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

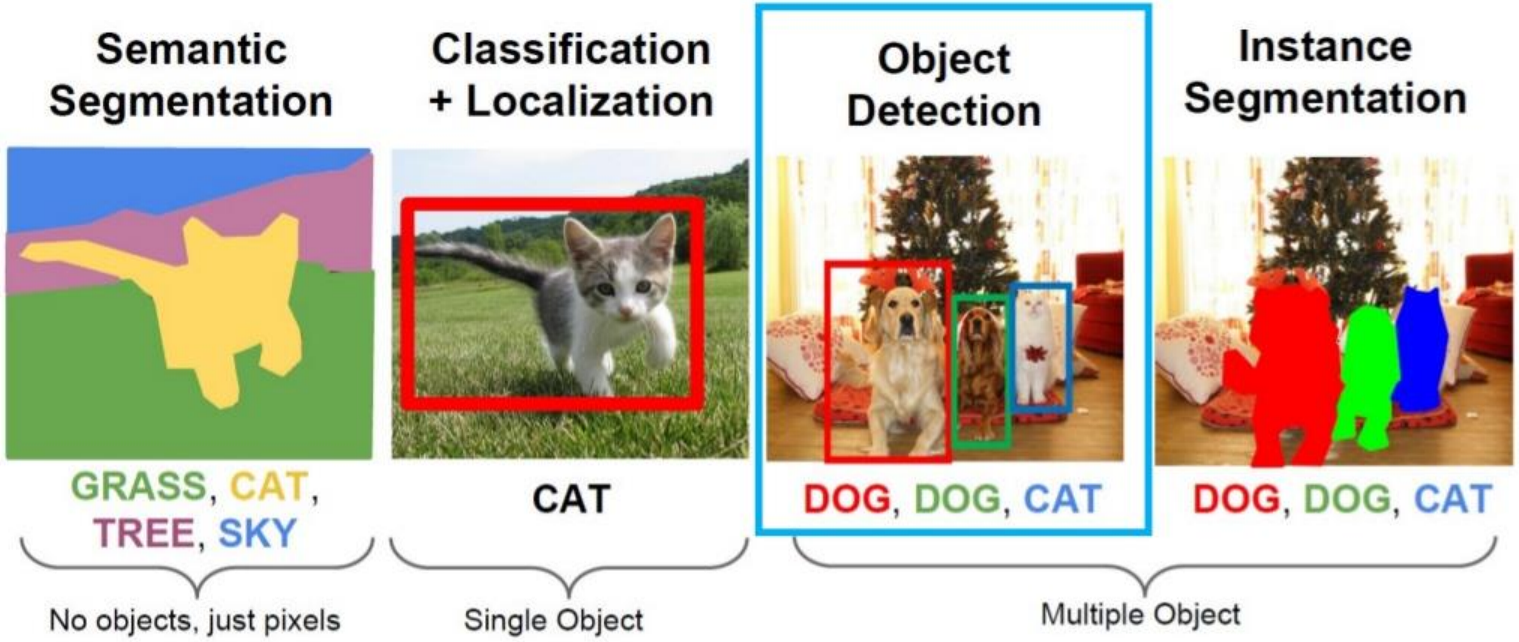
Theoretical Part

17기 이형구

Table of Contents

Computer Vision task

- Types of Computer Vison
  - One Object: Classification, Localization
  - Multiple Objects: Detection, Segmentation

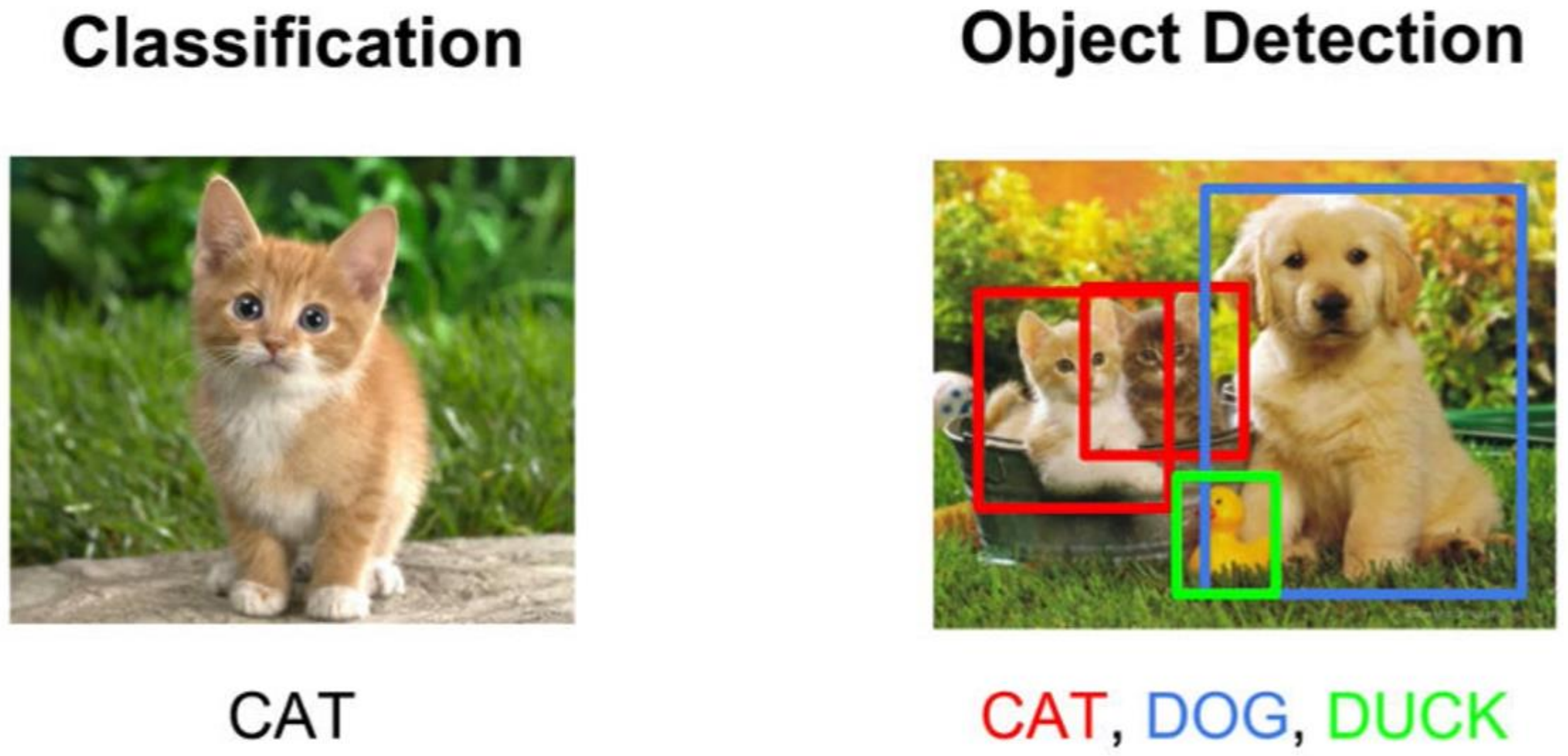


Detection task

- Types of Detection
  - One-stage Detector
    : Yolo, SSD, Retina-net
  - Two-stage detector
    : R-CNN, SPP-Net, Fast R-CNN, Faster- RCNN, Pyramid Networks

Object Detection

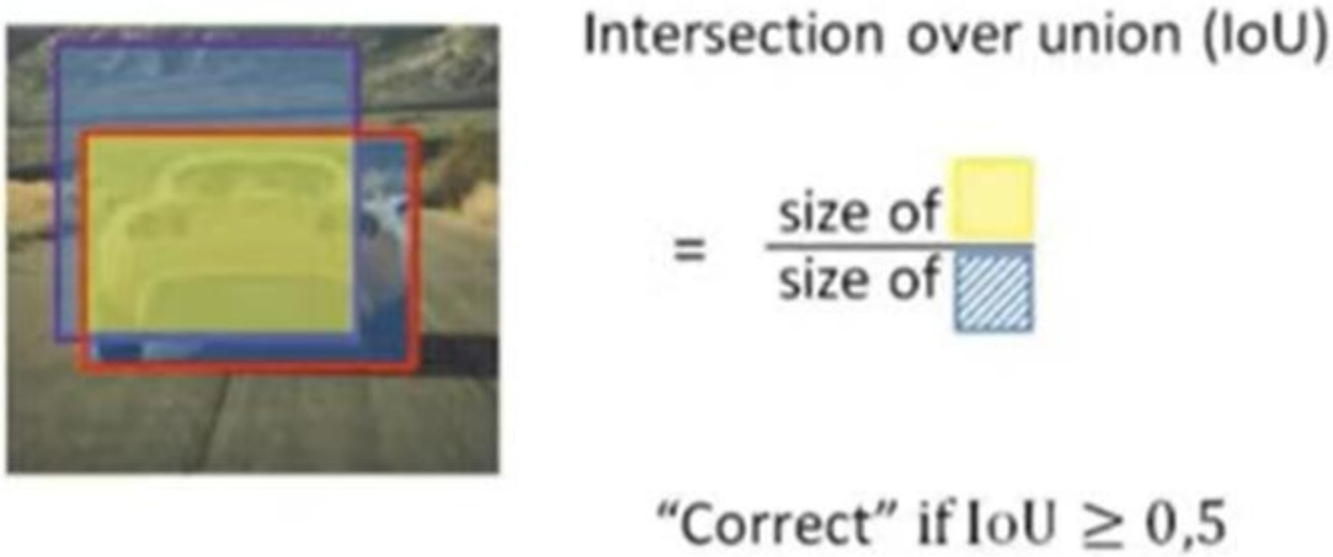
- Classification vs. Obiect detection



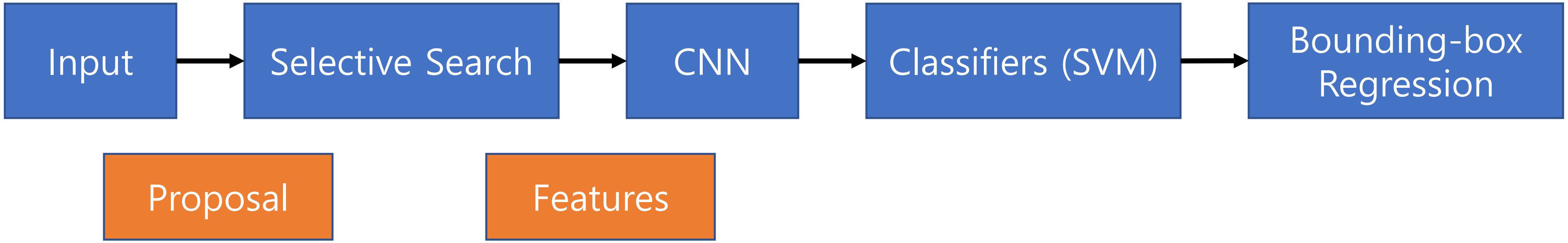
Prior Knowledge

- Selective Search
  - Bottom-up segmentation (merging regions at multiple scales)

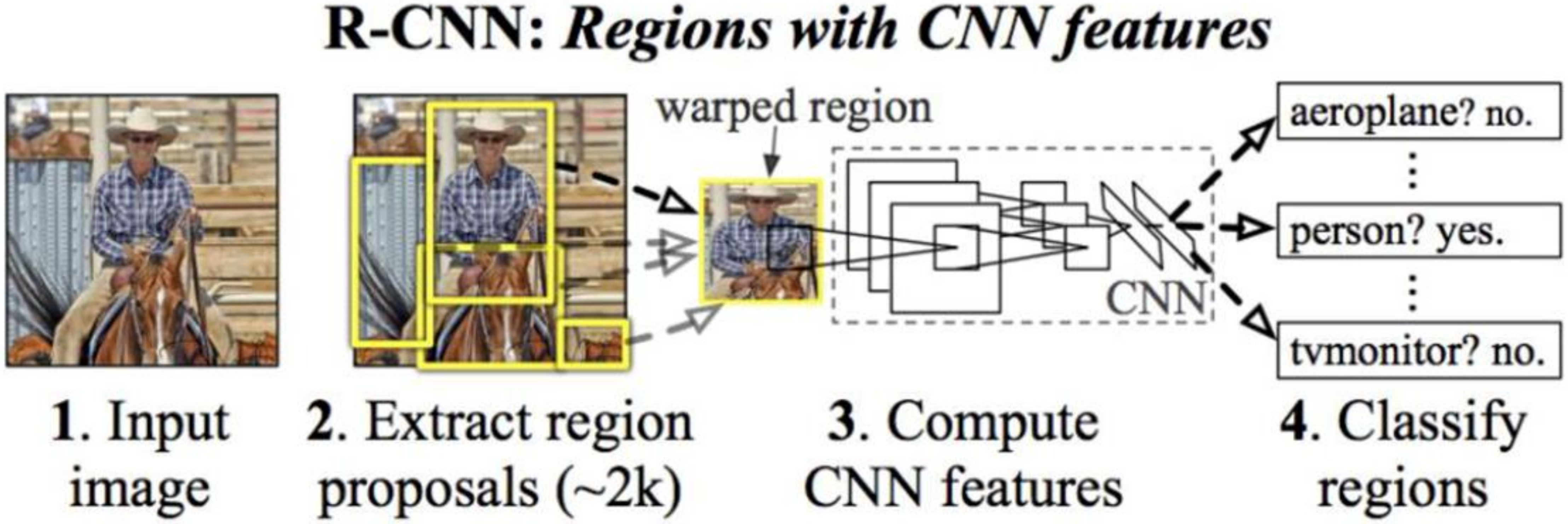
- IOU (Intersect Over Union)



1. R-CNN



- Input -> Selective Search -> CNN -> Classifiers(SVM) -> Bounding-box regression


R-CNN: Regions with CNN features
warped region
aeroplane? no.
person? yes.
tvmonitor? no.
CNN
1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

R-CNN

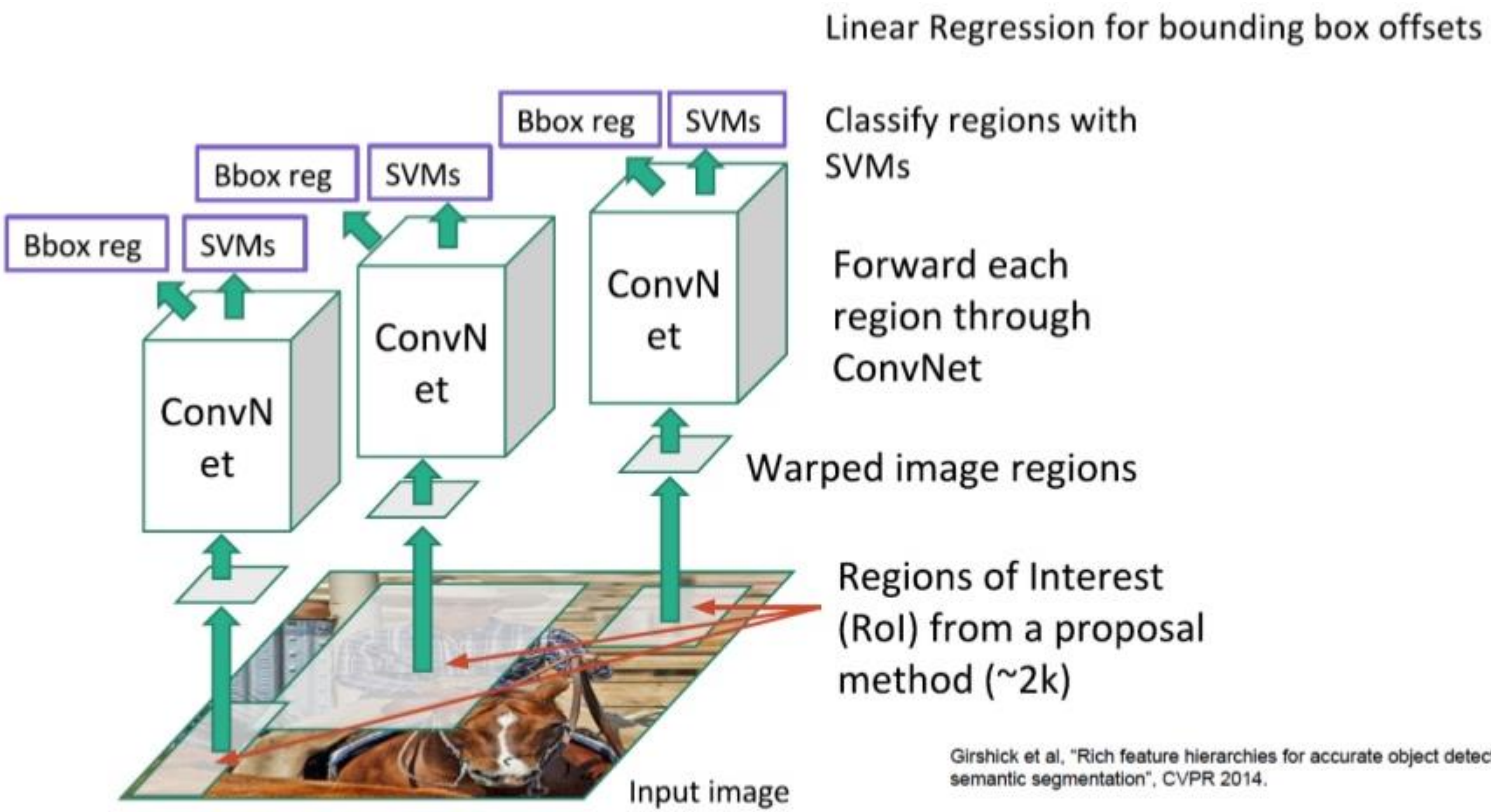
Characteristics of R-CNN

- Selective search
 - 2,000 region proposal

- Multi-stage
 - Convolutional fine tuning -> SVM classification -> Bounding-box regression

- Image -> Region -> Resize -> Convolution feature -> Classify

Architecture of R-CNN



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

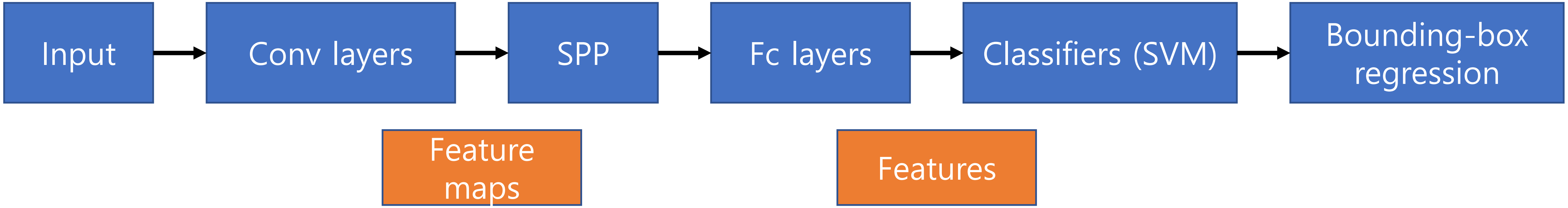
Problems of R-CNN

- Slow train speed
  - Need to apply CNN for each 2000 region proposal

- CNN features are not updated in response to SVM and Bounding-box regressors

- Complex multi-stage training pipeline
  - Requires 84 hours using K40 GPU

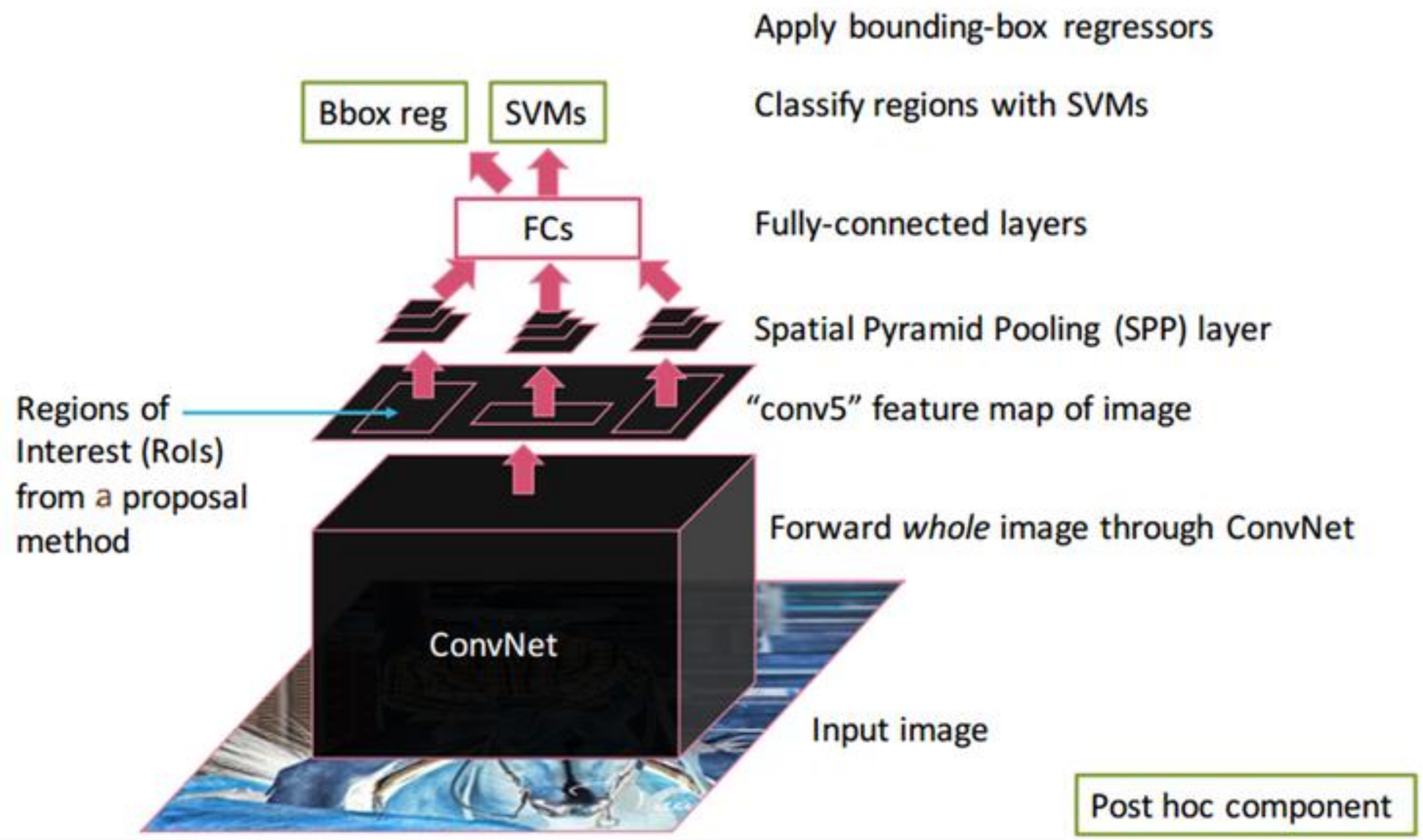
2. SPP-Net (Spatial Pyramid Pooling)



- R-CNN
  - Image -> Crop/Warp -> Conv layers -> Fc layers -> Output

- SPP-Net
  - Image -> Conv layers -> SPP -> Fc layers -> Output
    (Conv Features) (Region) (Classify)

Architecture of SPP-Net



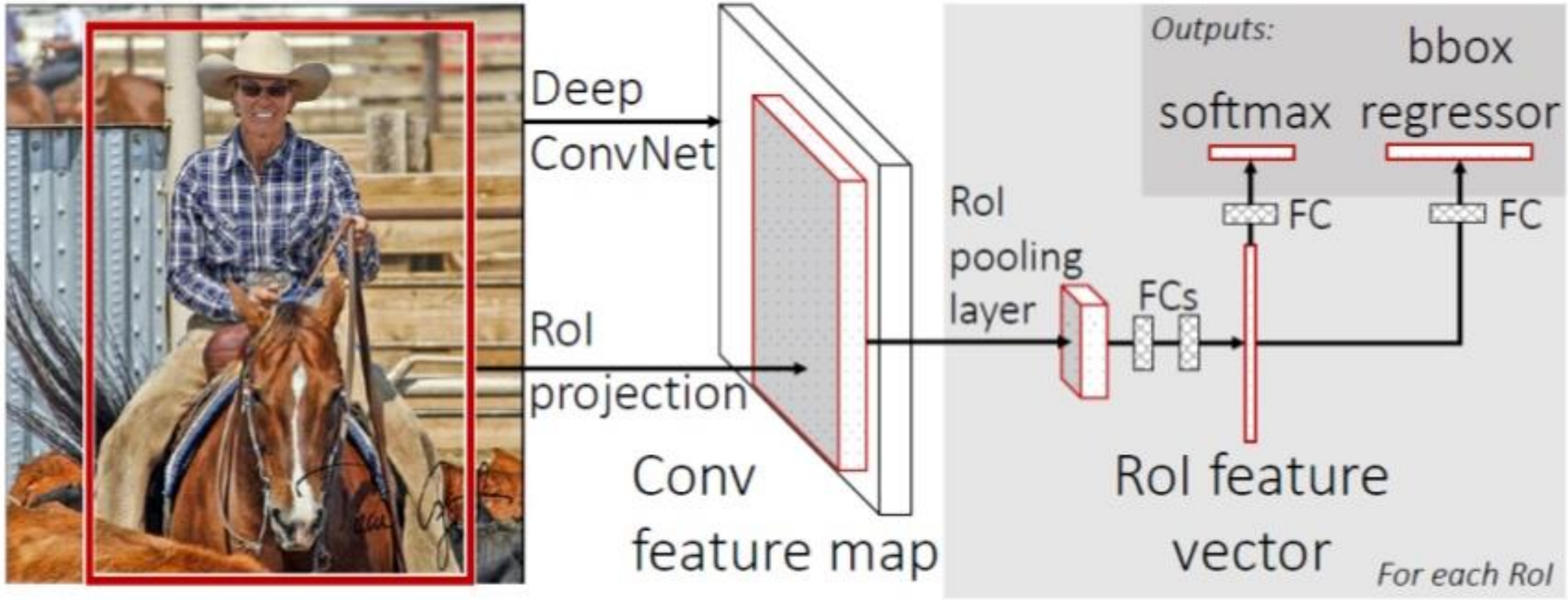
Still takes a lot of time!

- Multi-stage training
- Both time complexity and space complexity are too high
- Object Detection is slow

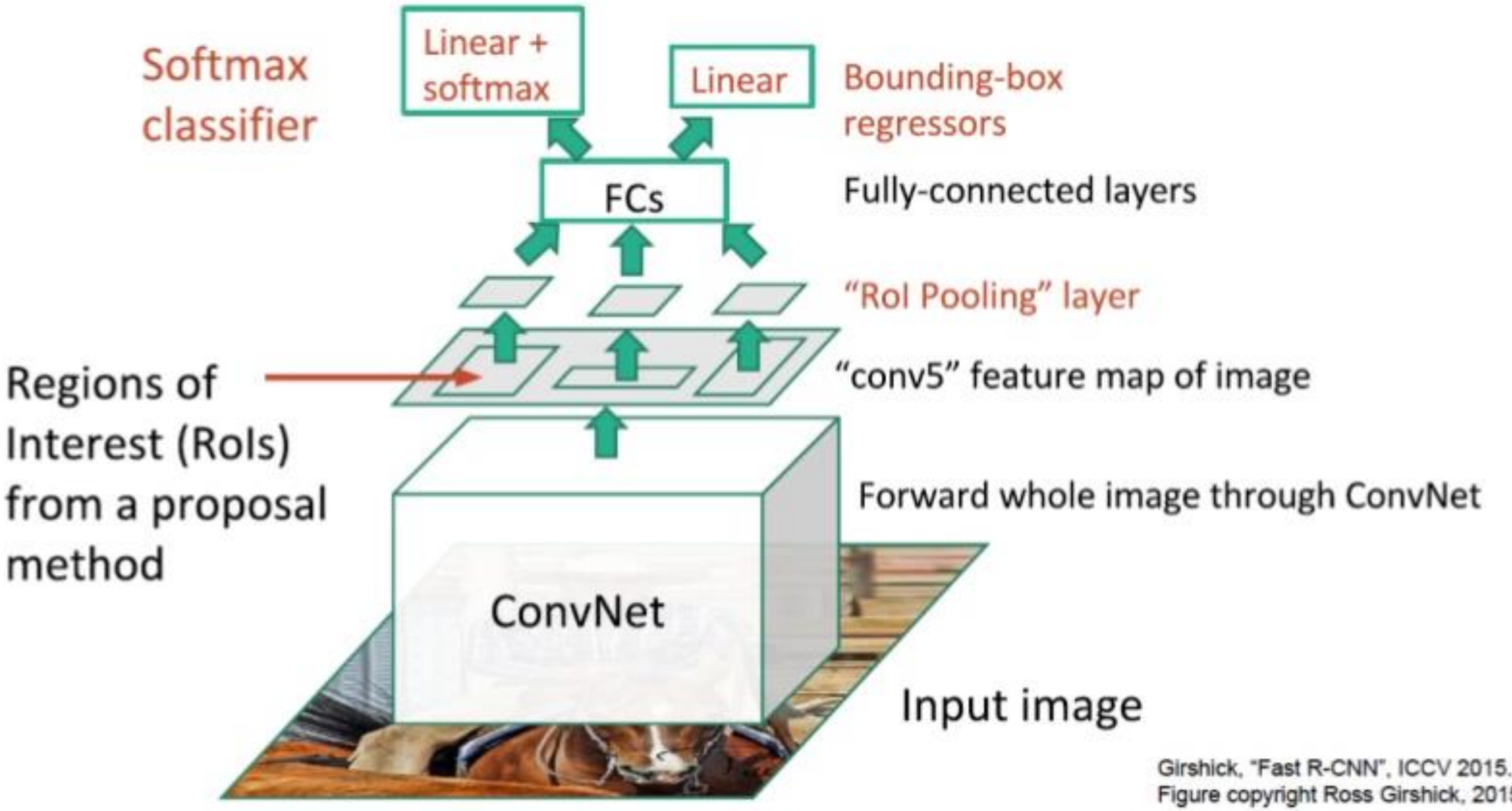
3. Fast R-CNN

- Single-stage
- Higher performance
- Faster testing

- Training Step
  - Input – Image & Object proposal
  - Create Convolutional feature map using the input image
  - Create Feature Vector through ROI pooling layer
  - Detect object class & adjust bounding box


Deep
ConvNet
RoI
projection
Conv
feature map
RoI
pooling
layer
FCs
Outputs:
softmax
bbox
regressor
FC
FC
RoI feature
vector
For each RoI

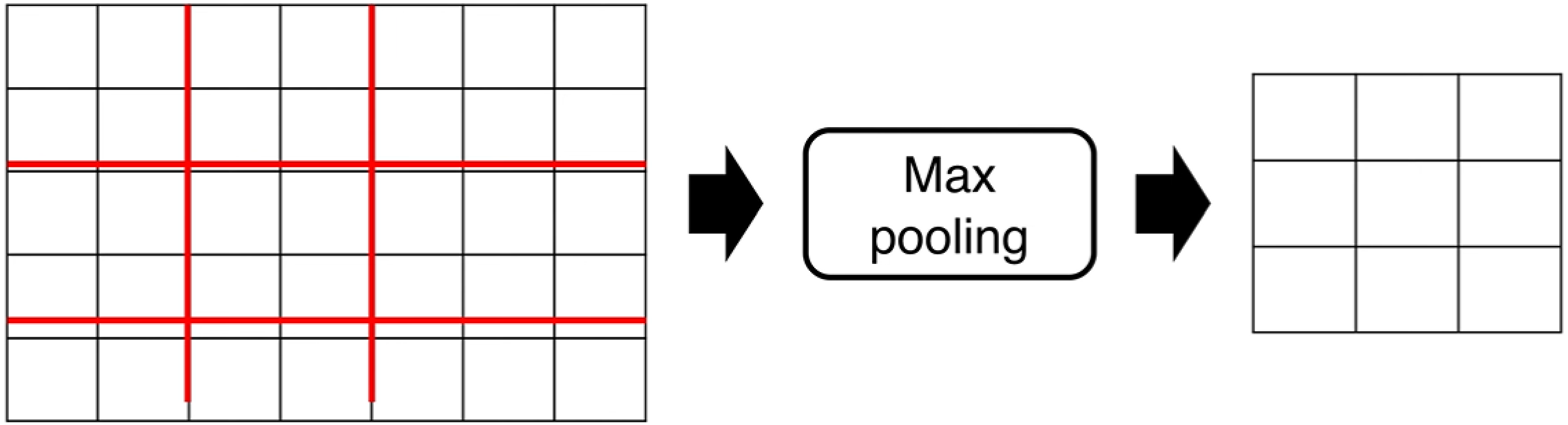
Architecture of Fast R-CNN

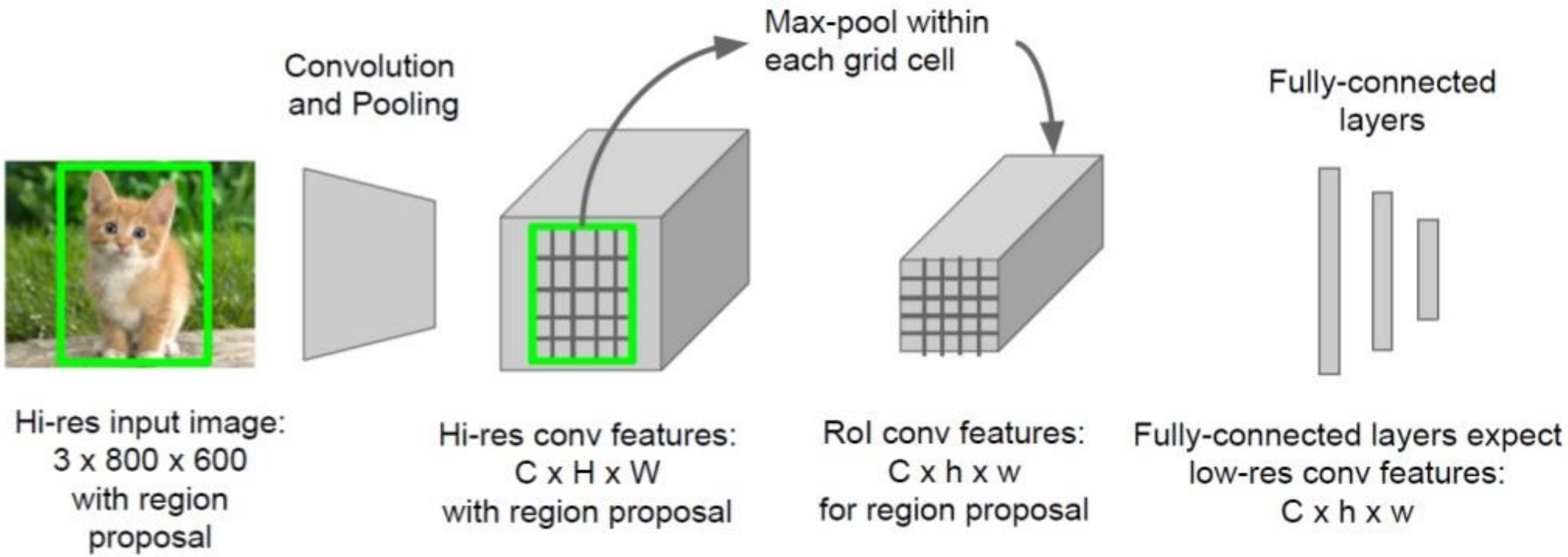


Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015;

What is ROI pooling layer?

- Reduce vector size into a fixed size through max pooling only in the ROI region
- Each ROI (r, c, h, w) -> (r, c): top-left corner coordinate
- 5x7 -> Max pooling -> 3x3




Convolution
and Pooling
Max-pool within
each grid cell
Fully-connected
layers
Hi-res input image:
3 x 800 x 600
with region
proposal
Hi-res conv features:
C x H x W
with region proposal
RoI conv features:
C x h x w
for region proposal
Fully-connected layers expect
low-res conv features:
C x h x w

Multi-task loss

- 두 개의 output layer
- 분류 : 각 RoI별 Discrete probability distribution(전체 K+1 카테고리)
- 회귀 : bounding box regression $t^k = (t_x^k, t_y^k, t_w^k, t_n^k)$
- u,v : ground truth class&target

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \underbrace{\lambda[u \geq 1]}_{\text{배경일 경우 0}} L_{\text{loc}}(t^u, v)$$

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

<분류>

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

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

<회귀>

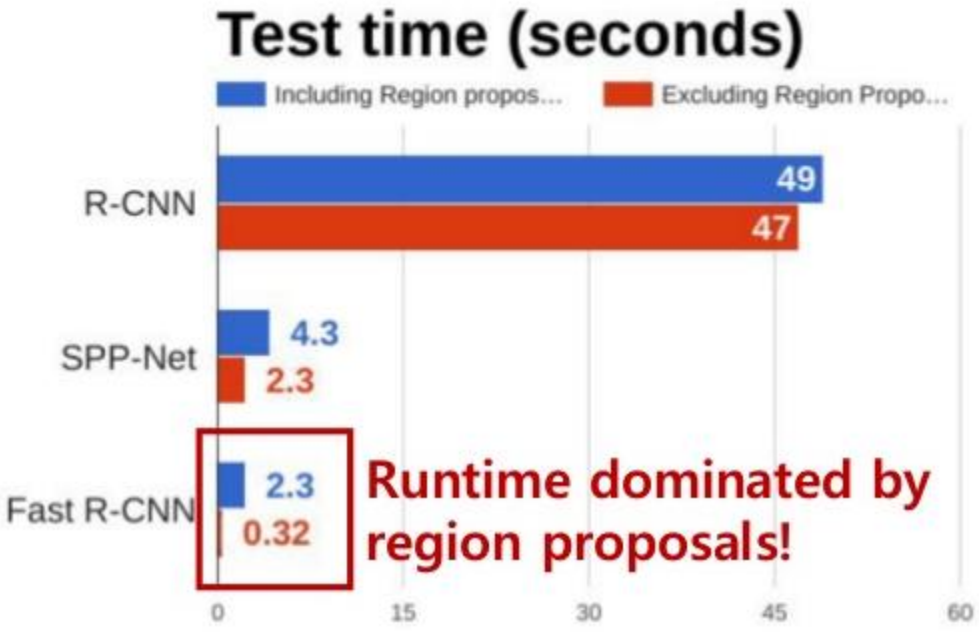
Mini-batch sampling

- N=2 mini batches
- R=128 (64 ROIs per image)

- 25% of ROI is positive sample (IOU >= 0.5)
- 0.1 < IOU < 0.5 indicates background (negative sample)

Test Results

- Slow when testing
- Why? Region Proposal takes a long time



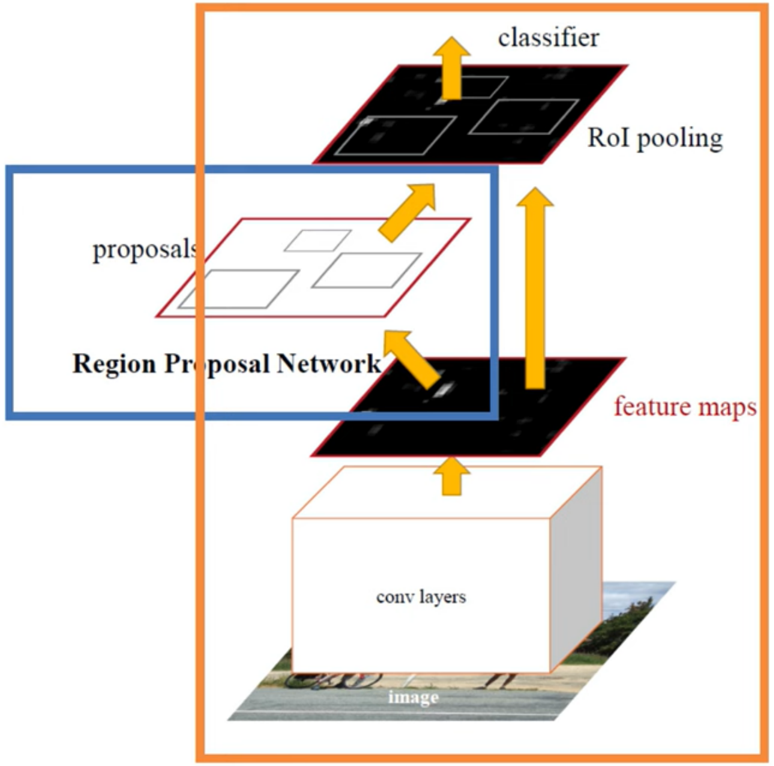
Conclusion

- Faster and more accurate than R-CNN and SPP-Net
- One-stage structure
- But still not fast enough

→ RPN (Region Proposal Network)

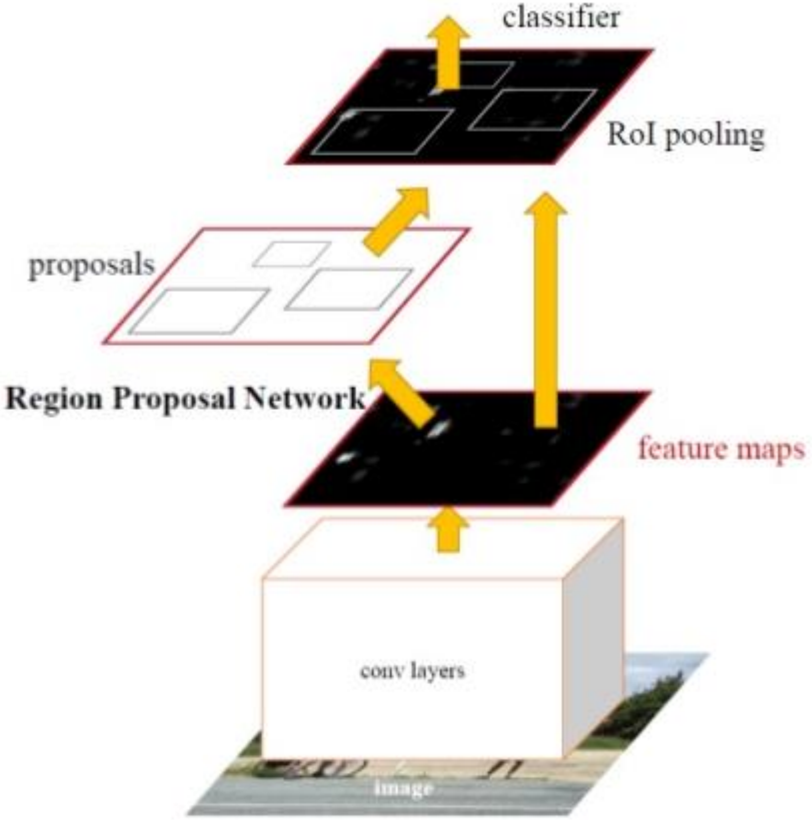
4. Faster R-CNN

- Apply Region proposal using Neural Network (CPU -> GPU)
- Feature map can be used for generating region proposal

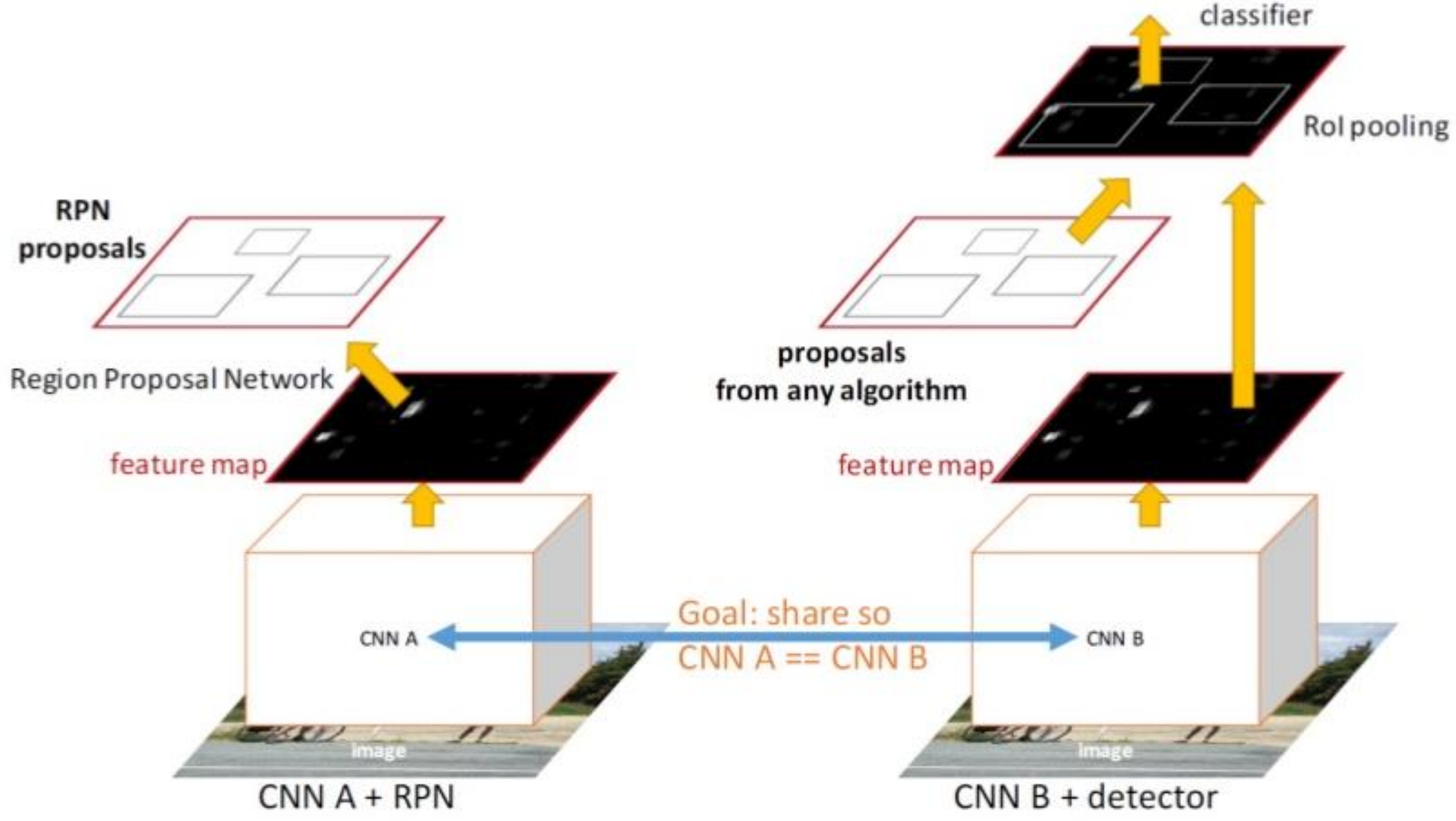


Architecture of Faster R-CNN

- 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

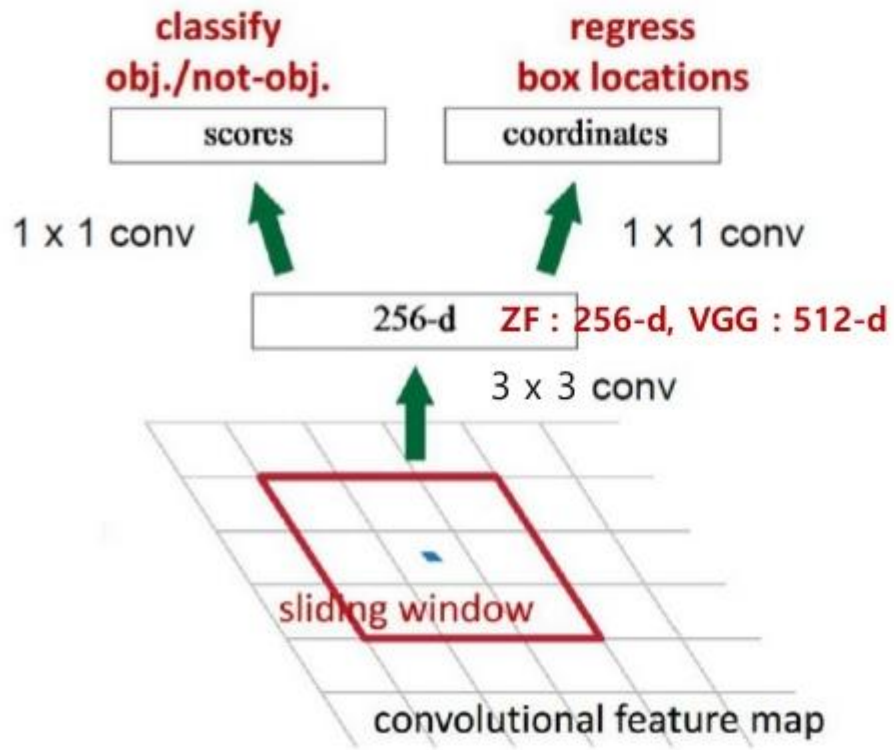


Training Step

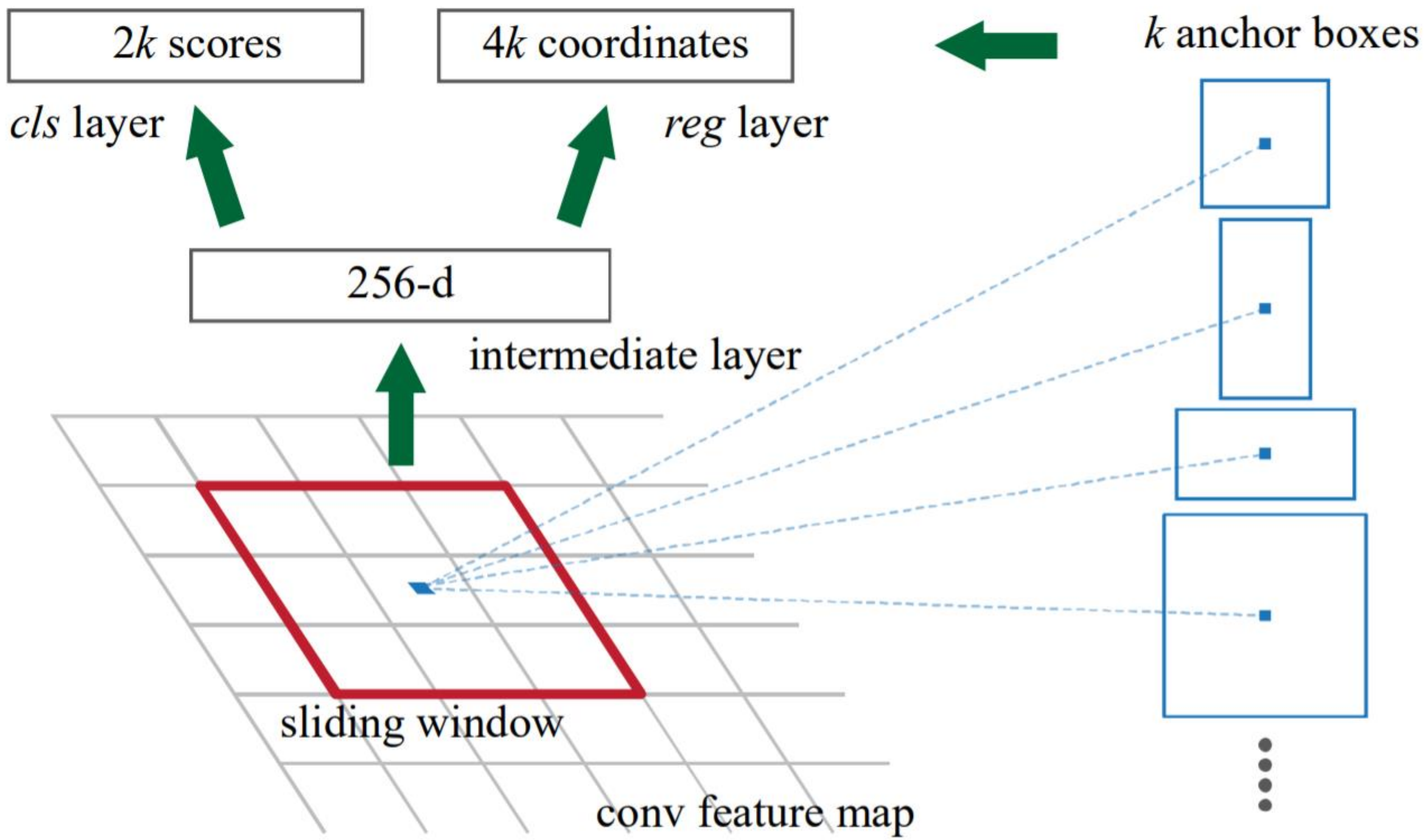
```
# Let M0 be an ImageNet pre-trained network

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
```

RPN (Region Proposal Networks)

- Slide a small window on the feature map
- Build a small network for both
  - Regressing bounding-box locations
  - Classifying object



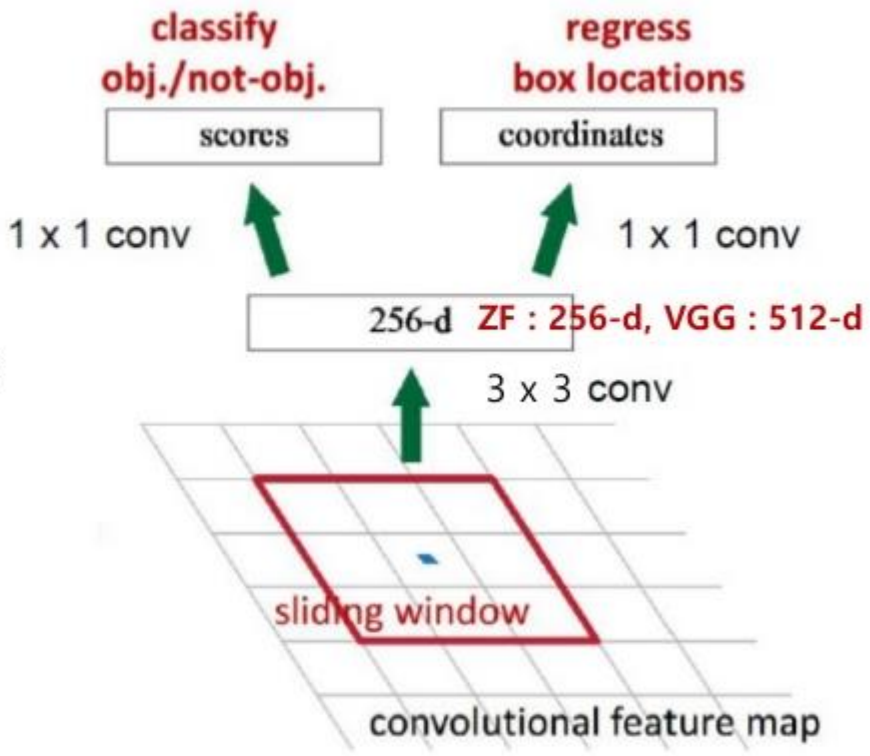
RPN – Anchor



2k scores
4k coordinates
k anchor boxes
cls layer
reg layer
256-d
intermediate layer
sliding window
conv feature map

RPN – Fully Convolutional 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



RPN – Loss Function

i = anchor index in minibatch

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i \underbrace{L_{cls}}_{\text{Log loss}}(p_i, \underbrace{p_i^*}_{\text{Ground truth objectness label}}) + \lambda \frac{1}{N_{reg}}\sum_i p_i^* \underbrace{L_{reg}}_{\text{Smooth L1 loss}}(t_i, \underbrace{t_i^*}_{\text{True box coordinates}}).$$

p_i: Predicted probability of being an object for anchor i

t_i: Coordinates of the predicted bounding box for anchor i

In practice λ = 10, so that both terms are roughly equally balanced

N_{cls} = Number of anchors in minibatch (~ 256)
N_{reg} = Number of anchor locations (~ 2400)

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 1: the learned average proposal size for each anchor using the ZF net (numbers for $s = 600$).

anchor	128^2, 2:1	128^2, 1:1	128^2, 1:2	256^2, 2:1	256^2, 1:1	256^2, 1:2	512^2, 2:1	512^2, 1:1	512^2, 1:2
proposal	188×111	113×114	70×92	416×229	261×284	174×332	768×437	499×501	355×715

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

Is it enough?

- ROI Pooling may introduce misalignments between ROI and extracted features

→ Mask R-CNN

References

- https://youtu.be/kcPAGIgBGRs
- https://youtu.be/Jo32zrxr6l8
- https://youtu.be/HmJWvwIpW5g