Object Detection

CNN, R-CNN, FAST/FASTER R-CNN

권지혜 이은진 명재성 구형석 유정아 원혜진

발표자 : 명재성, 유정아

KU-BIG

Object detection 시 고민 사항

직접 Faster R-CNN을 구현 VS. 기존 API 사용
*기존 API : Keras(Label 80~90개), Google Vision(Label 1900개)

Faster R-CNN 모델 (Label 600개)
TensorFlow Object Detection API 사용

목차

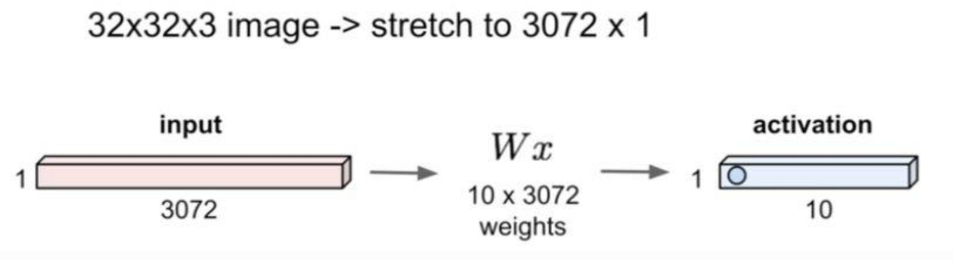
- CNN
- CNN Architectures
- R-CNN
- FAST R-CNN
- FASTER R-CNN

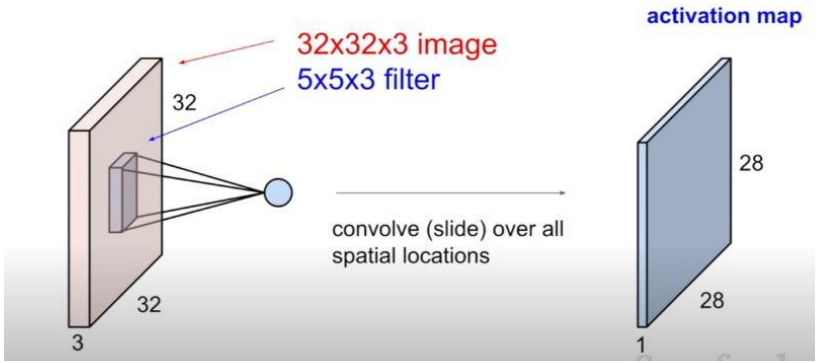
Reference : Stanford University CS234n, Spring 2017, 밑바닥부터 시작하는 딥러닝

1. Convolutional Neural Network
합성곱 신경망

Why do we use CNN? - 완전 연결 계층 vs. 합성곱 계층

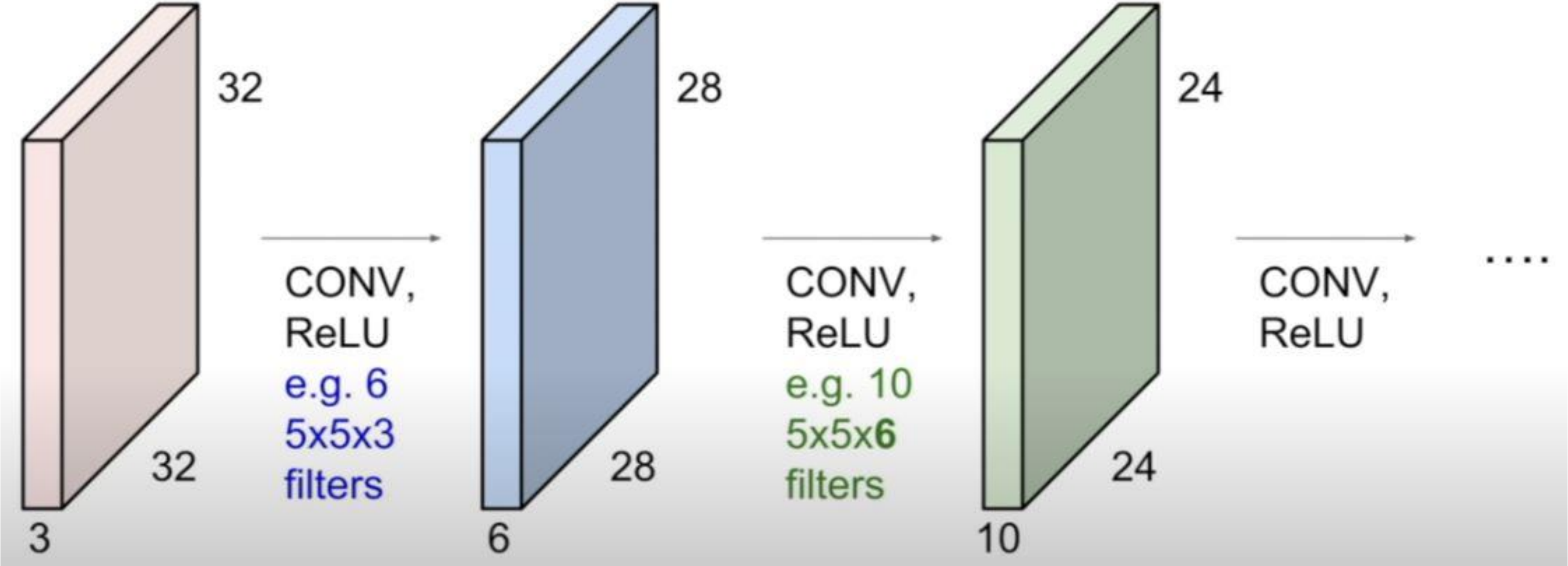
완전 연결 계층(Full Connected Layer)의 문제점 : 데이터의 형상이 무시된다





KU-BIG

CNN example



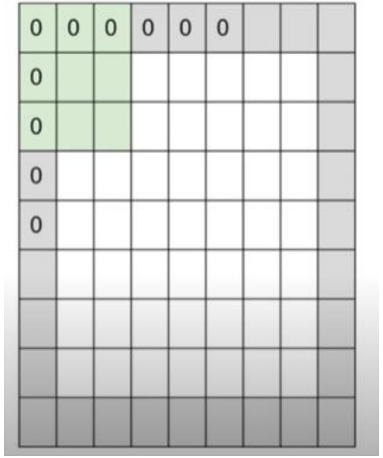
32
32
3
CONV,
ReLU
e.g. 6
5x5x3
filters
28
28
6
CONV,
ReLU
e.g. 10
5x5x**6**
filters
24
24
10
CONV,
ReLU
....

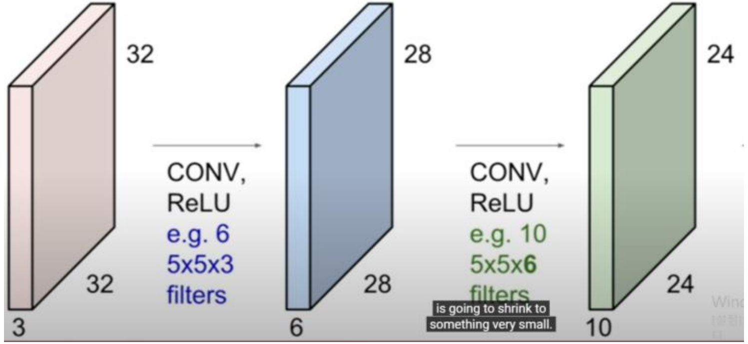
Padding

Padding : input image 테두리를 0으로 채움

목적 : Output의 size를 input과 동일하게 유지하기 위함

반복적으로 convolve를 진행하다 보면, output size가 큰 폭으로 줄어듦





Common Settings

K : Number of Filters
Powers of 2 (32, 64, 128, …)

F : Spatial extent of Filters (i.e. filter = FxF 모양)
3 또는 5

S : Stride
1 또는 2
클수록 down-sampling

P : amount of zero padding
Whatever that makes filter fit

Pooling

Pooling layer

역할 : down-sample input volume

(단, depth는 변화가 없음. Only pooling spatially)

E.g. 224x224x**64** -> 112x112x**64**

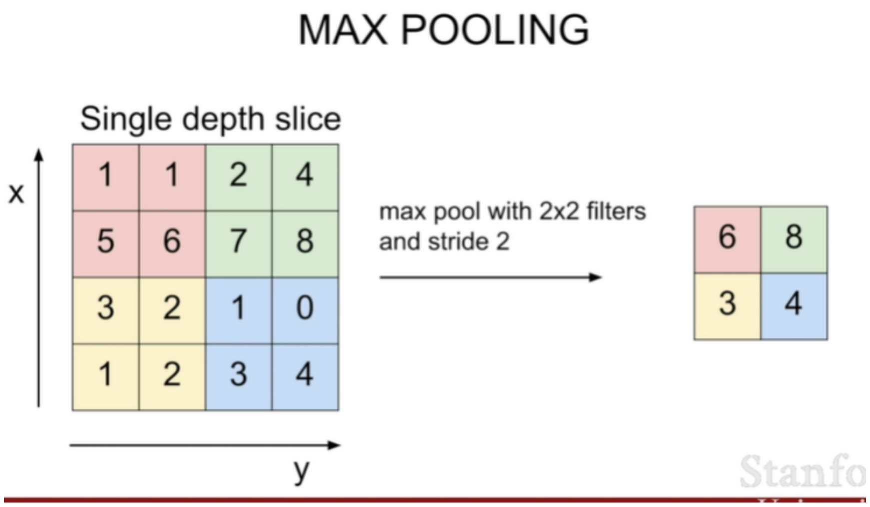
Max pooling

Stride하면서, max value를 뽑아 냄

Pooling layer common settings

F : spatial extent, S : stride

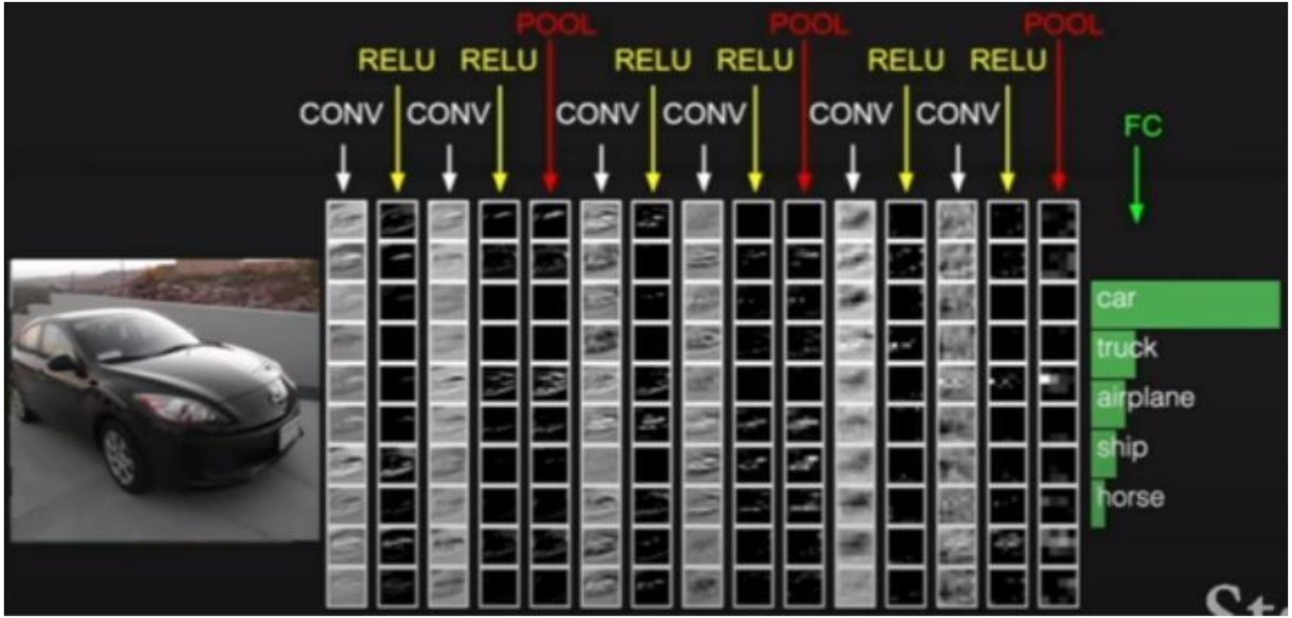
F = 2, S = 2



KU-BIG

[(CONV-RELU)*N – POOL]*M – (FC-RELU)*K, SOFTMAX

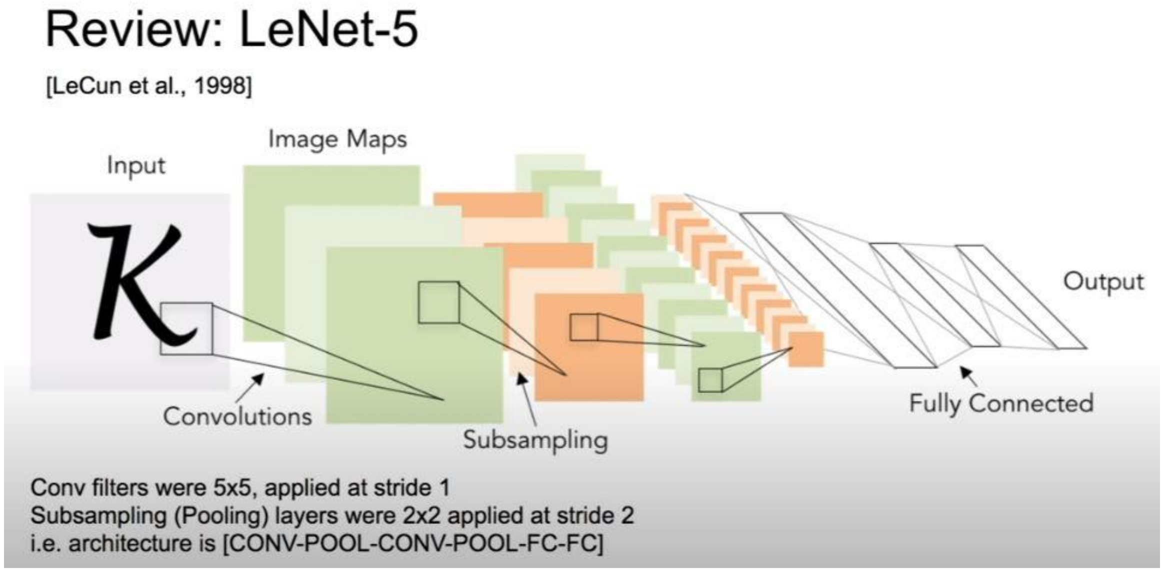
(N up to 5, M is large, 0<=K<=2)



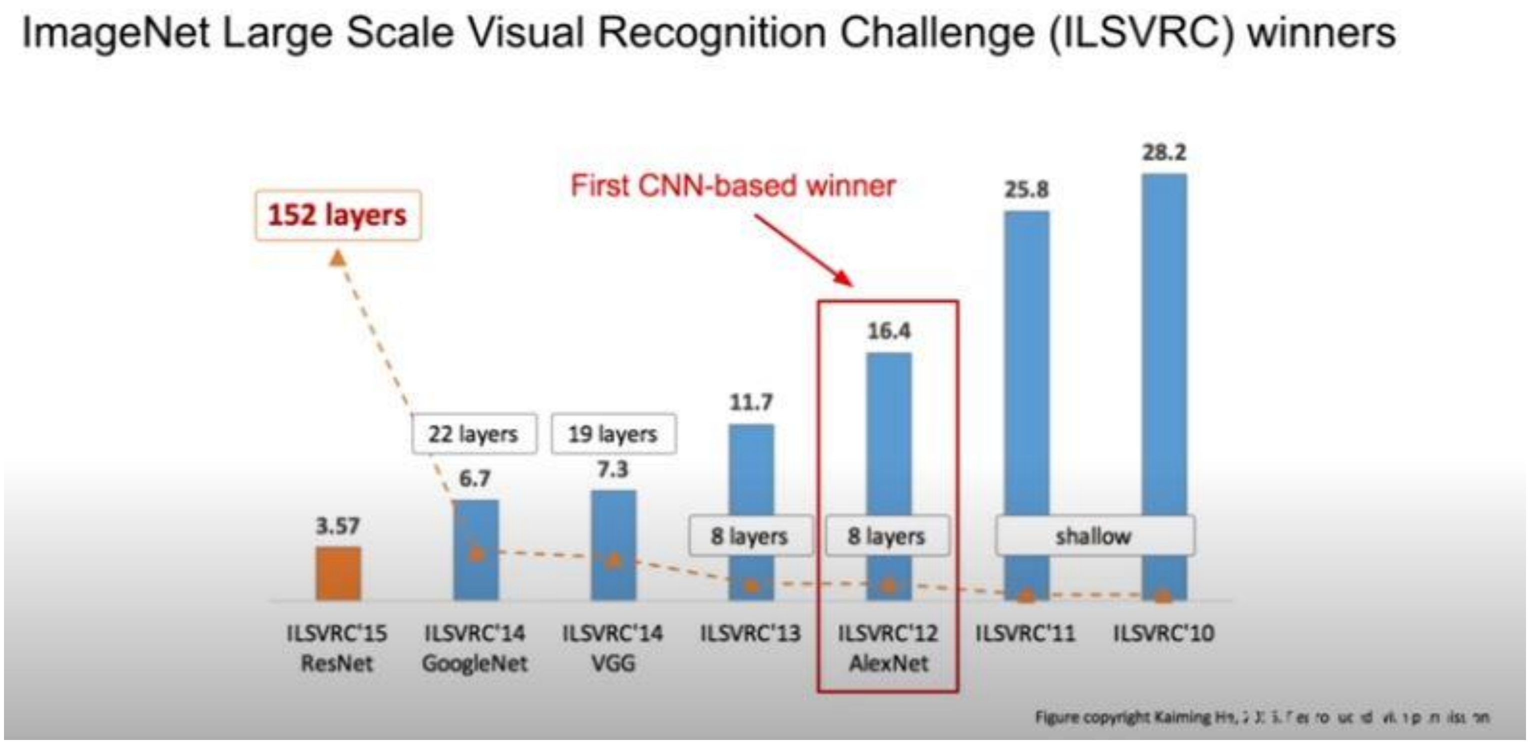
2. CNN Architectures

LeNet-5, AlexNet, VGG, GoogLeNet, ResNet

LeNet-5 (LeCun et al., 1998) 손글씨 숫자 인식하는 네트워크



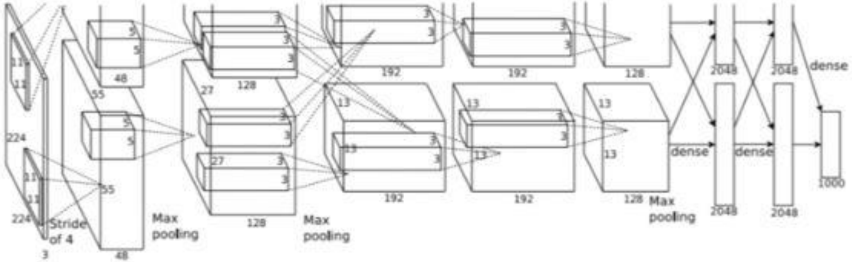
ILSVRC Winners



Case Study 01 - AlexNet[Krizhevsky et al. 2012]

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Case Study 02 - Deeper Networks #1 : VGGNet

Case Study: VGGNet

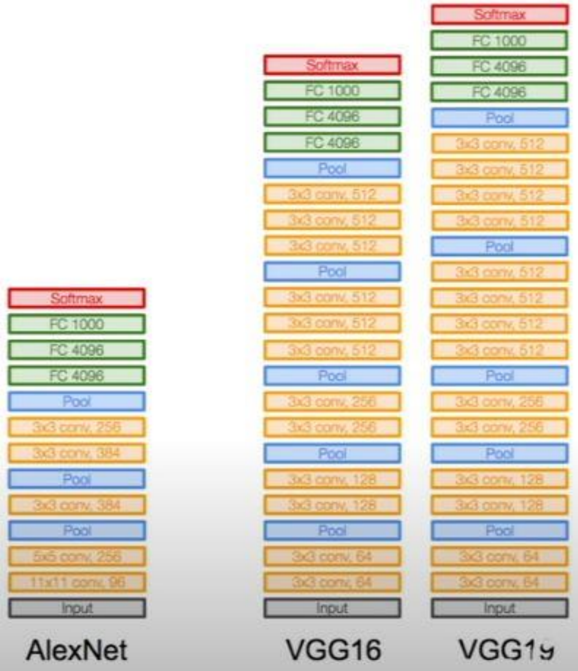
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

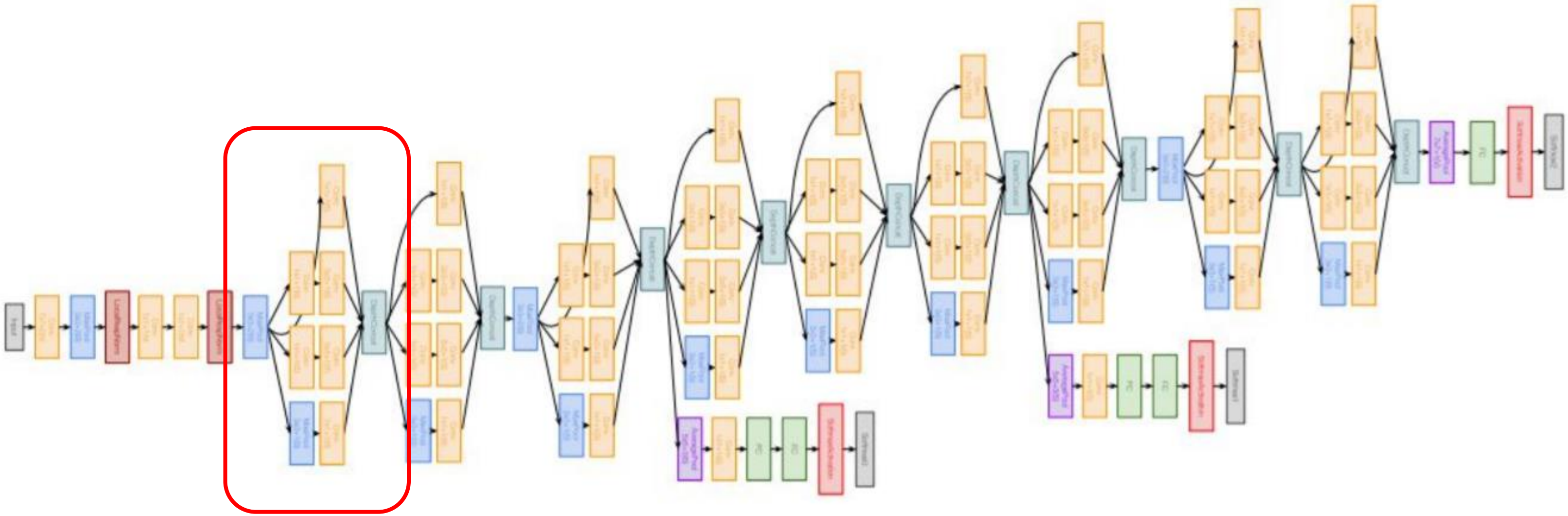
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

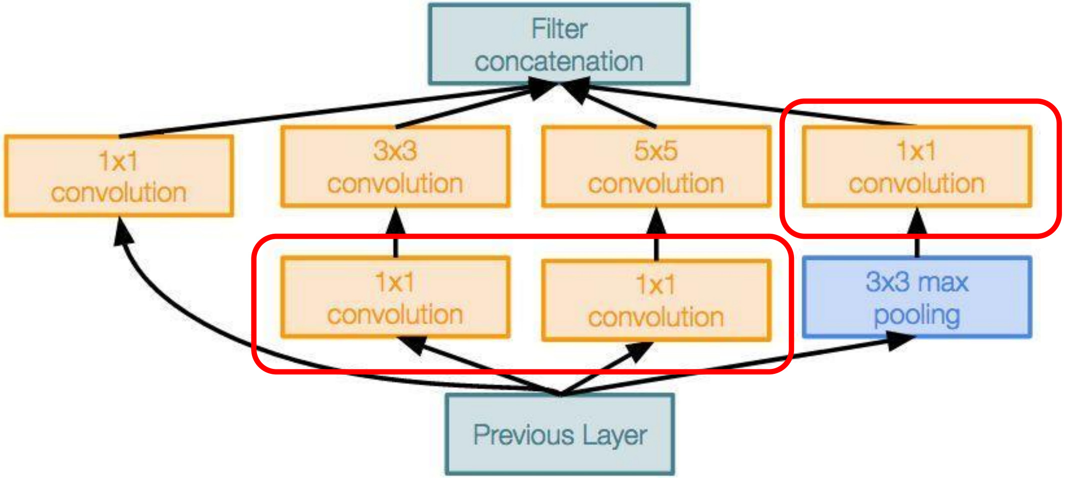


KU-BIG

Case Study 03 - Deeper Networks #2 : GoogLeNet

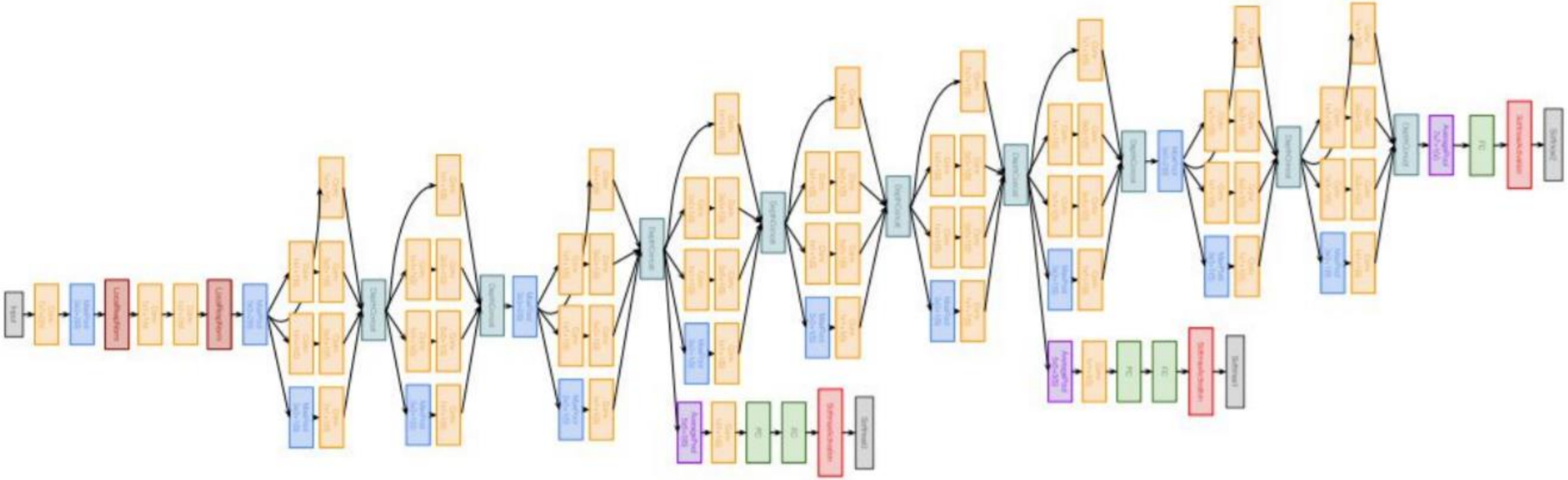


Inception Modules



Dimension reduction

Full GoogLeNet architecture

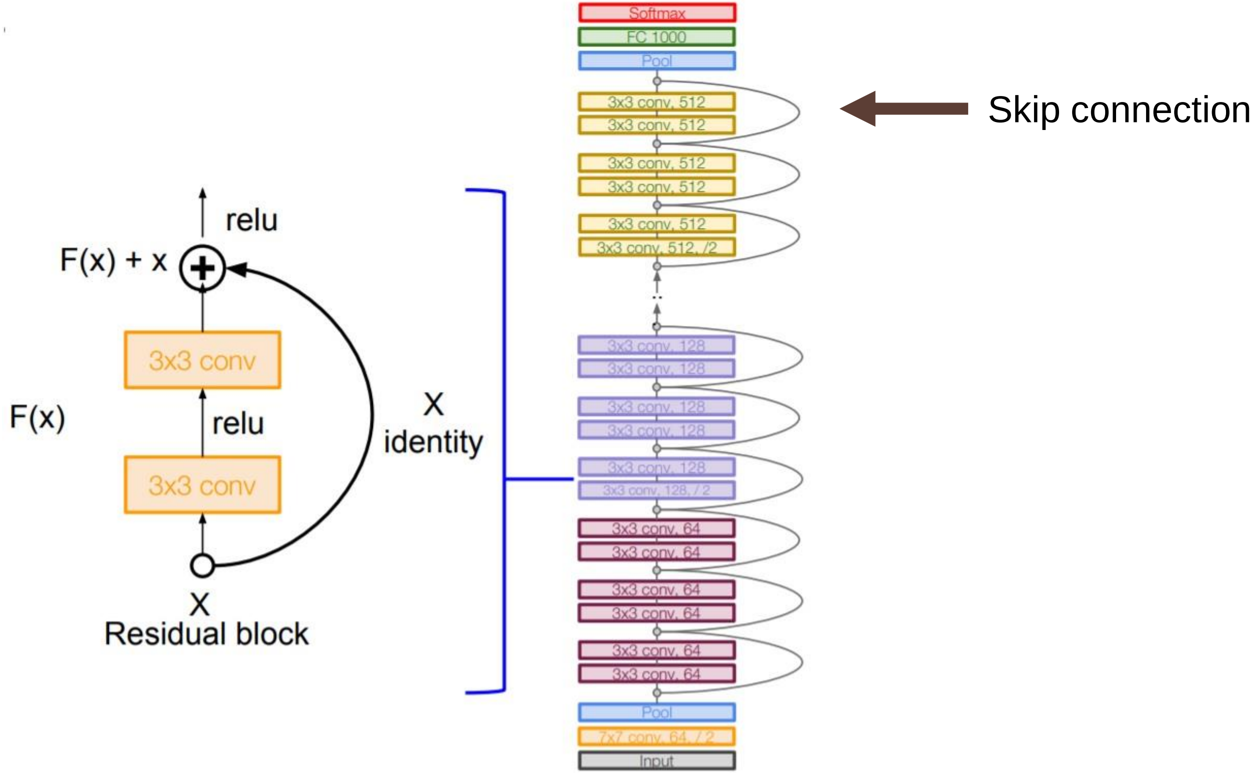


Stem Network

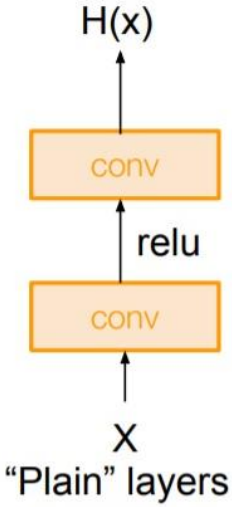
Stacked inception modules

Classifier Output

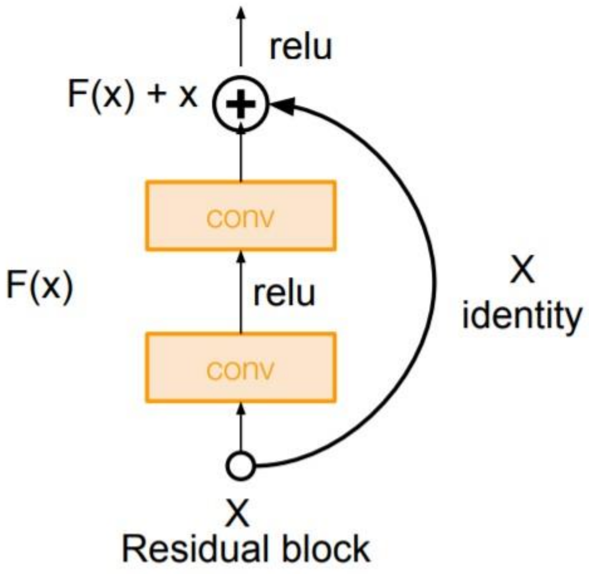
Case Study 04 - Deeper Networks #3 : ResNet



ResNet



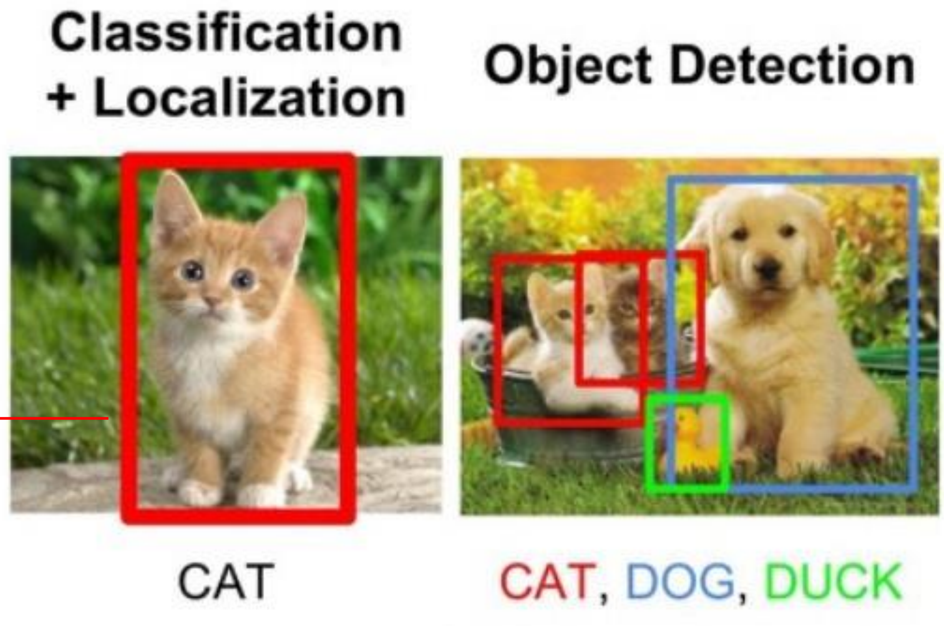
H(x) = F(x) + x



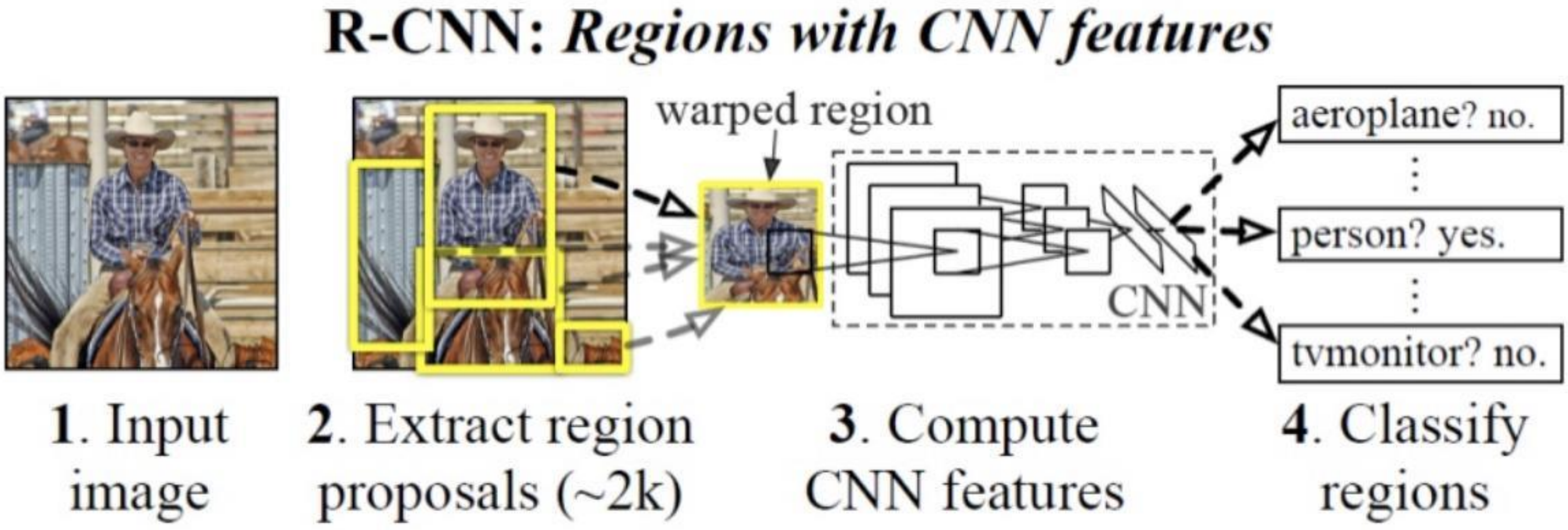
3. R-CNN

Computer vision task

Bounding box

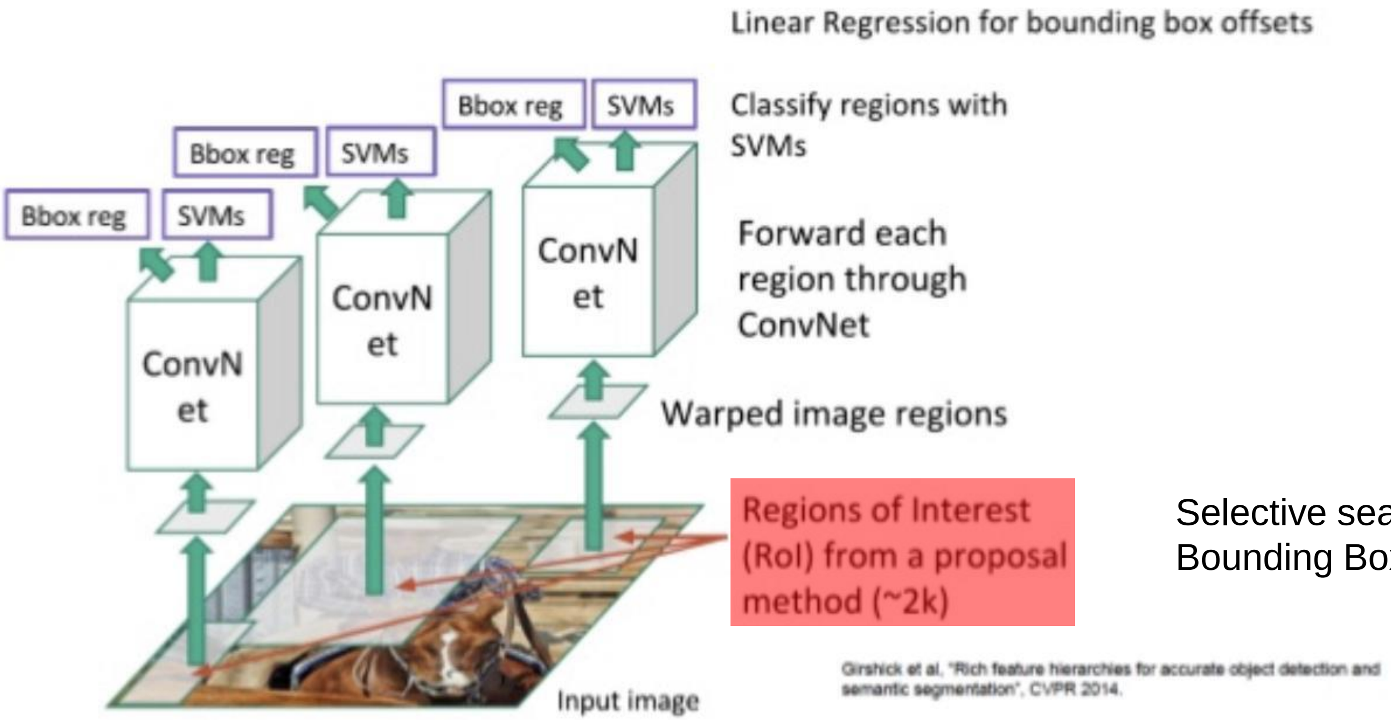


R-CNN Overview



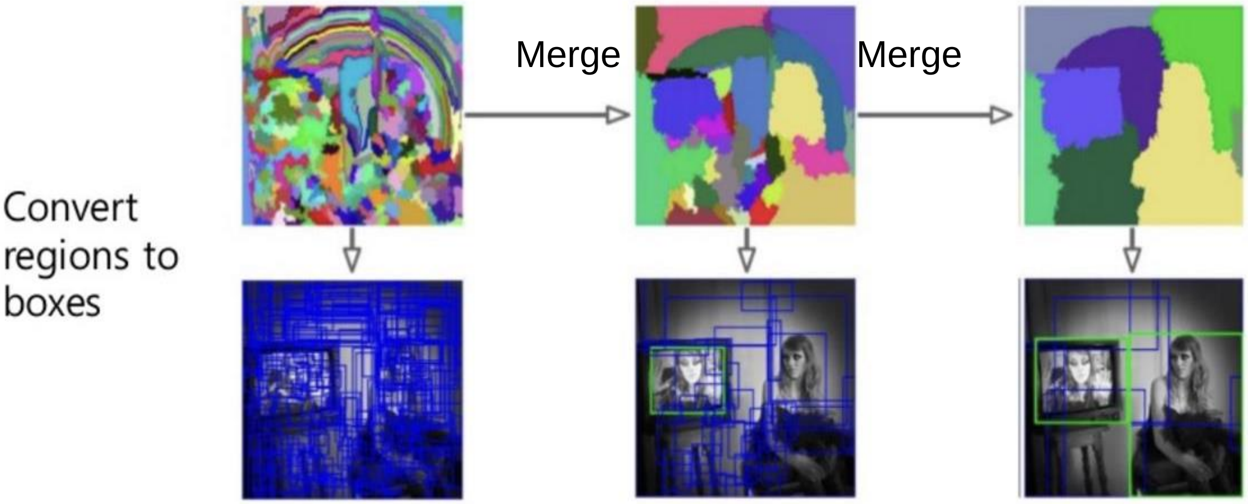
KU-BIG

R-CNN Architecture

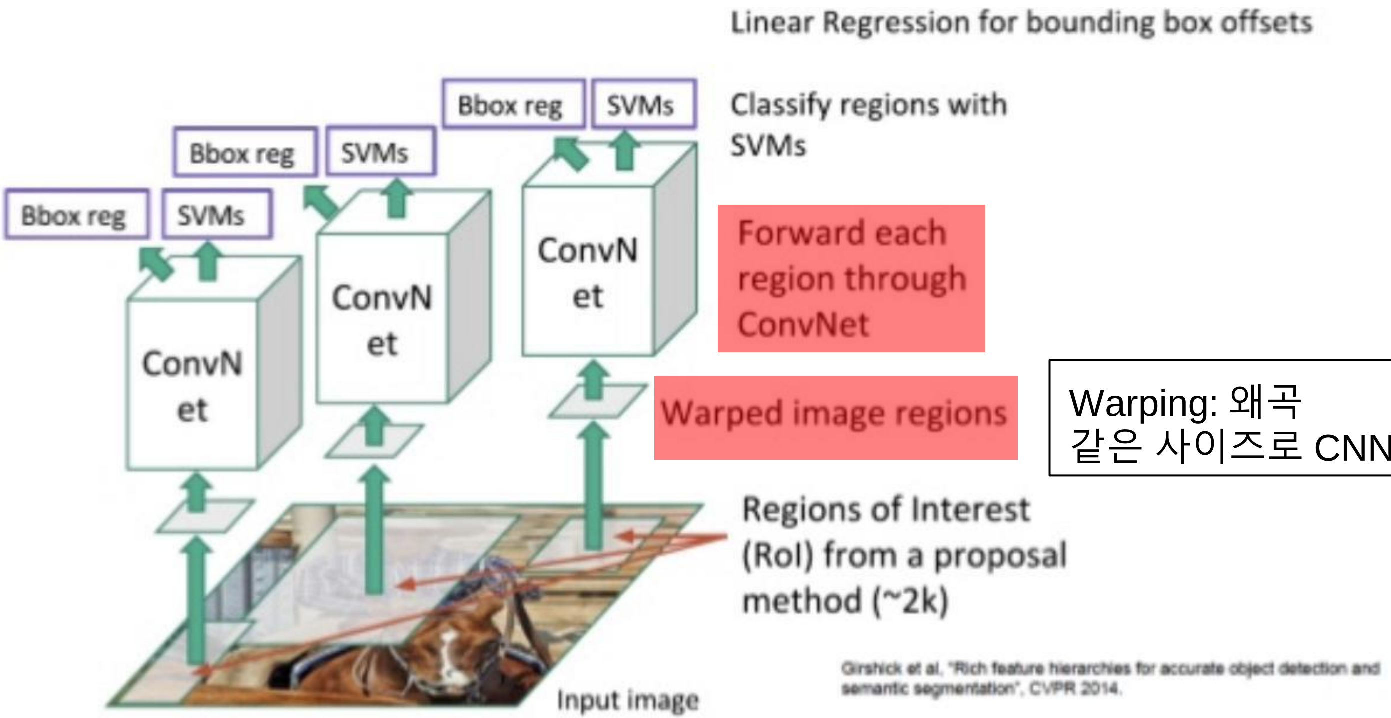


Selective search 이용하여
Bounding Box 선택

Selective search

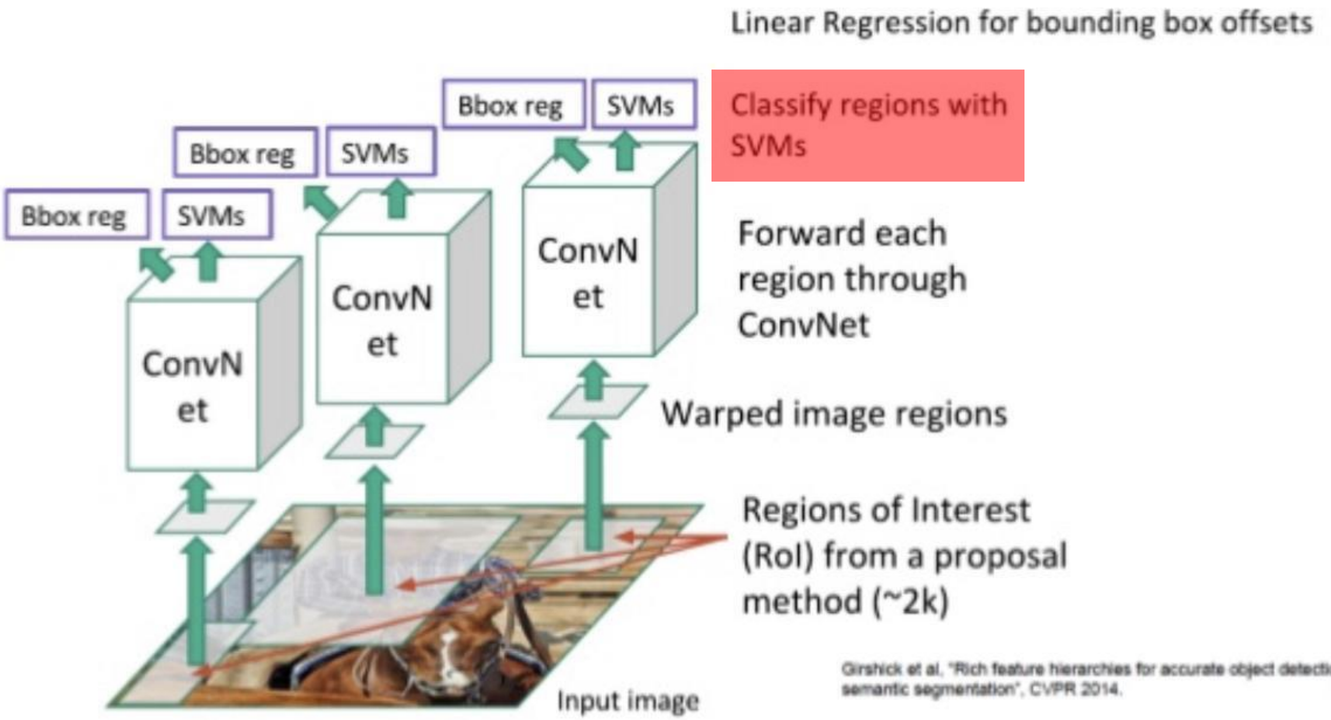


R-CNN Architecture



Warping: 왜곡
같은 사이즈로 CNN에 넣어주기 위해서

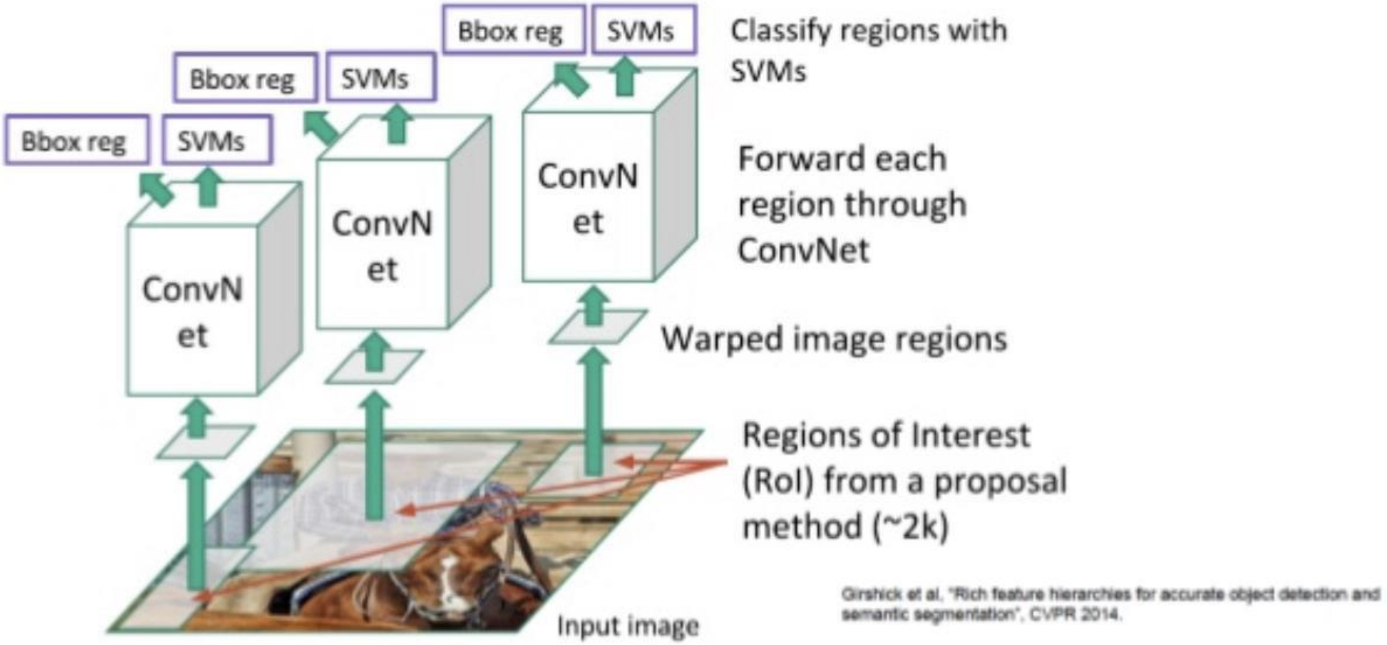
R-CNN Architecture



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

R-CNN Architecture

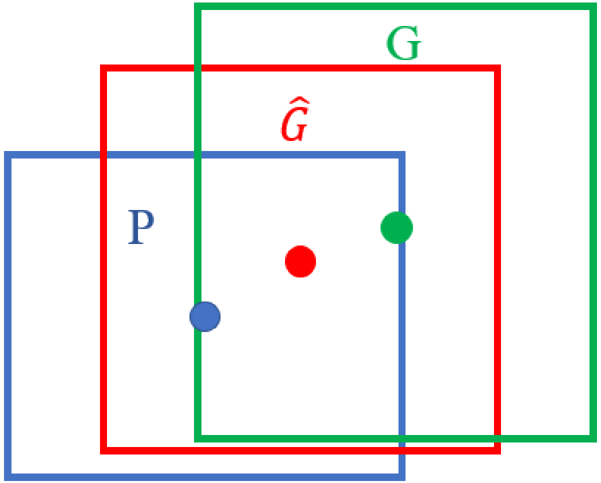
Linear Regression for bounding box offsets



Bounding box regression

$P^i = (P^i_x, P^i_y, P^i_w, P^i_h)$ 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



Bounding box regression

$P^i = (P^i_x, P^i_y, P^i_w, P^i_h)$ specifies the pixel coordinates of the center of proposal P^i's bounding box together with P^i's width and height in pixels

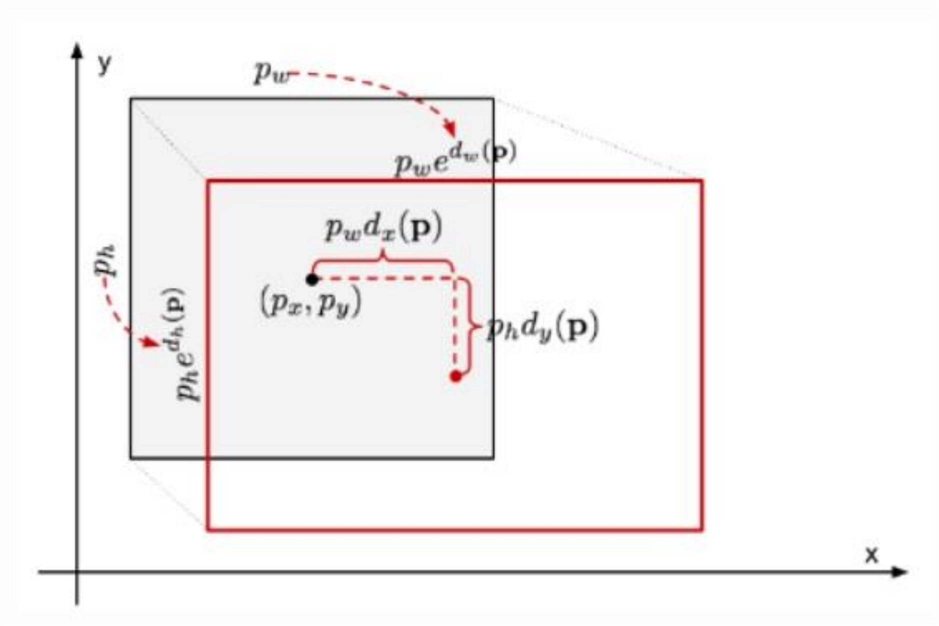
$G = (G_x, G_y, G_w, G_h)$ means the ground-truth bounding box

$$G_x = P_w t_x + P_x$$
$$G_y = P_h t_y + P_y$$
$$G_w = P_w \exp(t_w)$$
$$G_h = P_h \exp(t_h)$$

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

$$d_*(P) = \mathbf{w}_*^T \phi_5(P)$$

Bounding box regression



$$d_*(P) = \boldsymbol{w}_*^T \phi_5(P)$$

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

Bounding box regression

Loss function:
$$\boldsymbol{w}_* = \underset{\hat{\boldsymbol{w}}_*}{argmin} \sum_i^N (t_*^i - d_*(P))^2 + \lambda ||\hat{\boldsymbol{w}}_*||^2$$

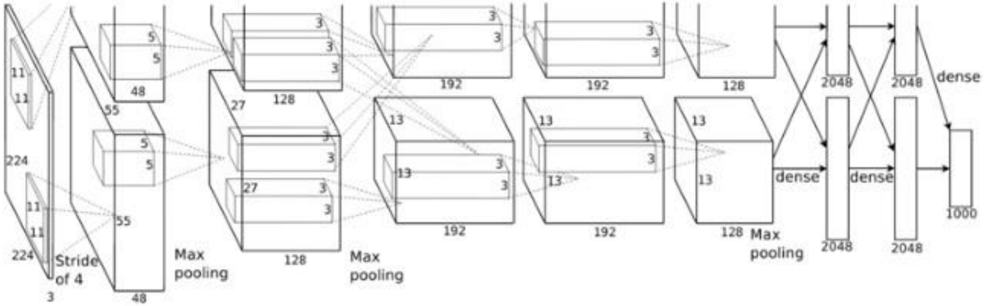
$$= \underset{\hat{\boldsymbol{w}}_*}{argmin} \sum_i^N (t_*^i - \hat{\boldsymbol{w}}_*^T \phi_5(P^i))^2 + \lambda ||\hat{\boldsymbol{w}}_*||^2$$

$$\Leftrightarrow \underset{\hat{\boldsymbol{w}}_*}{argmin} \sum_i^N (t_*^i - \hat{\boldsymbol{w}}_*^T \phi_5(P^i))^2$$

$$\text{subject to} \quad ||\hat{\boldsymbol{w}}_*||^2 \leq s$$

Training R-CNN

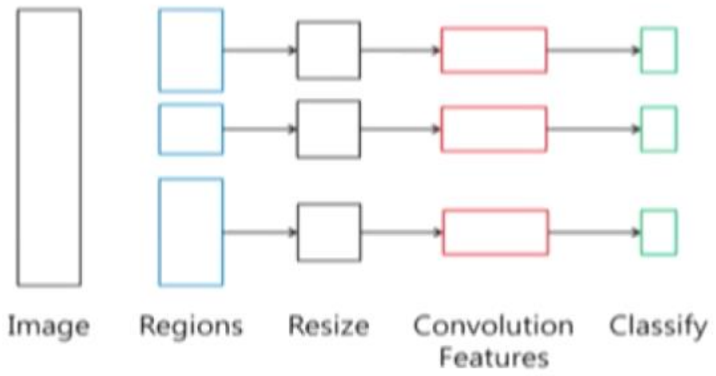
- Pre-train a ConvNet(AlexNet) for ImageNet classification dataset
- Fine-tune for object detection(softmax + log loss)
- Cache feature vectors to disk
- Train post hoc linear SVMs(hinge loss)
- Train post hoc linear bounding-box regressors(squared loss)



R-CNN Drawbacks

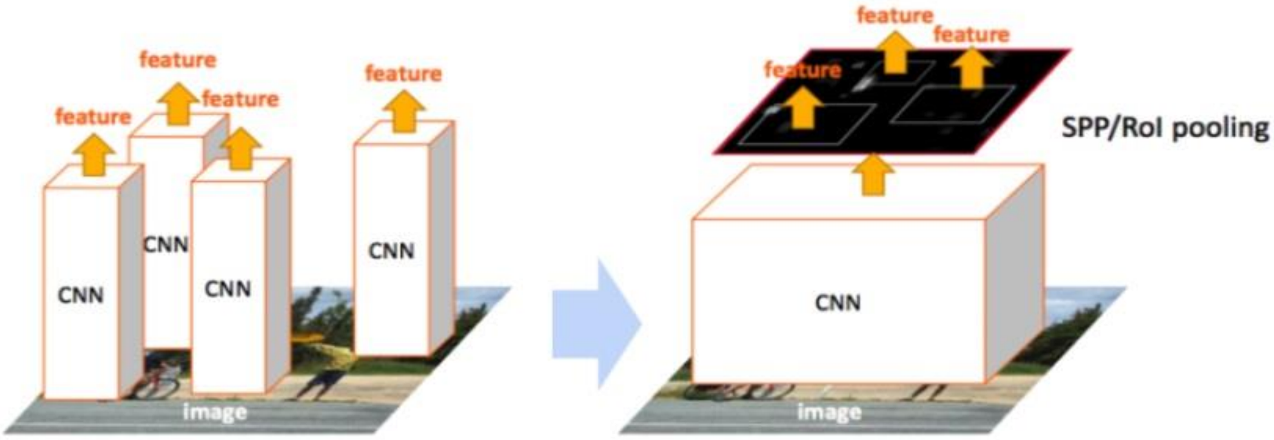
1. Time (Speed)
   - 3 multi-stage process
   - 2,000 region proposals

2. Performance
   - fine tuning (backpropagation)
   - warping issue



4. Fast R-CNN

Fast R-CNN overview



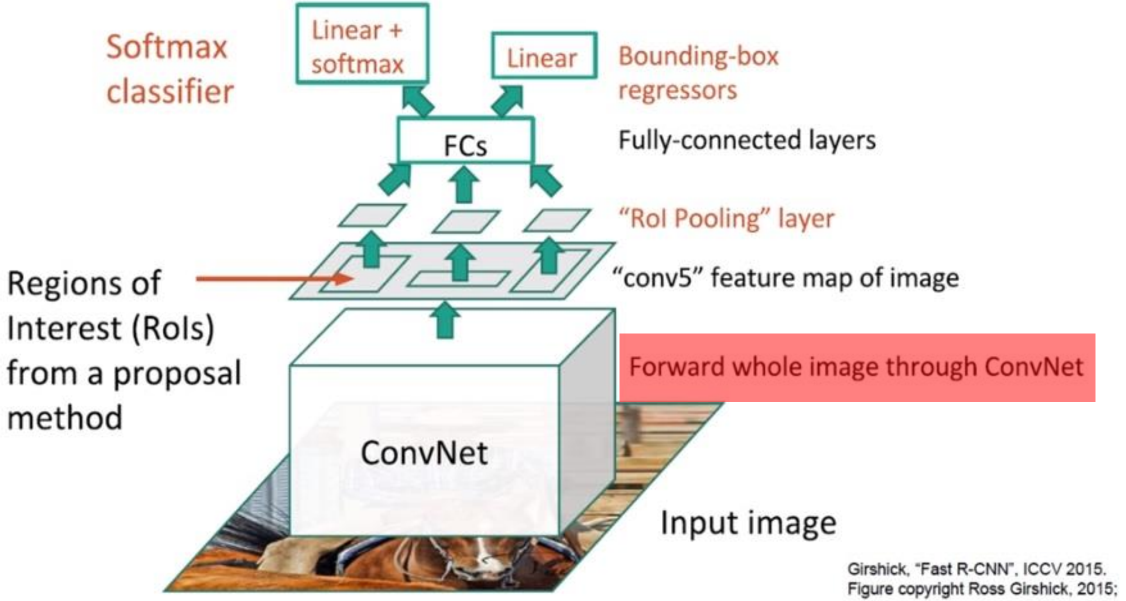
R-CNN

- Extract image regions
- 1 CNN per region (2000 CNNs)
- Classify region-based features
- Complexity: ~$224 \times 224 \times 2000$

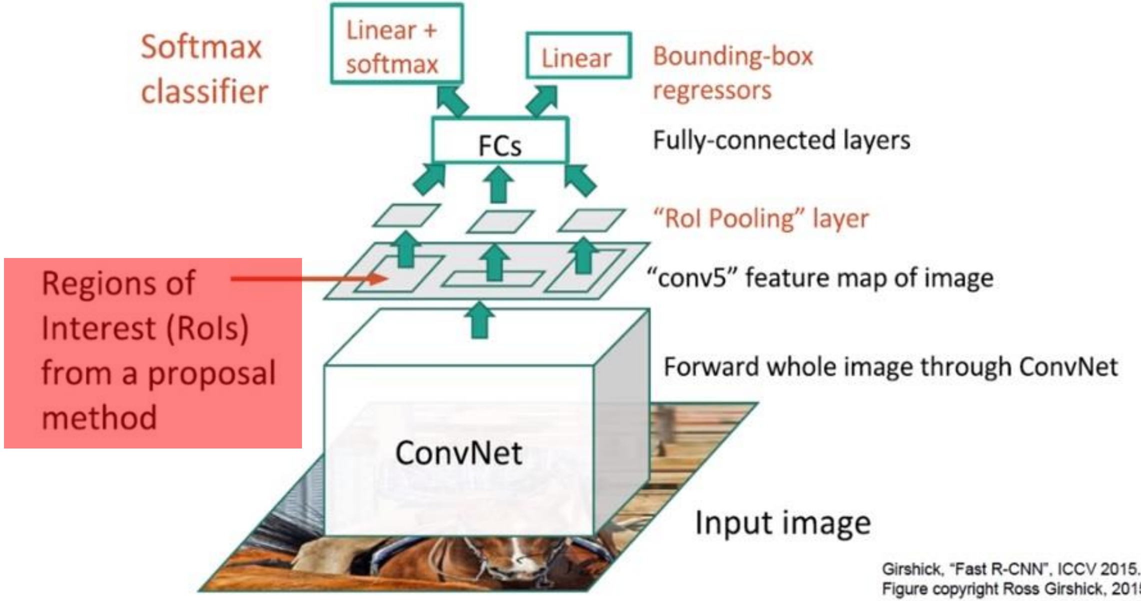
SPP-net & Fast R-CNN (the same forward pipeline)

- 1 CNN on the entire image
- Extract features from feature map regions
- Classify region-based features
- Complexity: ~$600 \times 1000 \times \mathbf{1}$
- ~160x **faster** than R-CNN

Fast R-CNN Architecture

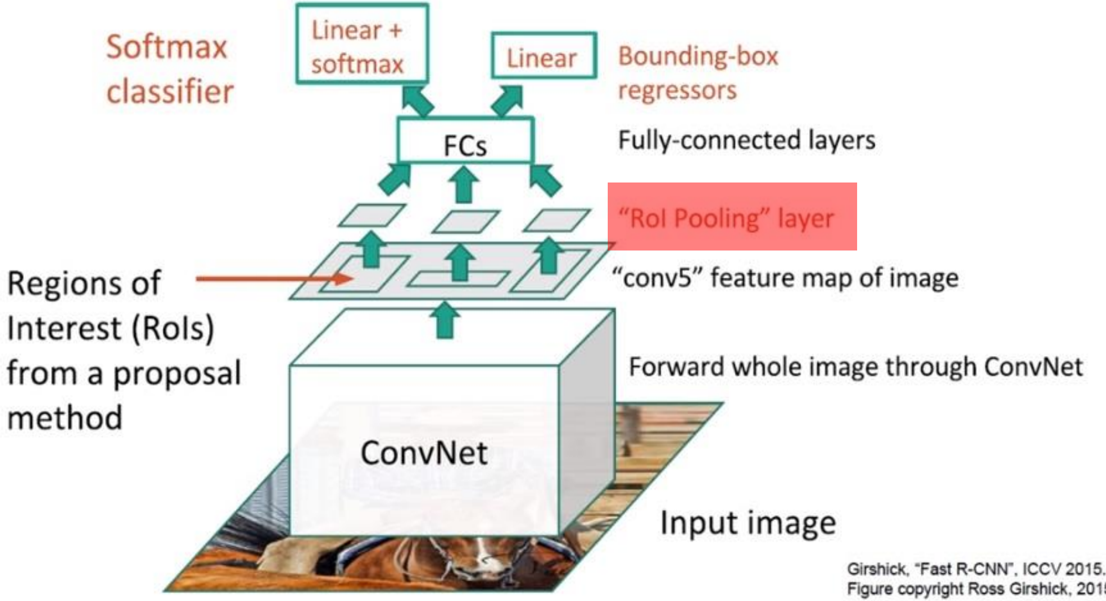


Fast R-CNN Architecture



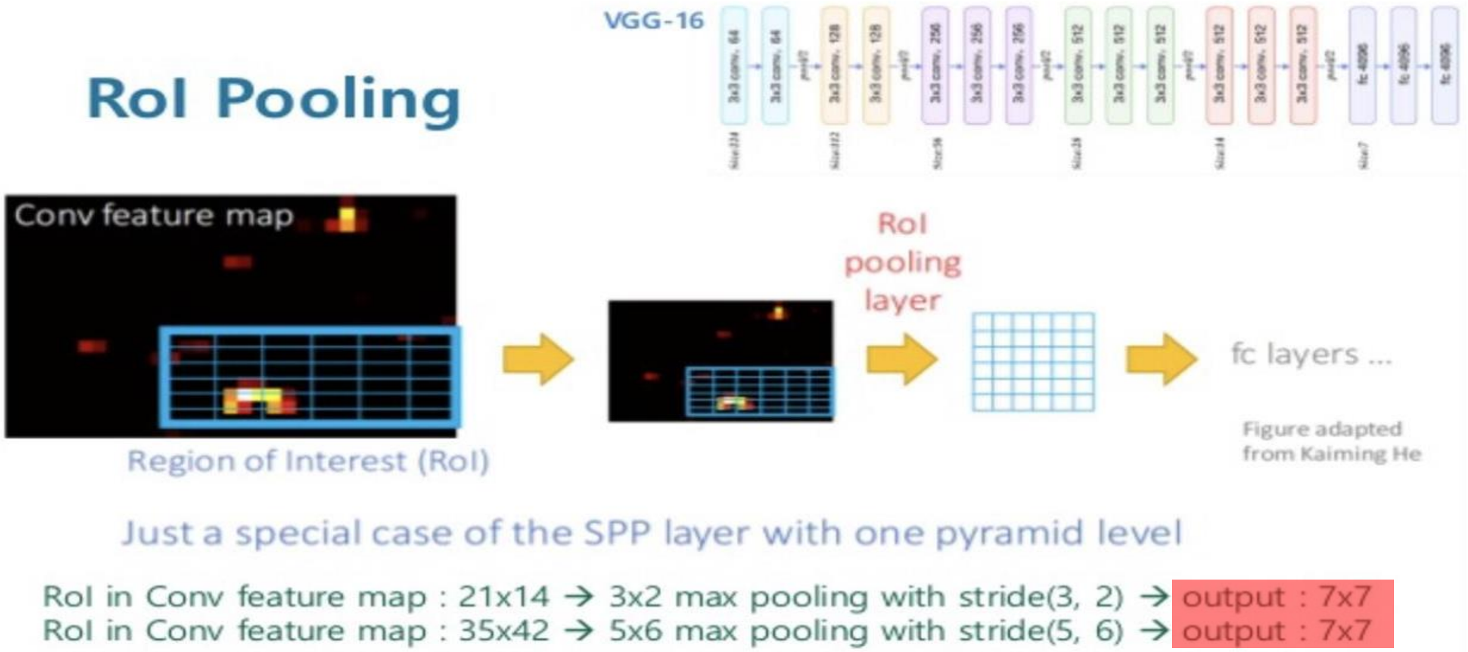
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015;

Fast R-CNN Architecture

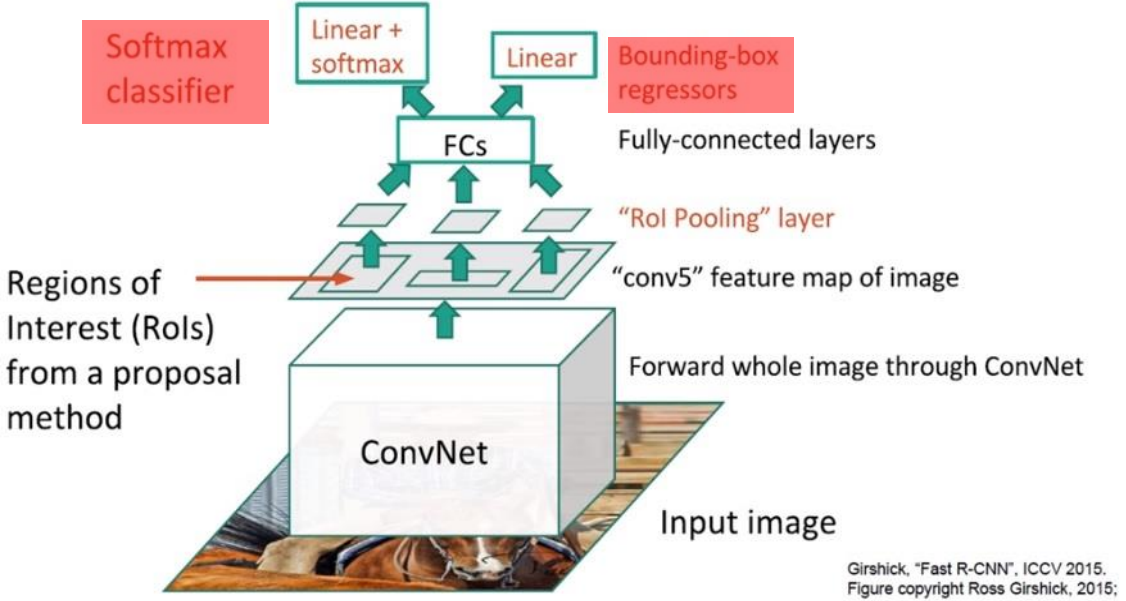


Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015;

RoI Pooling

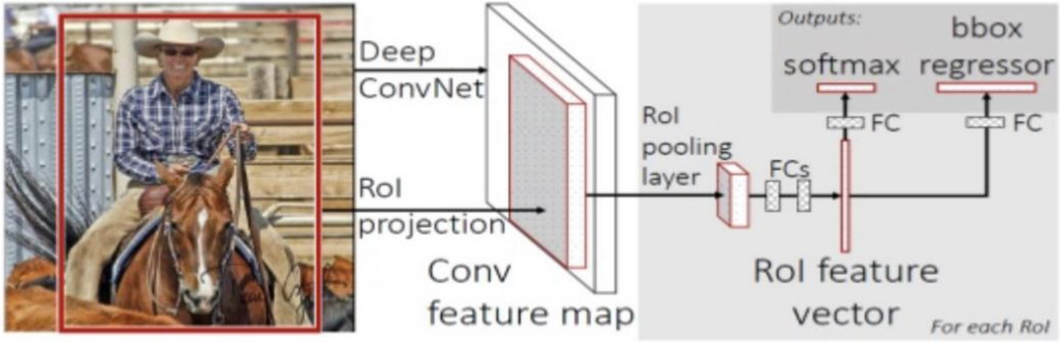

VGG-16
RoI Pooling
Conv feature map
RoI
pooling
layer
fc layers ...
Region of Interest (RoI)
Figure adapted
from Kaiming He
Just a special case of the SPP layer with one pyramid level
RoI in Conv feature map : 21x14 → 3x2 max pooling with stride(3, 2) → output : 7x7
RoI in Conv feature map : 35x42 → 5x6 max pooling with stride(5, 6) → output : 7x7

Fast R-CNN Architecture

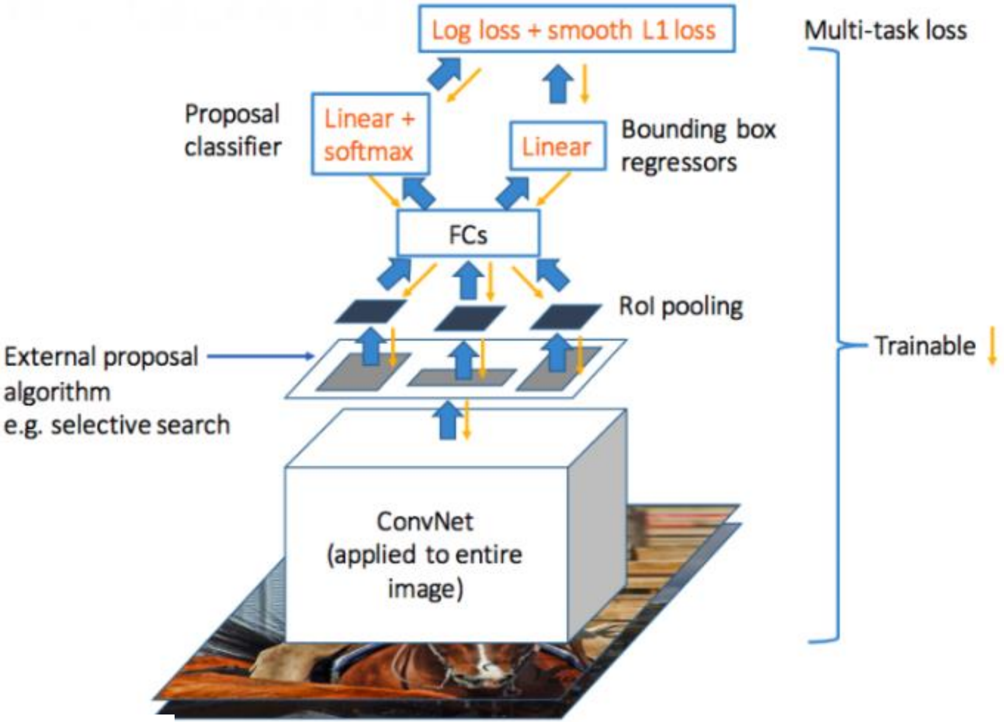


Training Fast R-CNN

- Takes an input and a set of bounding boxes
- Generate convolutional feature maps
- For each bounding box, get a fixed length feature vector from RoI pooling layer
- Outputs have two information
  - K+1 class labels (including background)
  - bounding box locations



Training Fast R-CNN



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

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

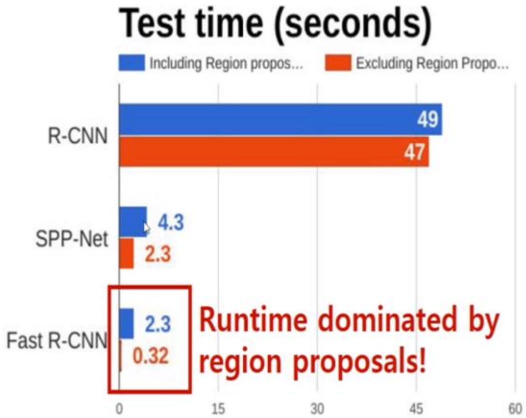
<분류>

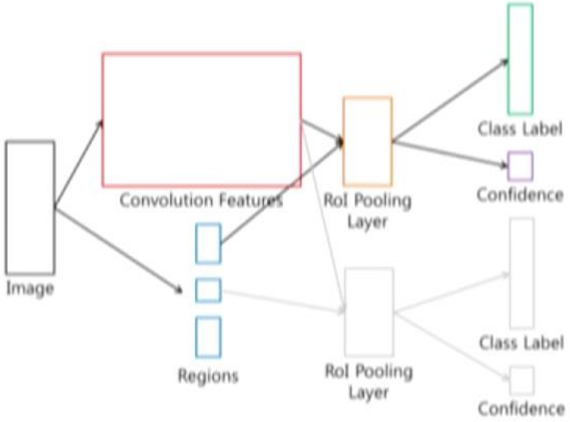
$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i)$$

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

<회귀>

Fast R-CNN Drawback



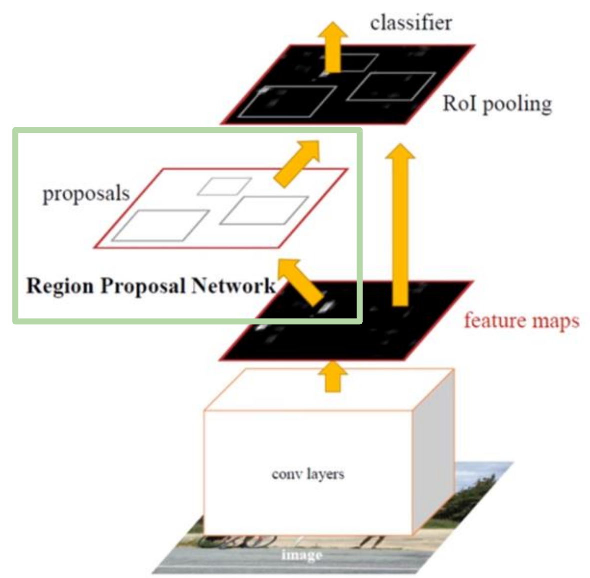


5. Faster R-CNN

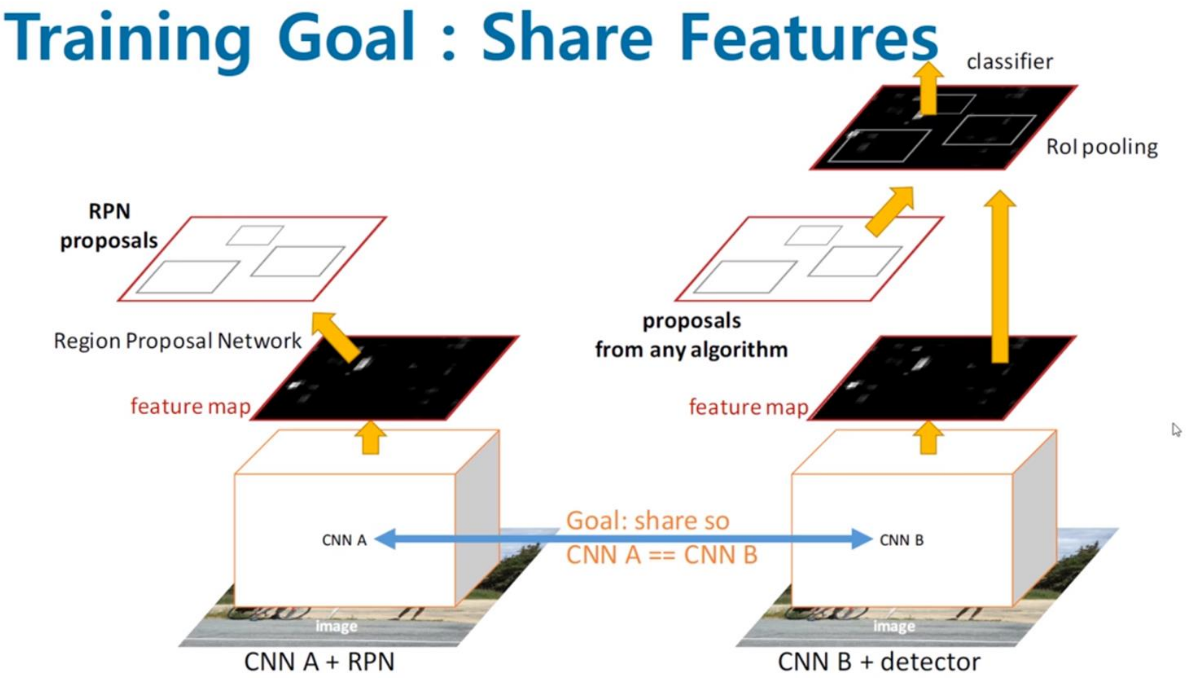
Faster R-CNN overview

Faster R-CNN = RPN + Fast R-CNN

RPN : Region Proposal Network

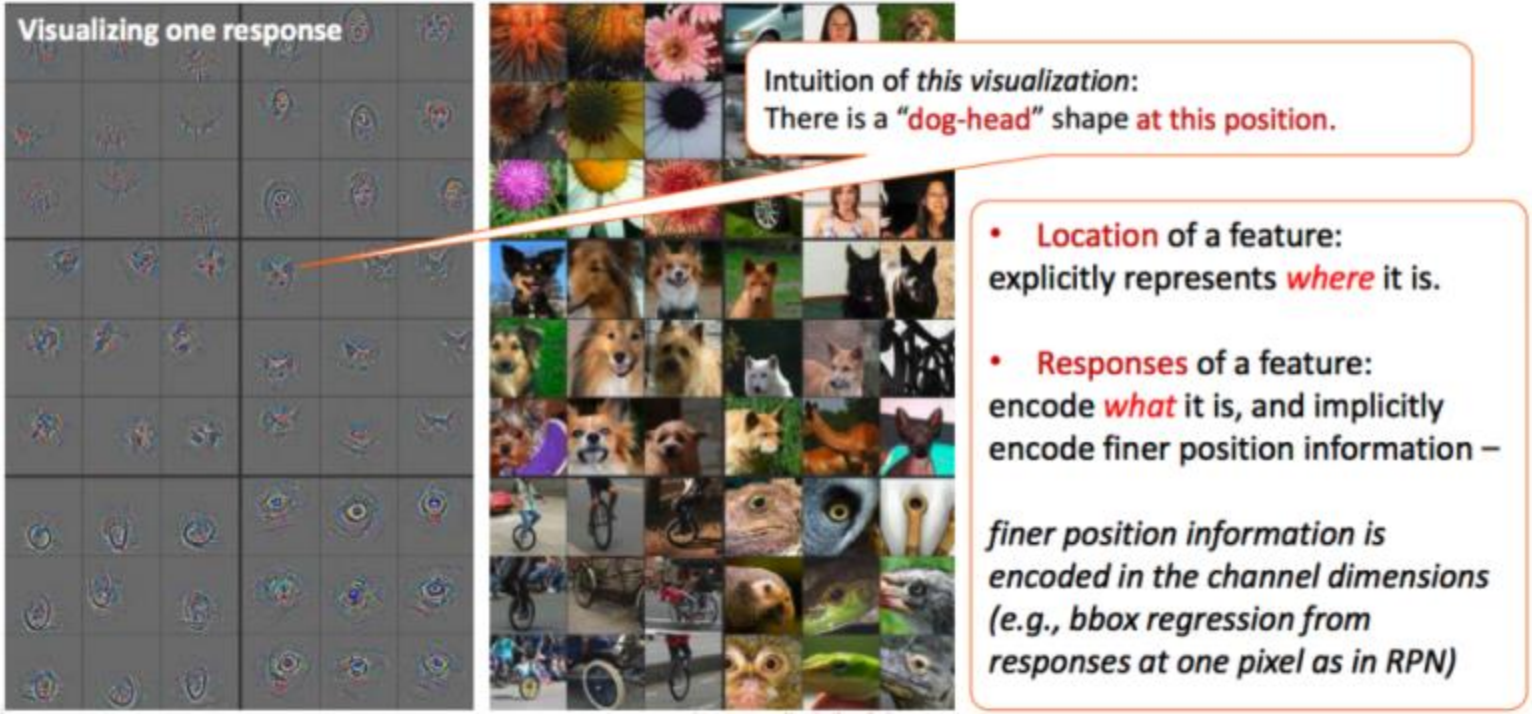


Faster R-CNN overview



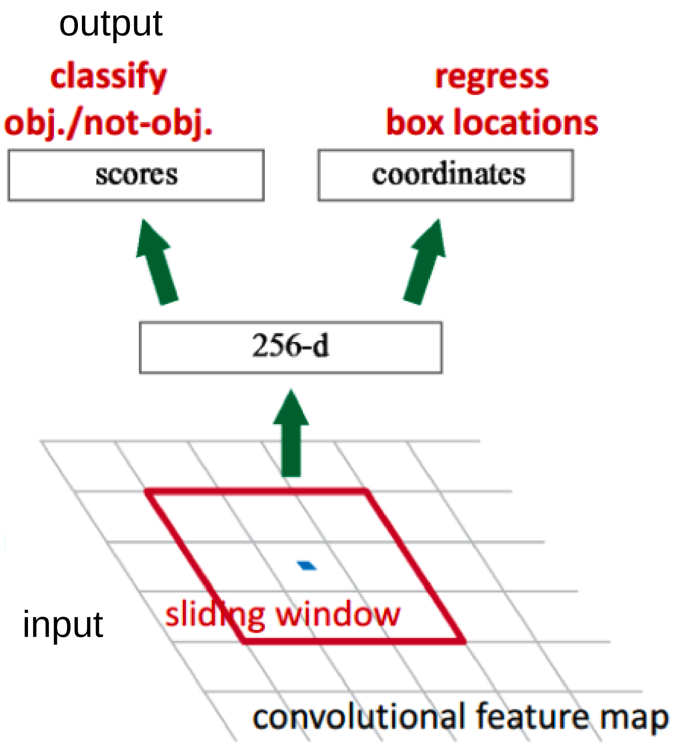
KU-BIG

RPN : Region Proposal Network

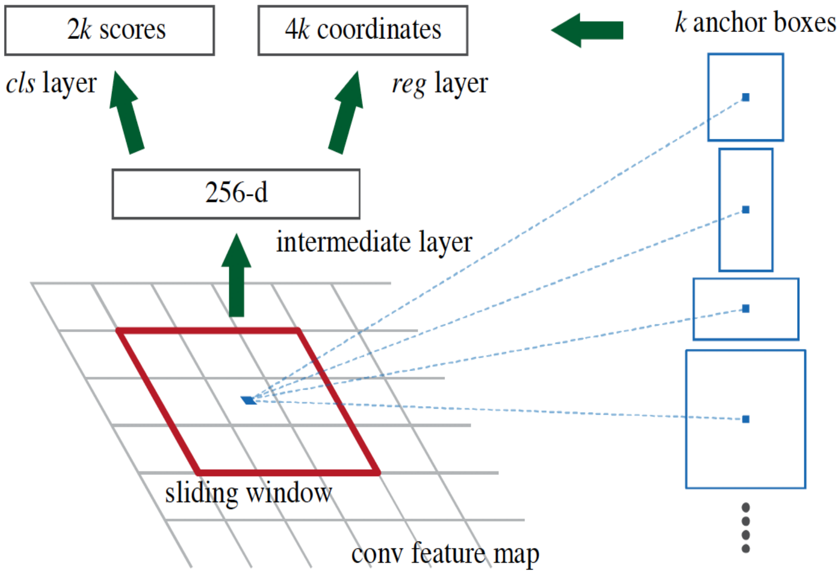


KU-BIG

RPN : Region Proposal Network



RPN : Region Proposal Network



RPN : Region Proposal Network

i = anchor index in minibatch

Log loss

Ground truth objectness label

Smooth L1 loss

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

True box coordinates

Coordinates of the predicted bounding box for anchor i

pi* = 물체 있으면 1, 없으면 0
=> 물체 없으면 regression loss 계산 X

Predicted probability of being an object 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)

Training RPN

- Anchor 중 실제 RoI pooling에 집어 넣을 sample 선택
  - positive / negative sample 추출
  - ground truth와 foreground 비율 가장 많이 겹치거나 / 70%이상 겹치는 anchor => POSITIVE
  - ground truth와 30% 이하로 겹치는 anchor => NEGATIVE
  - 나머지(IoU 30%-70%) anchors는 버림 (=> 모호함 제거)
- positive : negative = 1 : 1 비율을 맞춰 sampling 한 뒤 Fast R-CNN 모델에 집어넣음

Training Faster R-CNN

```
# 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
```

Faster R-CNN Performance

	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**

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**

끝!
감사합니다.