Deep Sync

Fast R-CNN

강동규

DeepSync, South Korea

Tasks of Computer Vision

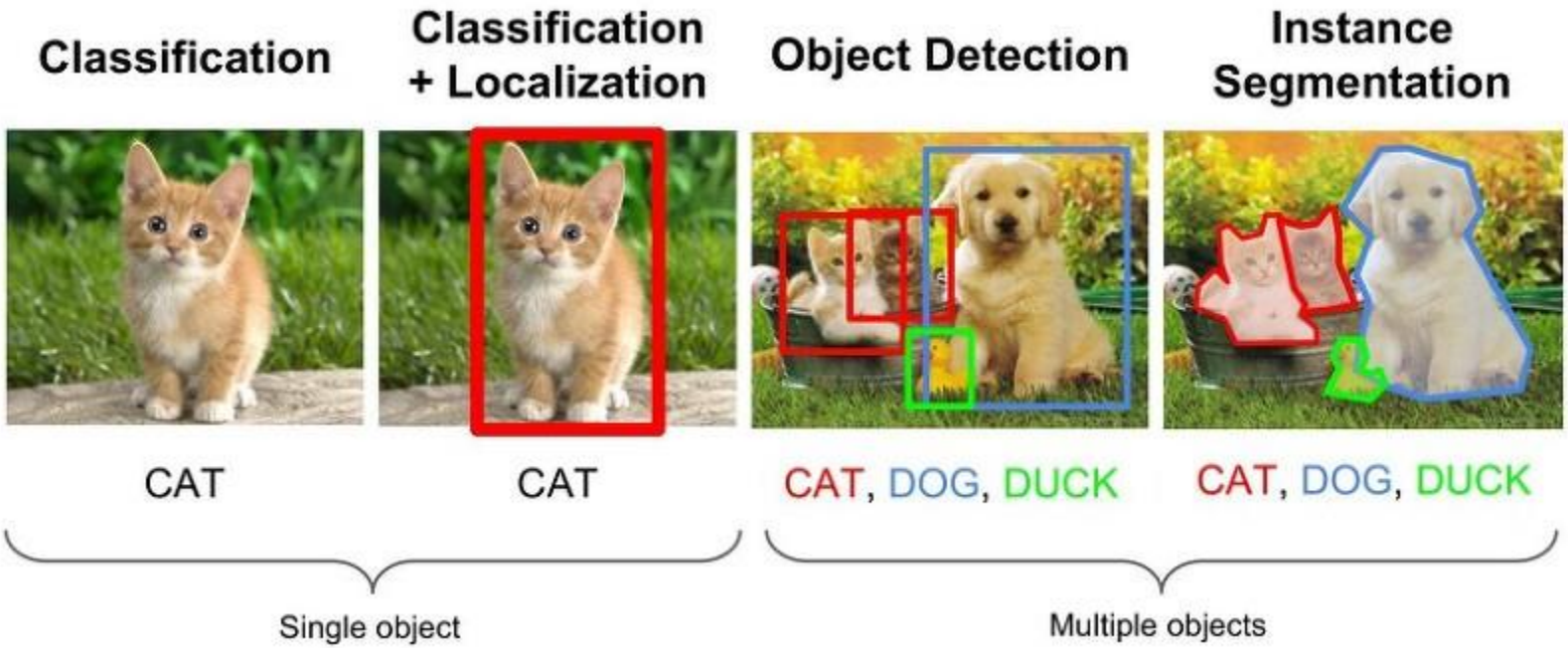


Fig1

Overview of R-CNN Structure

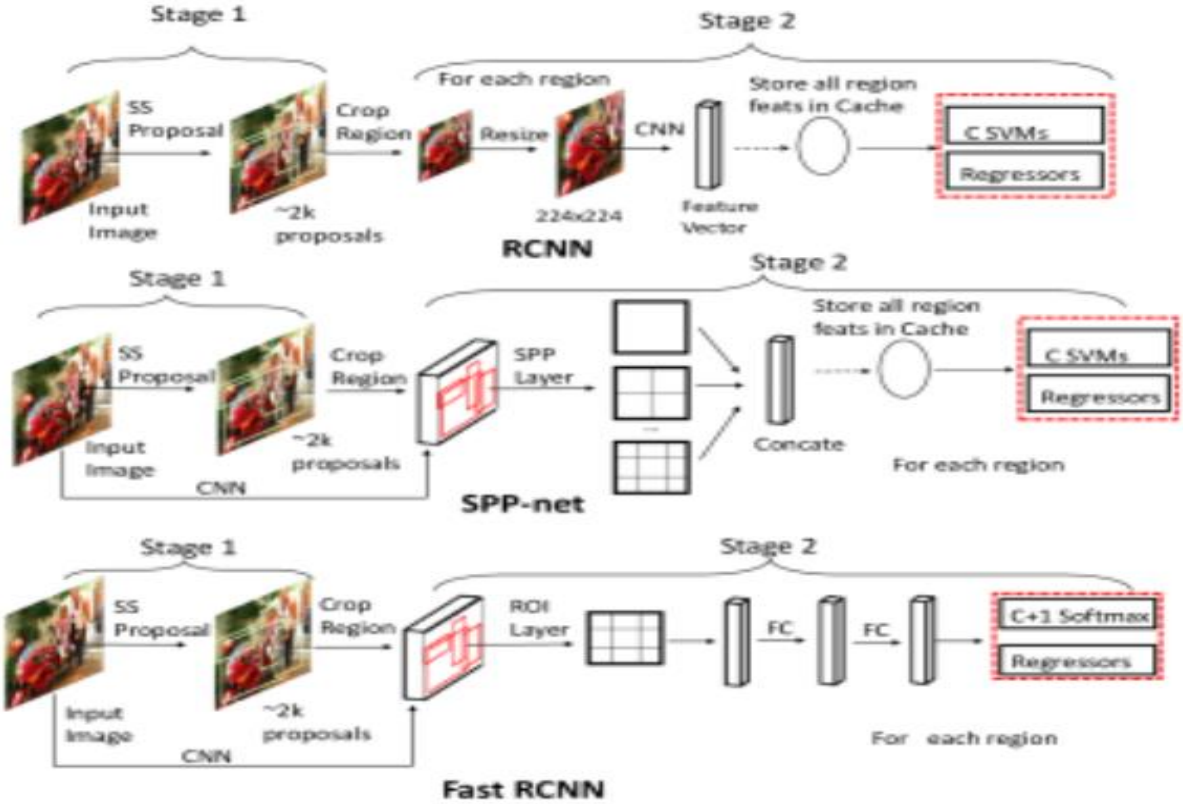


Fig2

Problems of RCNN – 1. Warping (Crop, Resize)

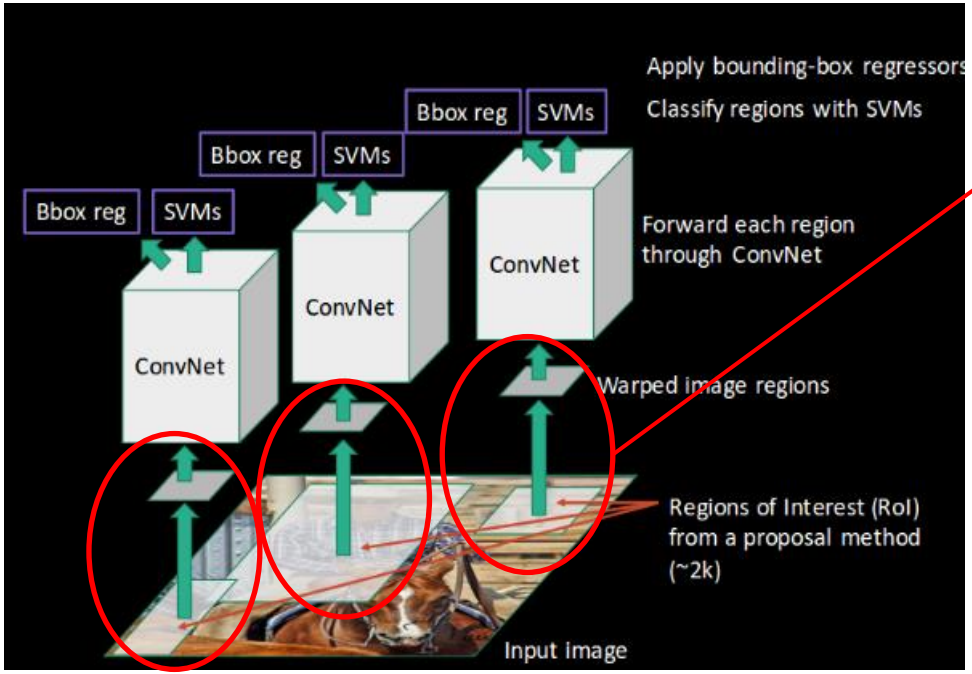
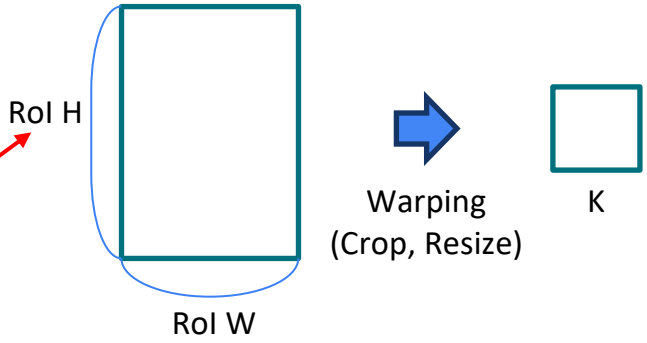


Fig3



- **Warping(Crop, Resize)**
- **CNN에 입력하기 위해 전처리**
- **Crop, Resize를 하면서 이미지 변형이 발생**
- **이미지 변형 -> 정확도에 악영향**

Problems of RCNN – 2. 2000 CNN

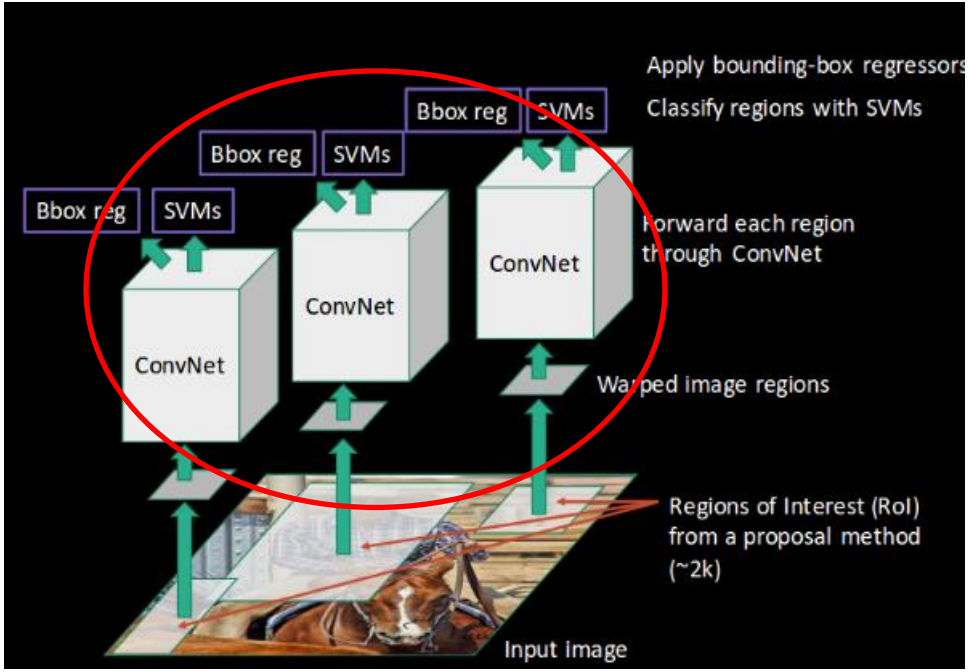


Fig3

- 2000개의 RoI에 개별적으로 CNN 적용

- RoI 각각에 적용하기 때문에 시간 복잡도 상승

- RoI 각각에 대해 CNN을 적용한 특징맵 값을 Bbox, SVM에 적용하기 위해 Caching -> 공간 복잡도 상승

Problems of RCNN – 3. Multi-stage Pipeline

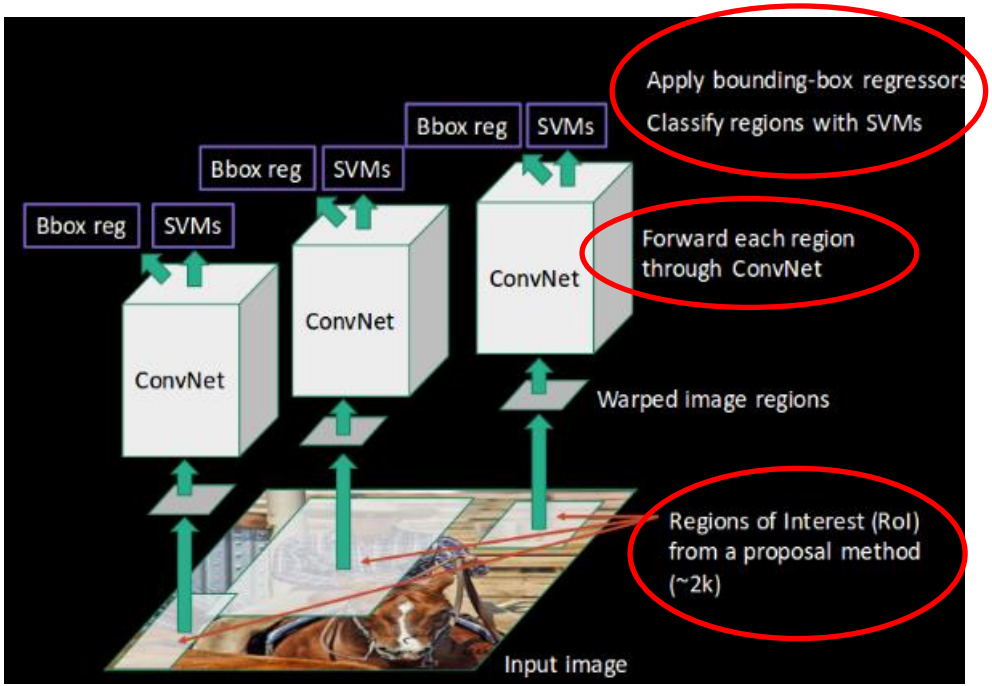


Fig3

- 여러 단계가 개별적으로 나누어져 있어 **End-to-End 학습**을 할 수 없다.
- **CNN**을 학습하더라도 **Selective Search** 와 같은 다른 **stage**가 학습되지 않는다.

SPPnet

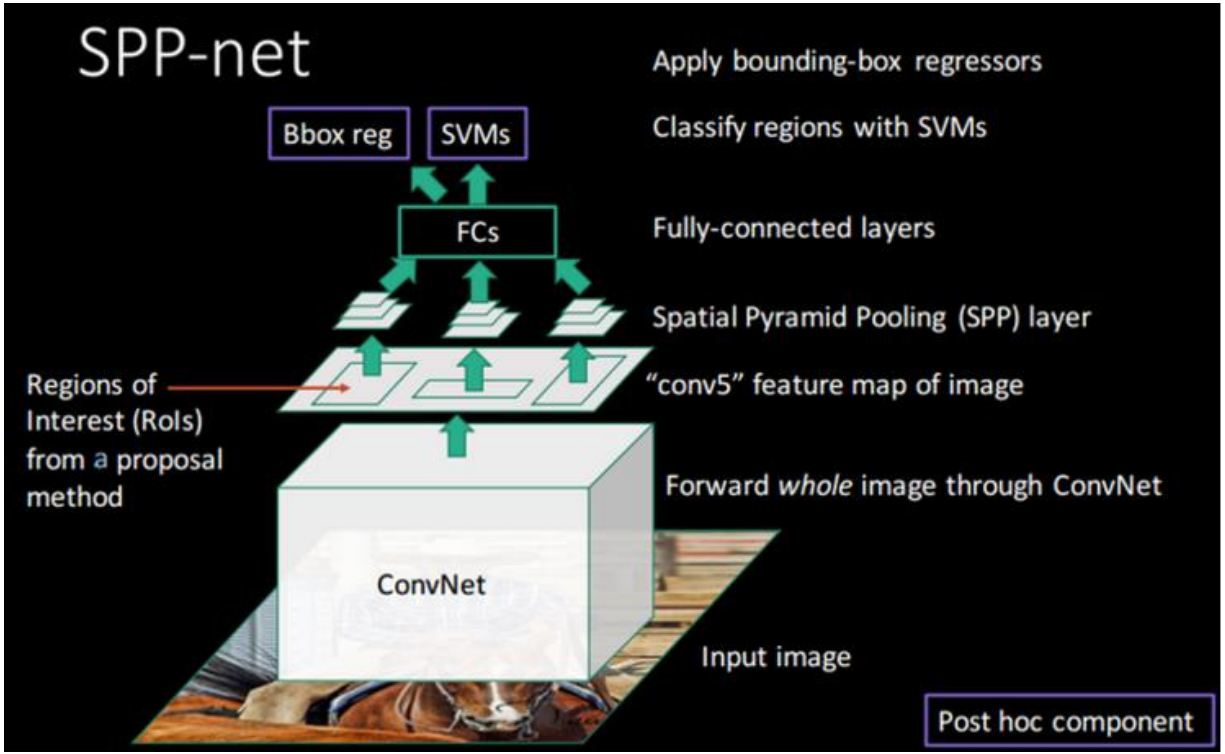

SPP-net
Apply bounding-box regressors
Bbox reg
SVMs
Classify regions with SVMs
FCs
Fully-connected layers
Spatial Pyramid Pooling (SPP) layer
Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image
Post hoc component

Fig4

Spatial Pyramid Pooling (SPP)

- 특징의 손실없이 고정된 길이의 표현 벡터를 생성하자

1. Spatial Bins 설정 (Hyperparameter, Bins = 21)
2. Bin의 개수에 맞는 Pooling 설정 (4x4, 2x2, 1x1)
3. 각 Pooling에 대해 Window Size, Stride를 adaptive하게 조정
4. 각 bin마다 MaxPooling 진행
5. Pooling 출력을 Concat, Flatten 하여 고정 길이 벡터 생성

*RoI 각각에 대해 다양한 크기의 Pooling을 적용한다.

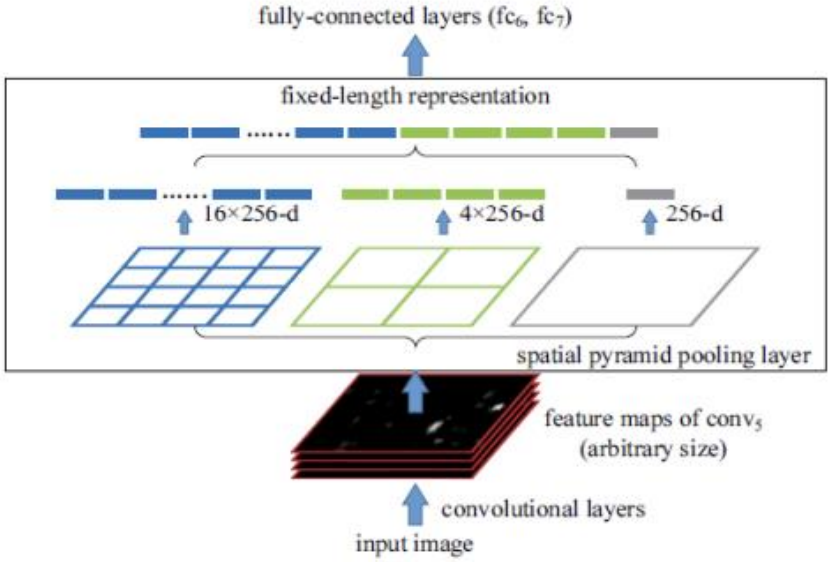
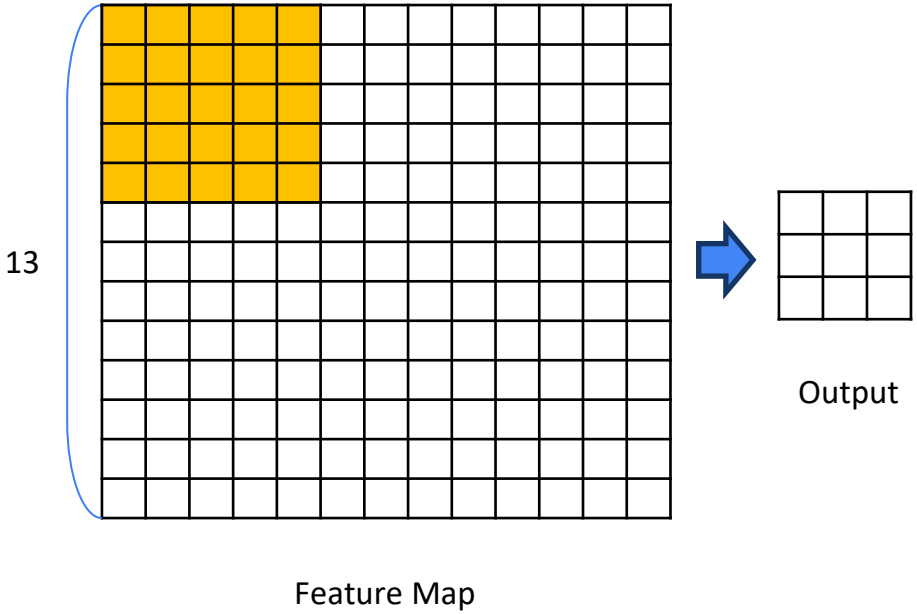


Fig5

cf) How to set Window_size and Stride?



- Window_size = Ceil(Feature Map Size / Pooling Size)
- Stride = Floor(Feature Map Size / Pooling Size)
- ex) 13x13 Feature Map, 3x3 Pooling

Window_size = Ceil(13/3) = Ceil(4.xx) = 5

Stride = Floor(13/3) = Floor(4.xx) = 4

Improvements of SPPnet

Deep Sync

1. 전체 이미지에 대해 CNN을 한 번만 적용한다.
(복잡도 감소)
2. 이미지, RoI의 크기, 비율에 영향을 받지 않는다.
3. Pyramid 방식을 통해 여러 공간적 특징을 추출해 정보의 손실을 줄이며 전달할 수 있다.

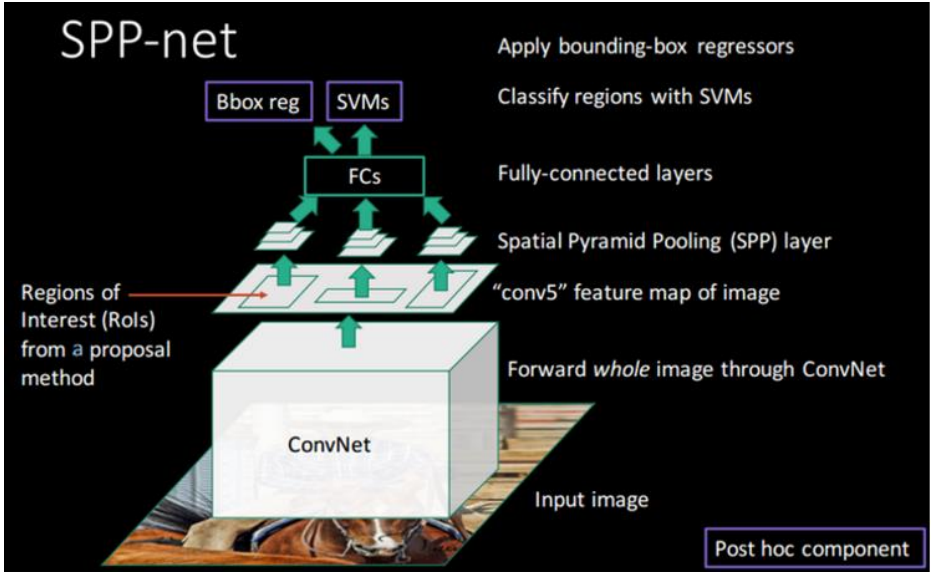
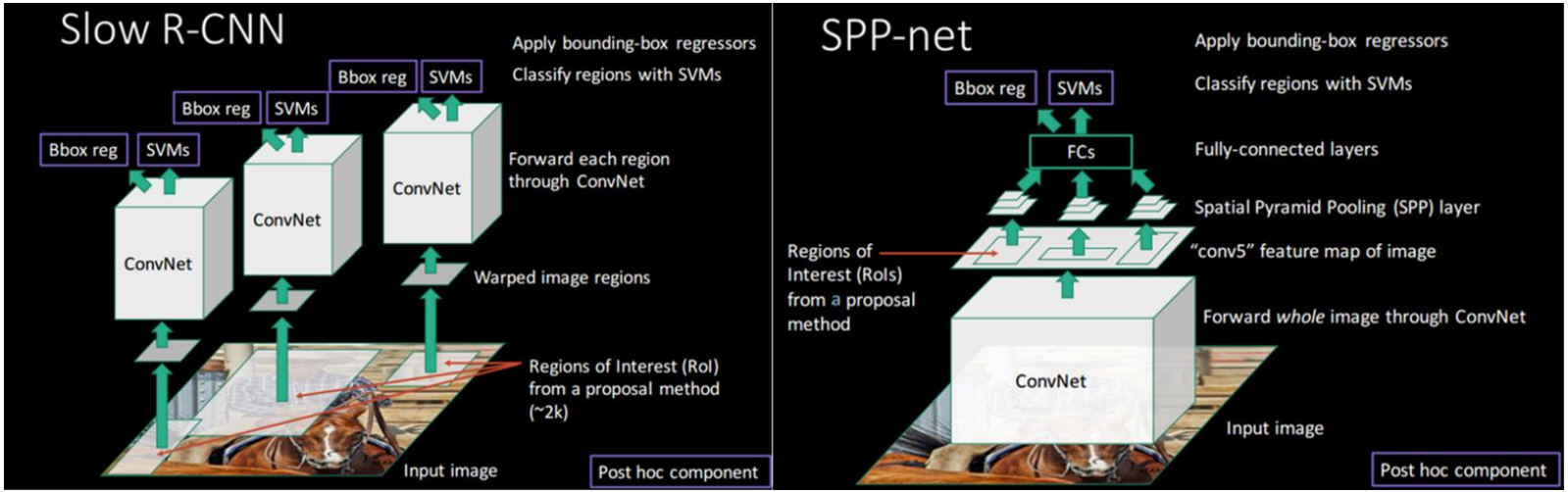


Fig4

Compare Structure of RCNN and SPPnet



- Problem of SPPnet

1. End-to-End 학습이 진행되지 않는다. -> FCs 가 학습되어도 CNN은 학습되지 않는다.

2. Bounding box regressor와 SVM이 개별적으로 처리 되기 때문에 cachin이 필요하다.

Overview of Fast R-CNN Structure

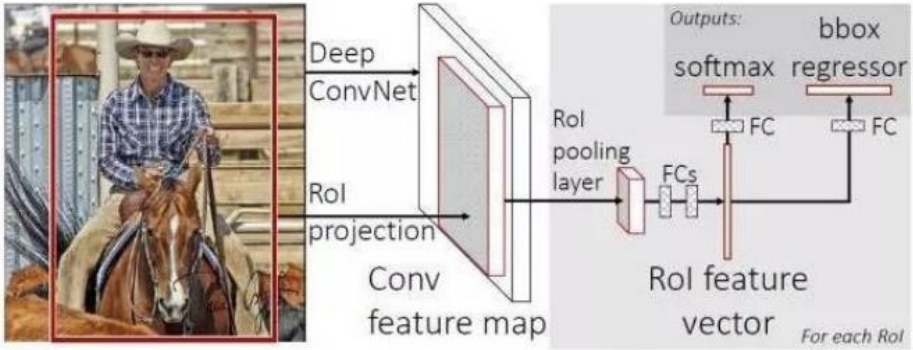


Fig6

- 모델 적용 과정

1. 전체 입력 이미지에 Selective Search로 RoI 추출
2. 전체 입력 이미지를 CNN에 입력해 특징맵 추출
3. RoI를 특징맵 크기에 맞도록 Projection
4. Projection된 RoI에 RoI Pooling 적용
5. FC layers 통과
6. 출력 벡터를 Softmax, Bbox regressor에 입력

Goals of Fast R-CNN Structure

Deep Sync

- Goals of Fast R-CNN
- End-to-End Training(Softmax, Multi-task loss)
1. RoI Pooling
2. Multi-task loss
3. Mini-batch Sampling

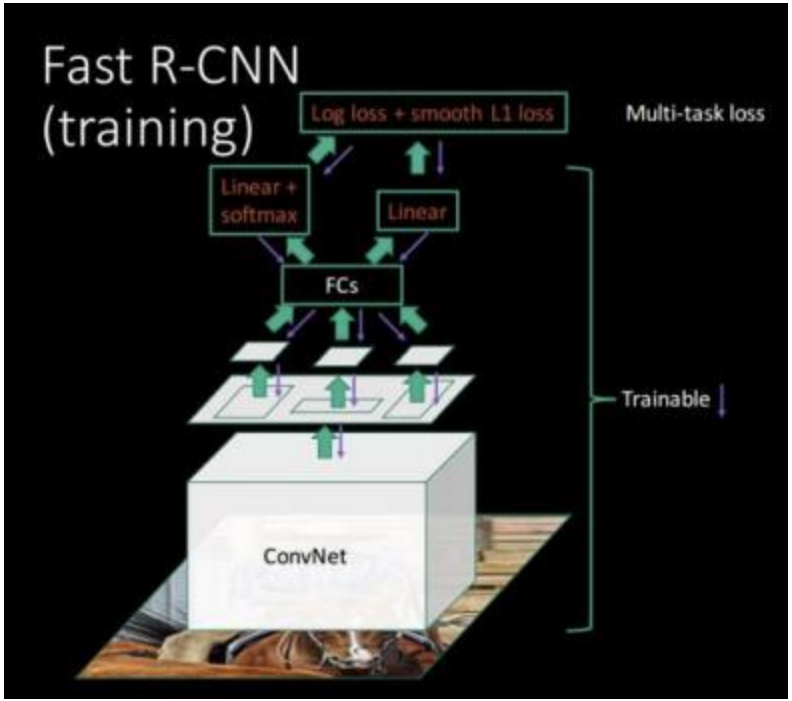


Fig7

RoI Projection & RoI Pooling

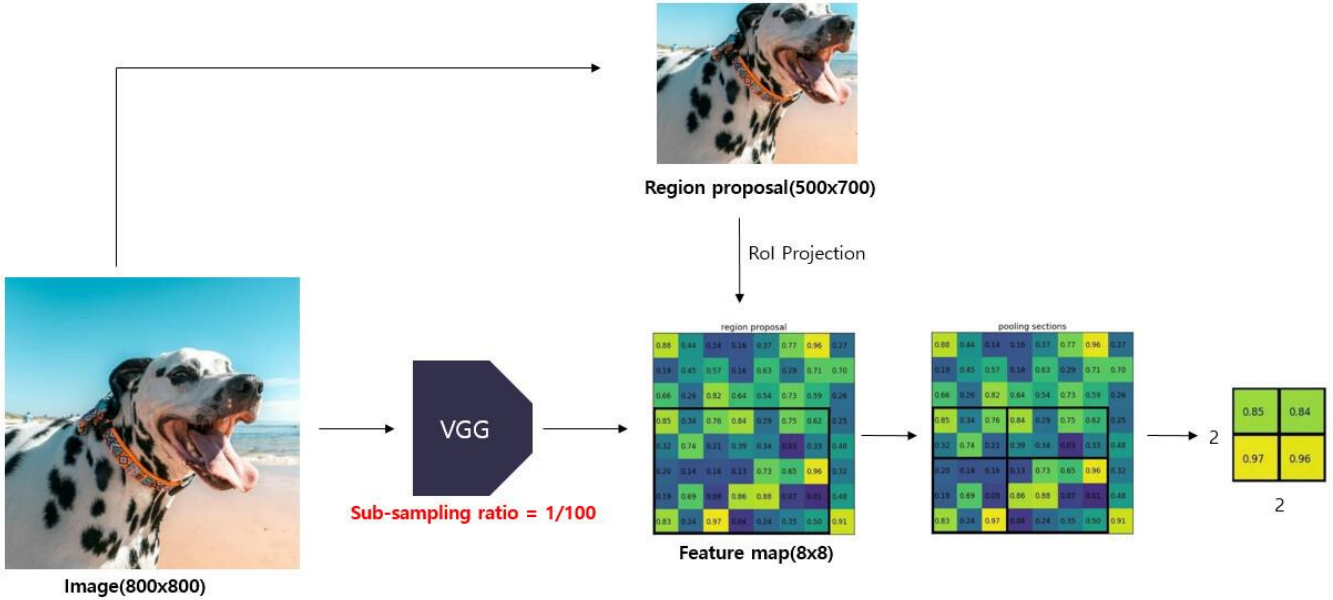


Fig8

Multi-Task Loss

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

- p : K+1 개의 Class Score(K개의 객체 + 배경)
- u : 정답 객체 Class Score
- t^u : 예측한 K 객체에 대한 bbox 좌표 조정값
- v : 정답 bbox 좌표값
- λ : 분류 task, 객체 탐지 task 사이 가중치를 조절하는 하이퍼파라미터
- [u ≥ 1] : u ≥ 1 인 경우 u = 1, 아니면 u = 0
- $L_{cls} = -logP_u$: 분류 점수에 대한 log loss

Multi-Task Loss

Deep Sync

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

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

- L2 Loss의 경우 너무 민감해서 학습률 조정이 어렵다.
- L1 Loss의 경우 이상치에 덜 민감하다.
- L1 Loss를 완화한 smooth L1 Loss를 사용했다.

Mini-Batch Sampling

64

64

IoU $\geq$ 0.5

0.1 $\leq$ IoU < 0.5

Positive, $u \geq 1$

Negative, $u = 0$

Review of Fast R-CNN

Deep Sync

과정

- Softmax, Bbox Regressor로 전달

- 특징맵에 RoI Projection
- RoI Pooling

- Selective Search -> RoI 추출
- RoI를 Mini-Batch로 전달

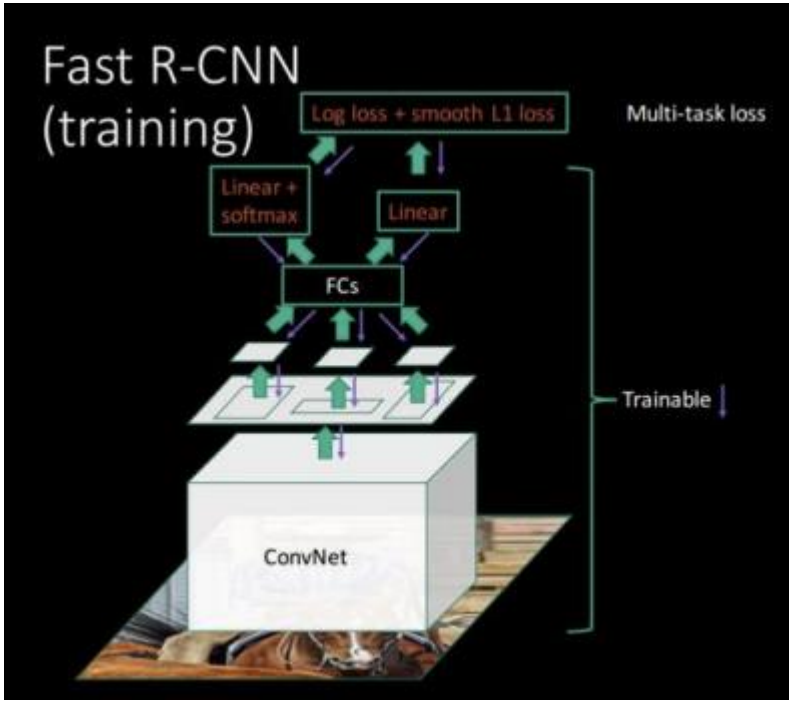


Fig7

Result

Deep Sync

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.

Fig9

Conclusion

Deep Sync

- RCNN, SPPnet 보다 더 높은 탐지 성능을 갖는다.
- Multi-Task Loss를 사용해 End-to-End 훈련을 진행할 수 있다.
- End-to-End 훈련을 통해 모든 층을 훈련할 수 있고, 더 통합적인 훈련을 할 수 있다.
- 특징을 저장(Caching)하기 위한 공간이 필요하지 않다.

Reference

Deep Sync

- Fig1 - https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852
- Fig2 - https://www.arxiv-vanity.com/papers/1908.03673/
- Fig3, Fig4, Fig6, Fig7 - https://cseweb.ucsd.edu/classes/sp17/cse252C-a/CSE252C_20170426.pdf
- Fig5 - https://arxiv.org/pdf/1406.4729.pdf
- Fig8 - https://herbwood.tistory.com/8
- Fig9 - https://arxiv.org/pdf/1504.08083.pdf

Deep Sync

Thank you for your time