



BIG DATA & AI ANALYTICS  
EXPERT COMPANY

One-Stage Detection  
YOLO v4



BIGDATA & AI ANALYTICS  
EXPERT COMPANY

YOLO 개요

## YOLO V1

빠른Detection 시간  
그러나낮은정확도

## YOLO V2

수행시간과성능모두개선

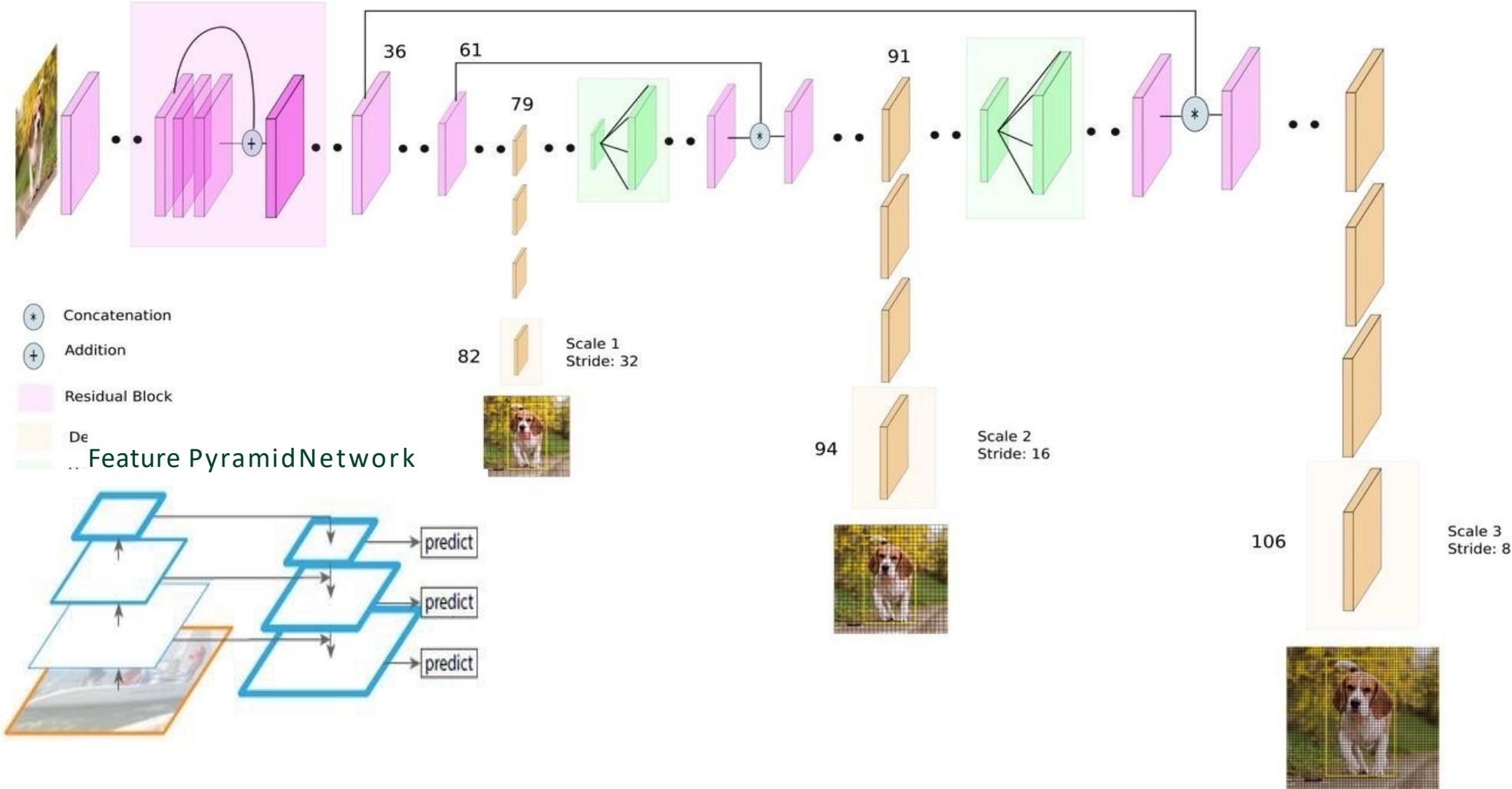
## YOLO V3

수행시간은조금느려졌으나  
성능대폭개선

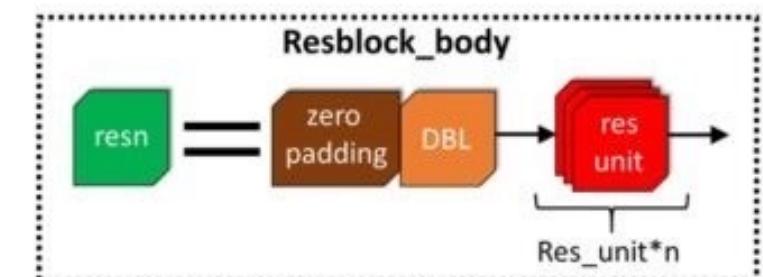
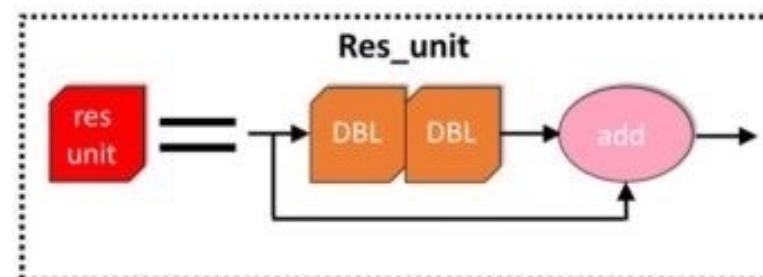
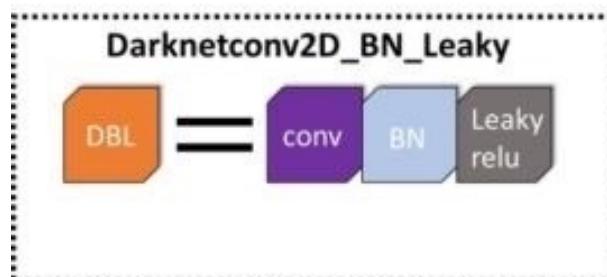
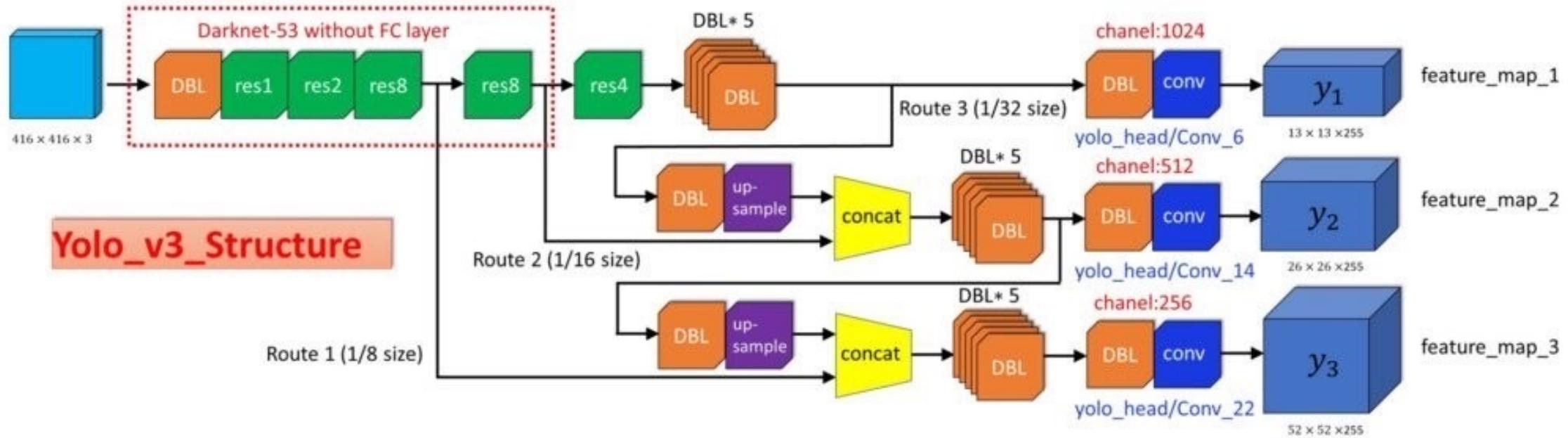
| Method               | mAP  | FPS | batch size | # Boxes | Input resolution |
|----------------------|------|-----|------------|---------|------------------|
| Faster R-CNN (VGG16) | 73.2 | 7   | 1          | ~ 6000  | ~ 1000 × 600     |
| Fast YOLO            | 52.7 | 155 | 1          | 98      | 448 × 448        |
| YOLO (VGG16)         | 66.4 | 21  | 1          | 98      | 448 × 448        |
| SSD300               | 74.3 | 46  | 1          | 8732    | 300 × 300        |
| SSD512               | 76.8 | 19  | 1          | 24564   | 512 × 512        |
| SSD300               | 74.3 | 59  | 8          | 8732    | 300 × 300        |
| SSD512               | 76.8 | 22  | 8          | 24564   | 512 × 512        |

| 항 목                                 | V1           | V2                 | V3  |
|-------------------------------------|--------------|--------------------|---|
| 원본이미지 크기                            | 446 X 446    | 416 X 416          | 416 X 416                                     |
| Feature Extractor                   | Inception 변형 | Darknet 19         | Darknet 53                                    |
| Grid당 Anchor Box 수                  | 2개           | 5개                 | Output Feature Map당 3개 서로 다른 크기와 스케일로 총 9개    |
| Anchor box 결정 방법                    |              | K-Means Clustering | K-Means Clustering                            |
| Output Feature Map 크기<br>(Depth 제외) |              | 13 x 13            | 13 x 13, 26 X 26, 52X52<br>3개의 Feature Map 사용 |
| Feature Map Scaling<br>기법           |              |                    | FPN(Feature Pyramid Network)                  |

# YOLOv3 Network 구조



# YoloV3 Network 구조

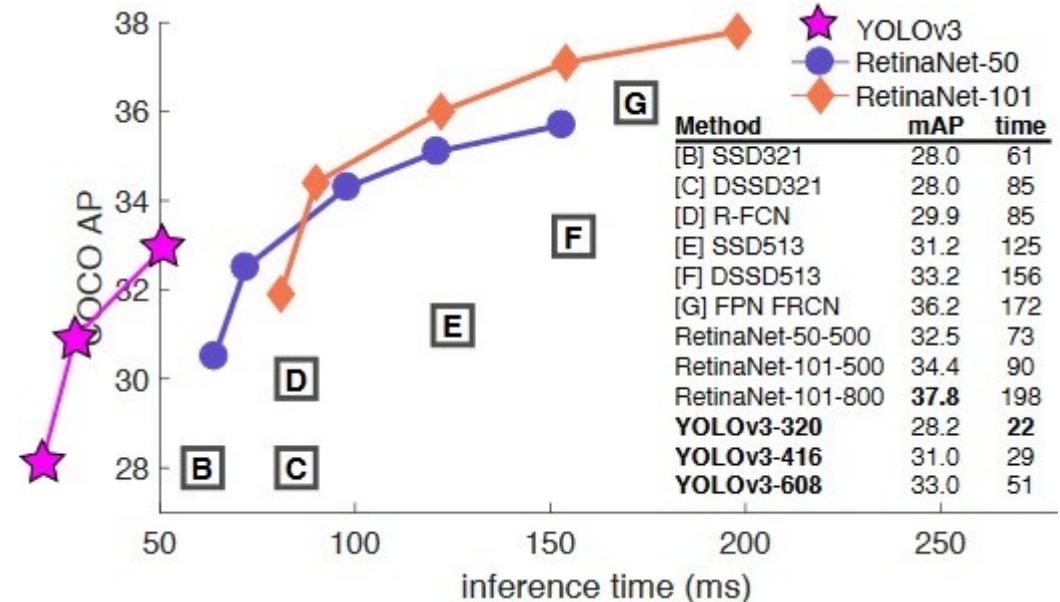


| Backbone        | Top-1       | Top-5       | Bn Ops | BFLOP/s     | FPS |
|-----------------|-------------|-------------|--------|-------------|-----|
| Darknet-19 [15] | 74.1        | 91.8        | 7.29   | 1246        | 171 |
| ResNet-101[5]   | 77.1        | 93.7        | 19.7   | 1039        |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

| Type          | Filters       | Size             | Output           |
|---------------|---------------|------------------|------------------|
| Convolutional | 32            | $3 \times 3$     | $256 \times 256$ |
| Convolutional | 64            | $3 \times 3 / 2$ | $128 \times 128$ |
| 1x            | Convolutional | 32               | $1 \times 1$     |
| Convolutional | 64            | $3 \times 3$     |                  |
|               | Residual      |                  | $128 \times 128$ |
| Convolutional | 128           | $3 \times 3 / 2$ | $64 \times 64$   |
| 2x            | Convolutional | 64               | $1 \times 1$     |
| Convolutional | 128           | $3 \times 3$     |                  |
|               | Residual      |                  | $64 \times 64$   |
| Convolutional | 256           | $3 \times 3 / 2$ | $32 \times 32$   |
| 8x            | Convolutional | 128              | $1 \times 1$     |
| Convolutional | 256           | $3 \times 3$     |                  |
|               | Residual      |                  | $32 \times 32$   |
| Convolutional | 512           | $3 \times 3 / 2$ | $16 \times 16$   |
| 8x            | Convolutional | 256              | $1 \times 1$     |
| Convolutional | 512           | $3 \times 3$     |                  |
|               | Residual      |                  | $16 \times 16$   |
| Convolutional | 1024          | $3 \times 3 / 2$ | $8 \times 8$     |
| 4x            | Convolutional | 512              | $1 \times 1$     |
| Convolutional | 1024          | $3 \times 3$     |                  |
|               | Residual      |                  | $8 \times 8$     |
|               | Avgpool       |                  | Global           |
|               | Connected     |                  | 1000             |
|               | Softmax       |                  |                  |

## Abstract

We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At  $320 \times 320$  YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP<sub>50</sub> in 51 ms on a Titan X, compared to 57.5 AP<sub>50</sub> in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at <https://pjreddie.com/yolo/>.

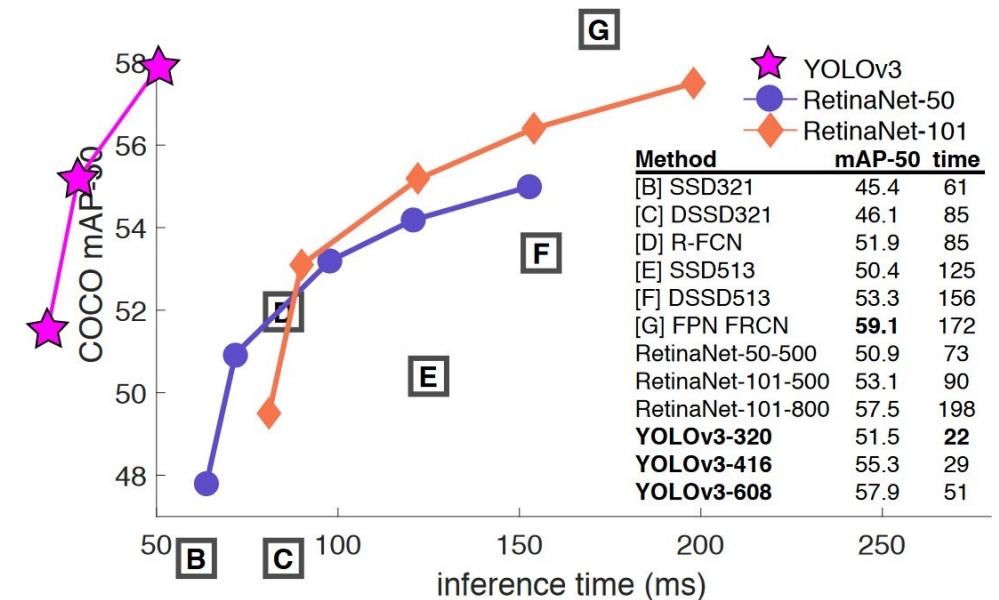
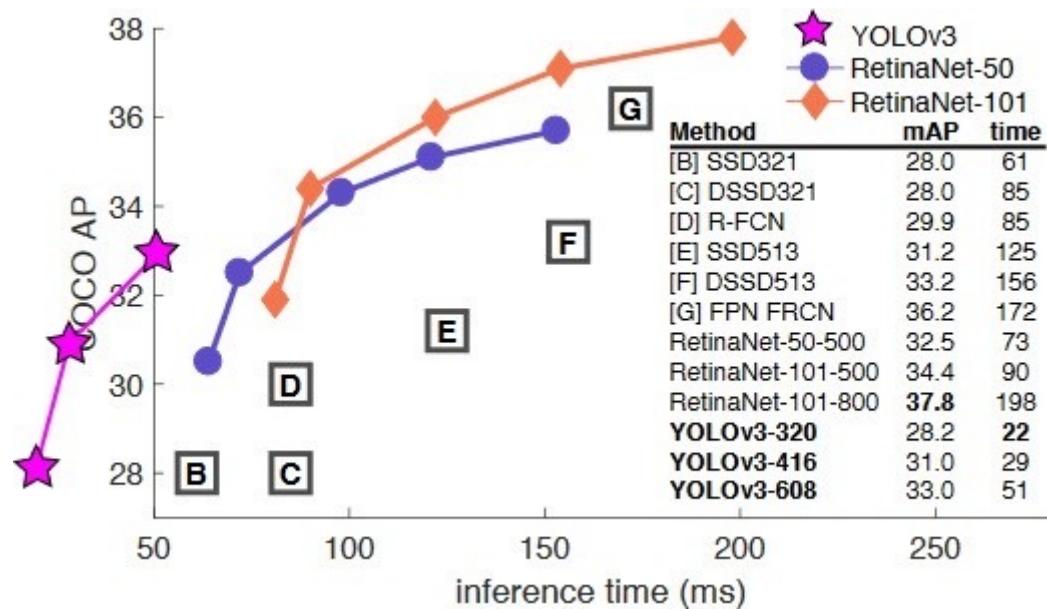


Speed



Accuracy

# YOLOv3 성능 비교



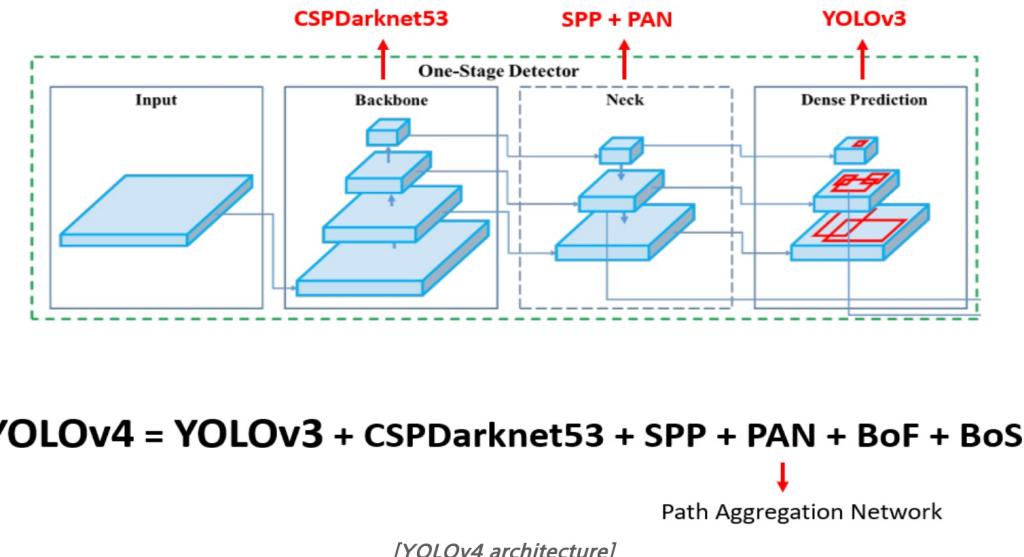
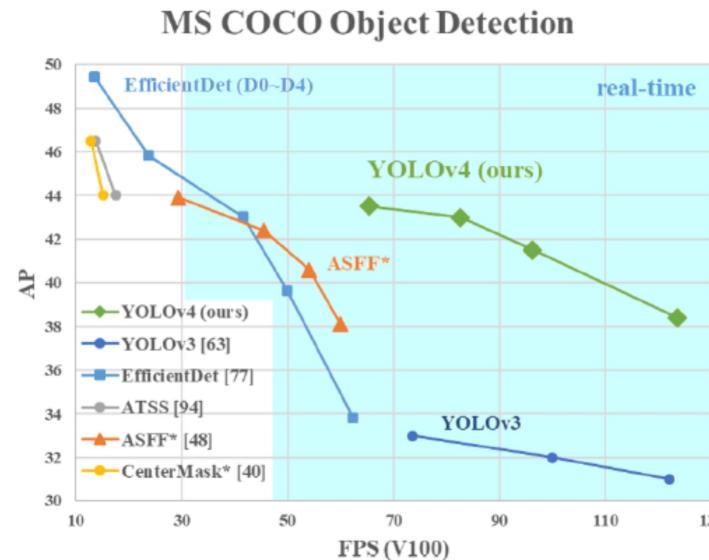


BIG DATA & AI ANALYTICS  
EXPERT COMPANY

YOLO v4

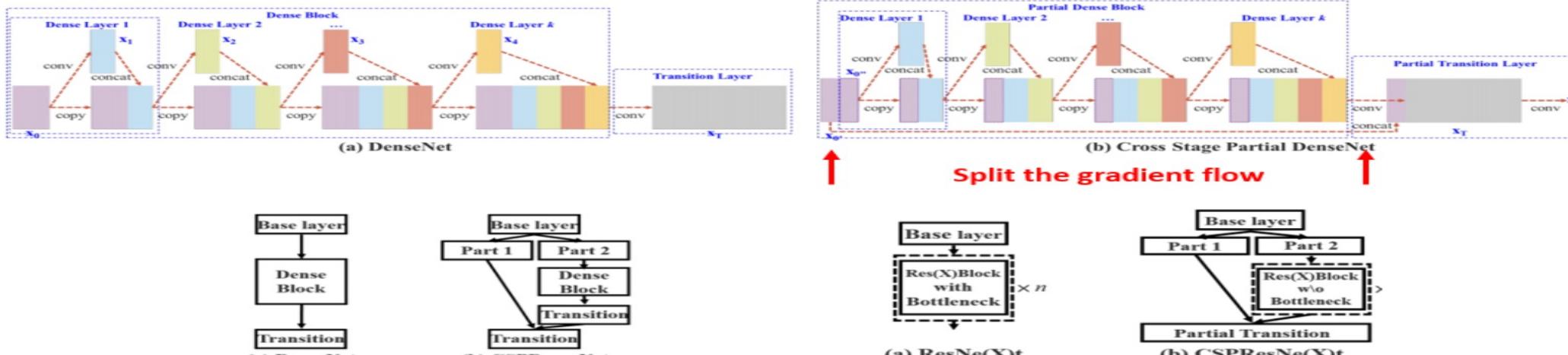
최신 Neural Networks들은 높은 정확도를 가지지만, 낮은 FPS로 인해 실시간으로 하기 어려우며 너무나 큰 mini-batch-size로 인해 학습하는데 많은 수의 GPU들이 필요하다는 단점이 있음.  
Yolo v4는 이러한 문제를 해결하기 위해 다음과 같은 3가지 목표를 가지고 진행함.

- 1) 1개의 GPU(ex : GTX 1080 Ti, 2080 Ti)만 있으면 충분히 높은 정확도와 빠른 빠른 object detector를 학습시킬 수 있도록 설계
- 2) BoF(Bag of Freeies)와 BoS(Bag of Specials) 기법을 통해 성능 향상
- 3) CBN, PAN, SAM을 포함한 기법을 활용하여 single GPU training에 효과적으로 작용



CSPNet 기반의 Backbone을 설계하고 CSPDarknet53을 제안하여 CSPResNetxt50, EfficientNet-B3에 비해 parameter수와 FLOPS는 많았지만 실제 Inference Time에서 가장 좋은 결과를 보이며 정확도를 높이며 본 속도에서의 손해를 줄임

CSPNet은 heavy한 inference cost를 완화시키며 정확도 손실을 최소로 할 수 있는 Cross Stage Partial Network 구조를 제안하였으며, 위의 그림과 같이 input feature map을 2개의 part로 나눈 뒤, 하나의 part는 연산에 참여시키지 않고 뒤에서 합쳐주는 방식을 기반으로 inference cost, memory cost 등을 줄이고 학습 관점에서 Gradient Flow를 나눠줘서 학습에 좋은 영향을 주고 정확도 손실을 줄임



# BoF(Bag of Freebies)

BoF란

보통의 object detector는 offline으로 training되므로, 연구자들은 이러한 이점을 활용하여 inference 비용을 늘리지 않고도 object detector의 정확도를 높일 수 있는 보다 나은 training 방법을 개발하는 것을 선호함  
training 전략만 바꾸거나 training에 소요되는 비용만 증가시키는 방법을 말함

Bag of Freebies (pre-processing + training strategy)

Training Phase

- Call methods that only change the **training** strategy or only increase the **training cost** as “BoF”

Data Augmentation

Regularization

Loss Function

- |                      |                   |                           |
|----------------------|-------------------|---------------------------|
| • Random erase       | • DropOut         | • MSE                     |
| • CutOut             | • DropPath        | • IoU                     |
| • MixUp              | • Spatial DropOut | • GIoU <b>Generalized</b> |
| • CutMix             | • DropBlock       | • CIoU <b>Complete</b>    |
| • Style transfer GAN |                   | • DIoU <b>Distance</b>    |

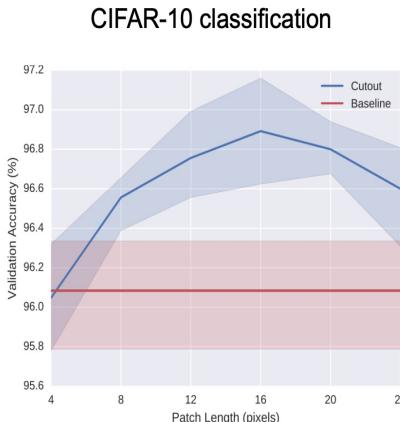
*[Bag of Freebies]*

Image Augmentation 이란 원본 이미지에 인위적인 변화를 주고 충분히 학습에 활용될 수 있는 데이터가 될 수 있도록 처리해주는 것으로, 알고리즘적인 해결방법이 아니라 전처리 기술 중 하나

## Cutout

DeVries and Taylor, 2017

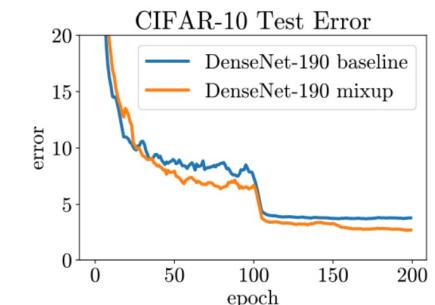
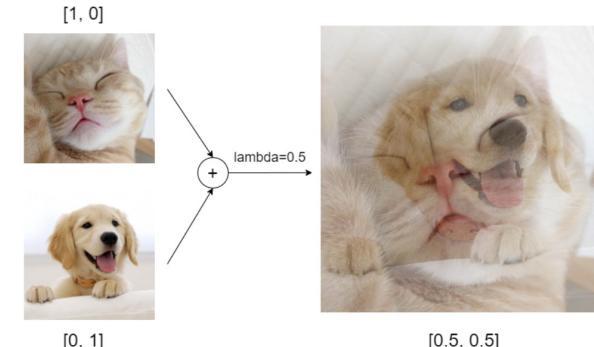
- Randomly replace square patches with noise.



## Mixup

Zhang et al., 2018

- Convex combination of pairs of images and their class labels.



(b) Test error evolution for the best ERM and *mixup* models.

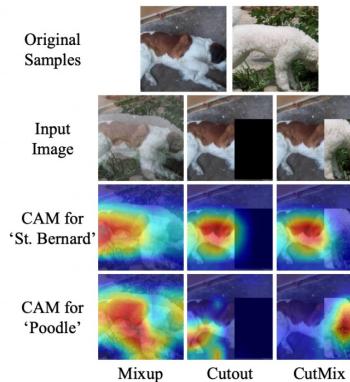
Image Augmentation 이란 원본 이미지에 인위적인 변화를 주고 충분히 학습에 활용될 수 있는 데이터가 될 수 있도록 처리해주는 것으로, 알고리즘적인 해결방법이 아니라 전처리 기술 중 하나

## CutMix

Yun et al., 2019

- Patches are cut and pasted among training images.
- Class labels are also mixed proportionally to the area of the patches.

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



## Style Transfer GAN

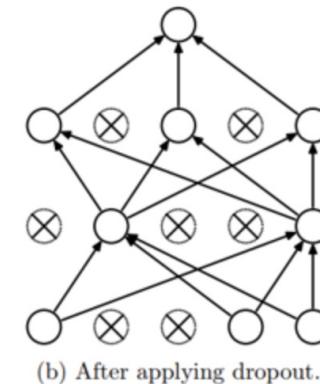
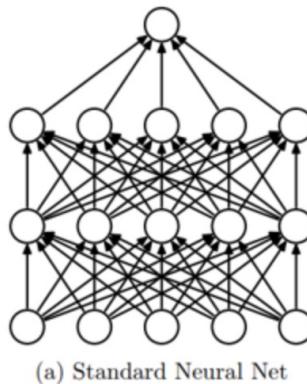


Style Transfer examples from the original paper

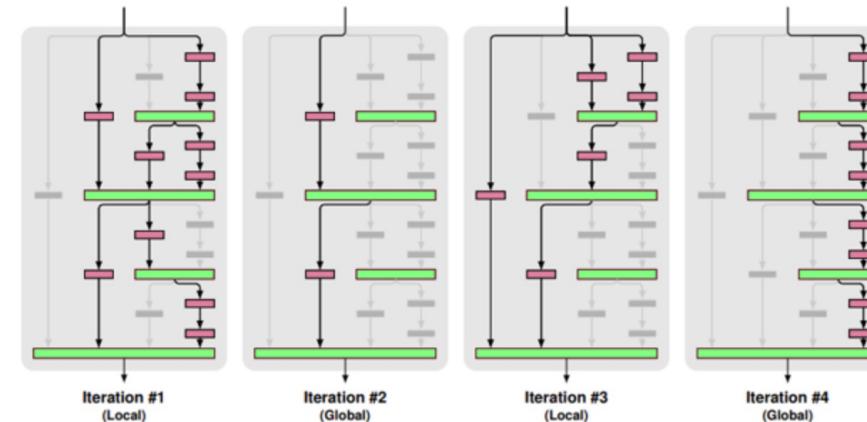
Regularization 이란, 통상적으로 cost function 혹은 Error function이 작아지는 쪽으로 학습을 진행하는 경우 특정 가중치 값들이 커지면서 오버 피팅이 발생하기 때문에 가중치에 패널티를 주는 방법

- 대표적인 generalization 기법 = **Dropout**

- 2014년 JMLR [논문](#).
- 신경망의 overfitting을 방지하기 위한 기법.
- 학습 과정에서 특정 node들을  $p$ 의 확률로 사용하겠다 ( $1-p$ 의 확률로 제거하겠다).  $0 < p < 1$ .



- Drop-path



- FractalNet의 한 path (a층부터 b층까지의 connection 경로) 내의 weight을 모두 제거하는 방식의 dropout.
- Fractal 구조인 경우에 한정되어 적용 가능.

Regularization 이란, 통상적으로 cost function 혹은 Error function이 작아지는 쪽으로 학습을 진행하는 경우 특정 가중치 값들이 커지면서 오버 피팅이 발생하기 때문에 가중치에 패널티를 주는 방법

## Drop Filter(=Spatial Drop out)

- 2018년 arXiv [논문](#). (거의 동일한 개념인 SpatialDropout은 2015년 CVPR에서 먼저 발표함. [논문](#).)
- Node간의 co-adaptation 문제는 같은 채널 내의 근처에 있는 값들에 의해서 발생하지만, 동일한 위치에 있는 다른 채널 간의 값들에 의해서 더 자주 발생함.
- 즉, channel간의 correlation이 존재함.
- 따라서 channel 하나를 통째로 drop하자. **DropFilter**. (SpatialDropout과 동일함.)

$$\mathbf{y}_i^{DropFilter} = r_i \mathbf{y}_i / p \quad (3)$$

$$r_i \sim \text{Bernoulli}(p)$$

$$p \in [0, 1]$$

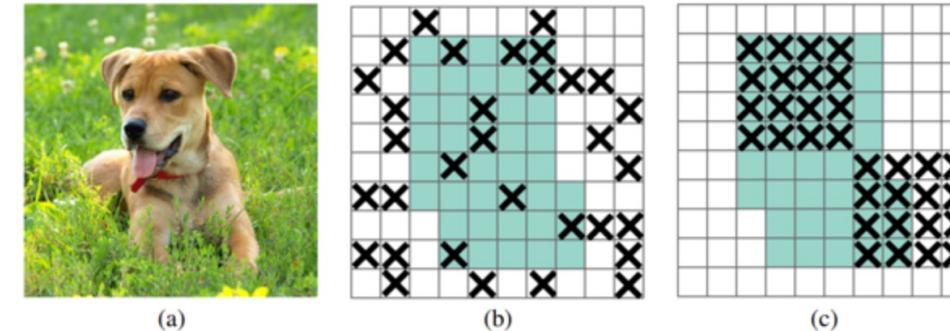
$$Y_{DropFilter} = \{\mathbf{y}_i^{DropFilter} \mid 1 \leq i \leq c\}$$

where  $r_i$  is a weight to determinate whether to drop the feature map  $\mathbf{y}_i$ .

- 간단하게, 일정 확률로 channel을 지우자.

## Drop Block

- 2018년 NIPS [논문](#).



- (b): Convolution layer의 output units은 공간적으로 연관되었으므로 (spatially correlated), 랜덤하게 몇 개의 activations를 선택해서 지우는 것으로는 연관된 정보 (correlated information, 초록색)를 제대로 지울 수 없음.
- (c): 랜덤하게 몇 개가 아니라, 연속된 몇 개의 node들을 지우자. **DropBlock**.

## 1) IoU loss [90]

예측된 BBox 영역과 ground truth BBox 영역의 coverage를 고려

IoU loss 계산 process는 ground truth와의 IoU를 수행한 후 생성된 결과를 전체 code로 연결함으로써, BBox의 4개 좌표 points에 대한 계산을 trigger함

IoU는 scale invariant한 표현이므로, 전통적인 방법으로  $\{x, y, w, h\}$ 의  $\ell_1$  또는  $\ell_2$  loss를 계산할 때 loss가 scale에 따라 증가되는 문제를 해결 가능

## 2) GIoU loss [65]

coverage 영역 이외에도 object의 shape과 orientation를 포함시킴

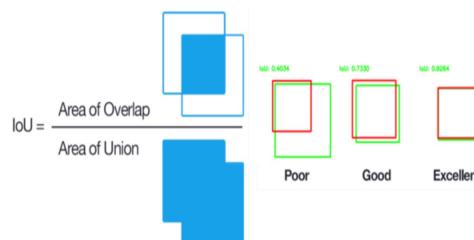
예측된 BBox와 ground truth BBox를 동시에 cover할 수 있는 가장 작은 영역의 BBox를 찾은 후, 이러한 BBox의 분모(denominator)를 원래 IoU loss에서 사용했던 분모로 대체하도록 제안

## 3) DIoU loss [99], CIoU loss [99]

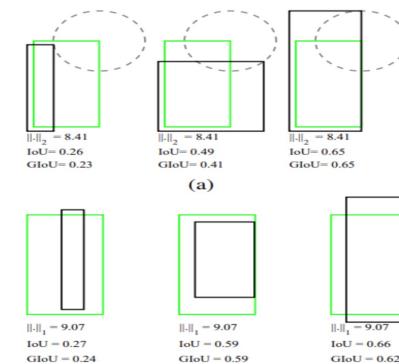
DIoU loss: 부가적으로, object 중심의 거리를 고려

CIoU loss: 겹침 영역, 중심 points 사이의 거리, aspect ratio 등을 동시에 고려. BBox regression 문제에 있어서 더 나은 수렴 속도와 정확도를 달성

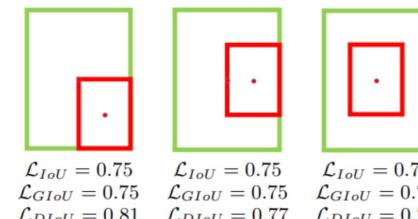
### IOU(Intersection Over Union)



### GIoU(Generalized IOU)



### Distance IOU(DIoU)



### Complete IOU(CIoU)

$$\mathcal{R}_{CIoU} = \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v, \quad v = \frac{4}{\pi^2} (\arctan \frac{w^{gt}}{h^{gt}} - \arctan \frac{w}{h})^2. \quad \alpha = \frac{v}{(1 - IoU) + v},$$

Figure 2: GIoU loss degrades to IoU loss for these cases, while our DIoU loss is still distinguishable. Green and red denote target box and predicted box respectively.

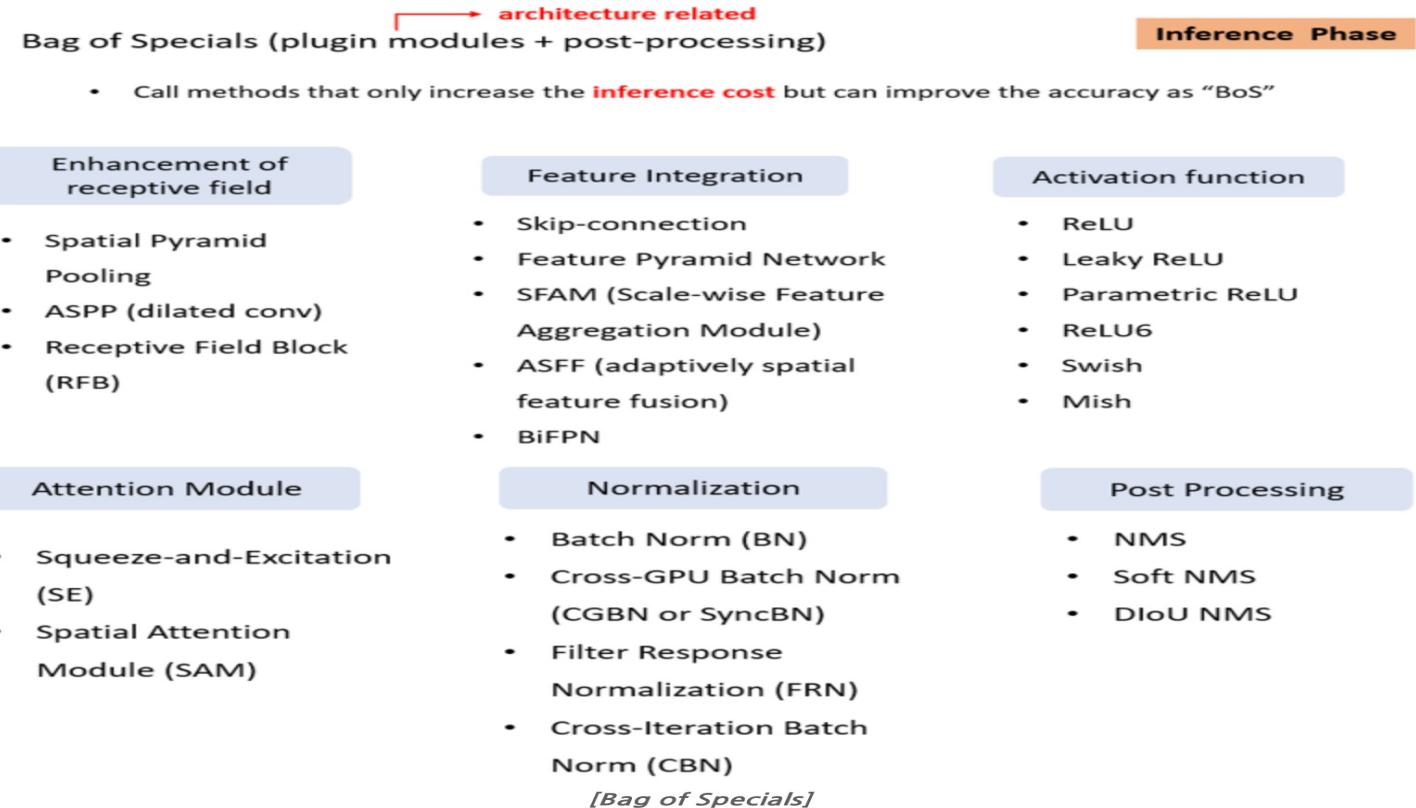
# BoS(Bag of Specials)

BoS란

inference 비용을 약간만 증가시키면서도, object detection의 정확도를 크게 향상시키는 것을 말함  
plugin modules과 post-processing으로 구성

plugin modules: 하나의 model 내 특정한 속성을 향상시키는 것(예: receptive field의 확장, attention mechanism 도입, feature integration capability 강화 등)

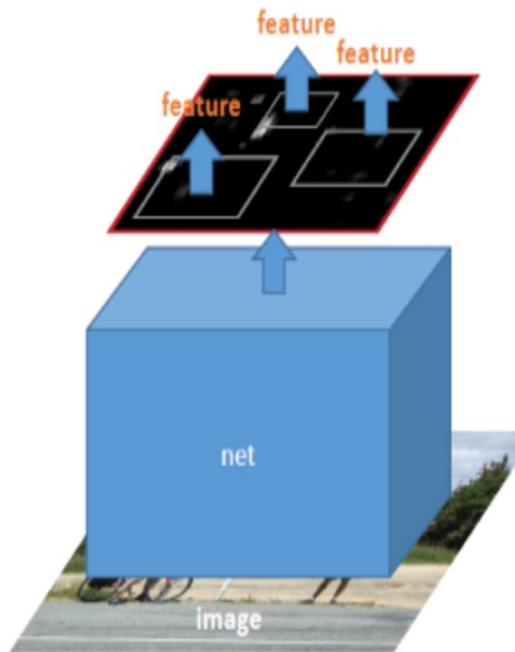
post-processing: 모델의 예측 결과를 선별(screening)하는 방법



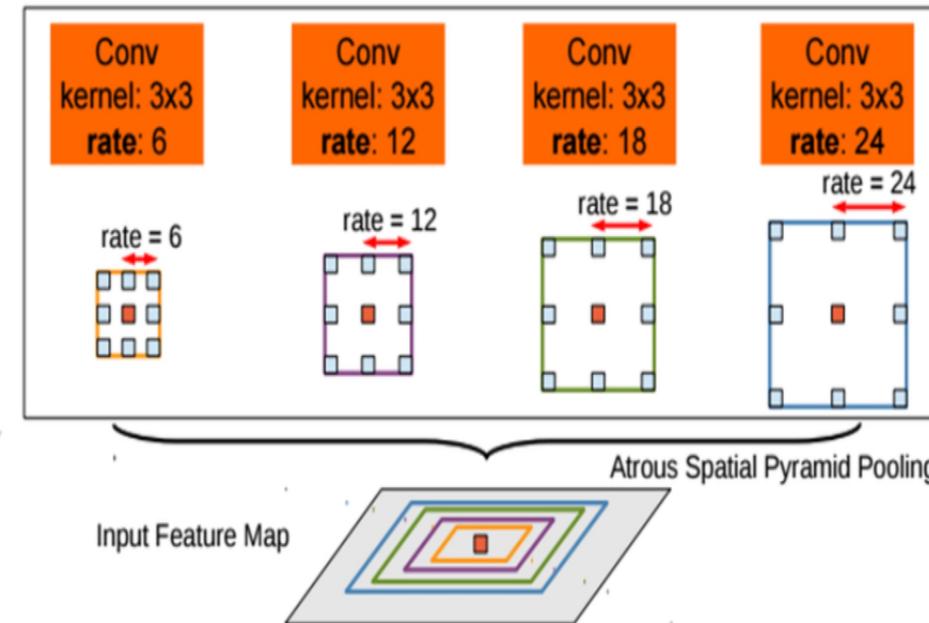
# Receptive Field

Receptive Field란 출력 레이어의 뉴런 하나에 영향을 미치는 입력 뉴런들의 공간의 크기

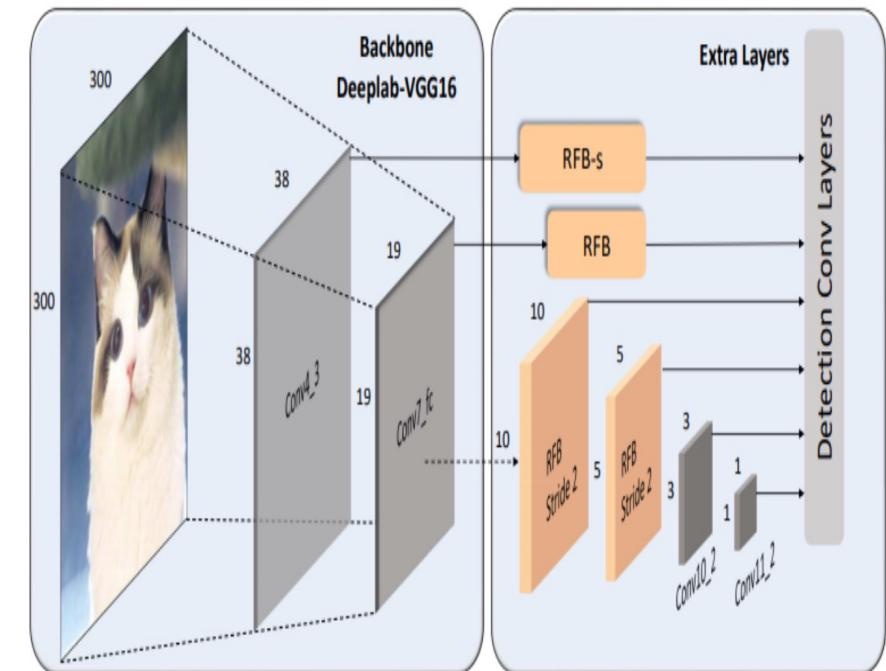
Yolo v4에서는 Spatial Pyramid Pooling(SPP), Atrous convolution을 적용한 ASPP, Receptive Field Block(RFB) 등을 사용  
Feature를 중간에서 합쳐주는 Skip-connection, Feature Pyramid Network(FPN), Scale wise Feature Aggregation  
Module(SFAM), adaptively spatial feature fusion(ASFF), BIFPN 등을 사용



SPP-net



ASPP(Atrous Spatial Pyramid Pooling)



Activation Function으로는 자주 이용되는 ReLU 계열의 activation function, AutoML로 찾은 Swish, Swish를 개선시킨 Mish 등을 사용하여 실험.

최종 모형에는 Mish를 사용

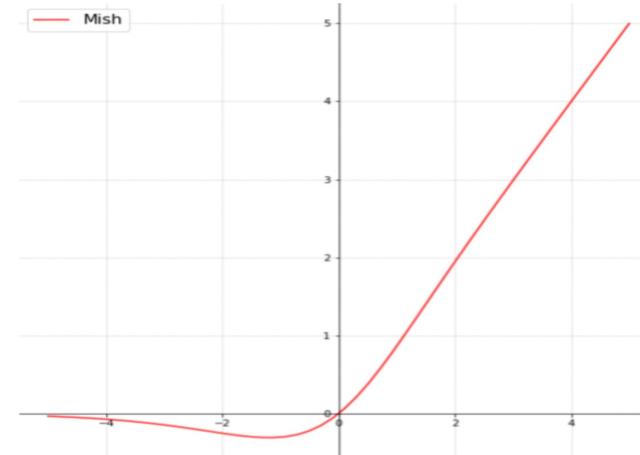
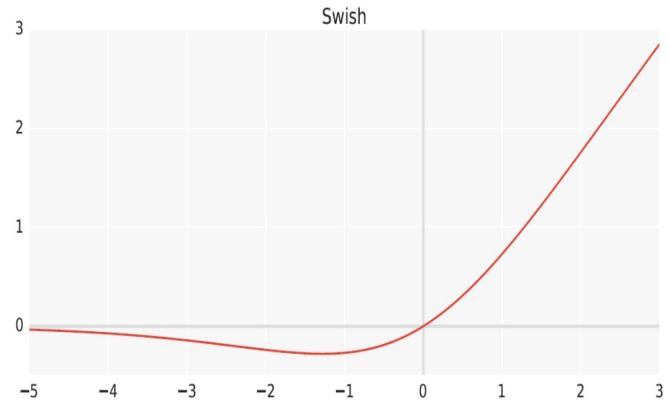
Swish는 매우 깊은 신경망에서 ReLU 보다 높은 정확도를 달성

모든 배치 크기에 대해 Swish는 ReLU를 능가하며, 모든  $x < 0$ 에 대해 함수를 감소시키거나 증가시키지 않음.

Mish와 마찬가지로 bounded below, unbounded above 특징을 가짐

Mish는 그래프가 무한대로 뻗어나가기 때문(unbounded above)에 캡핑으로 인한 포화를 피할 수 있으며 bounded below 이기 때문에 strong regularization이 나타날 수 있고 overfitting을 감소 시킬 수 있음.

약간의 음수를 허용하기 때문에 Relu Zero Bound 보다는 Gradient를 더 잘 흐르게 한다는 특징이 있음.



tensorflow implementation

```
def swish(x):  
    return x * tf.nn.sigmoid(x)  
  
def mish(x):  
    return x * tf.nn.tanh(tf.nn.softplus(x))
```

pytorch implementation

```
def swish(x):  
    return x * torch.sigmoid(x)  
  
def mish(x):  
    return x * torch.tanh(F.softplus(x))
```

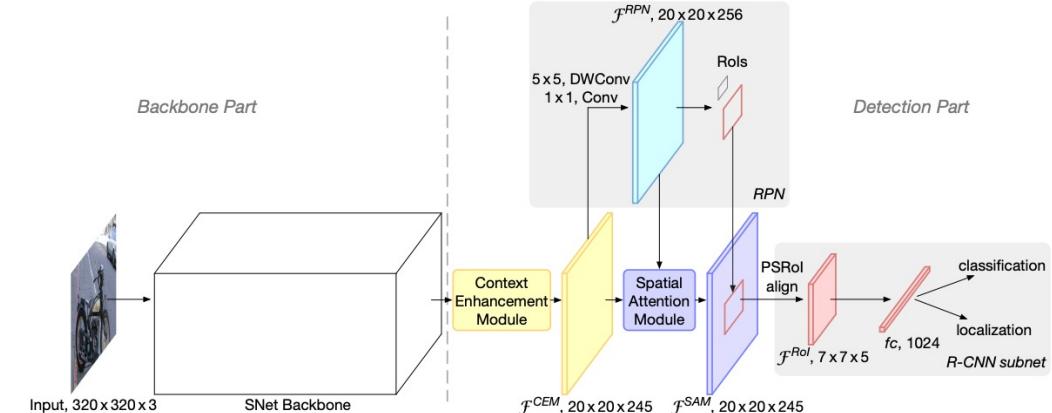
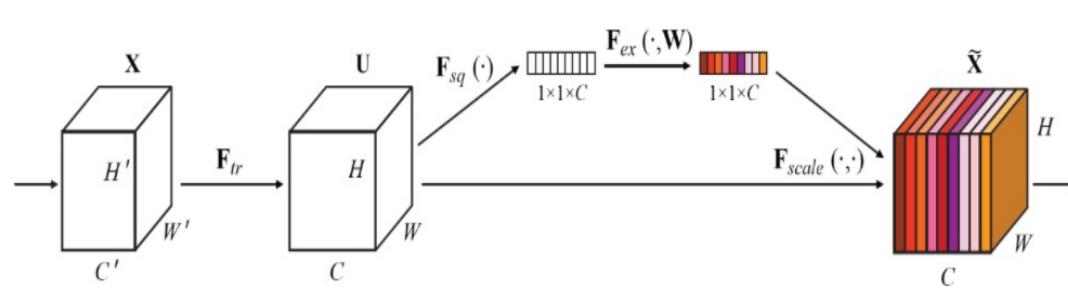
# Attention Module & Normalization

Attention Module이란 자연어처리에서 온 개념으로 디코더에서 출력 단어를 예측하는 데 예측해야 할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중(Attention)해서 보게 된다는 개념에서 나온 것으로 CNN에서는 feature 간 correlation을 캡처하는 데 유용한 기법

Yolo v4에서는 Squeeze-and Excitation Module(SE), Spatial Attention Module(SAM) 사용

Normalization 이란 입력 데이터의 특성이 단위를 무시할 수 있도록 원하는 범위로 정규화하는 것.

가장 많이 사용되는 Batch Normalization 외에도 Filter Response Normalization(FRN), Cross-Iterative Batch Normalization(CBN) 사용



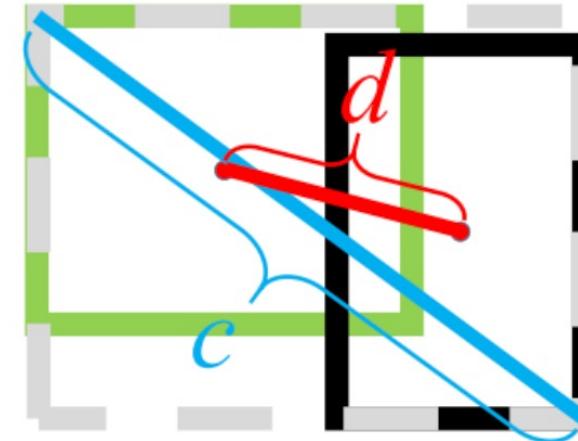
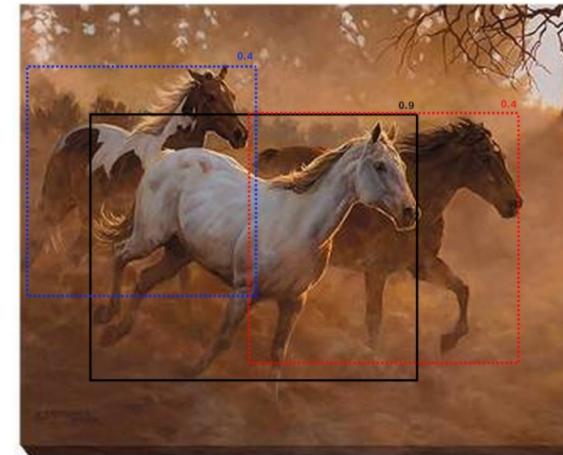
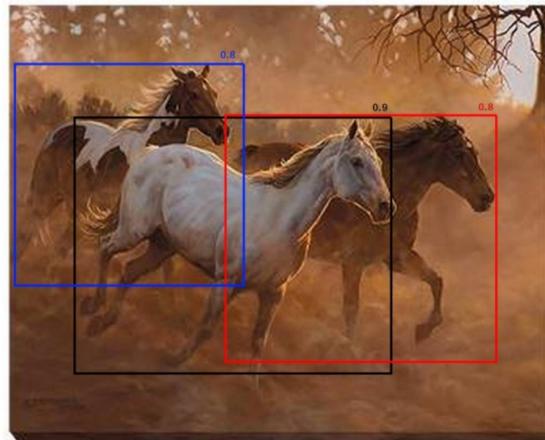
# Post Processing

Post Processing이란

Yolo v3에도 사용되었던 NMS, 이를 발전시킨 Soft NMS, DIoU NMS 등을 사용

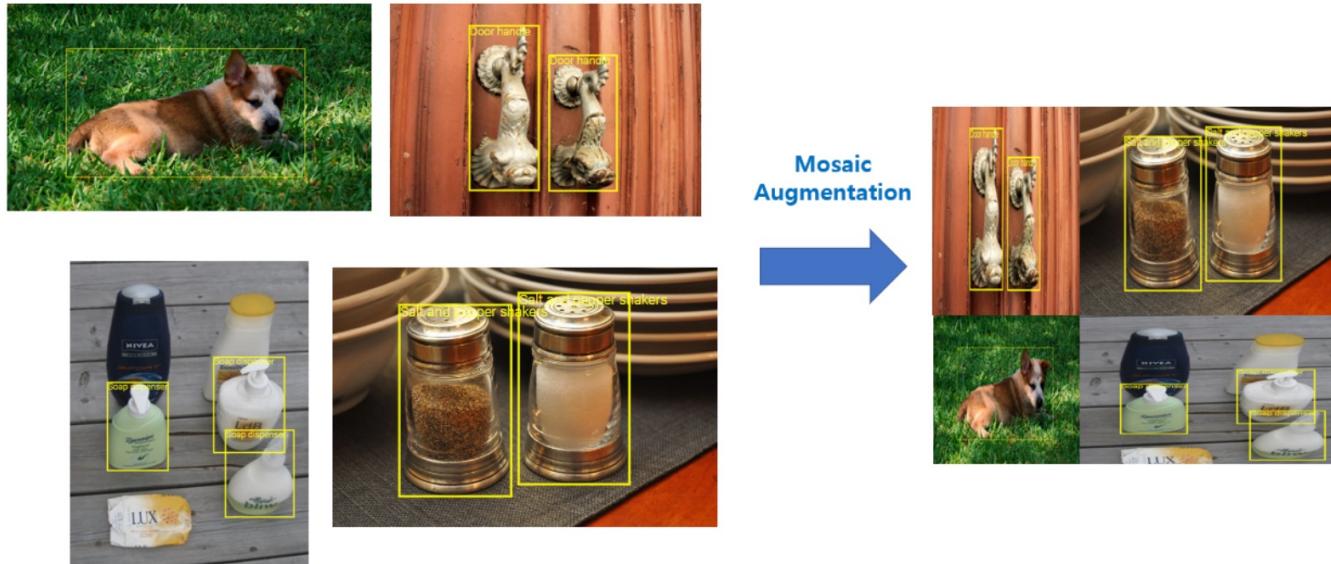
Soft NMS는 일정 비율 이상인( $IoU > threshold$ ) 겹치는 bbox들의 confidence를 0으로 만들어 없애지 않고 confidence를 줄여서 최종 mAP를 향상시키는 개념

DIoU NMS는 IoU 대신 DIoU를 사용하여 NMS 진행



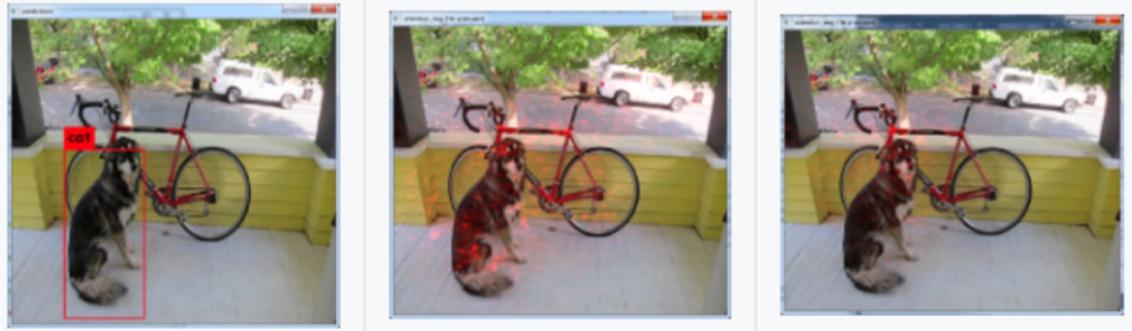
GT 박스와 Deteced 박스, 그리고 C박스

## Mosaic Augmentation



[Mosaic Augmentation]

| Adversarial attack   | Attention during training   | Attention during training on Adversarial-attacked image   |
|--|---|---|
| train already trained model for 500 iteration, but optimize the input image instead of weights (weights are freezed) #5105 | [net] adversarial_lr=0.05<br>attention=1 the network sees dog/bicycle/car | [net] adversarial_lr=0.05<br>attention=1 (image from the first column) the network sees cat here, without dog/bicycle/car |



As you can see in the edited image (adversarial attack) in the 1st/3d column, the network doesn't pay attention on dog/bicycle/car, because network thinks that that there are no dog/bicycle/car, and there is a cat instead of a dog. So network should be trained on this augmented image to pay attention to the more obvious details, as here you can clearly see the dog/bicycle/car.

*[Self-Adversarial Training]*

## Additional improvements

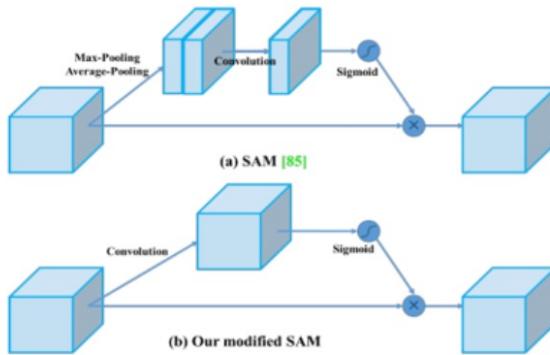


Figure 5: Modified SAM.

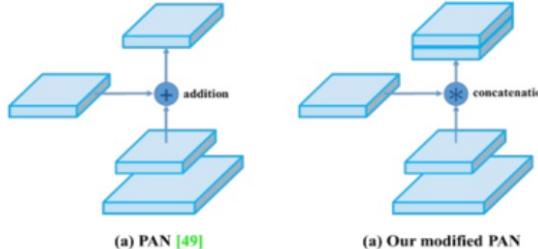


Figure 6: Modified PAN.

- Modify some existing methods for efficient training and detection
  - Modified SAM
  - Modified PAN
  - Cross mini-Batch Normalization (CmBN)

BN [32] – assume a batch contains four mini-batches

|                        |                              |                              |                            |
|------------------------|------------------------------|------------------------------|----------------------------|
| accumulate $W^{(0,3)}$ | accumulate $W^{(0,3 - 1,2)}$ | accumulate $W^{(0,3 - 1,1)}$ | accumulate $W^{(0,3 - 0)}$ |
| calculate $BN^{(0,3)}$ | calculate $BN^{(0,2)}$       | calculate $BN^{(0,1)}$       | calculate $BN^{(0)}$       |
| normalize BN           | normalize BN                 | normalize BN                 | normalize BN               |
|                        |                              |                              | update $W, ScaleShift$     |

CBN [89] – assume cross four iterations

|                               |                               |                               |                             |
|-------------------------------|-------------------------------|-------------------------------|-----------------------------|
| update $W^{(0,3)}$            | update $W^{(0,2)}$            | update $W^{(0,1)}$            | update $W^{(0)}$            |
| accumulate $BN^{(0,3 - 1,4)}$ | accumulate $BN^{(0,2 - 1,5)}$ | accumulate $BN^{(0,1 - 1,4)}$ | accumulate $BN^{(0 - 1,3)}$ |
| normalize BN                  | normalize BN                  | normalize BN                  | normalize BN                |
| update ScaleShift             | update ScaleShift             | update ScaleShift             | update ScaleShift           |



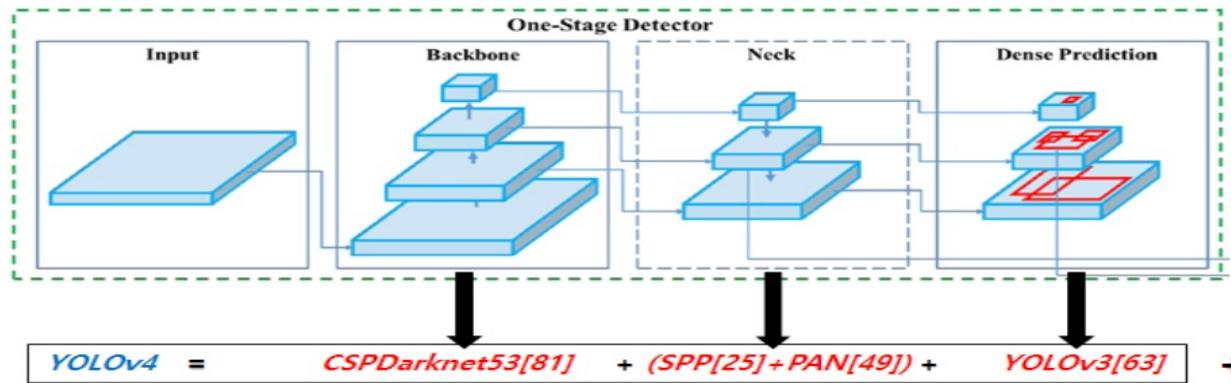
CmBN – assume a batch contains four mini-batches

|                         |                               |                               |                             |
|-------------------------|-------------------------------|-------------------------------|-----------------------------|
| accumulate $W^{(0,3)}$  | accumulate $W^{(0,3 - 1,2)}$  | accumulate $W^{(0,3 - 1,1)}$  | accumulate $W^{(0,3 - 0)}$  |
| accumulate $BN^{(0,3)}$ | accumulate $BN^{(0,3 - 1,2)}$ | accumulate $BN^{(0,3 - 1,1)}$ | accumulate $BN^{(0,3 - 0)}$ |
| normalize BN            | normalize BN                  | normalize BN                  | normalize BN                |
|                         |                               |                               | update $W, ScaleShift$      |

Figure 4: Cross mini-Batch Normalization.

[Modified SAM, PAN, CmBN]

## YOLOv4의 구성 요소 및 사용한 기법들 정리



### 최종적으로 선택된 기법들

- backbone
  - CSPDarknet53
- neck
  - additional blocks: SPP
  - path-aggregation blocks: PANet
- head
  - YOLOv3(anchor-based)

Cross-GPU Batch Normalization (CGBN 또는 SyncBN) 또는 고가의 특수 장비 등을 사용하지 않으므로, GTX 1080Ti 또는 RTX 2080Ti 등의 기준에 사용하던 GPU를 이용하여 누구나 최신의 결과를 재연 가능

### 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

Yolo v3보다 AP를 10 가량 높임

### MS COCO Object Detection

