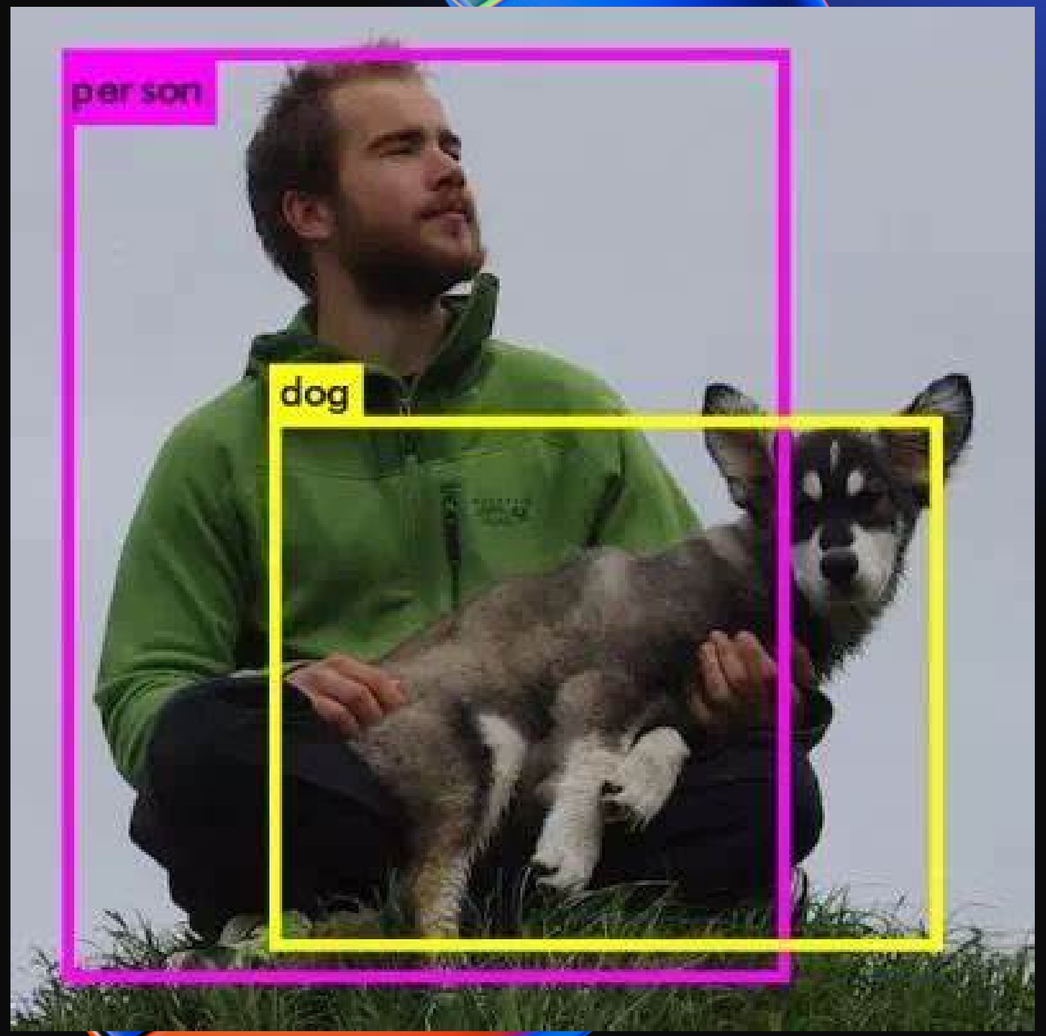
YOLO FAMILY

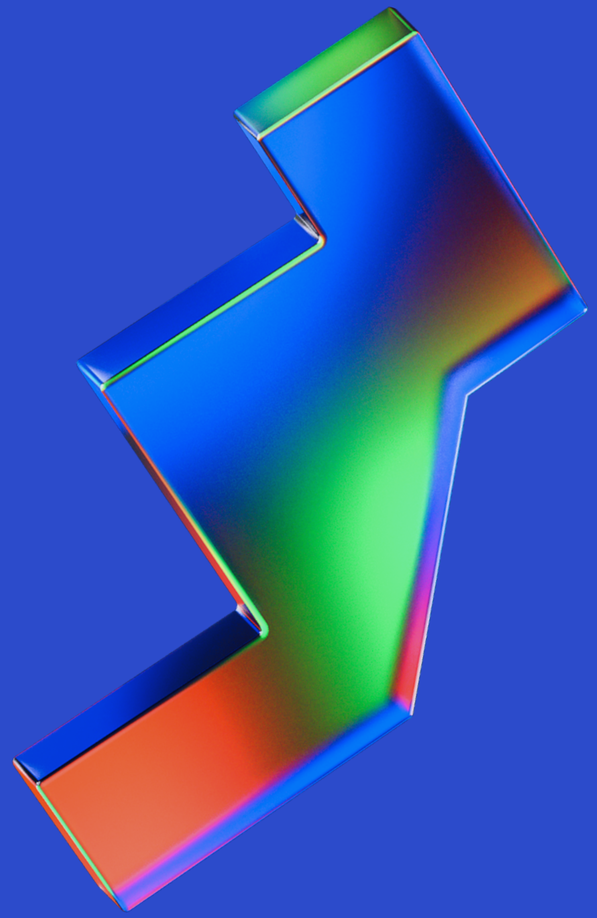
YOLOV4

PART TWO

Additional Improvements



Agenda



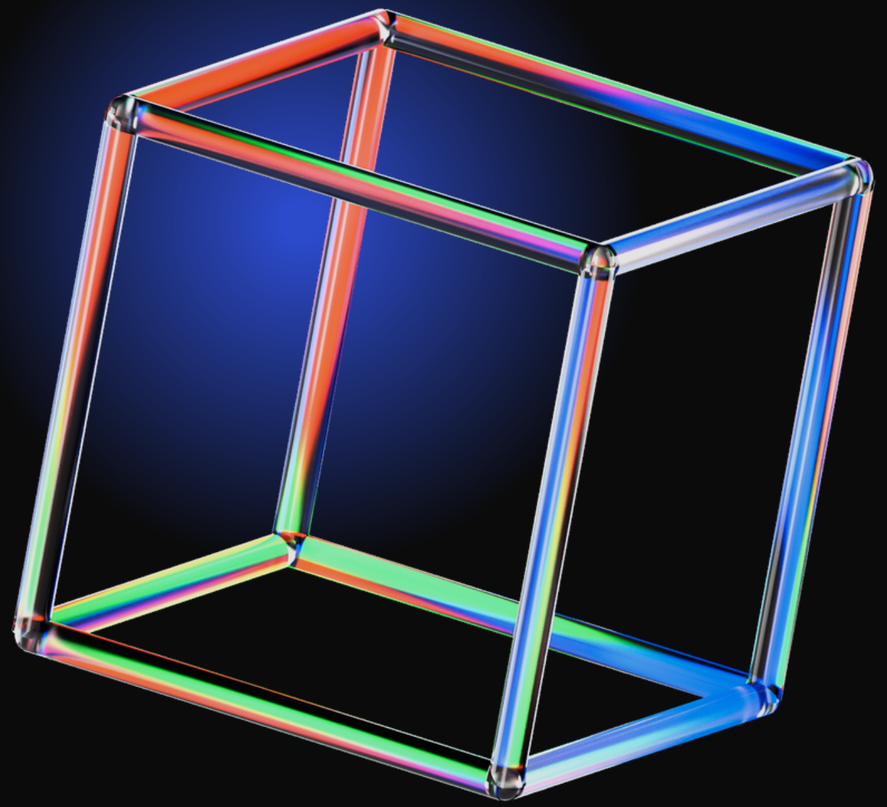
TOPICS COVERED

Bag of freebies (BoF)

Bag of specials (BoS)

Architecture Breakdown

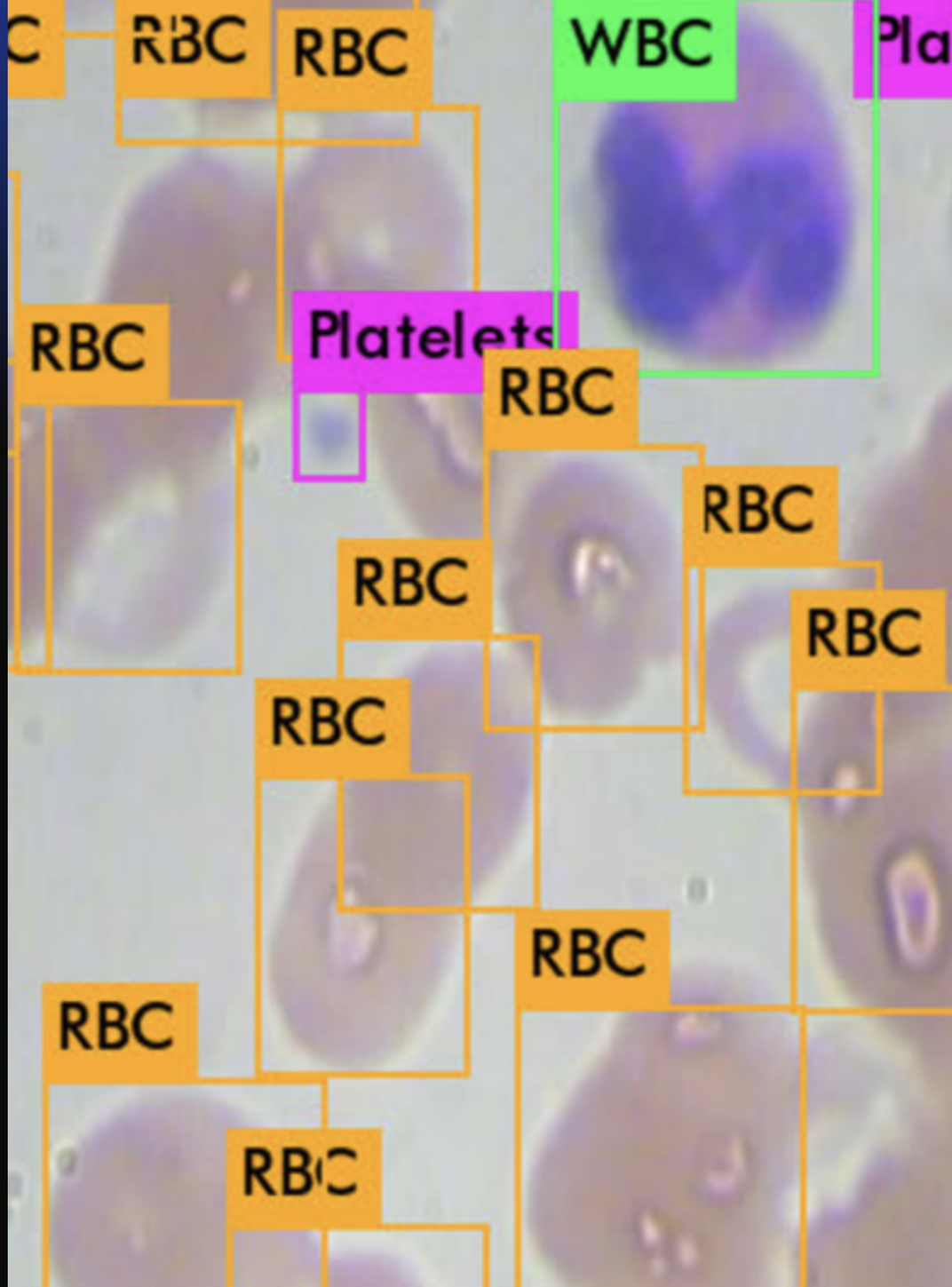
Experiments and results

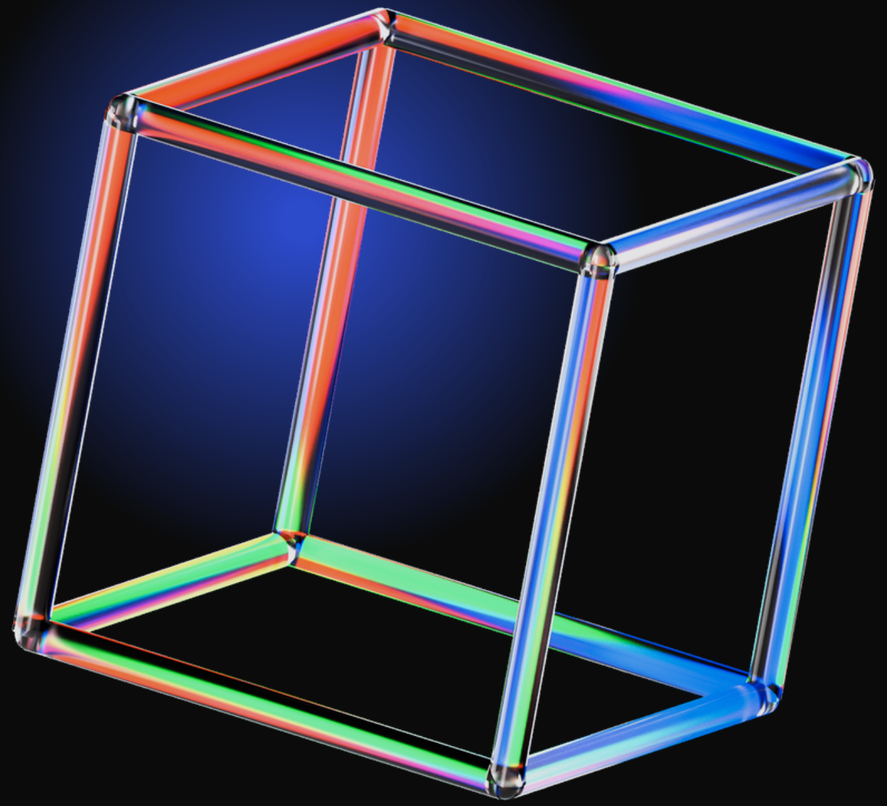


Introduction

Introduction

- **YOLO v3** is not the last product of the YOLO family
- Several versions were later published, such as *YOLO v4*, *YOLO v5* and *PP-YOLO* , all by different authors, in 2020 at a very short distance from each other.
- **YOLOv4 makes realtime detection a priority** and conducts training on a single GPU.
- The authors' intention is for vision engineers and developers to easily use their YOLOv4 framework in custom domains.

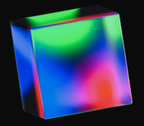




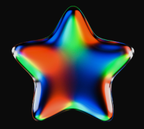
BoF and BoS

BAG OF FREEBIES AND BAG OF SPECIALS

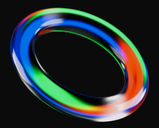
BoF and BoS for Backbone and Detector



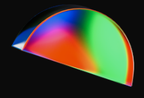
OBJECT DETECTION NETWORK CONSISTS OF BACKBONE AND DETECTOR.



BOF ARE THE TECHNIQUES WITHOUT AFFECTING THE ARCHITECTURE.



BOTH BACKBONE AND DETECTOR CAN USE BOF AND BOS.



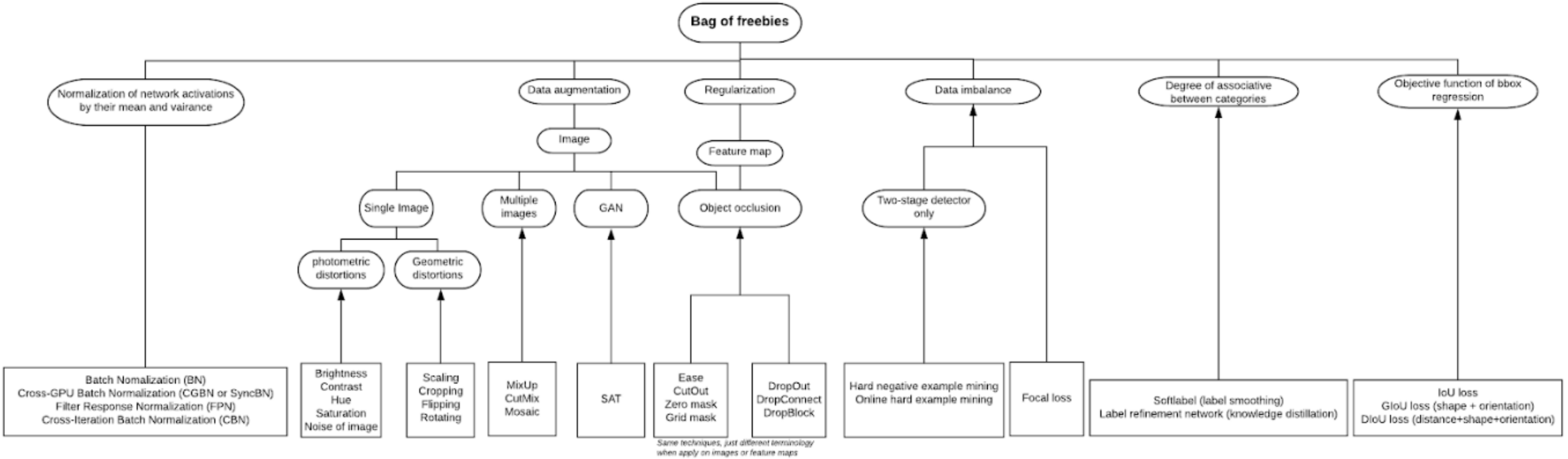
BOS ARE THE TECHNIQUES THAT WILL AFFECT THE ARCHITECTURE.

Bag of Freebies

- A set of methods that only change the training strategy or only increase the training cost is referred to as “bag of freebies.” (nothing to do with inference).

- BoF are the techniques without affecting the architecture.

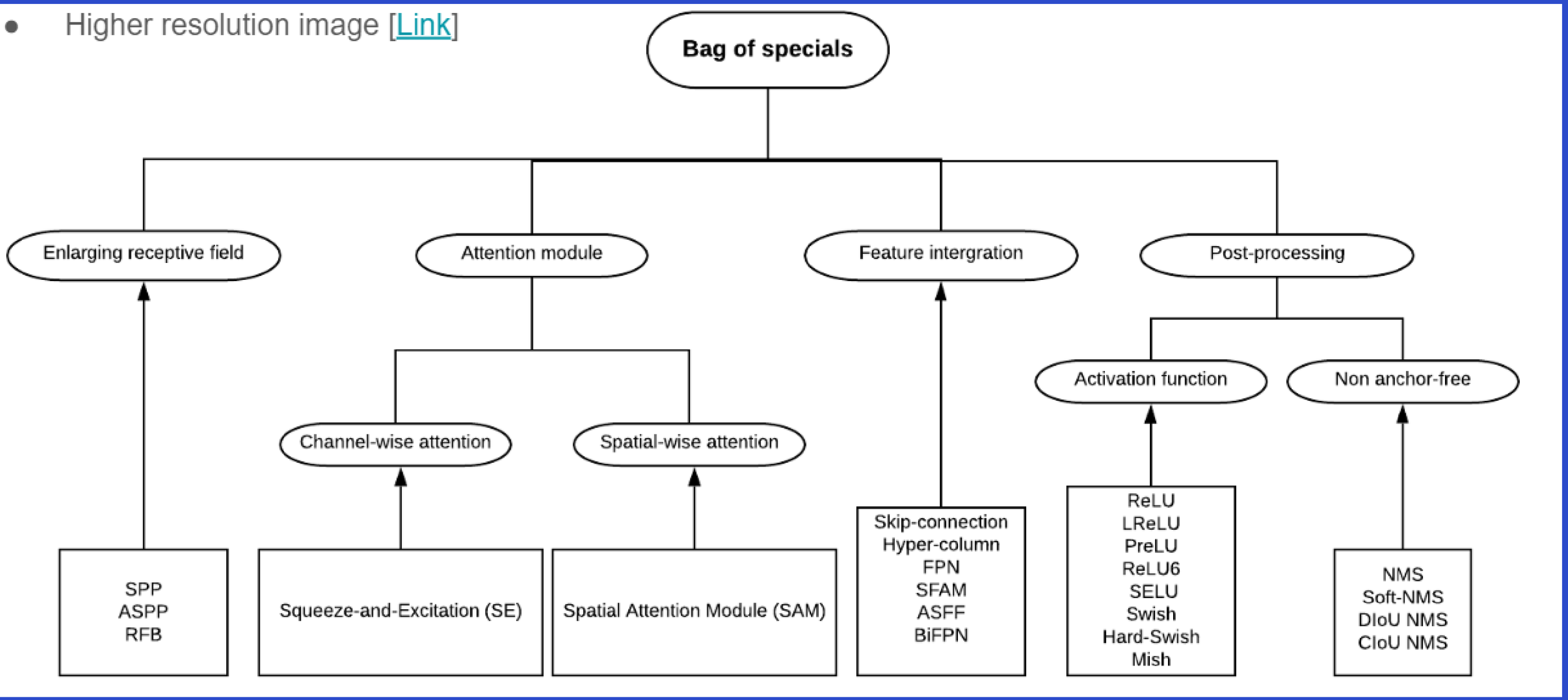
- So **bag** refers to a set of methods or strategies, and freebies means that your inference accuracy goes up without any cost to your hardware, So essentially you are getting the additional performance for **FREE**.


Bag of freebies
Normalization of network activations by their mean and vairance
Data augmentation
Regularization
Data imbalance
Degree of associative between categories
Objective function of bbox regression
Image
Feature map
Single Image
Multiple images
GAN
Object occlusion
Two-stage detector only
photometric distortions
Geometric distortions
Batch Nomalization (BN)
Cross-GPU Batch Normalization (CGBN or SyncBN)
Filter Response Normalization (FPN)
Cross-Iteration Batch Normalization (CBN)
Brightness
Contrast
Hue
Saturation
Noise of image
Scaling
Cropping
Flipping
Rotating
MixUp
CutMix
Mosaic
SAT
Ease
CutOut
Zero mask
Grid mask
DropOut
DropConnect
DropBlock
Hard negative example mining
Online hard example mining
Focal loss
Softlabel (label smoothing)
Label refinement network (knowledge distillation)
IoU loss
GIoU loss (shape + orientation)
DIoU loss (distance+shape+orientation)
Same techniques, just different terminology when apply on images or feature maps

Bag of Specials

- **Specials** refers to getting something of value at a discount or for cheap.
- Analogously the set of modules that only increase the inference cost by a small amount but significantly improve the accuracy of object detection, are called “bag of specials” (No training).
- **BoS are the techniques that will affect the architecture.**

- Higher resolution image [Link]



Backbone

BoF:

- Data augmentation
  - Mosaic, CutMix
- Regularization
  - DropBlock
- Class label smoothing

BoS:

- Mish activation
- Cross-stage partial connections (CSP)
- Multi-input weighted residual connections (MiWRC)

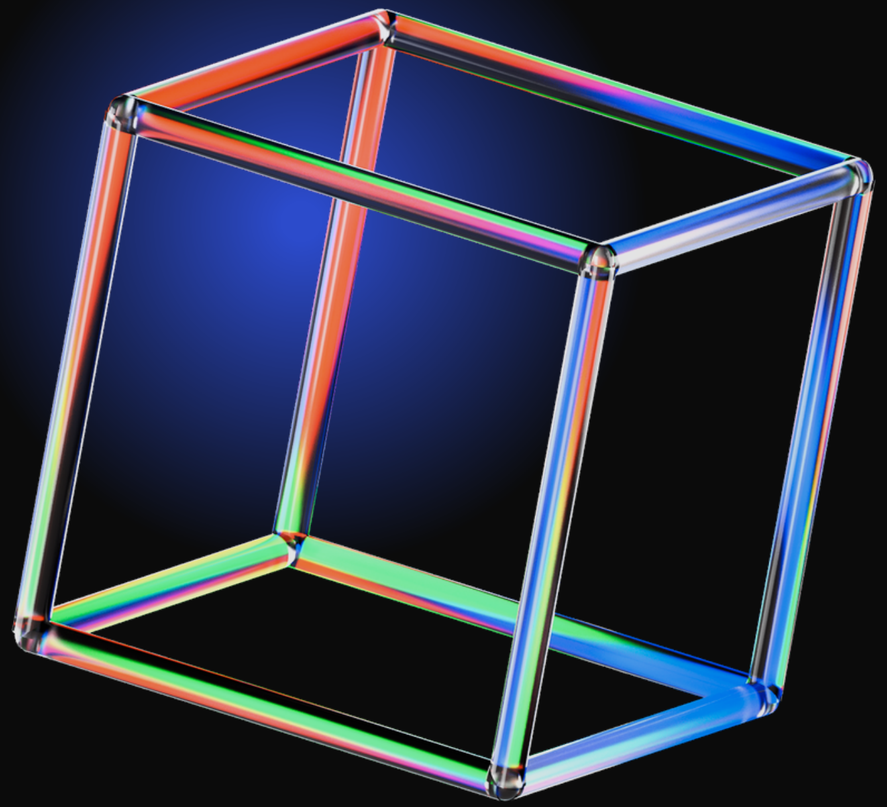
Detector

BoF:

- Data augmentation
  - Mosaic
  - Self-Adversarial Training
- CIoU-loss
- CmBN
- Eliminate grid sensitivity
- Multiple anchors for a single ground truth
- Cosine annealing scheduler
- Optimal hyper-parameters
- Random training shapes

BoS:

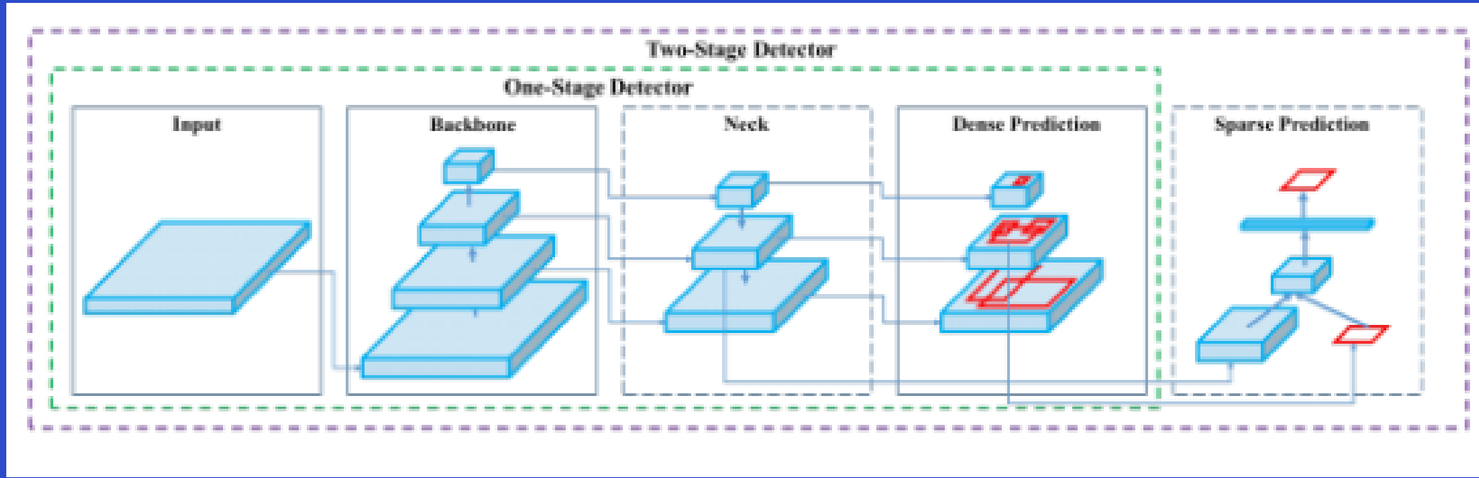
- Mish activation
- SPP-block
- SAM-block
- PAN path-aggregation block
- DIoU-NMS



Architecture

BACKBONE-NECK-HEAD

Architecture - breakdown



- **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)

Backbone

GPU

- VGG
- ResNet
- ResNeXt
- DenseNet
- EfficientNet-B0/B7
- CSPResNeXt50
- CSPDarkNet53

CPU

- SqueezeNet
- MobileNet
- ShuffleNet

Backbone

The authors considered the following backbones for the YOLOv4 object detector:

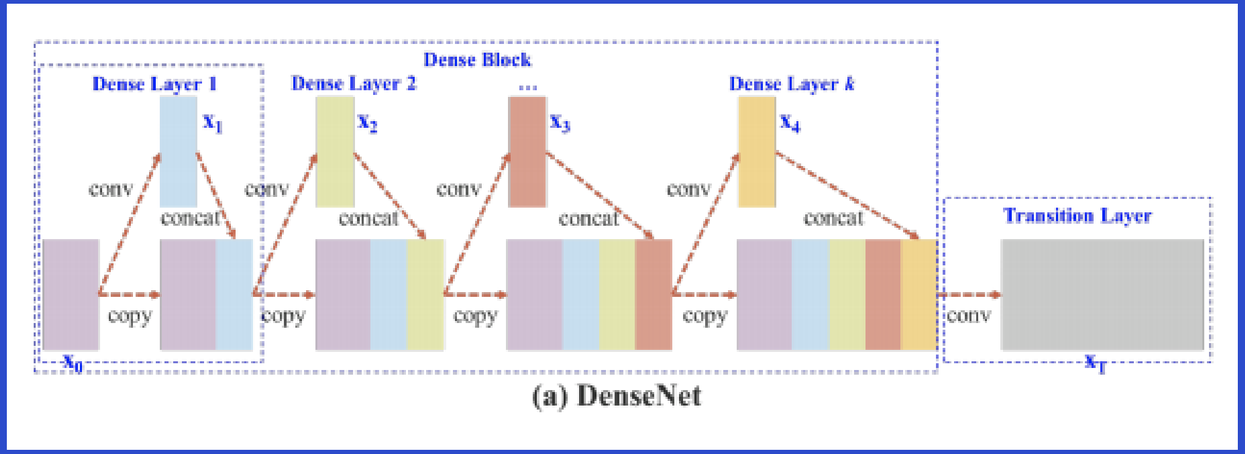
- CSPResNext50
- CSPDarknet53
- EfficientNet-B3

CSPNET (BoS - Backbone)

The CSPResNext50 and the CSPDarknet53 are both based on DenseNet.

DenseNet was designed to connect layers in convolutional neural networks with the following motivations:

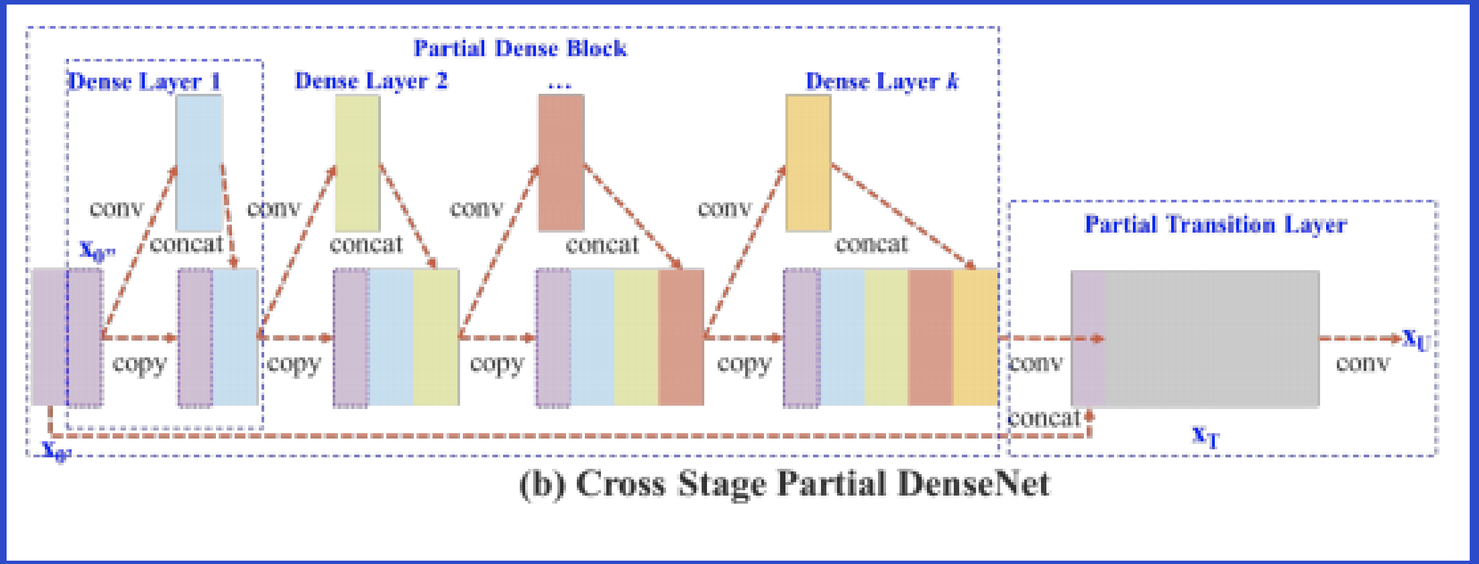
- Strong gradient flow, the error signal can be backprob easily to the early layers
- Less parameters
- Diversified features
- DenseNet works well when training data is insufficient



CSPNET

In CSPResNext50 and CSPDarknet53, the DenseNet has been edited to separate the feature map of the base layer by copying it and sending one copy through the dense block and sending another straight on to the next stage.

The idea with the CSPResNext50 and CSPDarknet53 is to remove computational bottlenecks in the DenseNet and improve learning by passing on an unedited version of the feature map.



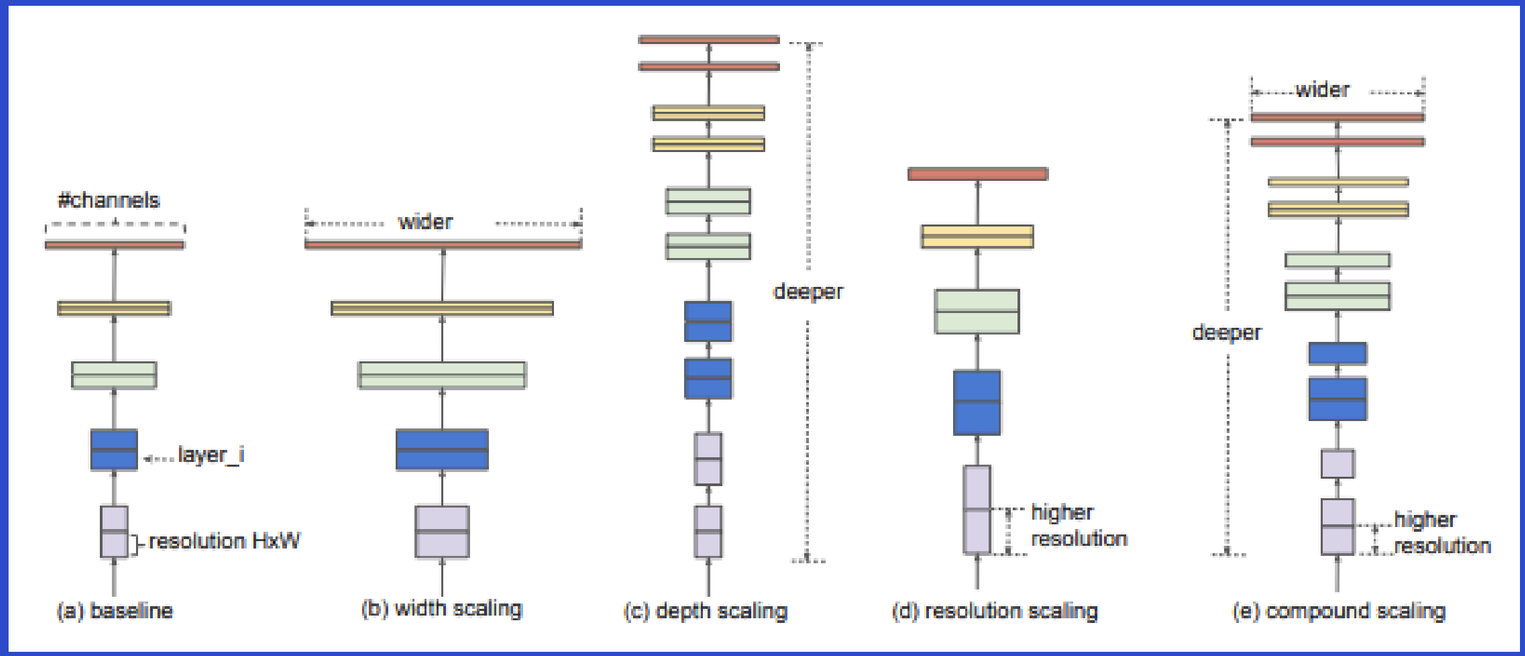
(b) Cross Stage Partial DenseNet

EfficientNet

EfficientNet was designed by Google Brain to primarily study the scaling problem of convolutional neural networks.

There are a lot of decisions you can make when scaling up your **ConvNet** including **input size**, **width scaling**, **depth scaling**, and scaling all of the above.

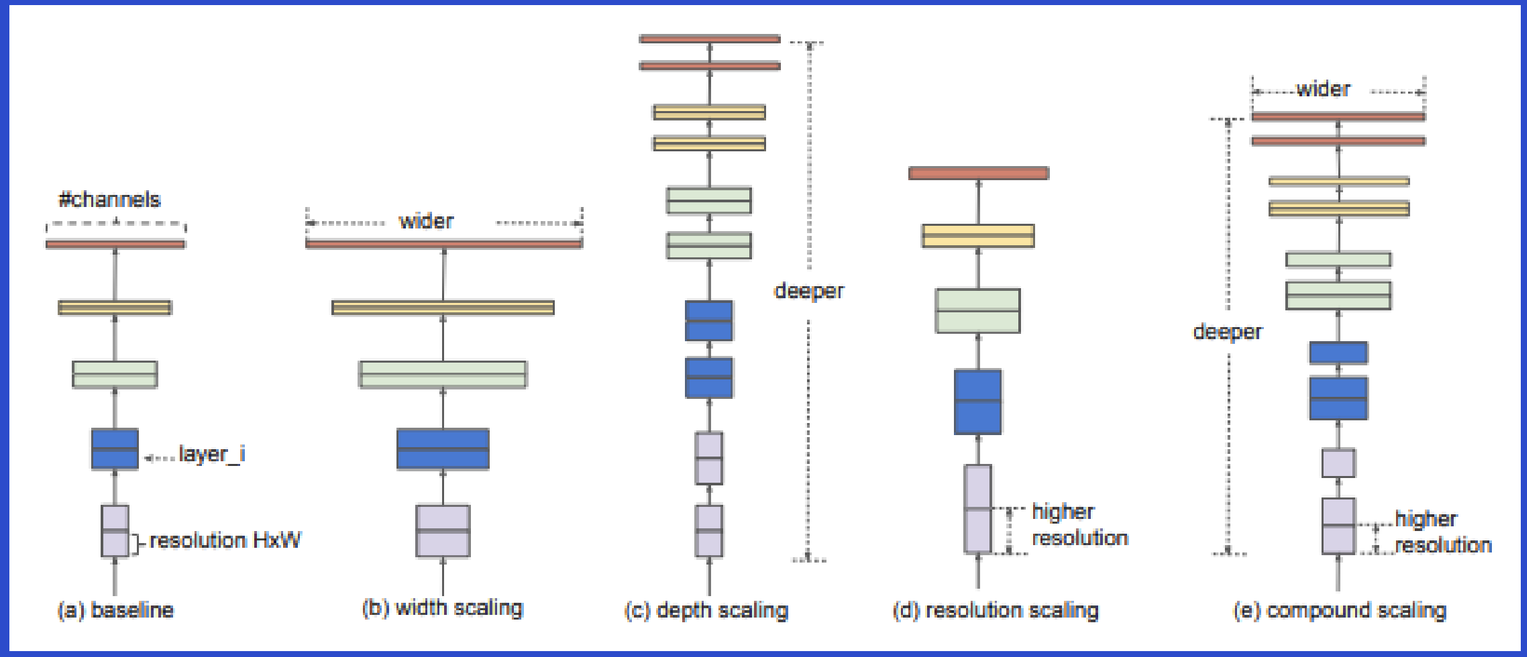
The EfficientNet paper posits that there is an optimal point for all of these and through search, they find it.



EfficientNet

EfficientNet outperforms the other networks of comparable size on image classification

YOLOv4 authors posit, however, that the other networks may work better in the object detection setting and decide to experiment with all of them.



Backbone Selection

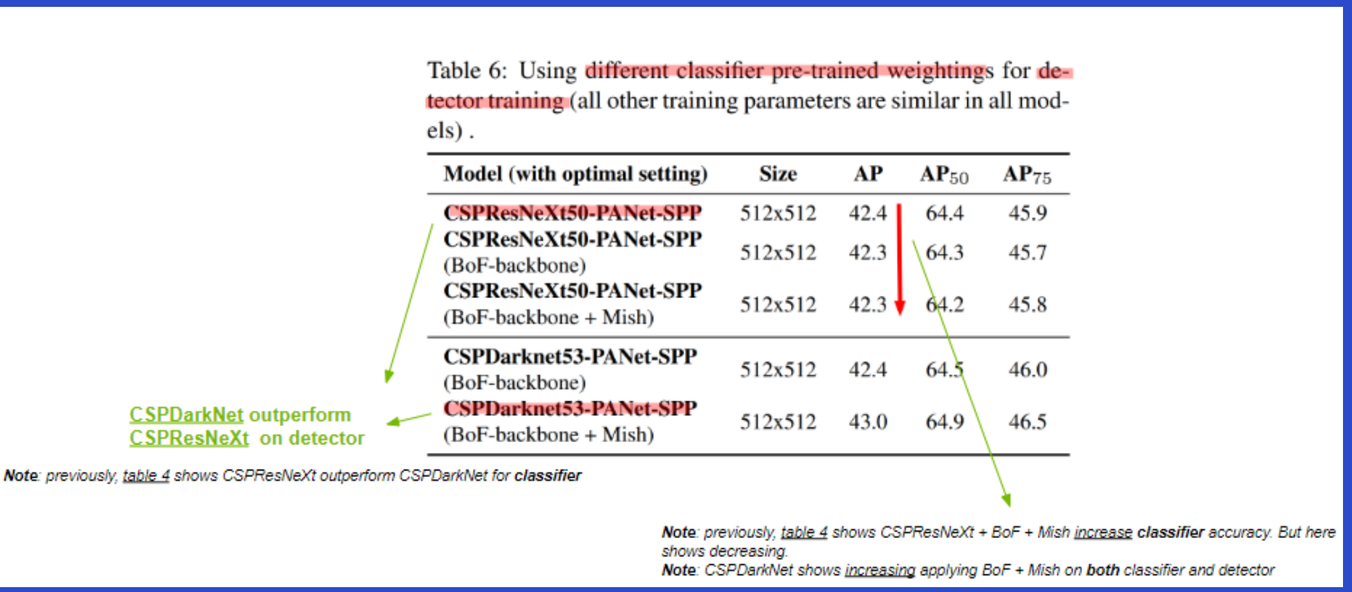


Table 6: Using different classifier pre-trained weightings for detector training (all other training parameters are similar in all models) .

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

CSPDarkNet outperform CSPResNeXt on detector

Note**: previously, table 4 shows CSPResNeXt outperform CSPDarkNet for **classifier

***Note**: previously, table 4 shows CSPResNeXt + BoF + Mish increase **classifier** accuracy. But here shows decreasing.*
***Note**: CSPDarkNet shows increasing applying BoF + Mish on **both** classifier and detector*

- The backbone network for an object detector is typically pretrained on ImageNet classification.
- It is found that CSPResNext50 is considerably better compared to CSPDarknet53 in terms of object classification on the ILSVRC2012 (ImageNet) dataset.
- However, conversely, the CSPDarknet53 is better compared to CSPResNext50 in terms of detecting objects on the MS COCO dataset.
- Note that this CSPNet can be applied to different backbones, such as ResNet, ResNeXt, and DenseNet, to become CSPResNet, CSPResNeXt, and CSPDenseNet.

Neck (subset of bag of specials)

Additional block

- SPP
- ASPP
- RFB
- SAM

Path-aggregation blocks

- FPN
- PAN
- NAS-FPN
- Fully-connected FPN
- BiFPN
- ASFF
- SFAM

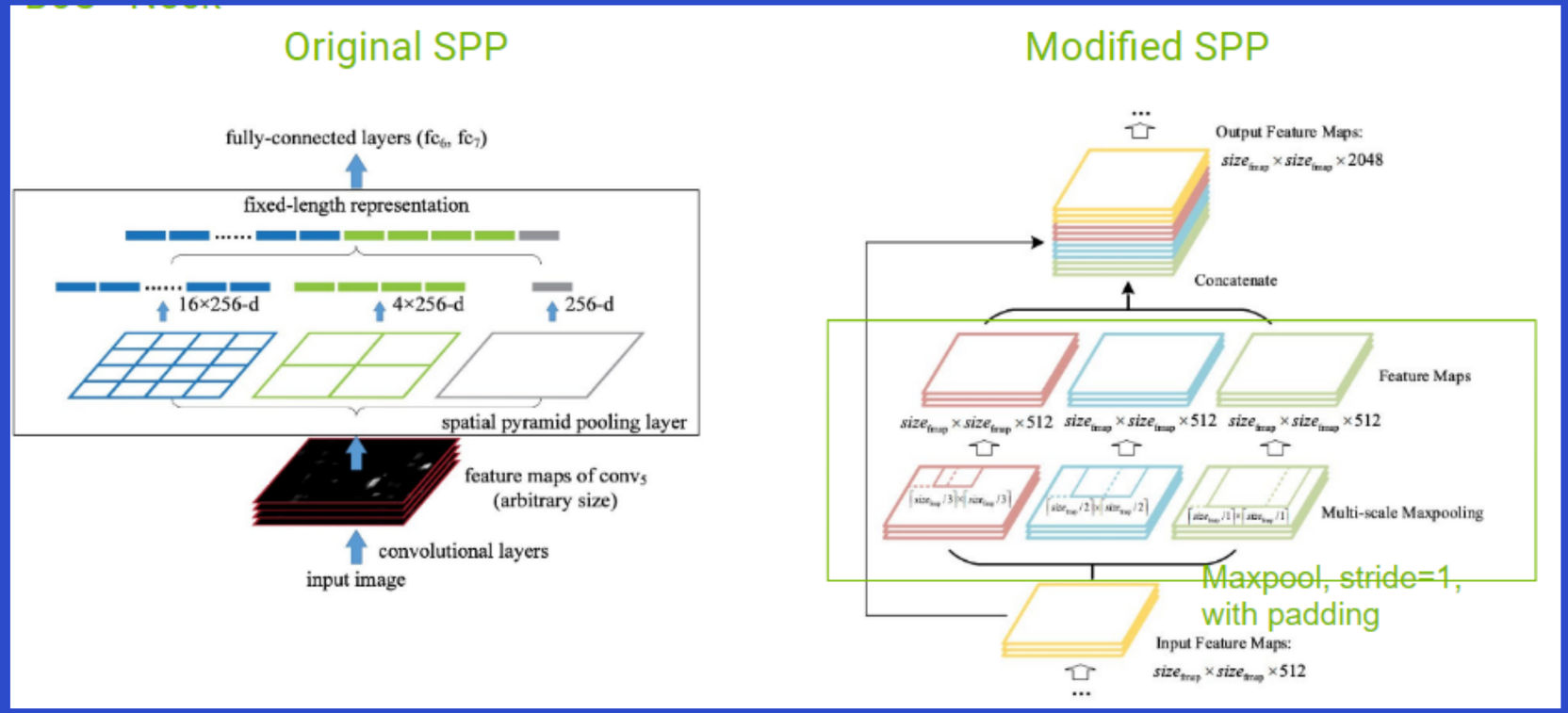
Additional Blocks (BoS)

In contrast to the image classification classifier, the detector requires the following:

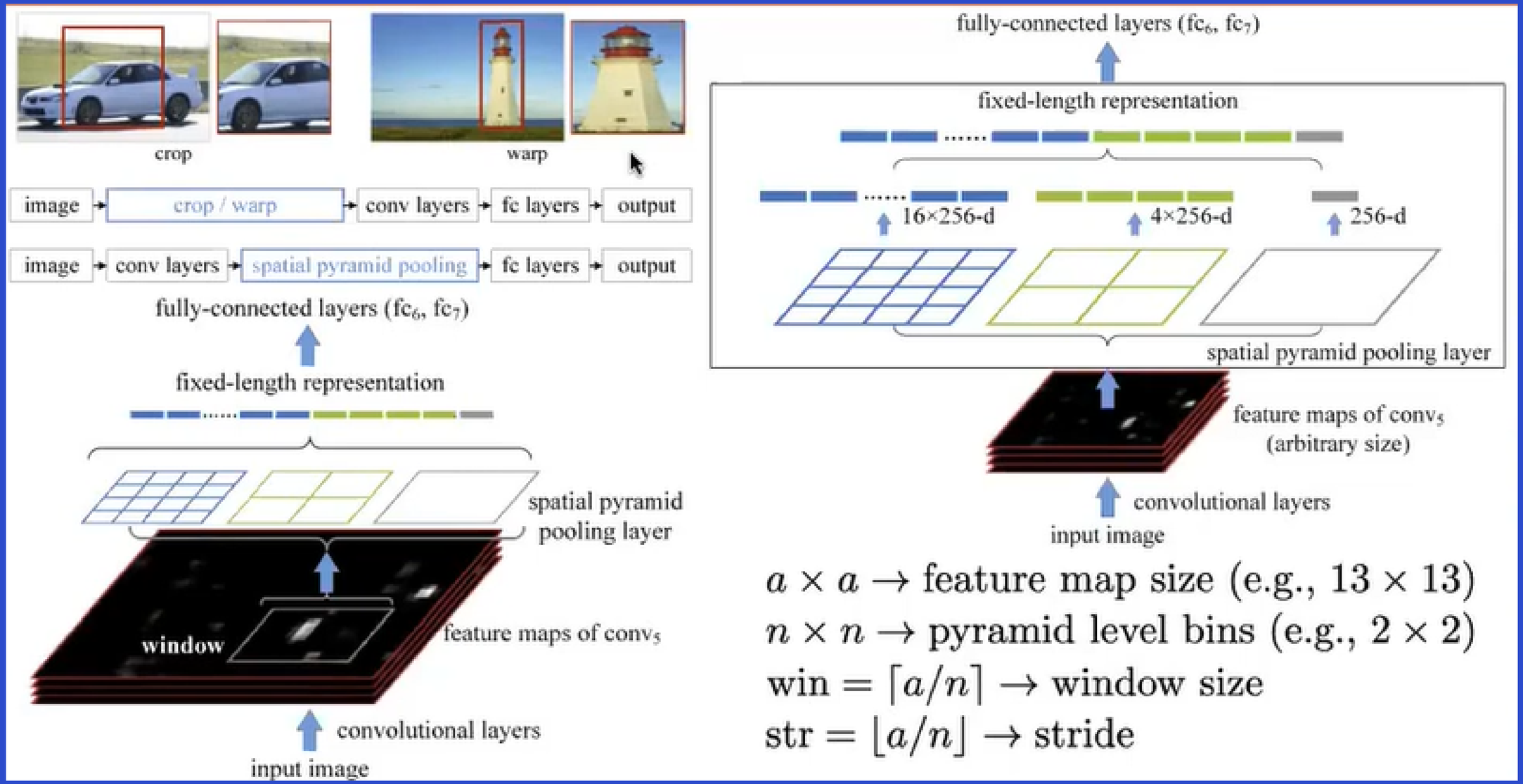
1. Higher input network size (resolution) — for detecting multiple small-sized objects.
2. More layers — for a higher receptive field to cover the increased size of input network.
3. More parameters — for greater capacity of a model to detect multiple objects of different sizes in a single image.

1. SPP(Spatial pyramid pooling)

SPP(Spatial pyramid pooling) block over the CSPDarknet53, since it significantly increases the receptive field, separates out the most significant context features and causes almost no reduction of the network operation speed.



1. SPP(Spatial pyramid pooling)



$a \times a \rightarrow$ feature map size (e.g., 13×13)

$n \times n \rightarrow$ pyramid level bins (e.g., 2×2)

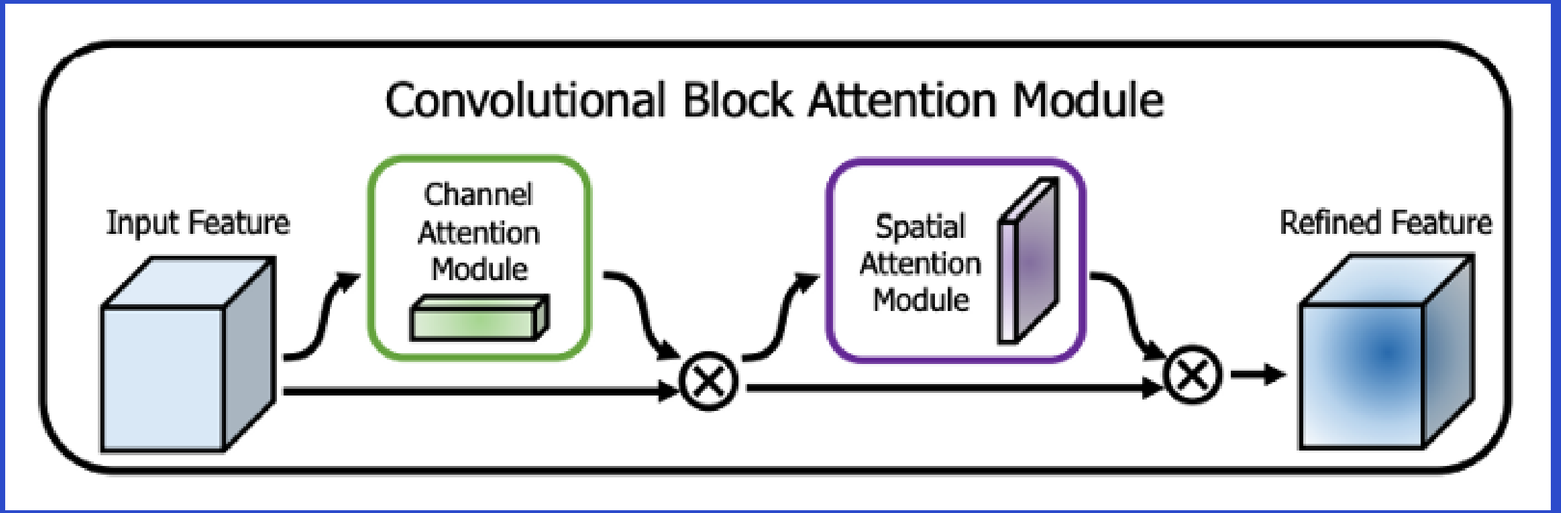
$\text{win} = \lceil a/n \rceil \rightarrow$ window size

$\text{str} = \lfloor a/n \rfloor \rightarrow$ stride

2. Spatial Attention Module

Attention modules are used to make CNN learn and focus more on the important information, rather than learning non-useful background information.

Generally, attention mechanisms are applied to spatial and channel dimensions.



2.1 Channel Attention

It is also class Squeeze and excitation module.

It helps to pay attention across the channels and identify which channel are important that others.

Squeeze

First we compress the spatial information (Squeeze) of a feature map with dimensions H x W x C to 1 x 1 x C.

We use global aveage pooling to compute the average of all feature across the H and W and divide by H multiplied by W.

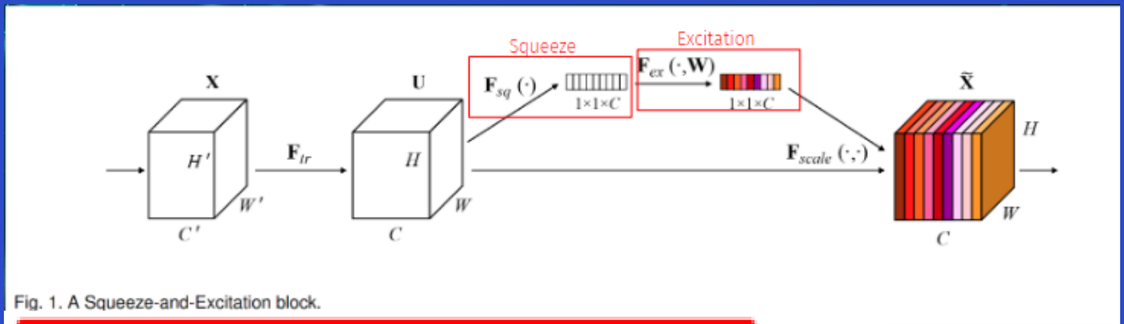


Fig. 1. A Squeeze-and-Excitation block.

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i,j). \quad (2)$$

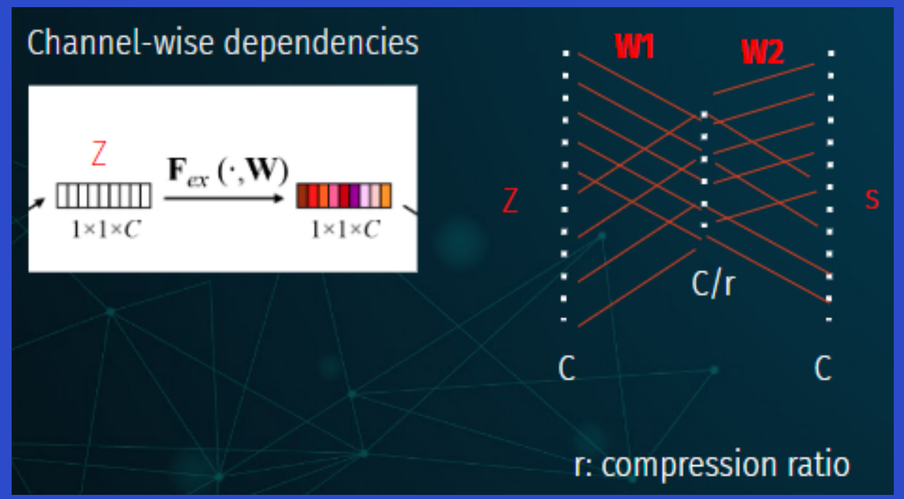
$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})), \quad (3)$$

Excitation

After we get the 1 x 1 x C from the squeeze operation we change to fully connected layer.

We add another FC layer with c/r number of neurons where 'r' is the compression ratio and apply ReLU.

Then we turn it back to C again and make it a third layer. Finally this will be passed to a sigmoid layer. The excitation block will be trained to figure out the weights w1 and w2.



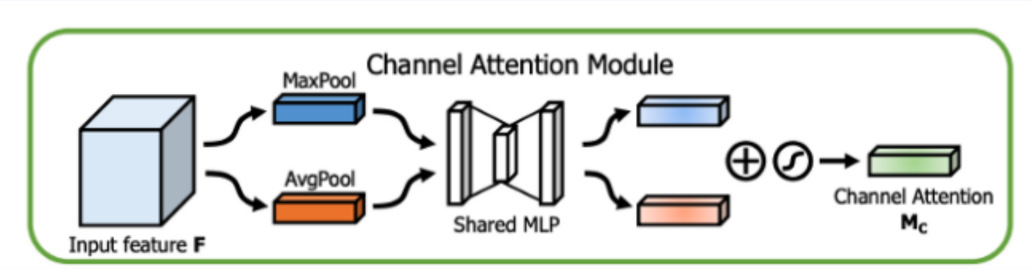
Convolutional Block Attention Module(CBAM)

Channel Attention Map

Same as Squeeze and Excitation but here with Average Pooling, Max Pooling is also added for getting more distinctive channel features.

$$\begin{aligned}\mathbf{M_c}(\mathbf{F}) &= \sigma(MLP(AvgPool(\mathbf{F})) + MLP(MaxPool(\mathbf{F}))) \\ &= \sigma(\mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{avg}})) + \mathbf{W_1}(\mathbf{W_0}(\mathbf{F^c_{max}}))),\end{aligned}$$

Here σ denotes the sigmoid function, W0 ∈ C/r×C, and W1 ∈ C×C/r. Note that the MLP weights, W0 and W1, are shared for both inputs and the ReLU activation function is followed by W0.

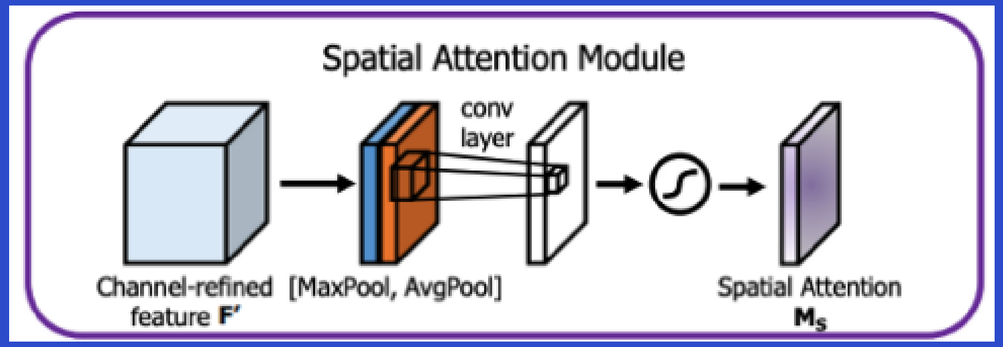


Spatial Attention Map

Take the input feature map F and generate two intermediate feature maps viz. Fsavg, and Fsmax ∈ 1xHxW. Compress the channel dimension.

$$\begin{aligned}\mathbf{M_s}(\mathbf{F}) &= \sigma(f^{7\times7}([AvgPool(\mathbf{F}); MaxPool(\mathbf{F})])) \\ &= \sigma(f^{7\times7}([\mathbf{F^s_{avg}}; \mathbf{F^s_{max}}])),\end{aligned}$$

Concatenate these two outputs **GAP**(Global Average Pool)and **MP**(Max Pooling)and pass it through a small convolutional block of 7x7 kernel size.

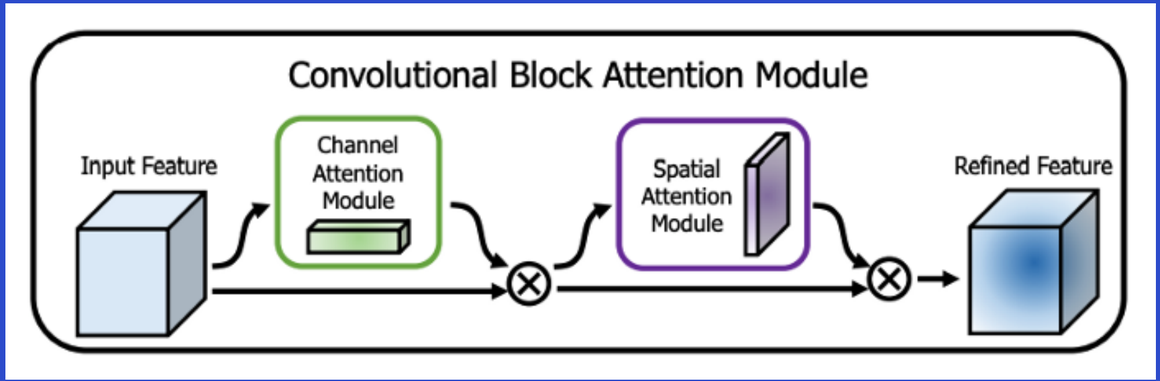


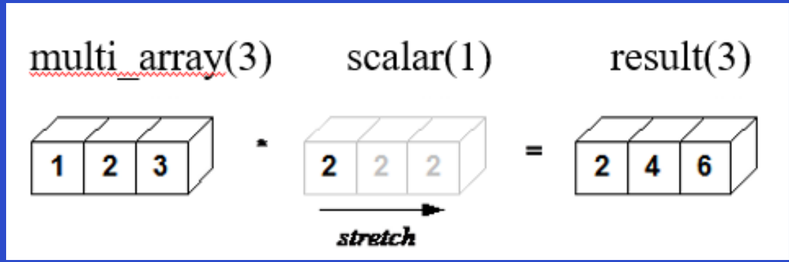
Convolutional Block Attention Module(CBAM)

Given an intermediate feature map F ∈ C×H×W as input, CBAM sequentially infers a 1D channel attention map Mc ∈ C×1×1 and a 2D spatial attention map Ms ∈ 1×H×W as shown in Fig. 6. The overall attention process can be summarized as:

$$\mathbf{F}' = \mathbf{M_c}(\mathbf{F}) \otimes \mathbf{F},$$
$$\mathbf{F}'' = \mathbf{M_s}(\mathbf{F}') \otimes \mathbf{F}',$$

Here ⊗ denotes element-wise multiplication. During multiplication, the attention values are broadcasted (copied) accordingly: channel attention values are broadcasted along the spatial dimension, and vice versa. F'' is the final refined output. Fig. 6 depicts the computation process of each attention map.

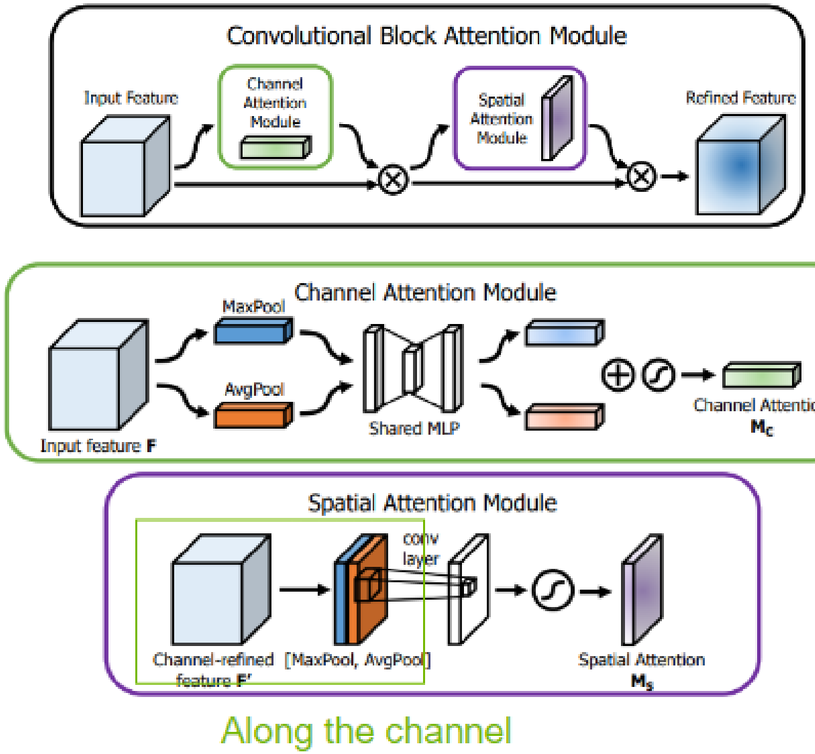




Modified Spatial Attention Module in YOLOv4

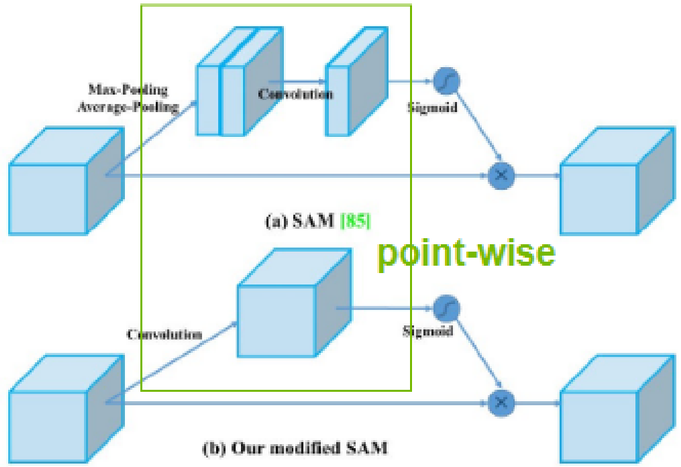
- Original CBAM is sequential order: Channel-wise -> Spatial-wise
- Spatial-wise first apply the **“max”** and **“average”** pool along the **“channels"**
- Max pooling and average pooling are replaced by convolution.

Original *CBAM

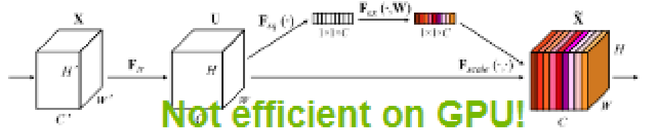


Along the channel

Modified SAM



Original SE



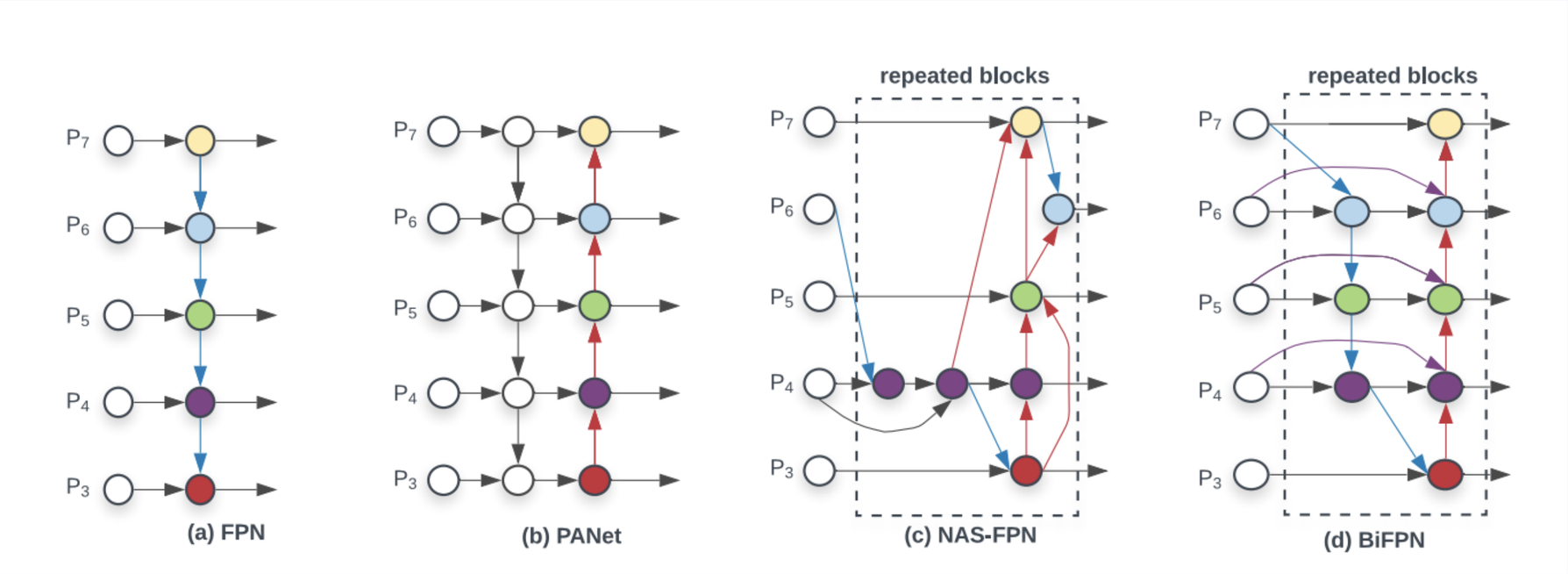
Not efficient on GPU!

Path Aggregation Blocks

The next step in object detection is to mix and combine the features formed in the CSPDarkNet53 backbone to prepare for the detection step.

YOLOv4 chooses PANet for the feature aggregation of the network.

PANet is used as the method of parameter aggregation from different backbone levels for different detector levels, instead of the FPN used in YOLOv3.



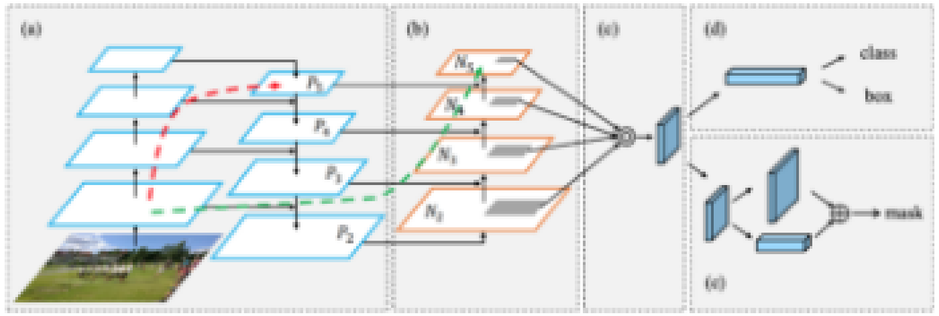
Path Aggregation Blocks

A bottom-up path (b) is augmented to make low-layer information easier to propagate to the top. In FPN, the localized spatial information traveled upward in the red arrow.

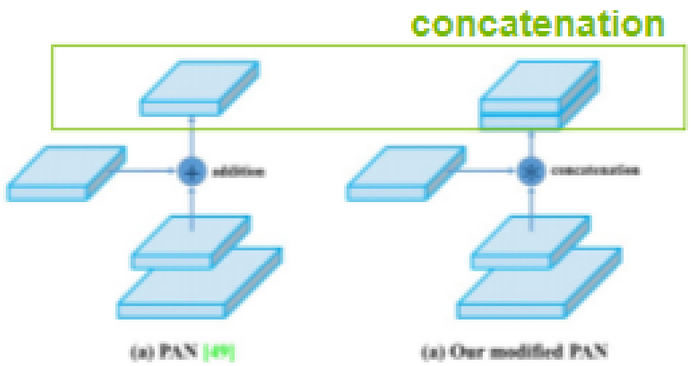
While not clearly demonstrated in the diagram, the red path goes through about **100+ layers**.

PAN introduced a short-cut path (the green path) which only takes about 10 layers to go to the top N_5 layer. This short-circuit concepts make **fine-grained localized information available to top layers**.

Original PAN



Modified PAN



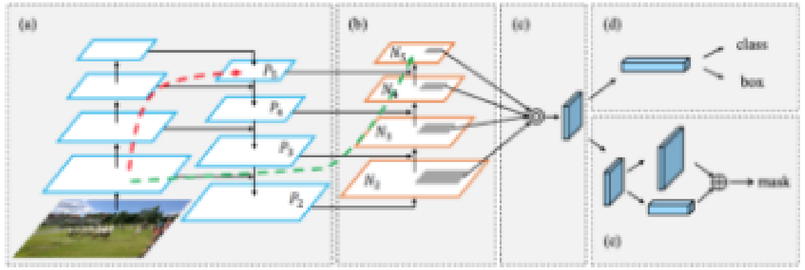
Path Aggregation Blocks

In FPN, objects are detected **separately and independently** at different scale levels.

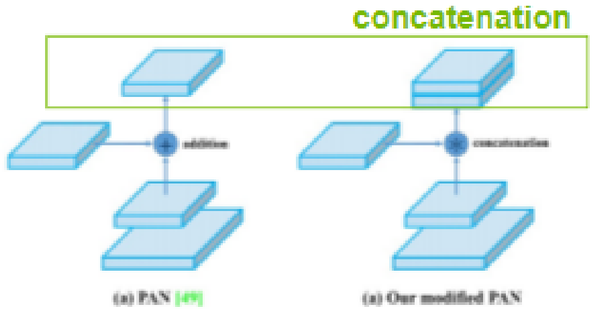
This may produce duplicated predictions and not utilize information from other feature maps.

PAN fuses the information together from all layers first using element-wise max operation

Original PAN

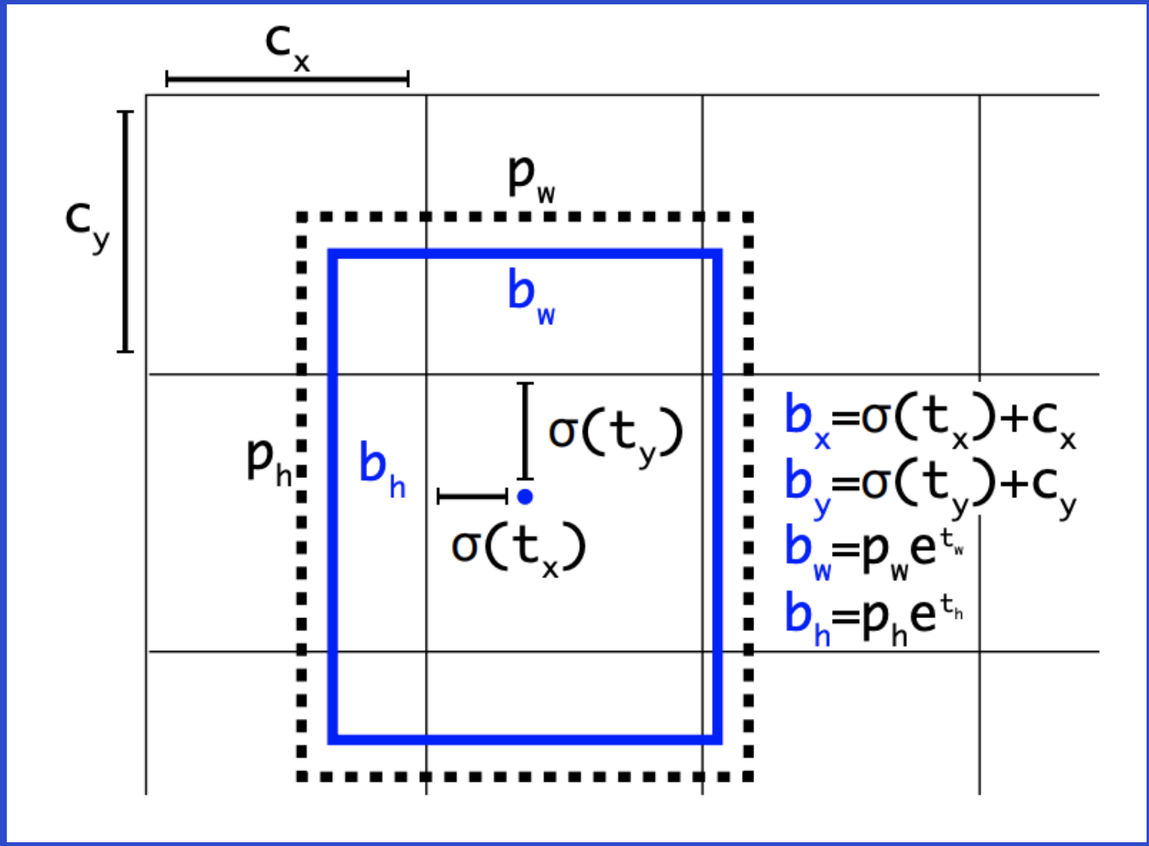


Modified PAN



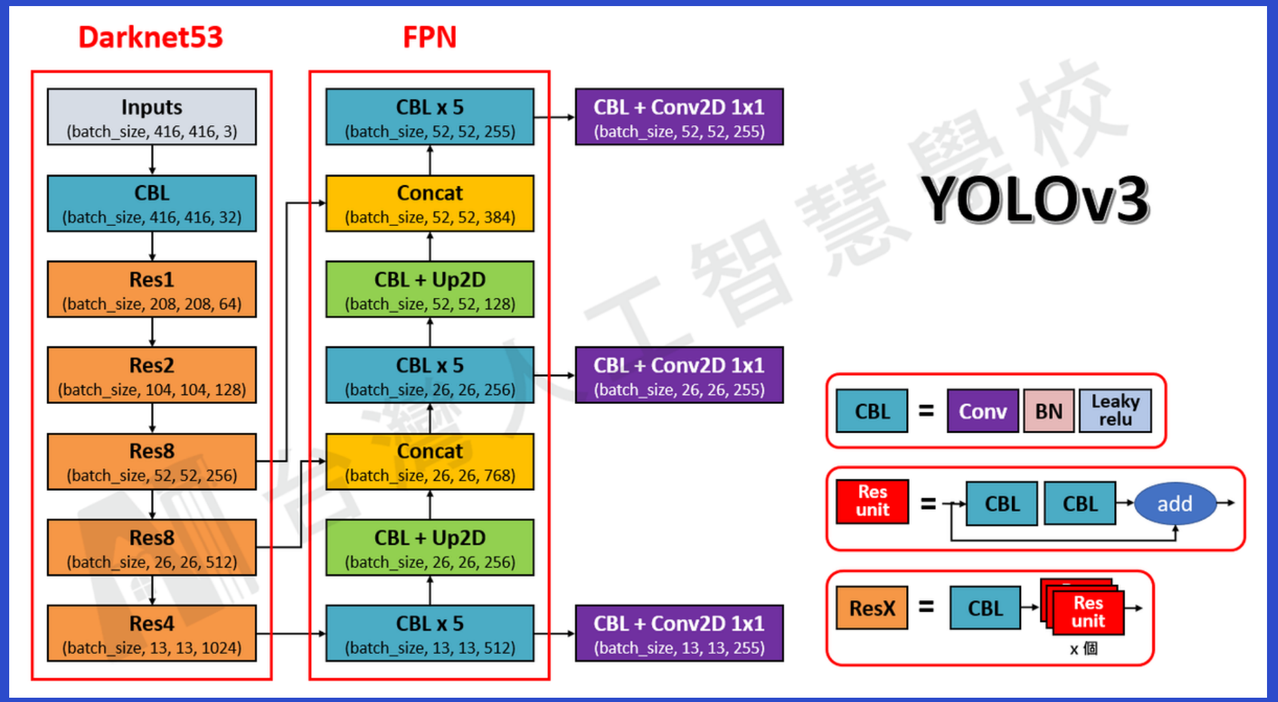
Head - The Detection Step

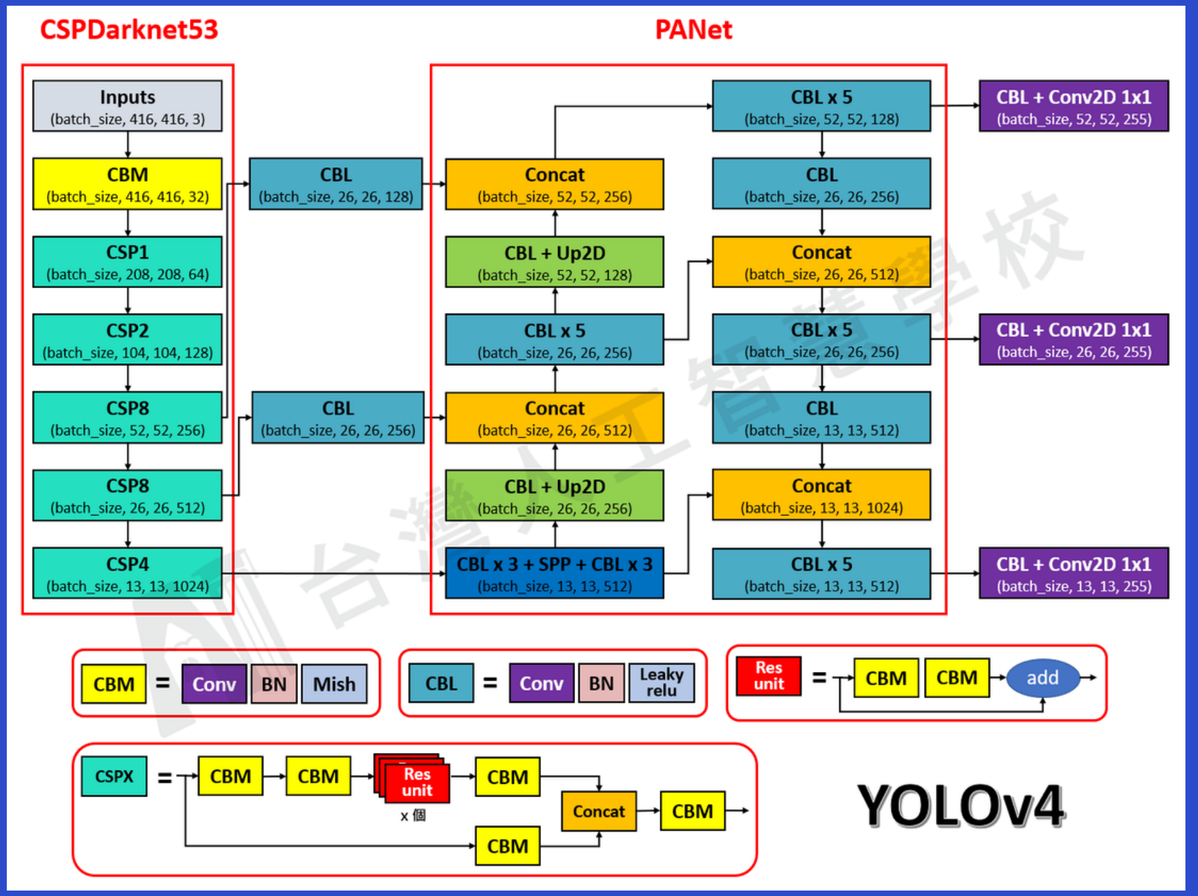
YOLOv4 deploys the same YOLO head as YOLOv3 for detection with the **anchor based detection steps, and three levels of detection granularity**.

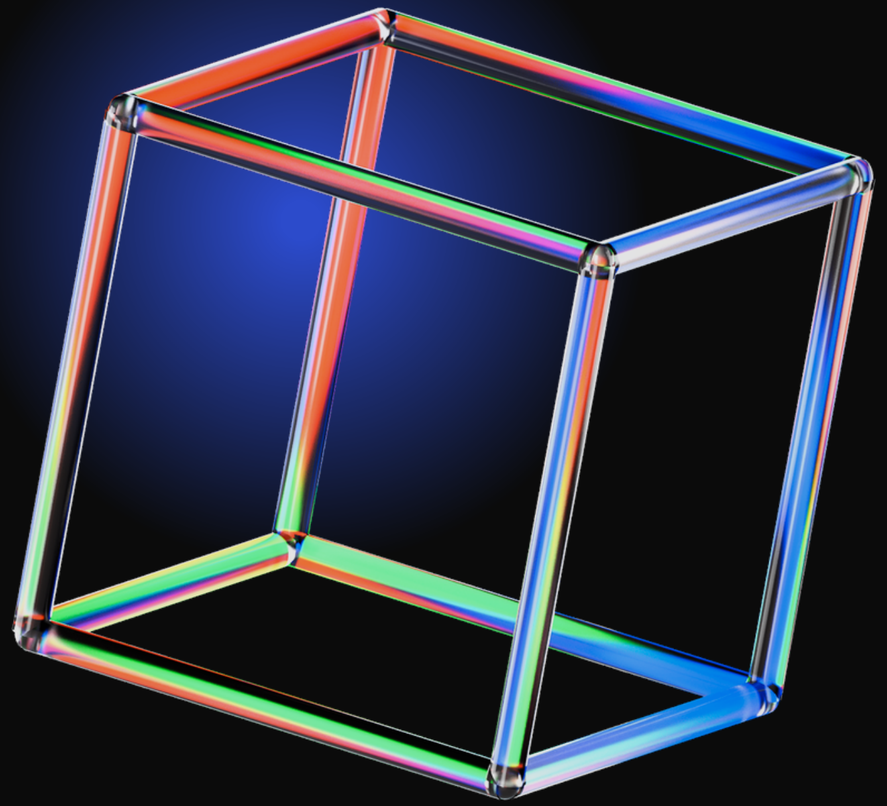


Final Architecture

Finally, CSPDarknet53, SPP additional module, PANet path-aggregation neck, and YOLOv3 (anchor based) head are chosen as the architecture of YOLOv4.







Additional Improvements

Mosaic (BoF)

Mosaic represents a new data augmentation method that mixes 4 training images.

Thus 4 different contexts are mixed, while CutMix mixes only 2 input images. This allows detection of objects outside their normal context.

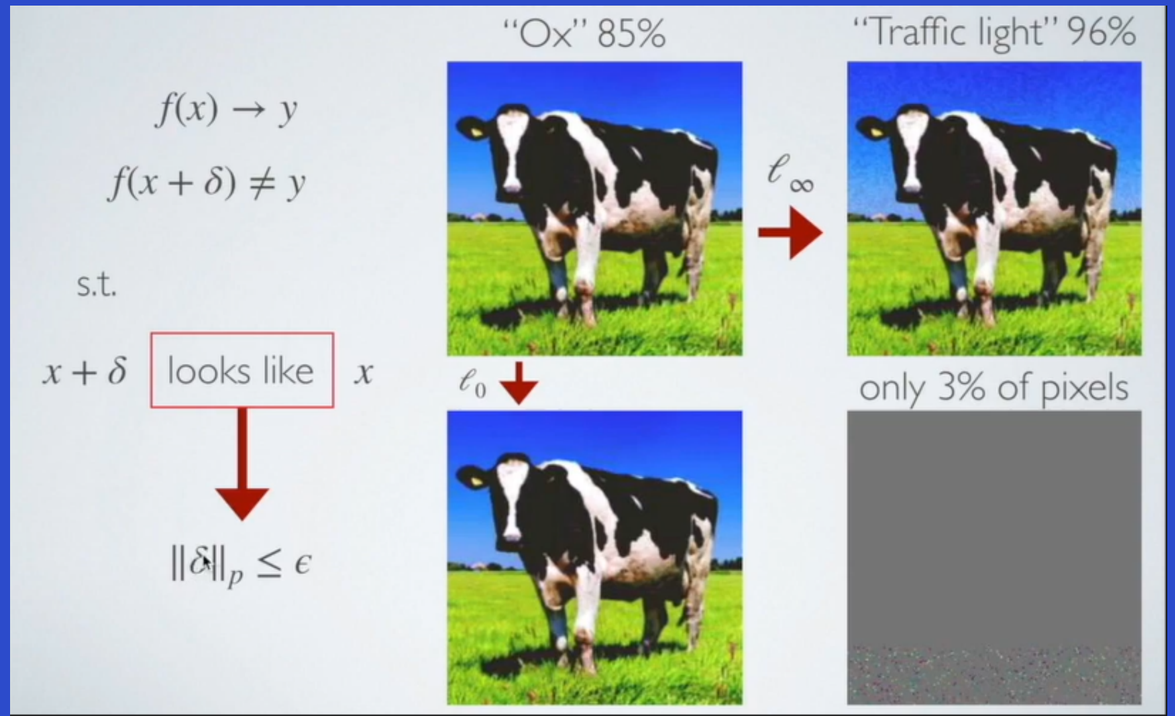
In addition, **batch normalization calculates activation statistics from 4 different images on each layer**. This significantly reduces the need for a large mini-batch size.

Self Adverserial Training (BoF)

Self-Adversarial Training (SAT) also represents a new data augmentation technique that is used to make our model generalize to adversarial examples.

It introduces perturbations on the original image which will cause the model to misclassify.

We introduce the perturbations in the form of l-norm to make our adversarial examples look more like our training example.



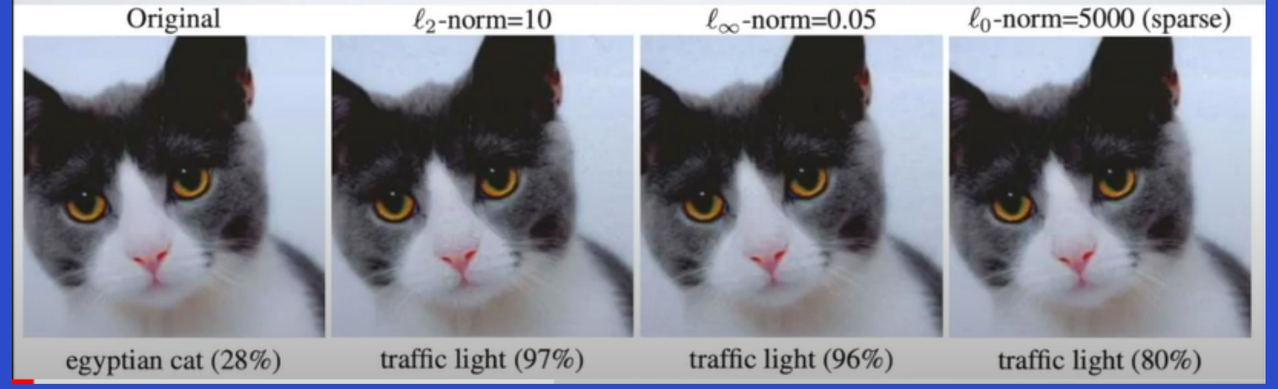
Self Adverserial Training (BoF)

Adverserial training is intended to make our model more robust and adapt to adverserial examples.

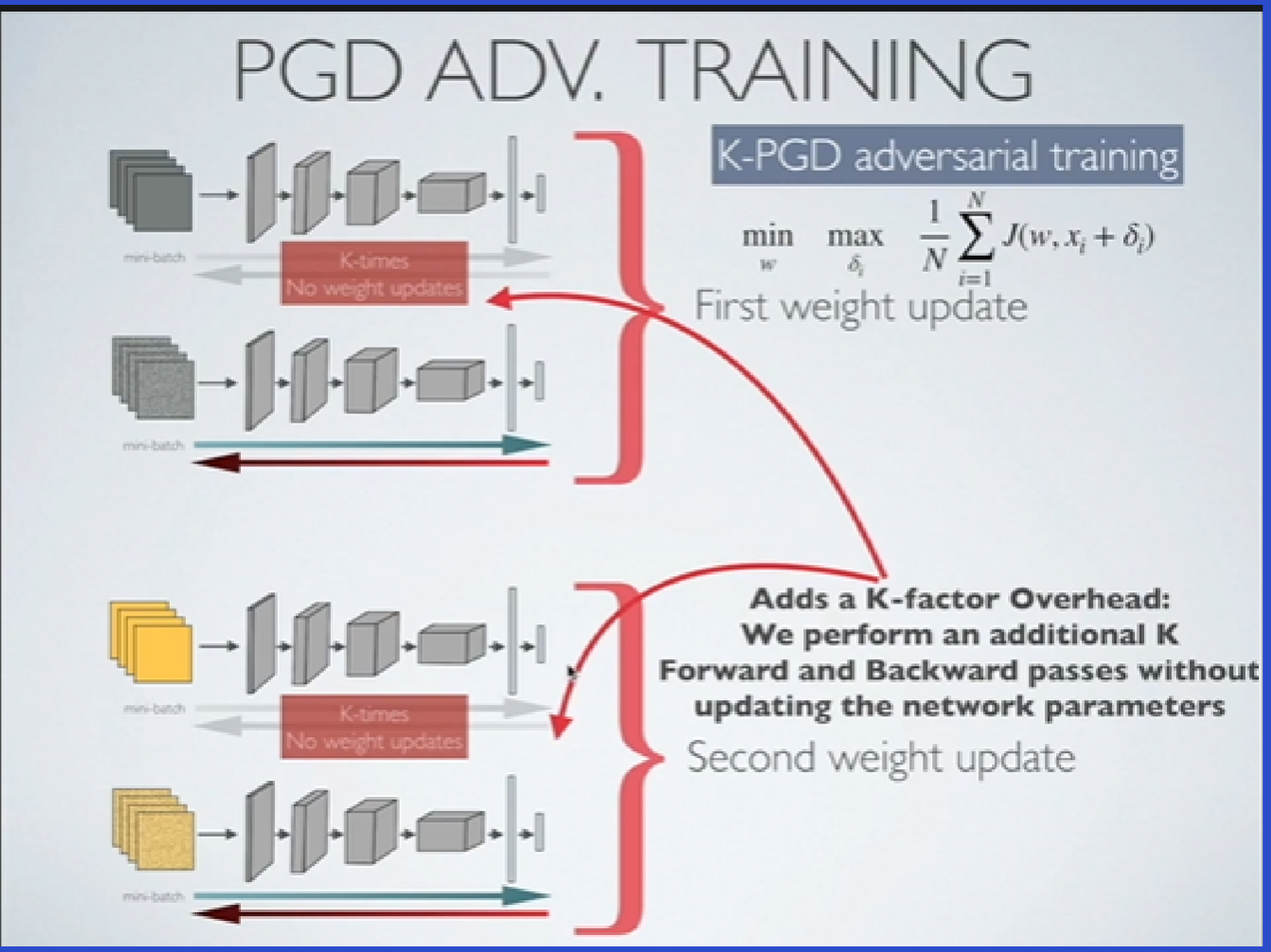
The training involves maximizing the delta terms (perturbations) and minimizing the the weights of the network. The perturbations are bounded.

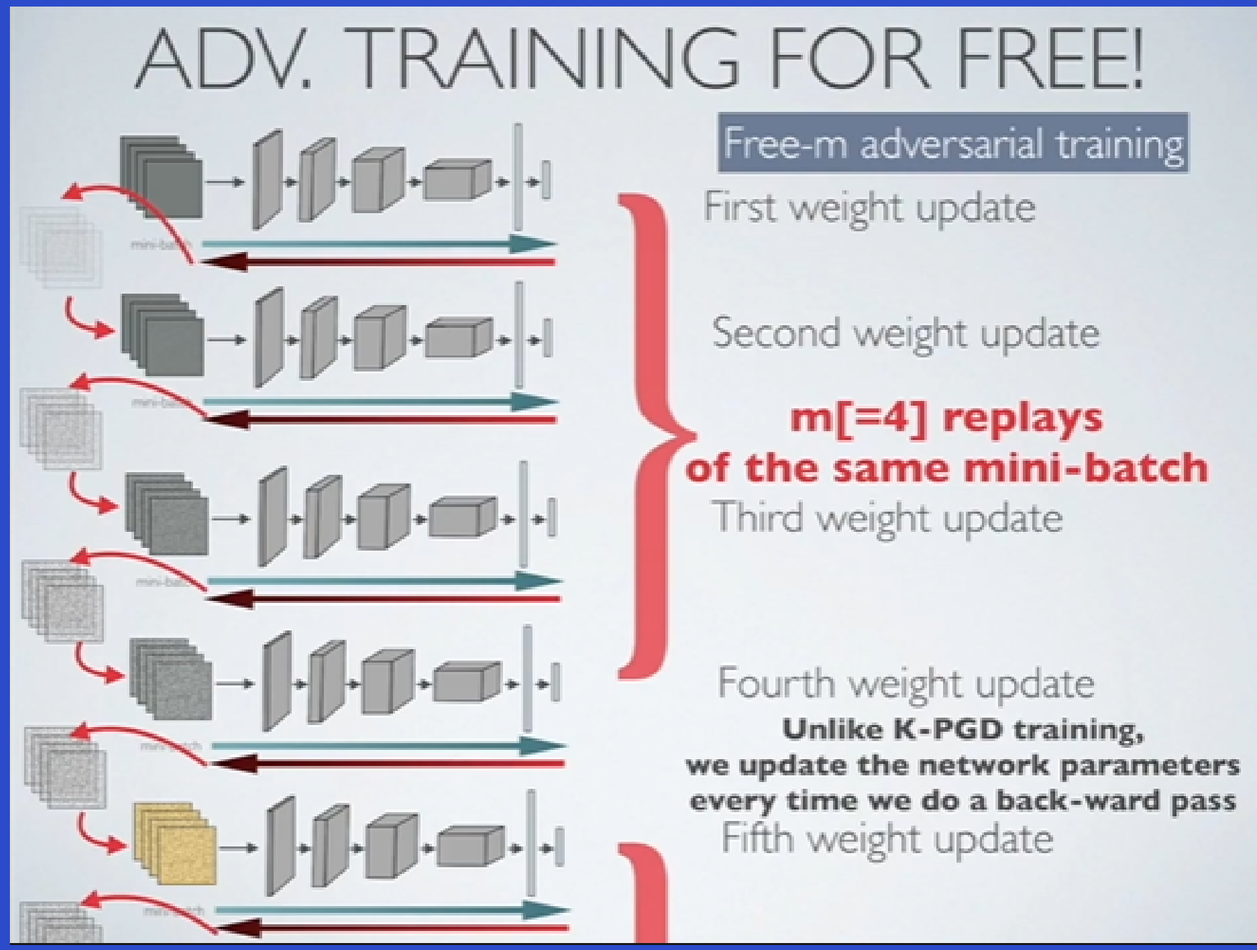
$$\min_{w} \max_{\delta_i} \frac{1}{N} \sum_{i=1}^{N} J(w, x_i + \delta_i)$$

$$\text{s.t.} \quad \|\delta_i\|_p \leq \epsilon \quad \forall i \in \{1..N\}$$



PGD vs ADV Training For FREE





Cross-Iteration Batch Normalization (BoF)

[Problem] A well-known issue of Batch Normalization is its significantly reduced effectiveness in the case of small mini-batch

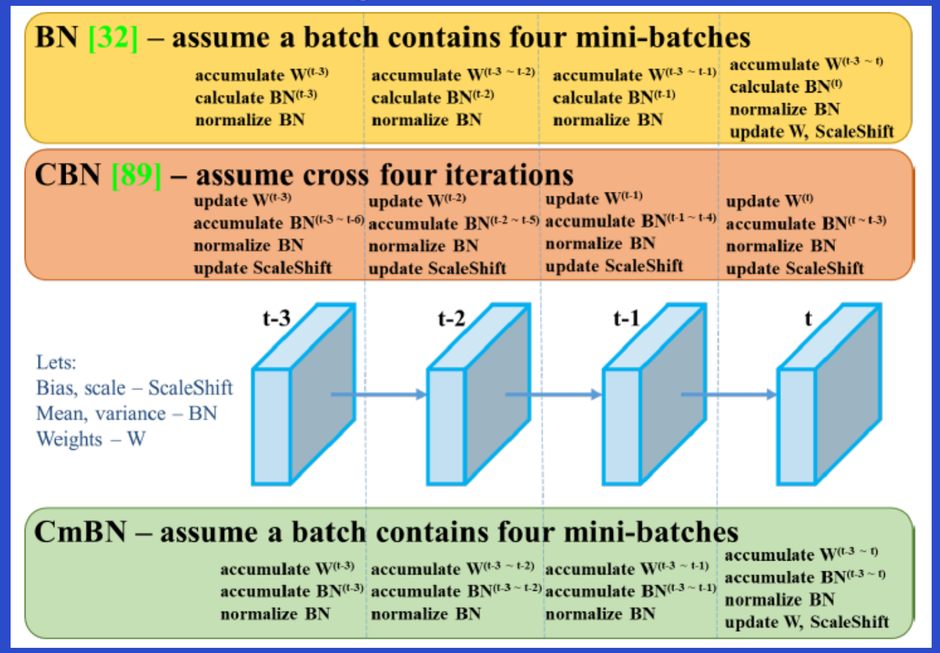
[Solution] Cross-iteration Batch normalization (CBM) joints multiple recent iterations

YOLOv4 uses Cross mini batch normalization (CmBN) that collects statistics only between mini-batches within a single batch.

Cross-Iteration Batch Normalization

	batch size per iter	#examples for statistics	Norm axis
IN	#bs/GPU * #GPU	1	(spatial)
LN	#bs/GPU * #GPU	1	(channel, spatial)
GN	#bs/GPU * #GPU	1	(channel group, spatial)
BN	#bs/GPU * #GPU	#bs/GPU	(batch, spatial)
syncBN	#bs/GPU * #GPU	#bs/GPU * #GPU	(batch, spatial, GPU)
CBN	#bs/GPU * #GPU	#bs/GPU * **temporal window**	(batch, spatial, **iteration**)

Table 1. Comparison of different feature normalization methods. #bs/GPU denotes batch size per GPU.



Batch Normalization

Consider a mini-batch $\mathcal{B}$ of size m. Since the normalization is applied to each activation independently, let us focus on a particular activation $x^{(k)}$ and omit k for clarity. We have m values of this activation in the mini-batch,

$$\mathcal{B} = \{x_{1...m}\}.$$

Let the normalized values be $\widehat{x}_{1...m}$, and their linear transformations be $y_{1...m}$. We refer to the transform

$$\mathrm{BN}_{\gamma,\beta} : x_{1...m} \rightarrow y_{1...m}$$

as the *Batch Normalizing Transform*. We present the BN Transform in Algorithm 1. In the algorithm, ϵ is a constant added to the mini-batch variance for numerical stability.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

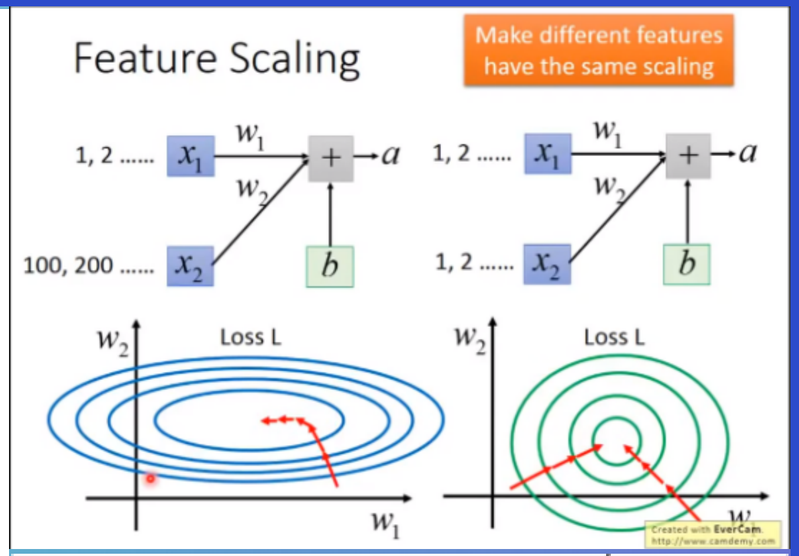
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

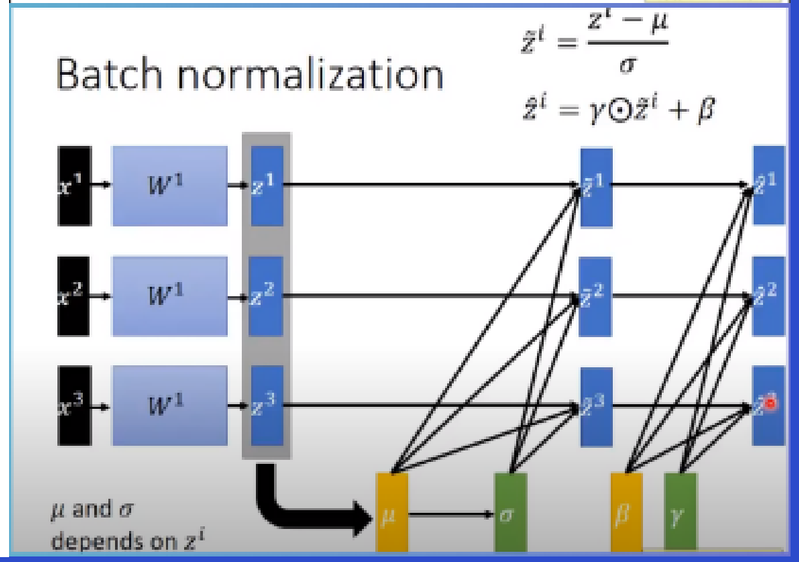
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to





- Before Batch Normalization, higher learning rate causes gradient explode or vanish and stuck in local minima.
- Batch normalization ONLY adds cost during training but nothing to do with Inference. Which means during runtime there is no overhead.
- It uses the mini-batch for updating the statistic of entire training set.

Cross Iteration Batch Normalization

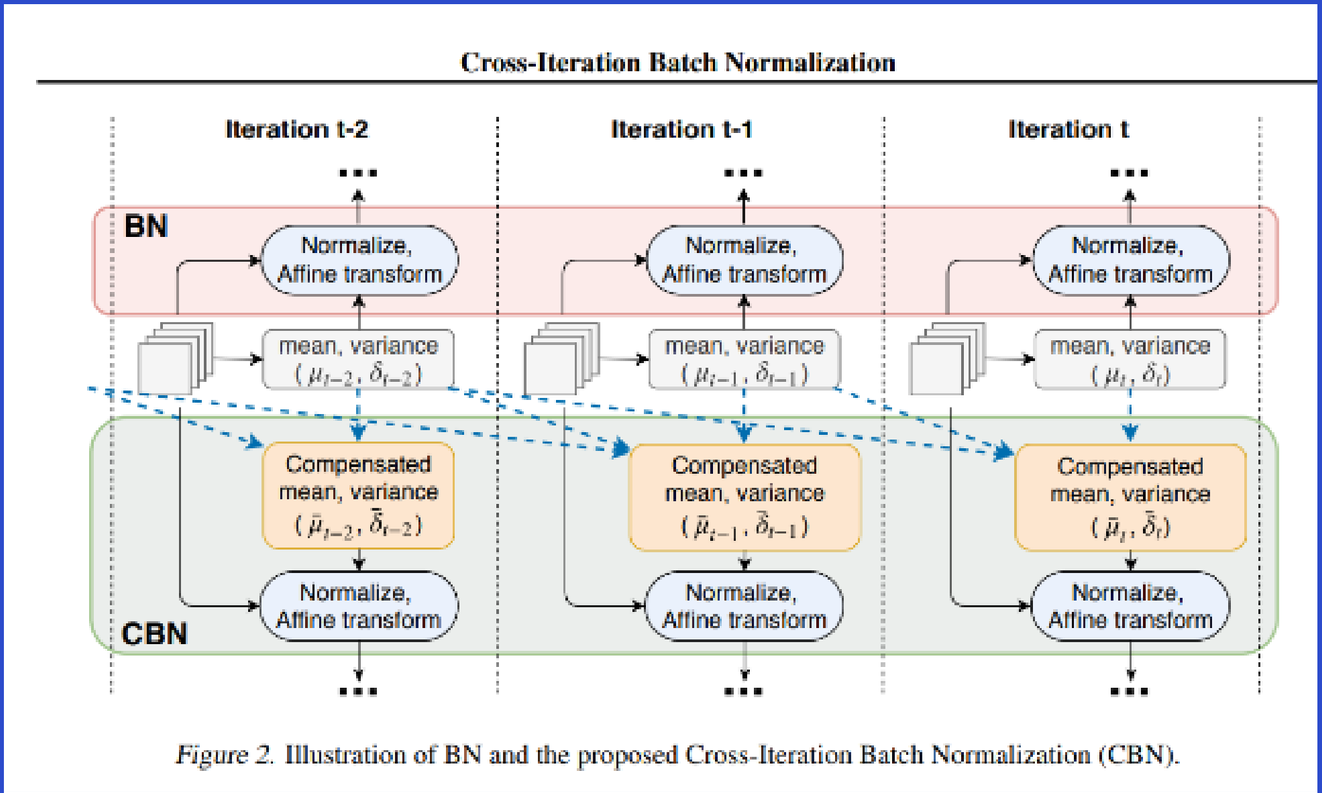


Figure 2. Illustration of BN and the proposed Cross-Iteration Batch Normalization (CBN).

- For normal batch normalization we compute the mean and variance for a batch.
- Using CBM we compute the mean and variance at different times and add the previous one to the next one.
- But since the weights are changing across different time, it's not easy to compute and add the means and biases.

Cross Iteration Batch Normalization

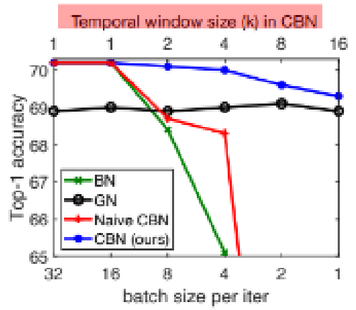


Figure 1. **Top-1 classification accuracy *vs.* batch sizes per iteration.** The base model is a ResNet-18 (He et al., 2016) trained on ImageNet (Russakovsky et al., 2015). The accuracy of BN (Ioffe & Szegedy, 2015) drops rapidly when the batch size is reduced. GN (Wu & He, 2018) exhibits stable performance but underperforms BN on adequate batch sizes. CBN compensates for the reduced batch size per GPU by exploiting approximated statistics from recent iterations (Temporal window size denotes how many recent iterationss are utilized for statistics computation). CBN shows relatively stable performance over different batch sizes. Naive CBN, which directly calculates statistics from recent iterations without compensation, is shown not to work well.

We observe that the network weights change smoothly between consecutive iterations, due to the nature of gradient-based training. This allows us to approximate $\mu_{t-\tau}(\theta_t)$ and $\nu_{t-\tau}(\theta_t)$ from the readily available $\mu_{t-\tau}(\theta_{t-\tau})$ and $\nu_{t-\tau}(\theta_{t-\tau})$ via a Taylor polynomial, i.e.,

$$\mu_{t-\tau}(\theta_t) = \mu_{t-\tau}(\theta_{t-\tau}) + \frac{\partial \mu_{t-\tau}(\theta_{t-\tau})}{\partial \theta_{t-\tau}}(\theta_t - \theta_{t-\tau}) + \mathbf{O}(||\theta_t - \theta_{t-\tau}||^2), \quad (5)$$

$$\nu_{t-\tau}(\theta_t) = \nu_{t-\tau}(\theta_{t-\tau}) + \frac{\partial \nu_{t-\tau}(\theta_{t-\tau})}{\partial \theta_{t-\tau}}(\theta_t - \theta_{t-\tau}) + \mathbf{O}(||\theta_t - \theta_{t-\tau}||^2), \quad (6)$$

where $\partial \mu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau}$ and $\partial \nu_{t-\tau}(\theta_{t-\tau})/\partial \theta_{t-\tau}$ are gradients of the statistics with respect to the network weights, and $\mathbf{O}(||\theta_t - \theta_{t-\tau}||^2)$ denotes higher-order terms of the Taylor polynomial, which can be omitted since the first-order term dominates when $(\theta_t - \theta_{t-\tau})$ is small.

batch size per GPU	32	16	8	4	2	1
BN	**70.2**	**70.2**	68.4	65.1	55.9	-
GN	68.9	69.0	68.9	69.0	69.1	68.9
BRN	70.1	68.5	68.2	67.9	60.3	-
CBN	**70.2**	**70.2**	**70.1**	**70.0**	**69.6**	**69.3**

Table 3. Top-1 accuracy of normalization methods with **different batch sizes** using ResNet-18 as the base model on ImageNet.

	Memory (GB)	FLOPs (M)	Training Speed (iter/s)	Inference Speed (iter/s)
BN	14.1	5155.1	1.3	6.2
GN	14.1	5274.2	1.2	3.7
CBN	15.1	5809.7	1.0	6.2

Table 6. Comparison of theoretical **memory, FLOPs** and practical **training and inference speed** between original BN, GN, and CBN in both training and inference on COCO.

- From the graph we can see that when the batch size drops the accurace for BN and Naive CBM drops too.
- During the training initially the weights biases change rapidly, but CBM is performed later in the training when the weights are changing smoothly.
- The computation cost for CBM is 11% and has 7% memory overhead during training and does not affect inference.

Mish Activation (BoS)

The experiments show that Mish tends to work better than both ReLU and Swish along with other standard activation functions in many deep networks across challenging datasets

$$f(x) = x \cdot tanh(\varsigma(x)) \quad (1)$$

where, $\varsigma(x) = \ln(1 + e^x)$ is the softplus activation [10] function. The graph of Mish is shown in Figure 1.

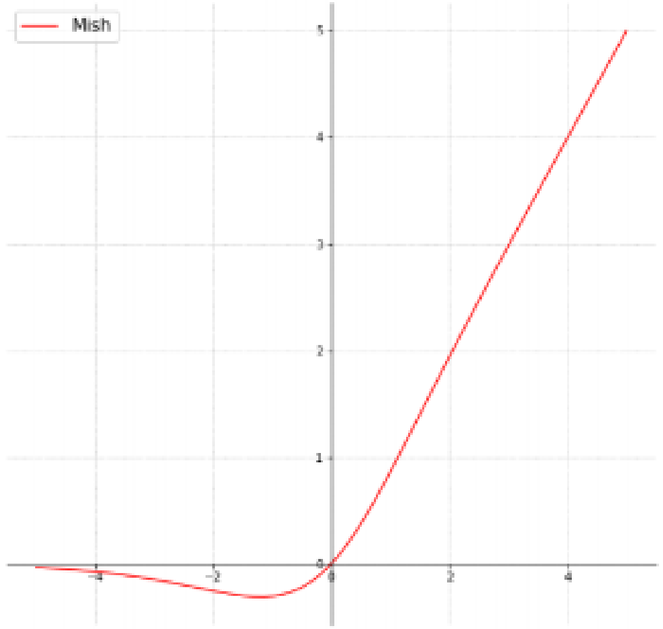


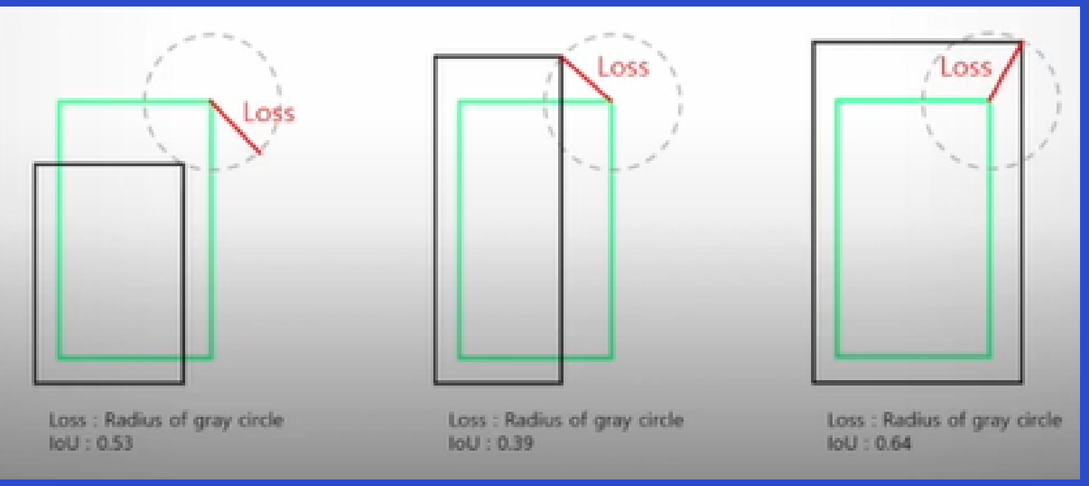
Figure 1. Mish Activation Function

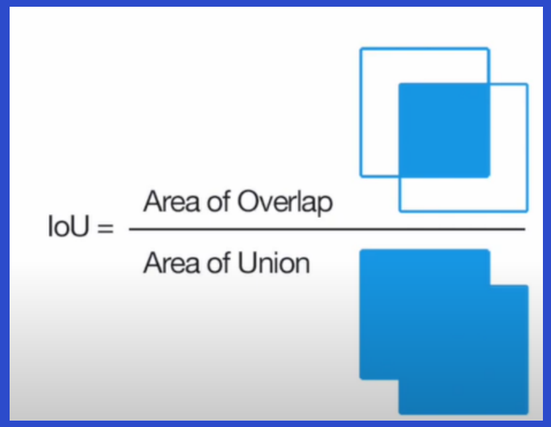
IoU Loss (BoF)

IoU is scale invariant which means the intersection over union is the same for large and small bounding boxes.

But its draw back is that there is no correlation between minimizing L1,L2 and huber loss and maximizing IoU.

It was perfect for overlapping boxes but suffered vanishing gradient problem due to non-overlapping boxes since the ioU is zero irrespective of the location of the two boxes.





IoU Loss = 1- IoU

$$L_{IoU} = 1 - \frac{|B \bigcap Bgt|}{|B \bigcup Bgt|}$$

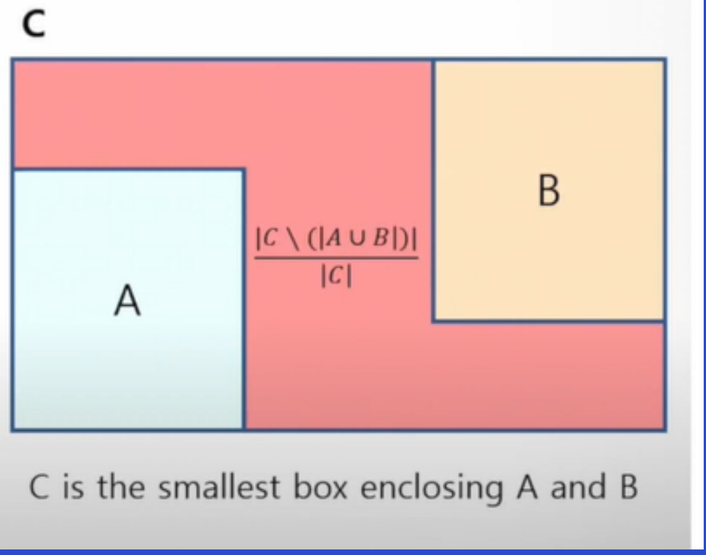
GIoU Loss (BoF)

C is the smallest box covering B(predicted bounding box) and B_gt(ground truth bounding box).

Maximizing GIoU implies

1. Maximizing IoU when boxes are overlapped
2. Minimizing distance between boxes when non-overlapping cases

$$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$$



GIoU Loss = 1- GIoU

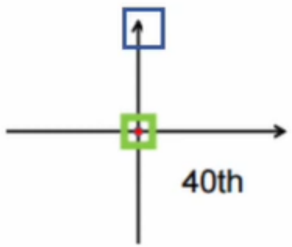
$$\mathcal{L}_{GIoU} = 1 - IoU + \frac{|C - B \cup B^{gt}|}{|C|},$$

GIoU Loss (BoF)

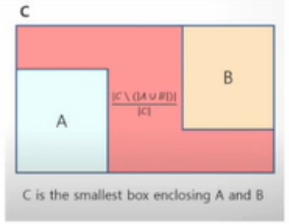
GIoU: A tedious remedy to non overlapping bounding boxes. It suffers from slow convergenceand inaccurate regression.

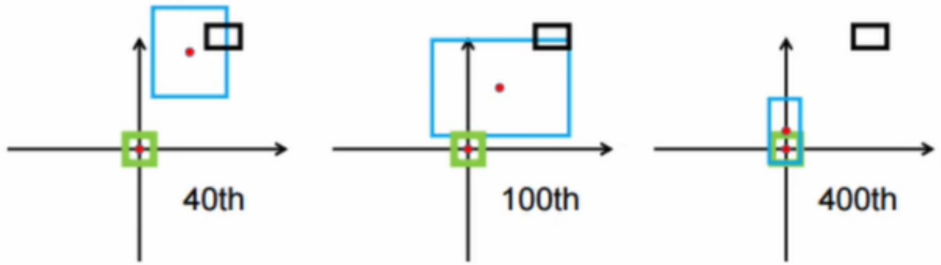
Due to the introduction of the penalty term, GIoU loss keeps expanding the size of the predicted box until it overlaps with the target box

GIoU loss has slow convergence especially for the boxes at horizontal and vertical orientations.

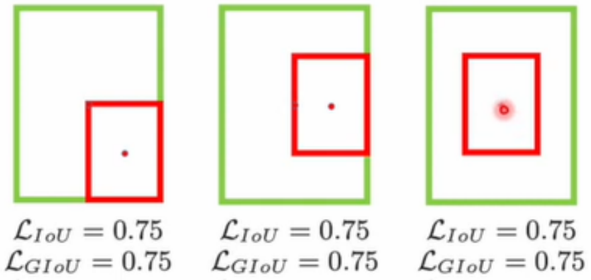


$$\mathcal{L}_{GIoU} = 1 - IoU + \frac{|C - B \cup B^{gt}|}{|C|},$$





Slow Convergence



Inaccurate Regression

Fig. 3: Horizontal stretch: Source:[Link]

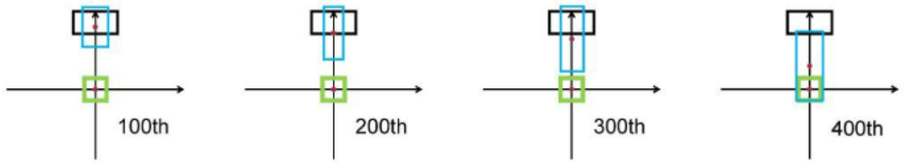
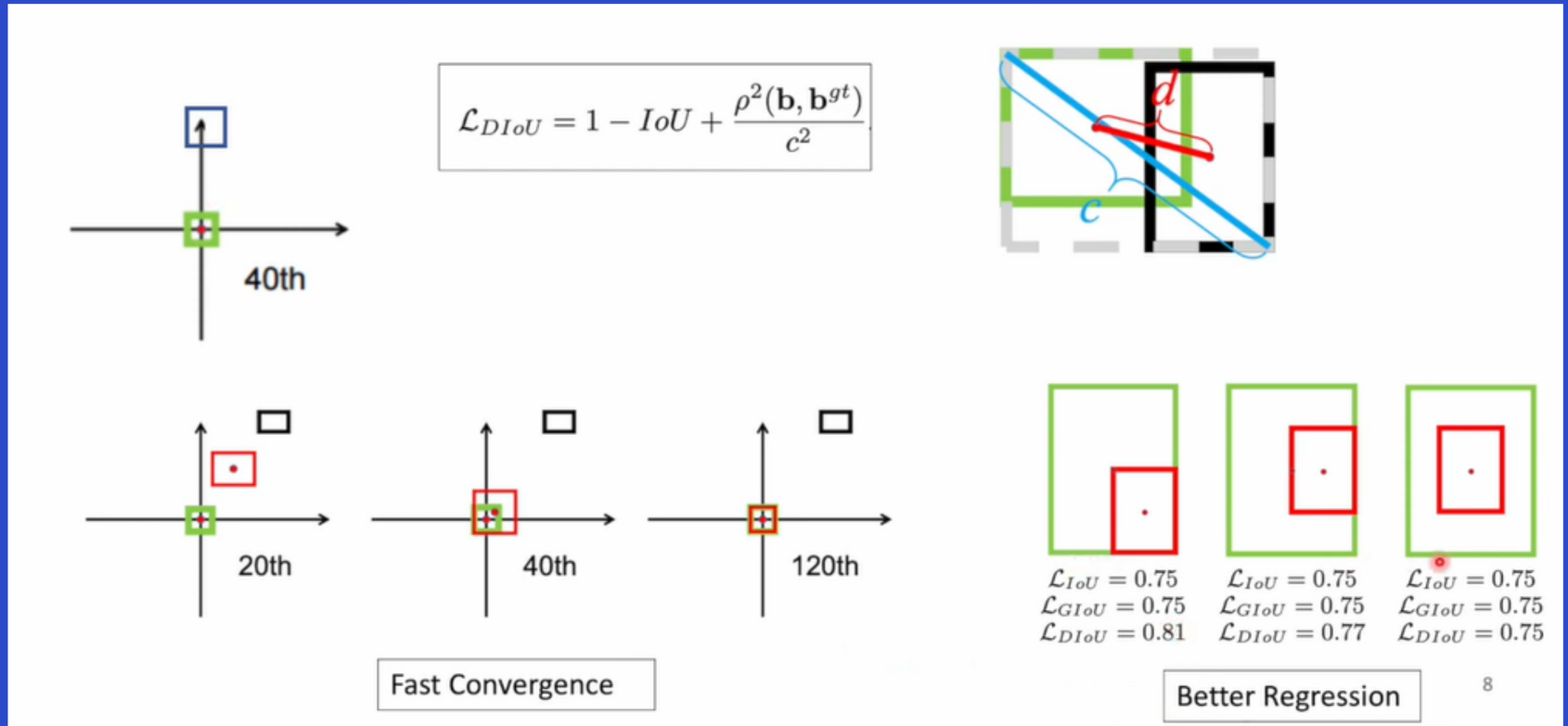


Fig. 4: Vertical stretch: Source:[Link]

DIoU Loss (BoF)

DIoU uses **Euclidean distance between center coordinates of the targeted bounding box and predicted bounding box** normalized by **'c' the diagonal length of the smallest enclosing box covering the two boxes**

Cons: It still misses out the consistency of aspect ratios for bounding boxes an important geometric factor.

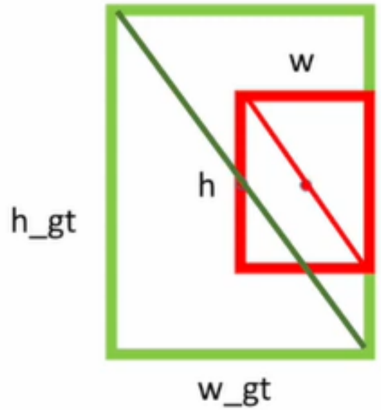


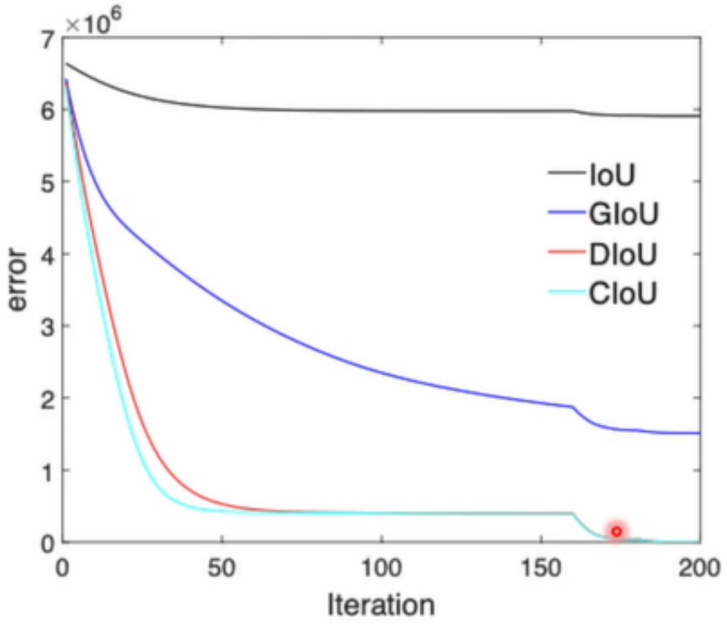
CIoU Loss: Master of All (BoF)

Complete IoU regression loss incorporates all geometric factors: overlapping area, distance, and aspect ratio.

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

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





NMS (BoS)

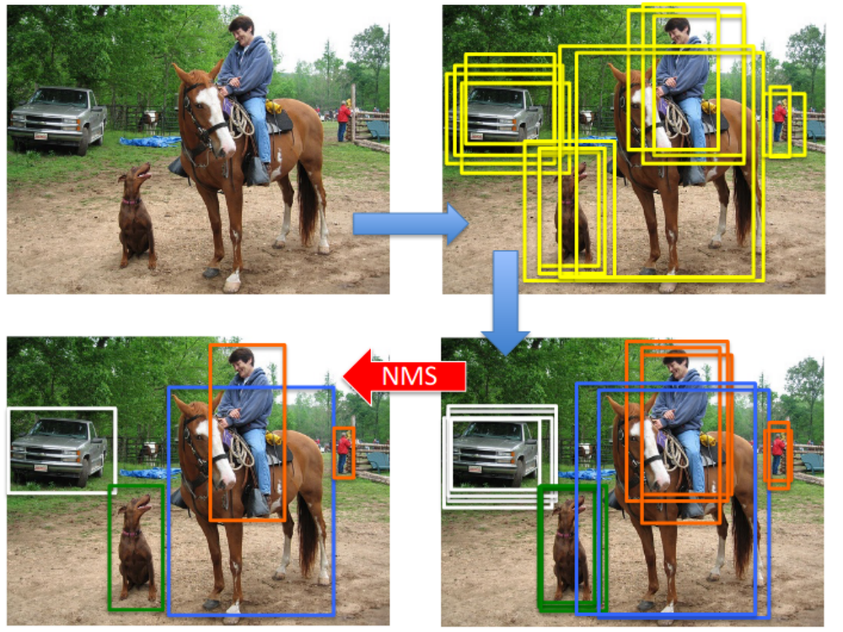
Greedy NMS is the most common and widely used post processing step.

Let B represent the list of predicted bounding boxes.

1. Select the bounding box with maximum confidence score, M and store in list D
2. Compute IoU between M and remaining boxes in B
3. Delete boxes with **IoU(M,B) > threshold**, Nt
4. Repeat process 1–3 until B is empty. D contains the retained bounding boxes

$$s_i = \begin{cases} s_i, & \text{iou}(\mathcal{M}, b_i) < N_t \\ 0, & \text{iou}(\mathcal{M}, b_i) \geq N_t \end{cases},$$

Greedy NMS



Soft NMS (BoS)

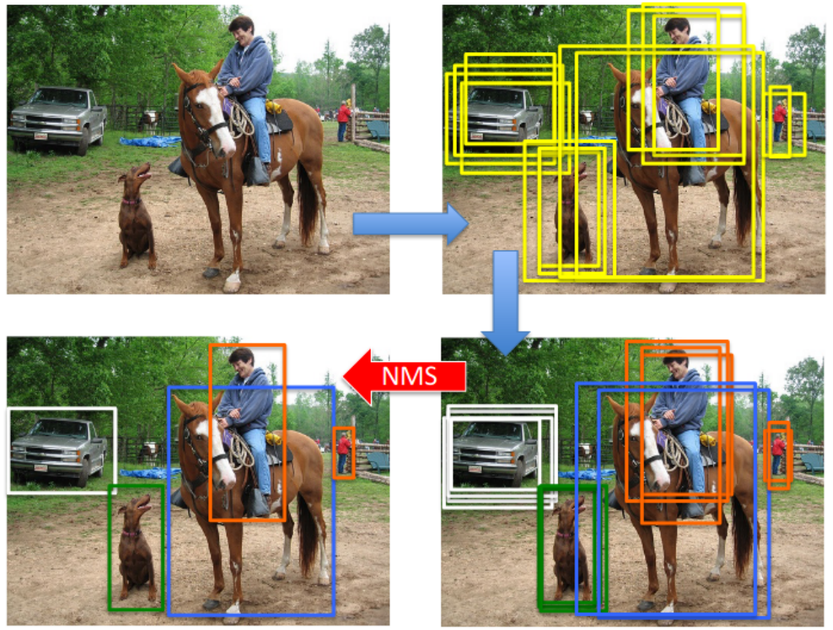
Greedy NMS completely removes boxes which do not satisfy the IoU threshold.

This may not be suitable when there are overlapping/crowded regions in the image.

There might be a scenario wherein a box overlapping with the most confident box, contains an object of interest but eventually gets removed due to the threshold. This may result in the object not being detected at all.

$$s_i = \begin{cases} s_i, & \text{iou}(\mathcal{M}, b_i) < N_t \\ s_i(1 - \text{iou}(\mathcal{M}, b_i)), & \text{iou}(\mathcal{M}, b_i) \geq N_t \end{cases},$$

Soft-NMS criteria



Soft NMS (BoS)

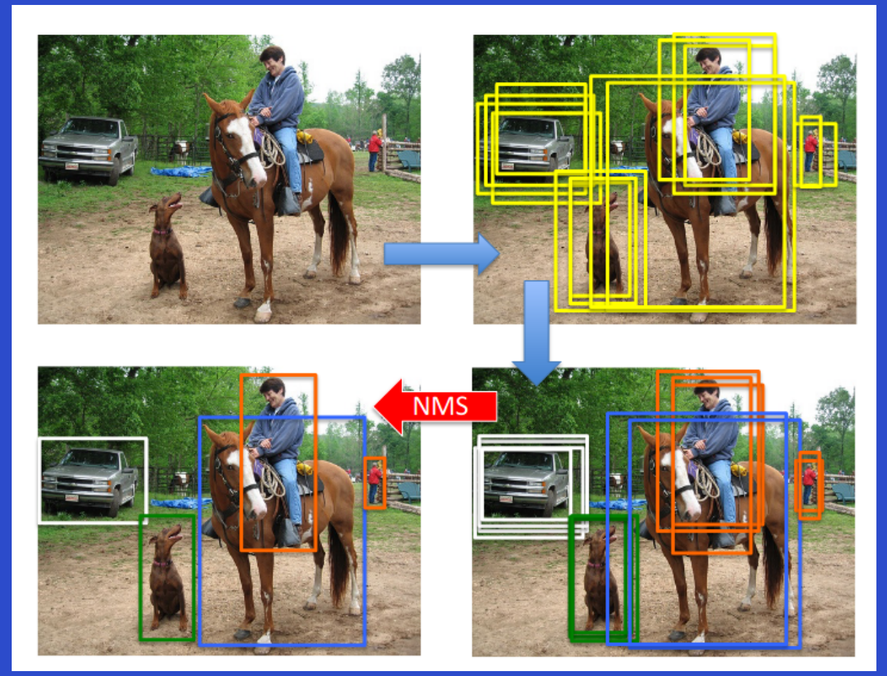
Soft-NMS proposes a very simple change to the Greedy NMS algorithm as a solution to the above problem.

$$s_i = \begin{cases} s_i, & \text{iou}(\mathcal{M}, b_i) < N_t \\ s_i(1 - \text{iou}(\mathcal{M}, b_i)), & \text{iou}(\mathcal{M}, b_i) \geq N_t \end{cases},$$

Soft-NMS criteria

It suggests that rather than removing the boxes which do not satisfy the threshold, the confidence scores of the boxes should be decayed by an amount proportional to the IoU.

In this way, the boxes having higher IoU are penalized more than the boxes which are farther away with lower IoU similar to Greedy NMS.



DIoU NMS (BoS)

DIoU NMS uses central point distance between two boxes along with the IoU in the threshold criterion.

Central point distance, RDIoU, between two boxes has been shown below where c is the diagonal length of the smallest enclosing box covering the two boxes and ρ(b,b_gt) is the euclidean distance between centers of the two boxes.

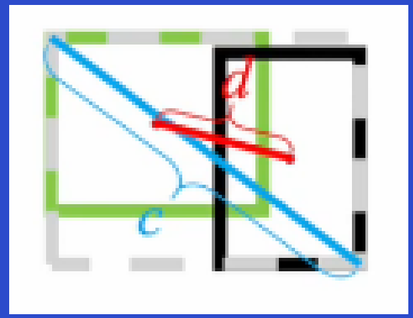
DIoU-NMS has been shown to work better in case of occlusions and ensures tighter boxes.

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

Central point distance equation

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

DIoU NMS criteria



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%**

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%**

The proposed BoF-backbone (Bag of Freebies) for classifier training includes the following:

CutMix , Mosaic data augmentation and **Class label smoothing.**

Influence of BoF on Detector Training (MS COCO)

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%**

Finally, S+M+IT+GA+OA+GIoU/CIoU obtains the highest accuracy.

Influence of BoS on Detector Training (MS COCO)

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%**

For BoS, the detector gets best performance when using SPP, PAN, and SAM.

Influence of Different Backbones and Pretrained Weightings on Detector Training

- Although classification accuracy of CSPResNeXt-50 models trained with different features is higher compared to CSPDarknet53 models, the CSPDarknet53 model shows higher accuracy in terms of object detection.

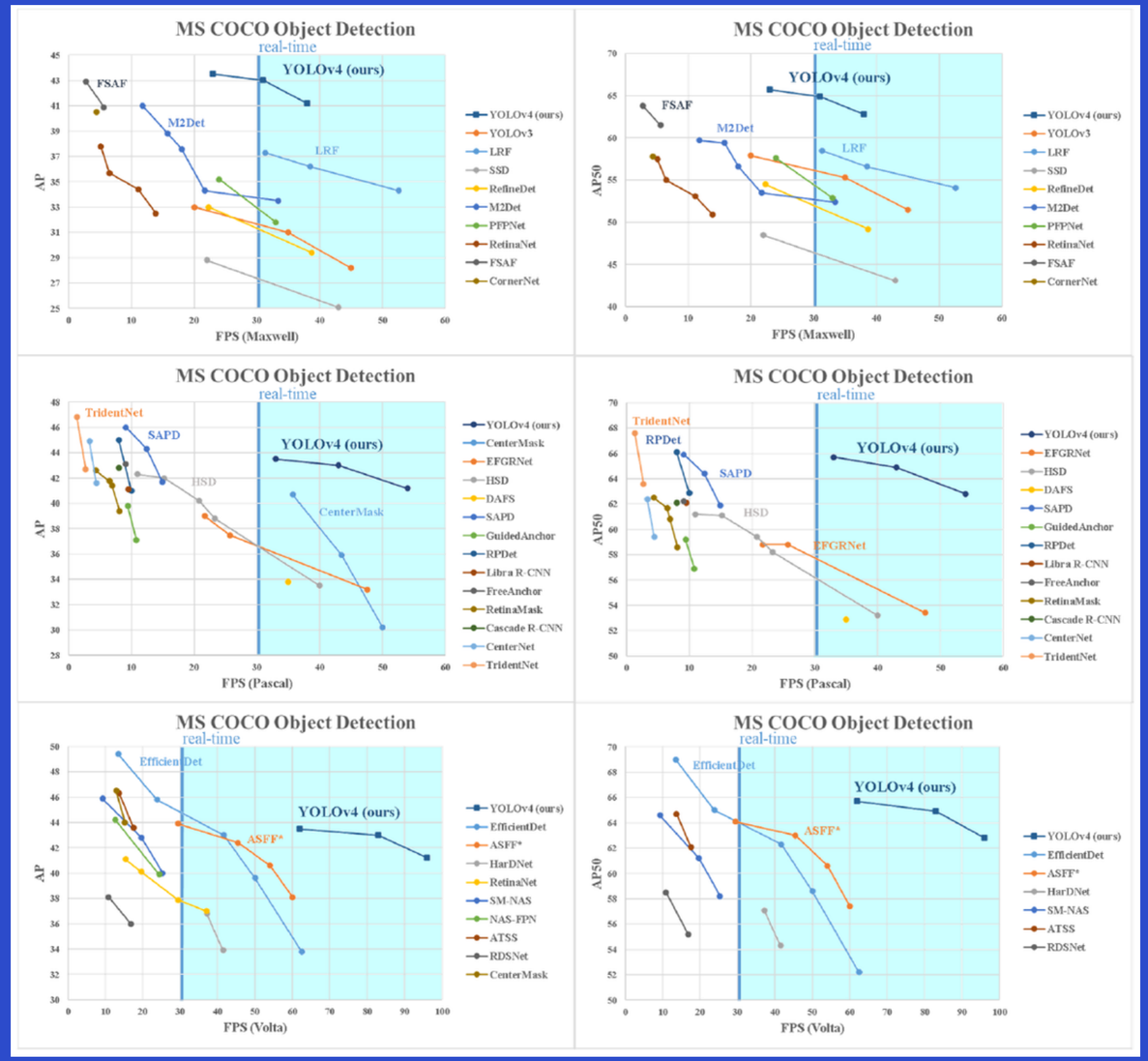
- Using BoF and Mish for the CSPResNeXt50 classifier training increases its classification accuracy, but further application of these pre-trained weightings for detector training reduces the detector accuracy.

- Using BoF and Mish for the CSPDarknet53 classifier training increases the accuracy of both the classifier and the detector which uses this classifier pre-trained weightings.

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

Comparison of the speed and accuracy of different object detectors Using one GPU of either Maxwell/Pascal/Volta Type

- As seen above, with different types of GPU, **YOLOv4 are located on the Pareto optimality curve** and **are superior to the fastest and most accurate detectors** in terms of both speed and accuracy.



Comparison of the speed and accuracy of different object detectors on the MS COCO dataset

- This table lists the frame rate comparison results of using Maxwell GPU. (There are also tables for Pascal and Volta GPU. Please feel free to read the paper.)

- Real-time detectors with FPS 30 or higher are highlighted in blue.

- YOLOv4 outperforms EfficientDet, ASFF, NAS-FPN, CenterNet, CornerNet, etc.

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%