Fast R-CNN 니가 그렇게 빠르냐!!

object detection 계보에서
R-CNN → SPP Net → Fast R-CNN 순서로
등장한 모델

Regional Proposal 을 제외한 detection time 이
0.3 s/img 로 상당히 빨라졌으며 기존의 3-stage training 방식을
single-stage train 이 가능하도록 한 중요한 모델.

Will Cover

1. What's improved ?
   ↳ new-concept, keywords

2. Understanding Architecture with flow
   ↳ Compared with previous models

3. Training Fast R-CNN
   ↳ multi-task loss function, Hierarchical sampling

4. Test methods
   ↳ truncated SVD

5. Fast R-CNN Limits

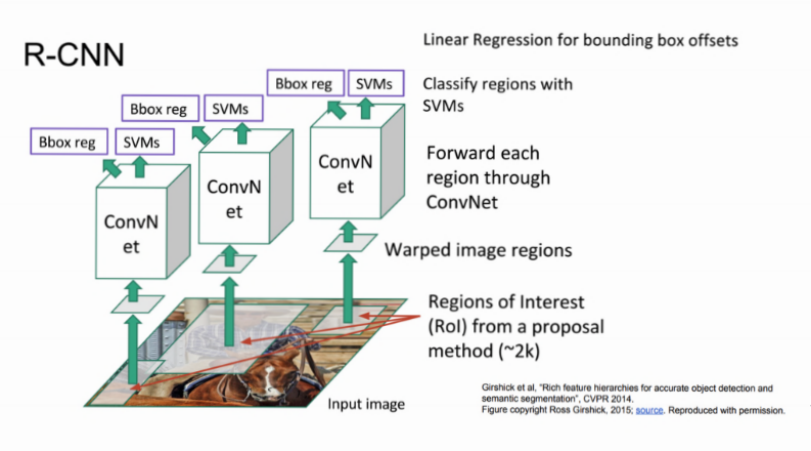
1. What's improved ?

	R-CNN	SPP Net	Fast R-CNN
multi-stage training ?	Yes	Yes	No
require storage ?	Yes	Yes	No
other drawbacks	Conv. Computation on every single R.P.	fixed Conv. layers	still using Selective-search (acc ↓, time ↑)

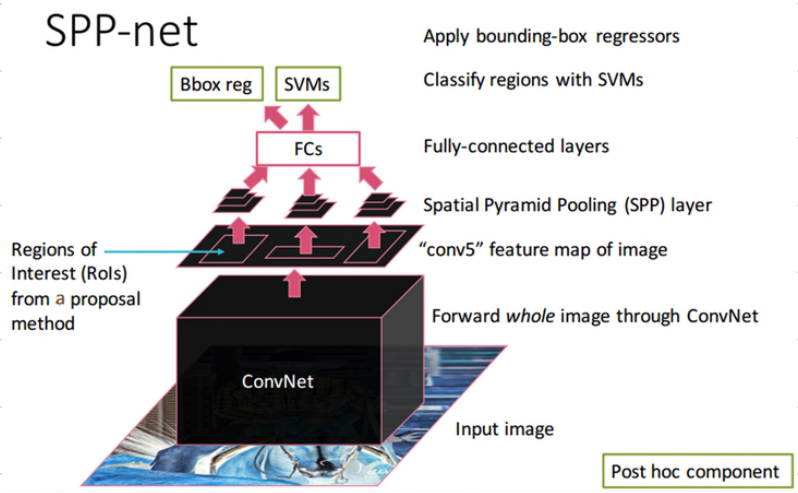
↳ end-to-end 가 가능해지며 시간/공간 이득이 상당해 졌다.

2. Understanding Architecture with flow

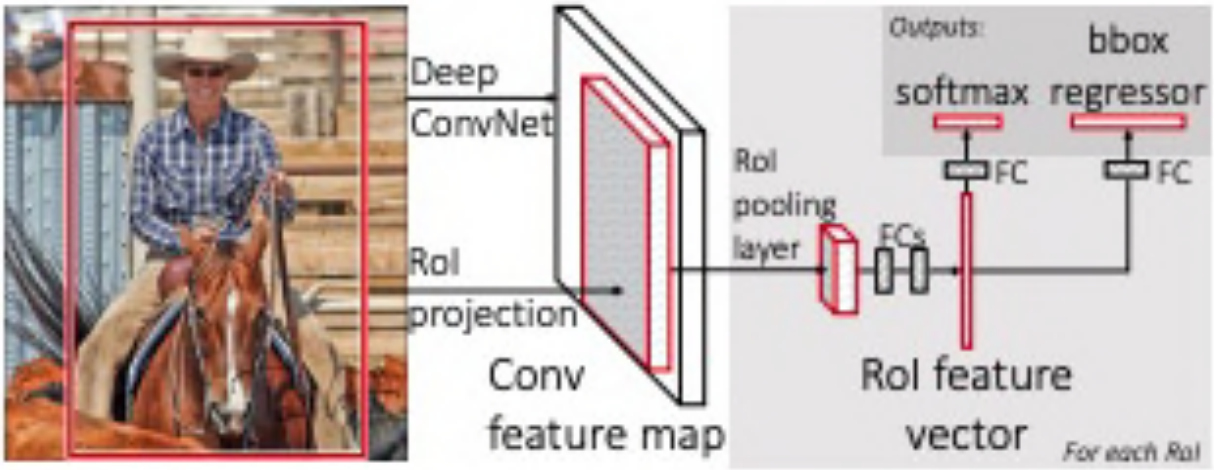
R-CNN



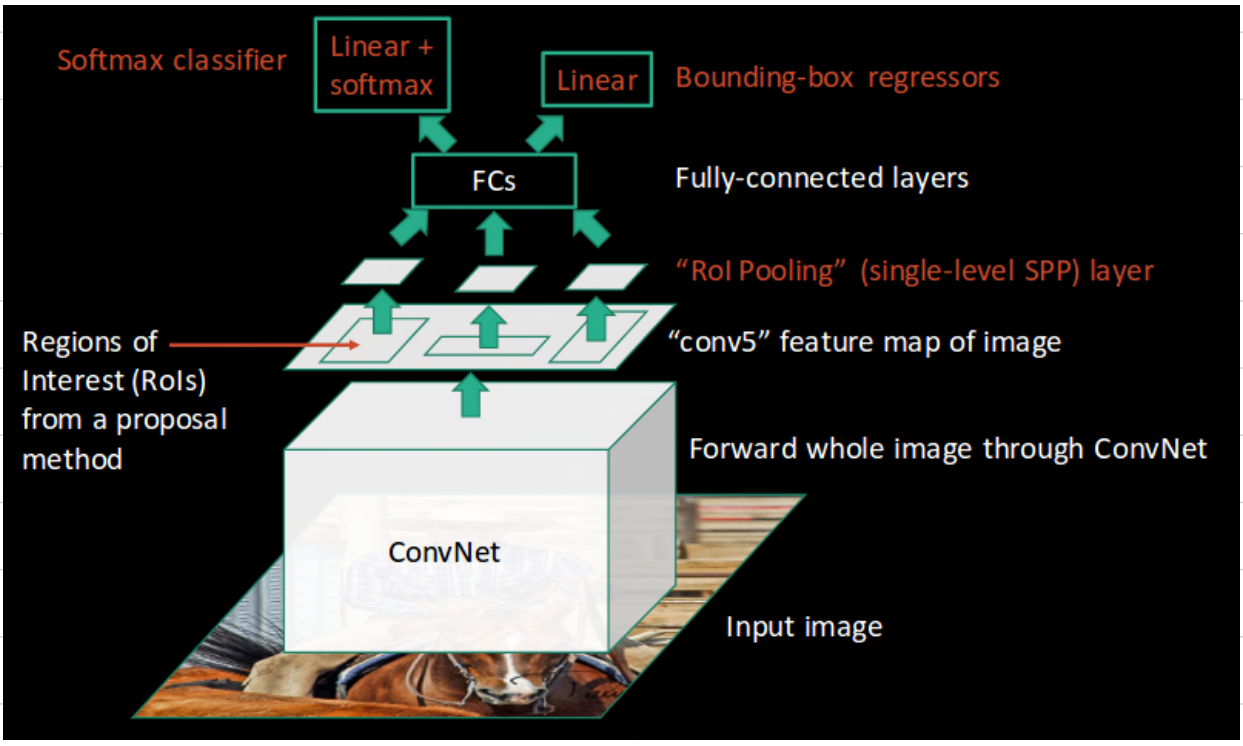
SPP Net



Fast R-CNN



[→ 방향 그림]



[↑ 방향 그림]

☆ Conv. module 변경 : Alex Net → VGG 16

Classifier 변경 : SVM → Soft max

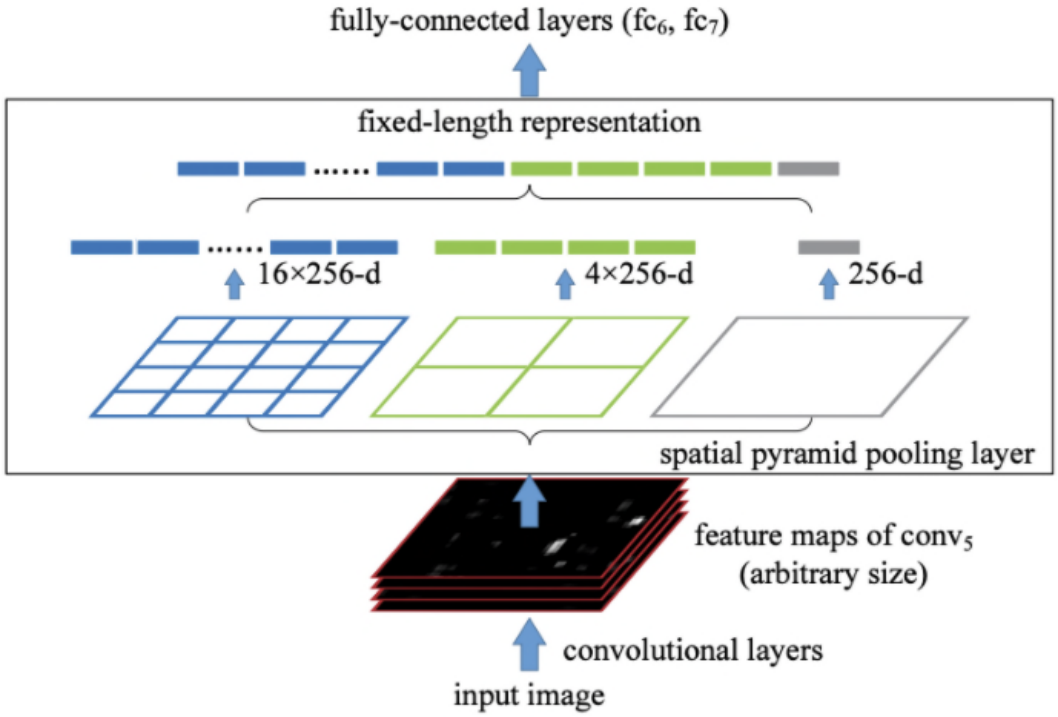
new-fixing-Pool : Spaicial-pyramid pooling → RoI pooling

RoI - Pooling ?

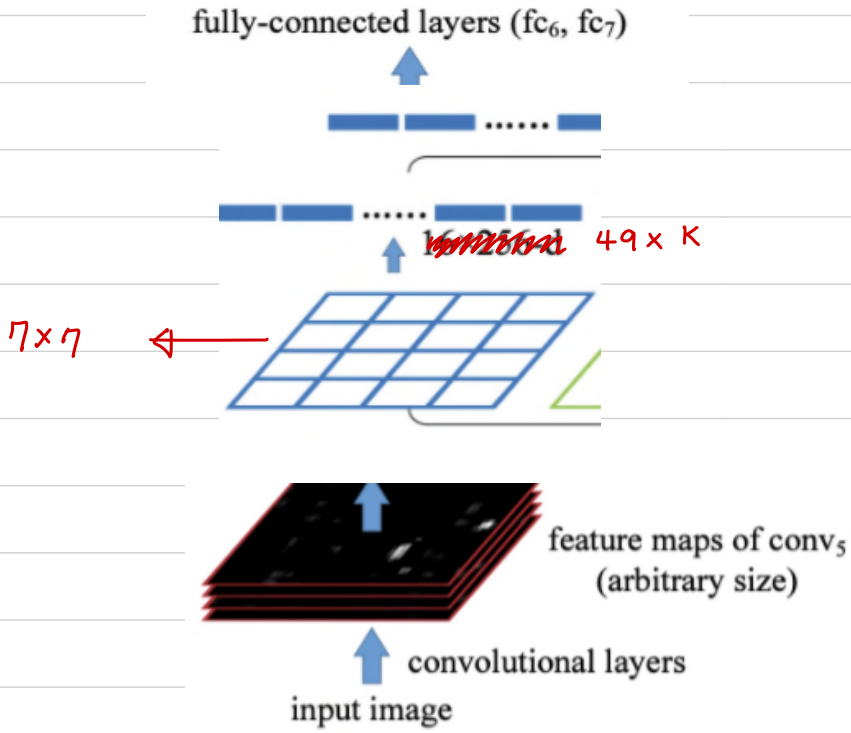
↳ one-level SPP with bin = 7x7

* 7x7 는 VGG16 기준 마지막 featuremap = 7x7xK 인것을 고려

(Spatial Pyramid Pooling)



(RoI Pooling)



3. Training R-CNN

그림을 역순으로 따라 이해

Loss ┌ classifier - fc - ROI pool layer - Conv. layer - update
     └ regressor - update

Loss : multi-task loss

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

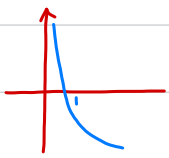
L_{cls} : classification loss

p : Softmax 출력 벡터 (K+1 size vector, each representing prob.)

u : 정답 레이블 번호 (ex. $u=4$ → 정답은 4번 객체)

$$L_{cls} = -\log p^u \quad (p^u : p \text{ 벡터에서 } u \text{ 번째})$$

* $-\log x$



, $p^u = 1$ 이면 loss = 0
$p^u \lll 1$ 이면 loss 무한

L_{loc} : localization loss

λ : 두 손실 L_{cls}, L_{loc} 사이 비중 조절을 위한 balancing parameter
* 실제로는 $\lambda=1$ 로 써서 큰 의미는 없지만 실험과정에서 비교를 위해 썼음.

$[u \geq 1]$: '1 if $u \geq 1$ else 0' 를 수행하는 로직으로 관례적으로 0번을 '배경'으로 설정한다고 함 ($u=0$)

t^u : t^u_x, t^u_y, t^u_w, t^u_h 의 묶음으로, u 번째 regressor 의 출력 offset 4개

v : v_x, v_y, v_w, v_h 의 묶음으로, 정답 오프셋 (정답 좌표 아님)

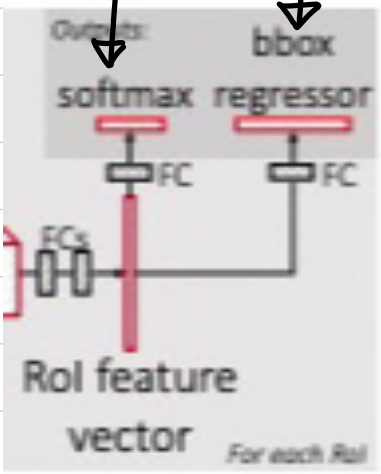
$$L_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} Smooth_{L_1}(t_i^u - v_i)$$

* $Smooth\ L_1(x) = \begin{cases} 0.5x^2 & if\ |x| < 1 \\ |x| - 0.5 & otherwise, \end{cases}$ ← 우리가 아는거 $\frac{1}{2}(y-\hat{y})^2$ (L_2 라고도 함) / ← 맨하튼 거리

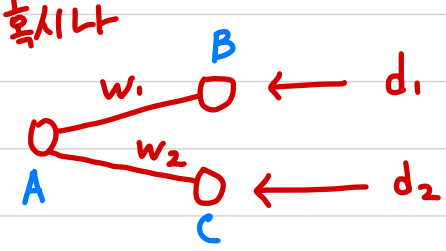
* 분류와 달리 오차의 상한선이 없기에 큰 오차에 대해 L_2 loss 적용시 기울기 폭주를 관찰하였다고 함. 때문에 $L_1 + L_2$ 섞은 손실을 씀.

이렇게 계산된 손실을 두 곳에 똑같이 전달.

$$L(p, u, t^u, v)$$



* 혹시나



$$B = w_1 \cdot A$$

$$\overline{w_1} = \overline{B} \cdot \left(\frac{dB}{dw_1}\right) = d_1 \cdot A$$

$$\overline{w_2} = \overline{C} \cdot \left(\frac{dC}{dw_2}\right) = d_2 \cdot A$$

$$\overline{A} = \overline{B} \cdot \left(\frac{dB}{dA}\right) + \overline{C} \cdot \left(\frac{dC}{dA}\right) = d_1 \cdot w_1 + d_2 \cdot w_2$$

(RoI pooling layer 전까지는 역전파 이해했다 가정하고)

RoI pooling layer 는 어떻게 넘어가지 ??

↳ 사실 RoI pooling layer 가 어떤 특별한 로직이 존재하는 것이 아니라 일반적인 CNN 모델에서 역전파시 max pooling layer 를 통과하는 로직과 완전히 동일하다.

↳ 심지어 SPP Net 도 사실 Conv. layer 학습이 아예 '불가능' 한게 아니라 아주 비효율 적이어서 '안' 했던 것이였다.

CNN 의 역전파를 빨리 확인하고 오자!!

https://ratsgo.github.io/deep%20learning/2017/04/05/CNNbackprop/

이를 이해했다면, RoI Pooling layer 는 사실 Pooling 결과가 7x7 이 되게끔 kernel size , stride 가 설정된 max pooling 층일 뿐이고 똑같은 방식의 역전파가 이루어진다는 것을 알 수 있다.

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

x_i : RoI Pooling layer 로 들어오는 featuremap 의 한 픽셀(?) 원소(?)

r : 현재 이미지의 r 번째 RoI

y_{rj} : r 번째 RoI 의 RoI pooling 결과의 j 번째 원소

$i^*(r,j)$: r 번째 RoI의 pooling 결과에서 j 번째 원소로 되게끔 max pooling 된 입력 원소의 인덱스 ;

↳ 간단 정리 : RoI pooling layer 전의 feature map 의 각 원소들에 전달되는 그래디언트는 각각의 RoI 에서 해당 원소가 가장 커서 max pool 된 y_{rj} 에서의 그래디언트들을 모두 합한 값.

+ 그 이후는 방금 확인한 커널 가중치 역전파 전달과정과 동일.

그럼 왜 SPP Net 은 Conv. layer 학습을 "포기" 했던걸까 ??

원 논문에서는 "for simplicity" 라고 표현하고 SPP Net 의 한계를 언급한
Fast R-CNN 에서는 "highly inefficient" 라고 표현 하고 있다.

구조적으로 불가능한줄 알고있었는데 사실 이론상으로는 전혀 문제가 되지 않았고
구조적 측면이 아닌 학습/검증 의 기술적 문제가 있던 것이었다.

R-CNN, SPP Net 의 학습 얘기를 할때,
IOU 임계값(0.5), 32:96 P/N 레이블, 구성을 통한 학습데이터 구축까지는
얘기 했으나 확인못한 부분이 있었다.

바로 이 128 개의 한 배치 사이즈를 이루는 (32P + 96N) ROI 정보가
한 이미지로 부터 생성한게 아니었던것 !!!!!
R-CNN, SPP Net 에서는 region-wise sampling 이란 이름의
방법으로 하나의 배치를 만들때, 128 개의 각기다른 이미지에서
1 개의 ROI 를 가져오는 방법으로 학습했던 것!

이렇게 하는게 왜 'Highly Inefficient' 하다는 걸까 ?

1. Conv. layer 가 학습중인 단계라면 어떤 한 이미지를
   넣었을때 나오는 feature map 이 계속 다르다.

2. 그렇기에 한 이미지에서 여러 ROI 를 참고하는 것이 아니라면
   128 개의 이미지를 매번 다시 conv. computation 을 통해
   바뀐 feature map 을 뽑아내야 한다.

3. Large Receptive Field 얘기는 정말정말정말 이해가 안갑니다...
   원 논문에서 SPP Net 같은 입력 고정 없이 Conv. layer 로 들어가는 모델의 경우
   그만큼 ROI도 224x224 보다는 훨씬 큰 영역일 확률이 크기에 더더욱 비효율적이라는데
   SPP Net 실제 학습때는 이미지 고정값 쓴다며 !!! Single-scale / Multi-scale 얘기는 뭔데 !!!!

암튼 그래서 Fast R-CNN은 어떻게 해결했을까?

Hierarchical sampling (image centric sampling , 계층적 샘플링?)

Fast R-CNN 에서도 마찬가지로 batch size = 128 로 설정하긴 했지만,
128 개의 이미지가 아닌, N 개의 이미지를 먼저 샘플링 후, (논문에서는 N=2)
각 이미지에서 R 개의 ROI 를 샘플링 하는 단계적(?) 샘플링으로 128 개를 만들었다고 한다.
(논문에서는 R=64)

이렇게 되면 1번 업데이트 하는데 (1 배치를 모두 forward pass 하는데)
128 번의 Conv. 연산이 아닌 N번의 Conv. 만 진행하면 되기에 훨씬 빠르고
불필요한 작업없이 학습이 가능하다!!

↳ 사실 이 계층적 샘플링 부분은 멤버들과 토의가 조금 필요해 보임.
(따지고 보면 별거 없는거 같은데 SPPNet 연구팀이 이걸 생각을 못했을까? 나만 이해를 잘못했나?)

---

4. Test method : Truncated SVD

(개인적으로 따로 정리할 예정이라 그냥 이런게 있구나 정도만 봐도 됩니다.)

test 과정에서도 더욱 빠른 검출을 위해 truncated SVD 라는
기법을 활용.

Classification 같은 경우 FC 연산에 걸리는 시간이 Conv. 연산에 비해
큰 편이 아니지만 각 이미지당 ROI 를 처리해야 하는 detection 문제의
경우 전체 소요시간의 46.3% 가 Conv., 38.7% 가 FC에 걸릴만큼
연산 시간이 늘어 남.

이에 조금더 파라미터가 적은 FC 를 통해 동일 수준의 성능을
얻기 위해 행렬 축소 기술을 사용함 (test 에만 쓰이는 것 명심!)

$u \times v$ 사이즈 행렬 W 에 대하여

$W \simeq U \Sigma V^T$ 로 특이 행렬 분해가 가능함을

활용하여 W 사이즈 FC 1층을 ΣV^T, U 사이즈 FC 2층으로 대체 하는 것.

Parameters

$u \times v \gg t(u+v)$

5. Fast R-CNN Limits

- 사실 하나 밖에 없다! 이전 모델들의 한계점을 end-to-end 로 싹 해결해버린 훌륭한 모델
- 아직까지 Regional Proposal 영역은 Selective-search 를 쓰기 때문에 검출에 병목현상이 발생한다.

  Total detection 2.3s / img
  2.3s (2s) + actual detection (0.3s) → Faster R-CNN 에서 해결!!

- Single-stage training 이긴 하나, Single-stage detector 는 아니다.
  아직 진정한 real-time detection 까지는 갈 길이 멀다!!