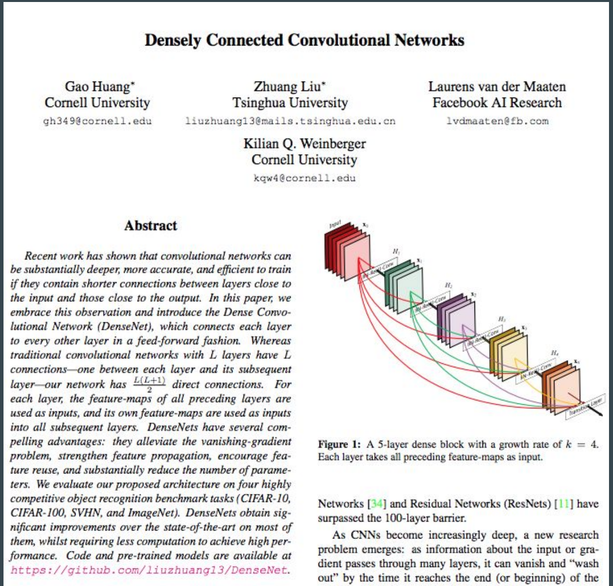
BOAZ Deep Learning Paper

분석 B조
김아영 남궁찬
성민석 류승룡

Index



Densely Connected Convolutional Networks

Gao Huang* Cornell University gh349@cornell.edu
Zhuang Liu* Tsinghua University liuzhuang13@mails.tsinghua.edu.cn
Laurens van der Maaten Facebook AI Research lvdmaaten@fb.com
Kilian Q. Weinberger Cornell University kqw4@cornell.edu

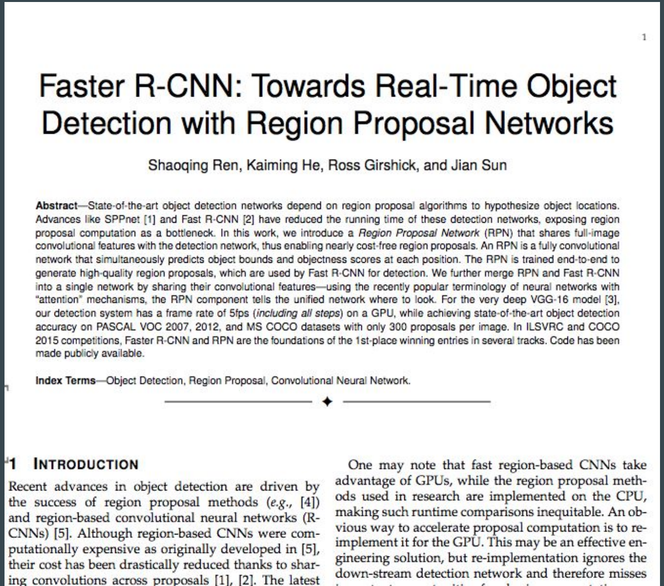
Abstract

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance. Code and pre-trained models are available at `https://github.com/liuzhuang13/DenseNet`.

Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Networks [34] and Residual Networks (ResNets) [11] have surpassed the 100-layer barrier.

As CNNs become increasingly deep, a new research problem emerges: as information about the input or gradient passes through many layers, it can vanish and "wash out" by the time it reaches the end (or beginning) of the



1

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

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

Abstract—State-of-the-art object detection networks depend on region proposal algorithms to hypothesize object locations. Advances like SPPnet [1] and Fast R-CNN [2] have reduced the running time of these detection networks, exposing region proposal computation as a bottleneck. In this work, we introduce a *Region Proposal Network* (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. We further merge RPN and Fast R-CNN into a single network by sharing their convolutional features—using the recently popular terminology of neural networks with "attention" mechanisms, the RPN component tells the unified network where to look. For the very deep VGG-16 model [3], our detection system has a frame rate of 5fps (*including all steps*) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks. Code has been made publicly available.

Index Terms—Object Detection, Region Proposal, Convolutional Neural Network.

1 INTRODUCTION

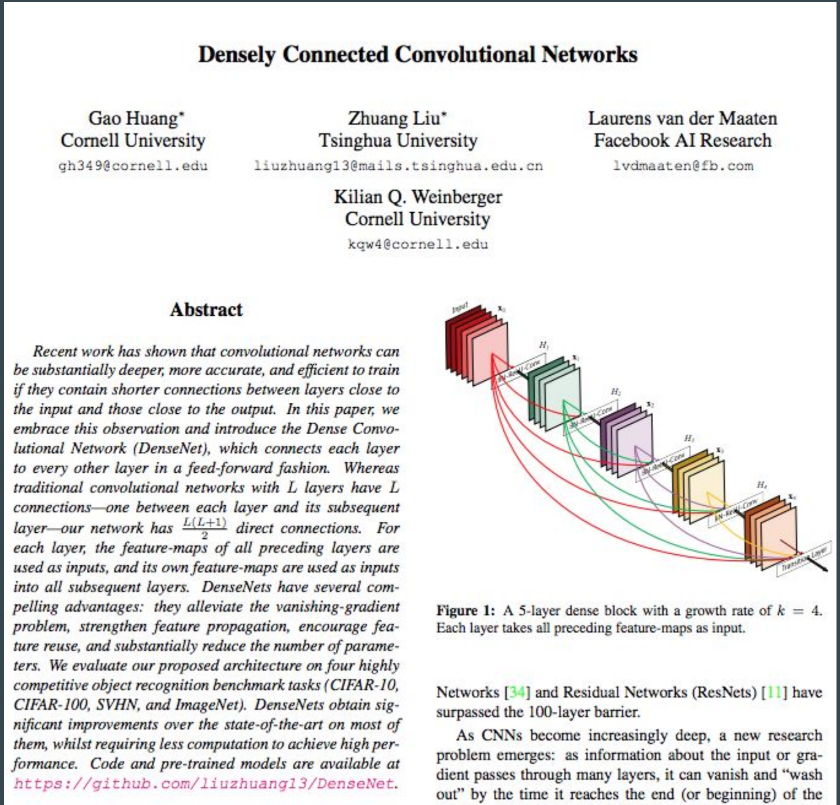
Recent advances in object detection are driven by the success of region proposal methods (*e.g.*, [4]) and region-based convolutional neural networks (R-CNNs) [5]. Although region-based CNNs were computationally expensive as originally developed in [5], their cost has been drastically reduced thanks to sharing convolutions across proposals [1], [2]. The latest

One may note that fast region-based CNNs take advantage of GPUs, while the region proposal methods used in research are implemented on the CPU, making such runtime comparisons inequitable. An obvious way to accelerate proposal computation is to re-implement it for the GPU. This may be an effective engineering solution, but re-implementation ignores the down-stream detection network and therefore misses

Densely Connected Convolutional Networks

DenseNet

0. Abstract
1. Introduction
2. Related Work
3. DenseNets
4. Experiments
5. Discussion
6. Conclusion



Densely Connected Convolutional Networks

Gao Huang* Cornell University gh349@cornell.edu

Zhuang Liu* Tsinghua University liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten Facebook AI Research lvdmaaten@fb.com

Kilian Q. Weinberger Cornell University kqw4@cornell.edu

Abstract

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance. Code and pre-trained models are available at https://github.com/liuzhuang13/DenseNet.

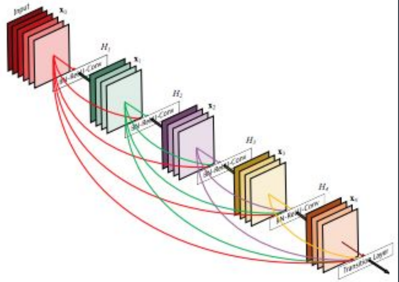


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Networks [34] and Residual Networks (ResNets) [11] have surpassed the 100-layer barrier.

As CNNs become increasingly deep, a new research problem emerges: as information about the input or gradient passes through many layers, it can vanish and "wash out" by the time it reaches the end (or beginning) of the

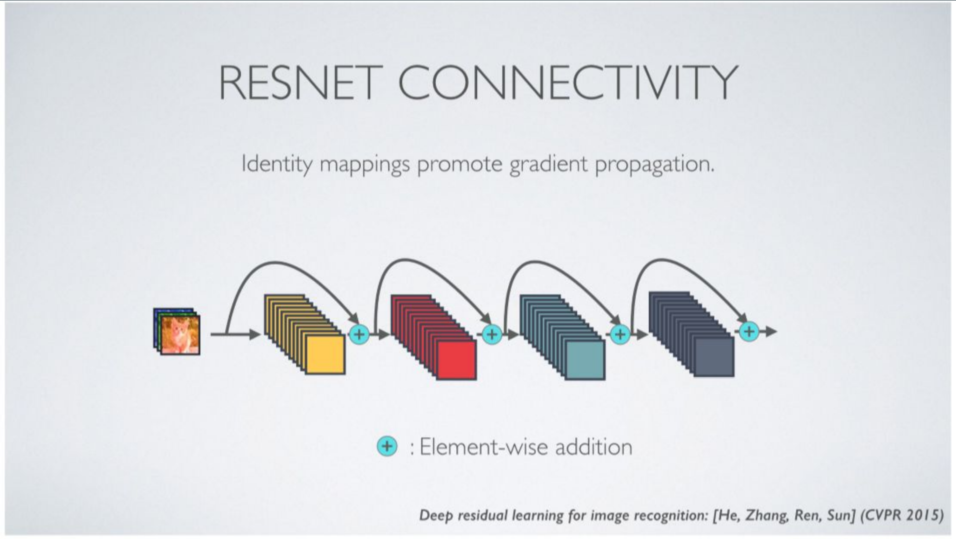
1. INTRODUCTION

- CNN은 깊을수록 더 정확하다.
  - 깊어지면서 input과 gradient 소실 문제가 생긴다.
- 해결책
  - ResNets, Highway Networks는 신호를 흘려보낸다(by-pass)
  - Stochastic depth는 ResNet의 layer를 랜덤하게 drop
  - FractalNet
    - 모두 경로를 짧게 하는 접근법들
- DenseNet
  - 모든 layer를 연결한다
    - 각 layer는 모든 이전 계층으로부터 입력을 얻는다
    - 결과 feature-map을 모든 후속 레이어로 전달한다

1. INTRODUCTION

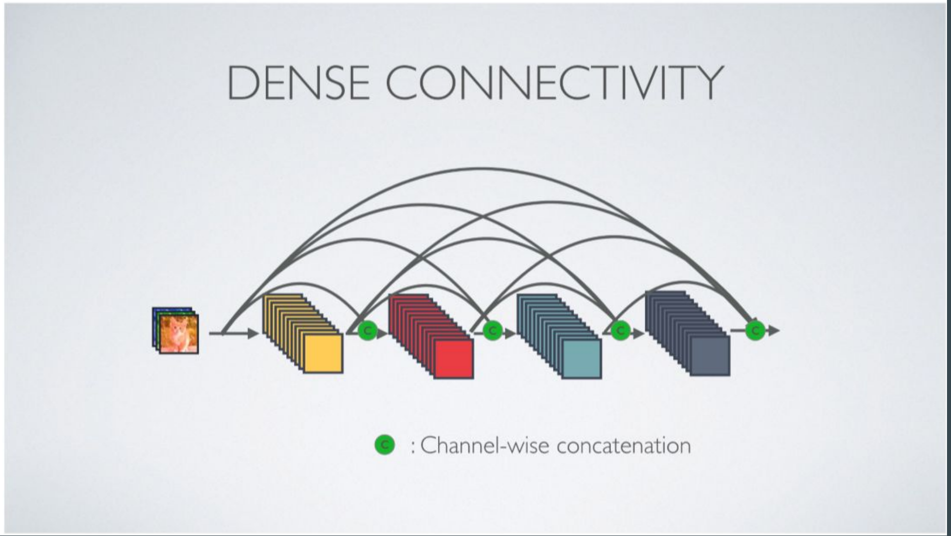
- ResNet과의 비교
  - feature를 summation 결합하지 않는다
  - 대신, concat하여 결합
    - *l* 번째 레이어는 *l*개의 input
    - 결과 feature-map은 L-*l* 개의 후속 레이어로 전달
    - L-layer 네트워크에서 총 L(L+1)/2 개의 연결을 가진다.

1. INTRODUCTION


RESNET CONNECTIVITY

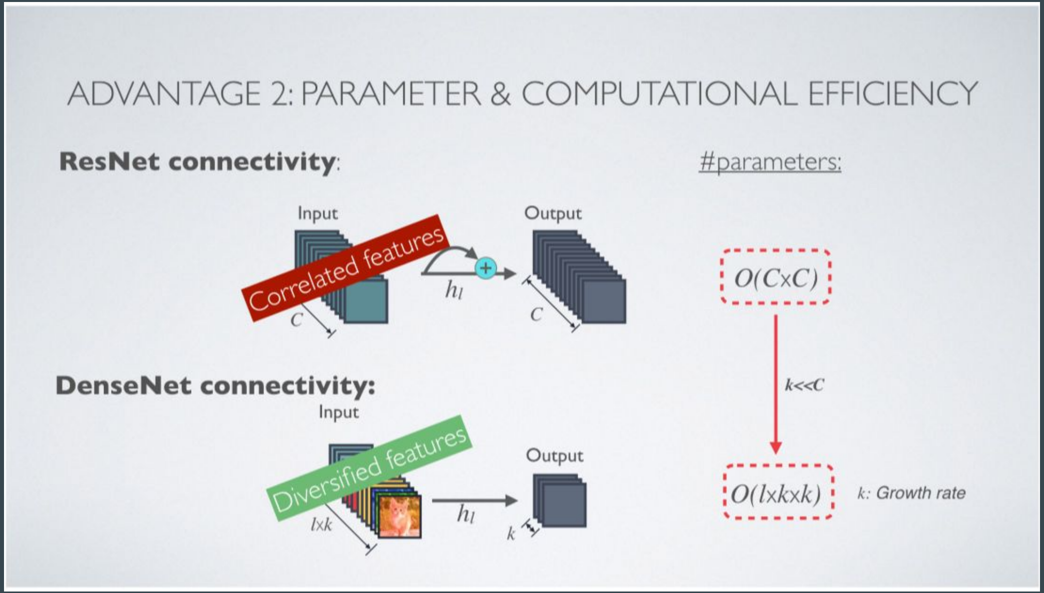
Identity mappings promote gradient propagation.

Deep residual learning for image recognition: [He, Zhang, Ren, Sun] (CVPR 2015)


DENSE CONNECTIVITY

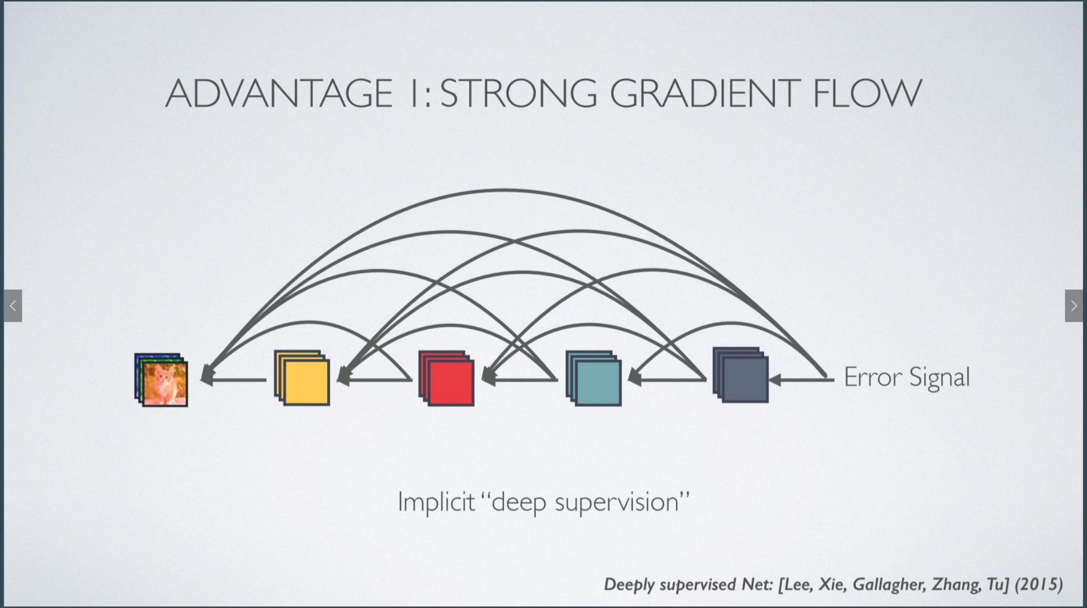
1. INTRODUCTION

- Parameter efficiency
  - 일반적인 feed-foward나 ResNet은 각 레이어가 고유의 weight를 가지기 때문에 점점 커진다.
  - DenseNet은 중복된 featrue-map을 재학습할 필요가 없음



1. INTRODUCTION

- Gradient Vanishing Problem 해결
  - 각 계층은 loss function과 input signal로부터 나온 gradient에 직접 접근한다.
  - 심층 네트워크 구조의 훈련이 쉬워진다.


ADVANTAGE 1: STRONG GRADIENT FLOW
Error Signal
Implicit "deep supervision"
Deeply supervised Net: [Lee, Xie, Gallagher, Zhang, Tu] (2015)

1. INTRODUCTION

- DenseNet 특징
  - 네트워크에 추가되는 정보와 보존되는 정보를 명시적으로 구별
  - layer가 매우 좁다. e.g., layer당 12개의 필터
  - 네트워크의 '집단적 지식'에 작은 집합의 feature-map만 추가한다
  - 마지막 분류기는 네트워크의 모든 feature-map에 기반하여 결정한다.
  - parameter 효율이 좋다
  - regularizing 효과가 있다. <- 설명 부족
    - 작은 데이터셋의 오버피팅을 줄인다.

2. RELATED WORK

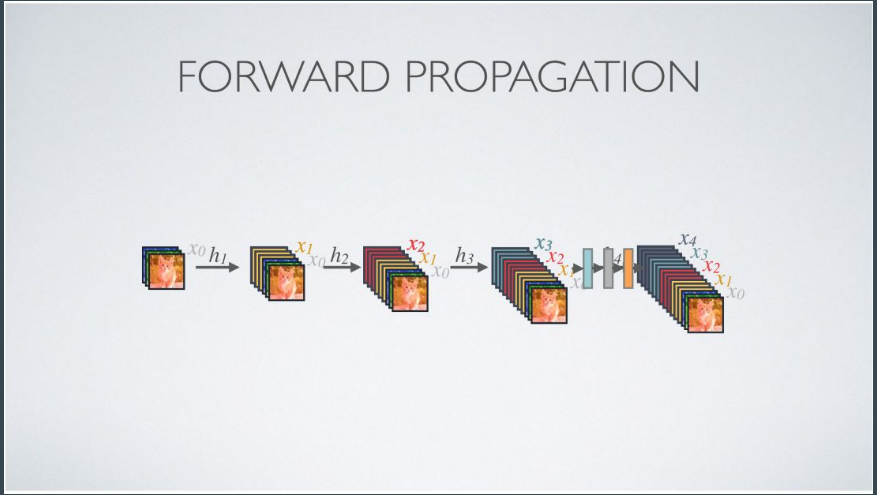
- fully-connected cascade network : parameter 개수의 한계
- skip-connections를 통한 다층 feature를 활용하는 것이 효과적이다.
    - Highway Networks
    - ResNet
        - Stochastic depth : 무작위로 layer를 drop하여 훈련을 개선
          -> 모든 layer 필요없다. 많은 중복성을 가지고 있다.
- 네트워크 너비의 증가
    - GoogLeNet : Inception module
    - ResNet의 변형
    - FractalNets
    - 사실, 깊이가 충분하면 각 레이어의 필터 수만 증가시켜도 성능 향상된다.

2. RELATED WORK

- DenseNet은!
  - 깊거나 넓은 아키텍처에서 표현력을 끌어내는 대신
  - feature-reuse 하여 학습하기 쉽고 매개 변수가 없는 모델을 산출합니다.
- Other Architecture
  - The Network in Network(NIN)
  - Deeply Supervised Network(DSN)
  - Ladder Networks
  - Deeply-Fused Nets(DFN)

3. DENSENETS : Connectivity

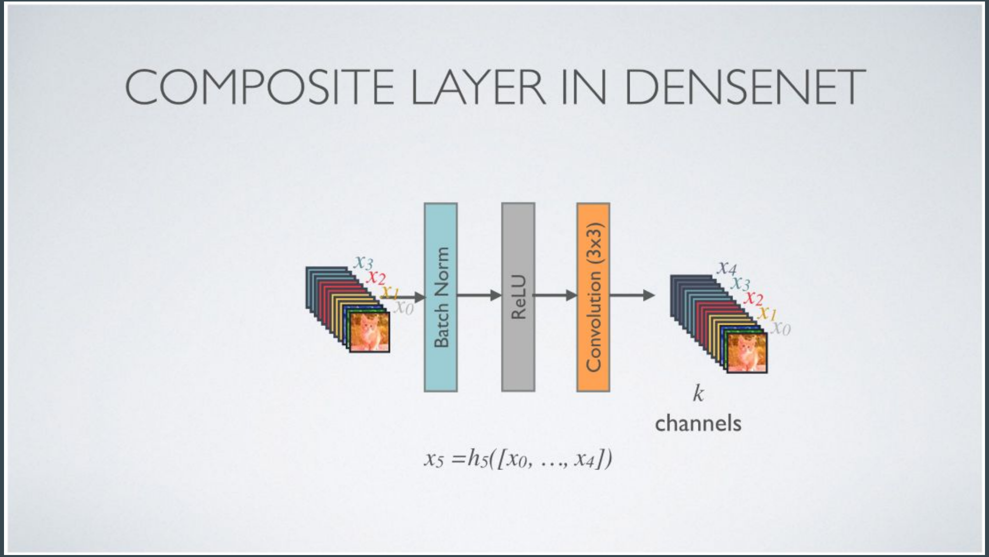
- 일반적인 feed-foward network : $\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1})$
- skip-connection : $\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}.$
    - summation이 정보의 흐름을 방해한다.
- dense-connection : $\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}])$
    - 다중 입력을 하나의 텐서로 concat



3. DENSENETS : Composite function

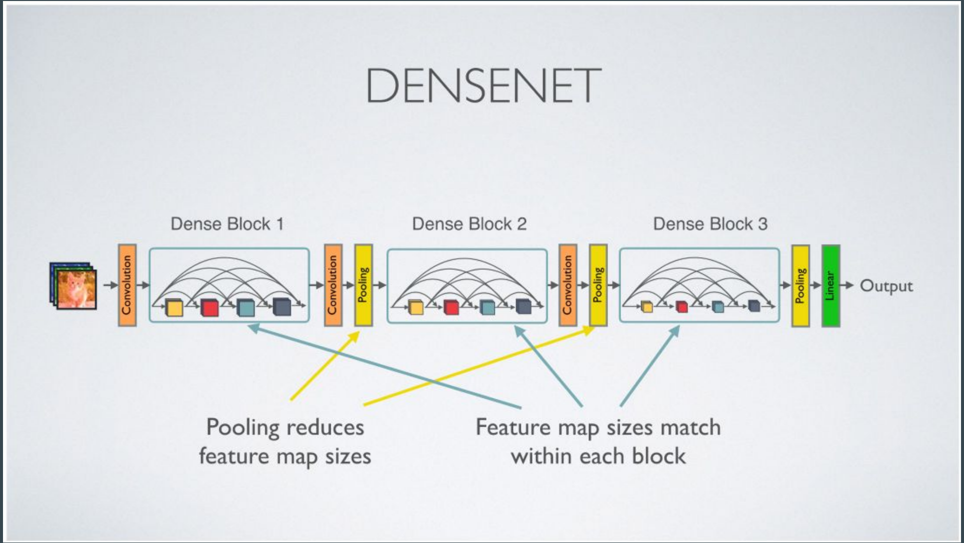
- 세 연산의 복합 함수로 정의한다.
  - 배치 정규화
  - ReLU
  - 3X3 Convolution

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}])$$



3. DENSENETS : Pooling layers

- concat은 feature-map size가 변하면 불가능
- down-sampling 하기 위해 네트워크를 “dense block”으로 나눈다
- block 사이의 계층을 transition layer라 한다.
    - 배치 정규화 계층과 1X1 컨볼루션, 이어서 2X2 average pooling 계층으로 구성



3. DENSENETS : Growth rate

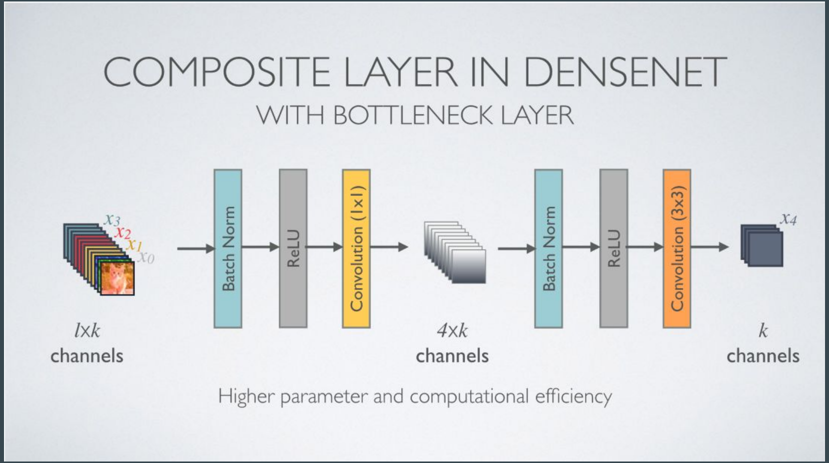
- hyperparameter k는 네트워크의 growth rate이다.
    - Hl이 k개의 feature-map을 생성

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}])$$

    - l번째 계층은 k0 + k*(l-1)개의 feature-map
        - k0는 입력 계층의 채널 수
- DensNet은 좁은 layer를 가진다 (e.g., k=12)
    - 상대적으로 작은 k를 가져도 SOTA 하기에 충분
- Feature-map을 네트워크의 global state로 볼 수 있다.
    - 각 계층은 global state에 자신의 feature-map을 추가한다.
    - growth rate는 새 정보가 얼마나 global state에 기여할지 제어한다.
    - global state는 네트워크 어디서나 접근 가능. 다른 아키텍쳐와 달리 복제할 필요가 없다.

3. DENSENETS : Bottleneck layers

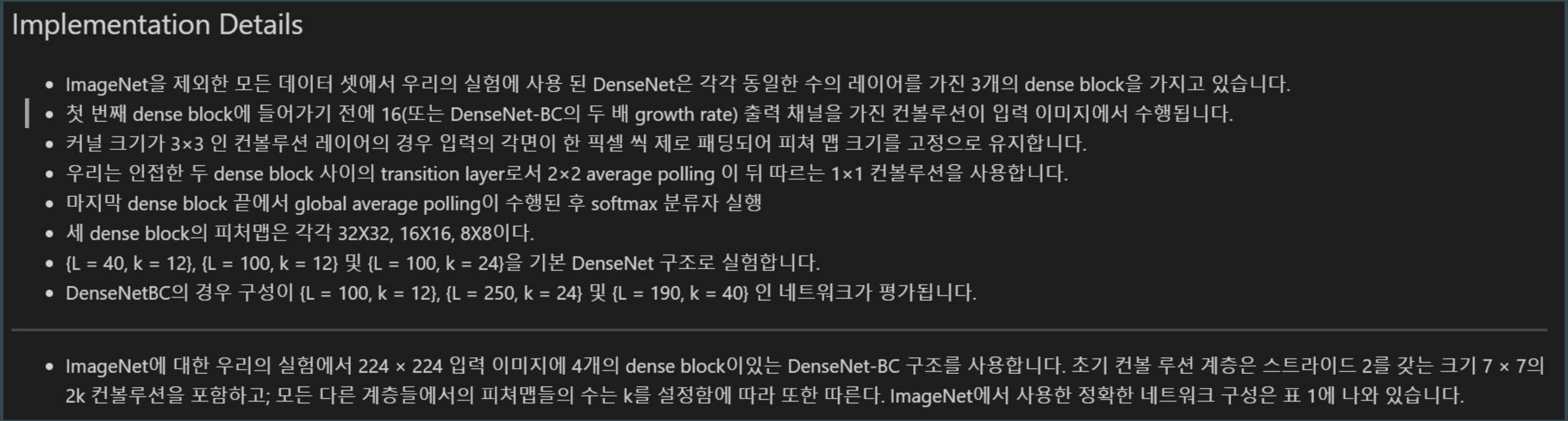
- input feature-map의 수를 줄이기 위해
  - 3X3 conv 이전에 1X1 conv를 병목 층으로 도입하여 연산 효율을 향상시킨다.
    - Densnet에 특히 효과적
    - e.g., DenseNet-B



3. DENSENETS : Compression

- 모델의 압축률 향상시키기 위해
  - dense block이 m개의 feature-map을 가지면 $\lfloor \theta m \rfloor$ 개의 output feature-map을 만든다.
    - 압축 인자 $0 < \theta \leq 1$
    - $\theta = 1$ 이면 feature-map수 변하지 않음
    - DenseNet-C : $\theta < 1$
    - DenseNet-BC : bottleneck 과 함께 적용

3. DENSENETS : Implementation Details



Implementation Details

- ImageNet을 제외한 모든 데이터 셋에서 우리의 실험에 사용 된 DenseNet은 각각 동일한 수의 레이어를 가진 3개의 dense block을 가지고 있습니다.
- 첫 번째 dense block에 들어가기 전에 16(또는 DenseNet-BC의 두 배 growth rate) 출력 채널을 가진 컨볼루션이 입력 이미지에서 수행됩니다.
- 커널 크기가 3×3 인 컨볼루션 레이어의 경우 입력의 각면이 한 픽셀 씩 제로 패딩되어 피처 맵 크기를 고정으로 유지합니다.
- 우리는 인접한 두 dense block 사이의 transition layer로서 2×2 average polling 이 뒤 따르는 1×1 컨볼루션을 사용합니다.
- 마지막 dense block 끝에서 global average polling이 수행된 후 softmax 분류자 실행
- 세 dense block의 피처맵은 각각 32X32, 16X16, 8X8이다.
- {L = 40, k = 12}, {L = 100, k = 12} 및 {L = 100, k = 24}을 기본 DenseNet 구조로 실험합니다.
- DenseNetBC의 경우 구성이 {L = 100, k = 12}, {L = 250, k = 24} 및 {L = 190, k = 40} 인 네트워크가 평가됩니다.

- ImageNet에 대한 우리의 실험에서 224 × 224 입력 이미지에 4개의 dense block이있는 DenseNet-BC 구조를 사용합니다. 초기 컨볼 루션 계층은 스트라이드 2를 갖는 크기 7 × 7의 2k 컨볼루션을 포함하고; 모든 다른 계층들에서의 피쳐맵들의 수는 k를 설정함에 따라 또한 따른다. ImageNet에서 사용한 정확한 네트워크 구성은 표 1에 나와 있습니다.

4. EXPERIMENTS : Datasets

- CIFAR-10, CIFAR-100 : train 50000 / test 10000
- SVHN : train 73000 / test 26000
- ImageNet : train 1200000 / test 50000

4. EXPERIMENTS : Training

모든 network에 SGD(확률적 경사 하강법) 사용

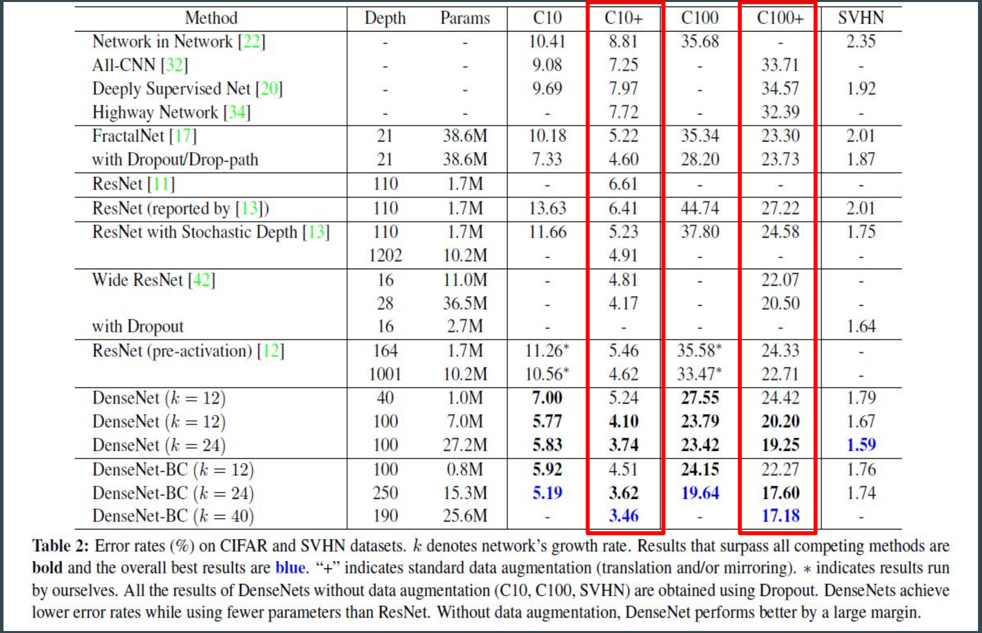
- CIFAR : 300 epoch, batch size=64, learning rate=0.1
- SVHN : 40 epoch, batch size=64, learning rate=0.1
- ImageNet : 90 epoch, batch size=256, learning rate=0.1

4. EXPERIMENTS : Classification Results

- CIFAR / SVHN

  ResNet과의 비교(Table 2)

  - 더 높은 Accuracy
  - L, k가 증가할 수록 향상되는 Model Capacity
  - 더 높은 Parameter Efficiency (더 적은 매개변수 필요)
  - Overfitting를 방지하기 위한 DenseNet-BC



Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

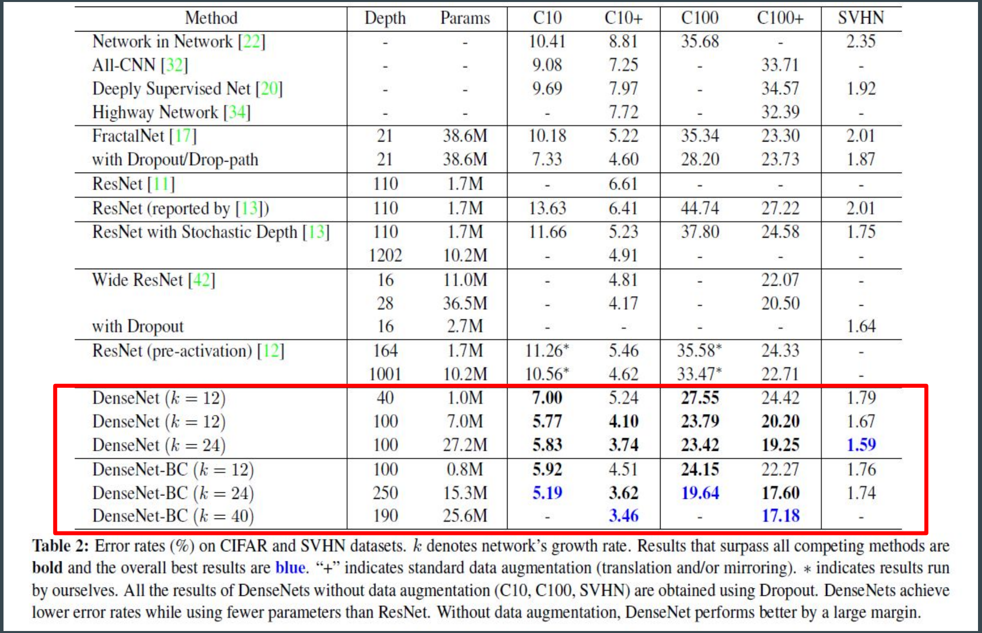
Table 2: Error rates (%) on CIFAR and SVHN datasets. k denotes network's growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. "+" indicates standard data augmentation (translation and/or mirroring). $*$ indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

4. EXPERIMENTS : Classification Results

- CIFAR / SVHN

  ResNet과의 비교(Table 2)

  - 더 높은 Accuracy
  - L, k가 증가할 수록 향상되는 Model Capacity
  - 더 높은 Parameter Efficiency (더 적은 매개변수 필요)
  - Overfitting를 방지하기 위한 DenseNet-BC



Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

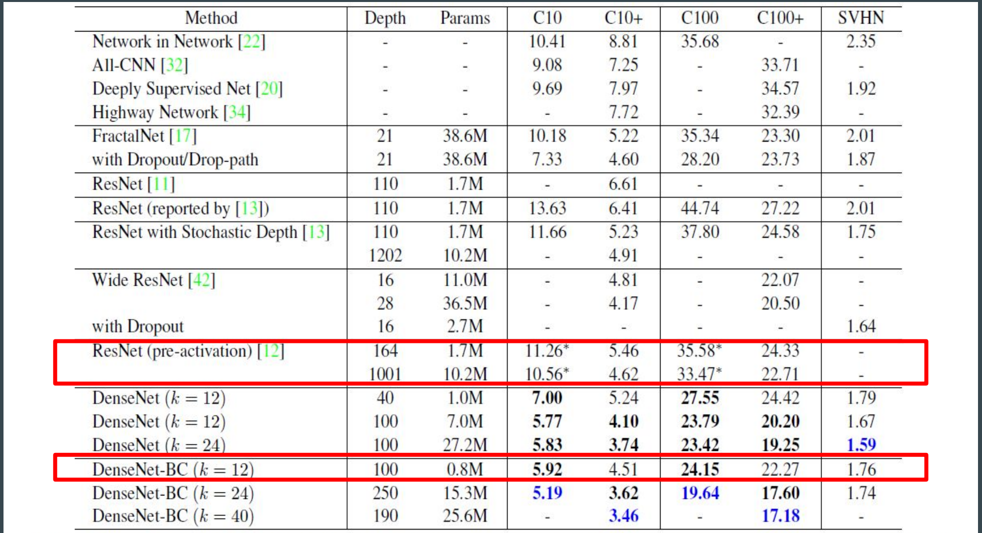
Table 2: Error rates (%) on CIFAR and SVHN datasets. k denotes network's growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. "+" indicates standard data augmentation (translation and/or mirroring). $*$ indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

4. EXPERIMENTS : Classification Results

- CIFAR / SVHN

  ResNet과의 비교(Table 2)

  - 더 높은 Accuracy
  - L, k가 증가할 수록 향상되는 Model Capacity
  - 더 높은 Parameter Efficiency (더 적은 매개변수 필요)
  - Overfitting를 방지하기 위한 DenseNet-BC



Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

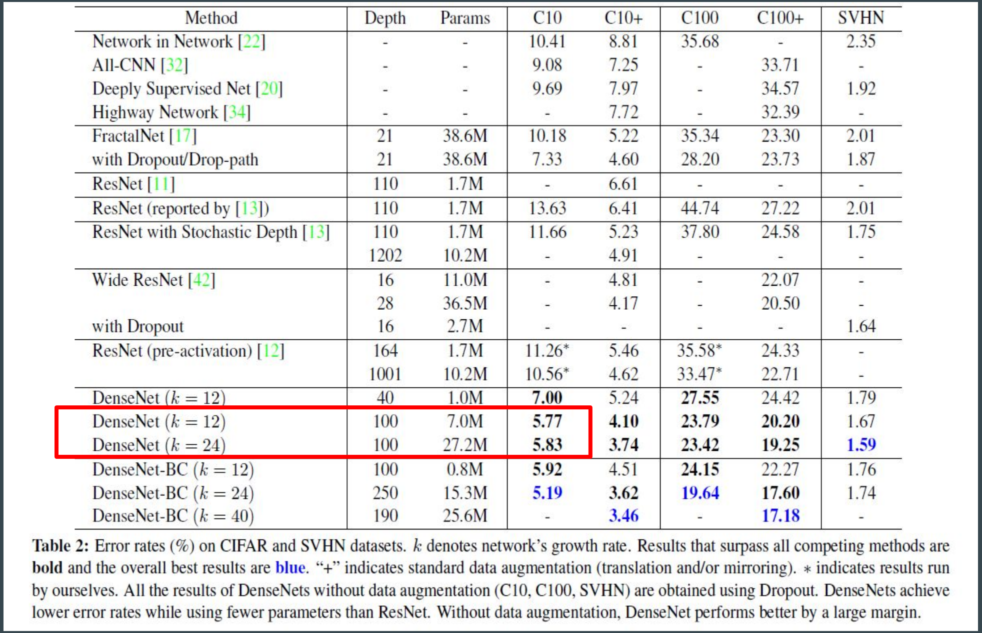
Table 2: Error rates (%) on CIFAR and SVHN datasets. k denotes network's growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. "+" indicates standard data augmentation (translation and/or mirroring). $*$ indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

4. EXPERIMENTS : Classification Results

- CIFAR / SVHN

  ResNet과의 비교(Table 2)

  - 더 높은 Accuracy
  - L, k가 증가할 수록 향상되는 Model Capacity
  - 더 높은 Parameter Efficiency (더 적은 매개변수 필요)
  - Overfitting를 방지하기 위한 DenseNet-BC



Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

Table 2: Error rates (%) on CIFAR and SVHN datasets. k denotes network's growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. "+" indicates standard data augmentation (translation and/or mirroring). $*$ indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

4. EXPERIMENTS : Classification Results

- ImageNet
  - ResNet-101과 같은 성능을 내는 DenseNet-201에서는 더 적은 연산(매개변수)를 필요로 함.

> 실험조건을 고려하면,
추 후, DenseNet에 최적화된
Hyper parameter 조정을 통해
모델 성능을 향상시킬 수 있음.

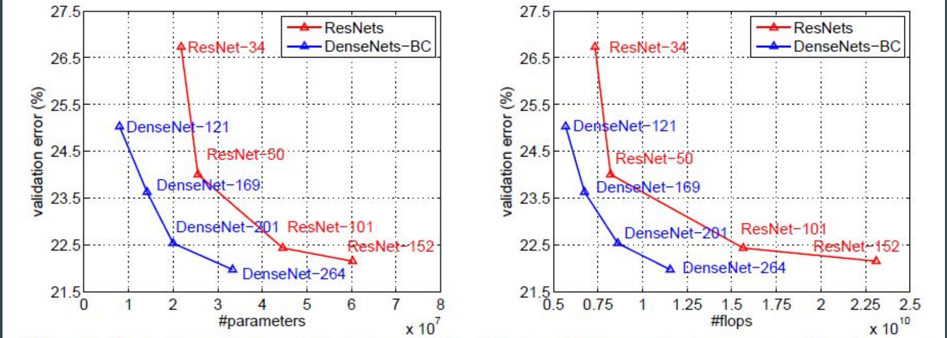


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

5. DISCUSSION

- Model Compactness
- 모든 layer에 대한 Implicit Deep Supervision
- Stochastic vs Deterministic connection
- Feature Reuse

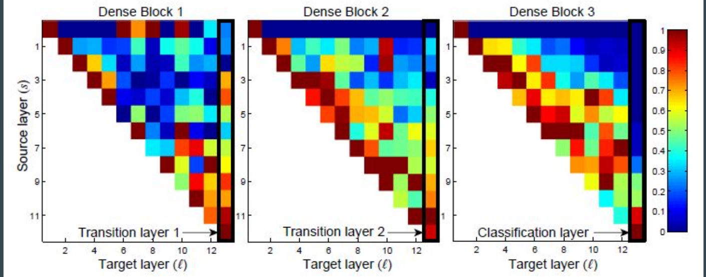


Figure 5: The average absolute filter weights of convolutional layers in a trained DenseNet. The color of pixel (s, ℓ) encodes the average $L1$ norm (normalized by number of input feature-maps) of the weights connecting convolutional layer s to ℓ within a dense block. Three columns highlighted by black rectangles correspond to two transition layers and the classification layer. The first row encodes weights connected to the input layer of the dense block.

6. CONCLUSION

- DenseNet-BC의 경우, 매개변수가 증가함에 따라 성능 저하나 overfitting 문제가 일어나지 않고 일관되게 정확도가 향상됨.
- 기존 model들에 비해 더 적은 매개변수, 더 적은 연산 필요
- deep supervision, compact한 내부 표현, feature 중복 방지의 장점
- DenseNet에 최적화된 hyper parameter, learning rate 조정을 통한 성능 향상이 기대되는 model

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]),$$

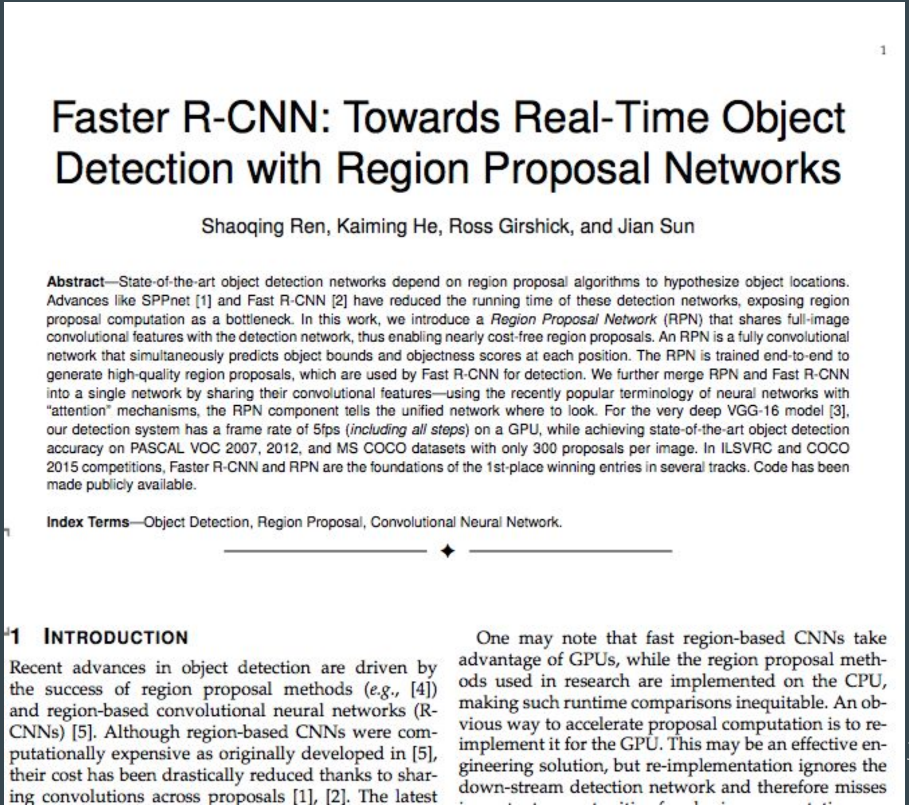
6. COMMENTS on this paper

- DenseNet의 장점 중 regularization 효과가 있어 작은 데이터셋에도 오버피팅이 일어나지 않는다고 했으나 설명이 너무 부족함.
- concat하면 파라미터 수가 적어진다고 하였는데, feature-map의 수가 더 증가하므로 층이 깊어질수록 연산이 복잡해지는 건 아닌지 의문이 듬
- 매개변수를 증가시켜도 overfitting이 되지 않고 compact한 모델을 만들었다는 결과에 대해 증거가 조금 부족해보임.

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

Faster R-CNN

0. Abstract
1. Introduction
2. Related Work
3. Faster R-CNN
4. Experiments
5. Conclusion



1

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

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

Abstract—State-of-the-art object detection networks depend on region proposal algorithms to hypothesize object locations. Advances like SPPnet [1] and Fast R-CNN [2] have reduced the running time of these detection networks, exposing region proposal computation as a bottleneck. In this work, we introduce a *Region Proposal Network* (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. We further merge RPN and Fast R-CNN into a single network by sharing their convolutional features—using the recently popular terminology of neural networks with "attention" mechanisms, the RPN component tells the unified network where to look. For the very deep VGG-16 model [3], our detection system has a frame rate of 5fps (*including all steps*) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks. Code has been made publicly available.

Index Terms—Object Detection, Region Proposal, Convolutional Neural Network.

1 INTRODUCTION

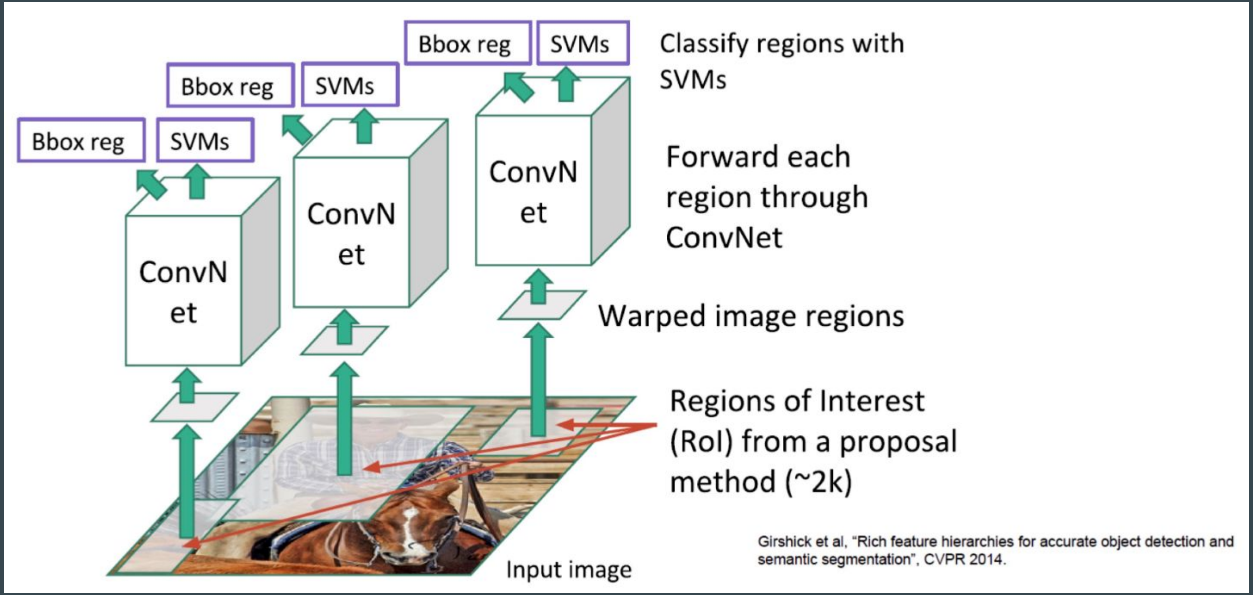
Recent advances in object detection are driven by the success of region proposal methods (*e.g.*, [4]) and region-based convolutional neural networks (R-CNNs) [5]. Although region-based CNNs were computationally expensive as originally developed in [5], their cost has been drastically reduced thanks to sharing convolutions across proposals [1], [2]. The latest

One may note that fast region-based CNNs take advantage of GPUs, while the region proposal methods used in research are implemented on the CPU, making such runtime comparisons inequitable. An obvious way to accelerate proposal computation is to re-implement it for the GPU. This may be an effective engineering solution, but re-implementation ignores the down-stream detection network and therefore misses

1. INTRODUCTION

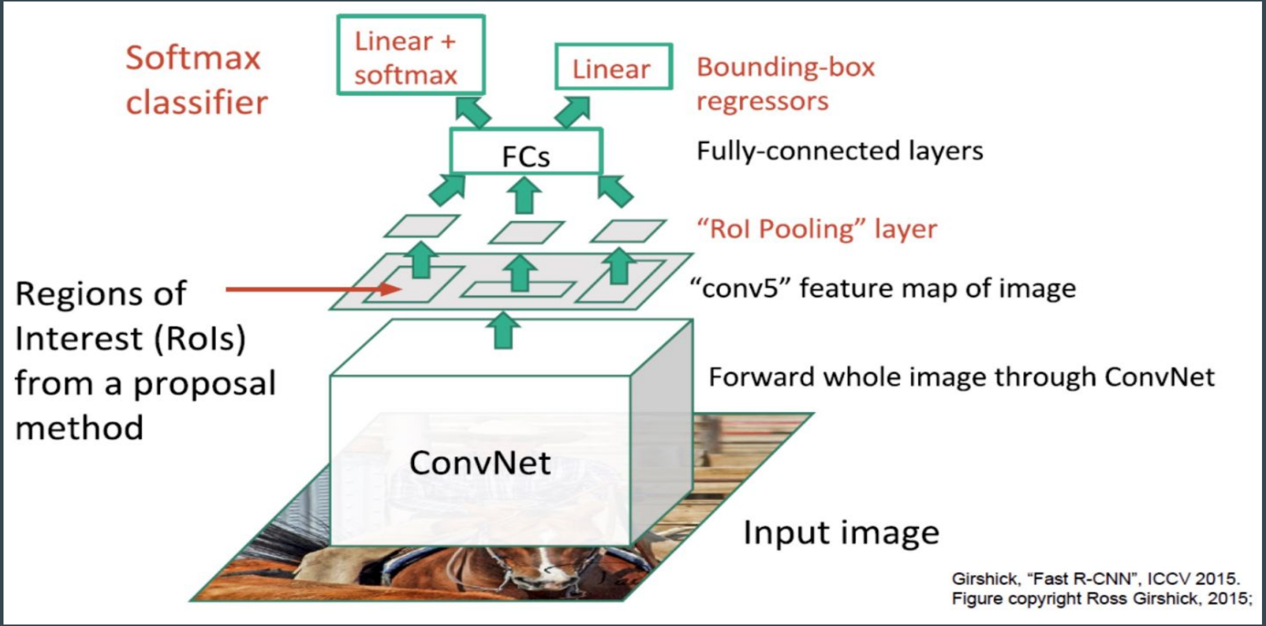
-기존의 region proposal method : Selective Search / EdgeBoxes

-R-CNN


Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Classify regions with SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

1. INTRODUCTION

-fast R-CNN

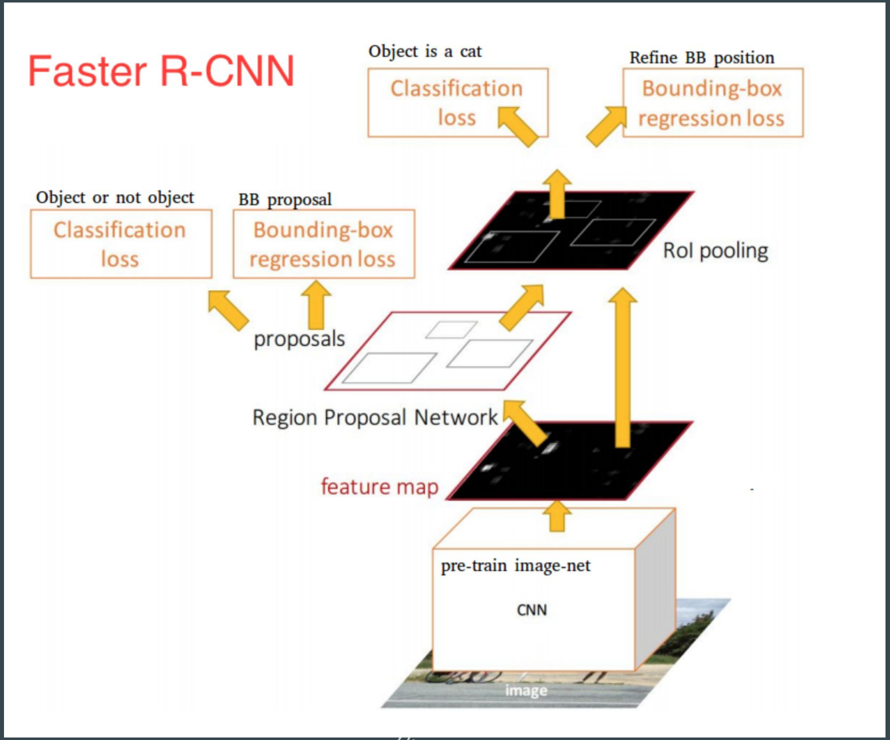


Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015;

1. INTRODUCTION

-RPN : region proposal network

-RPN에서 out되는 proposal들을 통해서 detection network에서 classification과 bounding box regression을 수행함.



2. RELATED WORK

-object proposals : grouping superpixels(Selective Search)

sliding windows(EdgeBoxes)

-OverFeat: single object를 가정한 상황에서 fully connected layer로 bounding box의 좌표를 예측하는 방법

-MultiBox: OverFeat의 일반화된 버젼으로 마지막 fully connected layer에서 여러 개의

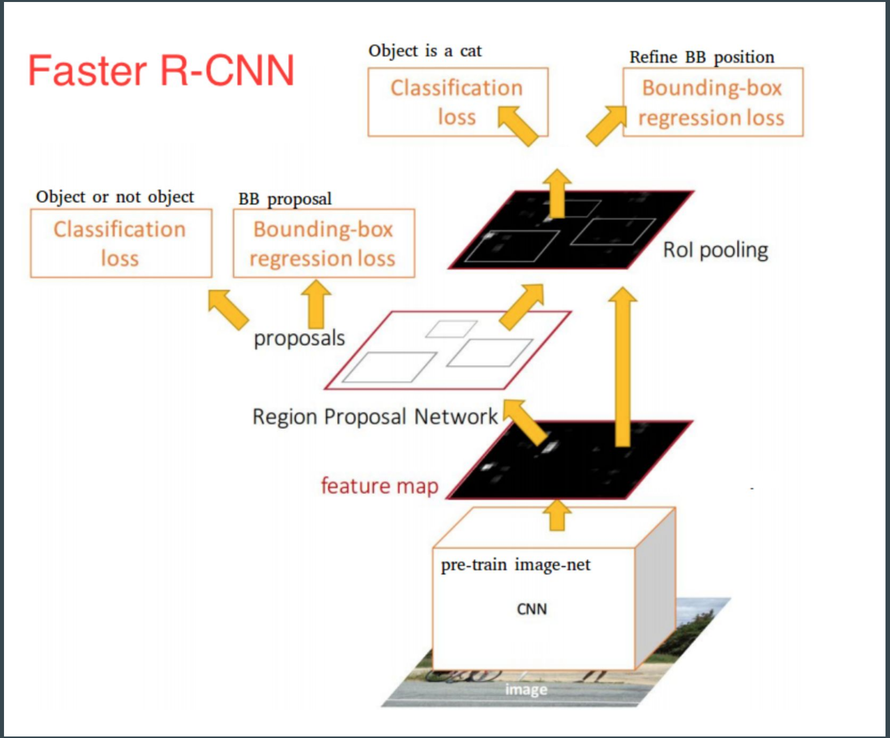
box의 좌표를 예측함

3. FASTER R-CNN

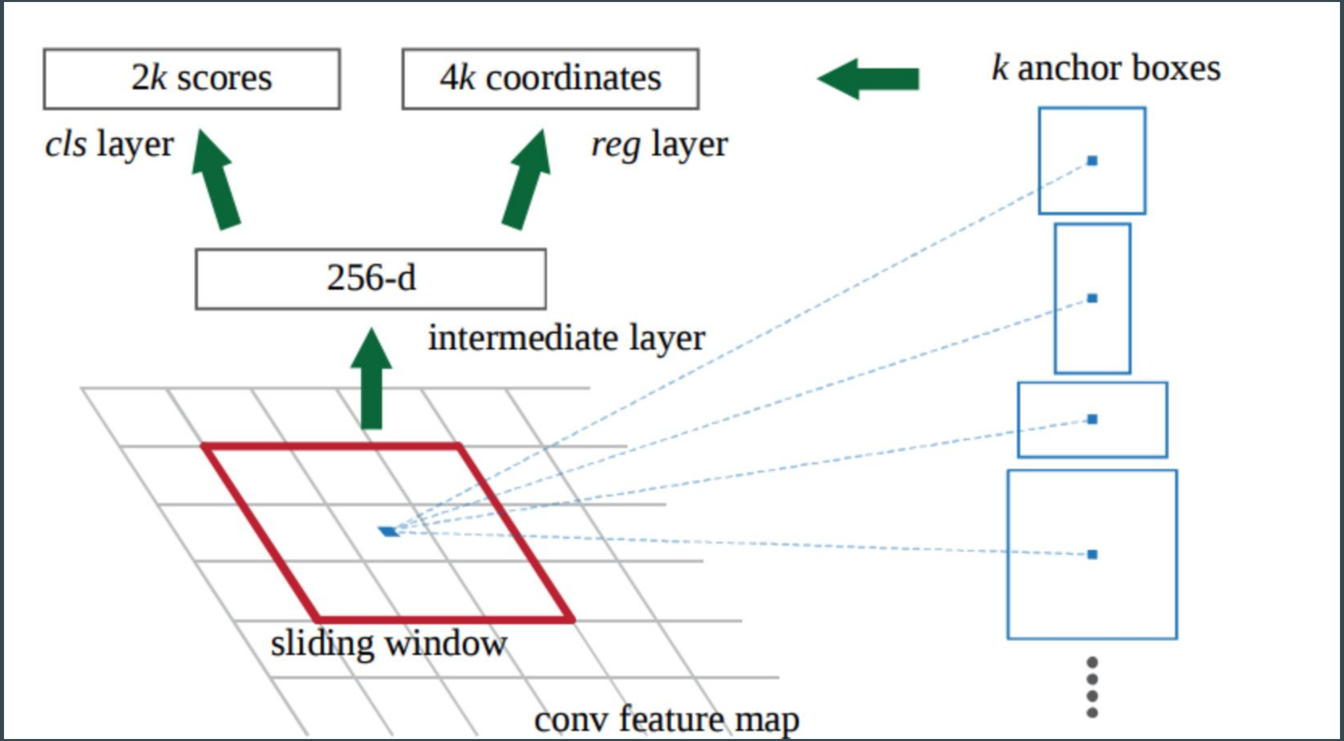
-feature network : pre-train된 network에서 (e.g. VGG) 마지막 몇 개의 layer만 제외하여 feature map을 전달

-Region Proposal Network

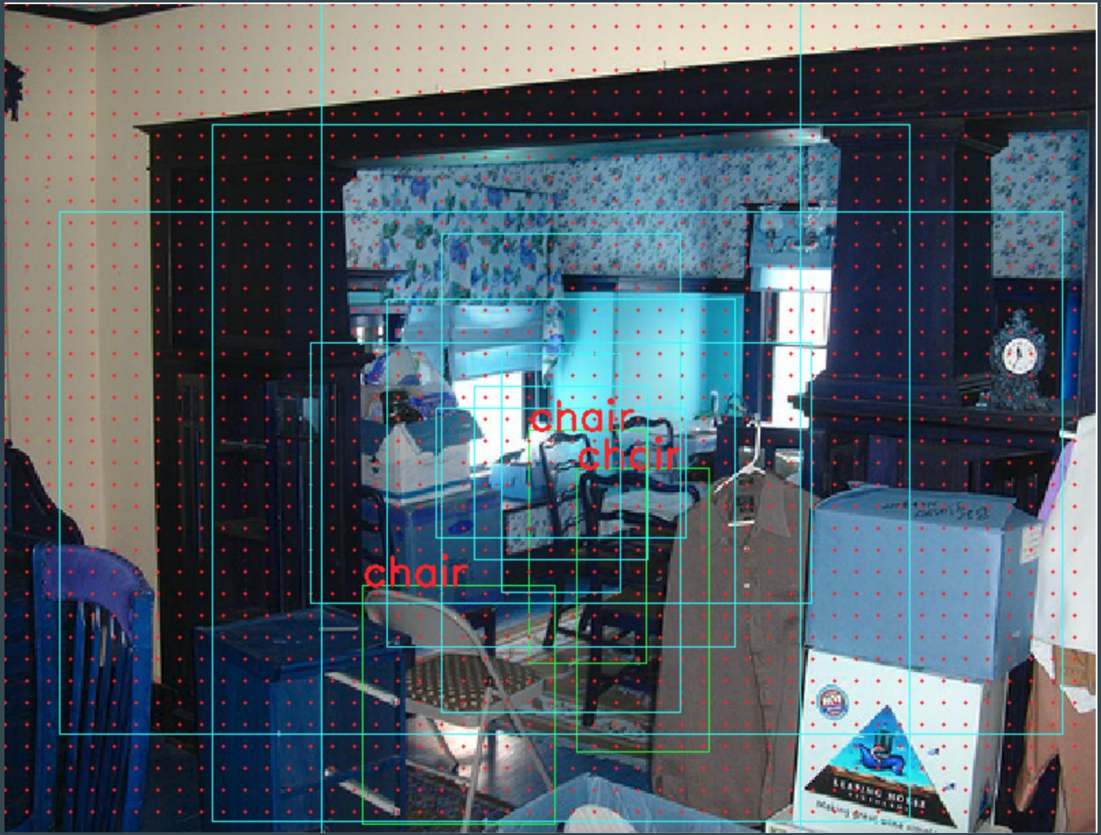
-Detector Network



3. FASTER R-CNN



3. FASTER R-CNN



3. FASTER R-CNN

$$IoU = \frac{\text{anchor} \cap \text{ground-truth box}}{\text{anchor} \cup \text{ground-truth box}}$$

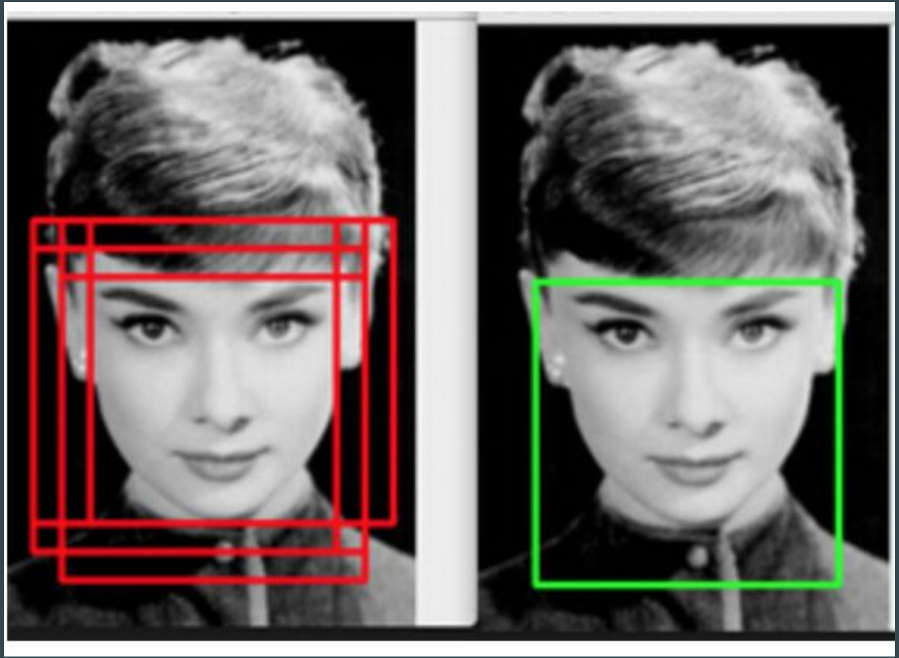
$$p^* = \begin{cases} 1 & \text{if } IoU > 0.7 \\ -1 & \text{if } IoU < 0.3 \\ 0 & \text{if otherwise} \end{cases}$$

3. FASTER R-CNN

-Non-maximum suppression

=> high score 순으로 proposal들을 sorting하여 score가 가장 높은 proposal 순으로 score가 높은 다른 proposal과의 overlapping이 threshold보다 높은 proposal 삭제

=> proposal의 redundancy를 줄여줌



3. FASTER R-CNN

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

-i : index of anchor

- pi : predicted probability / pi* : ground truth label

- ti : predicted coordinate vector / ti* : ground truth bounding box coordinate vector

- Lreg = R(ti - ti*) : smooth L1

- Ncls / Nreg / lambda : normalization term

3. FASTER R-CNN

$$\mathrm{smooth}_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

- pi : predicted probability / pi* : ground truth label

- ti : predicted coordinate vector / ti* : ground truth bounding box coordinate vector

- Lreg = R(ti - ti*) : smooth L1

- Ncls / Nreg / lambda : normalization term

3. FASTER R-CNN

- training RPN

=> image boundary를 넘어간 anchor들은 training phase에서는 고려안함

=> NMS로 reduction한 abchor 중에서 256를 random sampling 하되 positive label

과 negative label의 비율이 1:1이 되도록 함(positive가 부족하면 negative로 채움)

=> 각 weight들은 ~N(0 , 1^2)으로 초기화함

3. FASTER R-CNN

- training RPN

  => SGD optimization 방법으로 end-to-end로 학습함

- sharing features for RPN & Fast R-CNN

  *Alternate training

  *Approximate joint training

  *Non-approximate joint training

3. FASTER R-CNN

- sharing features for RPN & Fast R-CNN

  => 4-step Alternate Training

1. pre train model M0 => RPN model M1 학습
2. M1을 사용하여 region proposal P1 추출
3. P1 / M0 사용하여 Fast R-CNN model M2 학습
4. M2 paramter 고정시킨 상태에서 RPN model M3 학습
5. M3 사용하여 region proposal P2 추출
6. P2 / M3 사용하여 Fast R-CNN model M4 학습

4. EXPERIMENTS

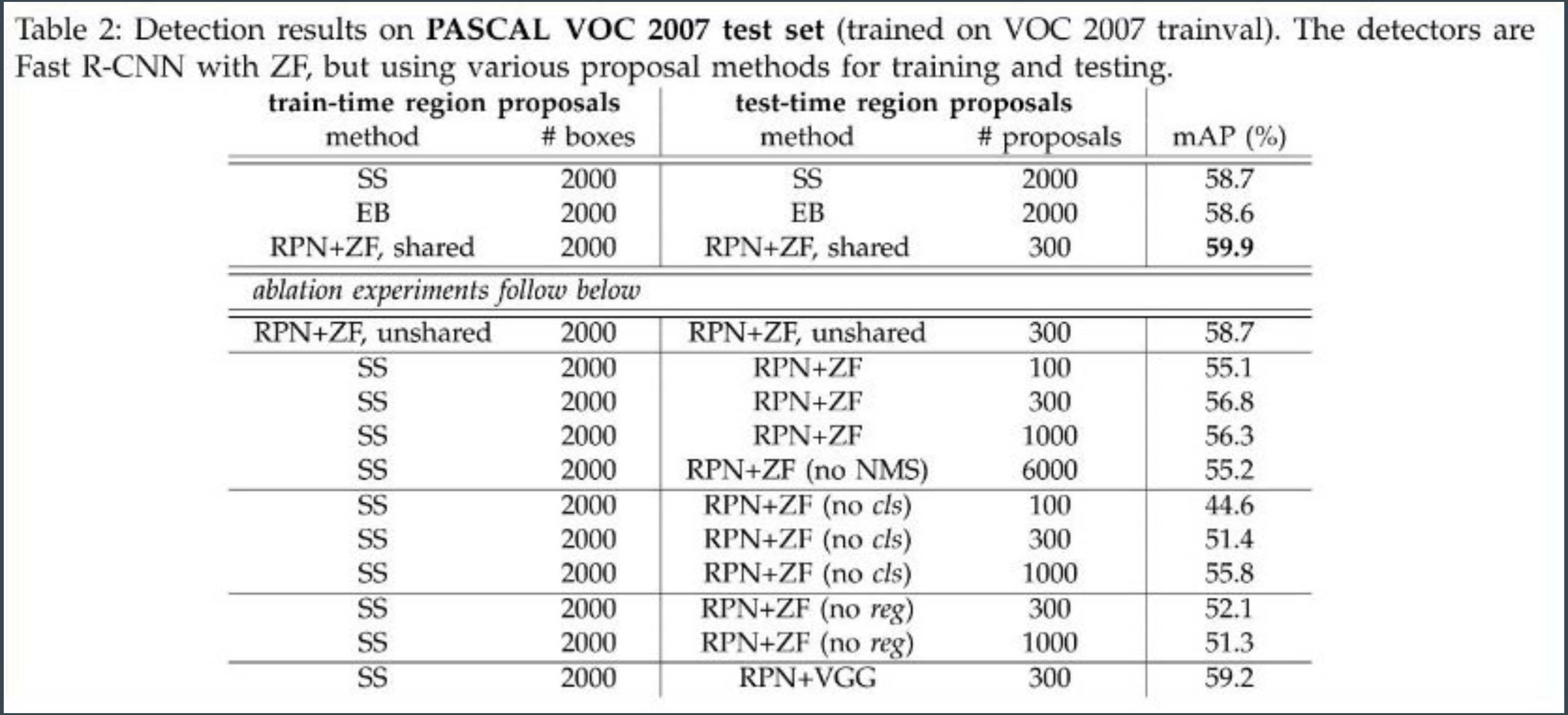


Table 2: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

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

4. EXPERIMENTS

mAP : mean Average Precision

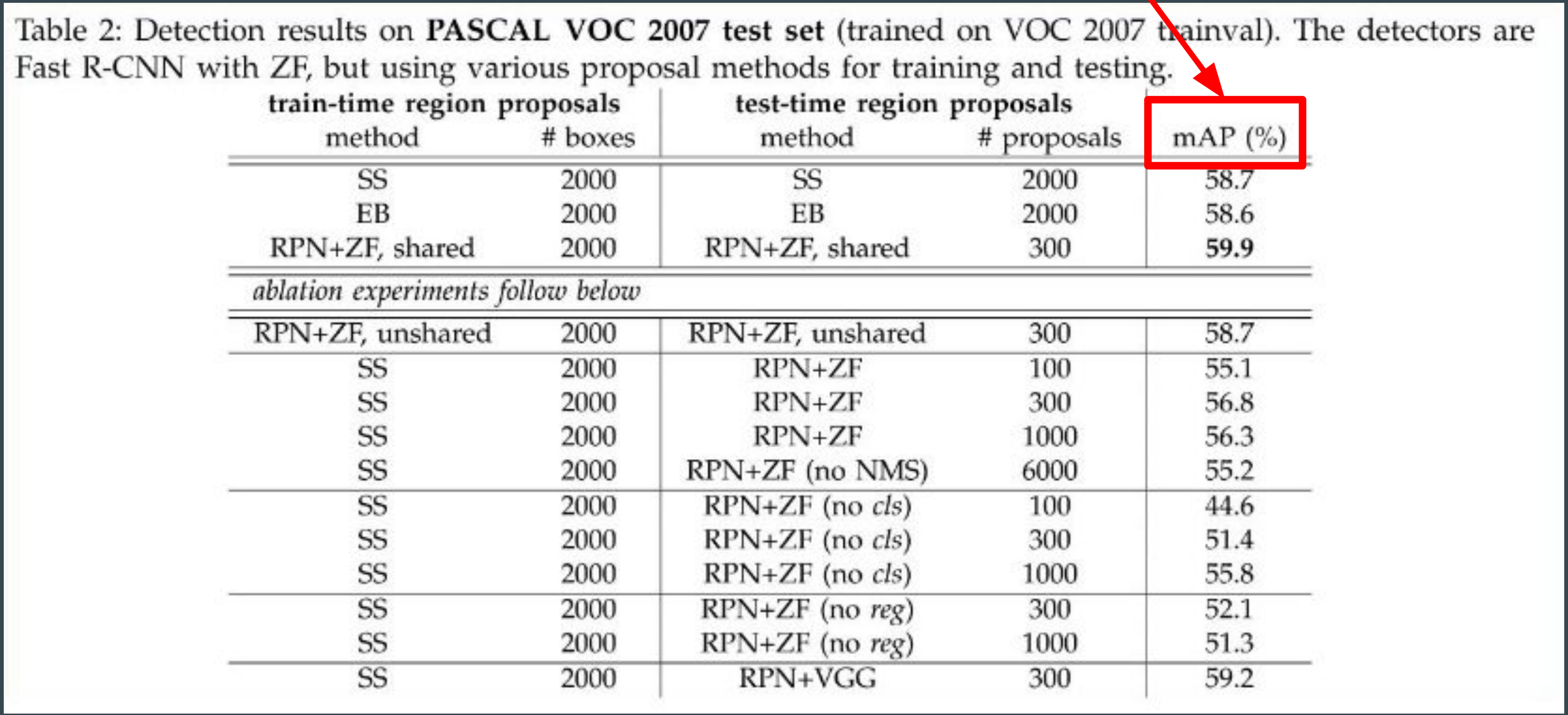


Table 2: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

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

4. EXPERIMENTS

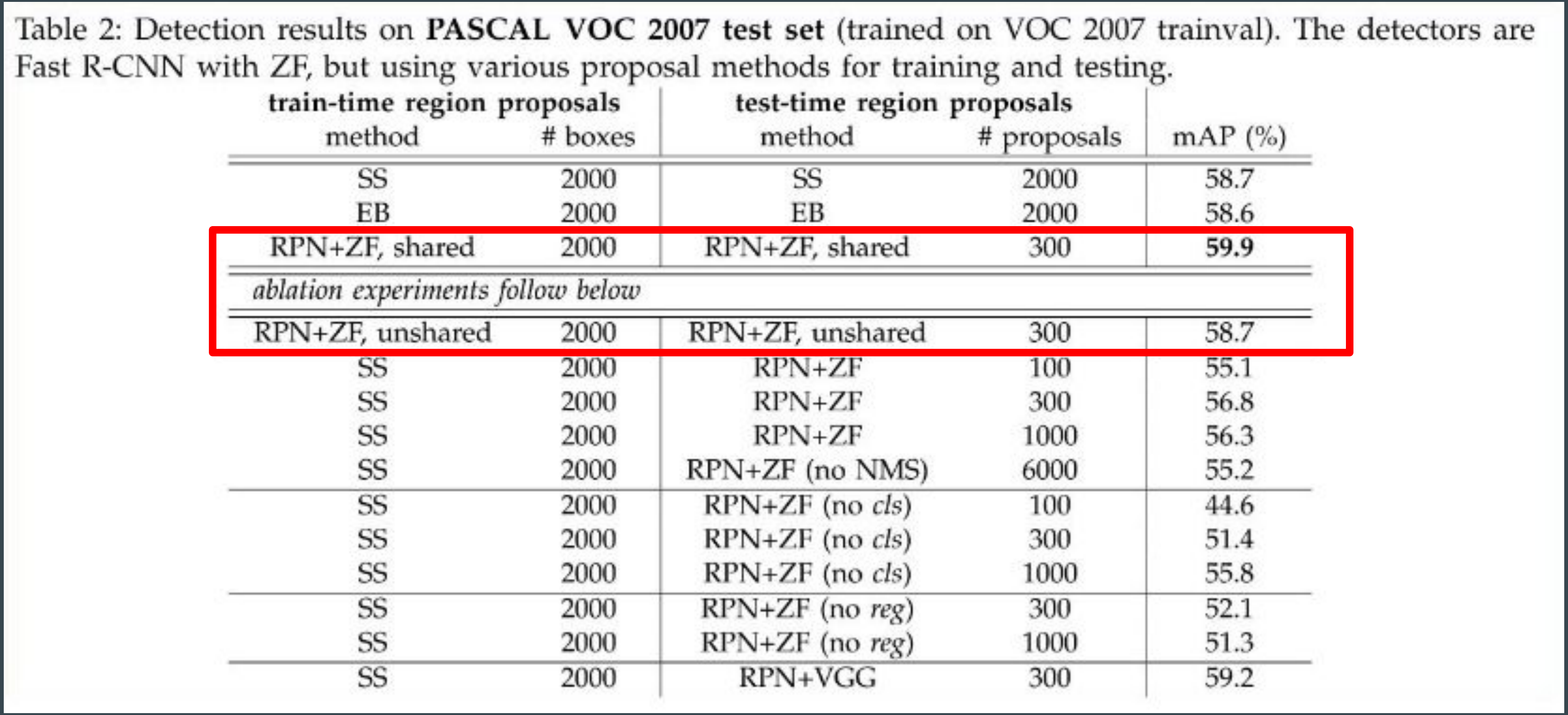


Table 2: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

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

4. EXPERIMENTS

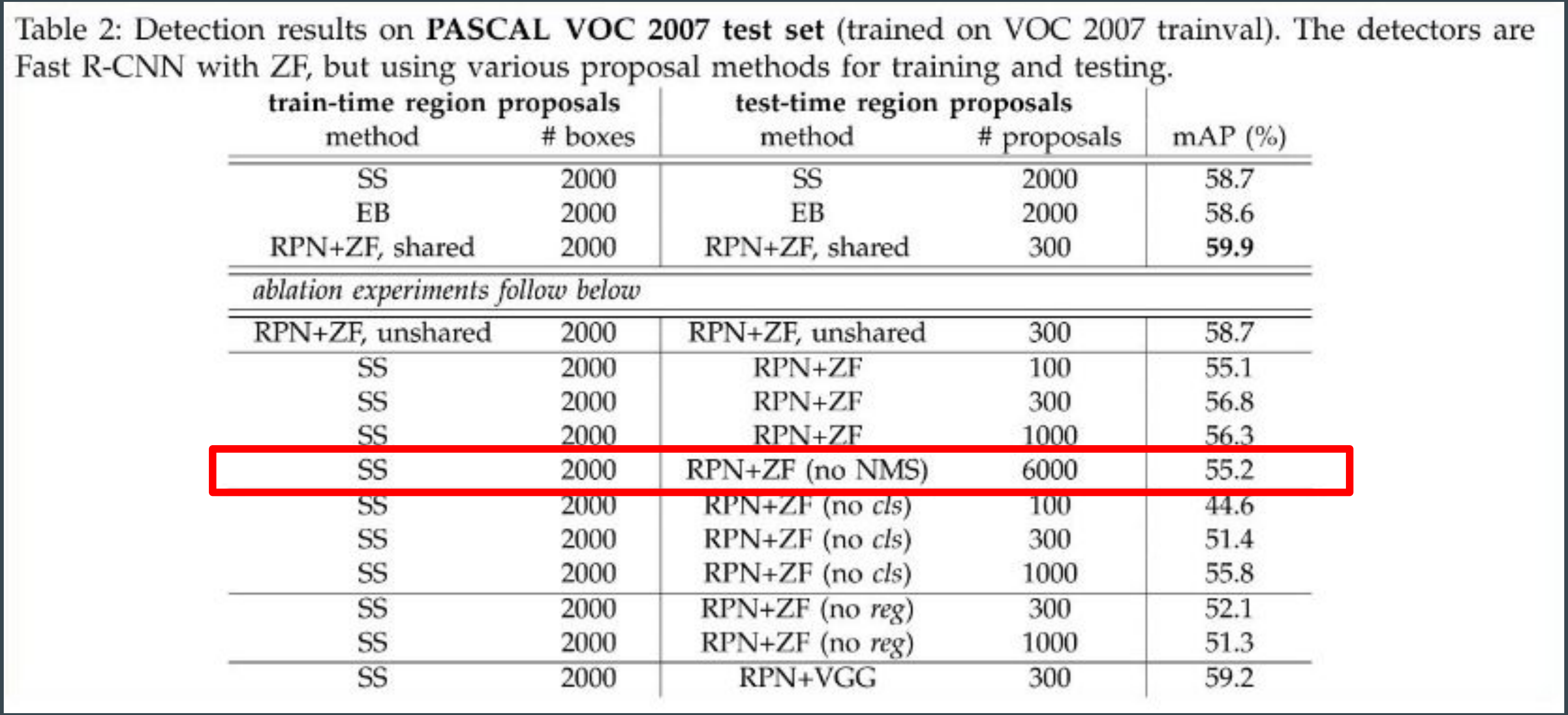


Table 2: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

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

4. EXPERIMENTS

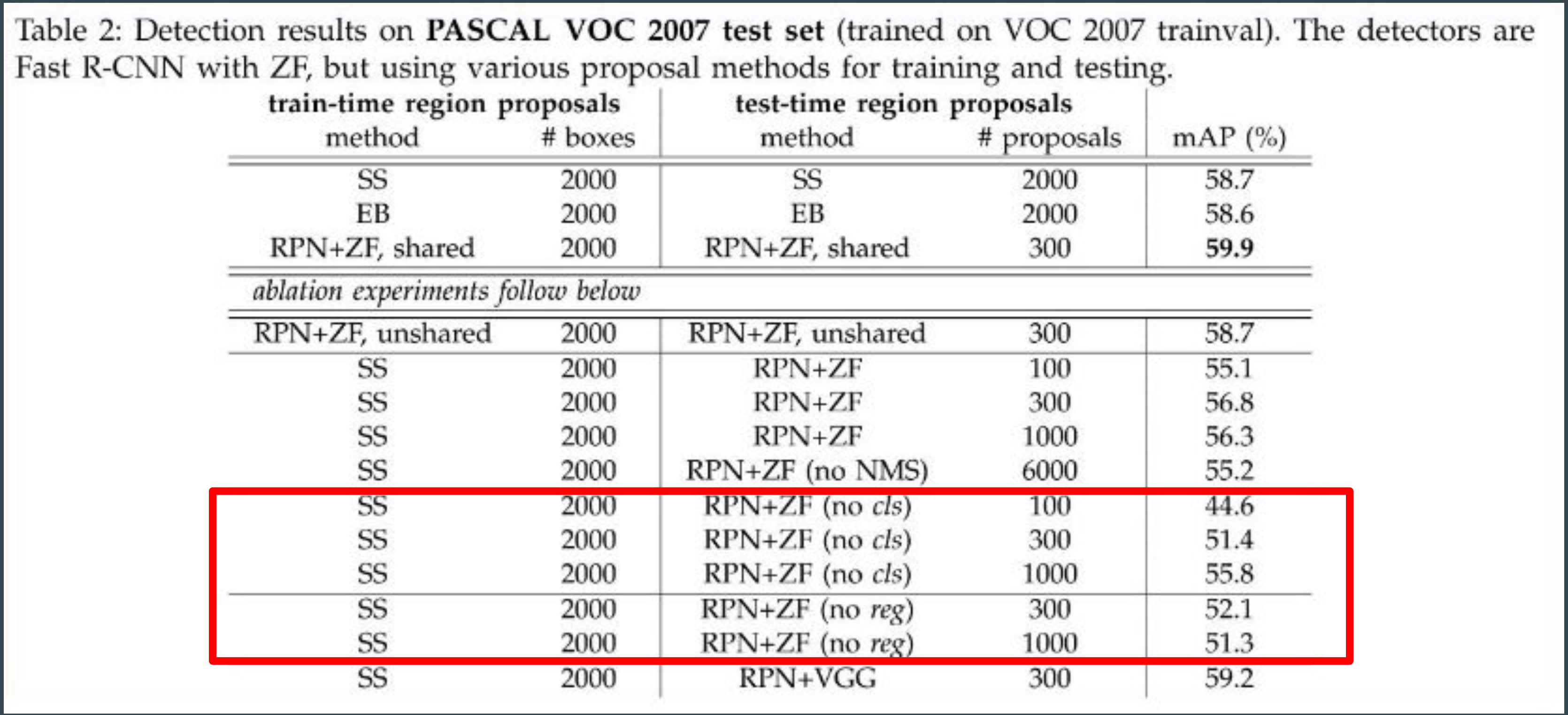


Table 2: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

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

4. EXPERIMENTS

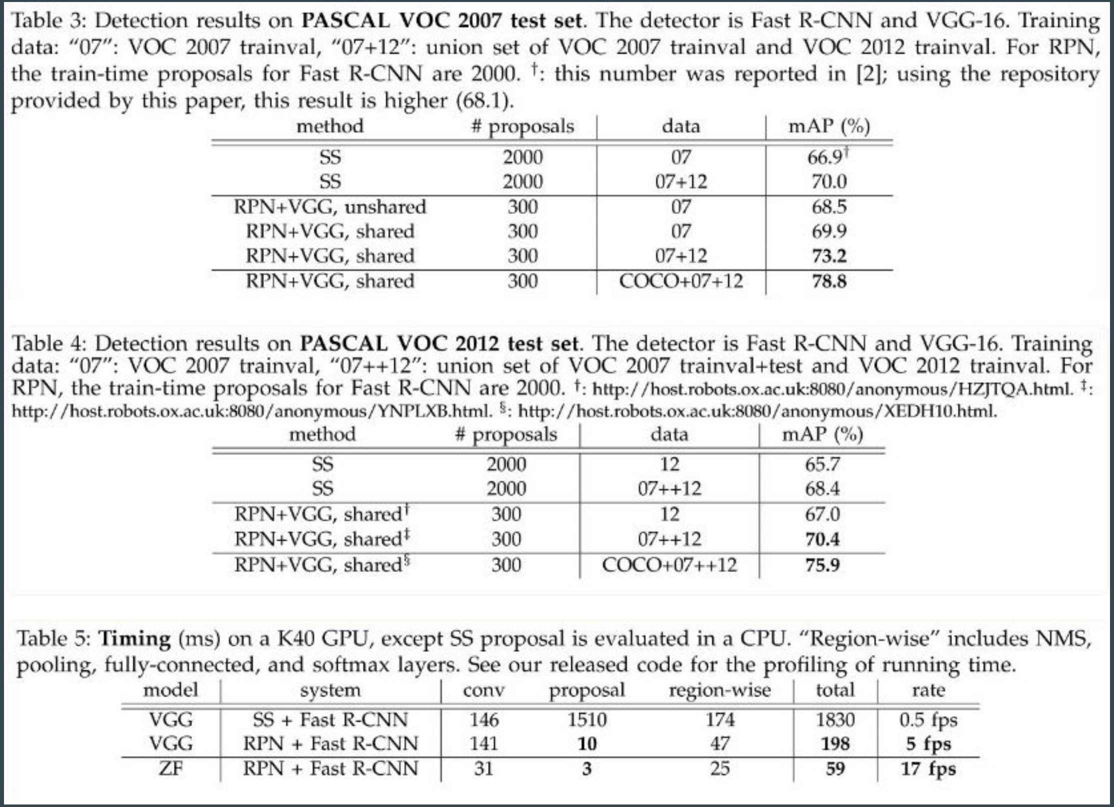


Table 3: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: this number was reported in [2]; using 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**

Table 4: Detection results on **PASCAL VOC 2012 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+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html. ‡: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html. §: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**
RPN+VGG, shared§	300	COCO+07++12	**75.9**

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**

4. EXPERIMENTS

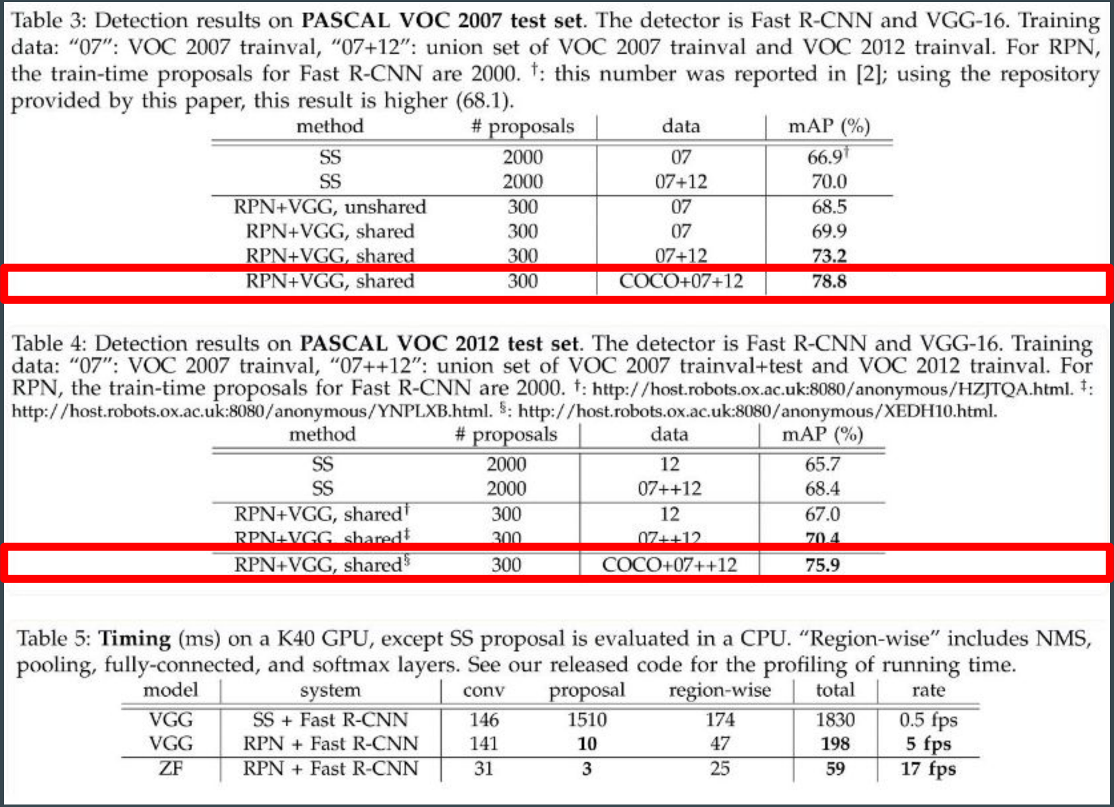


Table 3: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: this number was reported in [2]; using 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**

Table 4: Detection results on **PASCAL VOC 2012 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+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html. ‡: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html. §: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**
RPN+VGG, shared§	300	COCO+07++12	**75.9**

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**

4. EXPERIMENTS

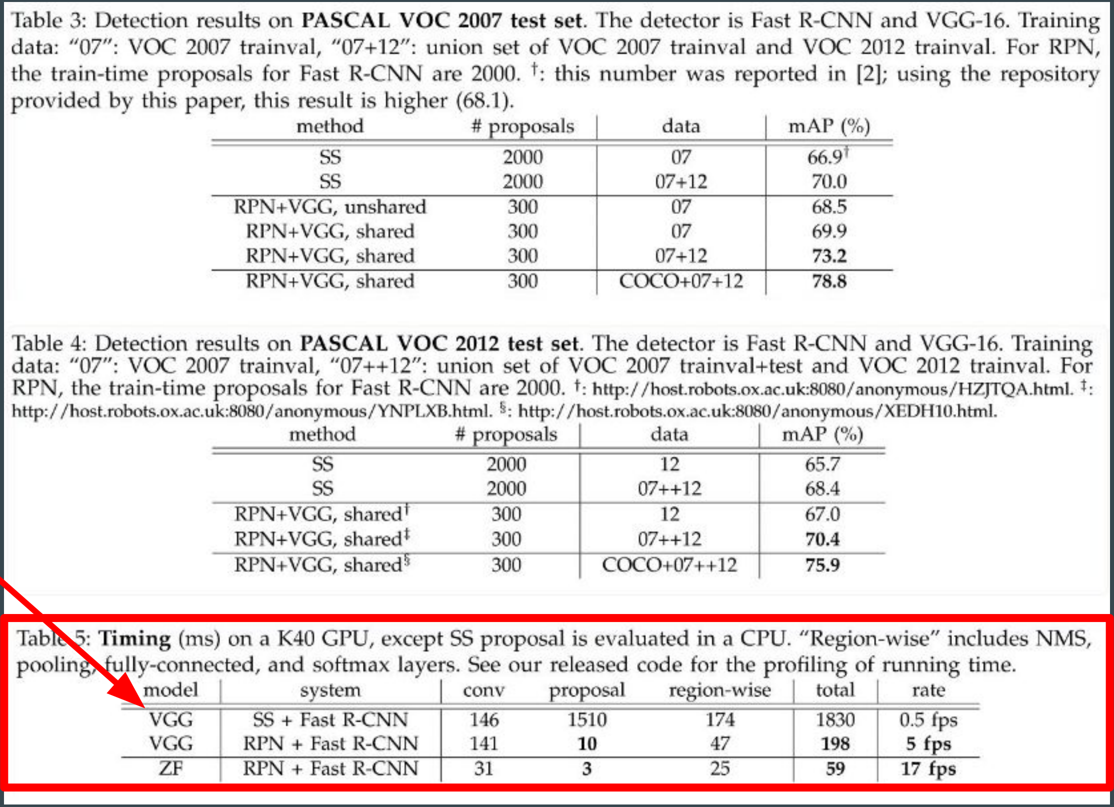


Table 3: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: this number was reported in [2]; using 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**

Table 4: Detection results on **PASCAL VOC 2012 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+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html. ‡: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html. §: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**
RPN+VGG, shared§	300	COCO+07++12	**75.9**

Fast R-CNN

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**

4. EXPERIMENTS

Faster R-CNN

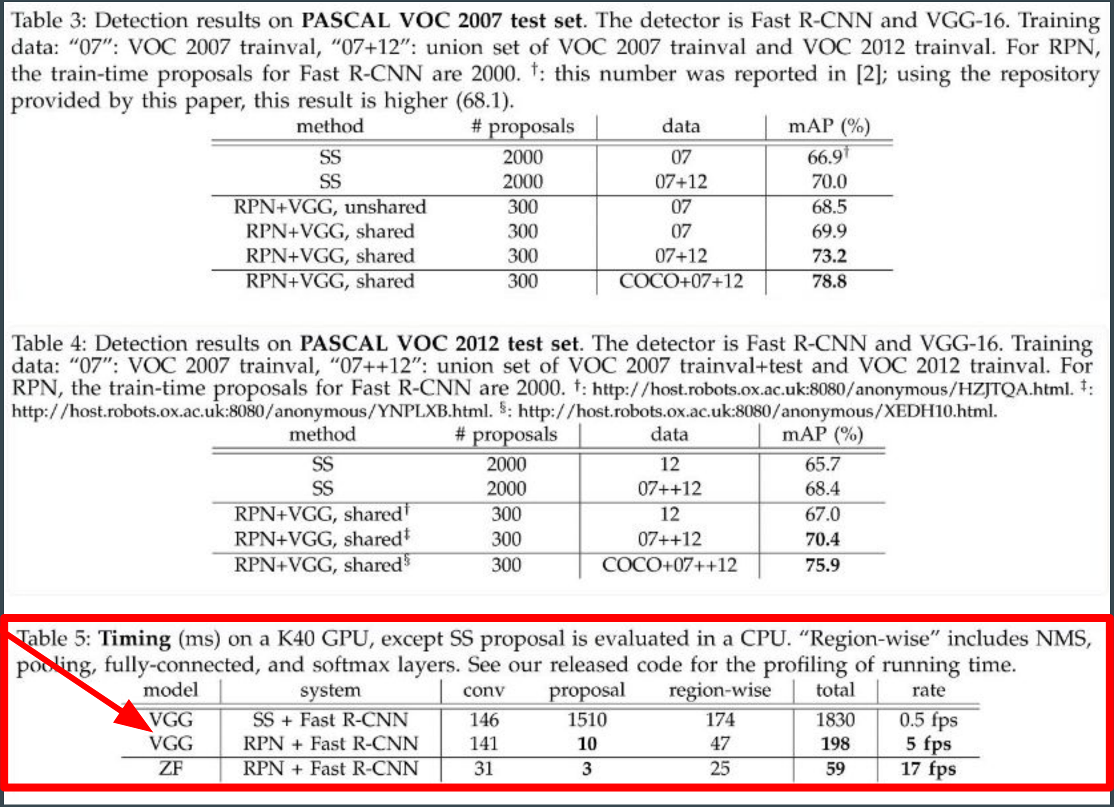


Table 3: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: this number was reported in [2]; using 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**

Table 4: Detection results on **PASCAL VOC 2012 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+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html. ‡: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html. §: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**
RPN+VGG, shared§	300	COCO+07++12	**75.9**

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**

4. EXPERIMENTS

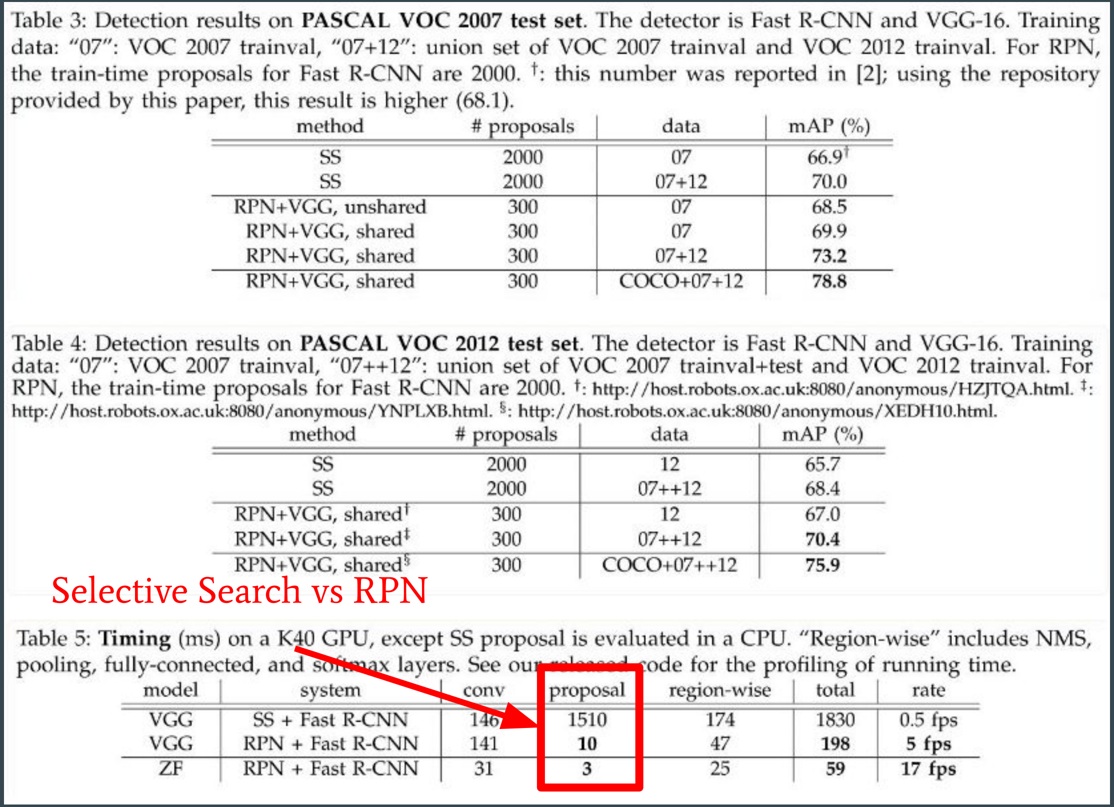


Table 3: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: this number was reported in [2]; using 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**

Table 4: Detection results on **PASCAL VOC 2012 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+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html. ‡: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html. §: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**
RPN+VGG, shared§	300	COCO+07++12	**75.9**

Selective Search vs 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**

4. EXPERIMENTS

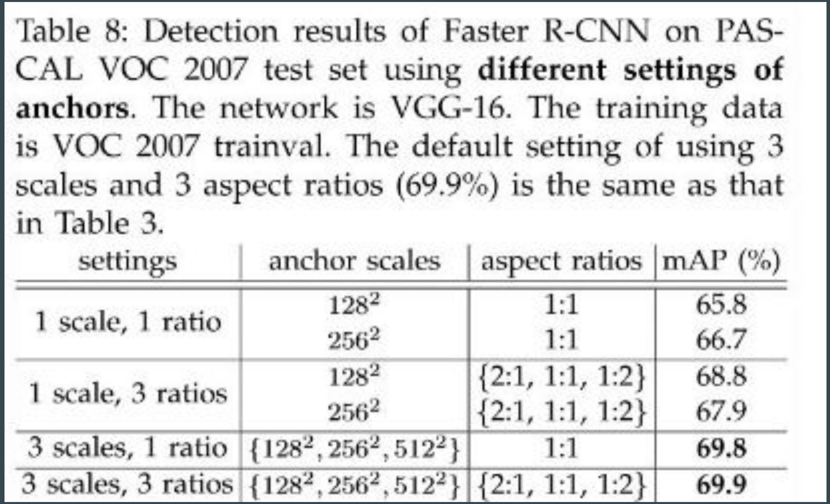


Table 8: Detection results of Faster R-CNN on PASCAL VOC 2007 test set using **different settings of anchors**. The network is VGG-16. The training data is VOC 2007 trainval. The default setting of using 3 scales and 3 aspect ratios (69.9%) is the same as that in Table 3.

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**

4. EXPERIMENTS

K=9로 잡은 이유 + 확장성

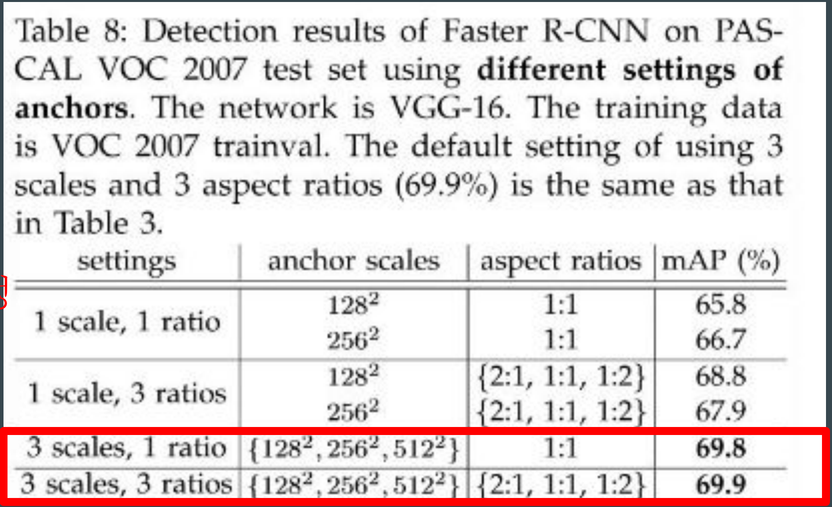


Table 8: Detection results of Faster R-CNN on PASCAL VOC 2007 test set using **different settings of anchors**. The network is VGG-16. The training data is VOC 2007 trainval. The default setting of using 3 scales and 3 aspect ratios (69.9%) is the same as that in Table 3.

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**

4. EXPERIMENTS

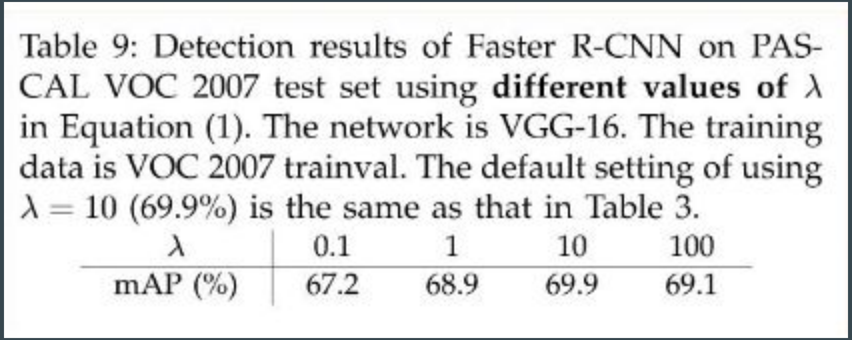


Table 9: Detection results of Faster R-CNN on PASCAL VOC 2007 test set using **different values of λ** in Equation (1). The network is VGG-16. The training data is VOC 2007 trainval. The default setting of using $\lambda = 10$ (69.9%) is the same as that in Table 3.

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

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

4. EXPERIMENTS

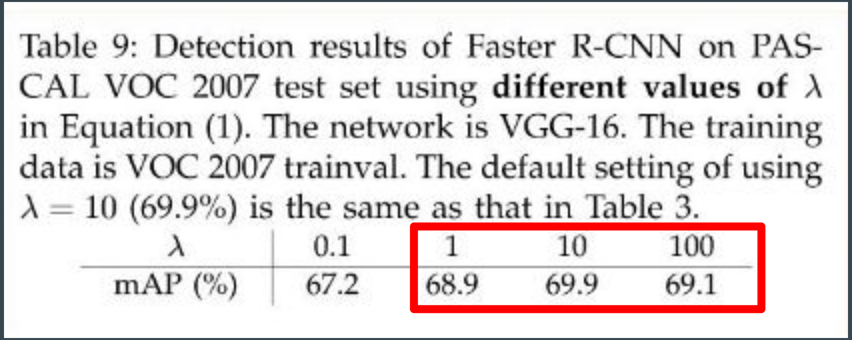
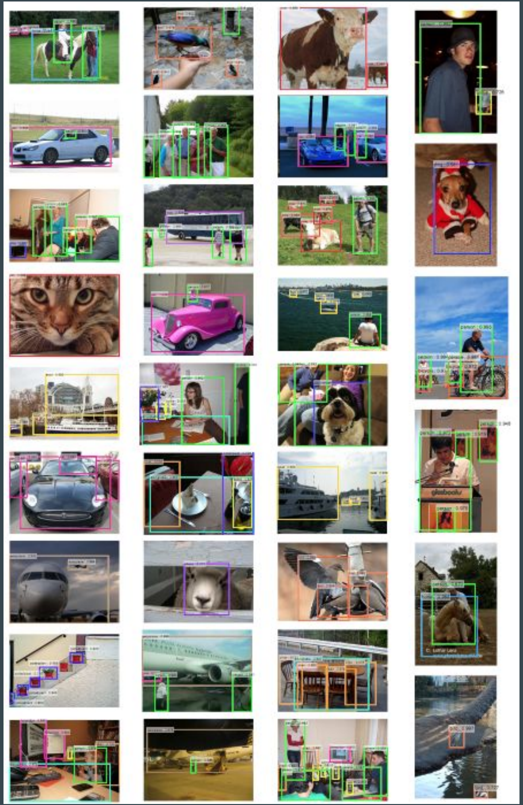


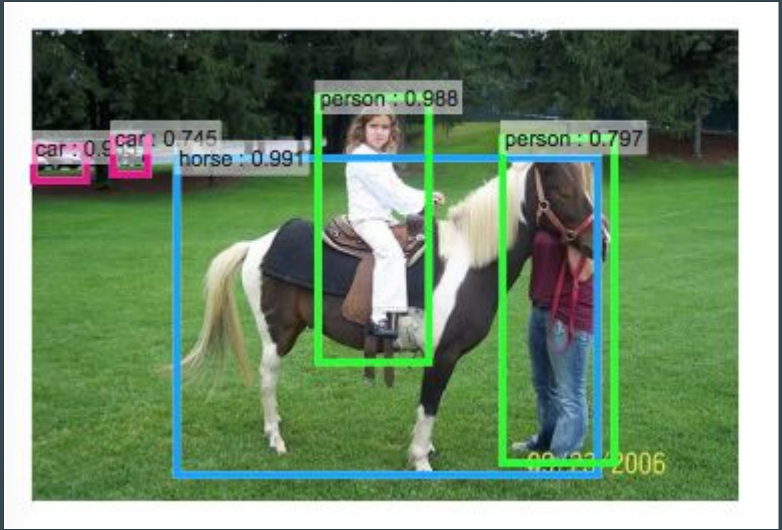
Table 9: Detection results of Faster R-CNN on PASCAL VOC 2007 test set using **different values of λ** in Equation (1). The network is VGG-16. The training data is VOC 2007 trainval. The default setting of using $\lambda = 10$ (69.9%) is the same as that in Table 3.

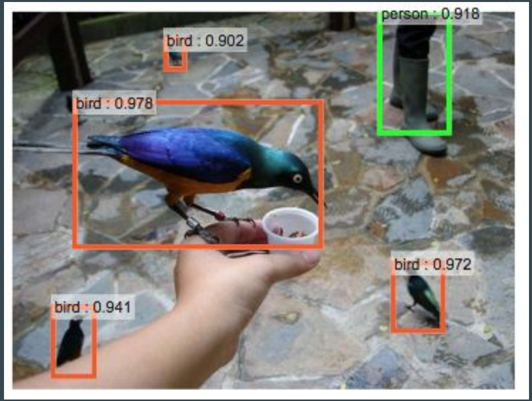
λ	0.1	1	10	100
mAP (%)	67.2	68.9	69.9	69.1

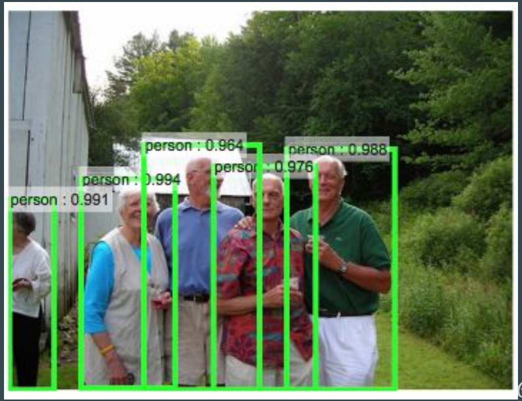
$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \boxed{\lambda} \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

5. CONCLUSION









5. CONCLUSION

실험결과에서 보이는 것처럼 약간의 정확도가 향상되었고, 실행시간이 현격히 줄어들었다. 하지만, 아직 실시간 영상처리 등에서 사용하기에는 다소 부족한 부분이 있는 것으로 보인다.

Further More

- YOLO
- SPP-Net
- Mask R-CNN

Reference

1. DenseNet
    a. https://arxiv.org/pdf/1608.06993.pdf
    b.
2. Faster R-CNN
    a. https://arxiv.org/pdf/1506.01497.pdf
    b. https://zzsza.github.io/data/2018/05/09/Faster-RCNN-review/
    c. https://tensorflow.blog/tag/fast-r-cnn/
    d. incredible.ai/deep-learning/2018/03/17/Faster-R-CNN/
    e. https://medium.com/@lsrock125/faster-r-cnn-%EB%85%BC%EB%AC%B8%EC%9D%BD%EA%B8%B0-1a7fdc9c43ee
    f. https://github.com/rbgirshick/py-faster-rcnn
    g. https://youtu.be/kcPAGIgBGRs
    h.

Thanks