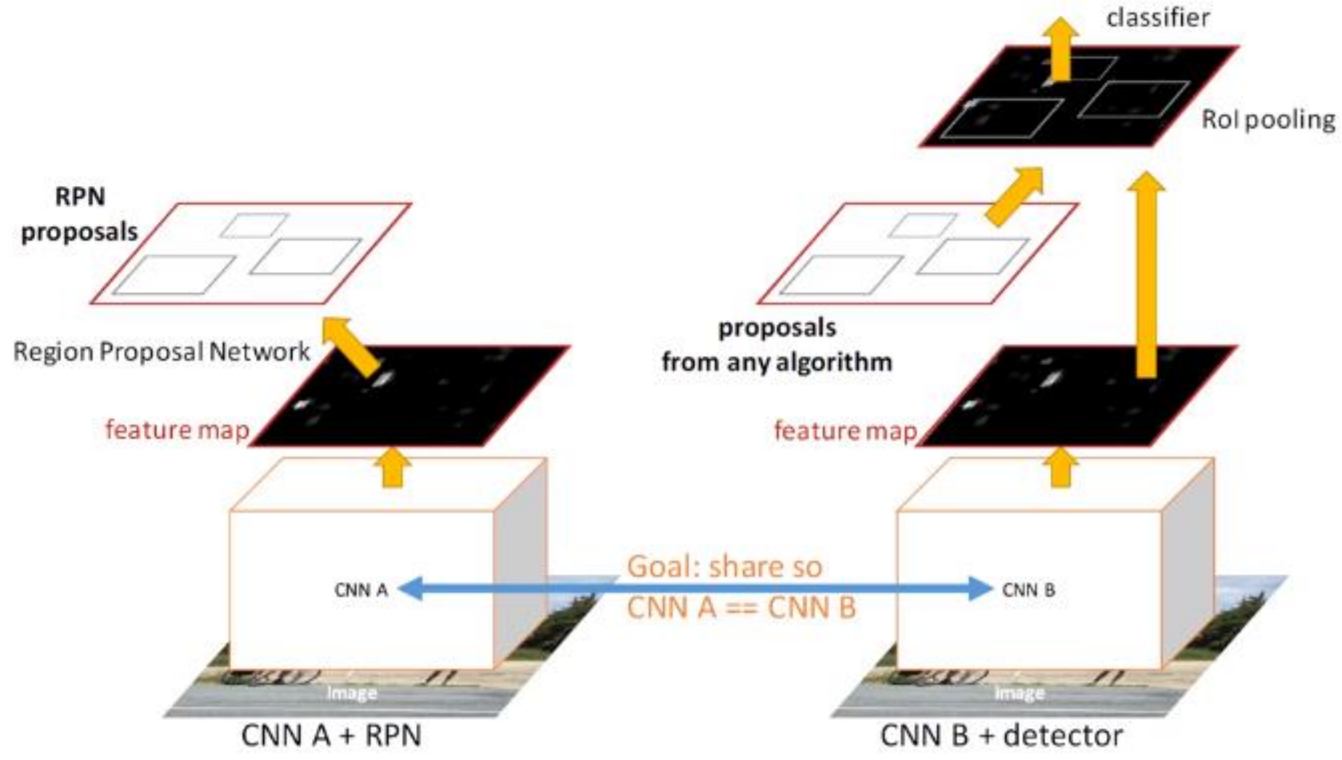


Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Faster R-CNN: Region Proposal Network

- Slide a small window on the feature map

- Build a small network for
  - Classifying object or not-object, and
  - Regressing bbox locations

- Position of the sliding window provides localizatio information with reference to the image

- Box regression provides finer localization information with reference to this sliding window

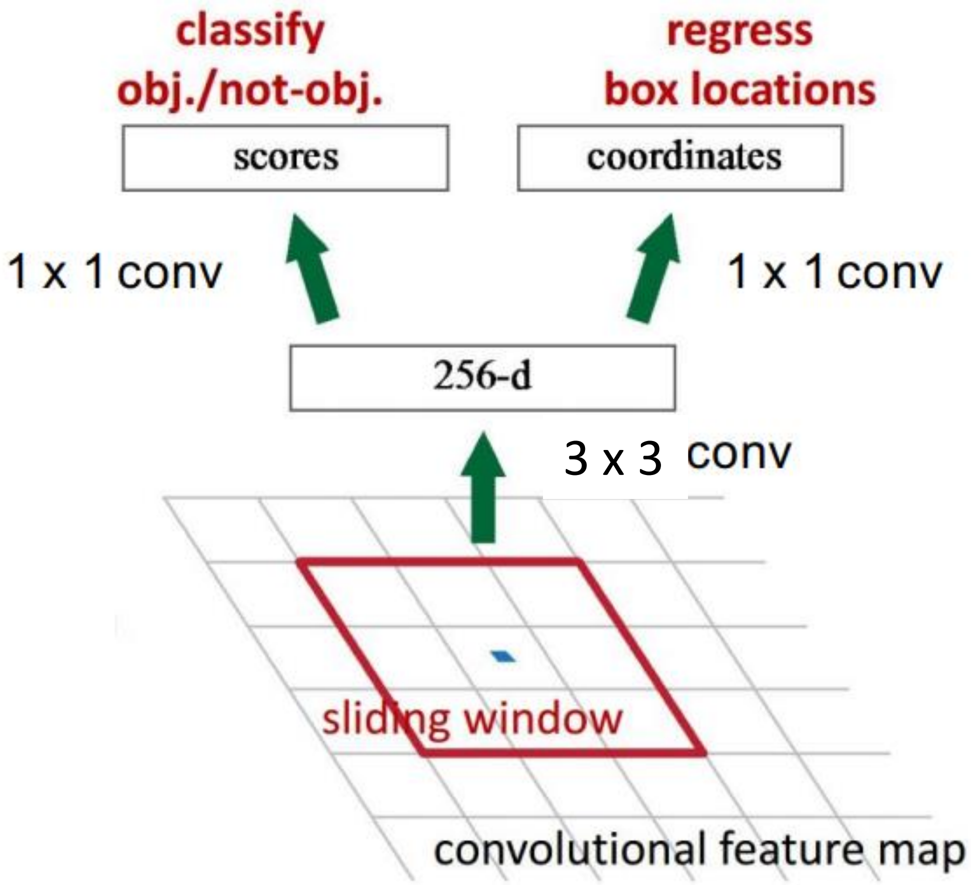


Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Faster R-CNN: Region Proposal Network

- Use N 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

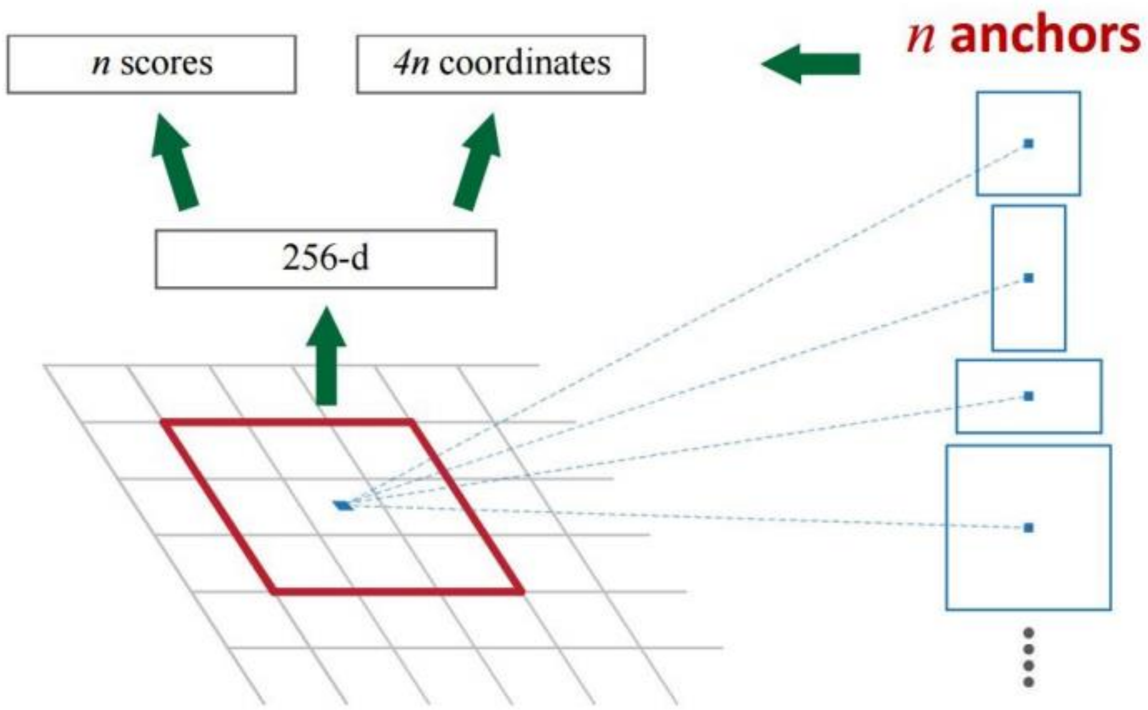


Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Faster R-CNN: Region Proposal Network

- Intermediate layer – 256(or 512) 3x3 filter, stride 1, padding 1
- Cls layer – 18(9x2) 1x1 filter, stride 1, padding 0
- Reg layer – 36(9x4) 1x1 filter, stride 1, padding 0

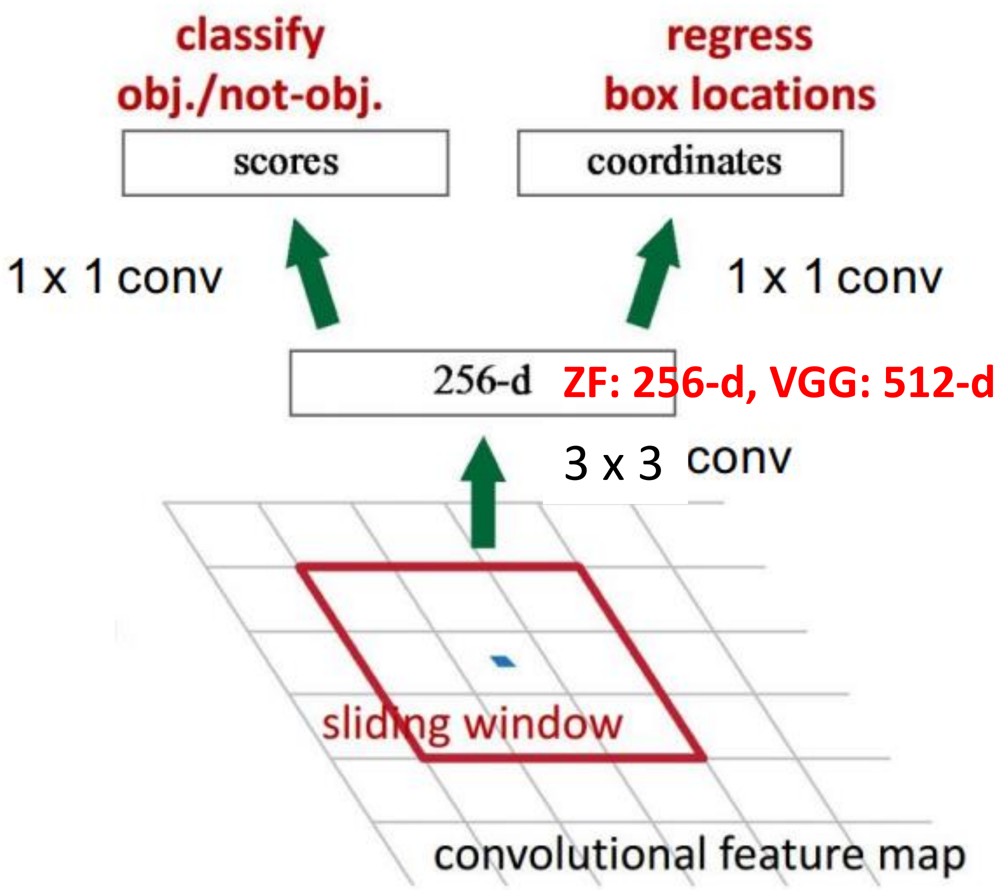


Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

Anchors

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

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: index of an anchor in a mini-batch

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

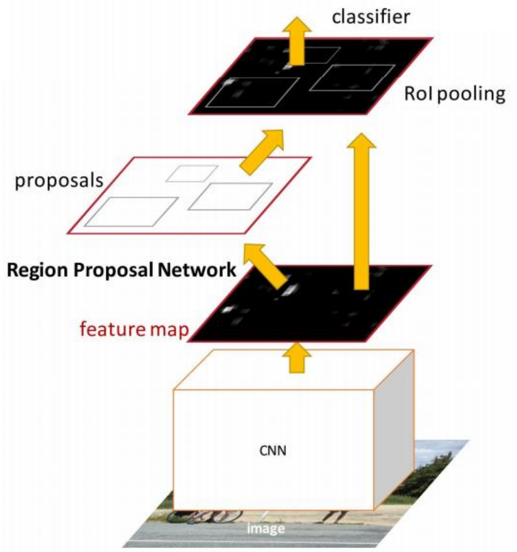
p_i: predicted probability of anchor i being an object
t_i: vector representing the 4 parameterized coordinates of the predicted bounding box
p_i^*: 1 if the anchor is positive, and 0 if the anchor is negative
t_i^*: ground-truth box associated with a positive anchor

L_{cls}: log loss over two classes (object vs. not object)
L_{reg}: smooth L1 loss

N_{cls}: number of anchors in mini-batch (~256)
N_{reg}: number of anchor locations (~2400)

λ: default is 10, so that both terms are roughly equally balanced

4-Step RPN Training



- M0 is 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 N2 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 sec	2 sec	**0.2 sec**
Speedup	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

Table 8: Detection results of Faster R-CNN on PASCAL VOC 2007 test set using **different settings of anchors**. The network is VGG-16. The training data is VOC 2007 trainval. The default setting of using 3 scales and 3 aspect ratios (69.9%) is the same as that in Table 3.

settings	anchor scales	aspect ratios	mAP (%)
1 scale, 1 ratio	128^2	1:1	65.8
	256^2	1:1	66.7
1 scale, 3 ratios	128^2	{2:1, 1:1, 1:2}	68.8
	256^2	{2:1, 1:1, 1:2}	67.9
3 scales, 1 ratio	$\{128^2, 256^2, 512^2\}$	1:1	**69.8**
3 scales, 3 ratios	$\{128^2, 256^2, 512^2\}$	{2:1, 1:1, 1:2}	**69.9**

Table 9: Detection results of Faster R-CNN on PASCAL VOC 2007 test set using **different values of λ** in Equation (1). The network is VGG-16. The training data is VOC 2007 trainval. The default setting of using $\lambda = 10$ (69.9%) is the same as that in Table 3.

λ	0.1	1	10	100
mAP (%)	67.2	68.9	69.9	69.1

Problems of Faster R-CNN

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

=> MASK R-CNN

Check the code

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

References

- Object detection
  - R-CNN: “Rich feature hierarchies for accurate object detection and semantic segmentation” (2013)
  - SPPNet: “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition” (2014)
  - Fast R-CNN: “Fast R-CNN” (2015)
  - Faster R-CNN: “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks” (2015)
  - SSD: “SSD: Single Shot MultiBox Detector” (2015)
  - Mask R-CNN: Mask R-CNN (2017)
- Others
  - Selective Search: “Selective Search for Object Recognition” (2012)
- Github
  - R-CNN
    - (Cafffe + MATLAB): https://github.com/rbgirshick/rcnn
  - Fast R-CNN
    - (Caffe + MATLAB): https://github.com/rbgirshick/fast-rcnn
  - Faster R-CNN
    - (tensorflow) https://github.com/endernewton/tf-faster-rcnn