(FAST(er)) R-CNN

컴퓨터소프트웨어학부

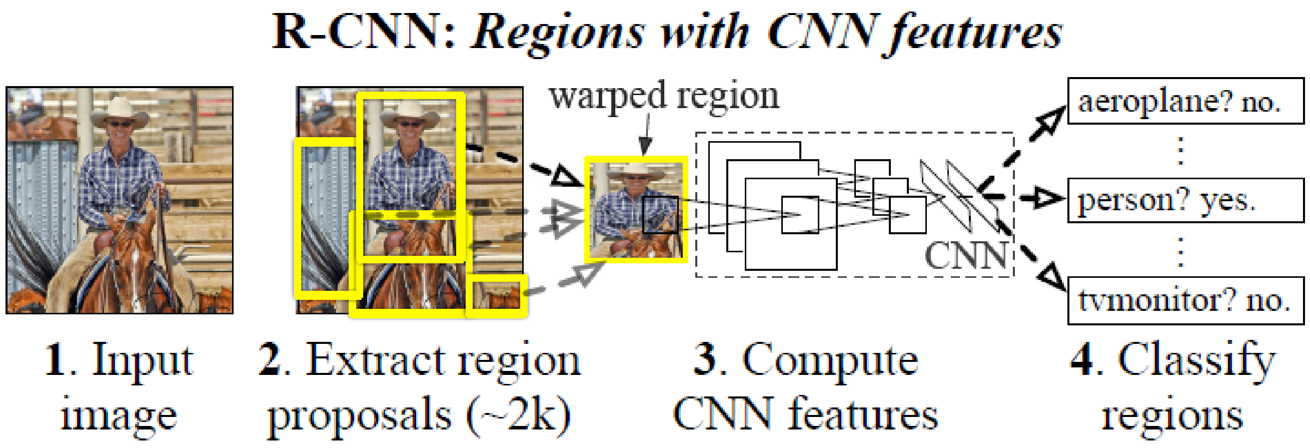
박현준

Contents

Ⅰ. R-CNN

INTRODUCTION

"Features matter"



Unlike Image classification, Object detection needs localization

[Problem]

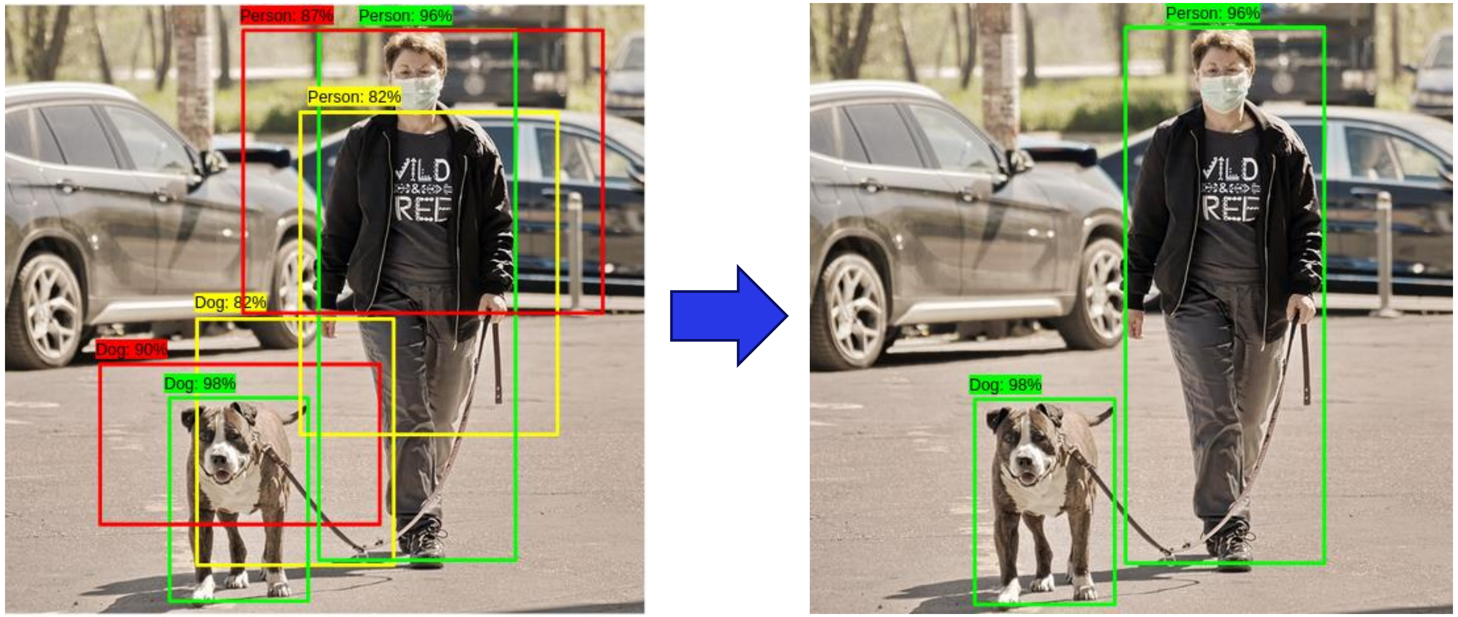
1. CNN (Sliding window) : Precise localization is difficult with large receptive field, stride
2. Data Scarcity : Labeled data for training large CNNs

[Solution]

1. R-CNN : 2000-category independent region proposals by Selective Search
2. Supervised pre-training on a large auxiliary dataset (ILSVRC)
   followed by domain specific fine-tuning on a smaller dataset (PASCAL)

INTRODUCTION

Non-Max Suppression (NMS)

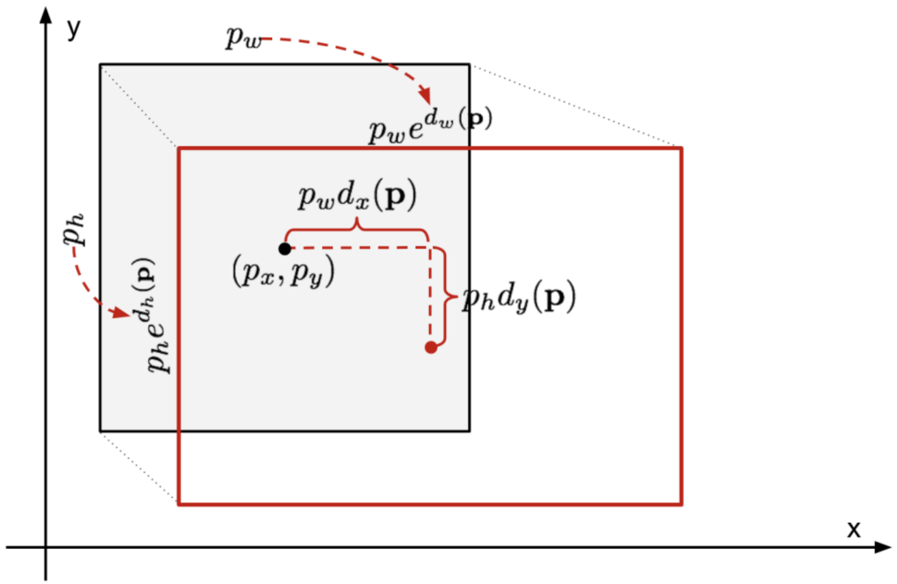


Algorithm 1 Non-Max Suppression

procedure NMS(B,c)
    $B_{nms} \leftarrow \emptyset$
    for $b_i \in B$ **do**
        $discard \leftarrow$ False
        for $b_j \in B$ **do**
            if same$(b_i, b_j) > \boldsymbol{\lambda_{nms}}$ **then**
                if score$(c, b_j) >$ score(c, b_i) **then**
                    $discard \leftarrow$ True
        if not $discard$ **then**
            $B_{nms} \leftarrow B_{nms} \cup b_i$
    return B_{nms}

INTRODUCTION

Bounding-box Regression




```python
def get_iou(bb1, bb2):
  # assuring for proper dimension.
    assert bb1['x1'] < bb1['x2']
    assert bb1['y1'] < bb1['y2']
    assert bb2['x1'] < bb2['x2']
    assert bb2['y1'] < bb2['y2']
  # calculating dimension of common area between these two boxes.
    x_left = max(bb1['x1'], bb2['x1'])
    y_top = max(bb1['y1'], bb2['y1'])
    x_right = min(bb1['x2'], bb2['x2'])
    y_bottom = min(bb1['y2'], bb2['y2'])
  # if there is no overlap output 0 as intersection area is zero.
    if x_right < x_left or y_bottom < y_top:
        return 0.0
  # calculating intersection area.
    intersection_area = (x_right - x_left) * (y_bottom - y_top)
  # individual areas of both these bounding boxes.
    bb1_area = (bb1['x2'] - bb1['x1']) * (bb1['y2'] - bb1['y1'])
    bb2_area = (bb2['x2'] - bb2['x1']) * (bb2['y2'] - bb2['y1'])
  # union area = area of bb1_+ area of bb2 - intersection of bb1 and bb2.
    iou = intersection_area / float(bb1_area + bb2_area - intersection_area)
    assert iou >= 0.0
    assert iou <= 1.0
    return iou
```

Making ground truth box and predicted box similar

Proceed only for values with $IoU \geq 0.5$ (positive) // ~~$IoU \leq 0.3$ (negative)~~

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

$$* \; \boldsymbol{IoU} : \frac{\boldsymbol{Area\ of\ Overlap}}{\boldsymbol{Area\ of\ Union}} = \frac{\boldsymbol{Area(B_{gt} \cap B_p)}}{\boldsymbol{Area(B_{gt} \cup B_p)}}$$

Object Detection with R-CNN

Module Design

< Region proposals >

→ **"Selective Search"**

(b)

Bottom-up segmentation, merging regions at multiple scales

1. Create myriad random bounding boxes
2. Merge bounding boxes using **'Hierarchical Grouping Algorithm'**
3. Proceed to Region proposal format suggesting RoI

Algorithm 1: Hierarchical Grouping Algorithm

Input: (colour) image
Output: Set of object location hypotheses L

Obtain initial regions $R = \{r_1, \cdots, r_n\}$ using [13]
Initialise similarity set $S = \emptyset$
foreach *Neighbouring region pair* (r_i, r_j) **do**
  Calculate similarity $s(r_i, r_j)$
  $S = S \cup s(r_i, r_j)$

while $S \neq \emptyset$ **do**
  Get highest similarity $s(r_i, r_j) = \max(S)$
  Merge corresponding regions $r_t = r_i \cup r_j$
  Remove similarities regarding $r_i : S = S \setminus s(r_i, r_*)$
  Remove similarities regarding $r_j : S = S \setminus s(r_*, r_j)$
  Calculate similarity set S_t between r_t and its neighbours
  $S = S \cup S_t$
  $R = R \cup r_t$

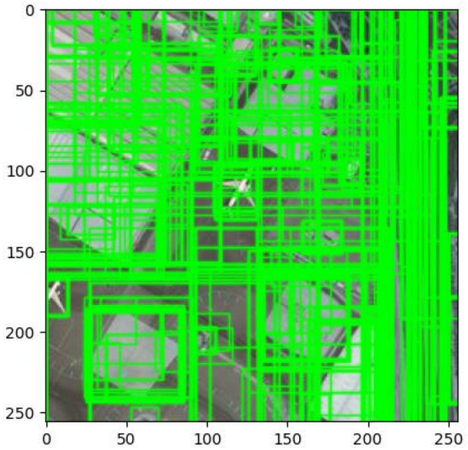
Extract object location boxes L from all regions in R

Object Detection with R-CNN

Module Design

< Region proposals >

→ **"Selective Search"**



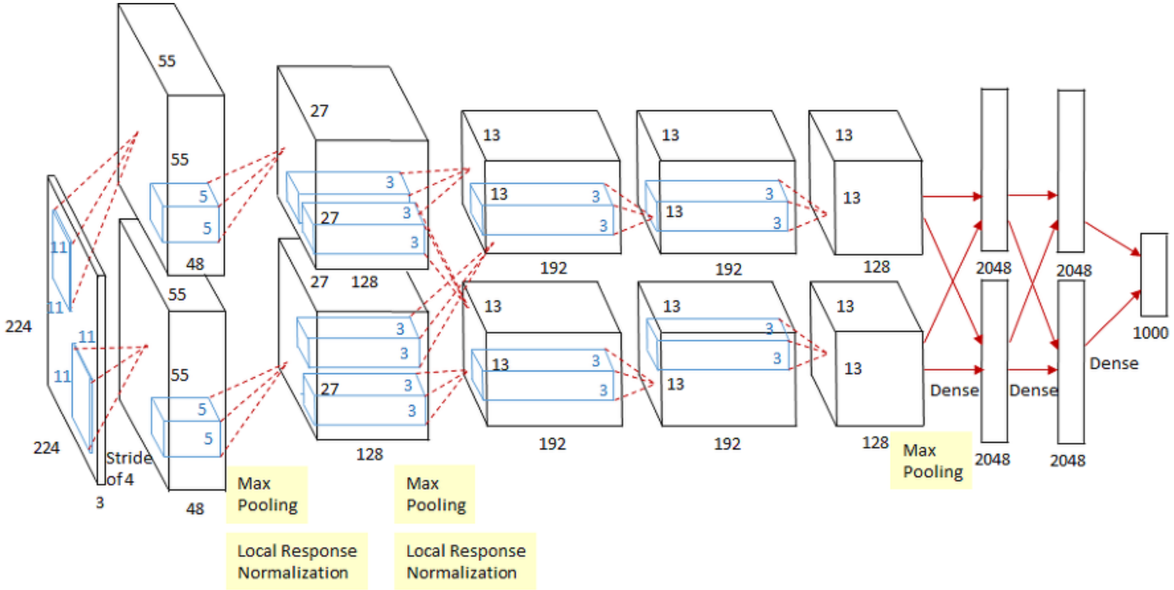

```python
cv2.setUseOptimized(True);
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()

...

for e,i in enumerate(os.listdir(annot)):
    try:
        if i.startswith("airplane"):
            filename = i.split(".")[0]+".jpg"
            print(e,filename)
            image = cv2.imread(os.path.join(path,filename))
            df = pd.read_csv(os.path.join(annot,i))
            gtvalues=[]
            for row in df.iterrows():
                x1 = int(row[1][0].split(" ")[0])
                y1 = int(row[1][0].split(" ")[1])
                x2 = int(row[1][0].split(" ")[2])
                y2 = int(row[1][0].split(" ")[3])
                gtvalues.append({"x1":x1,"x2":x2,"y1":y1,"y2":y2})
            ss.setBaseImage(image)                                  # setting given image as base image
            ss.switchToSelectiveSearchFast()                        # running selective search on bae image
            ssresults = ss.process()                                # processing to get the outputs
            imout = image.copy()
            counter = 0
            falsecounter = 0
            flag = 0
            fflag = 0
            bflag = 0
            for e,result in enumerate(ssresults):
                if e < 2000 and flag == 0:                          # till 2000 to get top 2000 regions only
                    for gtval in gtvalues:
                        x,y,w,h = result
                        # calculating IoU for each of the proposed regions
                        iou = get_iou(gtval,{"x1":x,"x2":x+w,"y1":y,"y2":y+h})
                        if counter < 30:                            # getting only 30 positive examples
                            if iou > 0.70:                          # IoU or being positive is 0.7
                                timage = imout[x:x+w,y:y+h]
                                resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER_AREA)
                                train_images.append(resized)
                                train_labels.append(1)
                                counter += 1
                        else :
                            fflag =1                                # to insure we have collected all psotive examples
                        if falsecounter <30:                        # 30 negatve examples are allowed only
                            if iou < 0.3:                           # IoU or being negative is 0.3
                                timage = imout[x:x+w,y:y+h]
                                resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER_AREA)
                                train_images.append(resized)
                                train_labels.append(0)
                                falsecounter += 1
                        else :
                            bflag = 1                               #to ensure we have collected all negative examples
                    if fflag == 1 and bflag == 1:
                        print("inside")
                        flag = 1          # to signal the complition of data extaction from a particular image
```

Object Detection with R-CNN

Module Design

< Feature extraction >

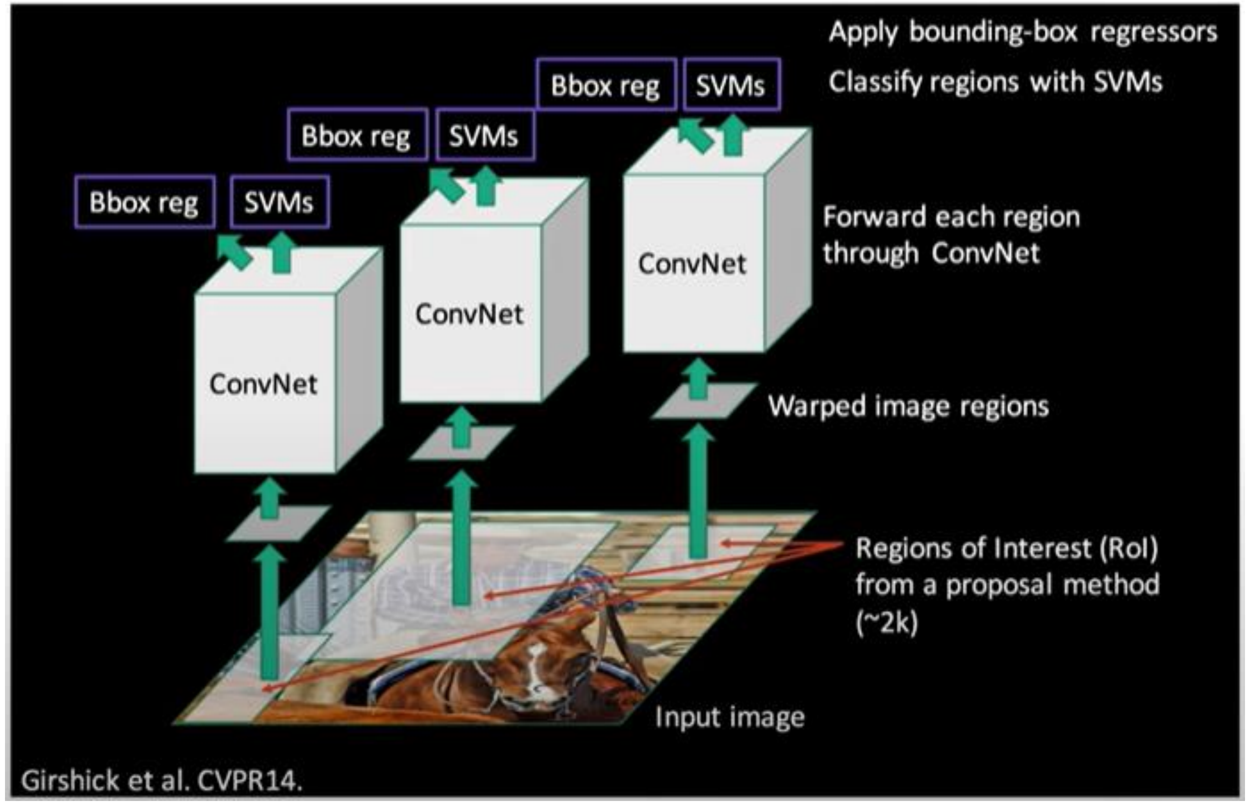
Figure 2: Warped training samples from VOC 2007 train.



- Use **AlexNet**
- Warping images regardless of the size or aspect ratio of the candidate region (227x227)
- **5 convolutional layers + 2 fully-connected layers**
- Output : 4096-dimensional feature vector
- **Transfer learning** → SVM (per class) in final part for image classification

Object Detection with R-CNN

Model Workflow



1. Pre-training CNN network on image classification task
2. Region proposal by using Selective Search
3. Warping the proposed image to a fixed size
   (∵ Region proposal size varies)
4. Fine-tuning CNN on warped proposal regions for
   N+1 classes (N : #class , 1 : background)
5. Fixed-length feature extraction using CNN
6. Class-specific linear SVM classification for each class
   independently based on extracted features
7. Bounding box regression to reduce mislocalization

Result

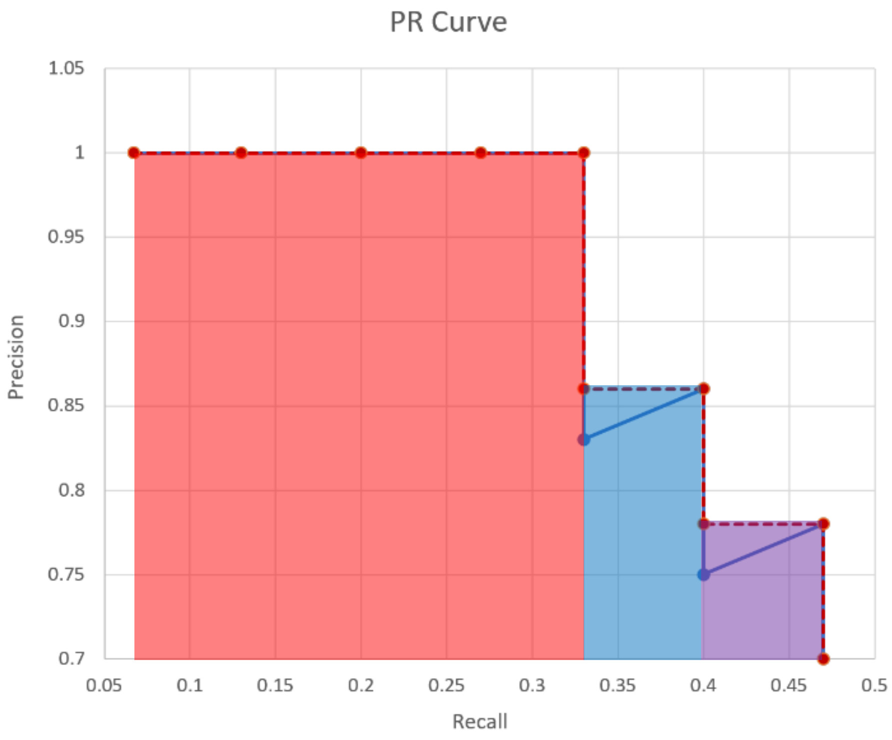
[mAP]

		실제 정답	
		Positive	Negative
실험 결과	Positive	True Positive	False Positive (Type 1 Error)
	Negative	False Negative (Type 2 Error)	True Negative

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

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{All\ Ground\ Truths}$$

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i$$



$$AP = 1 * 0.33 + 0.86 * (0.4 - 0.33) + 0.77 * (0.46 - 0.4) = 0.4364$$

Result

[Dataset]

20 classes

1. PASCAL VOC 2010

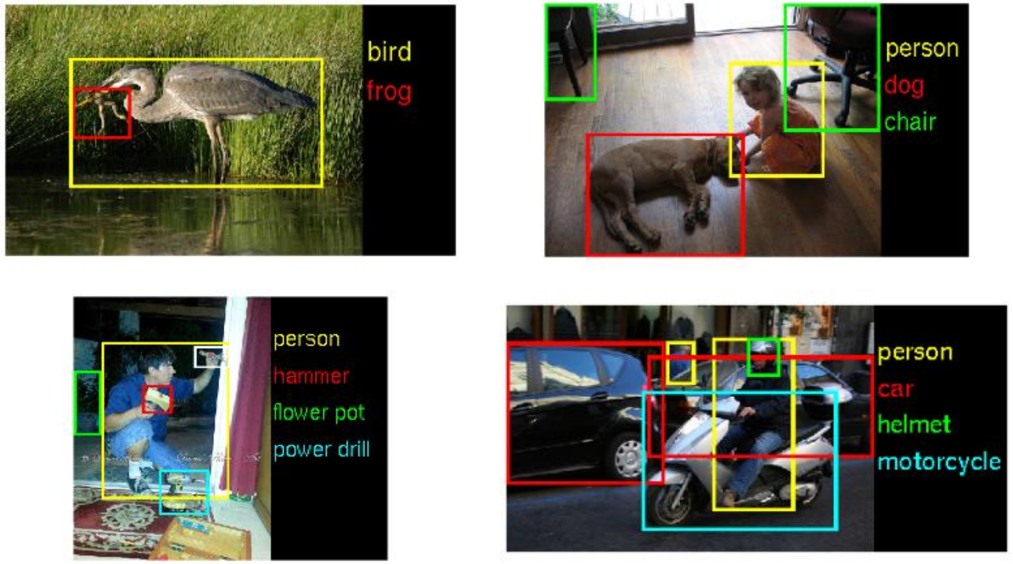
< Class > 20
- Person : person
- Animal : bird, cat, cow, dog, horse, sheep
- Vehicle : aeroplane, bicycle, boat, bus, car, motorbike, train
- Indoor : bottle, chair, dining table, potted plane, sofa, tv/monitor

Train/Val data : 10,103 images containing 23,374 RoI annotated objects and 4,203 segmentations

2. ILSVRC 2013

< Class > 200

Train / Val data : 395,909 + 20,121 images
Test data : 40,152 images

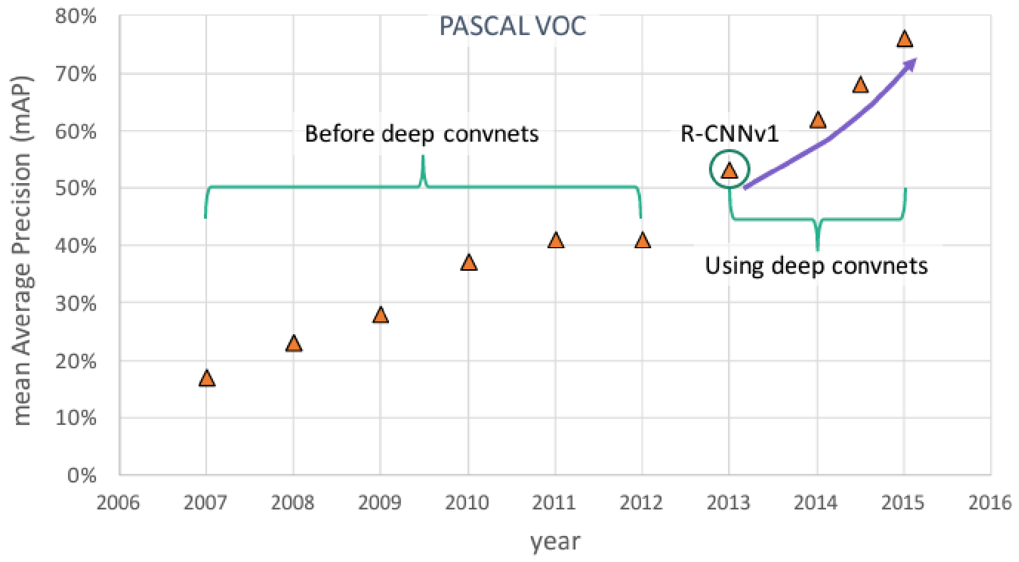


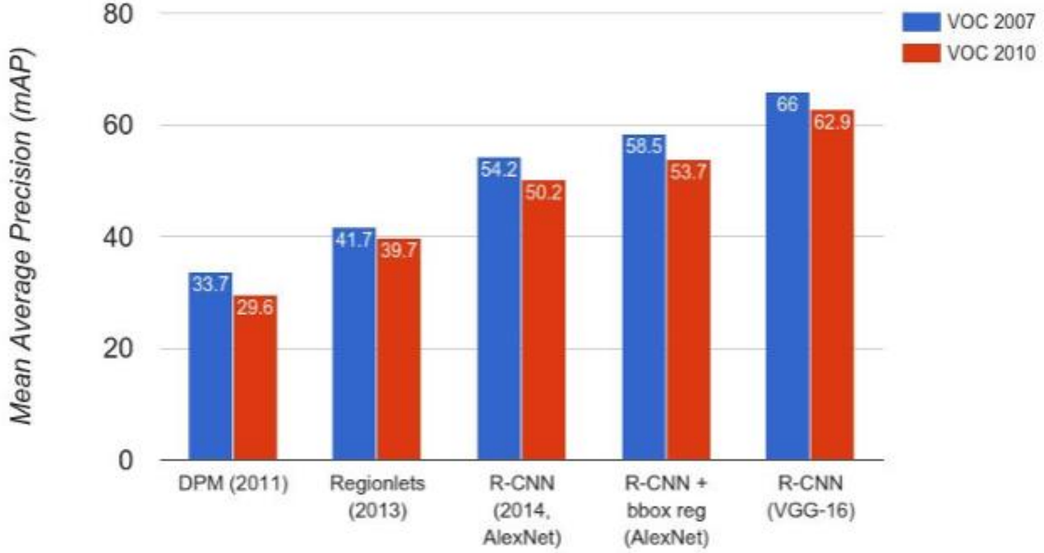
Result

[PASCAL VOC]

VOC 2010 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM v5 [20]†	49.2	53.8	13.1	15.3	35.5	53.4	49.7	27.0	17.2	28.8	14.7	17.8	46.4	51.2	47.7	10.8	34.2	20.7	43.8	38.3	33.4
UVA [39]	56.2	42.4	15.3	12.6	21.8	49.3	36.8	46.1	12.9	32.1	30.0	36.5	43.5	52.9	32.9	15.3	41.1	31.8	47.0	44.8	35.1
Regionlets [41]	65.0	48.9	25.9	24.6	24.5	56.1	54.5	51.2	17.0	28.9	30.2	35.8	40.2	55.7	43.5	14.3	43.9	32.6	54.0	45.9	39.7
SegDPM [18]†	61.4	53.4	25.6	25.2	35.5	51.7	50.6	50.8	19.3	33.8	26.8	40.4	48.3	54.4	47.1	14.8	38.7	35.0	52.8	43.1	40.4
R-CNN	67.1	64.1	46.7	32.0	30.5	56.4	57.2	65.9	27.0	47.3	40.9	66.6	57.8	65.9	53.6	26.7	56.5	38.1	52.8	50.2	50.2
R-CNN BB	**71.8**	**65.8**	**53.0**	**36.8**	**35.9**	**59.7**	**60.0**	**69.9**	**27.9**	**50.6**	**41.4**	**70.0**	**62.0**	**69.0**	**58.1**	**29.5**	**59.4**	**39.3**	**61.2**	**52.4**	**53.7**

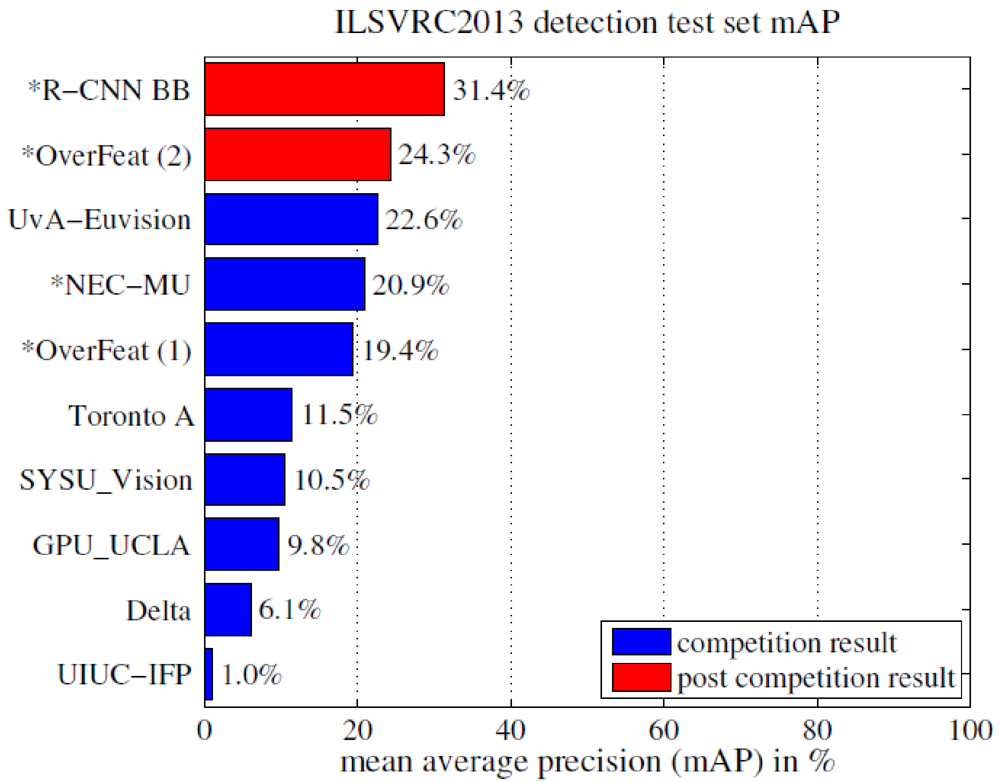
Table 1: Detection average precision (%) on VOC 2010 test. R-CNN is most directly comparable to UVA and Regionlets since all methods use selective search region proposals. Bounding-box regression (BB) is described in Section C. At publication time, SegDPM was the top-performer on the PASCAL VOC leaderboard. †DPM and SegDPM use context rescoring not used by the other methods.

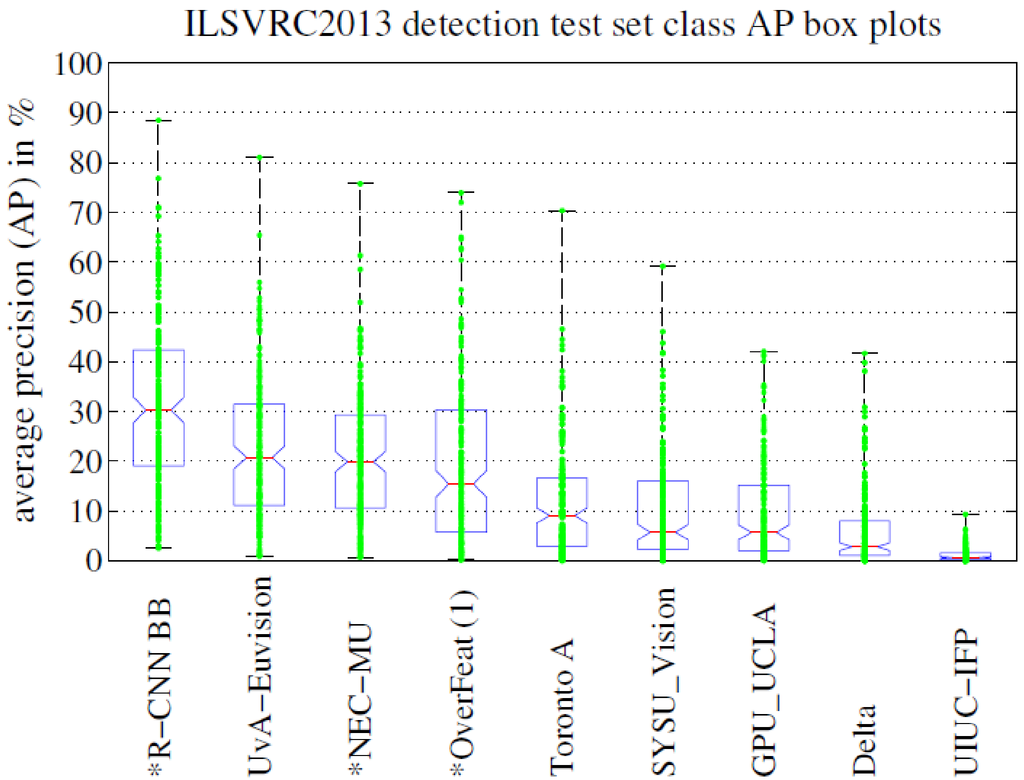




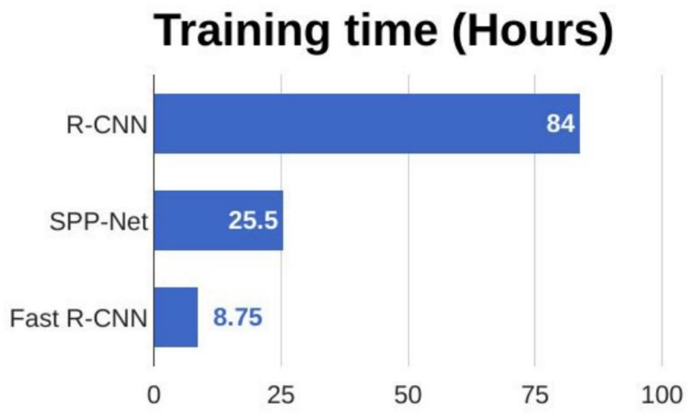
Result

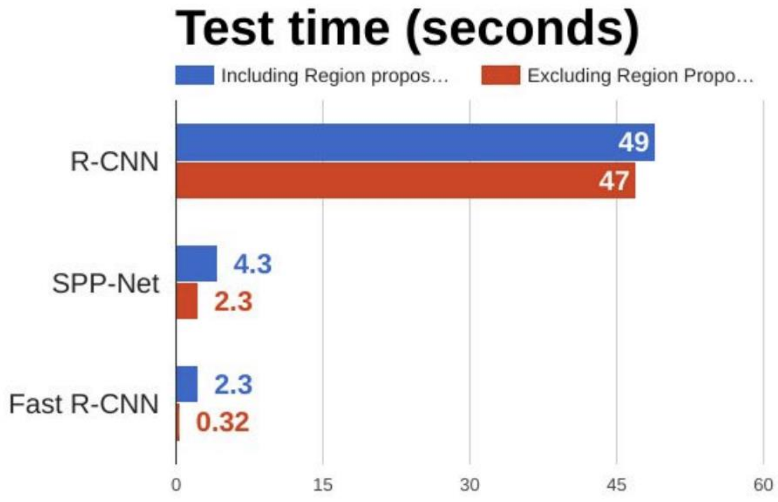
[ILSVRC 2013]





Conclusion





Strength

- Apply high-capacity CNN to bottom-up region proposals in order to **localize** and **segment** objects
- Paradigm for training large CNNs **when labeled training data is scarce**
- R-CNN can scale to thousands of object classes without resorting to approximate techniques, such as hashing

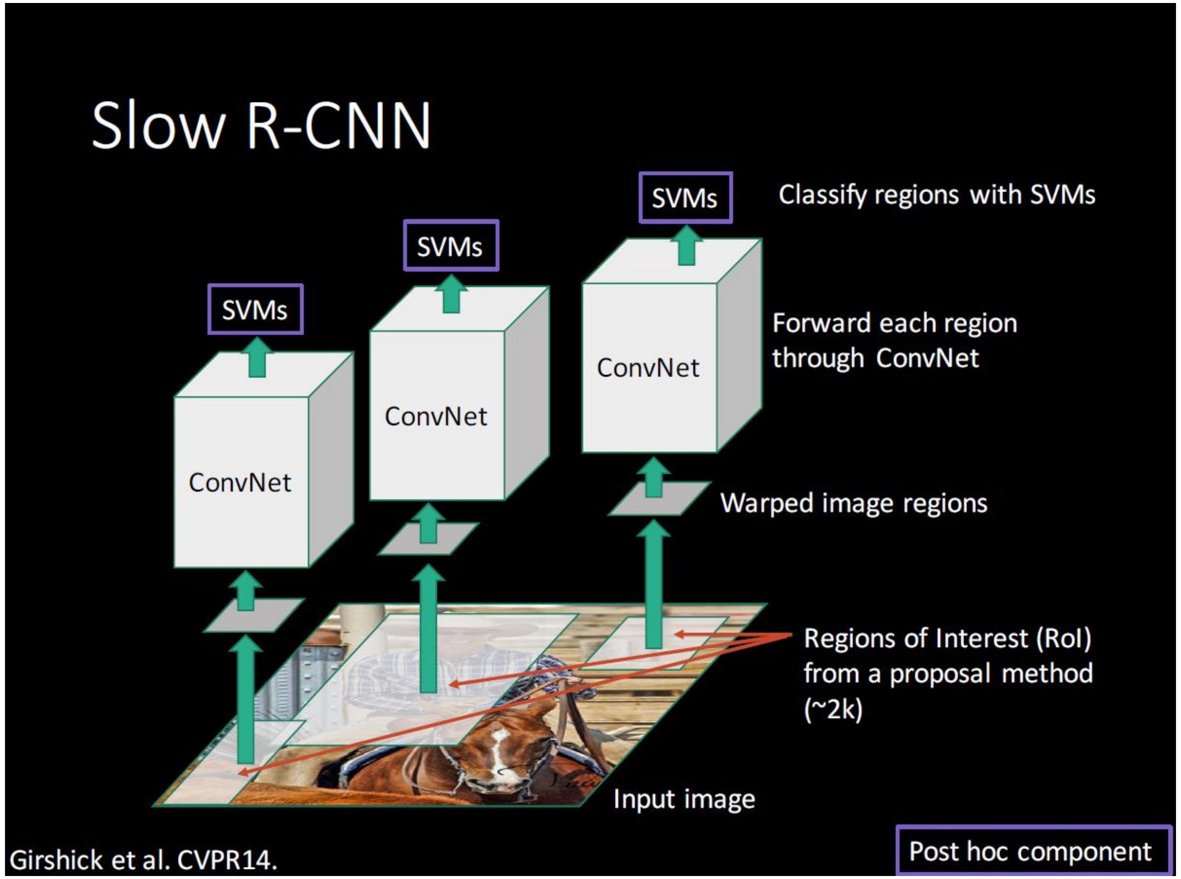
Weakness

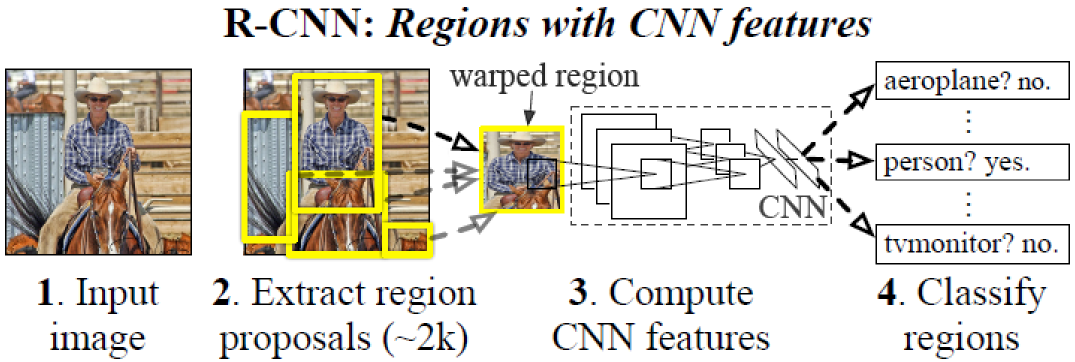
- Running Selective Search sequentially to propose 2000 region proposals for every image
- Degradation of performance because of warping & cropping
- CNN, SVM, and Bounding box regression do not share the operation, so they do not learn end-to-end

Ⅱ. Fast R-CNN

R-CNN

Model Workflow



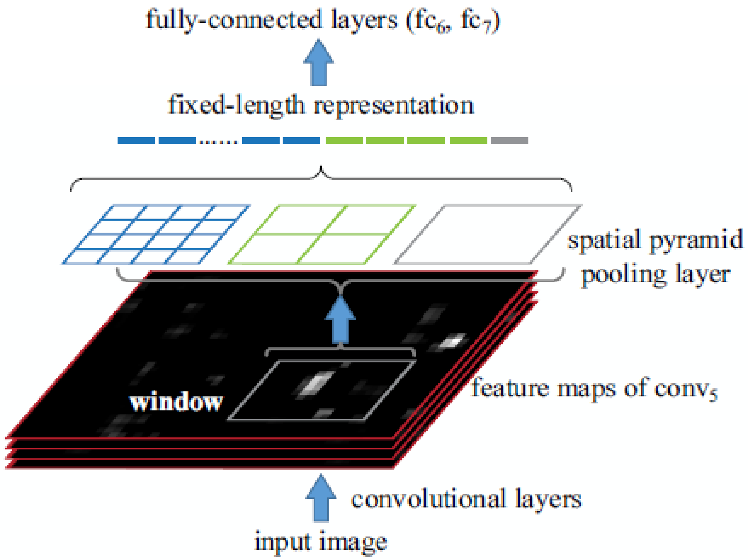


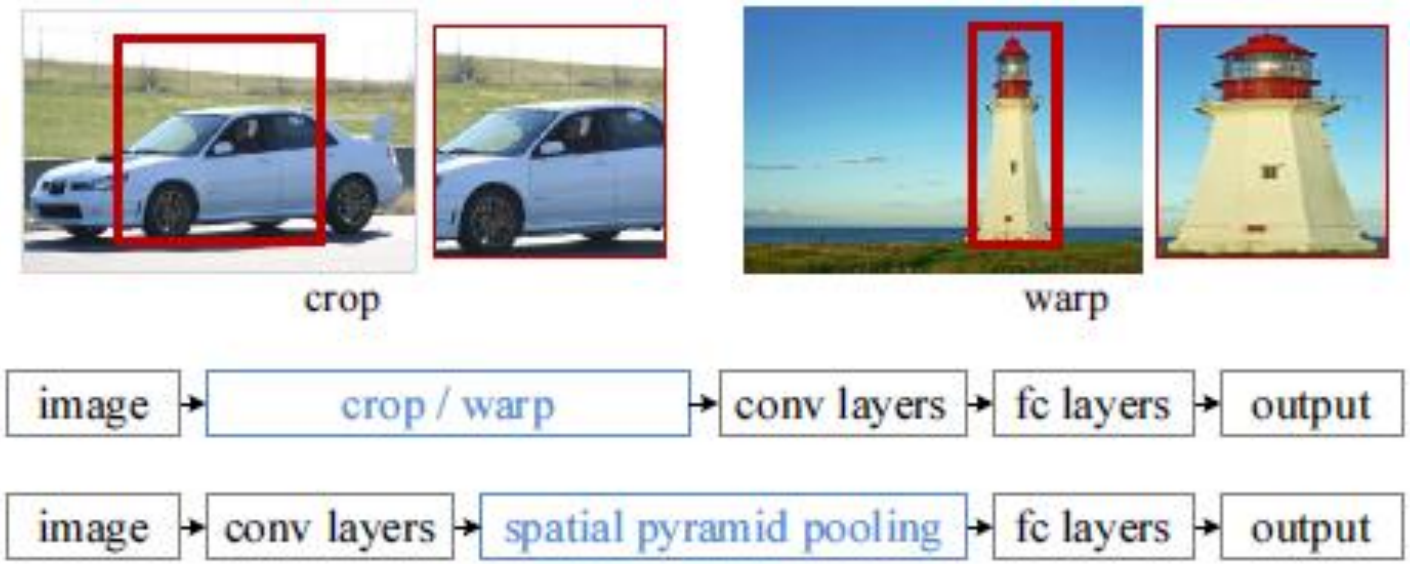
1. Pre-training CNN network on image classification task
2. Region proposal by using Selective Search
3. Warping the proposed image to a fixed size (227*227)
4. Fine-tuning CNN on warped proposal regions for **N+1** classes (N : #class , 1 : background)
5. Feature extraction using CNN
6. SVM classification for each class independently based on extracted features
7. Bounding box regression to reduce localization errors

→ **SLOW**

SPPnet (Spatial Pyramid Pooling net)

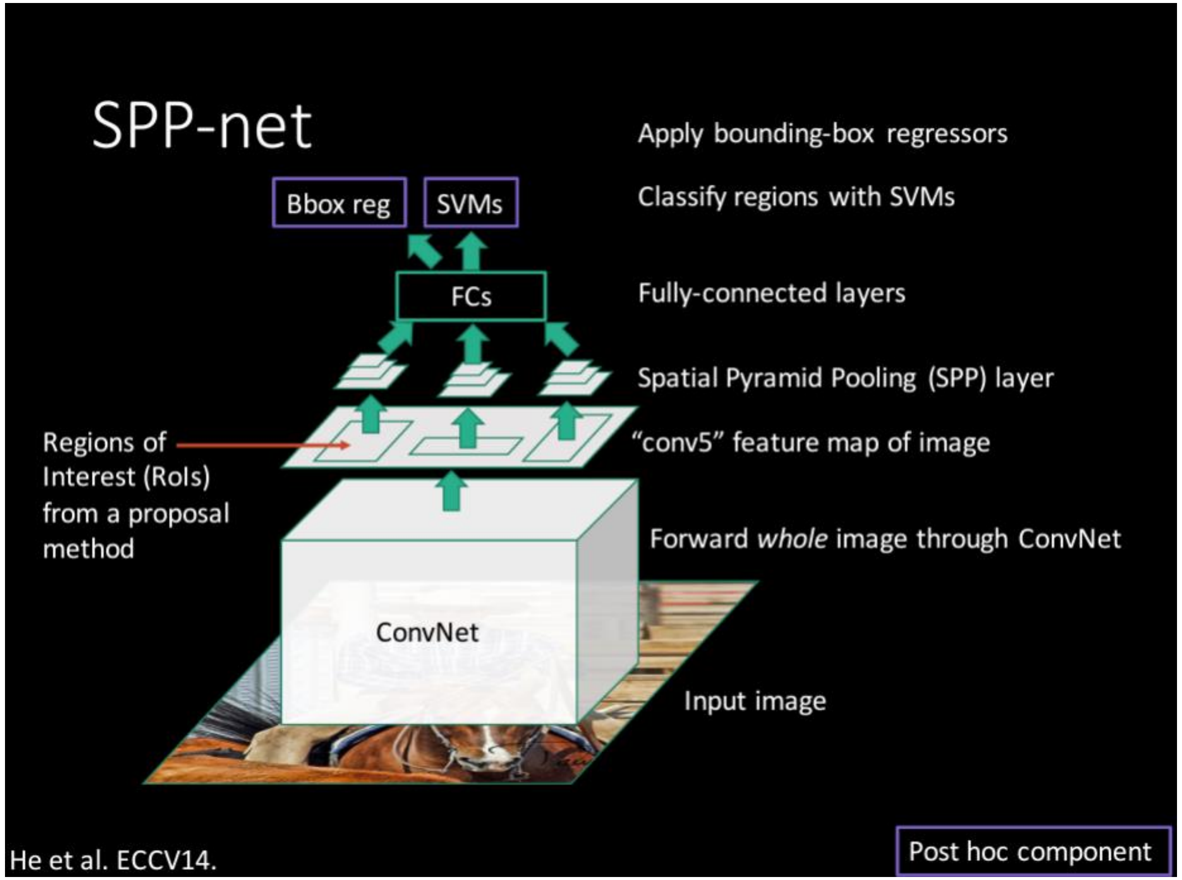
Model Workflow





SPPnet (Spatial Pyramid Pooling net)

Model Workflow

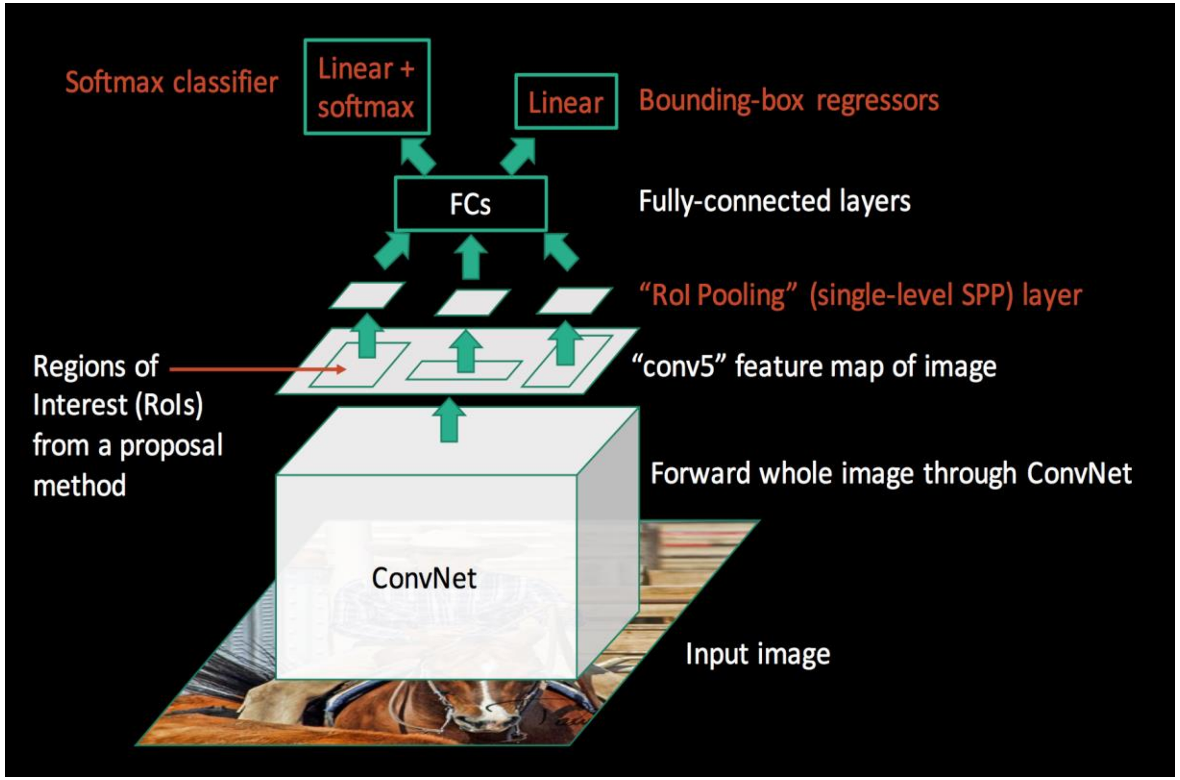


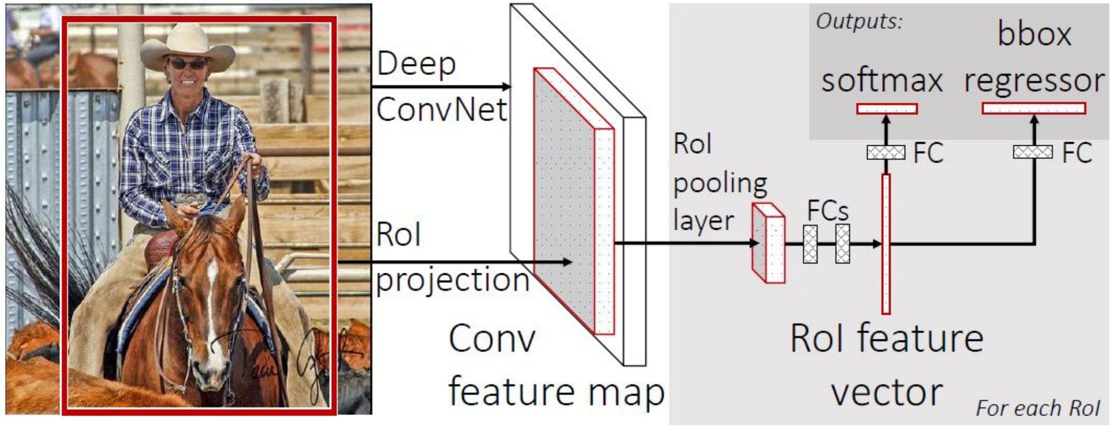
1. Forward **whole image** through ConvNet
2. Generate 2000 region proposals by using "fast" mode of Selective Search
3. Extract feature maps from the entire image
4. Forward pass 4-level **SPP** layer (1x1, 2x2, 3x3, 6x6) → Extract fixed size of feature vectors
5. Training binary linear SVM classifier for each category on these features
6. Bounding box regression to reduce mislocalization

- Still do not learn end-to-end
- when fine-tuning, it cannot train Conv Layers before SPP layer

Fast R-CNN

Model Workflow

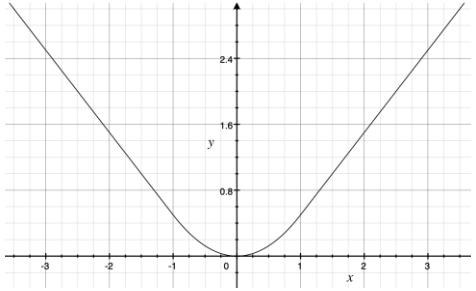




1. Pre-training CNN network on image classification task
2. Region proposal by Selective Search
3. Feature extraction by VGG16
4. Max pooling by **RoI pooling** (single-level SPP)
5. Fixed feature vector extraction by each object proposal
6. Class prediction by classifier
7. Detailed localization by Bounding box regressor
8. Train Softmax classifier and Bounding box regressor by multi-task loss

Fast R-CNN

Multi-task Loss



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

<Classification>

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

➡ Train a single model to optimize multiple objects

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

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

<Regression>

	S				M				L			
multi-task training?		✓		✓		✓		✓		✓		✓
stage-wise training?			✓				✓				✓	
test-time bbox reg?			✓	✓			✓	✓			✓	✓
VOC07 mAP	52.2	53.3	54.6	**57.1**	54.7	55.5	56.6	**59.2**	62.6	63.4	64.0	**66.9**

Table 6. Multi-task training (forth column per group) improves mAP over piecewise training (third column per group).

Mini-batch sampling

: SGD minibatch $N = 2$ images, size $R = 128$, sampling 64 RoIs from each image

- 25% of RoIs = $IoU \geq 0.5 \rightarrow$ labeled foreground object ($u = 1$)
- Remaining RoI = $0.1 \leq IoU < 0.5 \rightarrow$ labeled background object ($u = 0$)

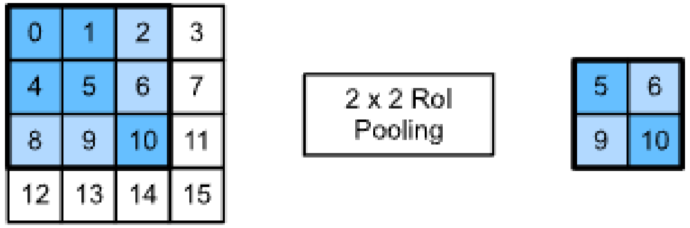
➡ Reduce computation cost and improve model's generalization performance

Fast R-CNN

Backpropagation through RoI pooling layers

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r,j)] \frac{\partial L}{\partial y_{rj}}$$

$$y_{rj} = x_{i^*(r,j)}$$
$$i^*(r,j) = argmax_{i' \in \mathcal{R}(r,j)} x_{i'}$$



➡ Generate fixed-size vector by using max pooling

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

SGD Hyper-parameters

- Softmax classification ~ $\mathcal{N}(0, (0.01)^2)$
- Bounding-box regression ~ $\mathcal{N}(0, (0.001)^2)$
- Learning rate : 0.001
- All layers use a per-layer learning rate of 1 for weights and 2 for biases
- VOC07 or VOC12 → SGD for $30k$ iterations → lower the lr to 0.0001 → train $10k$ iterations
- $Momentum\ =\ 0.9$, $Parameter\ decay\ =\ 0.0005$

Fast R-CNN

Fast R-CNN Detection

$$\mathrm{P}(class = k \mid r) \triangleq p_k \qquad (r: \ RoI\ (confidence),\ p\text{: } posterior,\ k: \ object\ class,\ \triangleq : defined\ as)$$

Truncated SVD

$$W \approx U\Sigma_t V^T$$

Truncated SVD reduces the parameter count from uv to $t(u+v)$ ($t < \min(u,v)$)

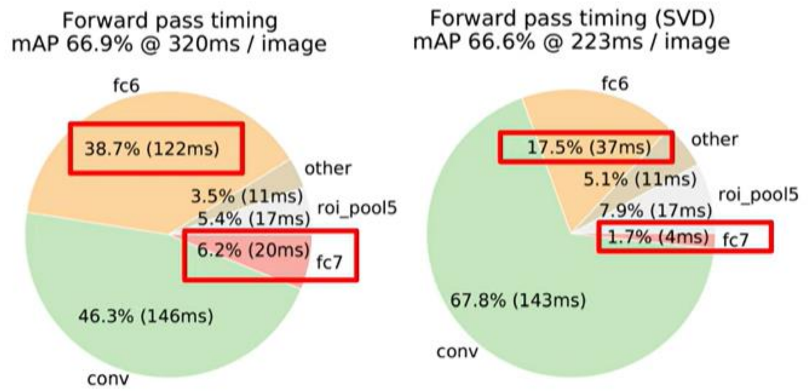
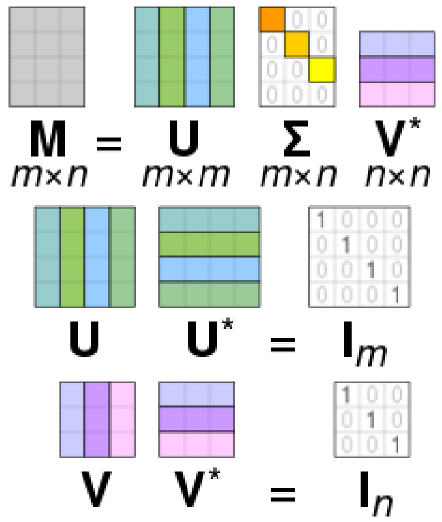


Figure 2. Timing for VGG16 before and after truncated SVD. Before SVD, fully connected layers fc6 and fc7 take 45% of the time.



To compress a network, the single fc layer corresponding to W is replaced by two fc layers ($\Sigma_t V$, U)

⇒ By compression, we can speedup when the number of RoIs is large

Result

[VOC 2007, 2010, 2012]

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [11]†	07 \ diff	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	73.5	63.1
R-CNN BB [10]	07	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	**35.6**	66.8	67.2	70.4	**71.1**	66.0
FRCN [ours]	07	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	66.9
FRCN [ours]	07 \ diff	74.6	**79.0**	68.6	57.0	39.3	79.5	**78.6**	81.9	**48.0**	74.0	67.4	80.5	80.7	74.1	69.6	31.8	67.1	68.4	75.3	65.5	68.1
FRCN [ours]	07+12	**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	**70.0**

Table 1. **VOC 2007 test** detection average precision (%). All methods use VGG16. Training set key: **07**: VOC07 trainval, **07 \ diff**: **07** without "difficult" examples, **07+12**: union of **07** and VOC12 trainval. †SPPnet results were prepared by the authors of [11].

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	77.7	73.8	62.3	48.8	45.4	67.3	67.0	80.3	41.3	70.8	49.7	79.5	74.7	78.6	64.5	36.0	69.9	55.7	70.4	61.7	63.8
R-CNN BB [10]	12	79.3	72.4	63.1	44.0	44.4	64.6	66.3	84.9	38.8	67.3	48.4	82.3	75.0	76.7	65.7	35.8	66.2	54.8	69.1	58.8	62.9
SegDeepM	12+seg	**82.3**	75.2	67.1	50.7	**49.8**	71.1	69.6	88.2	42.5	71.2	50.0	85.7	76.6	81.8	69.3	**41.5**	**71.9**	62.2	73.2	**64.6**	67.2
FRCN [ours]	12	80.1	74.4	67.7	49.4	41.4	74.2	68.8	87.8	41.9	70.1	50.2	86.1	77.3	81.1	70.4	33.3	67.0	63.3	77.2	60.0	66.1
FRCN [ours]	07++12	82.0	**77.8**	**71.6**	**55.3**	42.4	**77.3**	**71.7**	**89.3**	**44.5**	**72.1**	**53.7**	**87.7**	**80.0**	**82.5**	**72.7**	36.6	68.7	**65.4**	**81.1**	62.7	**68.8**

Table 2. **VOC 2010 test** detection average precision (%). BabyLearning uses a network based on [17]. All other methods use VGG16. Training set key: **12**: VOC12 trainval, **Prop.**: proprietary dataset, **12+seg**: **12** with segmentation annotations, **07++12**: union of VOC07 trainval, VOC07 test, and VOC12 trainval.

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6	63.2
NUS_NIN_c2000	Unk.	80.2	73.8	61.9	43.7	**43.0**	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	**38.6**	**68.3**	58.0	68.7	63.3	63.8
R-CNN BB [10]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
FRCN [ours]	12	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7	65.7
FRCN [ours]	07++12	**82.3**	**78.4**	**70.8**	**52.3**	38.7	**77.8**	**71.6**	**89.3**	**44.2**	**73.0**	**55.0**	**87.5**	**80.5**	**80.8**	**72.0**	35.1	**68.3**	**65.7**	**80.4**	**64.2**	**68.4**

Table 3. **VOC 2012 test** detection average precision (%). BabyLearning and NUS_NIN_c2000 use networks based on [17]. All other methods use VGG16. Training set key: see Table 2, **Unk.**: unknown.

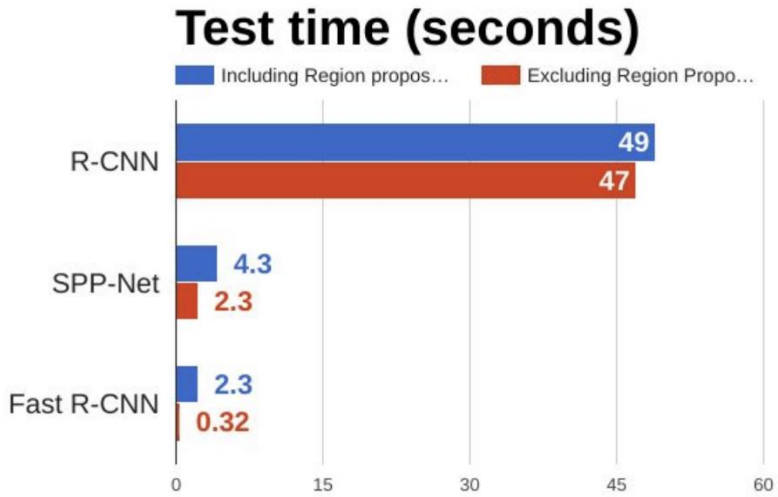
Runtime Comparison

	Fast R-CNN			R-CNN			SPPnet
	S	**M**	**L**	**S**	**M**	**L**	†**L**
train time (h)	**1.2**	2.0	9.5	22	28	84	25
train speedup	**18.3×**	14.0×	8.8×	1×	1×	1×	3.4×
test rate (s/im)	0.10	0.15	0.32	9.8	12.1	47.0	2.3
▷ with SVD	**0.06**	0.08	0.22	-	-	-	-
test speedup	98×	80×	146×	1×	1×	1×	20×
▷ with SVD	169×	150×	**213×**	-	-	-	-
VOC07 mAP	57.1	59.2	**66.9**	58.5	60.2	66.0	63.1
▷ with SVD	56.5	58.7	66.6	-	-	-	-

Table 4. Runtime comparison between the same models in Fast R-CNN, R-CNN, and SPPnet. Fast R-CNN uses single-scale mode. SPPnet uses the five scales specified in [11]. †Timing provided by the authors of [11]. Times were measured on an Nvidia K40 GPU.

Conclusion

Fast R-CNN



Strength

- End-to-end training of deep ConvNets for detection
- Fast training times
- Multiple models can be trained in single stage using **multi-task loss**
- Weight values of the network can be updated through **backpropagation**

	R-CNN	Fast R-CNN
Training Time:	84 hours	**9.5 hours**
(Speedup)	1x	**8.8x**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
mAP (VOC 2007)	66.0	**66.9**

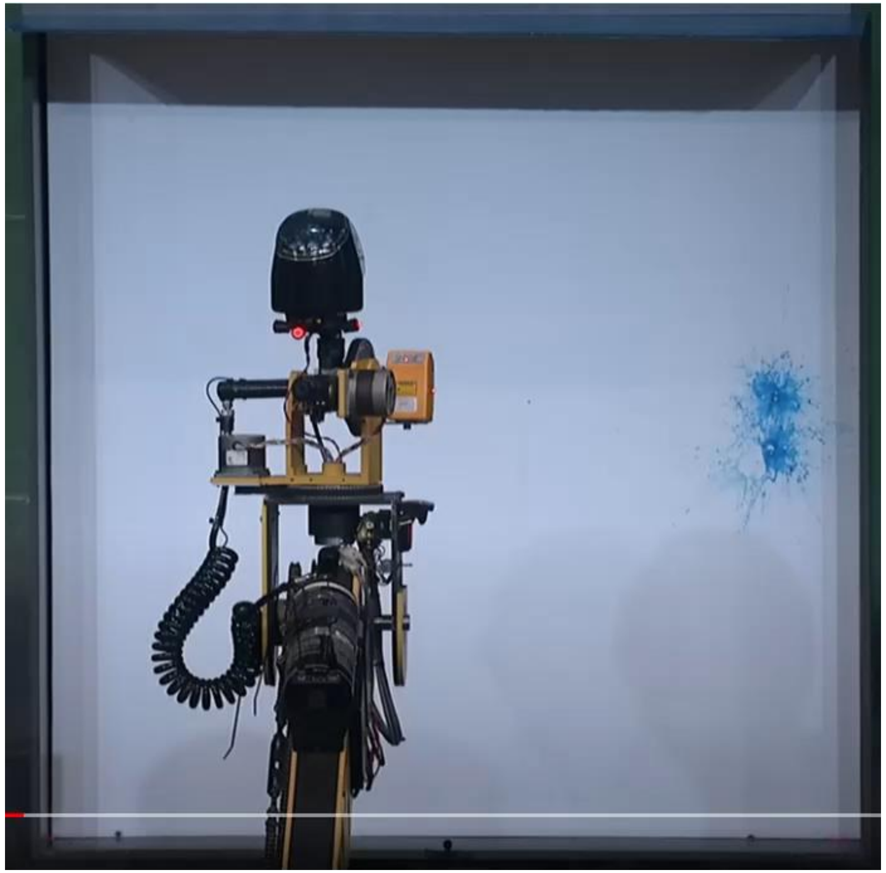
Using VGG-16 CNN on Pascal VOC 2007 dataset

Weakness

- Dependence on Selective Search → computationally expensive and slow
- Difficulty in Real-time Application because of reliance on region proposals (bottleneck)

Ⅲ. Faster R-CNN

More Faster?



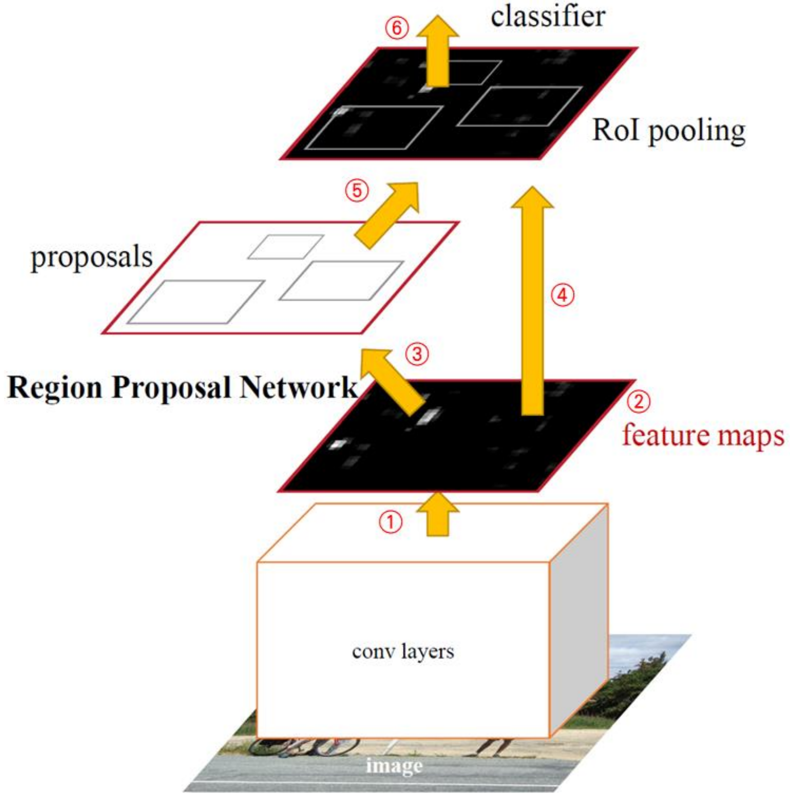
Selective Search

??

[Mythbusters Demo GPU versus CPU]

Faster R-CNN

Model Workflow

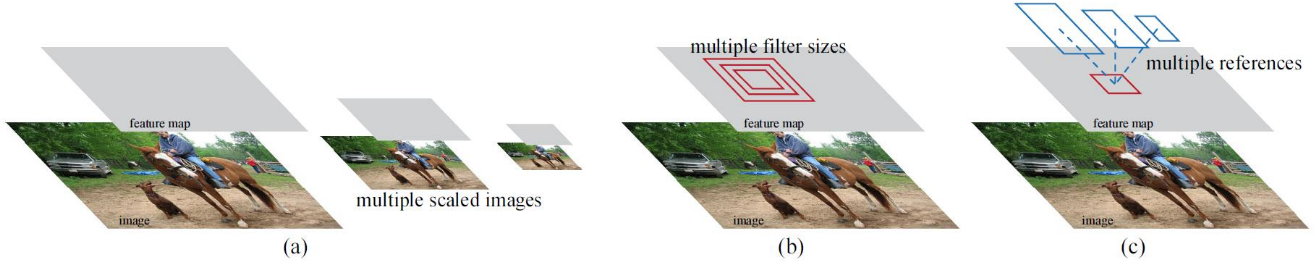


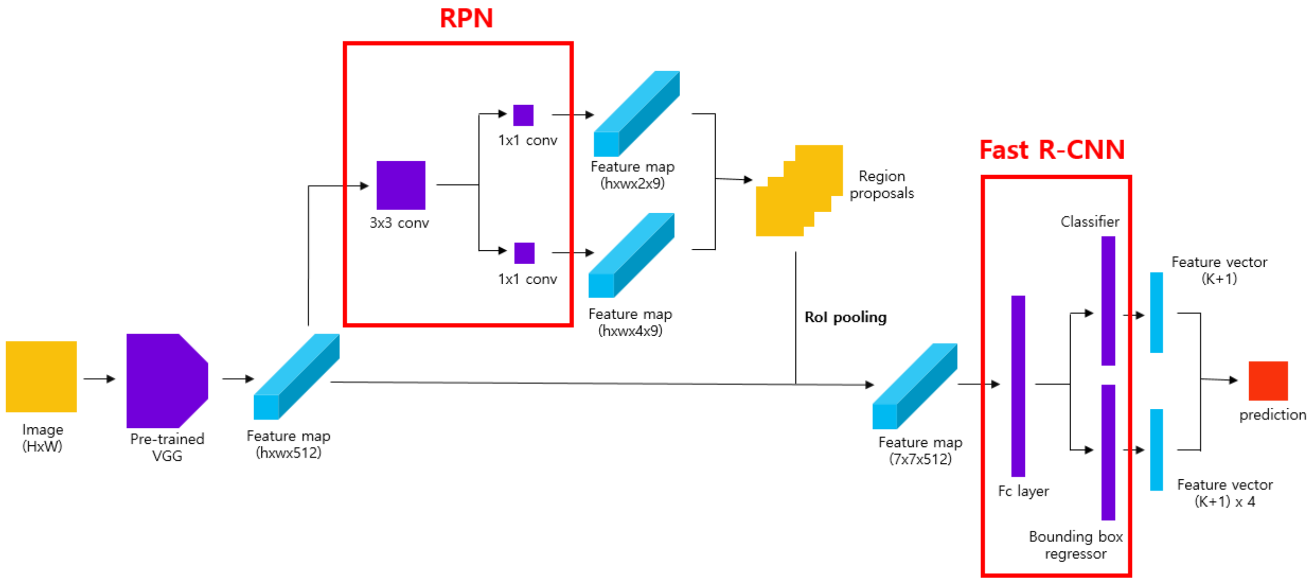
1. Pre-trained Convolutional Layers
   - Extract feature maps from input image
2. RPN (Region Proposal Network)
   - Use feature maps to generate region proposals
3. Generate region proposals directly
4. RoI pooling
   - Extract fixed size feature representations for each proposal
5. Classification and Bounding box regression

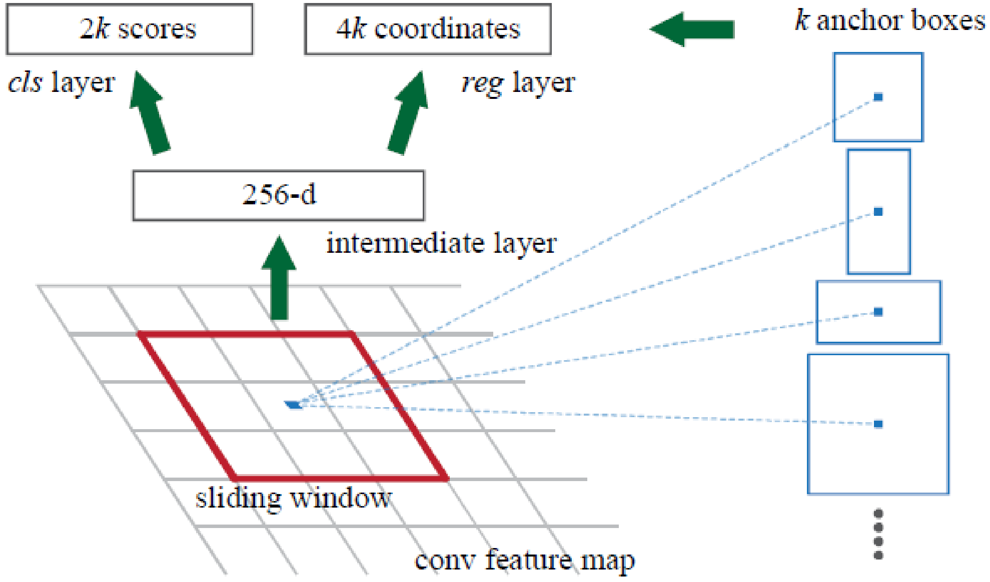
Faster R-CNN = Fast R-CNN + RPN

Faster R-CNN

RPN (Region Proposal Networks)



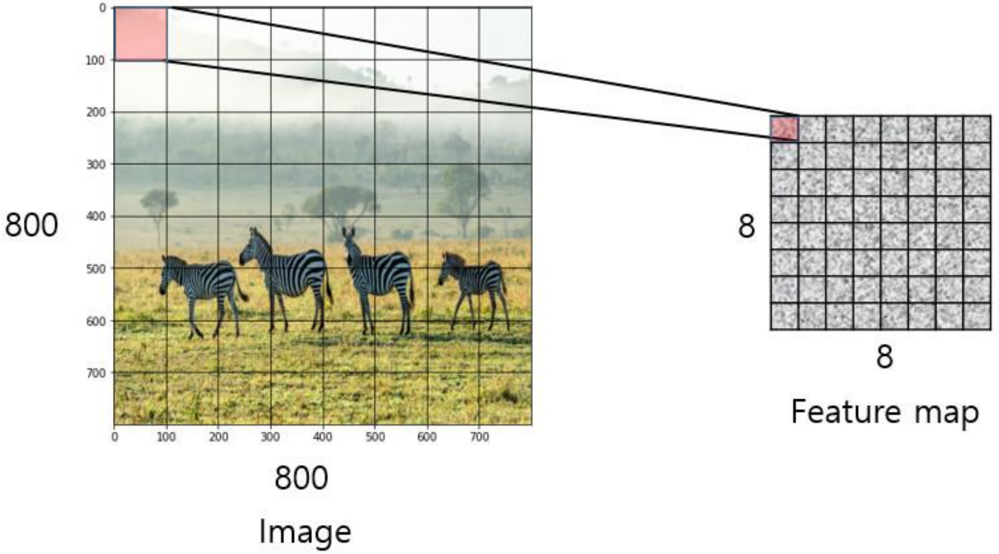


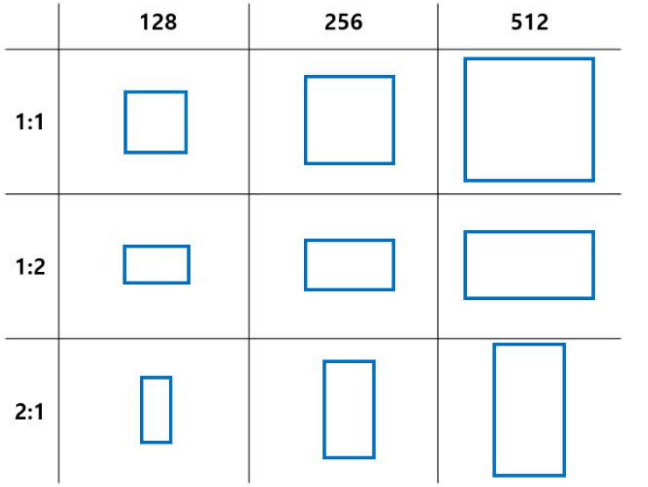


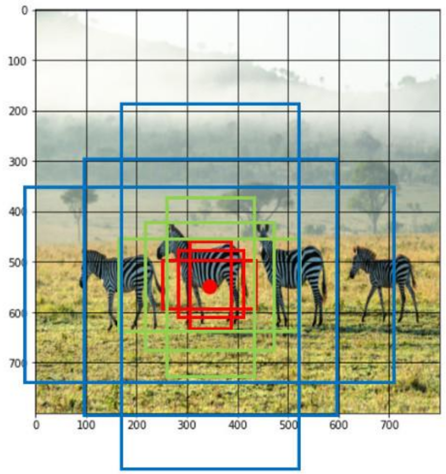
- Training **end-to-end** by backpropagation and SGD
- Sampling 256 anchors (128 (positive) + 128 (negative))
- $W \sim \mathcal{N}(0, (0.01)^2)$
- $lr = 0.001$ for $60k$ mini-batches $\rightarrow$ $lr = 0.0001$ for the next $20k$ mini-batches on the PASCAL VOC
- $Momentum = 0.9$, $weight\ decay = 0.0005$

Faster R-CNN

Anchors







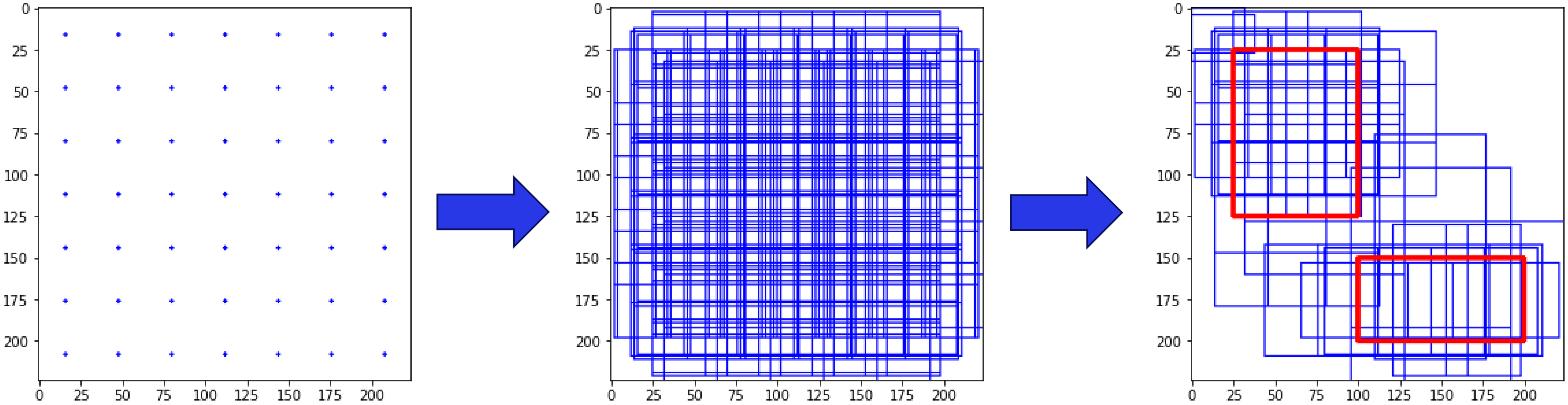
- Scale : $(128, 256, 512)$, Aspect Ratio : $(2{:}1, 1{:}1, 1{:}2)$
- $W * H * k$ anchors ($k = 9$ in paper)
- **Translation Invariant** → Guarantee same proposal
- Overfitting ↓ (Small dataset)

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**

Faster R-CNN

Anchors



$$\begin{cases} Positive\ \ (1) & if, IoU \geq 0.7 \\ Negative\ \ (0) & elif, IoU \leq 0.3 \\ No\ Contribution & O.T.W \end{cases}$$

Faster R-CNN

Loss Function

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

<Classification>　　　　<Regression>

p_i : *Predicted probability*
t_i : *Predicted bbox coordinate*
N_{cls} : *#anchors in minibatch* (256)
N_{reg} : *#anchor location*
p_i^* : *Ground truth object label*
t_i^* : *True box coordinates*
L_{cls} : *binary cross entropy loss*
L_{reg} : *Smooth L1 loss*
λ : *parameter* (*In practice*, $\lambda = 10$)

$$L_{cls} = -(y\log(p) + (1-y)\log(1-p))$$

$L_{reg}(t_i,t_i^*) = R(t_i - t_i^*)$ (R : $Smooth\ L_1$ in Fast R-CNN)

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

Result

Comparison

Model	Test time	Speed up	mAP (VOC07)	mAP (VOC07+12)
R-CNN	50s	1x	66.0	–
Fast R-CNN	2s	25x	66.9	70.0
Faster R-CNN	0.2s	250x	69.9	73.2

PASCAL VOC 2007

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. †: 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†
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**

Selective Search → RPN

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.

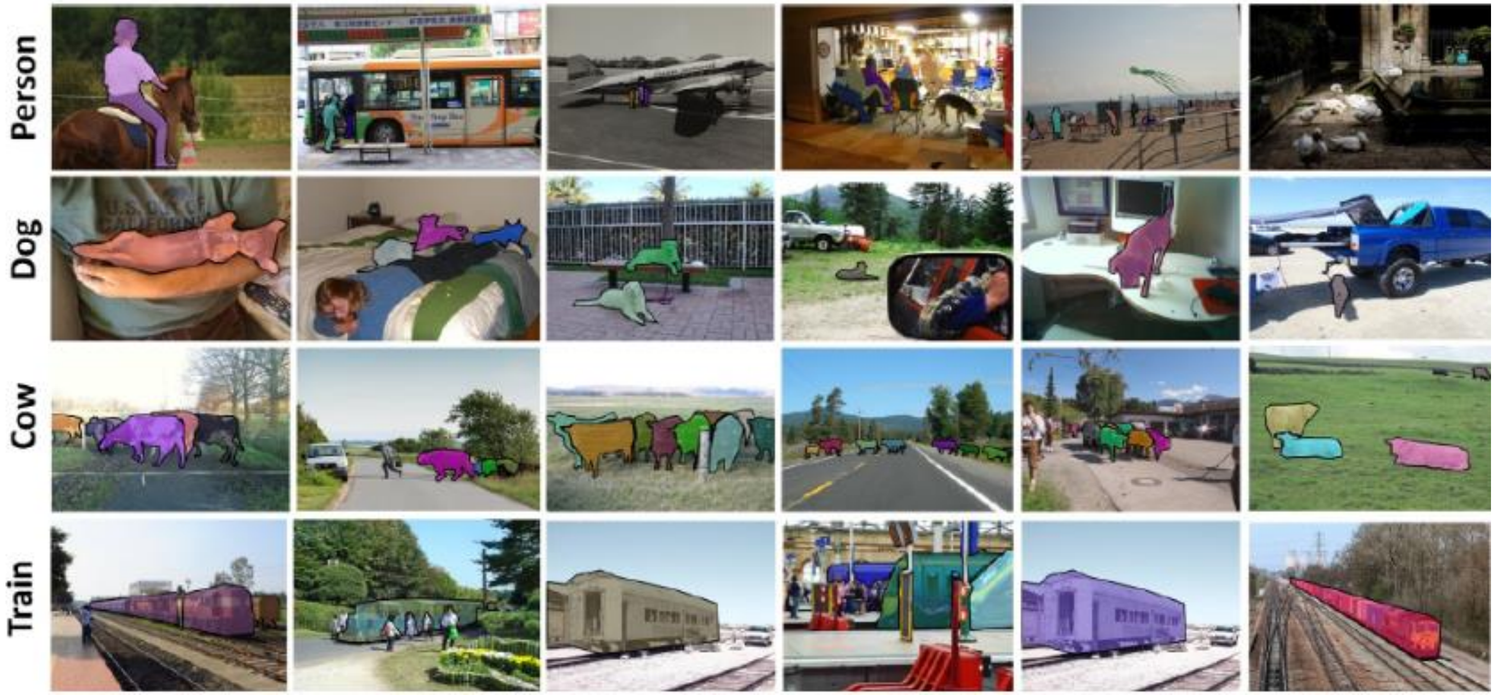
model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	**10**	47	**198**	**5 fps**
ZF	RPN + Fast R-CNN	31	**3**	25	**59**	**17 fps**

Result

MS COCO

Table 11: Object detection results (%) on the **MS COCO** dataset. The model is VGG-16.

method	proposals	training data	COCO val		COCO test-dev	
			mAP@.5	mAP@[.5, .95]	mAP@.5	mAP@[.5, .95]
Fast R-CNN [2]	SS, 2000	COCO train	-	-	35.9	19.7
Fast R-CNN [impl. in this paper]	SS, 2000	COCO train	38.6	18.9	39.3	19.3
Faster R-CNN	RPN, 300	COCO train	41.5	21.2	42.1	21.5
Faster R-CNN	RPN, 300	COCO trainval	-	-	**42.7**	**21.9**



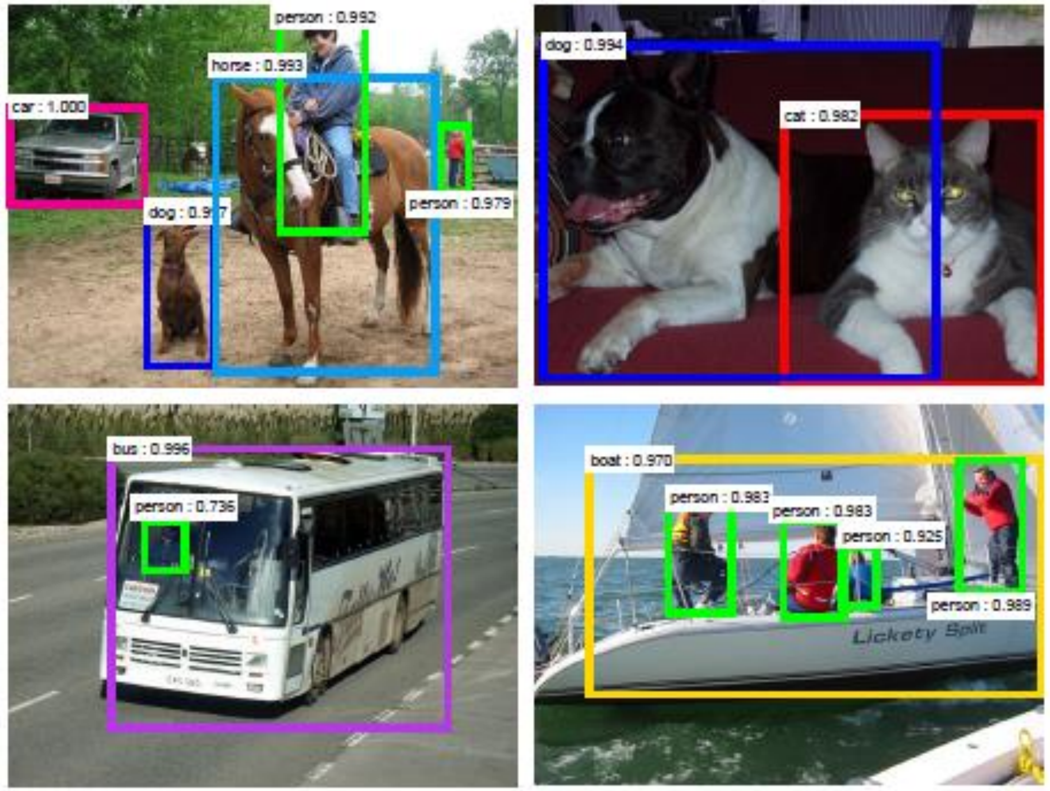
What is COCO?

COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

- Object segmentation
- Recognition in context
- Superpixel stuff segmentation
- 330K images (>200K labeled)
- 1.5 million object instances
- 80 object categories
- 91 stuff categories
- 5 captions per image
- 250,000 people with keypoints

Conclusion

Faster R-CNN



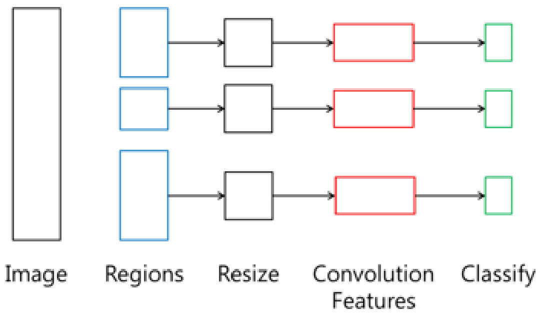
Strength

- Replace Selective Search (CPU) with RPN (GPU) ➡ **FAST**
- Translation invariant : Extract same feature anywhere
- Region proposal step is nearly cost-free
- End-to-end training

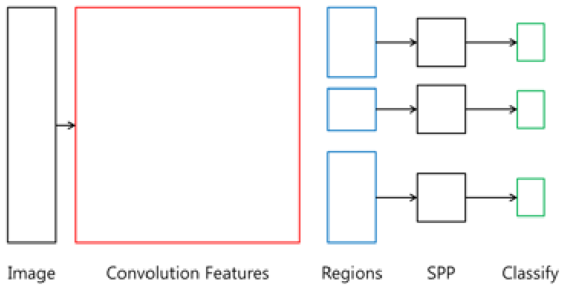
Weakness

- if the Region proposal is not a multiple of 7 when extracting the 7x7 feature map by RoI pooling, it must be discarded or rounded, and errors may occur here.

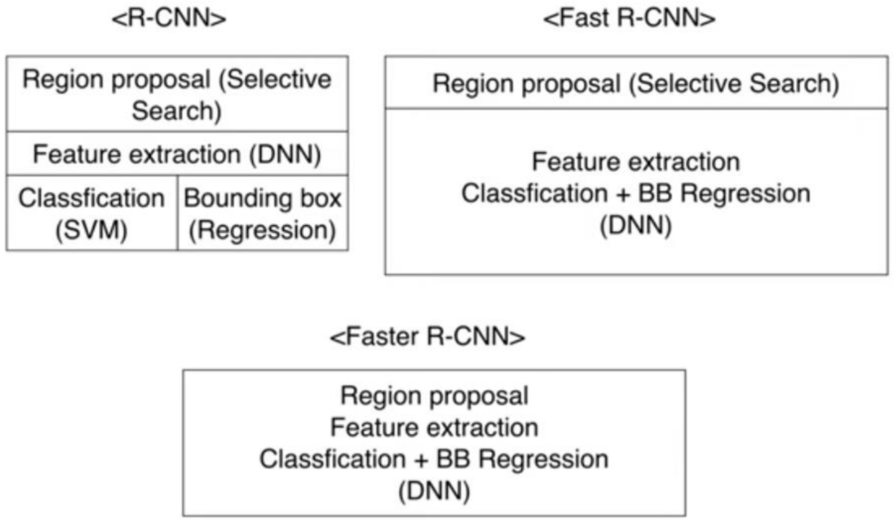
Conclusion

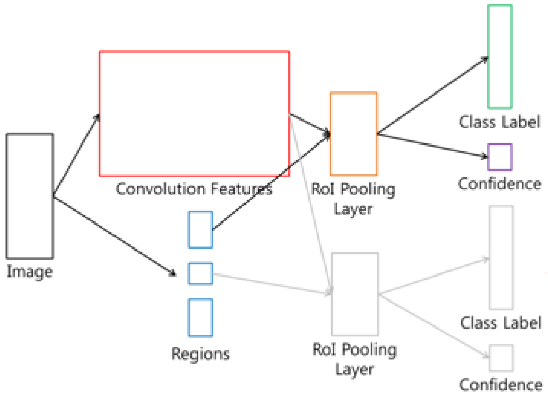


R-CNN

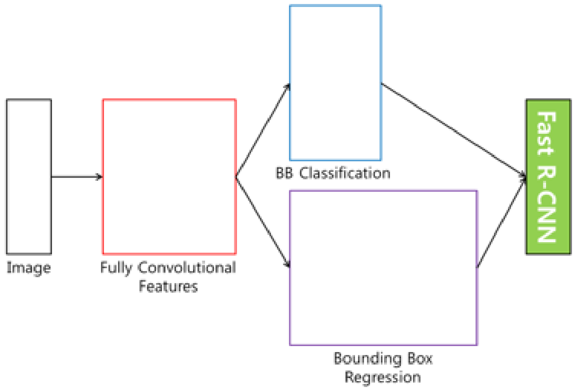


SPP net





Fast R-CNN



Faster R-CNN

Reference

- Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik.
  「**Rich feature hierarchies for accurate object detection and semantic segmentation**」, 2014
- J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, A.W.M. Smeulders. 「**Selective Search for Object Recognition**」, 2012
- Pedro F. Felzenszwalb, Daniel P. Huttenlocher. 「**Efficient Graph-Based Image Segmentation**」, 2004
- Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. 「**ImageNet Classification with Deep Convolutional Neural Networks**」, 2012
- Ross Girshick, Microsoft Research. 「**Fast R-CNN**」**,** 2015
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. 「**Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition**」**,** 2015
- Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. 「**Faster R-CNN: Toward Real-Time Object Detection with Region Proposal Networks**」**,** 2015
- https://www.robots.ox.ac.uk/~tvg/publications/talks/fast-rcnn-slides.pdf
- https://kikaben.com/r-cnn-original/
- Selecting the right bounding box using non max suppression with implementation
- https://lilianweng.github.io/posts/2017-12-31-object-recognition-part-3/
- https://learnopencv.com/understanding-alexnet/
- https://kikaben.com/fast-r-cnn-213/
- https://oi.readthedocs.io/en/latest/computer_vision/object_detection/faster_r-cnn.html
- https://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

THANK YOU