YOLOv4: Optimal Speed and Accuracy of Object Detection

Alexey Bochkovskiy*
alexeyab84@gmail.com

Chien-Yao Wang*
Institute of Information Science
Academia Sinica, Taiwan
kinyiu@iis.sinica.edu.tw

Hong-Yuan Mark Liao
Institute of Information Science
Academia Sinica, Taiwan
liao@iis.sinica.edu.tw

- ✓ Object Detection using Single GPU for Product
- ✓ Improvement using Human NAS
- ✓ Inheritance of YOLO Spirit

AI Lab

Chang Gi Moon

주요 약어

- **MiWRC : Multi-input-Weighted-Residual-Connections**
- **CSP: Cross-Stage-Partial-connections**
- **CmBN: Cross mini-Batch Normalization**
- **SAT: Self-Adversarial-Training**
- **CBN: Cross-iteration Batch Normalization**
- **PAN: Path Aggregation Network**
- **SAM: Spatial Attention Module**
- **GIoU: Generalized IoU**
- **DIoU: Distance IoU**
- **CIoU: Complete IoU**
- **SPP: Spatial Pyramid Pooling**
- **ASPP: Atrous Spatial Pyramid Pooling**
- **RFB: Receptive Field Block**
- **SE: Squeeze-and-Excitation**
- **FPN: Feature Pyramid Network**
- **SFAM: Scale-wise Feature Aggregation Module**
- **ASFF: Adaptively Spatial Feature Fusion**
- **BiFPN: Bidirectional Feature Pyramid Network**
- **NAS-FPN: Neural Architecture Search Feature Pyramid Network**
- **FCOS: Fully Convolutional One-Stage object detector**
- **DIoU NMS: Distance IoU Non Maximum Suppression**

YOLO Series(v1~v3): Quick Glance

1. YOLOv1(CVPR 2016): You Only Look Once: Unified, Real-Time Object Detection(https://arxiv.org/abs/1506.02640)

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

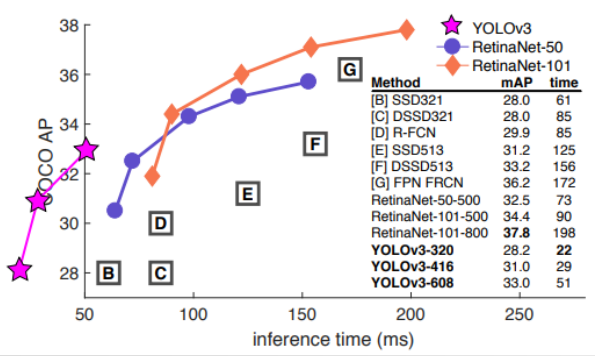
- 상당히 빠른 one stage 기반 detector
- object detection을 regression 문제로 생각
  - 이미지를 grid 로 분할 후 bounding boxes 좌표와 class의 확률을 같이 예측
- Modified GoogLeNet
- 단점
  - localization errors, low recall

2. YOLOv2(CVPR 2017): YOLO9000: Better, Faster, Stronger(https://arxiv.org/abs/1612.08242, best paper)

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

- YOLOv1에 다양한 기법들을 추가
- Better(v2)
  - Batch Normalization, High resolution classifier, Convolution with Anchor boxes, Dimension clusters, Direct location predictions, Fine-Grained Features, Multi-scale training
- Faster(v2)
  - Darknet-19, Training for classification, Training for detection
- Stronger(9000)
  - Hierarchical classification, Dataset combination with Word-tree, Joint classification and detection

3. YOLOv3(TECHNICAL REPORT 2018): YOLOv3: An Incremental Improvement(https://arxiv.org/abs/1804.02767)



- YOLOv2에 몇 가지 기법들을 더 추가
- 큰 차이점
  - Bounding box prediction: 각각의 ground truth object에 대해 단지 한개의 bounding box prior만 할당
  - Class prediction: Multilabel classification(considering hierarchical classes)
  - Prediction across scales: 각 scale마다 3개의 boxes를 예측하며, 3개의 scale을 이용
  - Darknet-53
  - 유사 FPN 아이디어 도입
- 성능
  - YOLOv2 대비 Better, Not Faster, Stronger(?)

YOLOv4: Abstract

- **YOLOv4(arXiv, 2020. 4. 23): YOLOv4: Optimal Speed and Accuracy of Object Detection(https://arxiv.org/abs/2004.10934)**

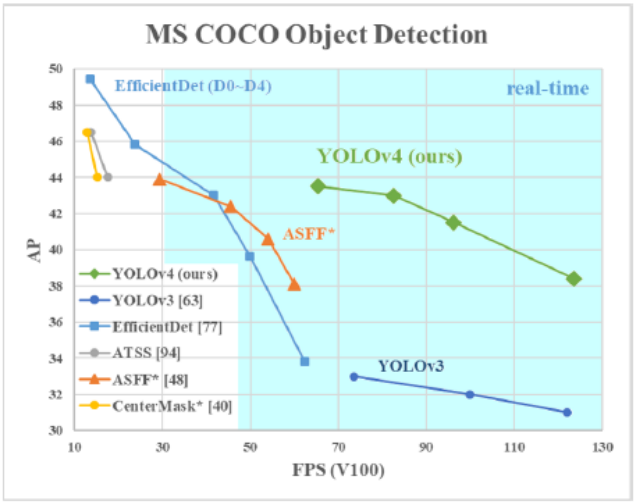
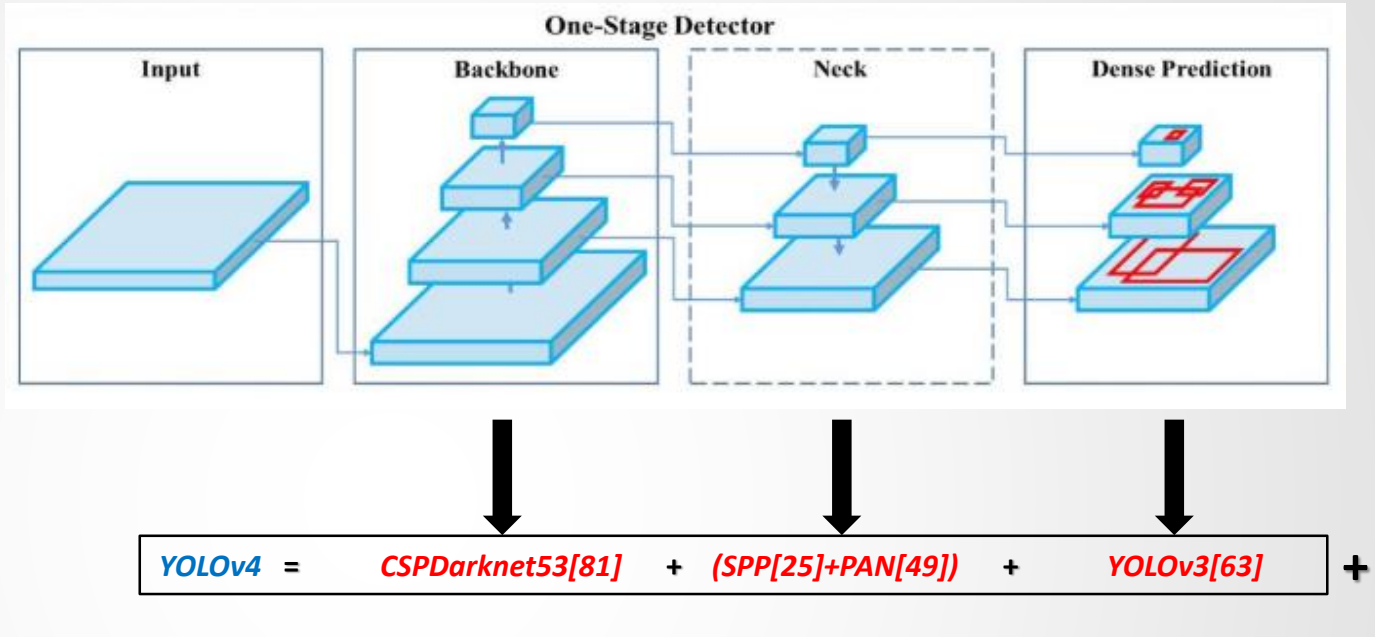


Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. YOLOv4 runs twice faster than EfficientDet with comparable performance. Improves YOLOv3's AP and FPS by 10% and 12%, respectively.



- ✓ CNN의 정확도 개선을 위한 수많은 features들이 존재
- ✓ 몇몇의 features들은 특정한 model, problem에 국한되어 제한적으로 동작
- ✓ WRC, CSP, CmBN, SAT, Mish 등의 universal features들도 존재
- ✓ 새로운 features들을 조합하여,
  43.5% AP(MS COCO 기준, AP_50: 65.7%), ~65 FPS 달성(Tesla V100)
- 결과 동영상: https://youtu.be/1_SiUOYUoOI

We demonstrate the nuances of comparing and using neural networks to detect objects. Our goal was to develop an algorithm for use in real production, and not just for moving science forward. Accuracy of YOLOv4 (608x608) — 43.5% AP / 65.7% AP50 Microsoft-COCO-testdev.

62 FPS — YOLOv4 (608x608 batch=1) on Tesla V100 — by using Darknet-framework

400 FPS — YOLOv4 (416x416 batch=4) on RTX 2080 Ti — by using TensorRT+tkDNN

32 FPS — YOLOv4 (416x416 batch=1) on Jetson AGX Xavier — by using TensorRT+tkDNN

As can be seen from the charts, in Real-time systems with FPS 30 or more:

- for YOLOv4–608 it is necessary to use RTX 2070, at **$ 450** (34 FPS), with an accuracy of **43.5%** AP / **65.7%** AP50
- for EfficientDet-D2, it is necessary to use TitanV, at **$ 2250** (42 FPS), with an accuracy of **43.0%** AP / **62.3%** AP50
- for EfficientDet-D0, it is necessary to use RTX 2070, at **$ 450** (34 FPS), with an accuracy of **33.8%** AP / **52.2%** AP50

I.e. YOLOv4 requires **5 times less expensive equipment** and yet is more accurate than EfficientDet-D2 (Google-TensorFlow). You can use EfficientDet-D0 (Google-TensorFlow) on cheap equipment, but then the accuracy will be 10% AP lower.

목차

1. Introduction

- **대부분의 CNN 기반 object detector는 정확도와 속도 사이의 Trade-Off 문제가 존재**
  - ✓ 주차 가능 공간 탐색: 느리지만 정확한 model들이 사용
  - ✓ 차량 추돌 경고: 빠르지만 부정확한 model들이 사용

- **정확도와 실시간 속도를 모두 만족하는 object detector가 필요**
  - ✓ 최신의 정확한 neural network들은 비실시간으로 동작, 큰 mini-batch 크기로 여러 개의 GPU를 이용한 훈련이 필요
  - ✓ 기존 GPU에서 실시간으로 동작하며, 훈련 시 1개의 GPU만 필요한 CNN을 제안하여 위와 같은 문제를 해결

- **본 논문의 목적**
  - ✓ 누구나 그림 1처럼 실시간/고품질의 결과를 얻을 수 있도록, 빠른 속도로 동작하는 object detector를 고안

- **본 논문의 기여**
  1. 효율적이며 강력한 model을 개발: 1080 Ti 또는 2080 Ti로 상당히 빠른 속도의 훈련 가능
  2. detector 훈련 시 사용 가능한 최신의 BoF 및 BoS 기법들이 주는 결과에 미치는 영향 분석
  3. 단일 GPU 훈련에 효율적이며 적합하도록, 최신의 기법들(CBN [89], PAN [49], SAM [85])을 수정

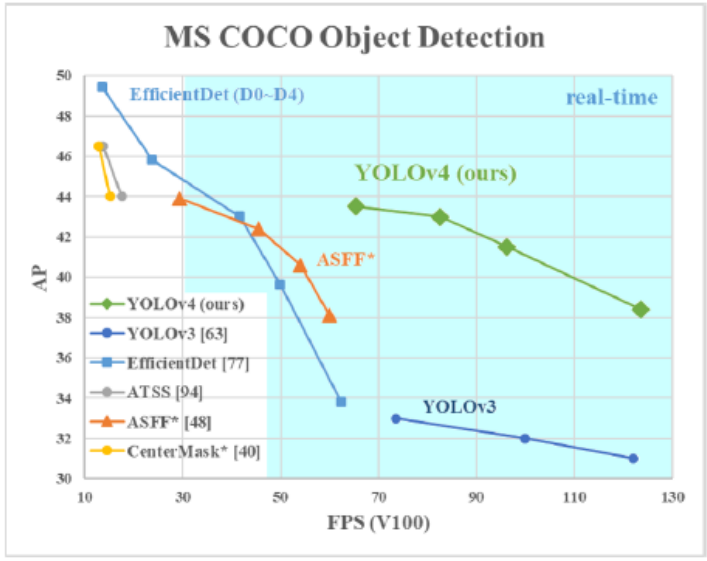


Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. YOLOv4 runs twice faster than EfficientDet with comparable performance. Improves YOLOv3's AP and FPS by 10% and 12%, respectively.

- ✓ EfficientDet-D2와 AP(43)는 유사하며 FPS는 2배(43→86) 빠름
- ✓ YOLOv3 대비 AP는 10%, FPS는 12% 개선됨

2. Related Work: *1. Object detection models*

- **일반적인 object detector의 구성 요소 및 종류**

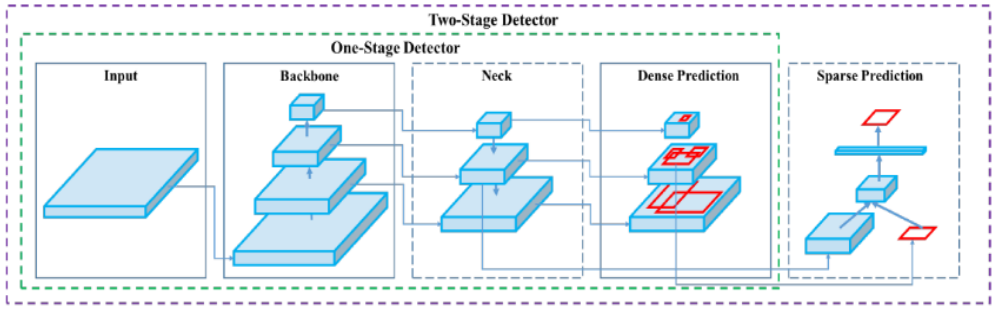


Figure 2: Object detector.

- **Input**: Image, Patches, Image Pyramid
- **Backbones**: VGG16 [68], ResNet-50 [26], SpineNet [12], EfficientNet-B0/B7 [75], CSPResNeXt50 [81], CSPDarknet53 [81]
- **Neck**:
  - **Additional blocks**: SPP [25], ASPP [5], RFB [47], SAM [85]
  - **Path-aggregation blocks**: FPN [44], PAN [49], NAS-FPN [17], Fully-connected FPN, BiFPN [77], ASFF [48], SFAM [98]
- **Heads:**:
  - **Dense Prediction (one-stage)**:
    - RPN [64], SSD [50], YOLO [61], RetinaNet [45] (anchor based)
    - CornerNet [37], CenterNet [13], MatrixNet [60], FCOS [78] (anchor free)
  - **Sparse Prediction (two-stage)**:
    - Faster R-CNN [64], R-FCN [9], Mask R-CNN [23] (anchor based)
    - RepPoints [87] (anchor free)

1. Backbone:

- 일반적으로 ImageNet을 이용하여 pre-trained됨
- For GPU: VGG [68], ResNet [26], ResNeXt [86], DenseNet [30]
- For CPU: SqueezeNet [31], MobileNet[28, 66, 27, 74], ShuffleNet [97, 53]

2. Neck:

- backbone과 head 사이에 삽입된 layers들로서, 서로 다른 stages들로부터 온 feature map을 모으는데 사용
- FPN [44], PAN [49], BiFPN [77], NAS-FPN [17]

3. Head:

- object에 대한 class와 bounding boxes를 예측하는데 사용
- 대표적 one-stage detector
  - anchor-based: YOLO [61, 62, 63], SSD [50], RetinaNet [45]
  - anchor-free: CenterNet [13], CornerNet [37, 38], FCOS [78]
- 대표적 two-stage detector
  - anchor-based: R-CNN [19] 시리즈(fast R-CNN [18], faster R-CNN [64], R-FCN [9], Libra R-CNN [58])
  - anchor-free: RedPoints [87]

4. Others:

- object detection 목적으로 새로운 backbone 개발 또는 전체 model 구축
- 새로운 backbone: DetNet [43], DetNAS [7]
- 새롭게 전체 model 구축: SpineNet [12], HitDetector [20]

2. Related Work: *2. Bag of Freebies*

- **"Bag of Freebies"란?**
  - ✓ 훈련 전략만 바꾸거나 훈련 비용만 증가시켜, object detector의 정확도를 높이는 방법들을 의미

종류	기법들
data augmentation	광학적 / 기하학적 왜곡: brightness, contrast, hue, saturation, noise / random scaling, cropping, flipping, rotating
	occlusion 문제 대응: random erase [100], CutOut [11], hide-and-seek [69], grid mask [6]
	여러 이미지를 융합: MixUp [92], CutMix [91]
	GAN 이용: style transfer GAN [15]
regularization	DropOut [71], DropConnect [80], DropBlock [16]
imbalance sampling	class 간의 데이터 비율 불균형: 1) one-stage detector: focal loss [45] 2) two-stage detector: hard negative example mining [72], online hard example mining [67]
	categories 간의 연관 정도에 대한 불균형: class labeling smoothing [73], knowledge distillation를 이용한 label refinement network [33]
objective function	기존 방식: bounding boxes의 좌표를 직접적으로 regression
	개선 방식: IoU loss [90], GIoU loss [65], DIoU loss [99], CIoU loss [99]

2. Related Work: *3. Bag of Specials*

- **"Bag of Specials"란?**
  - ✓ 추론 비용을 약간만 증가시켜, object detection의 정확도를 크게 향상시키는 방법들을 의미
  - ✓ plugin modules과 post-processing으로 구성

구성	종류	기법들
plugin modules	receptive field enhancement	SPP [25], ASPP [5], RFB [47]
	attention model	channel-wise attention model: SE [29]
		point-wise attention model: SAM [85]
	feature integration	초기 방식: skip connection [51], hyper-column [22], FPN [44]
		FPN 이후: SFAM [98], ASFF [48], BiFPN [77]
	activation function	초기 방식: ReLU [56]
		ReLU 이후: LReLU [54], PReLU [24], ReLU6 [28], Scaled Exponential Linear Unit (SELU) [35], Swish [59], hard-Swish [27], Mish [55]
post-processing	기존 방식: NMS 적용(without context information)	
	R-CNN [19]: classification confidence score를 기준으로 높은 score에서 낮은 score로 greedy NMS를 수행	
	R-CNN 이후: soft NMS [1], DIoU NMS [99]	

3. Methodology: *1. Selection of architecture*

- **Goals**
  - ✓ network 입력 해상도, convolutional layer 수, parameter 수, layer 출력 수 가운데서 최적의 balance를 발견
  - ✓ backbone level과 detector level 사이의 receptive field 향상을 위한 최상의 additional blocks 및 parameter aggregation 기법 선택

- **detector 성능 향상을 위한 고려 사항**
  - ✓ classification을 위한 model이 detector에도 최적은 아니며, 더 많은 고려 사항이 필요
    - ➢ 더 큰 크기의 network 입력 해상도: 작은 크기를 갖는 여러 object들의 검출을 위해 필요
    - ➢ 더 많은 layer 수: 증가된 network 입력 해상도를 수용할 수 있는 더 큰 receptive field가 필요
    - ➢ 더 많은 parameter 수: 한장의 이미지에서 서로 다른 크기의 여러 objects들을 검출하기 위해, model은 더 많은 capacity가 필요

- **receptive field가 크며, parameter 수가 더 많은 model을 backbone으로 선택하는 것이 유리**

Table 1: Parameters of neural networks for image classification.

Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPResNext50	512x512	425x425	20.6 M	**1058 K**	31 (15.5 FMA)	62
CSPDarknet53	512x512	725x725	**27.6 M**	950 K	52 (26.0 FMA)	**66**
EfficientNet-B3 (ours)	512x512	**1311x1311**	12.0 M	668 K	11 (5.5 FMA)	26

  - ✓ CSPDarknet53은 다른 backbone model들보다 receptive field의 크기가 크고, parameter 수가 많고 BFLOPs가 크지만, FPS는 가장 빠름
  - ✓ CSPDarknet53은 detector를 위한 최적의 backbone임을 나타냄

- **CSPDarknet53에 additional blocks으로 SPP block을 추가**
  - ✓ receptive field를 상당히 향상
  - ✓ context features를 분리
  - ✓ network의 동작 속도 저하 없음

- **CSPDarknet53에 parameter aggregation 기법으로 PAN을 이용**
  - ✓ 참고: YOLOv3에서는 유사 FPN을 이용

- **최종적으로 선택된 기법들**

  1. backbone: ***CSPDarknet53***
  2. neck:
     - additional blocks: ***SPP***
     - path-aggregation blocks: ***PAN***
  3. head: ***YOLOv3(anchor-based)***

  ✓ Cross-GPU Batch Normalization 또는 고가의 특수 장비를 사용하지 않으므로, 누구나 하나의 GPU로 최신의 결과를 재연 가능

3. Methodology: *2. Selection of BoF and BoS*

- **object detection의 훈련 개선을 위한 기법들**

  1) 선택된 BoF 후보 features들

  - **bounding box regression loss**: *MSE, IoU, GIoU,* ***CIoU****, DIoU*
  - **data augmentation:** *CutOut, MixUp,* ***CutMix***
  - **regularization method**: ~~*DropOut, DropPath [36], Spatial DropOut [79],*~~ ***DropBlock***
    - *DropBlock*의 저자들이 YOLOv4를 이용하여 다른 방법들과 비교했을 때, 우수한 성능을 보인다는 것을 입증

  2) 선택된 BoS 후보 features들

  - **activations:** *ReLU, leaky-ReLU,* ~~*parametric ReLU, ReLU6, SELU*~~*, Swish,* ***Mish***
    - *PReLU*와 *SELU*는 훈련이 많이 어려우므로 제외
    - *ReLU6*는 특별히 quantization network을 위해 고안되었으므로 제외
  - **normalization of the network activations:** *Batch Normalization (BN) [32],* ~~*Cross-GPU Batch Normalization (CGBN 또는 SyncBN) [93],*~~ *Filter Response Normalization (FRN) [70], Cross-iteration Batch Normalization (CBN) [89]*
    - 하나의 GPU만을 이용한 훈련 전략에 중점을 두고 있으므로, syncBN은 제외
  - **skip-connections:** *Residual connections, Weighted residual connections,* ***Multi-input weighted residual connections*****,** ***Cross stage partial connections (CSP)***

3. Methodology: *3. Additional improvements*

- **1개의 GPU를 이용한 훈련에 적합하도록, 추가적인 설계 및 개선을 수행**
  - 새로운 data augmentation 기법 도입
    - ✓ ***Mosaic, SAT***
  - genetic algorithms을 적용하여 최적의 hyper-parameters를 선택
  - 효율적인 훈련 및 검출을 위해 기존 기법들을 수정
    - ✓ ***modified SAM, modified PAN, CmBN***

Figure 3: Mosaic represents a new method of data augmentation.

- **BoF: 새롭게 도입한 data augmentation 기법들**
  - ***Mosaic***
    - ➢ CutMix는 단지 2개의 입력 이미지들만 mix, Mosaic은 4개의 훈련 이미지들을 1개로 mix
    - ➢ batch normalization은 각 layer 상에서 서로 다른 4개의 이미지들에 대한 activation statistic를 계산 가능
      - ✓ 따라서, 큰 크기의 mini-batch에 대한 필요성을 상당히 줄일 수 있음
  - ***SAT(Self-Adversarial Training)***
    - ➢ 2단계의 forward 및 backward 단계로 동작
    - ➢ 1단계: neural network는 network의 weight 대신에 원본 이미지를 변경
      - ✓ 이런 식으로 neural network는 자체적으로 adversarial attack을 수행하여, 이미지에 원하는 object가 없다는 속임수를 만들도록 원본 이미지를 변경
    - ➢ 2단계: neural network은 변경된 이미지에서 정상적인 방식으로 object를 검출 하도록 훈련됨
    - ➢ 관련 내용: https://github.com/AlexeyAB/darknet/issues/5117

- **BoS**: ***CmBN(Cross mini-Batch Normalization)***
  - CBN의 변경된 버전으로서, 단일 batch 내에서 mini-batches 사이에 대한 statistic을 수집
  - BN은 batch size가 작을 경우 examples들에 대한 정확한 statistic estimation이 어려우므로 효율성이 저하
    - ✓ CBN의 경우 estimation quality의 향상을 위해, 이전 iteration들의 statistic를 함께 활용

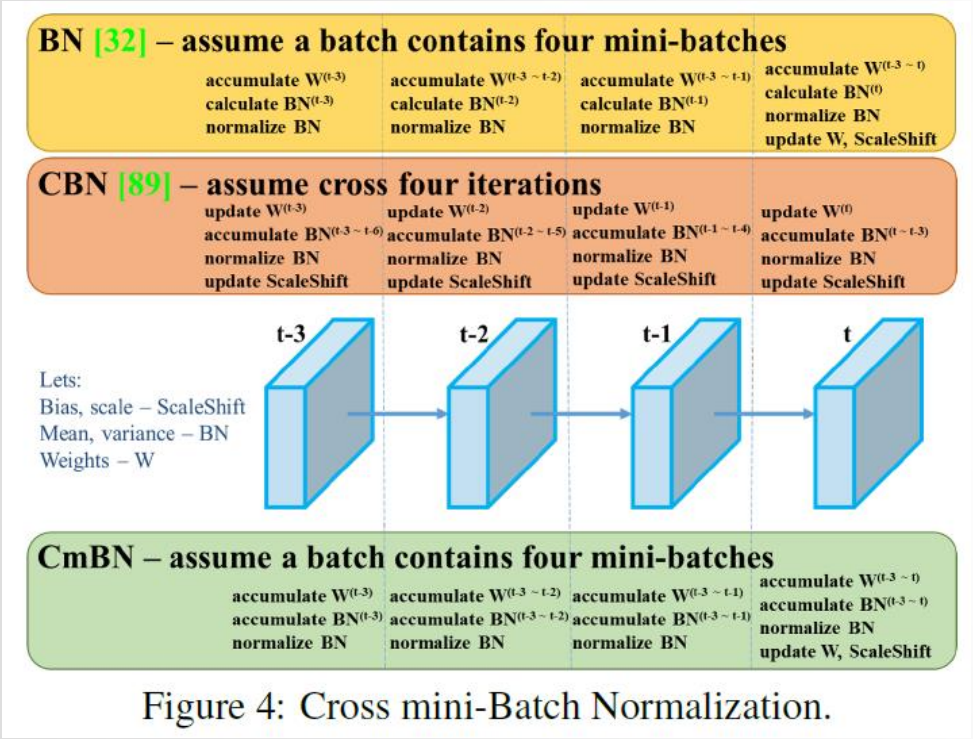


Figure 4: Cross mini-Batch Normalization.

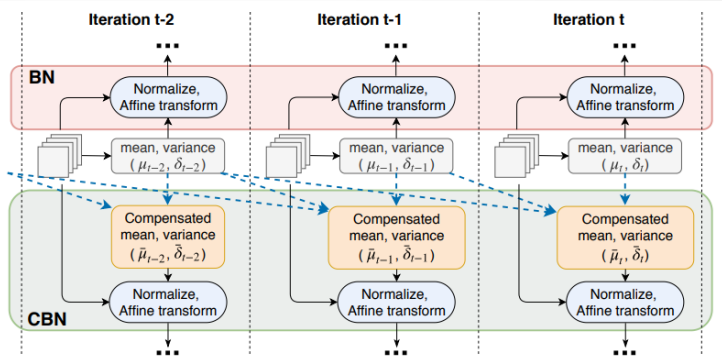


Figure 2. Illustration of BN and the proposed Cross-Iteration Batch Normalization (CBN).

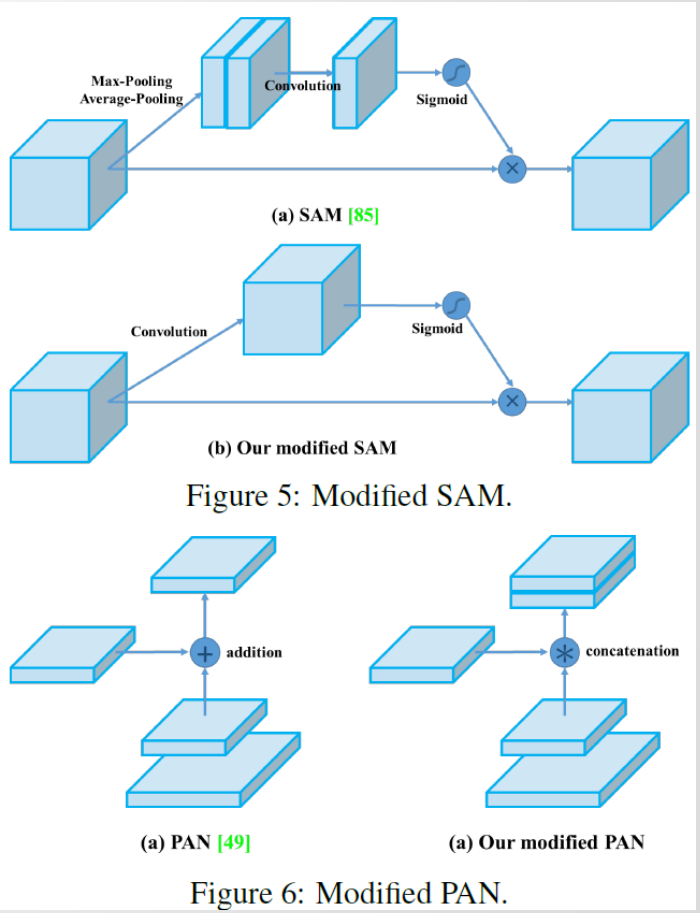


Figure 5: Modified SAM.

Figure 6: Modified PAN.

- **BoS**: ***modified SAM(Spatial Attention Module), modified PAN(Path Aggregation Network)***
  - SAM을 spatial-wise attention에서 point-wise attention으로 변경
  - PAN의 shortcut connection을 concatenation으로 교체

3. Methodology: *4. YOLOv4*

- YOLOv4의 구성 및 사용 기법들

YOLOv4 consists of:

- Backbone: CSPDarknet53 [81]
- Neck: SPP [25], PAN [49]
- Head: YOLOv3 [63]

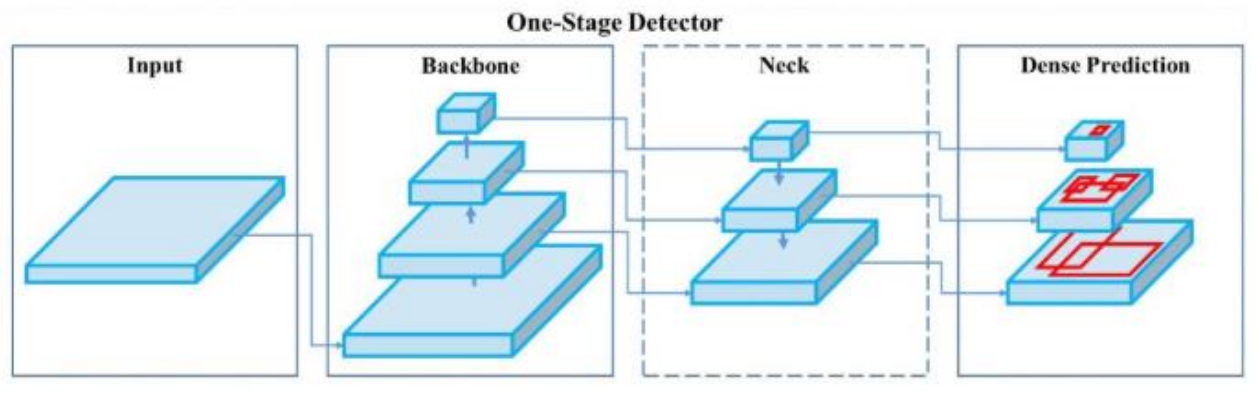
YOLO v4 uses:

- Bag of Freebies (BoF) for backbone: CutMix and Mosaic data augmentation, DropBlock regularization, Class label smoothing
- Bag of Specials (BoS) for backbone: Mish activation, Cross-stage partial connections (CSP), Multi-input weighted residual connections (MiWRC)
- Bag of Freebies (BoF) for detector: CIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for a single ground truth, Cosine annealing scheduler [52], Optimal hyper-parameters, Random training shapes
- Bag of Specials (BoS) for detector: Mish activation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS

정보가 너무 부족함ㅠ

YOLOv4의 구성 요소와
사용 기법(관련 논문 포함)들을
구체적으로 살펴보면!

구성 요소 및 사용한 기법들



YOLOv4 = CSPDarknet53[81] + (SPP[25]+PAN[49]) + YOLOv3[63] +

BoF:

1. for backbone
   - data augmentation: *CutMix [91], Mosaic*
   - imbalance sampling: *Class labeling smoothing [73]*
   - regularization: *DropBlock [16]*

2. for detector
   - objective function: *CIoU-loss [99]*
   - normalization of network activation: *CmBN*
   - regularization: *DropBlock [16]*
   - data augmentation: *Mosaic*, *Self-Adversarial Training*
   - hyper-parameters optimization: *Genetic algorithms*
   - learning rate scheduler: *Cosine annealing scheduler [52]*
   - others:
     - *Eliminate grid sensitivity*
     - *Using multiple anchors for a single ground truth*
     - *Random training shapes*

BoS:

1. for backbone
   - activation function: *Mish [55]*
   - skip-connection: *CSP [81]*, *MiWRC [?]*

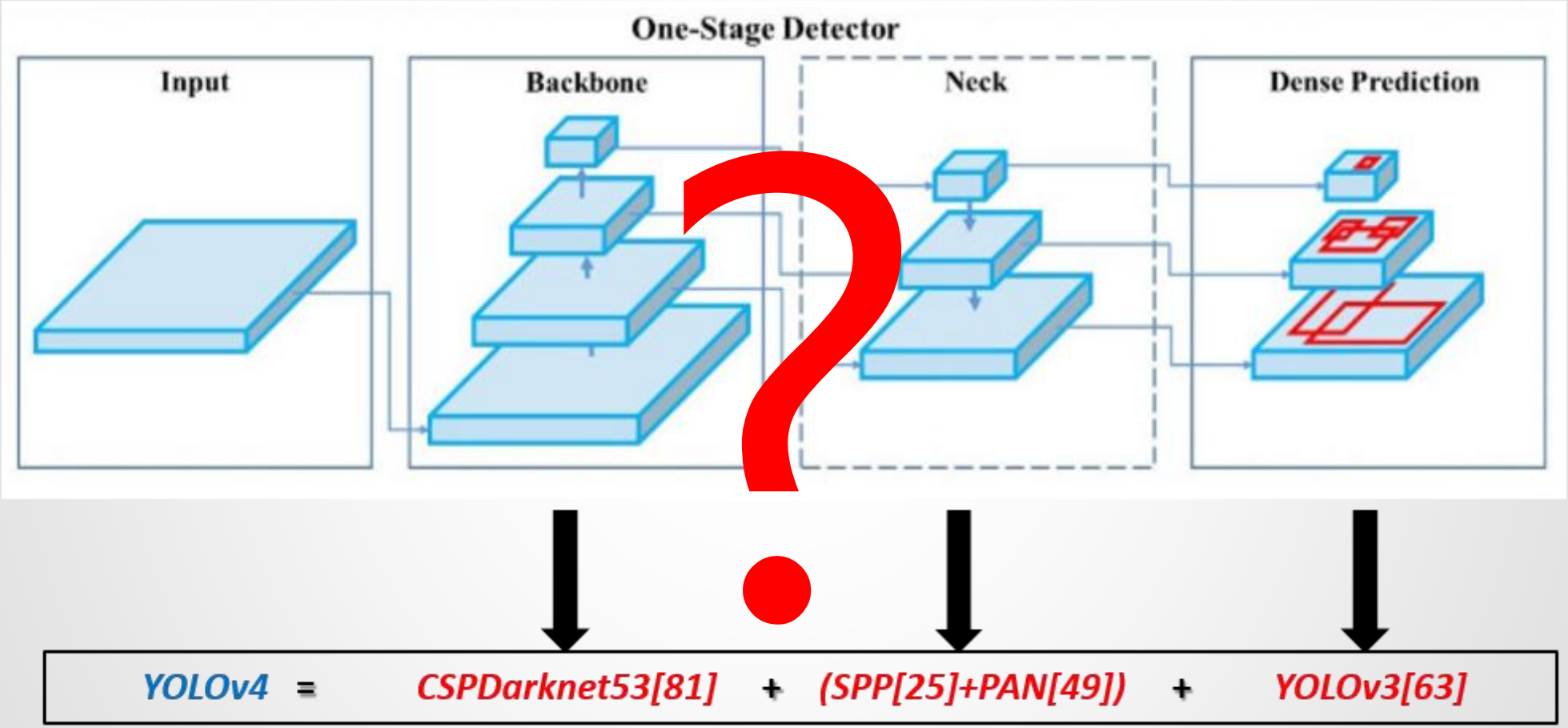
2. for detector
   - activation function: *Mish [55]*
   - receptive field enhancement: *SPP [25]*
   - attention: *SAM [85](Modified)*
   - feature integration: *PAN [49](Modified)*
   - post-processing: *DIoU NMS [99]*

※ Red: Existing, Purple: YOLOv4, Green: Unknown

관련 논문(Existing) 목록

- CSP and CSPDarknet53 [81]: Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CSPNet: A new backbone that can enhance learning capability of cnn. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPR Workshop), 2020. 2, 7
- SPP [25]: Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 37(9):1904–1916, 2015. 2, 4, 7
- PAN [49]: Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 8759–8768, 2018. 1, 2, 7
- YOLOv3 [63]: Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018. 2, 4, 7, 11
- CutMix [91]: Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. CutMix: Regularization strategy to train strong classifiers with localizable features. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), pages 6023–6032, 2019.3
- Class labeling smoothing [73]: Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), pages 2818–2826, 2016. 3
- DropBlock [16]: Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. DropBlock: A regularization method for convolutional networks. In Advances in Neural Information Processing Systems (NIPS), pages 10727–10737, 2018. 3
- CIoU-loss and DIoU NMS [99]: Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-IoU Loss: Faster and better learning for bounding box regression. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2020. 3, 4
- Cosine annealing scheduler [52]: Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. International Conference on Learning Representations(ICLR) 2017
- Mish [55]: Diganta Misra. Mish: A self regularized nonmonotonic neural activation function. arXiv preprint arXiv:1908.08681, 2019. 4
- SAM [85]: Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM: Convolutional block attention module. In Proceedings of the European Conference on Computer Vision (ECCV), pages 3–19, 2018. 1, 2, 4

YOLOv4의 구성 요소



Backbone: *CSPDarknet53(CSP connections + YOLOv3 backbone(Darknet53))*

- **DenseBlock and DenseNet(Densely Connected Convolutional Networks, CVPR 2017 best paper)**
  - image classification의 정확도 향상을 위해서는?
    - receptive field 확장, model의 complexity를 증가, 쉬운 훈련을 위한 skip-connection의 도입 등이 필요
    - 높은 수준의 interconnected된 layers를 이용하여 위의 개념을 확장 가능

  - DenseBlock
    - 여러 개의 convolutions layers들을 포함하며, 각 layer H_i 는 "batch normalization ➔ ReLU ➔ convolution" 순으로 구성
    - H_i 는 마지막 layer의 출력뿐만 아니라, 모든 이전 layers들의 출력을 입력으로 사용

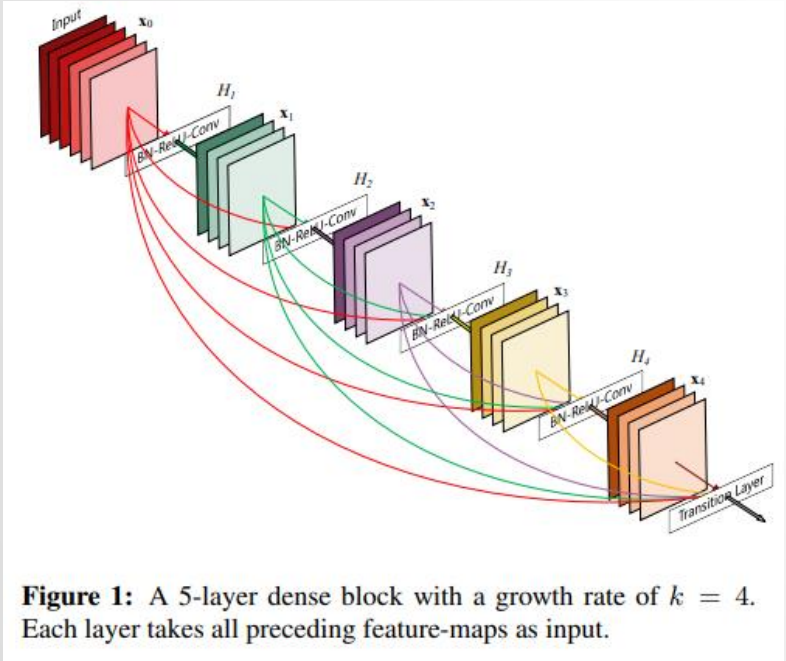


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

  - DenseNet
    - convolution과 pooling으로 구성된 transition layer를 DenseBlock 사이에 배치하여 DenseNet을 구성

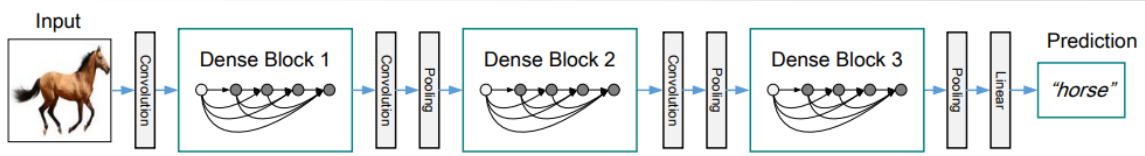


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

    - vanishing gradient 개선, feature propagation 강화, feature 재활용, parameter 수 절약

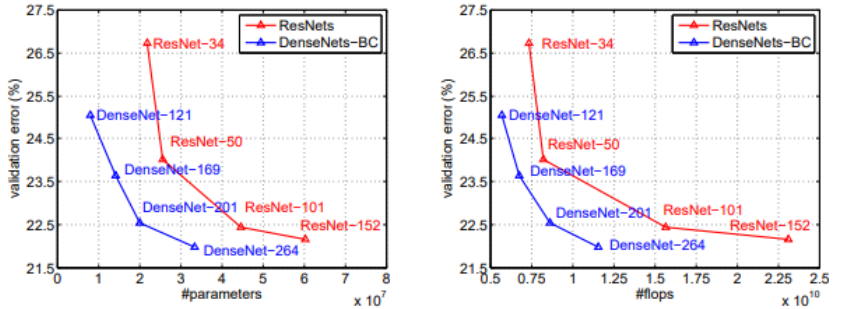
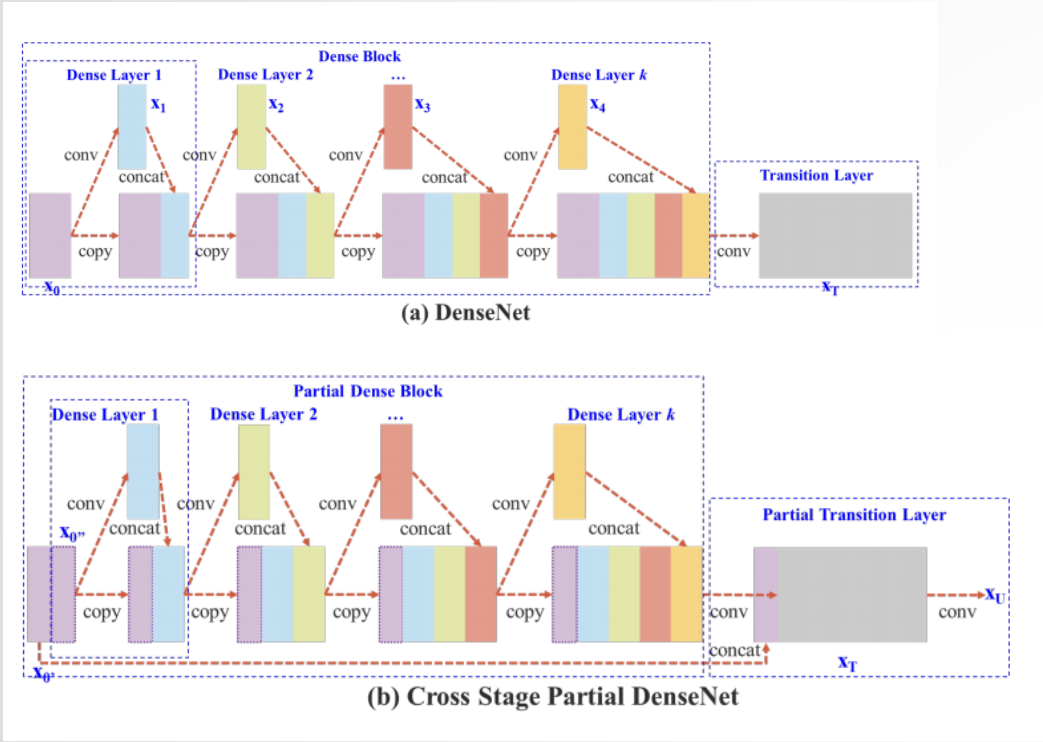


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

✓ ResNet보다 파라미터량 및 계산 비용이 상당히 적음에도 유사한 성능 보유
- 예: DenseNet-201은 ResNet-101보다 파라미터량이 25M 적음(45➔20)에도 error rate는 유사(22.5)

- **CSP(CSPNet: A new backbone that can enhance learning capability of cnn, CVPRW 2020)**
  - Base layer의 feature map x_0를 x_0', x_0"의 2개 part로 분리
  - x_0'은 DenseBlock을 건너뛰고 다음 transition layer의 입력으로 사용
  - x_0"은 DenseBlock 및 transition layer를 통과



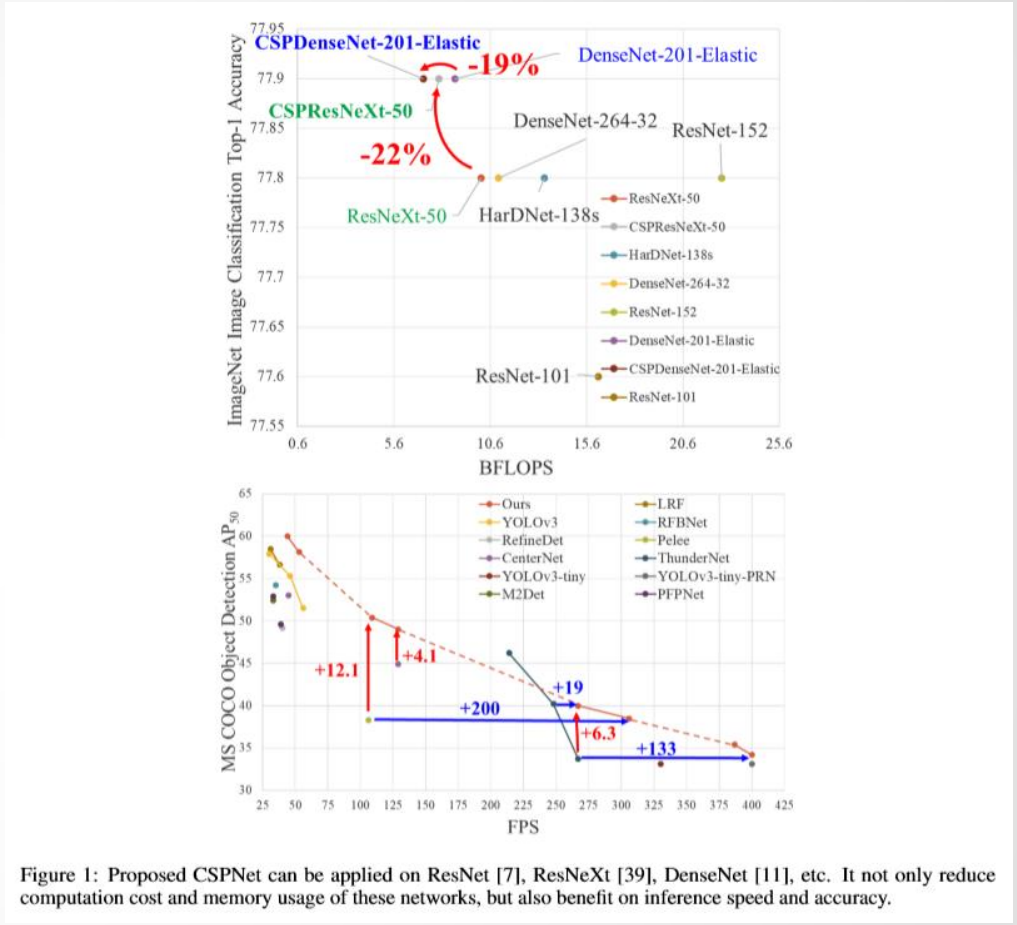
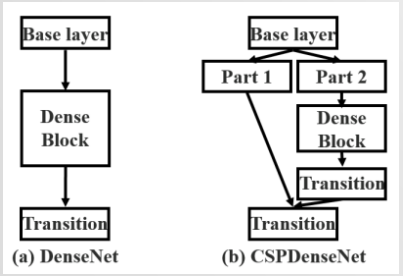


Figure 1: Proposed CSPNet can be applied on ResNet [7], ResNeXt [39], DenseNet [11], etc. It not only reduce computation cost and memory usage of these networks, but also benefit on inference speed and accuracy.



- CNN의 learning ability 강화
- 계산 bottleneck 제거
- 메모리 비용 절감

- CSPNet을 ResNet, ResNetXt, DenseNet 등에 적용 결과
  - ✓ 네트워크의 계산 비용 및 메모리 사용량 감소
    - 예: CSPDenseNet-201-Elastic은 DenseNet-201-Elastic보다 유사한 accuracy(78)를 보이면서도 계산 비용을 19% 줄임
  - ✓ 추론 속도 및 정확도 향상
    - 예: 유사한 AP(34)를 보이는 M2Det(FPS: 265) 보다 FPS가 133 빠름, 유사한 FPS 100을 보이는 Pelee보다 AP가 12.1 우수

- **Darknet-53(YOLOv3: An incremental improvement, technical report 2018)**
  - YOLOv2의 Darknet-19를 개선
  - 53개의 convolution layers를 사용
  - Residual Block 적용
  - ResNet-101이나 ResNet-152보다 좀 더 효율적

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Table 2. **Comparison of backbones.** Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

  - ✓ ResNet-101보다 accuracy는 미세하게 우수하며(+0.1), FPS는 1.5배(53→78) 빠름
  - ✓ ResNet-152보다 accuracy는 미세하게 나쁘지만(-0.4), FPS는 2배(37→78) 빠름

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			

Table 6: Darknet-19.

Neck: *SPP + PAN(+SAM)*

- **Neck의 필요성**
  - ➢ 다양한 scale의 객체 검출을 위해, head로 전달되는 정보를 풍부하게 할 수 있도록 bottom-up path의 공간 및 top-down path의 의미 정보를 활용
    - receptive field enhancement를 위해 SPP를 modified해서 적용
    - attention을 위해 SAM을 modified해서 적용
    - feature integration을 위해 PAN을 modified해서 적용

- **SPP(Spatial pyramid pooling in deep convolutional networks for visual recognition, PAMI 2015)**
  - ➢ convolution layer를 통해 추출된 feature map을 n개의 피라미드를 이용하여 고정된 길이의 feature representation을 생성
    - FC layer가 포함된 CNN model들은 특정 차원의 입력 이미지만 허용하나, SPP는 서로 다른 크기의 입력 이미지를 허용 가능
    - feature map을 m x n 크기의 bin으로 분할 후, 각 bin에 대해 max pooling을 수행된 결과들을 concatenation하여 FC layer로 전달
    - 공간 정보가 중요한 이미지 분할 등에 유용한 설계를 제공

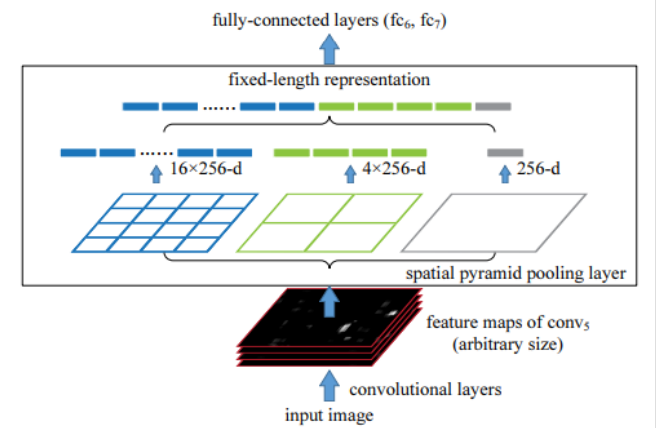


Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the $conv_5$ layer, and $conv_5$ is the last convolutional layer.

- **PAN(Path aggregation network for instance segmentation, CVPR 2018)**
  - ➢ layer들 사이의 정보 흐름의 중요성을 부각하여 설계된 model로서, path augmentation ➔ pooling ➔ fusion의 단계로 구성
    - 기존의 FPN을 backbone으로 이용, layer가 깊을수록 최하단 layer의 low-level feature가 최상단 layer까지 propagation이 잘 되지 않는 현상을 short-cut path를 도입하여 개선

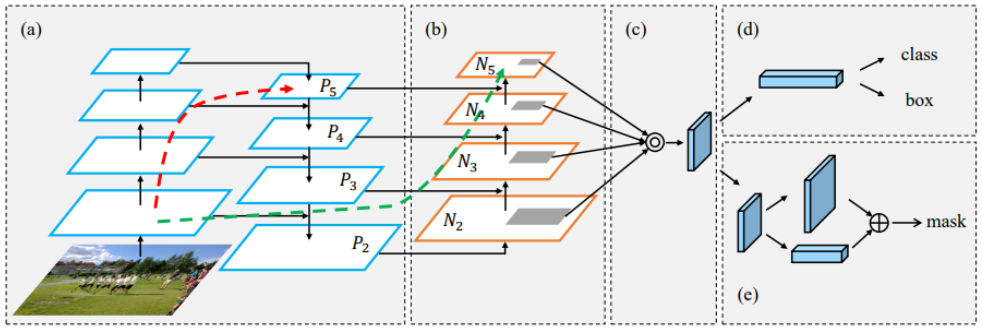


Figure 1. Illustration of our framework. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling. (d) Box branch. (e) Fully-connected fusion. Note that we omit channel dimension of feature maps in (a) and (b) for brevity.

- **SAM(CBAM: Convolutional block attention module, ECCV 2018)**
  - CBAM(Convolutional Block Attention Module)에서 Channel Attention과 함께 사용되는 Attention Module
    - Channel-refined feature map F′을 입력으로 받아 maximum pooling 및 average pooling을 개별 적용하여 2개의 feature map set을 생성
    - 해당 결과를 convolution layer를 통과시켜 sigmoid function을 적용 후 Spatial Attention M_s를 생성
    - 기존 network의 설계 요소와 orthogonal하므로, 상호 보완을 통해 적은 overhead로도 성능 향상 가능
    - Spatial Attention mask는 입력 feature에 적용되어 refined된 feature map을 출력

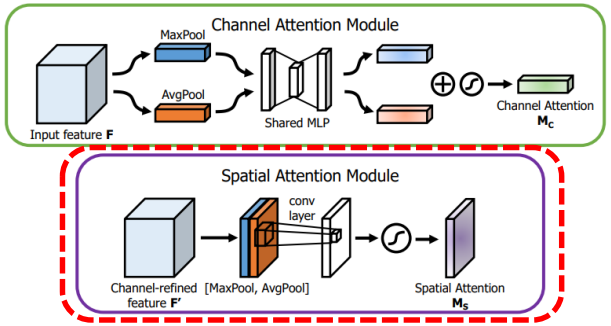


Fig. 2: **Diagram of each attention sub-module.** As illustrated, the channel sub-module utilizes both max-pooling outputs and average-pooling outputs with a shared network; the spatial sub-module utilizes similar two outputs that are pooled along the channel axis and forward them to a convolution layer.

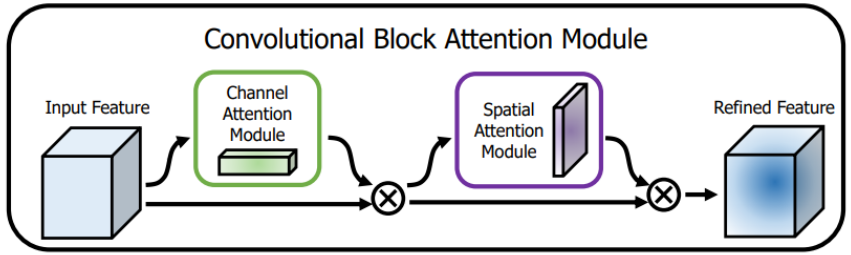


Fig. 1: **The overview of CBAM.** The module has two sequential sub-modules: *channel* and *spatial*. The intermediate feature map is adaptively refined through our module (CBAM) at every convolutional block of deep networks.

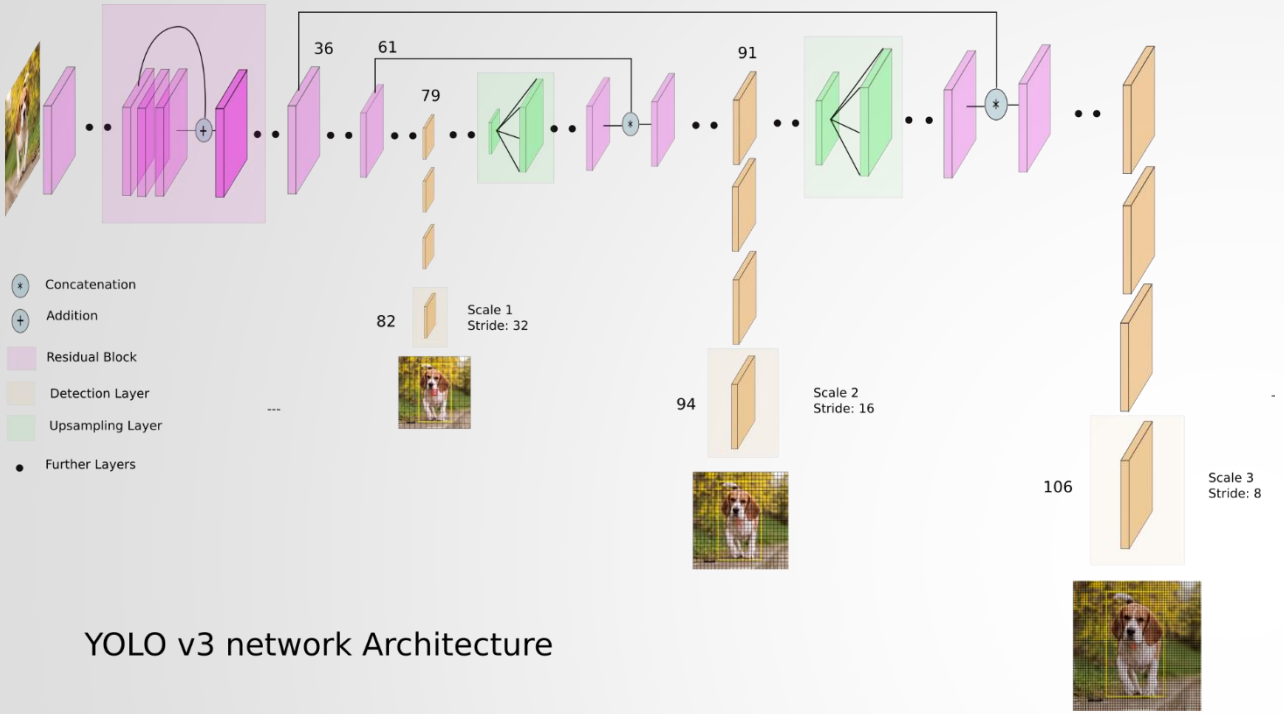
Description	Param.	GFLOPs	Top-1 Error(%)	Top-5 Error(%)
ResNet50 + channel (SE [28])	28.09M	3.860	23.14	6.70
ResNet50 + channel	28.09M	3.860	22.80	6.52
ResNet50 + channel + spatial (1x1 conv, k=3)	28.10M	3.862	22.96	6.64
ResNet50 + channel + spatial (1x1 conv, k=7)	28.10M	3.869	22.90	6.47
ResNet50 + channel + spatial (avg&max, k=3)	28.09M	3.863	22.68	6.41
ResNet50 + channel + spatial (avg&max, k=7)	28.09M	3.864	**22.66**	**6.31**

Table 2: **Comparison of different spatial attention methods.** Using the proposed channel-pooling (*i.e.* average- and max-pooling along the channel axis) along with the large kernel size of 7 for the following convolution operation performs best.

✓ 논문에는 SAM에 대해 ablation 테스트는 따로 수행하지 않음

✓ ResNet50 + SE block을 이용한 경우보다 제안한 방법이 동일한 파라미터량(28.09M)과 유사한 계산 비용(3.864)으로 우수한 error rate(Top-1 22.66)를 보임

Head: *YOLOv3*



YOLO v3 network Architecture

- **Backbone의 변화**
  - ✓ YOLOv2(Darknet-19) ➔ YOLOv3(Darknet-53)
- **3개의 scale을 고려하여 객체 검출 수행**
  - ✓ YOLOv2는 scale을 미고려
- **작은 크기의 객체 검출 성능 향상**
  - ✓ YOLOv2(AP_S: 5.0) ➔ YOLOv3(AP_S: 18.3)
- **더 많은 개수의 anchor boxes를 이용**
  - ✓ YOLOv2(5개) ➔ YOLOv3(9개)
- **많은 수의 bounding boxes를 사용**
  - ✓ YOLOv2(845개) ➔ YOLOv3(10,647개)
- **multi-label 분류를 고려하여 loss function을 변경**
  - ✓ YOLOv2(mean squared error) ➔ YOLOv3(logistic regression)

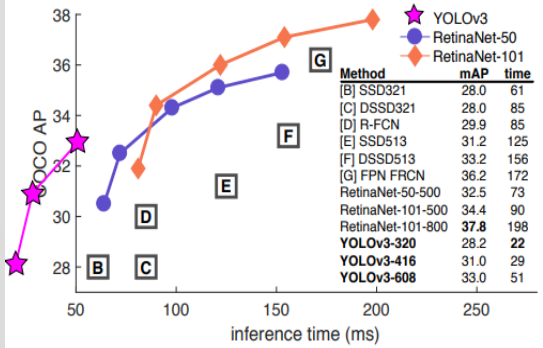


Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Table 3. I'm seriously just stealing all these tables from [9] they take soooo long to make from scratch. Ok, YOLOv3 is doing alright. Keep in mind that RetinaNet has like 3.8× longer to process an image. YOLOv3 is much better than SSD variants and comparable to state-of-the-art models on the AP_{50} metric.

- **YOLOv2 대비**
  - ✓ 속도는 느리지만, AP는 우수(21.6 ➔ 33.0)
- **RetinaNet 대비**
  - ✓ 속도는 3.8배 빠르지만, AP는 저하(39.1➔33.0)
- **다른 one-stage detector들 대비(RetinaNet 제외)**
  - ✓ 속도도 빠르며, AP는 다소 우수
  - ✓ AP_S 및 AP_M은 우수하나, AP_L은 저하

BoF:

1. for backbone
   - data augmentation: *CutMix [91]*, *Mosaic*
   - imbalance sampling: *Class labeling smoothing [73]*
   - regularization: *DropBlock [16]*

2. for detector
   - objective function: *CIoU-loss [99]*
   - normalization of network activation: *CmBN*
   - regularization: *DropBlock [16]*
   - data augmentation: *Mosaic*, *Self-Adversarial Training*
   - hyper-parameters optimization: *Genetic algorithms*
   - learning rate scheduler: *Cosine annealing scheduler [52]*
   - others:
     - *Eliminate grid sensitivity*
     - *Using multiple anchors for a single ground truth*
     - *Random training shapes*

BoS:

1. for backbone
   - activation function: *Mish [55]*
   - skip-connection: *CSP [81]*, *MiWRC [?]*

2. for detector
   - activation function: *Mish [55]*
   - receptive field enhancement: *SPP [25]*
   - attention: *SAM [85](Modified)*
   - feature integration: *PAN [49](Modified)*
   - post-processing: *DIoU NMS [99]*

※ Red: Existing, Purple: YOLOv4, Green: Unknown

YOLOv4에서 사용한 기법들

BoF:

1. for backbone
   - data augmentation: *CutMix [91]*, ~~*Mosaic*~~
   - imbalance sampling: *Class labeling smoothing [73]*
   - regularization: *DropBlock [16]*

data augmentation: *CutMix*

- **CutMix(CutMix: Regularization strategy to train strong classifiers with localizable features , ICCV 2019)**
  - 기존 data augmentation 기법들의 한계를 보완하고자 제안된 기법으로서, 다양한 vision task에 적용 시 성능 향상
    - MixUp: 2개의 이미지를 weighted linear interpolation을 이용하여 합침
    - CutOut: 1개의 이미지 내 일부 patch를 잘라내서 해당 영역을 0으로 채움
    - CutMix: 2개의 이미지를 이용하며, 첫번째 이미지의 일부 patch를 잘라낸 후 해당 영역을 두번째 이미지에서 가져온 patch로 대체
  - 상당히 작은 샘플 생성 비용으로도, 기존 기법들 대비 우수한 성능을 보이는 regularization 전략 제안

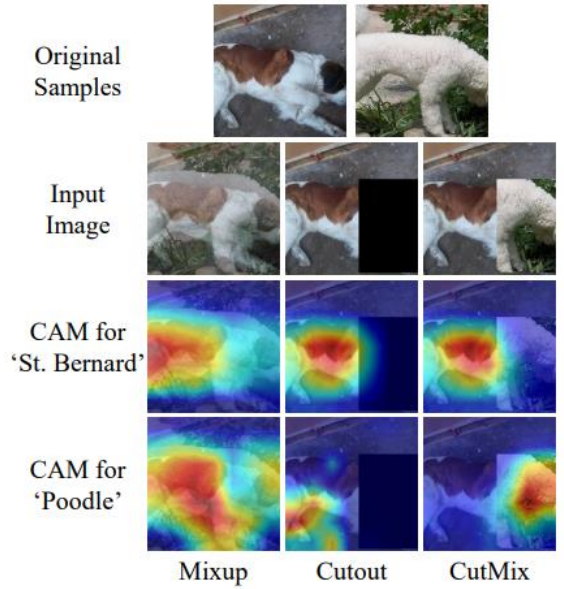


Figure 1: Class activation mapping (CAM) [52] visualizations on 'Saint Bernard' and 'Miniature Poodle' samples using various augmentation techniques. From top to bottom rows, we show the original images, input augmented image, CAM for class 'Saint Bernard', and CAM for class 'Miniature Poodle', respectively. Note that CutMix can take advantage of the mixed region on image, but Cutout cannot.

	ResNet-50	Mixup [48]	Cutout [3]	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	**78.6** **(+2.3)**
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	**47.3** **(+1.0)**
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	**76.7** **(+1.1)**

Table 1: Overview of the results of Mixup, Cutout, and our CutMix on ImageNet classification, ImageNet localization, and Pascal VOC 07 detection (transfer learning with SSD [24] finetuning) tasks. Note that CutMix significantly improves the performance on various tasks.

✓ MixUp, CutOut 보다 classification, localization, detection 성능이 모두 우수

✓ baseline인 ResNet-50보다 classification(+2.3), localization(+1.0), detection(+1.1) % 향상

imbalance sampling: *Class labeling smoothing*

- **Class labeling smoothing(Rethinking the inception architecture for computer vision , CVPR 2016)**
  - mislabeling된 label error가 존재할 경우 loss function 계산 시 잘못된 영향을 받으므로, 이를 regularization 하기 위해 label smoothing 기법을 적용
  - 아래 식을 적용하여 ground truth label의 distribution를 변경

$$q'(k) = (1-\epsilon)\delta_{k,y} + \frac{\epsilon}{K}.$$

  - q'(k): smoothing된 k번째 label / epsilon: 하이퍼 파라미터 / delta_k,y: one-hot으로 encoding된 k번째 label / K = 클래스 개수
  - epsilon = 1이면 uniform distribution이 되고, epsilon = 0이면 그냥 one-hot representation을 이용

Network	Top-1 Error	Top-5 Error	Cost Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	**1.5**
BN-Inception [7]	25.2%	7.8	2.0
Inception-v2	23.4%	-	3.8
Inception-v2 RMSProp	23.1%	6.3	3.8
Inception-v2 Label Smoothing	22.8%	6.1	3.8
Inception-v2 Factorized 7×7	21.6%	5.8	4.8
Inception-v2 BN-auxiliary	**21.2%**	**5.6%**	4.8

Table 3. Single crop experimental results comparing the cumulative effects on the various contributing factors. We compare our numbers with the best published single-crop inference for Ioffe at al [7]. For the "Inception-v2" lines, the changes are cumulative and each subsequent line includes the new change in addition to the previous ones. The last line is referring to all the changes is what we refer to as "Inception-v3" below. Unfortunately, He et al [6] reports the only 10-crop evaluation results, but not single crop results, which is reported in the Table 4 below.

In our ImageNet experiments with $K = 1000$ classes, we used $u(k) = 1/1000$ and $\epsilon = 0.1$. For ILSVRC 2012, we have found a consistent improvement of about 0.2% absolute both for top-1 error and the top-5 error (cf. Table 3).

✓ Inception-v2 RMSProp에 Label Smoothing 적용 결과

➔ 동일한 계산 비용으로도 Top-1 및 Top-5 error rate가 약 0.2% 향상됨

regularization: *DropBlock*

- **DropBlock(DropBlock: A regularization method for convolutional networks, NIPS 2018)**
  - ➢ DropOut의 random하게 feature를 drop하는 특성으로 인해 발생하는 overfitting 문제를 해결하여 성능을 향상
    - DropOut과 달리 일정 범위 내 일정 비율을 이용하여 feature를 drop함

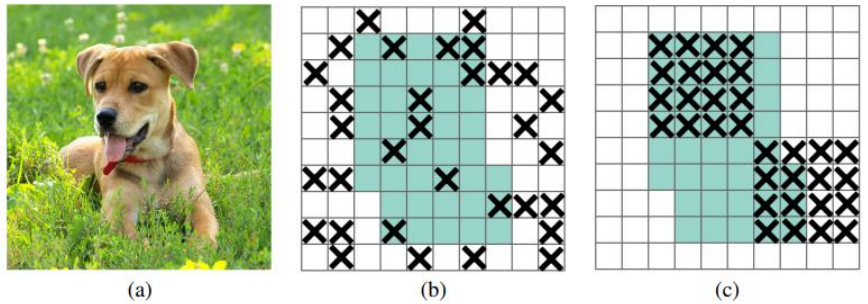


Figure 1: (a) input image to a convolutional neural network. The green regions in (b) and (c) include the activation units which contain semantic information in the input image. Dropping out activations at random is not effective in removing semantic information because nearby activations contain closely related information. Instead, dropping continuous regions can remove certain semantic information (e.g., head or feet) and consequently enforcing remaining units to learn features for classifying input image.

✓ 녹색 영역: 입력 이미지 내 의미 정보를 포함하고 있는 activation units

✓ (b)처럼 random하게 activation을 drop하는 것은 의미 정보를 제거하는데 비효율적

✓ (c)처럼 연속된 영역의 activation을 drop하면 특정한 의미 정보(예: 머리 또는 다리)의 제거가 가능

✓ 결과적으로 남아 있는 units들이 입력 이미지 분류를 위한 features들을 학습할 수 있도록 해줌

Model	top-1(%)	top-5(%)
ResNet-50	76.51 ± 0.07	93.20 ± 0.05
ResNet-50 + dropout (kp=0.7) [1]	76.80 ± 0.04	93.41 ± 0.04
ResNet-50 + DropPath (kp=0.9) [17]	77.10 ± 0.08	93.50 ± 0.05
ResNet-50 + SpatialDropout (kp=0.9) [20]	77.41 ± 0.04	93.74 ± 0.02
ResNet-50 + Cutout [23]	76.52 ± 0.07	93.21 ± 0.04
ResNet-50 + AutoAugment [27]	77.63	93.82
ResNet-50 + label smoothing (0.1) [28]	77.17 ±0.05	93.45 ±0.03
ResNet-50 + DropBlock, (kp=0.9)	78.13 ± 0.05	94.02 ± 0.02
ResNet-50 + DropBlock (kp=0.9) + label smoothing (0.1)	78.35 ± 0.05	94.15 ± 0.03

Table 1: Summary of validation accuracy on ImageNet dataset for ResNet-50 architecture. For dropout, DropPath, and SpatialDropout, we trained models with different *keep_prob* values and reported the best result. DropBlock is applied with $block\_size = 7$. We report average over 3 runs.

✓ baseline인 ResNet-50에 DropBlock을 적용 결과

➔ top 1 accuracy는 1.62% 향상(76.51→78.13)

➔ top 5 accuracy는 0.82% 향상(93.20→94.02)

➔ 또한, 기존 기법들 대비 우수한 accuracy를 보임

BoF:

1. for backbone
   - data augmentation: *CutMix [91]*, *Mosaic*
   - imbalance sampling: *Class labeling smoothing [73]*
   - regularization: *DropBlock [16]*

2. for detector
   - objective function: *CIoU-loss [99]*
   - normalization of network activation: *CmBN*
   - regularization: *DropBlock [16]*
   - data augmentation: *Mosaic*, *Self-Adversarial Training*
   - hyper-parameters optimization: *Genetic algorithms*
   - learning rate scheduler: *Cosine annealing scheduler [52]*
   - others:
     - *Eliminate grid sensitivity*
     - *Using multiple anchors for a single ground truth*
     - *Random training shapes*

BoS:

1. for backbone
   - activation function: *Mish [55]*
   - skip-connection: *CSP [81]*, *MiWRC [?]*

2. for detector
   - activation function: *Mish [55]*
   - receptive field enhancement: *SPP [25]*
   - attention: *SAM [85](Modified)*
   - feature integration: *PAN [49](Modified)*
   - post-processing: *DIoU NMS [99]*

※ Red: Existing, Purple: YOLOv4, Green: Unknown

YOLOv4에서 사용한 기법들

BoS:

1. for backbone
   - activation function: *Mish [55]*
   - skip-connection: ~~*CSP [81]*~~, *MiWRC [?]*

activation function: *Mish*

- **Mish(Mish: A self regularized nonmonotonic neural activation function, arXiv 2019)**
  - ➢ 기존의 여러 activation functions뿐만 아니라, 최근의 reinforcement learning을 적용한 Swish보다 성능이 우수한 activation function

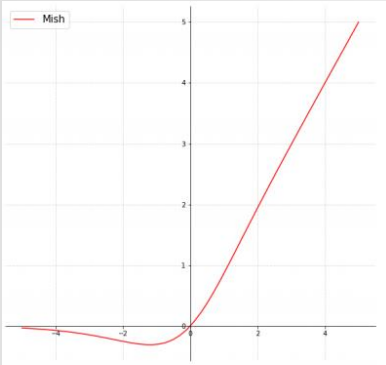


Figure 1. Mish Activation Function

$$f(x) = x \cdot tanh\big(softplus(x)\big) = x \cdot tanh(\ln(1 + e^x))$$

- ✓ ReLU나 Swish처럼 bounded below와 unbounded above한 특징을 가지며, 범위는 [-0.31, ∞)
- ✓ unbounded above: gradient가 0에 가까울 경우 훈련 속도가 급격히 느려지는 saturation 문제를 줄임
- ✓ bounded below: 강한 regularization 효과와 overfitting 문제를 줄임
- ✓ ReLU에서 발생하던 dying neuron으로 인한 gradient vanishing 문제를 줄임

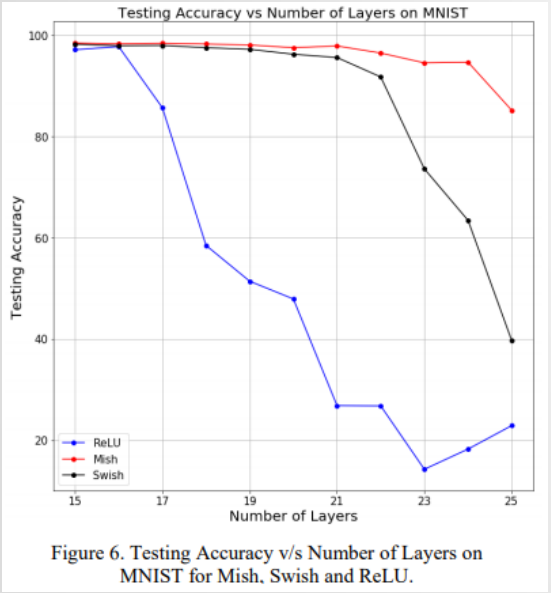


Figure 6. Testing Accuracy v/s Number of Layers on MNIST for Mish, Swish and ReLU.

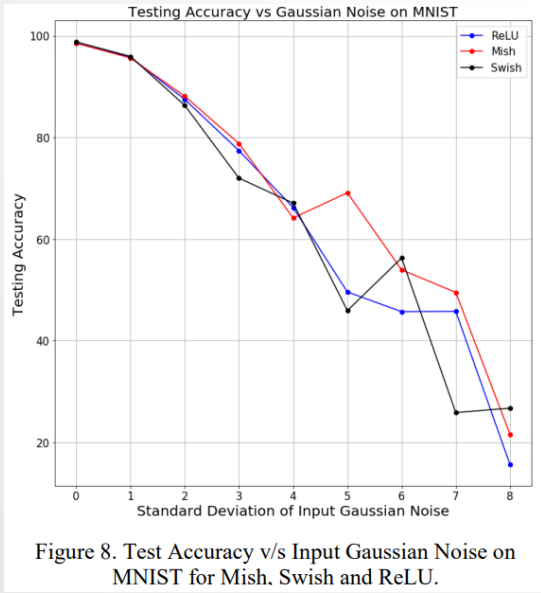


Figure 8. Test Accuracy v/s Input Gaussian Noise on MNIST for Mish, Swish and ReLU.

✓ Layer의 개수, Gaussian 잡음량을 변화시키면서 실험한 결과

➔ ReLU나 Swish보다 우수한 accuracy를 보임

➔ 이외에도, 다양한 하이퍼 파라미터들(batch size, optimizer, learing rate, regularizer 등)을 변화시키면서 실험한 결과 ReLU, Swish보다 우수한 accuracy를 보임

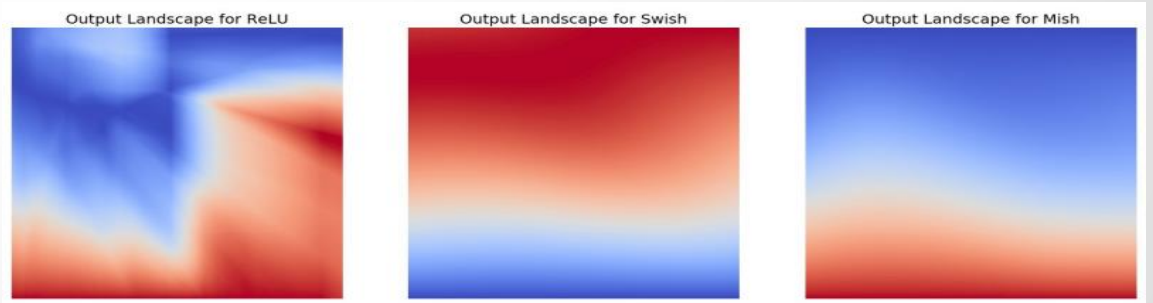


Figure 5. Output Landscape of a Random Neural Network with ReLU, Swish and Mish

- random 초기값을 이용한 ReLU, Swish, Mish의 출력 결과 비교
  - ➔ Swish 및 Mish와 비교했을 때 ReLU 좌표의 스칼라 크기 사이에는 sharp한 transition이 존재
  - ➔ smooth한 transition은 보다 부드러운 최적화 기능을 제공하여 loss을 줄이면서 일반화를 잘할 수 있음

skip-connection: *MiWRC*

- **MiWRC(Multi-input Weighted Residual Connections, ?)**
  - 관련된 참고 논문 및 내용이 없으나, EfficientDet(https://arxiv.org/abs/1911.09070)의 Weighted Feature Fusion을 이용해 Residual Connections을 수행한 것으로 유추
  - Weighted Feature Fusion
    - FPN에서는 서로 다른 해상도의 입력 feature들을 합칠 때, 해상도를 갖게 만들도록 resize 수행 후 단순히 더하는 방식을 이용
    - 입력 feature들의 해상도가 다르면, 출력 feature들에 기여하는 weight도 다르게 적용하는 것이 효율적
    - 각각의 입력에 부가적인 weight를 주고, 각 입력 feature들의 중요성을 network가 학습할 수 있는 방식을 적용
    - 3개의 feature fusion(Unbounded, Softmax-based, Fast normalized) 방식을 고려하였고, 최종적으로 Fast normalized fusion을 적용

$$O = \sum_i \frac{w_i}{\epsilon + \sum_j w_j} \cdot I_i$$

- ✓ weight들은 ReLU를 통과하기 때문에 non-zero임이 보장
- ✓ 분모가 0이 되는 것을 방지하기 위해 epsilon = 0.0001로 설정
- ✓ weight 값이 0~1 사이로 정규화되는 것은 Softmax와 유사

Model	Softmax Fusion AP	Fast Fusion AP (delta)	Speedup
Model1	33.96	33.85 (-0.11)	1.28x
Model2	43.78	43.77 (-0.01)	1.26x
Model3	48.79	48.74 (-0.05)	1.31x

Table 6: **Comparison of different feature fusion –** Our fast fusion achieves similar accuracy as softmax-based fusion, but runs 28% - 31% faster.

✓ Softmax fusion 및 Fast fusion에 모델 크기 변화를 시키면서 성능을 비교한 결과

➔ Softmax fusion보다 미세한 AP 손실(-0.05~-0.11)이 있지만, 28~31%의 향상된 속도를 보임

BoF:

1. for backbone
   - data augmentation: *CutMix [91]*, *Mosaic*
   - imbalance sampling: *Class labeling smoothing [73]*
   - regularization: *DropBlock [16]*

2. for detector
   - objective function: *CIoU-loss [99]*
   - normalization of network activation: *CmBN*
   - regularization: *DropBlock [16]*
   - data augmentation: *Mosaic*, *Self-Adversarial Training*
   - hyper-parameters optimization: *Genetic algorithms*
   - learning rate scheduler: *Cosine annealing scheduler [52]*
   - others:
     - *Eliminate grid sensitivity*
     - *Using multiple anchors for a single ground truth*
     - *Random training shapes*

BoS:

1. for backbone
   - activation function: *Mish [55]*
   - skip-connection: *CSP [81]*, *MiWRC [?]*

2. for detector
   - activation function: *Mish [55]*
   - receptive field enhancement: *SPP [25]*
   - attention: *SAM [85](Modified)*
   - feature integration: *PAN [49](Modified)*
   - post-processing: *DIoU NMS [99]*

※ Red: Existing, Purple: YOLOv4, Green: Unknown

YOLOv4에서 사용한 기법들

BoF:

2. for detector

- **objective function:** ***CIoU-loss [99]***
- **normalization of network activation:** ~~***CmBN***~~
- **regularization:** ~~***DropBlock [16]***~~
- **data augmentation:** ~~***Mosaic, Self-Adversarial Training***~~
- **hyper-parameters optimization:** ***Genetic algorithms***
- **learning rate scheduler:** ***Cosine annealing scheduler [52]***
- **others:**
  - ***Eliminate grid sensitivity***
  - ***Using multiple anchors for a single ground truth***
  - ***Random training shapes***

objective function: *CIoU-loss*

- **GIoU-loss(Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression, CVPR 2019)**
  - 기존의 SSD, YOLOv3, Faster-RCNN 등은 bbox 사이의 거리 측정을 위해 GT bbox와 predicted bbox의 좌표에 대해 L_n-norm(n=1, 2) loss를 적용
  - 이러한 방식은 point 좌표에 기반하지 않으므로, 최적의 IoU metric을 얻는데 있어서 적합한 선택이 아니며, Generalized IoU loss가 제안됨

$$\mathcal{L}_{IoU} = 1 - \frac{|B \cap B^{gt}|}{|B \cup B^{gt}|} \qquad \mathcal{L}_{GIoU} = 1 - IoU + \frac{|C - B \cup B^{gt}|}{|C|}$$

IoU　　IoU loss　　penalty term

  - B: predicted bbox, B^{gt}: ground truth bbox
  - C: B와 B^{gt} 를 묶을 수 있는 가장 작은 크기의 bbox

- **CIoU-loss(Distance-IoU Loss: Faster and better learning for bounding box regression, AAAI 2020)**
  - IoU loss나 GIoU loss 등은 L_n-norm loss 대비 성능이 개선되었지만 여전히 느린 convergence와 부정확한 regression 문제가 발생
  - GIoU loss보다 convergence가 빠르며 정확한 DIoU loss와 DIoU loss에 종횡비의 consistency를 위한 penalty term(αv)이 추가된 CIoU loss가 제안됨

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v$$

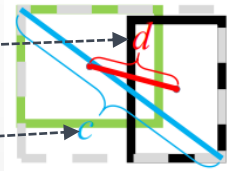
IoU loss　penalty term　penalty term

Distance IoU loss

Complete IoU loss

$$\alpha = \frac{v}{(1 - IoU) + v}$$

$$v = \frac{4}{\pi^2}(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h})^2$$



- ρ(•): euclidean distance
- b: predicted bbox(검은색)의 center point, b^{gt}: ground truth bbox(녹색)의 center point
- c: predicted bbox와 ground truth bbox를 묶을 수 있는 가장 작은 크기의 bbox(회색)에 대한 diagonal length
- α: trade-off 파라미터, v: 종횡비의 consistency를 측정

Table 1: Quantitative comparison of **YOLOv3** (Redmon and Farhadi 2018) trained using $\mathcal{L}_{IoU}$ (baseline), $\mathcal{L}_{GIoU}$, $\mathcal{L}_{DIoU}$ and $\mathcal{L}_{CIoU}$. (D) denotes using DIoU-NMS. The results are reported on the test set of PASCAL VOC 2007.

Loss / Evaluation	AP		AP75	
	IoU	**GIoU**	**IoU**	**GIoU**
$\mathcal{L}_{IoU}$	46.57	45.82	49.82	48.76
$\mathcal{L}_{GIoU}$	47.73	46.88	52.20	51.05
Relative improv. %	2.49%	2.31%	4.78%	4.70%
$\mathcal{L}_{DIoU}$	48.10	47.38	52.82	51.88
Relative improv. %	3.29%	3.40%	6.02%	6.40%
$\mathcal{L}_{CIoU}$	**49.21**	**48.42**	**54.28**	**52.87**
Relative improv. %	**5.67%**	**5.67%**	**8.95%**	**8.43%**
$\mathcal{L}_{CIoU}(D)$	**49.32**	**48.54**	**54.74**	**53.30**
Relative improv. %	**5.91%**	**5.94%**	**9.88%**	**9.31%**

✓ 다양한 objective function들을 이용하여 훈련 후 IoU와 GIoU를 이용하여 평가한 실험 결과

➔ CIoU loss는 baseline인 IoU loss 대비 AP가 5.67%(IoU, GIoU 모두) 향상

➔ 기존 GIoU loss 대비 AP는 3.18%(IoU), 3.36%(GIoU) 향상

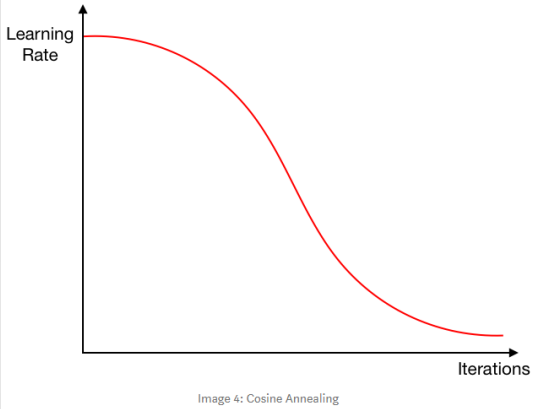
hyper-parameters optimization: *Genetic algorithms*

- **Genetic algorithms**
  - ➢ 논문에는 관련 내용이 거의 없으며, 아래와 같은 내용만 기술
    - ✓ 부가적인 성능 개선을 위해 genetic algorithms을 이용하여 최적의 hyper-parameter를 선택
    - ✓ MS COCO를 이용한 object detection 실험 시 genetic algorithms을 이용하여 hyper-parameter 탐색 실험 수행
      - GIoU loss를 이용하여 YOLOv3-SPP를 훈련 시, 전체 시간 중 처음 10% 정도만 genetic algorithms을 적용
      - min-val 5k sets에 대해 300 epochs을 탐색
      - 채택된 parameters
        - Learning rate: 0.00261
        - Momentum: 0.949
        - IoU threshold: 0.213(ground truth 할당용)
        - Loss normalizer: 0.07
    - ✓ CSPResNeXt50-PANet-SPP를 detector로 이용하고 loss를 MSE를 적용 했을 때, genetic algorithms을 적용 결과 아래와 같은 성능 향상을 보임
      - 미적용: AP(38.0%), AP_50(60.0%), AP_75(40.8%)
      - 적용: AP(38.9%), AP_50(61.7%), AP_75(41.9%)

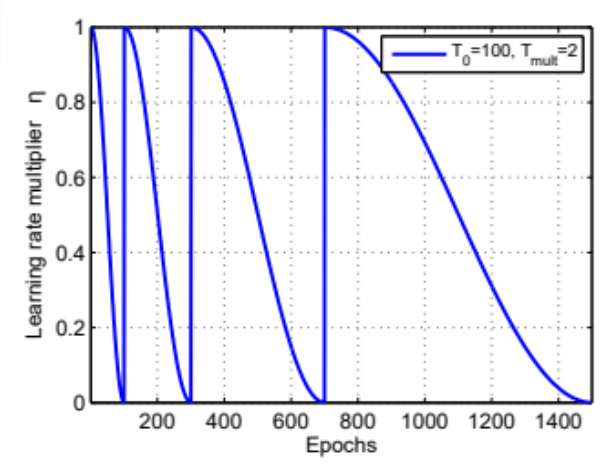
  - ➢ genetic algorithms을 이용한 hyper-parameter 최적화의 일반적인 과정
    1. n개의 hyper-parameter set들을 random으로 선택
    2. n개의 model들을 만들어서 훈련 후 가장 성능이 우수한 k개의 model들을 선택
    3. 선택된 각각의 model에 대해 원래 model을 따라 약간의 변이된(mutated) k개의 hyper-parameter set들을 생성
    4. 새롭게 생성된 k개의 hyper-parameter set들을 이용하여 model을 재훈련 후 가장 성능이 우수한 model을 선택
    - ✓ 계속적인 반복을 통해 최적의 hyper-parameter set을 찾음

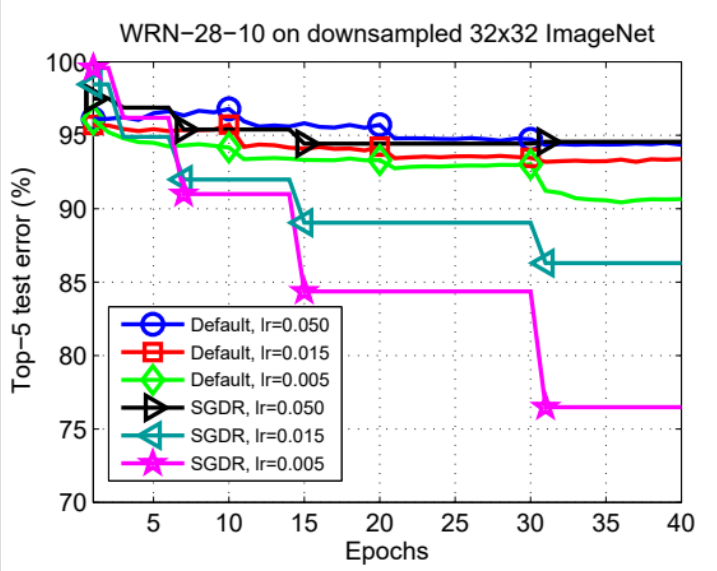
learning rate scheduler: *Cosine annealing scheduler*

- **Cosine annealing scheduler(SGDR: Stochastic gradient descent with warm restarts, ICLR 2017)**
  - ➢ Cosine annealing scheduler
    - ✓ half cosine 그래프를 따라서 learning rate를 감소시키는 learning rate annealing 기법들 중 하나



  - ➢ Warm restart
    - ✓ 훈련 도중에 learning rate를 중간 중간 증가시켜, 큰 폭의 weight update를 통해 가파른 local minimum에서 빠져 나올 수 있도록 기회를 제공





  - ➢ Epoch의 변화에 따른 default와 SGDR의 비교 결과
    - ✓ ImageNet을 32 x 32로 downsampling 하고, WRN-28-10을 이용하여 훈련
    - ✓ learning rate 초기값은 0.050, 0.015, 0.005의 3가지로 설정
    - ✓ Default: SGD + momentum
    - ✓ SGDR: T_0=1(Warm restart 초기값), T_mult=2(다음번 Warm restart를 위한 multiplicator 값)

    - ➔ learning rate 초기값이 작을수록 test error rate가 낮음
    - ➔ Warm restart를 적용한 SGDR이 Default 보다 test error rate가 상당히 낮으므로 generalization이 잘됨
      - 예(4O Epoch): Default, lr=0.005(91%) < SGDR, lr=0.005(77%) ➔ 14% 우수

others

- **Eliminate grid sensitivity**
  - YOLOv3의 bounding box prediction을 위한 equation을 변경시켜, object가 검출되지 않는 grid의 영향을 제거

- S: Eliminate grid sensitivity the equation $b_x = \sigma(t_x) + c_x, b_y = \sigma(t_y) + c_y$, where c_x and c_y are always whole numbers, is used in YOLOv3 for evaluating the object coordinates, therefore, extremely high t_x absolute values are required for the b_x value approaching the c_x or $c_x + 1$ values. We solve this problem through multiplying the sigmoid by a factor exceeding 1.0, so eliminating the effect of grid on which the object is undetectable.

- **Using multiple anchors for a single ground truth**
  - IoU(truth, anchor) > IoU_threshold 인 경우에 single ground truth에 대해 multiple anchors를 사용

- IT: IoU threshold - using multiple anchors for a single ground truth IoU (truth, anchor) $>$ IoU_threshold

- **Random training shapes**
  - 다양한 이미지 크기에 대해 강건한 성능을 위해 network 훈련 시 single-scale의 이미지가 아닌 multi-scale의 이미지를 이용하여 훈련
    - YOLOv2, YOLOv3에서도 multi-scale 훈련을 이용

- DM: Dynamic mini-batch size - automatic increase of mini-batch size during small resolution training by using Random training shapes

BoF:

1. for backbone
   - data augmentation: *CutMix [91]*, *Mosaic*
   - imbalance sampling: *Class labeling smoothing [73]*
   - regularization: *DropBlock [16]*

2. for detector
   - objective function: *CIoU-loss [99]*
   - normalization of network activation: *CmBN*
   - regularization: *DropBlock [16]*
   - data augmentation: *Mosaic*, *Self-Adversarial Training*
   - hyper-parameters optimization: *Genetic algorithms*
   - learning rate scheduler: *Cosine annealing scheduler [52]*
   - others:
     - *Eliminate grid sensitivity*
     - *Using multiple anchors for a single ground truth*
     - *Random training shapes*

BoS:

1. for backbone
   - activation function: *Mish [55]*
   - skip-connection: *CSP [81]*, *MiWRC [?]*

2. for detector
   - activation function: *Mish [55]*
   - receptive field enhancement: *SPP [25]*
   - attention: *SAM [85](Modified)*
   - feature integration: *PAN [49](Modified)*
   - post-processing: *DIoU NMS [99]*

※ Red: Existing, Purple: YOLOv4, Green: Unknown

YOLOv4에서 사용한 기법들

BoS:

2. for detector

- **activation function:** ~~*Mish [55]*~~
- **receptive field enhancement:** ~~*SPP [25]*~~
- **attention:** ~~*SAM [85](Modified)*~~
- **feature integration:** ~~*PAN [49](Modified)*~~
- **post-processing:** *DIoU NMS [99]*

post-processing: *DIoU NMS*

- **DIoU-NMS(Distance-IoU Loss: Faster and better learning for bounding box regression, AAAI 2020)**
  - ➢ 원래 NMS는 중복된 bbox들을 제거하기 위해 IoU만을 사용하며, 겹친 영역만이 metric을 위한 유일한 factor임
    - ✓ occlusion이 발생 시 종종 잘못된 suppression 결과를 야기
  - ➢ NMS를 위한 더 나은 기준인 Distance IoU를 기반으로 하는 metric을 제안
    - ✓ suppression을 위한 기준으로 겹친 영역뿐만 아니라, 2개의 bbox 내 center point 간의 거리를 함께 고려

$$s_i = \begin{cases} s_i, & IoU - \mathcal{R}_{DIoU}(\mathcal{M}, B_i) < \varepsilon, \\ 0, & IoU - \mathcal{R}_{DIoU}(\mathcal{M}, B_i) \geq \varepsilon, \end{cases} \quad (13)$$

- S_i: classification score, M: 가장 높은 score를 가진 predicted bbox
- B_i: ground truth bbox, epsilon: NMS threshold

$$\mathcal{R}_{DIoU} = \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}, \quad (6)$$

- R_DIoU: DIoU loss의 penalty term

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v$$

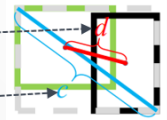
IoU loss / penalty term / penalty term

Distance IoU loss

Complete IoU loss

$$\alpha = \frac{v}{(1 - IoU) + v}$$

$$v = \frac{4}{\pi^2}(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h})^2$$



- ρ(•): euclidean distance
- b: predicted bbox(검은색)의 center point, b^{gt}: ground truth bbox(녹색)의 center point
- c: predicted bbox와 ground truth bbox를 묶을 수 있는 가장 작은 크기의 bbox(회색)에 대한 diagonal length
- α: trade-off 파라미터, v: 종횡비의 consistency를 측정

Table 1: Quantitative comparison of **YOLOv3** (Redmon and Farhadi 2018) trained using $\mathcal{L}_{IoU}$ (baseline), $\mathcal{L}_{GIoU}$, $\mathcal{L}_{DIoU}$ and $\mathcal{L}_{CIoU}$. (D) denotes using DIoU-NMS. The results are reported on the test set of PASCAL VOC 2007.

Loss / Evaluation	AP		AP75	
	IoU	**GIoU**	**IoU**	**GIoU**
$\mathcal{L}_{IoU}$	46.57	45.82	49.82	48.76
$\mathcal{L}_{GIoU}$	47.73	46.88	52.20	51.05
Relative improv. %	2.49%	2.31%	4.78%	4.70%
$\mathcal{L}_{DIoU}$	48.10	47.38	52.82	51.88
Relative improv. %	3.29%	3.40%	6.02%	6.40%
$\mathcal{L}_{CIoU}$	**49.21**	**48.42**	**54.28**	**52.87**
Relative improv. %	**5.67%**	**5.67%**	**8.95%**	**8.43%**
$\mathcal{L}_{CIoU}(D)$	**49.32**	**48.54**	**54.74**	**53.30**
Relative improv. %	**5.91%**	**5.94%**	**9.88%**	**9.31%**

- ➢ center point 사이의 거리가 먼 2개의 bbox들은 다른 위치에 놓여 있는 서로 다른 object들이므로, suppression 시 제거 대상에서 제외
- ➢ code 몇줄만 추가해서 어떠한 object detection 파이프라인에도 통합될 수 있을 만큼 매우 유연

✓ 다양한 objective function들을 이용하여 훈련 후 IoU와 GIoU를 이용하여 평가한 실험 결과

➔ CIoU loss를 구할 때 DIoU-NMS를 적용한 경우 baseline인 IoU loss 대비 AP가 5.91%(IoU), 5.94%(GIoU) 향상

➔ DIoU-NMS를 적용하지 않은 CIoU loss 대비 AP가 0.24%(IoU), 0.27%(GIoU) 향상

4. Experiments: *1. Experimental setup*

- **Image Classification 및 Object Detection에 대해 실험**

task	dataset	default hyper-parameters	BoF	BoS	hyper-parameters optimization	others
Image Classification	ImageNet (ILSVRC 2012 val)	• training steps: 8,000,000 • batch size: 128 • mini-batch size: 32 • initial learning rate: 0.1 / polynomial decay learning rate scheduling 전략 적용 • warm-up steps: 1000 • momentum: 0.9 • weight decay: 0.005	• 모두 동일한 hyper-parameters를 기본 설정으로 이용 / 50%의 training steps을 추가 적용 • MixUp, CutMix, Mosaic, Blurring data augmentation, label smoothing regularization 기법들을 검증	• 모두 동일한 hyper-parameters를 기본 설정으로 이용 • LReLU, Swish, Mish 등의 activation function들이 주는 영향에 대해 비교	-	• 모든 실험은 1080 Ti 또는 2080 Ti GPU를 이용하여 훈련
Object Detection	MS COCO (test-dev 2017)	• training steps: 500,500 • initial learning rate: 0.01 / step decay learning rate scheduling 전략 적용 / 400,000 steps과 450,000 steps에서 각각 0.1을 곱함 • momentum: 0.9 • weight decay: 0.0005 • batch size: 64 / 하나의 GPU로 multi-scale training을 고려 • mini-batch size: 8 또는 4 / 아키텍처 및 GPU 메모리 제한에 따라 결정	• grid sensitivity elimination / mosaic data augmentation / IoU threshold / genetic algorithm / class label smoothing / cross mini-batch normalization / self-adversarial training / cosine annealing scheduler / dynamic mini-batch size / DropBlock / Optimized Anchors / 서로 다른 종류의 IoU losses	• Mish / SPP / SAM / RFB / BiFPN / Gaussian YOLO [8]	• GIoU loss를 이용하여 YOLOv3-SPP를 훈련 시 genetic algorithm을 사용 / min-val 5k sets에 대해 300 epochs을 탐색 • 채택된 parameters: - learning rate: 0.00261 - momentum: 0.949 - IoU threshold: 0.213 / ground truth 할당용 - loss normalizer: 0.07	• hyper-parameter 탐색 실험에 genetic algorithm을 사용하는 것을 제외하고는 모든 실험에서 기본 설정을 이용 • 훈련에 하나의 GPU만 사용하는 것을 고려 / 여러개의 GPU를 이용하여 최적화하는 syncBN 등의 기법은 미사용

4. Experiments: *2. Influence of different features on Classifier training*

- **classifier 훈련 시 서로 다른 features들이 주는 영향에 대해 실험**
  - ✓ 후보 features
    - ➢ class label smoothing / data augmentation(bilateral blurring, MixUp, CutMix, Mosaic) / activation function(Leaky-ReLU(by default), Swish, Mish)
- **결과**

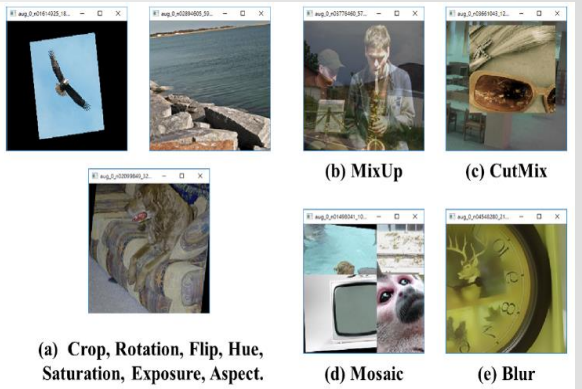


Figure 7: Various method of data augmentation.

표 2: CSPResNeXt-50를 backbone으로 이용했을 때, BoF와 Mish가 classifier의 정확도에 주는 영향

Table 2: Influence of BoF and Mish on the CSPResNeXt-50 classifier accuracy.

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.9%	94.0%
✓							77.2%	**94.0%**
	✓						**78.0%**	**94.3%**
		✓					**78.1%**	**94.5%**
			✓				77.5%	93.8%
				✓			**78.1%**	**94.4%**
					✓		64.5%	86.0%
						✓	**78.9%**	**94.5%**
	✓	✓		✓			**78.5%**	**94.8%**
	✓	✓		✓		✓	**79.8%**	**95.2%**

➔ CutMix 및 Mosaic data augmentation / Class label smoothing / Mish activation 등의 경우 정확도 향상에 유리

표 3: CSPDarknet-53(YOLOv4의 backbone)를 backbone으로 이용했을 때, BoF와 Mish가 classifier의 정확도에 주는 영향

Table 3: Influence of BoF and Mish on the CSPDarknet-53 classifier accuracy.

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.2%	93.6%
	✓	✓		✓			**77.8%**	**94.4%**
	✓	✓		✓		✓	**78.7%**	**94.8%**

➔ CSPResNeXt-50의 ablation 실험 결과를 통해 정확도 향상이 입증된 3개의 features들을 CSPDarknet-53에 적용 결과 역시 정확도 향상에 유리

➔ 추가적으로 Mish activation을 적용했을 때 보다 향상된 정확도 확보

4. Experiments: *3. Influence of different features on Detector training*

A. BoF의 다양한 features들이 detector 훈련에 주는 영향

기법	설명
S: Eliminate grid sensitivity	• YOLOv3에서 object의 좌표 계산식을 변경하여, object가 검출되지 않는 grid에 대한 영향을 제거
M: Mosaic data augmentation	• 훈련 동안에 단일 이미지 대신 4개의 image mosaic을 이용
IT: IoU threshold	• IoU(truth, anchor) > IoU_threshold 인 경우에 single ground truth에 대해 multiple anchors를 사용
GA: Genetic algorithms	• 전체 시간 중 처음 10% 기간 동안 network 학습 시 최적의 hyper-parameters 선택을 위해 Genetic Algorithms을 적용
LS: Class label smoothing	• sigmoid activation을 이용하여 Class label smoothing을 수행
CBN: CmBN	• 단일 mini-batch 내 statistic 수집이 아닌, 전체 batch 내 statistic 수집을 위해 Cross mini-Batch Normalization을 이용
CA: Cosine annealing scheduler	• 훈련 시 learning rate를 변경하는 Cosine annealing scheduler를 적용
DM: Dynamic mini-batch size	• Random training shapes을 이용하여 작은 해상도의 훈련 중에는 mini-batch 크기를 자동으로 확대
OA: Optimized Anchors	• 512 x 512 크기의 network 해상도를 이용한 훈련 시 최적화된 anchors를 사용
GIoU, CIoU, DIoU, MSE	• bounded된 box를 regression 시 서로 다른 종류의 loss 알고리즘들을 사용

- **결과(표 4): CSPResNeXt50-PANet-SPP을 detector로 이용했을 때, 서로 다른 BoF에 대한 Ablation 결과**

1) loss를 MSE로 고정시킨 후 다양한 BoF에 대한 실험 결과 (OA는 후보군에서 제외)

Table 4: Ablation Studies of Bag-of-Freebies. (CSPResNeXt50-PANet-SPP, 512x512).

S	M	IT	GA	LS	CBN	CA	DM	OA	loss	AP	AP_{50}	AP_{75}
									MSE	38.0%	60.0%	40.8%
✓									MSE	37.7%	59.9%	40.5%
	✓								MSE	**39.1%**	**61.8%**	**42.0%**
		✓							MSE	36.9%	59.7%	39.4%
			✓						MSE	**38.9%**	**61.7%**	**41.9%**
				✓					MSE	33.0%	55.4%	35.4%
					✓				MSE	**38.4%**	**60.7%**	**41.3%**
						✓			MSE	**38.7%**	**60.7%**	**41.9%**
							✓		MSE	35.3%	57.2%	38.0%
✓									GIoU	**39.4%**	59.4%	**42.5%**
✓									DIoU	**39.1%**	58.8%	**42.1%**
✓									CIoU	**39.6%**	59.2%	**42.6%**
✓	✓	✓	✓						CIoU	**41.5%**	**64.0%**	**44.8%**
	✓		✓					✓	CIoU	36.1%	56.5%	38.4%
✓	✓	✓	✓					✓	MSE	**40.3%**	**64.0%**	**43.1%**
✓	✓	✓	✓					✓	GIoU	**42.4%**	**64.4%**	**45.9%**
✓	✓	✓	✓					✓	CIoU	**42.4%**	**64.4%**	**45.9%**

➔ M(Mosaic data augmentation) / GA(Genetic algorithms) / CBN(Cross mini-Batch Normalization) / CA(Cosine annealing scheduler) 등은 포함하는 것이 정확도 향상에 유리

2) loss를 GIoU, DIoU, CIoU 등으로 변화시키면서 S, M, IT, GA, OA 등의 BoF에 대한 실험 결과(LS, CBN, CA, DM은 후보군에서 제외)

Table 4: Ablation Studies of Bag-of-Freebies. (CSPResNeXt50-PANet-SPP, 512x512).

S	M	IT	GA	LS	CBN	CA	DM	OA	loss	AP	AP_{50}	AP_{75}
									MSE	38.0%	60.0%	40.8%
✓									MSE	37.7%	59.9%	40.5%
	✓								MSE	**39.1%**	**61.8%**	**42.0%**
		✓							MSE	36.9%	59.7%	39.4%
			✓						MSE	**38.9%**	**61.7%**	**41.9%**
				✓					MSE	33.0%	55.4%	35.4%
					✓				MSE	**38.4%**	**60.7%**	**41.3%**
						✓			MSE	**38.7%**	**60.7%**	**41.9%**
							✓		MSE	35.3%	57.2%	38.0%
✓									GIoU	**39.4%**	59.4%	**42.5%**
✓									DIoU	**39.1%**	58.8%	**42.1%**
✓									CIoU	**39.6%**	59.2%	**42.6%**
✓	✓	✓	✓						CIoU	**41.5%**	**64.0%**	**44.8%**
	✓		✓					✓	CIoU	36.1%	56.5%	38.4%
✓	✓	✓	✓					✓	MSE	**40.3%**	**64.0%**	**43.1%**
✓	✓	✓	✓					✓	GIoU	**42.4%**	**64.4%**	**45.9%**
✓	✓	✓	✓					✓	CIoU	**42.4%**	**64.4%**	**45.9%**

➔ S(Eliminate grid sensitivity the equation) / M(Mosaic data augmentation) / IT(IoU threshold) / GA(Genetic algorithms) 등을 함께 적용하고 loss를 CIoU를 사용하는 것이 정확도 향상에 유리

- **결과(표 4): CSPResNeXt50-PANet-SPP을 detector로 이용했을 때, 서로 다른 BoF에 대한 Ablation 결과(con`t)**

3) loss를 CIoU를 이용하며, S, M, IT, GA 등의 BOF를 사용할 때 OA의 적용 유무에 따른 실험 결과 (LS, CBN, CA, DM은 후보군에서 제외)

Table 4: Ablation Studies of Bag-of-Freebies. (CSPResNeXt50-PANet-SPP, 512x512).

S	M	IT	GA	LS	CBN	CA	DM	OA	loss	AP	AP_{50}	AP_{75}
									MSE	38.0%	60.0%	40.8%
✓									MSE	37.7%	59.9%	40.5%
	✓								MSE	**39.1%**	**61.8%**	**42.0%**
		✓							MSE	36.9%	59.7%	39.4%
			✓						MSE	**38.9%**	**61.7%**	**41.9%**
				✓					MSE	33.0%	55.4%	35.4%
					✓				MSE	**38.4%**	**60.7%**	**41.3%**
						✓			MSE	**38.7%**	**60.7%**	**41.9%**
							✓		MSE	35.3%	57.2%	38.0%
✓									GIoU	**39.4%**	59.4%	**42.5%**
✓									DIoU	**39.1%**	58.8%	**42.1%**
✓									CIoU	**39.6%**	59.2%	**42.6%**
✓	✓	✓	✓						CIoU	**41.5%**	**64.0%**	**44.8%**
	✓		✓					✓	CIoU	36.1%	56.5%	38.4%
✓	✓	✓	✓					✓	MSE	**40.3%**	**64.0%**	**43.1%**
✓	✓	✓	✓					✓	GIoU	**42.4%**	**64.4%**	**45.9%**
✓	✓	✓	✓					✓	CIoU	**42.4%**	**64.4%**	**45.9%**

➔ OA(Optimized Anchors)를 적용하는 것이 정확도 향상에 유리

4) S, M, IT, GA, OA 등의 BOF를 사용하며, loss를 MSE, GIoU, CIoU 등으로 변화시킬 때의 실험 결과 (LS, CBN, CA, DM은 후보군에서 제외)

Table 4: Ablation Studies of Bag-of-Freebies. (CSPResNeXt50-PANet-SPP, 512x512).

S	M	IT	GA	LS	CBN	CA	DM	OA	loss	AP	AP_{50}	AP_{75}
									MSE	38.0%	60.0%	40.8%
✓									MSE	37.7%	59.9%	40.5%
	✓								MSE	**39.1%**	**61.8%**	**42.0%**
		✓							MSE	36.9%	59.7%	39.4%
			✓						MSE	**38.9%**	**61.7%**	**41.9%**
				✓					MSE	33.0%	55.4%	35.4%
					✓				MSE	**38.4%**	**60.7%**	**41.3%**
						✓			MSE	**38.7%**	**60.7%**	**41.9%**
							✓		MSE	35.3%	57.2%	38.0%
✓									GIoU	**39.4%**	59.4%	**42.5%**
✓									DIoU	**39.1%**	58.8%	**42.1%**
✓									CIoU	**39.6%**	59.2%	**42.6%**
✓	✓	✓	✓						CIoU	**41.5%**	**64.0%**	**44.8%**
	✓		✓					✓	CIoU	36.1%	56.5%	38.4%
✓	✓	✓	✓					✓	MSE	**40.3%**	**64.0%**	**43.1%**
✓	✓	✓	✓					✓	GIoU	**42.4%**	**64.4%**	**45.9%**
✓	✓	✓	✓					✓	CIoU	**42.4%**	**64.4%**	**45.9%**

➔ GIoU와 CIoU loss를 적용하는 것이 정확도 향상에 유리, 둘은 같은 결과를 보임

B. BoS의 다양한 features들이 detector 훈련에 주는 영향

- ✓ 후보 features(backbone: CSPResNXt50)
  - ➢ PAN, RFB, SAM, Gaussian YOLO (G), ASFF

- **결과**

Table 5: Ablation Studies of Bag-of-Specials. (Size 512x512).

Model	AP	AP_{50}	AP_{75}
CSPResNeXt50-PANet-SPP	42.4%	64.4%	45.9%
CSPResNeXt50-PANet-SPP-RFB	41.8%	62.7%	45.1%
CSPResNeXt50-PANet-SPP-SAM	**42.7%**	**64.6%**	**46.3%**
CSPResNeXt50-PANet-SPP-SAM-G	41.6%	62.7%	45.0%
CSPResNeXt50-PANet-SPP-ASFF-RFB	41.1%	62.6%	44.4%

➔ SPP(Spatial Pyramid Pooling) / PAN(Path Aggregation Network) / SAM(Spatial Attention Module)을 함께 사용한 경우 가장 우수한 정확도를 보임

4. Experiments: *4. Influence of different backbone and pretrained weightings on Detector training*

- **서로 다른 backbone model들이 detector에 정확도에 주는 영향에 대한 실험**
  - ✓ 후보 backbones
    - CSPResNeXt50 model
      - ➢ CSPResNeXt50-PANet-SPP / CSPResNeXt50-PANet-SPP(BoF-backbone) / CSPResNeXt50-PANet-SPP(BoF-backbone + Mish)
    - CSPDarknet53 model
      - ➢ CSPDarknet53-PANet-SPP(BoF-backbone) / CSPDarknet53-PANet-SPP(BoF-backbone + Mish)
- **결과**

표 6: detector 훈련을 위해 서로 다른 classifier로 pre-trained된 weight를 사용한 경우(다른 모든 훈련 parameters들은 모든 모델이 유사)

Table 6: Using different classifier pre-trained weightings for detector training (all other training parameters are similar in all models) .

Model (with optimal setting)	Size	AP	AP_{50}	AP_{75}
CSPResNeXt50-PANet-SPP	512x512	42.4	64.4	45.9
CSPResNeXt50-PANet-SPP (BoF-backbone)	512x512	42.3	64.3	45.7
CSPResNeXt50-PANet-SPP (BoF-backbone + Mish)	512x512	42.3	64.2	45.8
CSPDarknet53-PANet-SPP (BoF-backbone)	512x512	42.4	64.5	46.0
CSPDarknet53-PANet-SPP (BoF-backbone + Mish)	512x512	43.0	64.9	46.5

Table 2: Influence of BoF and Mish on the CSPResNeXt-50 classifier accuracy.

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.9%	94.0%
✓							77.2%	**94.0%**
	✓						**78.0%**	**94.3%**
		✓					**78.1%**	**94.5%**
			✓				77.5%	93.8%
				✓			**78.1%**	**94.4%**
					✓		64.5%	86.0%
						✓	**78.9%**	**94.5%**
	✓	✓		✓			**78.5%**	**94.8%**
	✓	✓		✓		✓	**79.8%**	**95.2%**

>

Table 3: Influence of BoF and Mish on the CSPDarknet-53 classifier accuracy.

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.2%	93.6%
	✓	✓		✓			**77.8%**	**94.4%**
	✓	✓		✓		✓	**78.7%**	**94.8%**

- Image Classification 정확도: CSPResNeXt50 model > CSPDarknet53 model
- But Object Detection 정확도: CSPResNeXt50 model < CSPDarknet53 model

➔ classification에서 가장 우수한 정확도를 보이는 model이 detector의 정확도 관점에서도 항상 우수한 것은 아님!

- CSPResNeXt50 model에 BoF와 Mish를 사용할 경우 classification 정확도가 향상
- But 이렇게 pretrained된 weight를 detector에 적용하면 정확도 저하
- CSPDarknet53 model은 BoF와 Mish를 사용할 경우 classification 정확도 뿐만 아니라 pretrained된 weight를 detector에 적용한 경우 모두에서 정확도 향상

➔ CSPDarknet53이 CSPResNeXt50보다 detector에 적합한 backbone이라고 할 수 있음!

4. Experiments: *5. Influence of different mini-batch size on Detector training*

- **서로 다른 mini-batch size를 이용하여 훈련된 model이 detector에 주는 영향에 대한 실험**
  - ✓ mini-batch size
    - 4
      - ➢ CSPResNeXt50-PANet-SPP(without BoF/BoS) / CSPDarknet53-PANet-SPP(with BoF/BoS)
    - 8
      - ➢ CSPResNeXt50-PANet-SPP(without BoF/BoS) / CSPDarknet53-PANet-SPP(with BoF/BoS)
- **결과**

표 7: detector 훈련 시 서로 다른 mini-batch size를 사용한 경우

Table 7: Using different mini-batch size for detector training.

Model (without OA)	Size	AP	AP_{50}	AP_{75}
CSPResNeXt50-PANet-SPP (without BoF/BoS, mini-batch 4)	608	37.1	59.2	39.9
CSPResNeXt50-PANet-SPP (without BoF/BoS, mini-batch 8)	608	38.4	60.6	41.6
CSPDarknet53-PANet-SPP (with BoF/BoS, mini-batch 4)	512	41.6	64.1	45.0
CSPDarknet53-PANet-SPP (with BoF/BoS, mini-batch 8)	512	41.7	64.2	45.2

➔ CSPResNeXt50-PANet-SPP와 CSPDarknet53-PANet-SPP 모두 mini-batch size의 변경 및 BoF/BoS 훈련 전략의 추가 유무는 정확도에 거의 영향을 주지 않음

➔ BoF와 BoS를 도입하면 훈련에 값비싼 GPU가 불필요, 누구나 기존의 GPU를 이용하여 우수한 성능의 detector 훈련이 가능함

5. Results: 다른 최신의 object detector들과의 비교 결과

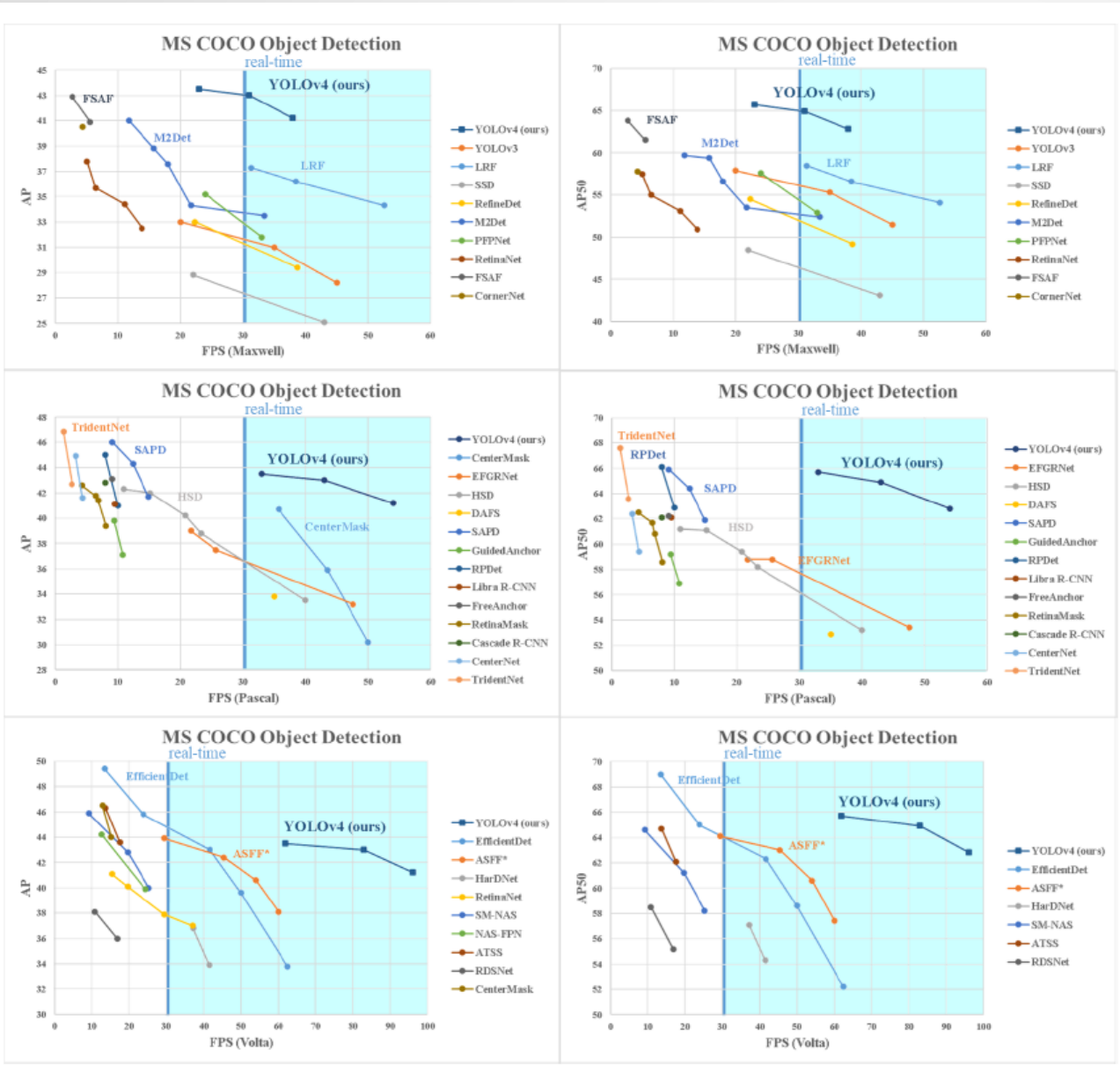


Figure 8: Comparison of the speed and accuracy of different object detectors. (Some articles stated the FPS of their detectors for only one of the GPUs: Maxwell/Pascal/Volta)

- object detector들마다 inference 시간 검증 시 서로 다른 아키텍처의 GPU를 이용
- 일반적인 M(axwell), P(ascal), V(olta) 아키텍처의 GPU를 사용하여 성능 비교
  - M: GTX Titan X or Tesla M40
  - P: Titan X or Titan Xp or GTX 1080 Ti or Tesla P100
  - V: Titan Volta or Tesla V100

➔ YOLOv4는 Pareto optimality 곡선에 위치하며, 속도 및 정확도 측면에서 가장 빠르고 정확한 detector임

5. Results: Maxwell(표 8) 및 Pascal GPU(표 9)를 사용한 frame rate 비교

Table 8: Comparison of the speed and accuracy of different object detectors on the MS COCO dataset (test-dev 2017). (Real-time detectors with FPS 30 or higher are highlighted here. We compare the results with batch=1 without using tensorRT.)

Method	Backbone	Size	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	38 (M)	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4	CSPDarknet-53	512	31 (M)	**43.0%**	**64.9%**	**46.5%**	**24.3%**	**46.1%**	**55.2%**
YOLOv4	CSPDarknet-53	608	23 (M)	43.5%	65.7%	47.3%	26.7%	46.7%	53.3%
Learning Rich Features at High-Speed for Single-Shot Object Detection [84]									
LRF	VGG-16	300	76.9 (M)	32.0%	51.5%	33.8%	12.6%	34.9%	47.0%
LRF	ResNet-101	300	52.6 (M)	34.3%	54.1%	36.6%	13.2%	38.2%	50.7%
LRF	VGG-16	512	38.5 (M)	36.2%	56.6%	38.7%	19.0%	39.9%	48.8%
LRF	ResNet-101	512	31.3 (M)	37.3%	58.5%	39.7%	19.7%	42.8%	50.1%
Receptive Field Block Net for Accurate and Fast Object Detection [47]									
RFBNet	VGG-16	300	66.7 (M)	30.3%	49.3%	31.8%	11.8%	31.9%	45.9%
RFBNet	VGG-16	512	33.3 (M)	33.8%	54.2%	35.9%	16.2%	37.1%	47.4%
RFBNet-E	VGG-16	512	30.3 (M)	34.4%	55.7%	36.4%	17.6%	37.0%	47.6%
YOLOv3: An incremental improvement [63]									
YOLOv3	Darknet-53	320	45 (M)	28.2%	51.5%	29.7%	11.9%	30.6%	43.4%
YOLOv3	Darknet-53	416	35 (M)	31.0%	55.3%	32.3%	15.2%	33.2%	42.8%
YOLOv3	Darknet-53	608	20 (M)	33.0%	57.9%	34.4%	18.3%	35.4%	41.9%
YOLOv3-SPP	Darknet-53	608	20 (M)	36.2%	60.6%	38.2%	20.6%	37.4%	46.1%
SSD: Single shot multibox detector [50]									
SSD	VGG-16	300	43 (M)	25.1%	43.1%	25.8%	6.6%	25.9%	41.4%
SSD	VGG-16	512	22 (M)	28.8%	48.5%	30.3%	10.9%	31.8%	43.5%
Single-shot refinement neural network for object detection [95]									
RefineDet	VGG-16	320	38.7 (M)	29.4%	49.2%	31.3%	10.0%	32.0%	44.4%
RefineDet	VGG-16	512	22.3 (M)	33.0%	54.5%	35.5%	16.3%	36.3%	44.3%
M2det: A single-shot object detector based on multi-level feature pyramid network [98]									
M2det	VGG-16	320	33.4 (M)	33.5%	52.4%	35.6%	14.4%	37.6%	47.6%
M2det	ResNet-101	320	21.7 (M)	34.3%	53.5%	36.5%	14.8%	38.8%	47.9%
M2det	VGG-16	512	18 (M)	37.6%	56.6%	40.5%	18.4%	43.4%	51.2%
M2det	ResNet-101	512	15.8 (M)	38.8%	59.4%	41.7%	20.5%	43.9%	53.4%
M2det	VGG-16	800	11.8 (M)	41.0%	59.7%	45.0%	22.1%	46.5%	53.8%
Parallel Feature Pyramid Network for Object Detection [34]									
PFPNet-R	VGG-16	320	33 (M)	31.8%	52.9%	33.6%	12%	35.5%	46.1%
PFPNet-R	VGG-16	512	24 (M)	35.2%	57.6%	37.9%	18.7%	38.6%	45.9%
Focal Loss for Dense Object Detection [45]									
RetinaNet	ResNet-50	500	13.9 (M)	32.5%	50.9%	34.8%	13.9%	35.8%	46.7%
RetinaNet	ResNet-101	500	11.1 (M)	34.4%	53.1%	36.8%	14.7%	38.5%	49.1%
RetinaNet	ResNet-50	800	6.5 (M)	35.7%	55.0%	38.5%	18.9%	38.9%	46.3%
RetinaNet	ResNet-101	800	5.1 (M)	37.8%	57.5%	40.8%	20.2%	41.1%	49.2%
Feature Selective Anchor-Free Module for Single-Shot Object Detection [102]									
AB+FSAF	ResNet-101	800	5.6 (M)	40.9%	61.5%	44.0%	24.0%	44.2%	51.3%
AB+FSAF	ResNeXt-101	800	2.8 (M)	42.9%	63.8%	46.3%	26.6%	46.2%	52.7%
CornerNet: Detecting objects as paired keypoints [37]									
CornerNet	Hourglass	512	4.4 (M)	40.5%	57.8%	45.3%	20.8%	44.8%	56.7%

Table 9: Comparison of the speed and accuracy of different object detectors on the MS COCO dataset (test-dev 2017). (Real-time detectors with FPS 30 or higher are highlighted here. We compare the results with batch=1 without using tensorRT.)

Method	Backbone	Size	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	54 (P)	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4	CSPDarknet-53	512	43 (P)	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4	CSPDarknet-53	608	33 (P)	**43.5%**	**65.7%**	**47.3%**	**26.7%**	**46.7%**	53.3%
CenterMask: Real-Time Anchor-Free Instance Segmentation [40]									
CenterMask-Lite	MobileNetV2-FPN	600×	50.0 (P)	30.2%	-	-	14.2%	31.9%	40.9%
CenterMask-Lite	VoVNet-19-FPN	600×	43.5 (P)	35.9%	-	-	19.6%	38.0%	45.9%
CenterMask-Lite	VoVNet-39-FPN	600×	35.7 (P)	40.7%	-	-	22.4%	43.2%	**53.5%**
Enriched Feature Guided Refinement Network for Object Detection [57]									
EFGRNet	VGG-16	320	47.6 (P)	33.2%	53.4%	35.4%	13.4%	37.1%	47.9%
EFGRNet	VG-G16	512	25.7 (P)	37.5%	58.8%	40.4%	19.7%	41.6%	49.4%
EFGRNet	ResNet-101	512	21.7 (P)	39.0%	58.8%	42.3%	17.8%	43.6%	54.5%
Hierarchical Shot Detector [3]									
HSD	VGG-16	320	40 (P)	33.5%	53.2%	36.1%	15.0%	35.0%	47.8%
HSD	VGG-16	512	23.3 (P)	38.8%	58.2%	42.5%	21.8%	41.9%	50.2%
HSD	ResNet-101	512	20.8 (P)	40.2%	59.4%	44.0%	20.0%	44.4%	54.9%
HSD	ResNeXt-101	512	15.2 (P)	41.9%	61.1%	46.2%	21.8%	46.6%	57.0%
HSD	ResNet-101	768	10.9 (P)	42.3%	61.2%	46.9%	22.8%	47.3%	55.9%
Dynamic anchor feature selection for single-shot object detection [41]									
DAFS	VGG16	512	35 (P)	33.8%	52.9%	36.9%	14.6%	37.0%	47.7%
Soft Anchor-Point Object Detection [101]									
SAPD	ResNet-50	-	14.9 (P)	41.7%	61.9%	44.6%	24.1%	44.6%	51.6%
SAPD	ResNet-50-DCN	-	12.4 (P)	44.3%	64.4%	47.7%	25.5%	47.3%	57.0%
SAPD	ResNet-101-DCN	-	9.1 (P)	46.0%	65.9%	49.6%	26.3%	49.2%	59.6%
Region proposal by guided anchoring [82]									
RetinaNet	ResNet-50	-	10.8 (P)	37.1%	56.9%	40.0%	20.1%	40.1%	48.0%
Faster R-CNN	ResNet-50	-	9.4 (P)	39.8%	59.2%	43.5%	21.8%	42.6%	50.7%
RepPoints: Point set representation for object detection [87]									
RPDet	ResNet-101	-	10 (P)	41.0%	62.9%	44.3%	23.6%	44.1%	51.7%
RPDet	ResNet-101-DCN	-	8 (P)	45.0%	66.1%	49.0%	26.6%	48.6%	57.5%
Libra R-CNN: Towards balanced learning for object detection [58]									
Libra R-CNN	ResNet-101	-	9.5 (P)	41.1%	62.1%	44.7%	23.4%	43.7%	52.5%
FreeAnchor: Learning to match anchors for visual object detection [96]									
FreeAnchor	ResNet-101	-	9.1 (P)	43.1%	62.2%	46.4%	24.5%	46.1%	54.8%
RetinaMask: Learning to Predict Masks Improves State-of-The-Art Single-Shot Detection for Free [14]									
RetinaMask	ResNet-50-FPN	800×	8.1 (P)	39.4%	58.6%	42.3%	21.9%	42.0%	51.0%
RetinaMask	ResNet-101-FPN	800×	6.9 (P)	41.4%	60.8%	44.6%	23.0%	44.5%	53.5%
RetinaMask	ResNet-101-FPN-GN	800×	6.5 (P)	41.7%	61.7%	45.0%	23.5%	44.7%	52.8%
RetinaMask	ResNeXt-101-FPN-GN	800×	4.3 (P)	42.6%	62.5%	46.0%	24.8%	45.6%	53.8%
Cascade R-CNN: Delving into high quality object detection [2]									
Cascade R-CNN	ResNet-101	-	8 (P)	42.8%	62.1%	46.3%	23.7%	45.5%	55.2%
Centernet: Object detection with keypoint triplets [13]									
Centernet	Hourglass-52	-	4.4 (P)	41.6%	59.4%	44.2%	22.5%	43.1%	54.1%
Centernet	Hourglass-104	-	3.3 (P)	44.9%	62.4%	48.1%	25.6%	47.4%	57.4%
Scale-Aware Trident Networks for Object Detection [42]									
TridentNet	ResNet-101	-	2.7 (P)	42.7%	63.6%	46.5%	23.9%	46.6%	56.6%
TridentNet	ResNet-101-DCN	-	1.3 (P)	46.8%	67.6%	51.5%	28.0%	51.2%	60.5%

5. Results: Volta GPU(표 10)를 사용한 frame rate 비교

Table 10: Comparison of the speed and accuracy of different object detectors on the MS COCO dataset (test-dev 2017). (Real-time detectors with FPS 30 or higher are highlighted here. We compare the results with batch=1 without using tensorRT.)

Method	Backbone	Size	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	96 (V)	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4	CSPDarknet-53	512	83 (V)	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4	CSPDarknet-53	608	62 (V)	**43.5%**	**65.7%**	47.3%	**26.7%**	46.7%	53.3%
EfficientDet: Scalable and Efficient Object Detection [77]									
EfficientDet-D0	Efficient-B0	512	62.5 (V)	33.8%	52.2%	35.8%	12.0%	38.3%	51.2%
EfficientDet-D1	Efficient-B1	640	50.0 (V)	39.6%	58.6%	42.3%	17.9%	44.3%	56.0%
EfficientDet-D2	Efficient-B2	768	41.7 (V)	43.0%	62.3%	46.2%	22.5%	**47.0%**	**58.4%**
EfficientDet-D3	Efficient-B3	896	23.8 (V)	45.8%	65.0%	49.3%	26.6%	49.4%	59.8%
Learning Spatial Fusion for Single-Shot Object Detection [48]									
YOLOv3 + ASFF*	Darknet-53	320	60 (V)	38.1%	57.4%	42.1%	16.1%	41.6%	53.6%
YOLOv3 + ASFF*	Darknet-53	416	54 (V)	40.6%	60.6%	45.1%	20.3%	44.2%	54.1%
YOLOv3 + ASFF*	Darknet-53	608×	45.5 (V)	42.4%	63.0%	**47.4%**	25.5%	45.7%	52.3%
YOLOv3 + ASFF*	Darknet-53	800×	29.4 (V)	43.9%	64.1%	49.2%	27.0%	46.6%	53.4%
HarDNet: A Low Memory Traffic Network [4]									
RFBNet	HarDNet68	512	41.5 (V)	33.9%	54.3%	36.2%	14.7%	36.6%	50.5%
RFBNet	HarDNet85	512	37.1 (V)	36.8%	57.1%	39.5%	16.9%	40.5%	52.9%
Focal Loss for Dense Object Detection [45]									
RetinaNet	ResNet-50	640	37 (V)	37.0%	-	-	-	-	-
RetinaNet	ResNet-101	640	29.4 (V)	37.9%	-	-	-	-	-
RetinaNet	ResNet-50	1024	19.6 (V)	40.1%	-	-	-	-	-
RetinaNet	ResNet-101	1024	15.4 (V)	41.1%	-	-	-	-	-
SM-NAS: Structural-to-Modular Neural Architecture Search for Object Detection [88]									
SM-NAS: E2	-	800×600	25.3 (V)	40.0%	58.2%	43.4%	21.1%	42.4%	51.7%
SM-NAS: E3	-	800×600	19.7 (V)	42.8%	61.2%	46.5%	23.5%	45.5%	55.6%
SM-NAS: E5	-	1333×800	9.3 (V)	45.9%	64.6%	49.6%	27.1%	49.0%	58.0%
NAS-FPN: Learning scalable feature pyramid architecture for object detection [17]									
NAS-FPN	ResNet-50	640	24.4 (V)	39.9%	-	-	-	-	-
NAS-FPN	ResNet-50	1024	12.7 (V)	44.2%	-	-	-	-	-
Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection [94]									
ATSS	ResNet-101	800×	17.5 (V)	43.6%	62.1%	47.4%	26.1%	47.0%	53.6%
ATSS	ResNet-101-DCN	800×	13.7 (V)	46.3%	64.7%	50.4%	27.7%	49.8%	58.4%
RDSNet: A New Deep Architecture for Reciprocal Object Detection and Instance Segmentation [83]									
RDSNet	ResNet-101	600	16.8 (V)	36.0%	55.2%	38.7%	17.4%	39.6%	49.7%
RDSNet	ResNet-101	800	10.9 (V)	38.1%	58.5%	40.8%	21.2%	41.5%	48.2%
CenterMask: Real-Time Anchor-Free Instance Segmentation [40]									
CenterMask	ResNet-101-FPN	800×	15.2 (V)	44.0%	-	-	25.8%	46.8%	54.9%
CenterMask	VoVNet-99-FPN	800×	12.9 (V)	46.5%	-	-	28.7%	48.9%	57.2%

6. Conclusions

- 사용 가능한 모든 대안의 detectors들보다 빠르고(FPS) 정확한(MS COCO AP50 ... 95 및 AP50) 최신의 detector를 제안
- 제안한 detector는 8~16 GB의 VRAM으로 기존의 GPU에서 훈련 및 사용이 가능하므로, 광범위한 적용이 가능
- one-stage의 anchor-based detectors의 원래 개념은 object detection task에서 충분한 생존력(viability)을 입증
- classifier와 detector 모두의 정확도 개선을 위해 많은 features들을 검증한 후 선택
- 이러한 features들은 향 후 연구 개발을 위한 모범 사례(best-practice)로 사용 가능

감사합니다.