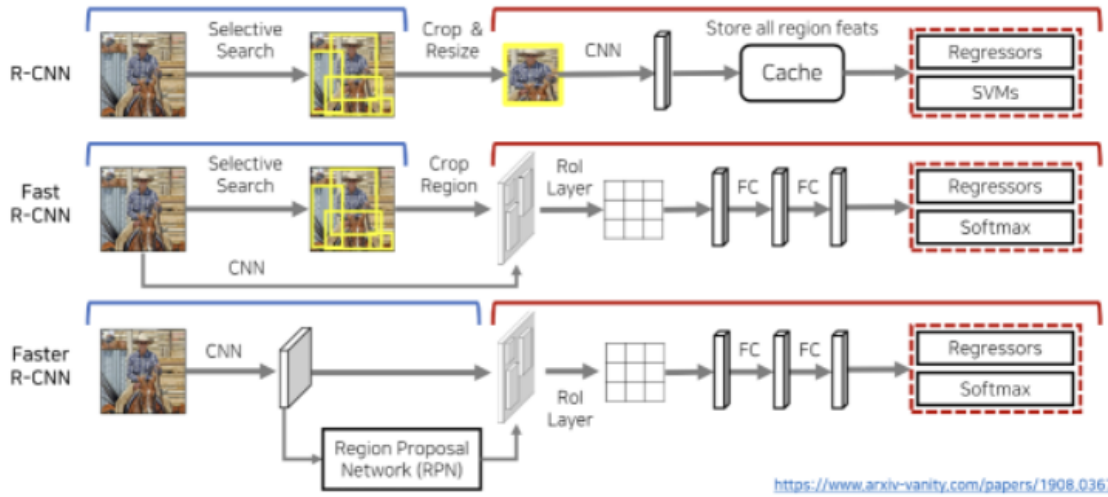
Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

R-CNN & Fast R-CNN



R-CNN

- 1차적으로 cpu 상에서 Selective Search 진행 → 물체가 있을 법한 위치 약 2000개 정도 찾음
- Cropping & Resizing을 수행한 뒤, 개별적으로 CNN network 거치면서 feature vector 추출
- 추출한 feature vector는 SVM을 통해 Classification 진행 + Regressor를 통해 bounding box 예측

Fast R-CNN

- R-CNN보다 좀 더 빠른 성능
- 기존의 R-CNN과 마찬가지로 Selective Search를 통해 Region Proposal
- CNN을 거쳐 Feature Vector 추출

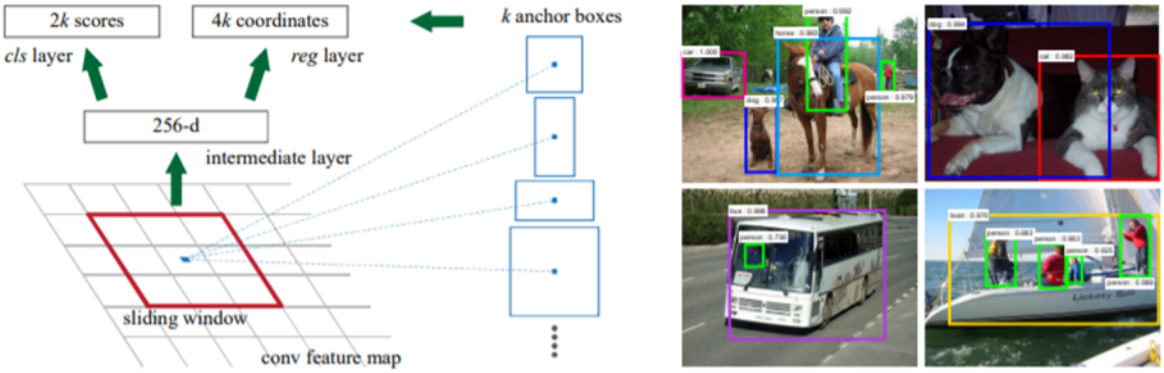
- 기존의 RCNN은 Selective Search된 이미지 모두 CNN을 통과 but! Fast RCNN은 단 한 번만 거침 ⇒ 더 빠른 성능
- ROI pooling을 통해 각각의 Region에 대해서 Feature에 대한 정보 추출
- **Softmax layer**를 거쳐서 각각의 class에 대한 확률 값을 구하고 이를 이용해 classification!

Faster R-CNN

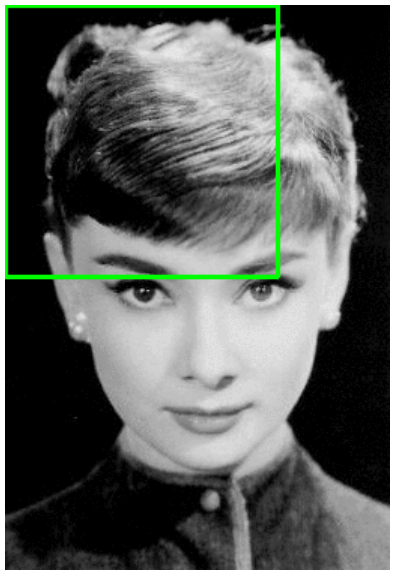
- 두 개의 모듈로 구성 → 전체 시스템은 OD를 위한 단일 통합 네트워크
  - Deep Fully Convolutional Network - 영역 제안
  - Fast R-CNN detector - 제안된 영역 사용
- RPN(Region Proposal Networks) - Fast R-CNN 모듈에게 찾아야 할 곳을 제안

RPN(Region Proposal Networks)

- 이미지를 입력으로 사용하고, 사각형태의 **object proposal**과 **objectness score**의 세트 출력
- fully convolutional network를 사용하여 모델링
- 목표 : Fast R-CNN과 계산을 공유하는 것 → 두 네트가 공통의 convolution layer를 공유한다고 가정
- 마지막 공유 conv layer로부터 출력된 convolution feature map 위로 **작은 네트워크** 추가
  - n x n의 spatial window → sliding window (저차원 feature로 매핑) ⇒ 본 논문에서는 n=3으로 설정
  - 매핑된 feature는 2개의 fully-connected layers(box-regression(reg)/box-classification(cls) layers)에 공급
  - reg와 cls는 1x1 conv layer로 구현 ⇒ 왼쪽 그림 참고



Anchors



Sliding Window 예시

- 각 sliding window의 위치에서 여러 region proposals를 동시에 예측하며, 각 위치에 대하여 가능한 **최대 proposals 수**가 표시됨 → **k**
- **reg layer**는 상자의 좌표를 인코딩하므로 **4k**의 출력을 가지고, **cls layer**는 각 proposal에 대해 object일 확률/object 아닐 확률을 추정하는 **2k** 출력
- k proposal은 k reference box에 대해 매개 변수화됨 ⇒ **anchor(앵커)**
- 앵커는 해당 슬라이딩 윈도우의 중앙에 위치하며, 스케일 및 가로•세로 비율과 관련
- 기본적으로 3개의 스케일과 3개의 가로•세로비를 사용하며, 각 슬라이딩 위치에서 k=9개의 앵커 박스를 산출
- 크기가 W x H의 conv feature map의 경우에는 총 W Hk 앵커가 있음.

Translation-Invariant Anchors

- **본 논문의 접근법** : 앵커와 앵커에 관련된 proposal을 계산하는 함수 모두 translation-invariant함
- 객체를 변환하는 경우, proposal도 변환해야 하며, 두 위치에서 모두 동일한 기능으로 proposal을 예측해야 함
- MultiBox 방법은 k-means를 사용하여 800개의 앵커 생성 → translation-invariant X

  ⇒ 객체를 번역할 때 동일한 제안이 생성됨을 보장 X
- translation-invariant 속성은 모델 크기 줄임
  - MultiBox - (4+1)*800 차원의 fully connected output layer. // 출력층 매개변수 : $6.1 * 10^6$개
  - **our Method** - (4+2)*9 차원의 conv ouput layer (k=9일 때). // 출력층 매개변수 : $2.8 * 10^4$개

    ⇒ MultiBox의 출력층 매개변수 크기의 2배 작은 크기를 가짐.
- feature projection layer를 고려하면, 본 논문의 proposal layer는 MultiBox보다 훨씬 작은 매개변수를 갖고 있음 ⇒ **소규모 데이터셋에서 오버피팅 위험 적음**

Multi-Scale Anchors as Regression References

- Multi-Scale Prediction을 위한 두가지 방법
  - image/feature pyramids에 기반
    - 여러 척도로 image의 크기 조정한 후, 피쳐맵을 각 척도에 대해 계산
    - 시간이 오래 걸린다는 단점
  - feature map에서 multiple scale의 슬라이딩 window 사용
    - 일반적으로 위의 방법과 공동으로 채택됨
    - **DPM** - 가로 세로 비율이 서로 다른 모델 → 서로 다른 필터 크기를 사용하여 별도로 학습
- 앵커 기반 방식
  - 앵커 피라미드 위에 구축되어 있으므로, 비용적으로 효율적
  - 다중 척도 및 가로 세로 비율의 앵커 박스 기준으로 bounding box를 분류/회귀
  - 단일 스케일의 이미지와 피쳐맵에 의존하고 단일 사이즈의 필터 사용
  - 앵커 기반의 multi-scale 설계로 인해, 단일 스케일 이미지에서 계산되는 컨볼루션 기능을 간단히 사용 가능 → Fast R-CNN detector에서의 수행과 같음

- scale 문제를 해결하는데 추가 비용 없이 feature를 공유할 수 있음

Loss Function

- training RPN의 경우, 각 앵커에 객체인지 아닌지에 대한 binary class label을 할당
- 두 종류의 앵커에 positive label을 지정
  - ground-truth box와 겹치는 가장 높은 IoU를 갖는 앵커
  - 모든 ground-truth box와 0.7이상 겹치는 앵커

  🗨 (groud-truth VS label) → label은 정해진 정답이지만, ground-truth는 우리가 원하는 정답임
- 두번째 조건은 positive sample을 확인하는데 충분하지만, 가끔 두번째 조건에서 positive sample을 발견하지 못하는 상황 발생

  ⇒ 첫번째 조건 채택!
- IoU 비율이 모든 ground-truth box에 대해 0.3보다 작은 경우에는 negative label 할당
- positive/negative 에 포함되지 않는 앵커는 training objective에서 제외

- 손실 함수

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

  - **p*i Lreg** = p*i가 1인 경우(positive anchor)에만 활성화.

    p*i가 0인 경우(negative anchor)에는 비활성화

- **pi**: Predicted probability of anchor
- **pi***: Ground-truth label (1: anchor is positive, 0: anchor is negative)

- **lambda**: Balancing parameter. Ncls와 Nreg 차이로 발생하는 불균형을 방지하기 위해 사용.

  cls에 대한 mini-batch의 크기가 256(=Ncls)이고, 이미지 내부에서 사용된 모든 anchor의 location이 약 2,400(=Nreg)라 하면 lamda 값은 10 정도로 설정함

- **ti**: Predicted Bounding box
- **ti***: Ground-truth box
- Bounding box regression(Lreg) 과정에서의 손실함수

$$
\begin{aligned}
t_{\mathrm{x}} &= (x - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t_{\mathrm{y}} = (y - y_{\mathrm{a}})/h_{\mathrm{a}}, \\
t_{\mathrm{w}} &= \log(w/w_{\mathrm{a}}), \quad t_{\mathrm{h}} = \log(h/h_{\mathrm{a}}), \\
t^*_{\mathrm{x}} &= (x^* - x_{\mathrm{a}})/w_{\mathrm{a}}, \quad t^*_{\mathrm{y}} = (y^* - y_{\mathrm{a}})/h_{\mathrm{a}}, \\
t^*_{\mathrm{w}} &= \log(w^*/w_{\mathrm{a}}), \quad t^*_{\mathrm{h}} = \log(h^*/h_{\mathrm{a}}),
\end{aligned}
\tag{2}
$$

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

in which

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

- 4개의 좌표값을 사용 → t = 4개 좌표값을 가진 하나의 벡터
- 위의 계산을 취한 후, Smooth L1 loss function을 통해 loss 계산
- R-CNN / Fast R-CNN에서는 모든 Region of Interest가 그 크기와 비율에 상관없이 weight를 공유했던 것에 비해, 이 anchor 방식에서는 k개의 anchor에 상응하는 k개의 regressor를 갖게 됨

Training RPNs

- end-to-end로 back-propagation 사용
- Stochastic gradient descent(SGD) 사용
- 한 이미지당 랜덤하게 256개의 sample anchor들을 사용 → Sample은 positive anchor:negative anchor = 1:1 비율로
  - 혹시 positive anchor의 개수가 128개보다 낮을 경우, 빈 자리는 negative sample로 채움 ⇒ 이미지 내에 negative sample이 positive sample보다 훨씬 많으므로

- 모든 weight는 랜덤하게 초기화 ⇒ μ=0; σ=0.01인 가우스 분포를 따름
- ImageNet classification으로 fine-tuning
- Learning Rate :
  - 처음 60k의 mini-batches : 0.001
  - 다음 20k의 mini-batches : 0.0001
- Momentum : 0.9
- Weight Decay : 0.0005

Sharing Features for RPN and Fast R-CNN

- 3가지 방법 존재
  - i) **Alternating training** ⇒ 본 논문에서는 Alternating training을 채택하여 사용
  - ii) Approximate joint training
  - iii) Non-approximate joint training
- 4-step Alternating training
  1. Train RPNs → RPN 훈련\\ 이미지넷으로 사전 훈련된 모델로 초기화하고 영역 추정 작업을 위해 end-to-end 파인 튜닝
  2. Train Fast R-CNN using the proposals from RPNs → 1에서 생성한 영역 추정 bounding box를 활용하여 독립적인 Fast R-CNN 훈련
  3. Fix the shared convolutional layers and fine-tune unique layers to RPN →RPN 훈련 초기화를 위해 Fast R-CNN 사용\\ 공유된 합성곱 계층은 고정 & RPN만 파인튜닝
  4. Fine-tune unique layers to Fast R-CNN

  ⇒ RPN과 Fast R-CNN은 같은 합성곱 계층 공유 + 단일 네트워크 구성

Implementation Details

- 앵커
  - 박스 면적이 1282, 2562, 5122 픽셀을 갖는 3개의 스케일과 1:1, 1:2, 2:1의 3개의 크기 비율을 사용
  - 이미지 경계를 교차하는 앵커 박스는 주의해서 처리해야 함

→ 훈련할 때 boundary-crossing outliers를 무시하도록 처리함 or 훈련 속도 느리고 성능 낮음

- RPN proposal의 overlap 발생
  - NMS(Non-maximum suppression; 비최댓값 억제)의 IoU 임계값을 0.7로 수정
    → 이미지당 2000개의 proposal regions
  - ⇒ 제안 횟수 줄여줌 → 속도 향상 및 성능 유지

Experiments

Experiments on PASCAL VOC

train-time region proposals		test-time region proposals		
method	# boxes	method	# proposals	mAP (%)
SS	2000	SS	2000	58.7
EB	2000	EB	2000	58.6
RPN+ZF, shared	2000	RPN+ZF, shared	300	**59.9**
ablation experiments follow below				
RPN+ZF, unshared	2000	RPN+ZF, unshared	300	58.7
SS	2000	RPN+ZF	100	55.1
SS	2000	RPN+ZF	300	56.8
SS	2000	RPN+ZF	1000	56.3
SS	2000	RPN+ZF (no NMS)	6000	55.2
SS	2000	RPN+ZF (no *cls*)	100	44.6
SS	2000	RPN+ZF (no *cls*)	300	51.4
SS	2000	RPN+ZF (no *cls*)	1000	55.8
SS	2000	RPN+ZF (no *reg*)	300	52.1
SS	2000	RPN+ZF (no *reg*)	1000	51.3
SS	2000	RPN+VGG	300	59.2

- 객체 탐지 벤치마크 데이터셋인 PASCAL VOC 2007(훈련/검증 이미지 5,000개와 테스트 이미지 5,000개, 20 classes)에서 Faster R-CNN의 성능을 평가
- 베이스라인 모델로 ZF-net 활용
- Faster R-CNN은 피처를 공유하고, 테스트 단계에서 bounding box 개수도 적기 때문에 Selective Search(SS)나 EdgeBoxes(EB)에 비해 더 빠름 → **mAP 59.9%**

Ablation Experiments on RPN

- 위 그림 참고(Experiments on PASCAL VOC)

Ablation Experiments란 요소를 하나씩 없애면서 해당 요소가 전체 시스템에 어떤 영향을 주는지 확인하는 분석 기법을 뜻함

- RPN과 Fast R-CNN이 합성곱 계층을 공유하는 것의 효과에 대해 먼저 살펴 봄
  - 4 step Alternating Training에서 2단계까지 진행한 결과들과 비교

    → 공유하지 않은 결과보다 공유한 결과(mAP-59.9%)의 성능이 더 높음

    ⇒ 3단계에서 RPN을 파인 튜닝하므로 성능이 좋은 영역 추정 bounding box를 출력했기 때문
- RPN의 효과에 대해 살펴봄
  - 훈련 단계에서 RPN을 selective search한 후, 테스트 단계에서는 RPN을 사용 ⇒ mAP : 56.8%
- cls와 reg의 영향
  - cls가 없는 경우 ⇒ 성능 bad
  - reg가 없는 경우 ⇒ 성능 bad
- ZF-Net 대신 VGG 사용 ⇒ 동일한 조건을 갖고 ZF-Net을 사용한 Experiment와 비교했을 때 성능 good!(**mAP : 59.2%**)

Performance of VGG-16

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**

- ZF-Net 대신 VGG를 사용
- data: 07 - PASCAL VOC 2007 \\ 07+12 - PASCAL VOC 2007&2012 \\ MS COCO

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**

- VGG-16을 활용한 네트워크가 ZF-net에 비해 복잡하다보니 성능은 좋은 반면 속도는 5 fps로 느린 편

Sensitive to Hyper-Parameters

- 앵커박스 설정에 따른 mAP 차이를 보여줌 (PASCAL VOC 2007 사용)

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**

  - 스케일과 비율이 각각 3개씩 있을 때 성능이 가장 좋은 걸 알 수 있음

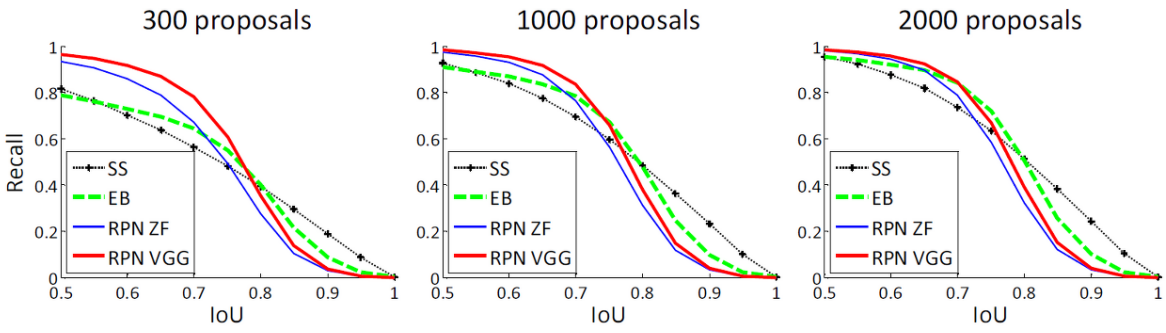
- λ에 따른 mAP 차이

λ	0.1	1	10	100
mAP (%)	67.2	68.9	69.9	69.1

  - 상대적으로 덜 민감한 파라미터(차이 1%미만)

Analysis of Recall-to-IoU

- IoU 값에 따른 영역 추정 경계 박스의 재현율(recall)을 계산
- 300개, 1,000개, 2,000개의 영역 추정 경계 박스에 따른 재현율과 IoU 관계를 나타냄



- 300일 때 최적!

- SS와 EB의 recall은 proposals이 적을 때 RPN보다 더 빠르게 감소

One-Stage Detection vs Two-Stage Proposal + Detection

- One-Stage Detection과 Two-Stage Proposal + Detection의 성능 차이
- 물체의 위치를 찾는 문제(localization)과 분류(classification) 문제를 **한 번에\\순차적으로 해결**하는 방식
- OverFeat을 모방해 one-stage Fast R-CNN을 만든 후 성능 비교 ⇒ Two-Stage가 더 좋음

	proposals		detector	mAP (%)
Two-Stage	RPN + ZF, unshared	300	Fast R-CNN + ZF, 1 scale	58.7
One-Stage	dense, 3 scales, 3 aspect ratios	20000	Fast R-CNN + ZF, 1 scale	53.8
One-Stage	dense, 3 scales, 3 aspect ratios	20000	Fast R-CNN + ZF, 5 scales	53.9

Experiments on MS COCO

- Faster R-CNN이 Fast R-CNN보다 더 높은 성능
- 모델은 VGG-16을 사용

			COCO val		COCO test-dev	
method	proposals	training data	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**

Conclusion

- 본 논문에서는 빠르고 정확한 영역 추정을 하기 위해 RPN을 제안함
- RPN과 객체 탐지기가 합성곱 피처를 공유함으로써 영역 추정 비용을 크게 줄임
- 실시간 객체 탐지가 가능할 정도로 빠르고, 전체적인 정확도도 기존 모델들보다 뛰어남!