



Object Detection - Yolo

Artificial Intelligence

Creating the Future

Dong-A University

Division of **C**omputer **E**ngineering &
Artificial **I**ntelligence

[Joseph Redmon, Ali Farhadi, “YOLO9000: Better, Faster, Stronger,” arXiv:1612.08242, 2016](#)

[발췌 자료]

[PR-023] YOLO9000 Better, Faster, Stronger - JinWon Lee (2017)

<https://www.slideshare.net/JinwonLee9/pr12-yolo9000>

<https://www.youtube.com/watch?v=6fdclSGgeio>

YOLO v2 – Object Detection

<https://medium.com/@arashilen/note-for-yolo-v2-yolo9000-d00285962cb4>

<https://medium.com/analytics-vidhya/deep-dive-on-yolov2-and-yolo9000-2eba212dcf8a>

<https://www.geeksforgeeks.org/yolo-v2-object-detection/>

<https://taeu.github.io/paper/deeplearning-paper-yolov2/>

<https://yeomko.tistory.com/47>

Abstract

- Improving YOLO to YOLOv2.
- YOLOv2 outperforms state-of-the-art methods like Faster R-CNN with ResNet and SSD. : 76.8 mAP at 67 FPS, 78.6 mAP at 40 FPS on VOC2007.
- Proposing a method to **jointly train on the COCO object detection dataset and ImageNet classification dataset.**
- Jointly training allows **YOLO9000 to predict detections for object classes that don't have labelled detection data.**
- YOLO9000 trained by jointly train can detect over **9,000 object categories.**

Better

- Batch normalization
- Hight resolution classifier
- Convolution with anchor boxes
- Dimension clusters
- Direct location prediction
- Fine-grained features
- Multi-scale training

Faster

- Darkent-19
- Training for classification
- Training for detection

Stronger

- Hierarchical classification
- Dataset combination with Word-tree
- Joint classification and detection

YOLOv2

YOLO9000

Introduction

- Detection framework have become increasingly fast and accurate
→ However, **most detection methods are still constrained to a small set of objects**
- **Joint training algorithm** is also proposed that allows to **train object detectors on both detection and classification data.**
- First, improving upon the base *YOLOv1* detection system to produce **YOLOv2 (SOTA real-time detector)**
- Then, **using dataset combination method and joint training algorithm** to **train model on more than 9000 classes!**
- *YOLOv1*'s shortcomings relative to SOTA detection systems
 - *YOLOv1* makes **a significant number of localization errors.**
 - *YOLOv1* has relatively **low recall** compared to region proposed based method
- Focus mainly on **improving recall and localization while maintain classification accuracy.**
- It is still very important to detect **FAST!**

Better

Batch normalization

- Adding **BN** on **all of conv layers** in YOLOv1 : **2%** improvement in mAP
- Removing dropout** without overfitting.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

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

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

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

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

batch normalization benefits the model in the following ways:

- ✓ Allows to use *higher learning rate* to speed up the learning process as it ensures that no activation is too high or low
- ✓ It *reduces overfitting* by adding some regularization by adding some noise to the activations

[발췌 자료] [PR-023] YOLO9000 Better, Faster, Stronger - JinWon Lee

High resolution classifier

- YOLOv1 trains the classifier network @224x224 and increase resolution to 448 for detection → The network has to simultaneously switch to learning object detection and adjust to the new input resolution.
- YOLOv2, fine tune the classifier network @448x448 resolution** for 10 epochs on Imagenet and then fine tune the resulting network on detection – 고해상도로 학습된 앞단의 classification network에 의하여 almost **4%** improvement in mAP.

Fine-tune 448x448 Classifier: +3.5% mAP

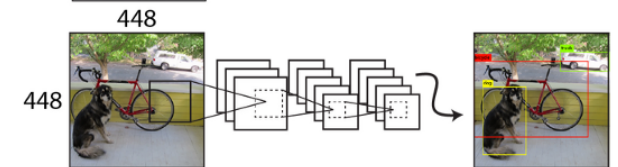
Train on ImageNet



Resize, fine-tune on ImageNet



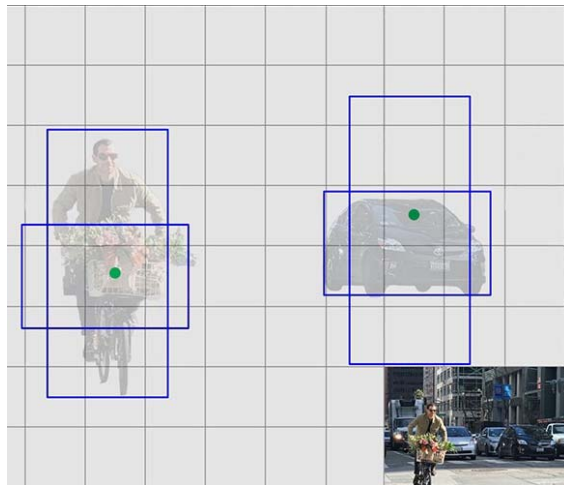
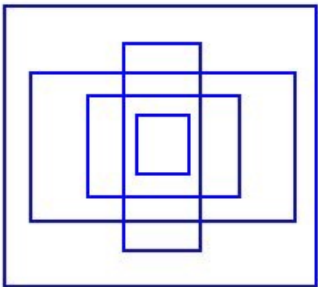
Fine-tune on detection



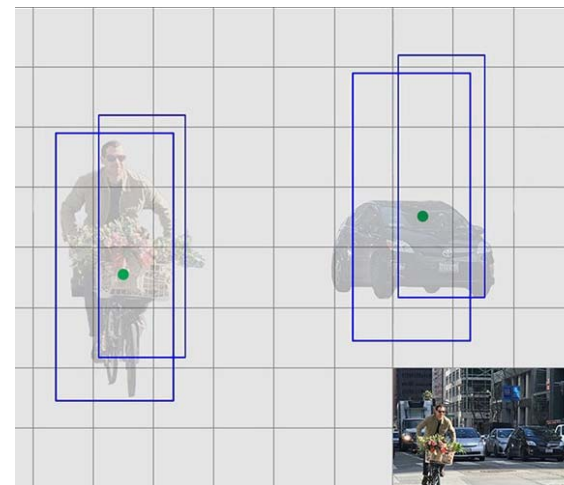
Better

Convolutional With **Anchor Boxes**

- YOLOv1 predicts the bounding box coordinates directly using fully connected (FC) layers at the end of the model, on top of all convolutional layers (feature extractors).
- *Faster R-CNN* has a better technique — using *hand-picked priors (anchor boxes)*, it *predicts offsets and confidences for these boxes at every location in a feature map*. This simplifies the problem and makes it easier for the network to learn.
- We create 5 anchor boxes.



Predict the offsets constrained to the anchor boxes. **Maintaining the diversity of predictions.**



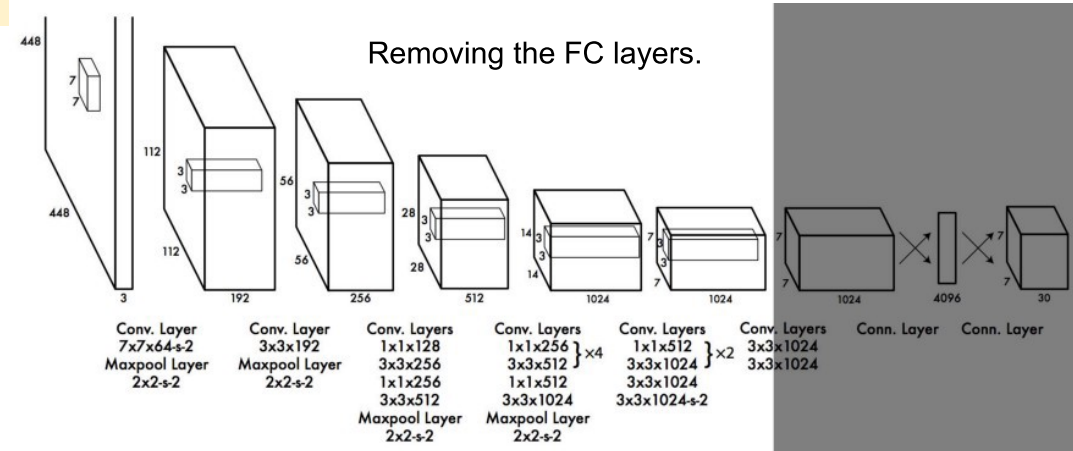
Predict arbitrary bounding boxes for each object (like YOLO does). **Unstable training**

Better

Convolutional With Anchor Boxes

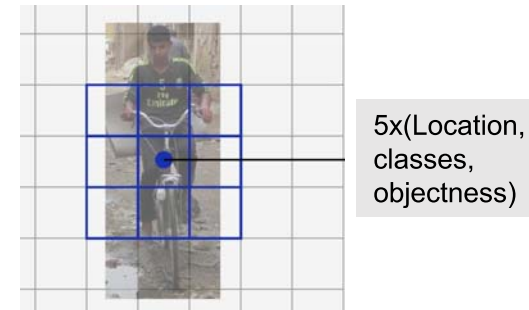
- 1) **YOLOv2 removes the fc layers (convolutional connect로 변경) and use anchor boxes to predict bounding boxes.**
- 2) Shrinking the network to operate on 416x416 input images instead of 448x448
 - ✓ YOLO's CNN downsample the image by a factor of 32.
 - ✓ Objects, especially large objects, tend to occupy the center of the image, so it is better to have one cell in the middle than to have four cells. → odd number of grid cells are required (416x416 → 13x13), Output feature map can be 13x13 since 13 is odd. **(13xF) x (13xF) = 416, F(Factor)=31**
- 3) **Predicting class and objectness for every anchor box (Next page)**
 - Using anchor boxes makes a small decrease in accuracy.
 - ✓ 69.5mAP to 69.2mAP but 81% recall to 88% recall → high recall means there is more room to improve

[발췌 자료] [PR-023] YOLO9000 Better, Faster, Stronger - JinWon Lee



YOLOv1 vs. YOLOv2 on BB attribute count:

- YOLOv1: $B \times (4 + 1) + C \quad B \times 5 + 20$
- YOLOv2: $B \times (4 + 1 + C) \quad B \times (5 + 20)$



YOLOv1 vs. YOLOv2 on performance:

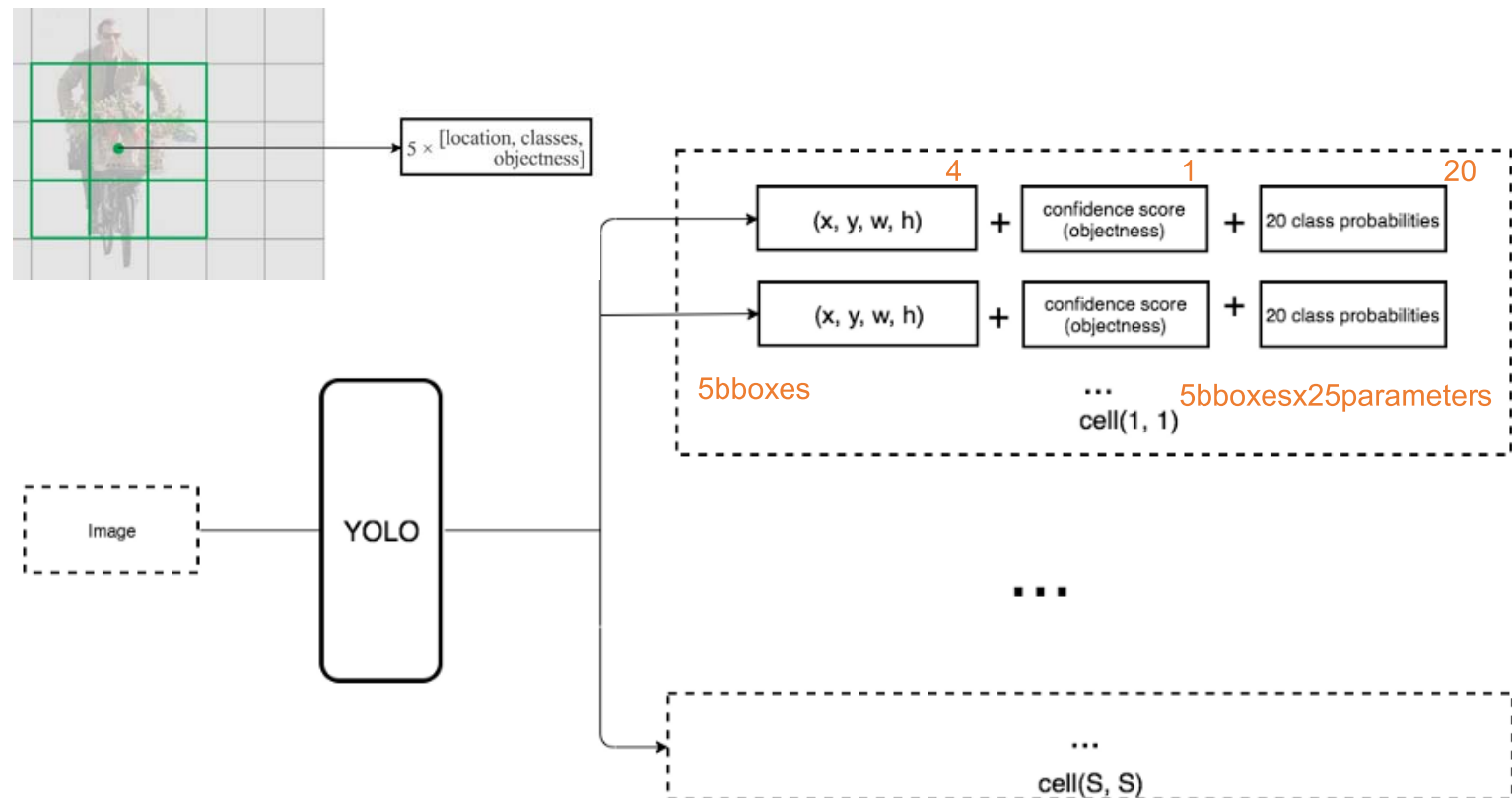
- YOLOv1: predicts 98 BBs per image ($7 \times 7 \times 2 = 98$)
mAP = 69.5, recall = 81%
- YOLOv2 with 9 hand-picked priors / anchor boxes: predicts > 1,000 BBs per image ($13 \times 13 \times 9 = 1,521$)
mAP = 69.2, recall = 88% (accuracy decreased, recall increased)

Better

Convolutional With **Anchor Boxes**

3. Decouple the class prediction mechanism from spatial locations and shift it to bounding box level.

- For **every bounding box**, we now have **25 parameters** — **4 for the bounding box**, **1 box confidence score** (called *objectness* in the paper), and **20 class probabilities** (for the 20 classes in the VOC dataset).
- We have **5 bounding boxes per cell**, thus **each cell** will have **25×5** , i.e. **125 parameters**.
- Similar to YOLO, the *objectness* prediction mechanism predicts the IOU (intersection over union) of the ground truth and the proposed box



Better

Dimension Cluster

- First issue with anchor boxes In YOLOv1, : the box dimension is *hand picked*, and then adjusted by NN.
- Instead of choosing priors by hand, running **k-means clustering on the BBs in training set** to automatically find good priors.
- To prevent larger boxes from making more error, they use a *different distance metric* (not Euclidean distance). **Distance metric for k-means** (sort of ratio, independent of the BB size):

$$d(\text{box}, \text{centroid}) = 1 - \text{IoU}(\text{box}, \text{centroid})$$
- **k(Clusters)=5** is chosen as a good tradeoff between complexity and high recall

- 9개의 앵커박스를 두었을때 그만큼 다양한 anchor box로 잘 예측할 수 있지만, 5개를 두었을 때랑 크게 차이가 나지 않는다고 판단하고(computation은 2배가 되는데에 비해) YOLO v2에서는 5개의 anchor box를 사용하기로 결정.

[발췌 자료] [PR-023] YOLO9000 Better, Faster, Stronger - JinWon Lee

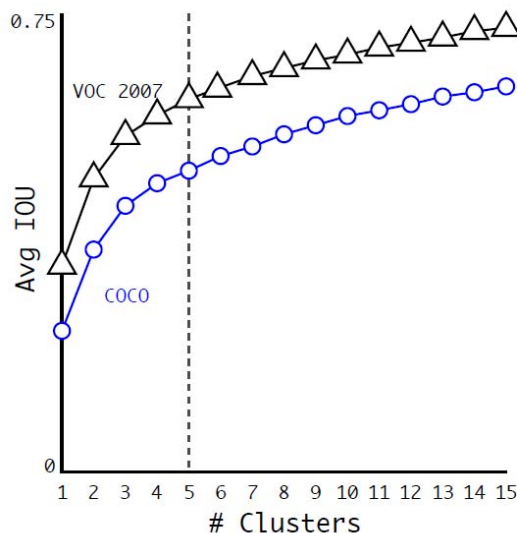


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that k = 5 gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

Box Generation	#	Avg IOU
Cluster SSE	5	58.7
Cluster IOU	5	61.0
Anchor Boxes [15]	9	60.9
Cluster IOU	9	67.2

Table 1: Average IOU of boxes to closest priors on VOC 2007. The average IOU of objects on VOC 2007 to their closest, unmodified prior using different generation methods. Clustering gives much better results than using hand-picked priors.

Better

Direct Location Prediction

- The second issue with anchor boxes in YOLOv1, *model instability*
- Most of the instability comes from predicting (x, y) locations of the box. RPNs(region proposal network) predict values (t_x, t_y) and (x, y) center coordinates; $x = (t_x \times w_a) - x_a$, $y = (t_y \times h_a) - y_a$.
- the **anchor box can end up at any point in the image**, regardless of what location predicated the box. → With random initialization, the model **takes a long time to stabilize to predicting sensible offsets**.
- anchor box를 예측하되 기존의 Predicted box의 중심좌표들이 초기 단계에 잘못 설정된다면 학습이 잘 안될 수 있다(예측해야 되는 곳보다 멀어질 수 있음). (SSD의 경우는 I, predict된 값이 default box로 normalize(?)된 값이라 상대적으로 잘 될 것). YOLO v2는 이런 값들이 범위를 크게 벗어나지 않고 적절히 학습하기 위해서 다음과 같은 설정을 한다.
- Instead of predicting offsets, YOLOv2 predicts **locations relative to the location of the grid cells**.
- **Predict 5 bounding boxes at each cell in the output feature map and 5 coordinates for each bounding box** $(t_x, t_y, t_w, t_h, t_o)$
- → almost **5% improvement** over the version with anchor boxes.

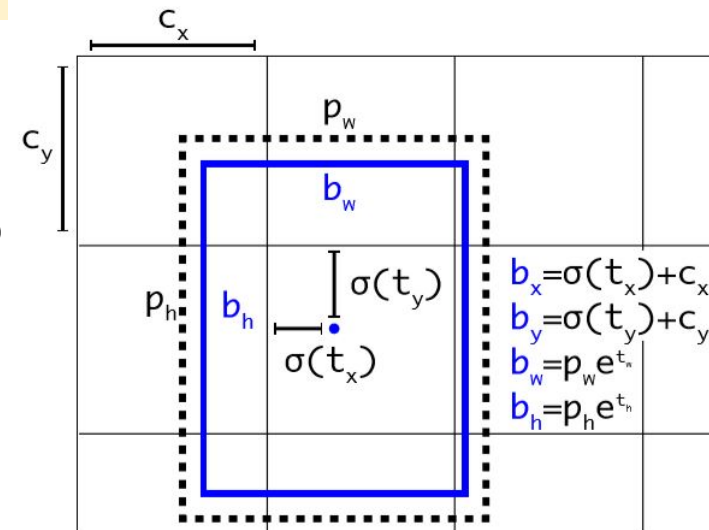


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned}$$

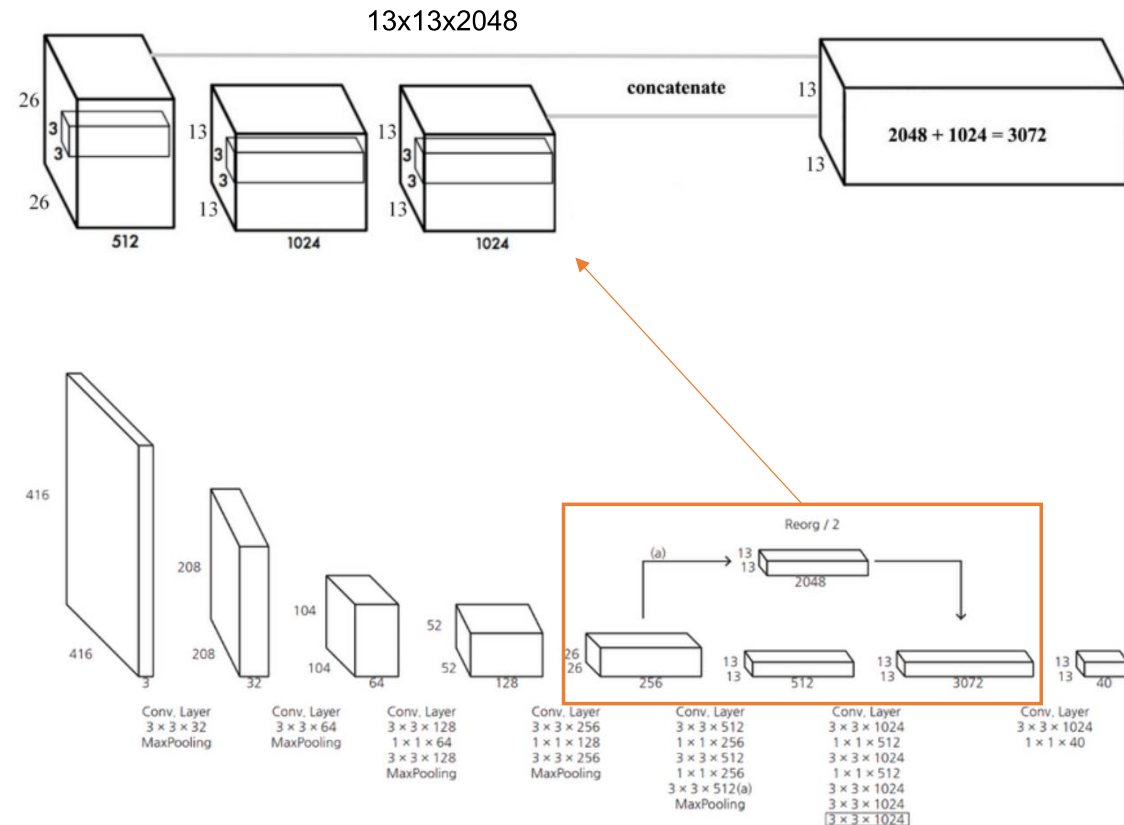
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

- c_x, c_y : Grid cells의 좌상단 끝 offset
- p_w, p_h : Prior(우선순위 앵커박스)의 width, height
- t_x, t_y, t_w, t_h : 우리가 예측해야 할 값들
- b_x, b_y, b_w, b_h : 각 값들을 조정하여 실제 GT와 IOU를 계산할 최종 bounding box의 offset 값들
- 중심좌표는 sigmoid를 통해서 0으로 initialize되면 중심에 가고, 너비와 높이 역시 0으로 초기화했을때 prior, anchor box의 값들에서 시작될 수 있게 한 조치이다. 이렇게 하면 안정적으로 학습이 잘 될 것.

Fine-Grained Features

- YOLOv2 predicts detections on a 13x13 feature maps.
 - ✓ While it is sufficient for large objects, benefit from finer grained features for localizing small objects.
 - Simply adding a passthrough layer that brings features from an earlier layer at 26x26 resolution to 13x13 output feature map.
 - ✓ *The passthrough layer concatenates the higher resolution features with lower one by stacking adjacent features into different channels instead of spatial locations, similar to the identity mappings in ResNet.*
 - It turns 26x26x512 feature map into a 13x13x2048 feature map which can be concatenated with original features.
 - Detector runs on top of this expanded features
 - 1% performance increase
- 13x13은 큰 이미지를 검출하기엔 충분한 feature map이지만, 작은 물체를 detect하기에는 약간 충분하지 않을 수 있다. 따라서 26x26 layer 에서 그 다음 conv하지 않고 26x26x512의 특징맵을 13x13x(512x4)로 변환한다음(26x26에서 중심을 기준으로 2x2로 나눈 네 조각을 concatenate) detection을 위한 output으로 이어준다

[발췌 자료] [PR-023] YOLO9000 Better, Faster, Stronger - JinWon Lee



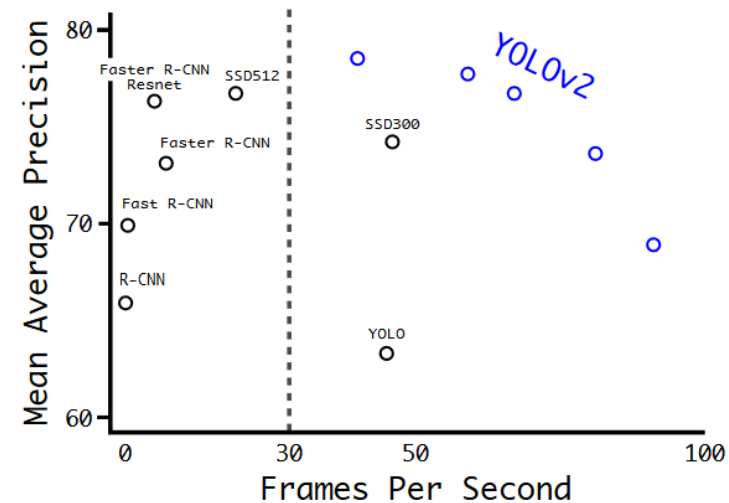
Better

Multi-Scale Training

- YOLOv1 used 448x448, which got changed to 416x416 by adding anchor boxes. But has no fc layers (just convolutional and pooling layers) → **Can be resized on the fly.**
- Every few iterations - every 10 batches, YOLOv2 randomly choose a new image dimension size** in {320, 352, ..., 608) as downsamples by 32 factor.
- Effect of the input size
 - ✓ **At low resolution, fair detection performance but runs fast.** 91 FPS and 69.0 mAP at At 288x288,
 - ✓ **At high resolution, SOTA detection (78.6 mAP) and slower but still above real time speeds (40 FPS)**

[발췌 자료] [PR-023] YOLO9000 Better, Faster, Stronger - JinWon Lee

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

Detection frameworks
on VOC 2007 datasetAccuracy and speed
on VOC 2007

Faster

Darknet-19

- *Classification model* of the base of YOLOv2, based on the *Googlenet architecture*
- Use mostly **3x3 filters** and double the number of channels after every pooling step (similar to VGG)
- Following the work on network in Network (NIN), use **global average pooling** to make predictions as well as **1x1 filters to compress the feature representation** between 3x3 convolutions
- Use **batch normalization** to stabilize training, speed up convergence, and regularize the model.
- Darknet-19 : **19 convolutional layers & 5 max-pooling layers**
- **5.58 billion operations** : **72.9% top-1** & **91.2% top-5** accuracy (30.69 billion operations & 90.0% top-5 accuracy for VGG-16)

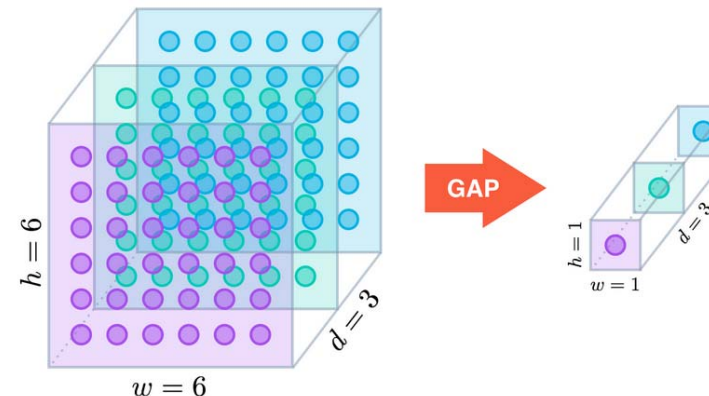
[발췌 자료] [PR-023] YOLO9000 Better, Faster, Stronger - JinWon Lee

Darknet19: A good balance of speed and accuracy

	Top 1	Top 5	FLOPs	GPU Speed
VGG-16	70.5	90.0	30.95 Bn	100 FPS
Extraction (YOLOv1)	72.5	90.8	8.52 Bn	180 FPS
Resnet50	75.3	92.2	7.66 Bn	90 FPS
Darknet19	74.0	91.8	5.58 Bn	200 FPS

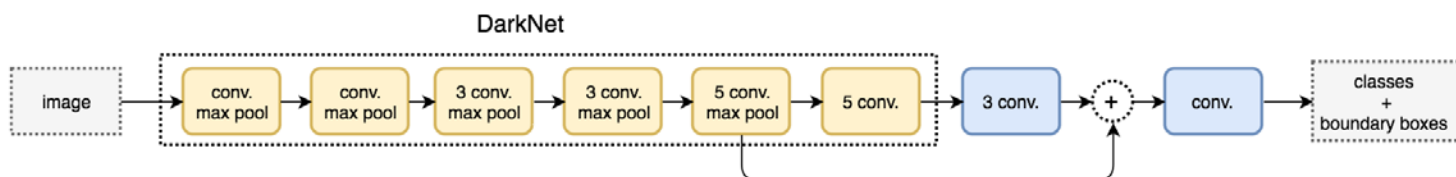
Darknet19 compared in accuracy, speed and #parameters to other models.

- ✓ **Global average pooling layers** are used to cut down overfitting by reducing the number of parameters in the network.
- ✓ It's a type of dimensionality reduction where tensors having dimensions of $(h \times w \times d)$ are converted to have a dimension of $(1 \times 1 \times d)$



Faster

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	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		$2 \times 2/2$	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			



Model architecture

The portion striked through in the image above is replaced with **three 3×3 conv layers (1024 filters each)** followed by a **1×1 conv layer** with the number of outputs needed for detection, the final output is $7 \times 7 \times 125$.

We replace the last convolution layer (the cross-out section) with three 3×3 convolutional layers each outputting 1024 output channels. Then we apply a final 1×1 convolutional layer to convert the $7 \times 7 \times 1024$ output into $7 \times 7 \times 125$. (5 boundary boxes each with 4 parameters for the box, 1 objectness score and 20 conditional class probabilities)

Darknet-19

Model
architecture

Faster

Training for classification

- Trained on ImageNet 1000 classes for 160 epochs
- **Standard data augmentation** : random crops, rotations, hue, saturation, and exposure shifts.
- Initial training : 224x224 → 448x448 fine-tuning for 10 epochs
- Higher resolution achieves a top-5 accuracy of 93.3%

Training for detection

- Modify for detection by removing the last conv layer and instead adding **3x3 conv layers with 1024 filters / channels each followed by a final 1x1 conv layer with the number of outputs we need for detection.**
- For VOC, predict **5 boxes with 5 coordinate each and 20 classes per box, so 125 filters / channels**

$$B \times (4 + 1 + C) - 5 \times (5 + 20) = 125$$

- We also add a **passthrough layer** from the final $3 \times 3 \times 512$ layer to the second to last convolutional layer so that our model can use fine grain features.
- 160 epochs with a start learning rate of 10^{-3} dividing it by 10 at 60 and 90 epochs

Stronger

- image classification은 클래스가 몇천~몇만개 정도로 많지만 detection의 라벨은 20~몇 백개 정도가 전부이다. 이런 gap을 완화하기위한 전략을 소개
 - ✓ training때 classification과 detection data를 섞어서 씀
 - ✓ dataset에서 detection data가 들어오면 원래 loss function을 활용해 계산
 - ✓ dataset에서 classification data가 들어오면 loss function에서 classification loss만 활용해 계산
- 여기서 의문점은, detection에는 ‘개’라고 되어있지만 classification에서는 ‘시츄’, ‘비글’, ‘푸들’ 등과 같이 개 종류만 수백 종류가 있다. 따라서 라벨들을 일관성있게 통합해야하고, 시츄를 개라고 했다고 해서 완전히 틀린 것은 아니므로 상호 배타적이지 않은 예시에 대해서 multi-label 모델을 사용한다.
- Vanilla **detection datasets** contain images in the range 10^3 - 10^5 with dozens to hundreds of tags. (e.g. the **COCO dataset** has **330K images with just 80 object categories**)
- **Classification datasets** are huge in that, they contain millions of images with thousands to tens of thousands of classes (e.g. **ImageNet** contains around **14 million images** with ~22,000 distinct object classes).
- We could use a multi-label model to combine the detection and classification datasets which does not assume mutual exclusions.

Hierarchical classification – Combining detection and classification datasets

- ImageNet labels are pulled from WordNet, a language DB that structures concepts and how they relate.
 - ✓ “Norfolk terrier” and “Yorkshire terrier” are both hyponyms of “terrier” which is a type of “hunting dog”, which is a type of “dog”, which is a “canine”, etc
- WordNet is structured as a directed graph, not a tree, because language is complex. Instead of using the full graph structure, we **build a hierarchical tree** from the concepts in ImageNet.
- To make a tree not a graph, if a concept has two paths to the root, **choose the shorter path**
- Root node is a “Physical Object”

Stronger

Hierarchical classification – Combining detection and classification datasets

- To perform classification with **WordTree**, predicting conditional probabilities at every node for the probability of each hyponym of that synset

$$Pr(\text{Norfolk terrier} | \text{terrier})$$

$$Pr(\text{Yorkshire terrier} | \text{terrier})$$

$$Pr(\text{Bedlington terrier} | \text{terrier})$$

...

- Then if a picture of a Norfolk terrier is encountered, its probability is calculated as below

$$Pr(\text{Norfolk terrier}) = Pr(\text{Norfolk terrier} | \text{terrier})$$

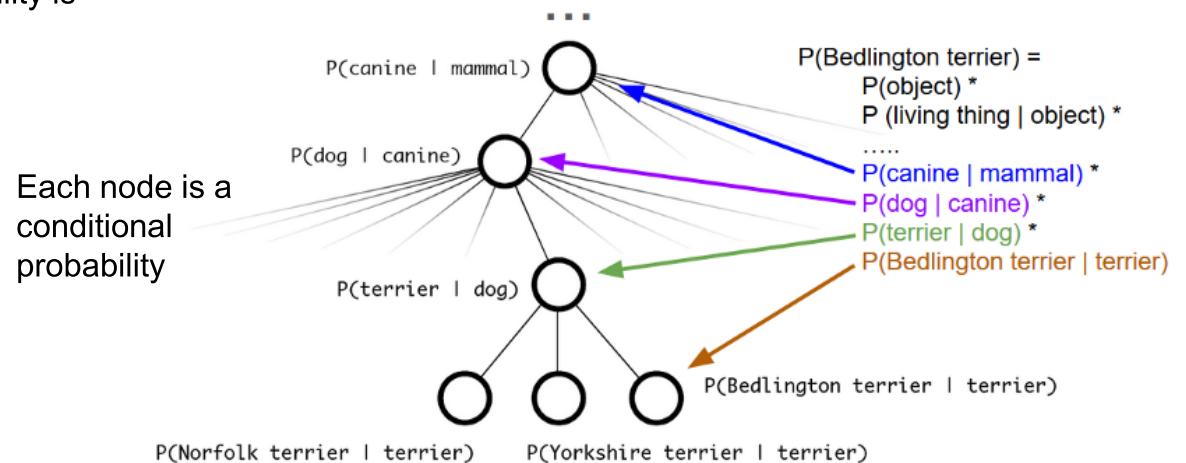
$$* Pr(\text{terrier} | \text{hunting dog})$$

* ... *

$$* Pr(\text{mammal} | Pr(\text{animal}))$$

$$* Pr(\text{animal} | \text{physical object})$$

- Train Darkent-19 model on WordTree build using 1000 class ImageNet.
- To build **WordTree1k**, adding in all of the intermediate nodes which expands the label space from 1000 to 1369.
- During training, ground truth labels are propagated up
 - ✓ If an image is labelled as a “Norfolk terrier” it also gets labelled as a “dog” and a “mammal” etc.
- To compute the **conditional probabilities**, our model predicts a vector of 1369 values and we compute the softmax over all synsets that are hyponyms of the same concept → 90.4% top-5 accuracy



Stronger

Hierarchical classification – Combining detection and classification datasets

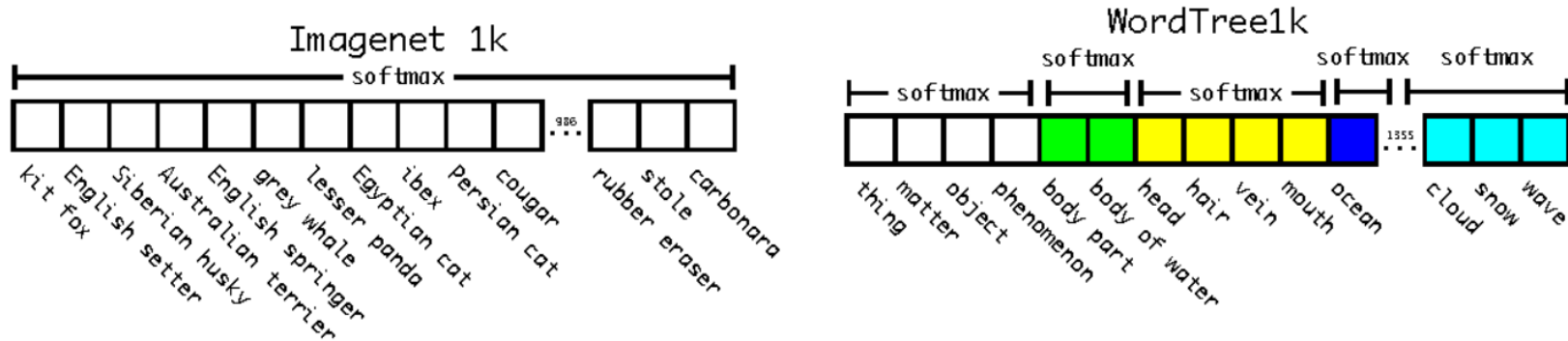


Figure 5: Prediction on ImageNet vs WordTree. Most ImageNet models use one large softmax to predict a probability distribution. Using WordTree we perform multiple softmax operations over co-hyponyms.

Stronger

Dataset Combination with WordTree

- Example of using WordTree to combine the labels from ImageNet and COCO.
 - ✓ COCO – **general concept** (higher node)
 - ✓ ImageNet – **specific concepts** (lower nodes and leaves)

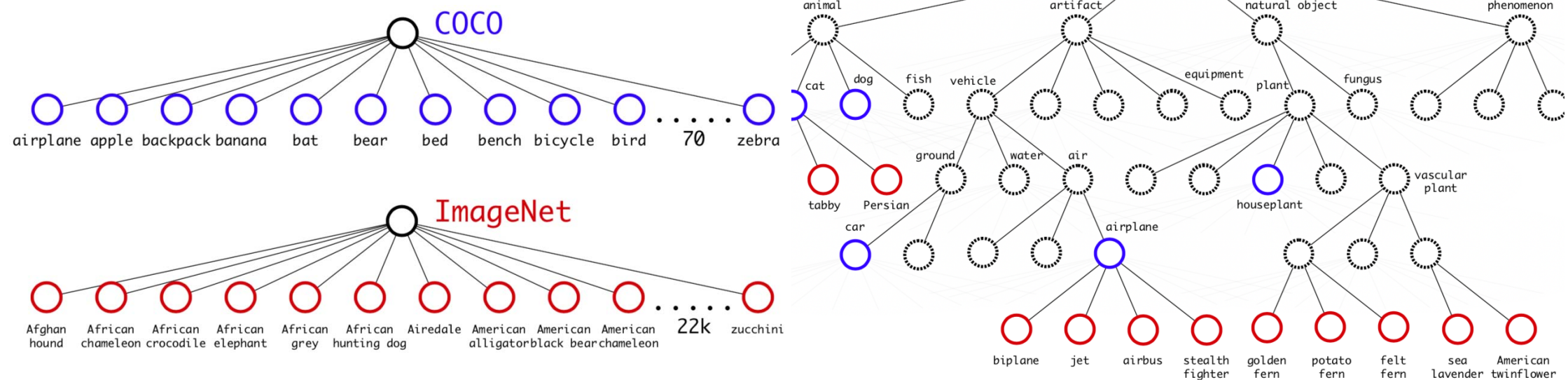


Figure 6: Combining datasets using WordTree hierarchy. Using the WordNet concept graph we build a hierarchical tree of visual concepts. Then we can merge datasets together by mapping the classes in the dataset to synsets in the tree. This is a simplified view of WordTree for illustration purposes.

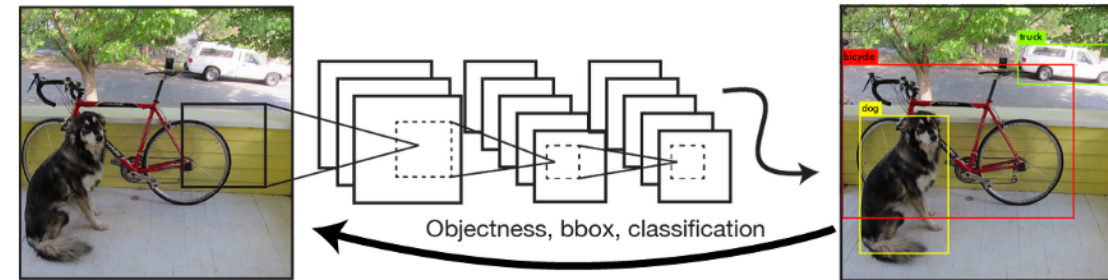
Stronger

Joint Classification and Detection (YOLO9000)

- Created new combined dataset using the **COCO dataset (for detection)** and the **top 9000 classes from the full ImageNet (for classification) release**. - the combined dataset contains 9418 classes.
- ImageNet detection challenge dataset is also added for evaluation.
- To tackle this unbalanced dataset problem, we oversample COCO so that the ratio of images in ImageNet and COCO is 4:1
- We train YOLO9000 model using YOLOv2 but only **3 priors (anchor boxes)** instead of 5 to limit output size.
- On a detection image, the network backpropagates loss as usual.**
 - *objectness* (box confidence score, i.e. confidence that the image contains an object), bounding box, and classification error
- On a classification image**, the network backpropagates *objectness loss* and *classification loss*.
- For *objectness loss*, we backpropagate using the assumption that the predicted box overlaps the ground truth label by ≥ 0.3 IOU.
- For *classification loss*, we find the bounding box that predicts the highest probability for the class and the loss on just its predicted tree is computed.

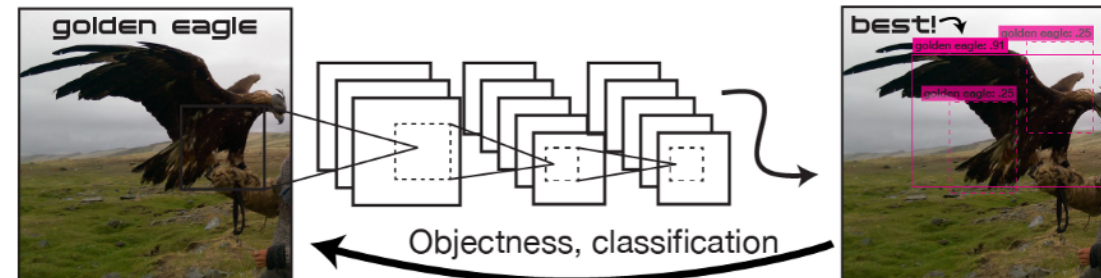
[발췌 자료] [PR-023] YOLO9000 Better, Faster, Stronger - JinWon Lee

When you see detection data, backpropagate objectness, bbox, and classification error.



Backpropagating loss on a detection image

When you see classification data, pick best prediction, backpropagate objectness, classification error.

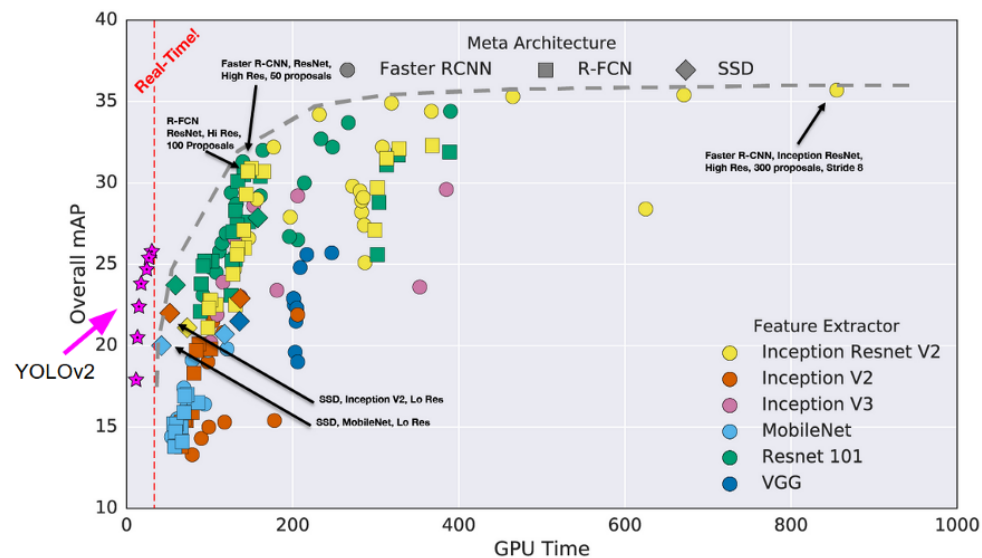


Backpropagating loss on a classification image

Stronger

Joint Classification and Detection (YOLO9000)

- Evaluating YOLO9000 on the ImageNet detection task (200 categories)
 - ✓ ImageNet only share 44 object categories with COCO (80 categories)
 - ✓ 19.7 mAP overall with 16.0 mAP on the disjoint 156 object classes that it has never seen
- YOLO9000 learns new species of animals well but struggles with learning categories like clothing and equipment.



Accuracy vs. speed, for various architectures and feature extractors

diaper	0.0
horizontal bar	0.0
rubber eraser	0.0
sunglasses	0.0
swimming trunks	0.0
...	
red panda	50.7
fox	52.1
koala bear	54.3
tiger	61.0
armadillo	61.7

Table 7: YOLO9000 Best and Worst Classes on ImageNet. The classes with the highest and lowest AP from the 156 weakly supervised classes. YOLO9000 learns good models for a variety of animals but struggles with new classes like clothing or equipment.

Joseph Redmon, Ali Farhadi, "YOLOv3: An Incremental Improvement," arXiv:1804.02767

<https://arxiv.org/abs/1804.02767>

<https://pjreddie.com/darknet/yolo/>

What's new in YOLO v3?

<https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b>

How to implement a YOLO (v3) object detector from scratch in PyTorch

<https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/>

- This is a **TECH REPORT**
- I managed to make some improvements to YOLO. But, honestly, nothing like super interesting, just **a bunch of small changes that make it better**
- Better, Not Faster, Stronger(?)

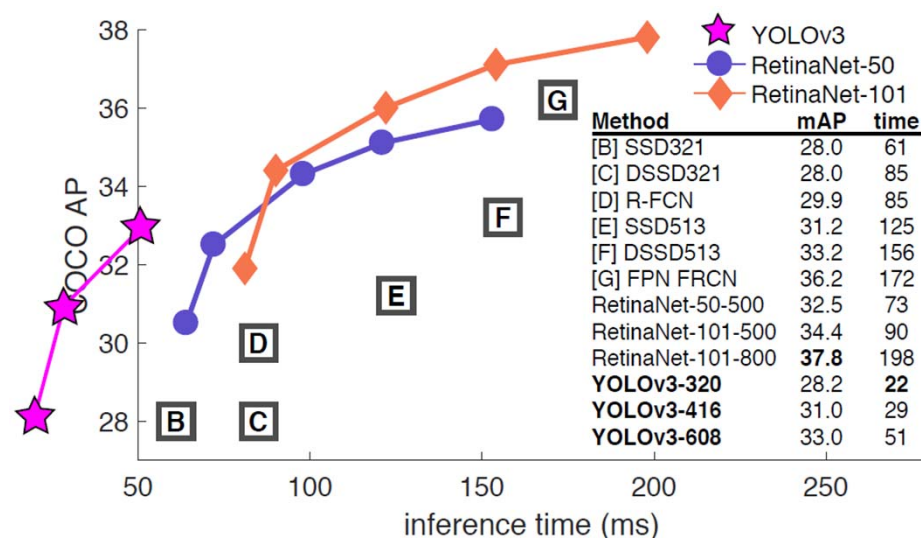
Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

[발췌 자료] [PR-207] YOLOv3: An Incremental Improvement - JinWon Lee

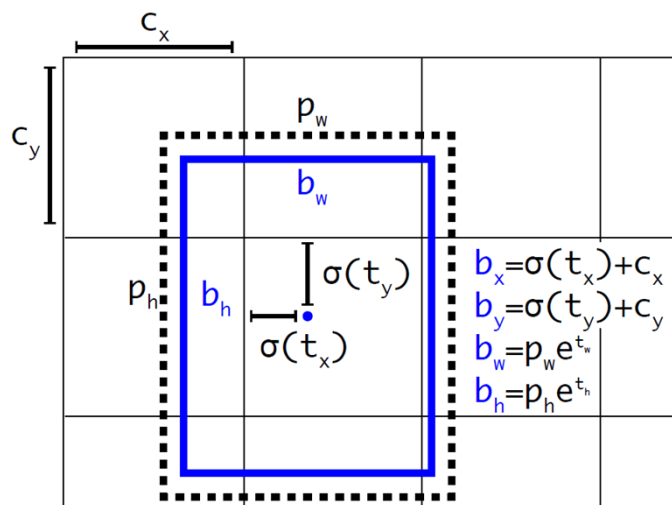
Abstract

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 320x320 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 AP₅₀ in 51 ms on a Titan X, compared to 57.5 AP₅₀ in 198 ms by RetinaNet, similar performance but 3.8x faster.



Bounding Box Prediction



- YOLOv3 predicts bounding boxes using dimension clusters as anchor boxes. The network predicts 4 coordinates for each bounding box, t_x, t_y, t_w, t_h

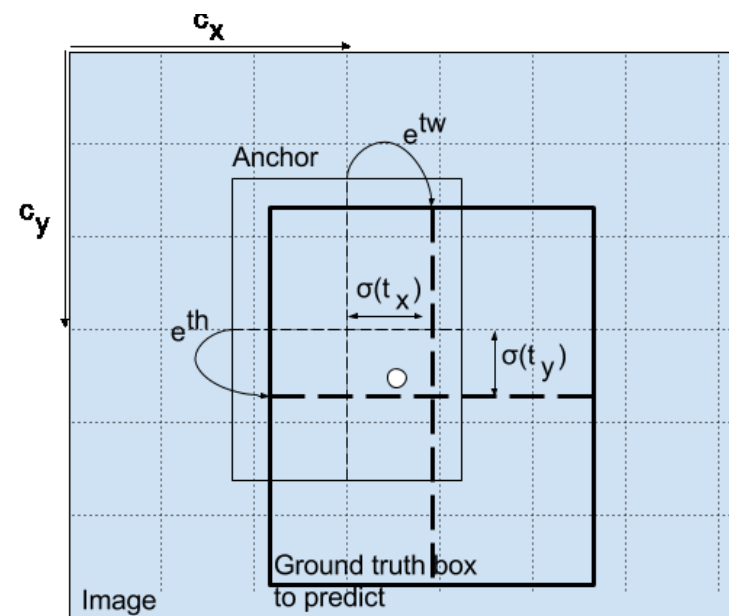
- Use sum of squared error loss

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$



- YOLOv3 predicts **an objectness score for each bounding box using logistic regression.**
- This should be 1 if **the bounding box prior overlaps a ground truth object by more than any other bounding box prior.**
- They use the threshold of 0.5. Unlike Faster R CNN, **YOLOv3 only assigns one bounding box prior for each ground truth object.**

Class Prediction

- YOLO v3 now performs **multilabel classification** for objects detected in images
- Softmaxing classes rests on the assumption that classes are mutually exclusive.
- However, when we have classes like Person and Women in a dataset, then the above assumption fails. This is the reason why the authors of YOLO have refrained from softmaxing the classes. **Instead, each class score is predicted using logistic regression and a threshold is used to predict multiple labels for an object**

3

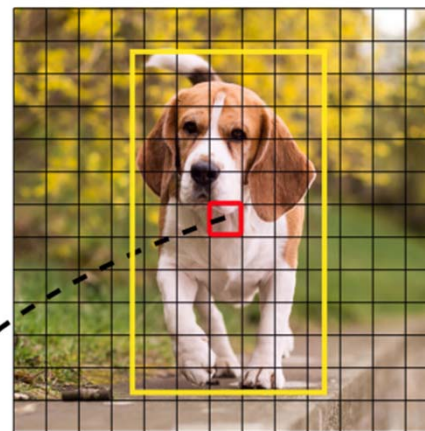
Yolo v3 : An Incremental Improvement

[발췌 자료] [PR-207] YOLOv3: An Incremental Improvement - JinWon Lee

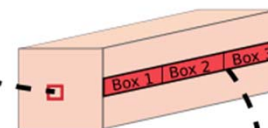
Predictions Across Scales

- The most salient feature of YOLOv3 is that it makes detections at three different scales
 - YOLOv3 predicts boxes at 3 scales
 - YOLOv3 predicts 3 boxes at each scale
- in total 9 boxes
- ✓ So the tensor is $N \times N \times (3 \times (4 + 1 + 80))$
 - ✓ for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions

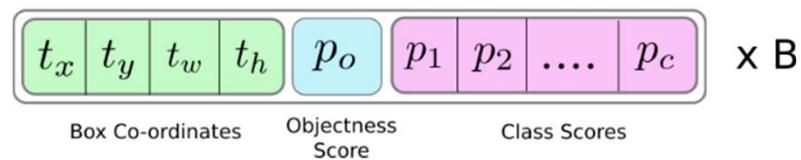
Image Grid. The Red Grid is responsible for detecting the dog



Prediction Feature Map



Attributes of a bounding box



Anchor Box

- They still use **k means clustering to determine bounding box priors**. They just sort of chose 9 clusters and 3 scales arbitrarily and then divide up the clusters evenly across scales.
- On the COCO dataset the 9 clusters were: (10x13), (16x30), (33x23), (30x61), (62x45), (59x119), (116x90), (156x198), (373x326)

Number of Bounding Box

- **YOLOv1** predicts **98 boxes** (7x7 grid cells, 2 boxes per cell @448x448)
- **YOLOv2** predicts **845 boxes** (13x13 grid cells, 5 anchor boxes @416x416)
- **YOLOv3** predicts **10,647 boxes** (@416x416)
- YOLOv3 predicts more than **10x the number of boxes** predicted by YOLOv2

Feature Extraction

• Darknet-53

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.

- Darknet-53 is better than ResNet-101 and 1.5x faster. **Darknet-53 has similar performance to ResNet-152 and is 2x faster**
- Darknet-53 also achieves **the highest measured floating point operations per second**. This means the network structure better utilizes the GPU, making it more efficient to evaluate and thus faster.

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

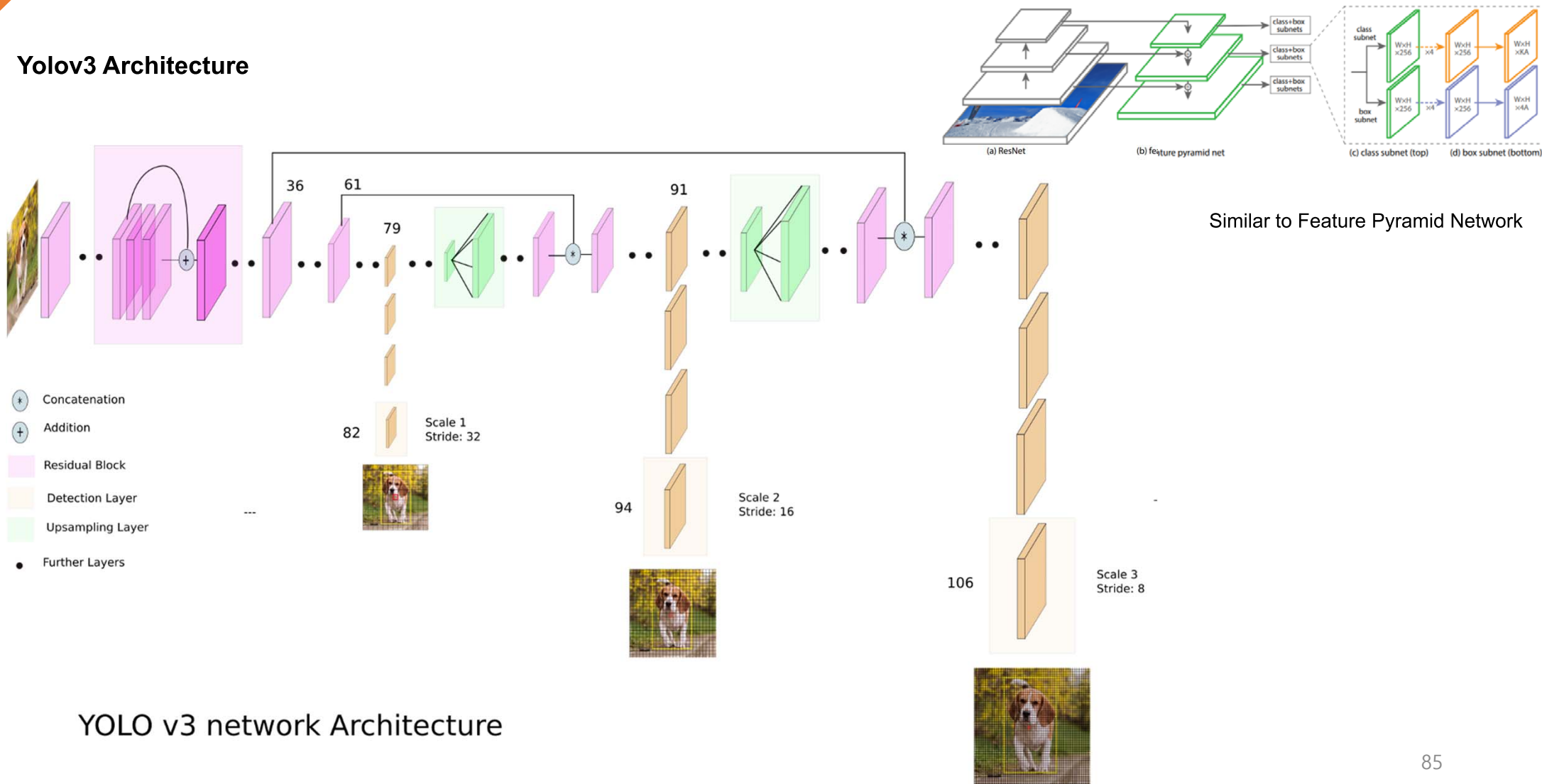
Table 1. **Darknet-53.**

3

Yolo v3 : An Incremental Improvement

[발췌 자료] [PR-207] YOLOv3: An Incremental Improvement - JinWon Lee

Yolov3 Architecture



Training

- Authors still train on **full images with no hard negative mining** or any of that stuff.
- They use multi scale training, lots of data augmentation, batch normalization, all the standard stuff.

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 AP50 metric.

	backbone	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
<i>Two-stage methods</i>							
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
<i>One-stage methods</i>							
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

Results

- It is still quite a bit behind other models like RetinaNet in this metric though
- However, when we look at the “old” detection metric of **mAP at IOU=0.5 (or AP₅₀ in the chart)**, YOLOv3 is very strong.
- With the new multi scale predictions, **YOLOv3 has relatively high APs performances.**

Results

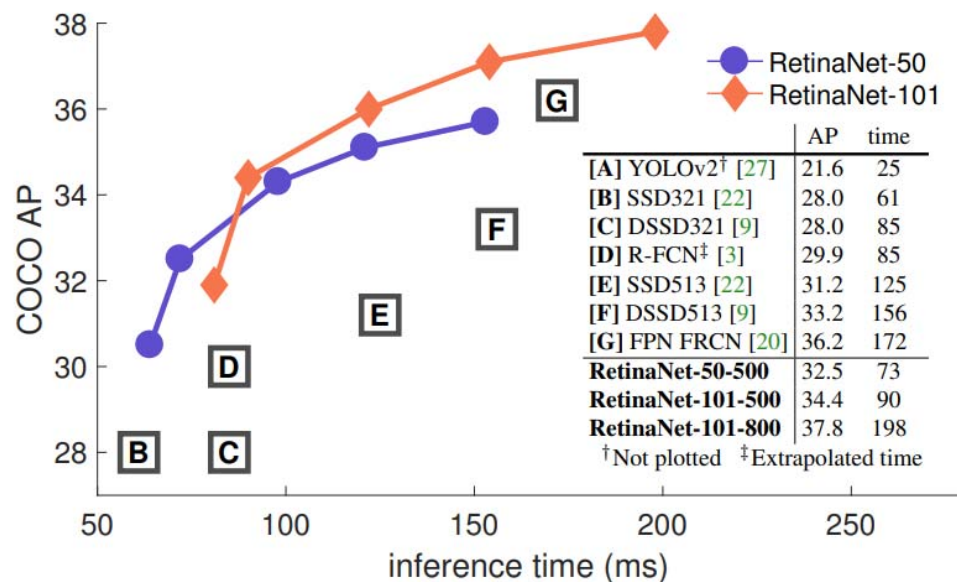
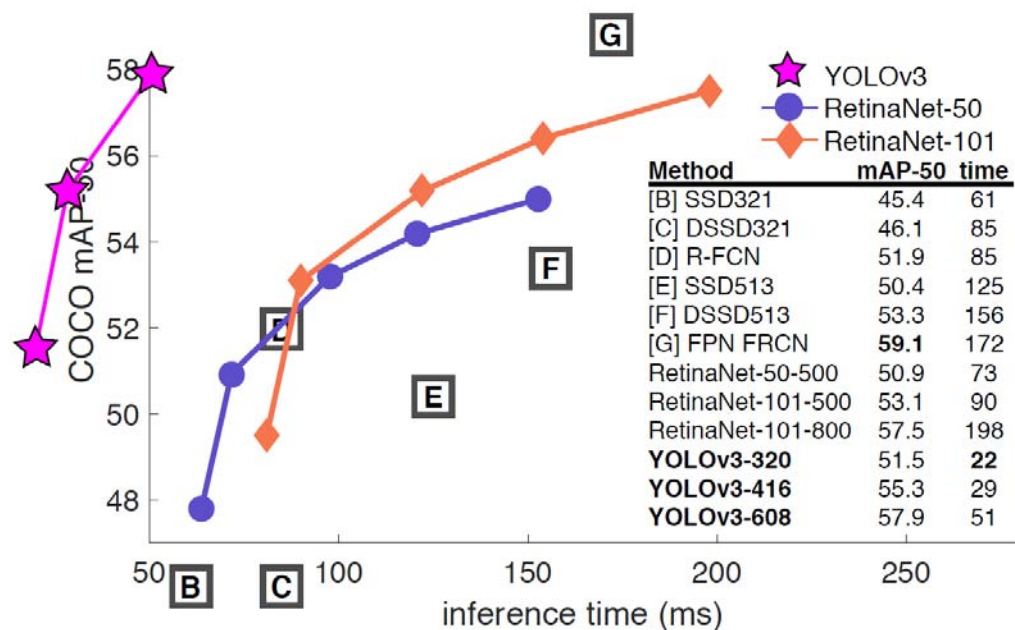


Figure 3. Again adapted from the [9], **this time displaying speed/accuracy tradeoff on the mAP at .5 IOU metric**. You can tell YOLOv3 is good because it's very high and far to the left. Can you cite your own paper? Guess who's going to try, this guy ! [16]. Oh, I forgot, we also fix a data loading bug in YOLOv2, that helped by like 2 mAP. Just sneaking this in here to not throw off layout.

Things We Tried That Didn't Work

- **Anchor box x, y offset predictions**
 - ✓ Using the normal anchor box prediction mechanism where you predict the x, y offset as a multiple of the box width or height
- **Linear x, y predictions instead of logistic**
- **Focal loss**
 - ✓ Focal loss dropped YOLOv3's mAP about 2 points
- **Dual IOU thresholds and truth assignment**
 - ✓ Faster R CNN uses two IOU thresholds during training. If a prediction overlaps the ground truth by 0.7 it is as a positive example, by [0.3 0.7] it is ignored, less than 0.3 for all ground truth objects it is a negative example.

$$\hat{G}_x = P_w d_x(P) + P_x$$

$$\hat{G}_y = P_h d_y(P) + P_y$$

$$\hat{G}_w = P_w \exp(d_w(P))$$

$$\hat{G}_h = P_h \exp(d_h(P)).$$

What This All Means

- YOLOv3 is a good detector. **It's fast, it's accurate** . It's not as great on the COCO average AP between .5 and .95 IOU metric. But **it's very good on the old detection metric of .5 IOU**
- Russakovsky et al report that that **humans have a hard time distinguishing an IOU of .3 from .5**
 - ✓ Training humans to visually inspect a bounding box with IOU of 0.3 and distinguish it from one with IOU 0.5 is surprisingly difficult.

Rebuttal

- Graphs have not one but two non zero origins

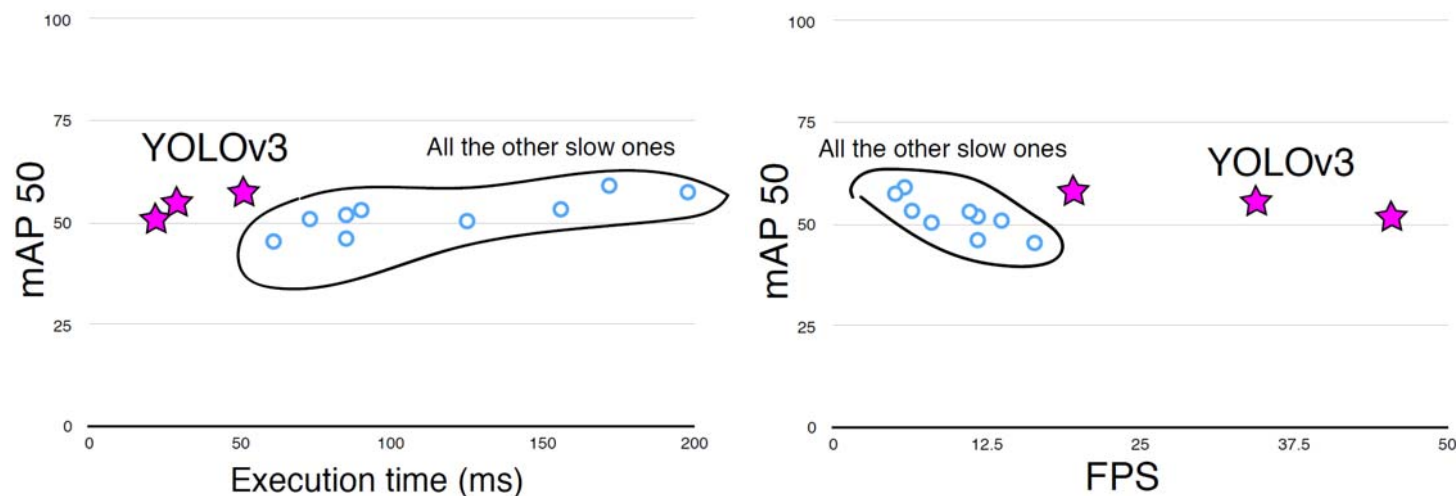


Figure 4. Zero-axis charts are probably more intellectually honest... and we can still screw with the variables to make ourselves look good!

- For PASCAL VOC, the IOU threshold was “set deliberately low to account for inaccuracies in bounding boxes in the ground truth data”.
- COCO can have better labelling than VOC since COCO has segmentation masks. But there was the lack of justification for updating mAP.
- Emphasis must mean it de emphasizes something else, in this case classification accuracy. A miss classified example is much more obvious than a bounding box that is slightly shifted.

mAP problems

- They are both perfect! (mAP = 1.0)

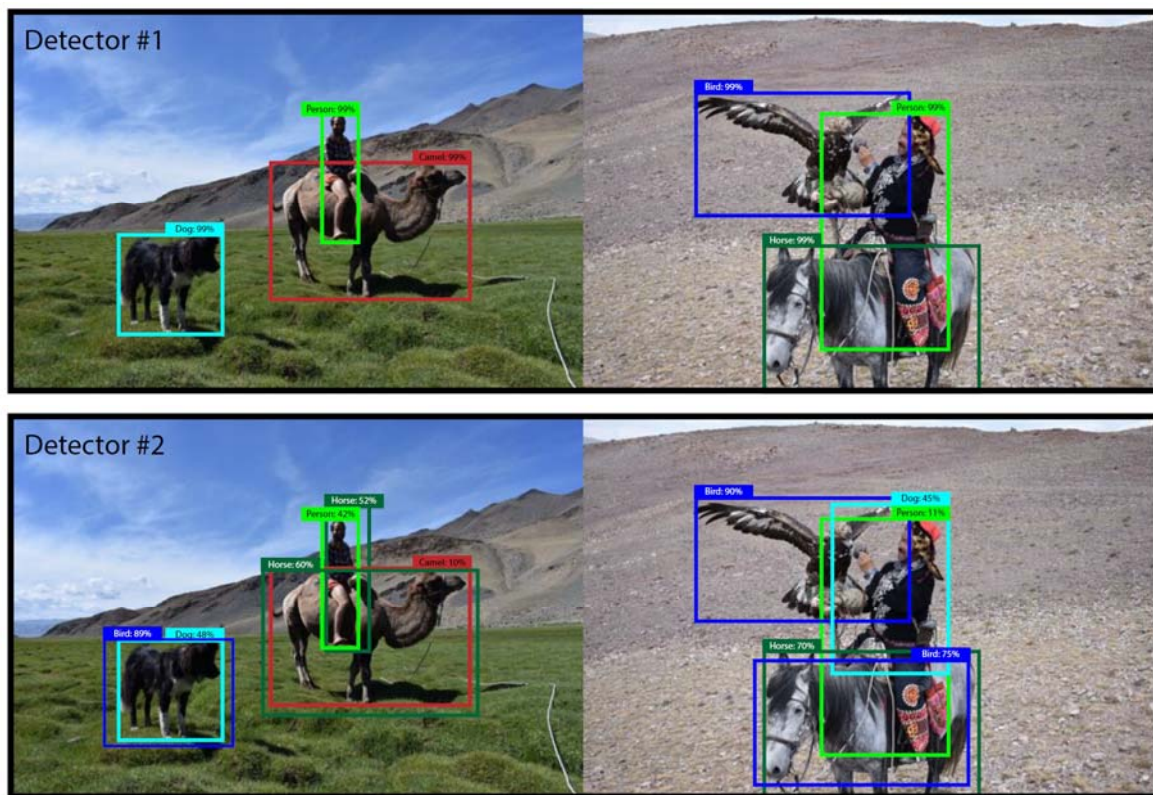


Figure 5. These two hypothetical detectors are perfect according to mAP over these two images. They are both perfect. Totally equal.

New Proposal

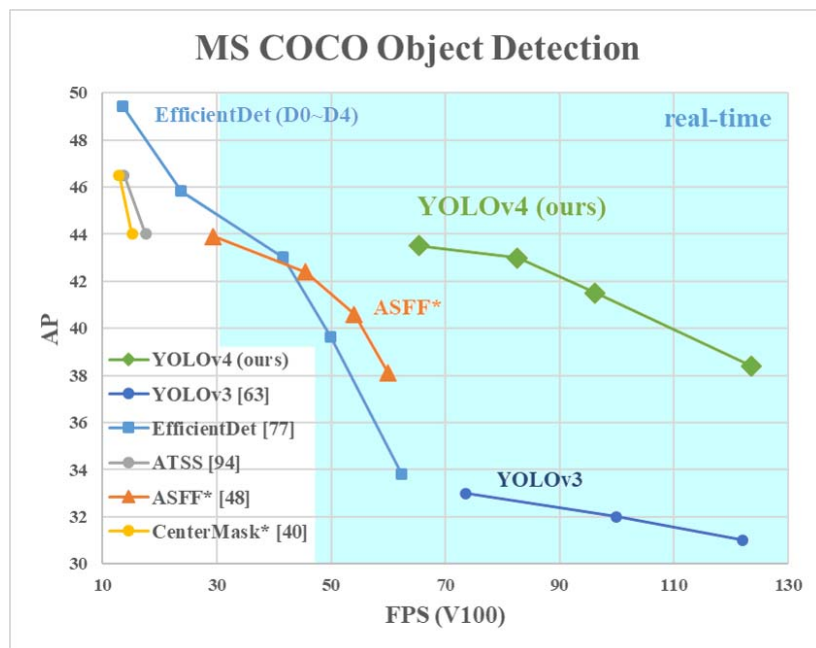
- mAP is screwed up because **all that matters is per class rank ordering**
- What about getting rid of per class AP and just doing **a global average precision?**
- Or doing an **AP calculation per image and averaging over that?**
- Boxes are stupid anyway though, I'm probably a true believer in masks except I can't get YOLO to learn them.

Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection”, arXiv:2004.10934, 2020

<https://arxiv.org/abs/2004.10934>

<https://github.com/AlexeyAB/darknet>

<https://hoya012.github.io/blog/yolov4/>



Abstract

There are a huge number of features which are said to improve Convolutional Neural Network (CNN) accuracy. **Practical testing of combinations of such features on large datasets, and theoretical justification of the result**, is required.

Some features operate on certain models exclusively and for certain problems exclusively, or only for small-scale datasets; while some features, such as batch-normalization and residual-connections, are applicable to the majority of models, tasks, and datasets.

We assume that such universal features include Weighted-Residual-Connections (WRC), Cross-Stage-Partial-connections (CSP), Cross mini-Batch Normalization (CmBN), Self-adversarial-training (SAT) and Mish-activation.

We use new features: **WRC, CSP, CmBN, SAT, Mish activation, Mosaic data augmentation, CmBN, DropBlock regularization, and CloU loss**, and combine some of them to achieve state-of-the-art results: 43.5% AP (65.7% AP50) for the MS COCO dataset at a realtime speed of 65 FPS on Tesla V100. Source code is at

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

4

YOLO v4 : Optimal Speed and Accuracy of Object Detection

[발췌 자료] [PR-249] YOLOv4: Optimal Speed and Accuracy of Object Detection – Ho Seung Lee

Introduction

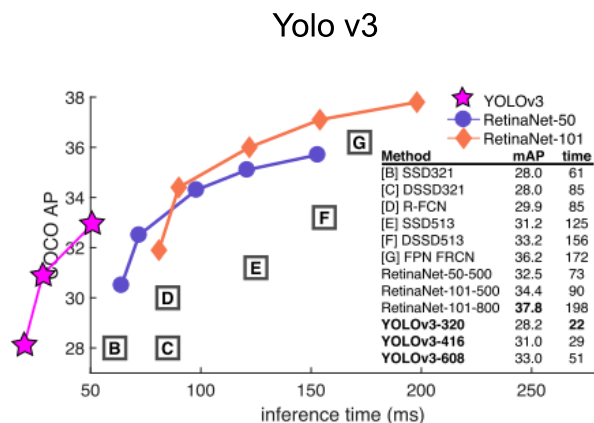
The majority of Object Detectors are largely applicable **only for** recommendation systems

- Searching for free parking space → it's okay to be slow → more accurate
- Car collision warning → need to fast → inaccurate

→ **Need to design a fast and accurate object detector for production systems**

Main Contributions

- We develop an efficient and powerful object detection model. It makes everyone can use a **1080 Ti or 2080 Ti GPU to train a super fast and accurate object detector**.
- We **verify the influence of state-of-the-art Bag-of-Freebies and Bag-of-Specials methods** of object detection during the detector training.
- We modify state-of-the-art methods and make them **more efficient and suitable for single GPU training**, including **CBN** [89], **PAN** [49], **SAM** [85], etc.

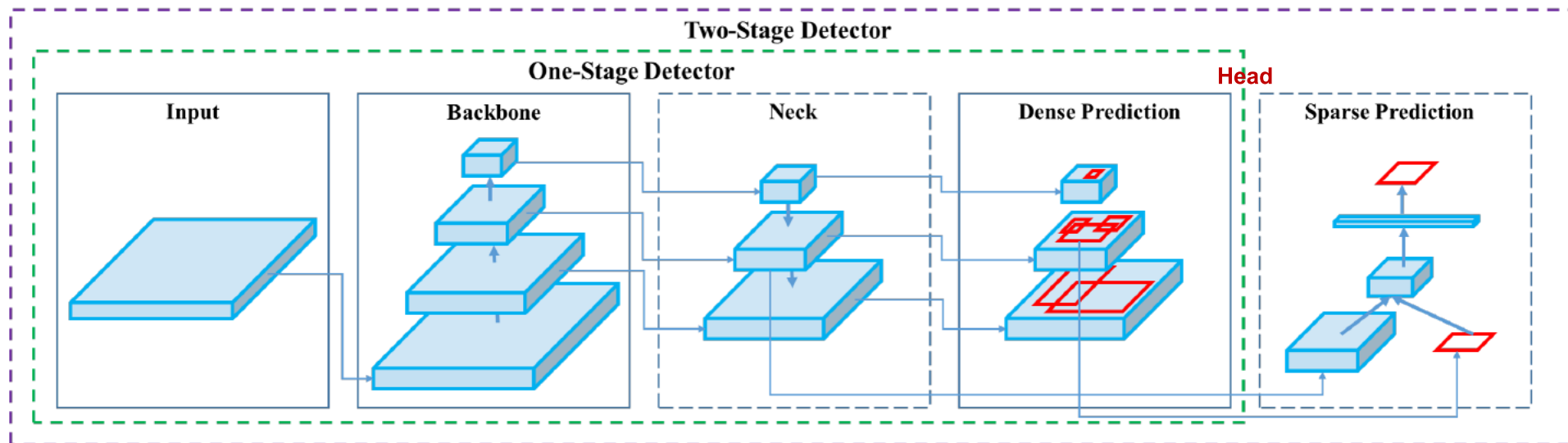


Bag of Freebies

Bag of Specials

CSPDarknet53 + SPP, PAN

Related work : Object detection models



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 RCNN [23] (*anchor based*), RepPoints [87] (*anchor free*)

Related work : Bag of Freebies (Pre-processing + Training strategy)

Training Stage

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

Data Augmentation

- Random erase
- CutOut
- MixUp
- CutMix
- Style transfer GAN

Regularization

- DropOut
- DropPath
- Spatial DropOut
- DropBlock

Loss Function

- MSE
- IoU
- GIoU **Generalized**
- CIoU **Complete**
- DIoU **Distance**

Related work : Bag of Specials (Plugin modules + Post-processing)

Inference Stage

architecture related

- Call methods that only increase the **inference cost** but can improve the accuracy as “BoS”

Enhancement of receptive field

- Spatial Pyramid Pooling
- ASPP (dilated conv)
- Receptive Field Block (RFB)

Feature Integration

- Skip-connection
- Feature Pyramid Network
- SFAM (Scale-wise Feature Aggregation Module)
- ASFF (adaptively spatial feature fusion)
- BiFPN

Activation function

- ReLU
- Leaky ReLU
- Parametric ReLU
- ReLU6
- Swish
- Mish

Attention Module

- Squeeze-and-Excitation (SE)
- Spatial Attention Module (SAM)

Normalization

- Batch Norm (BN)
- Cross-GPU Batch Norm (CGBN or SyncBN)
- Filter Response Normalization (FRN)
- Cross-Iteration Batch Norm (CBN)

Post Processing

- NMS
- Soft NMS
- DIoU NMS

Yolov4 : Selection of architecture

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

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	Better
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	

CSPNet: A New Backbone that can Enhance Learning Capability of CNN, 2020 CVPRW

- Propose Cross Stage Partial Network to mitigate heavy inference computations
- Partition feature map of the base layer into two parts and the merge them

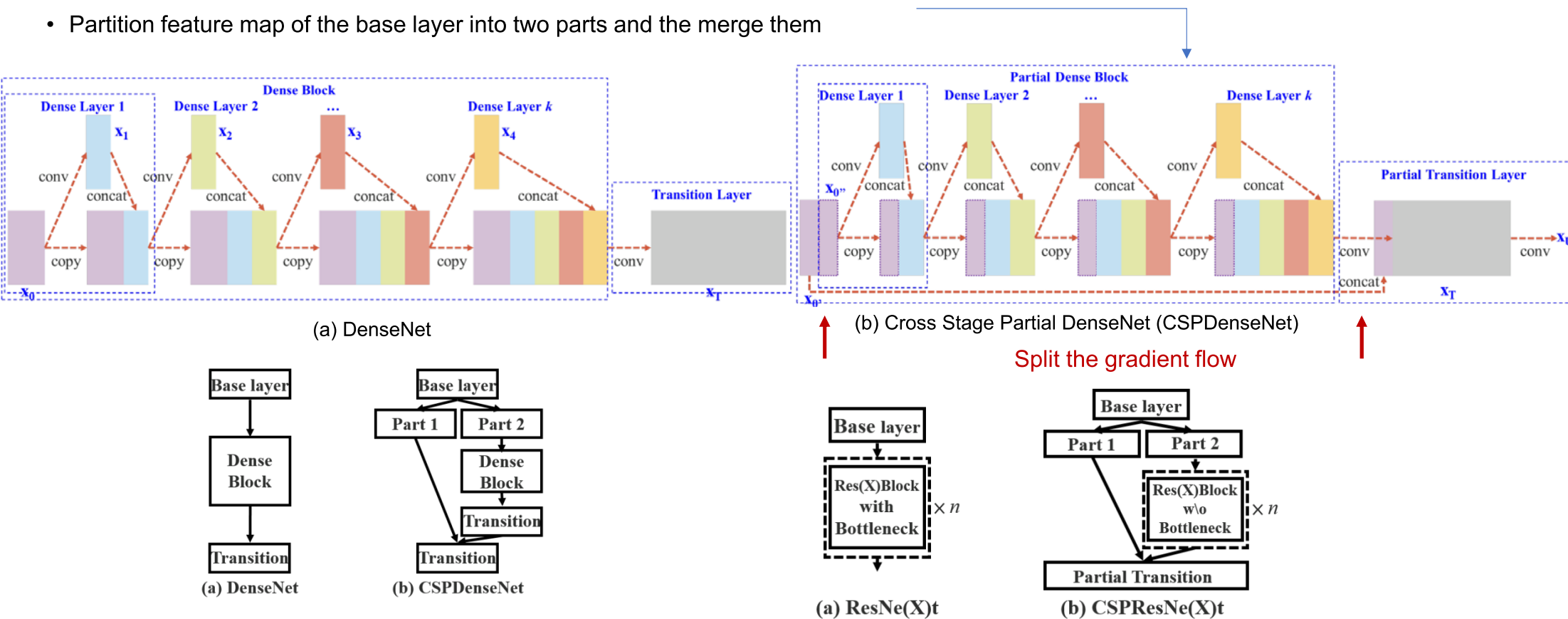
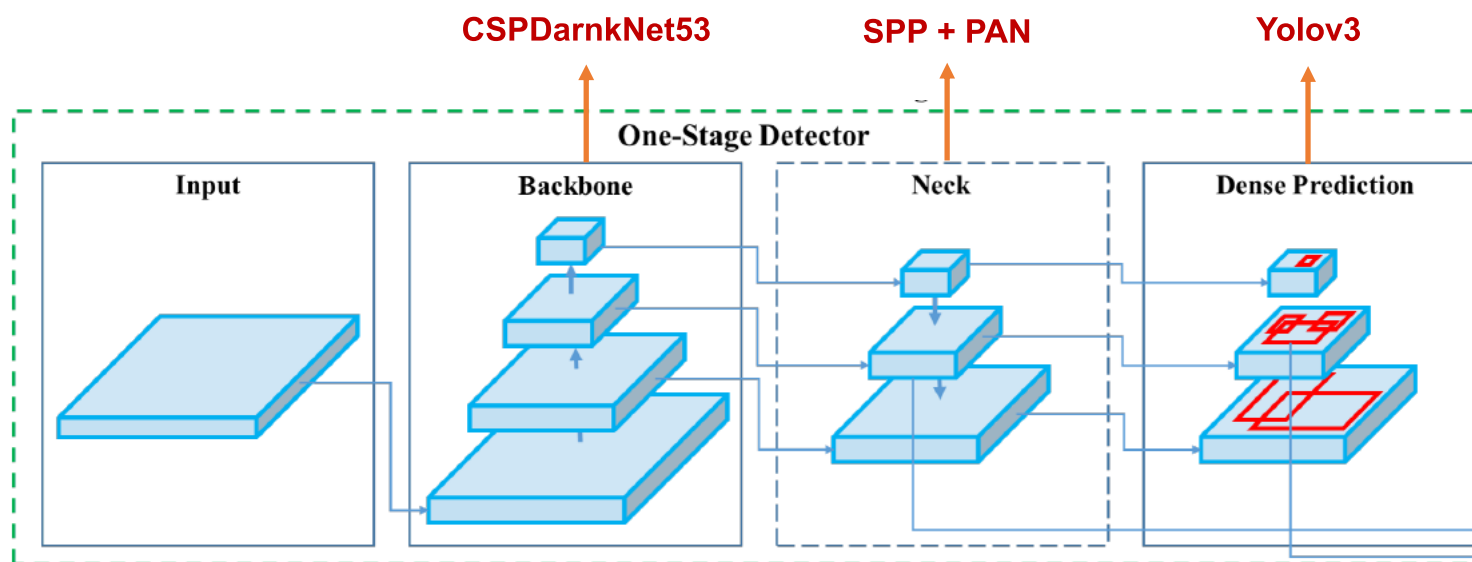


Fig. 4. (a) single path DenseNet, (b) proposed CSPDenseNet: transition → concatenation → transition

Fig. 5. Applying CSPNet to ResNe(X)t.

Yolov4 : Selection of architecture



$$\text{YOLOv4} = \text{YOLOv3} + \text{CSPDarknet53} + \text{SPP} + \text{PAN} + \text{BoF} + \text{BoS}$$

Path Aggregation Network

Yolov4 : Selection of BoF and BoS

- **Activations:** ReLU, leaky-ReLU, ~~parametric ReLU, ReLU6, SELU~~, Swish, or Mish
- **Bounding box regression loss:** MSE, IoU, GIoU, CIoU, DIoU
- **Data augmentation:** CutOut, MixUp, CutMix
- **Regularization method:** ~~DropOut, DropPath [36], Spatial DropOut [79]~~, or DropBlock
- **Normalization of the network activations by their mean and variance:** Batch Normalization (BN) [32], ~~Cross-GPU Batch Normalization (CGBN or SyncBN) [93]~~, Filter Response Normalization (FRN) [70], or Cross-Iteration Batch Normalization (CBN) [89]
- **Skip-connections:** Residual connections, Weighted residual connections, Multi-input weighted residual connections, or Cross stage partial connections (CSP)
- PReLU, SELU → difficult to train
- ReLU6 → designed for quantization network
- DropBlock's author have compared their method with other method and has won a lot
- Only use single GPU → SyncBN is not considered

Yolov4 : Additional Improvements

- Introduce a new **data augmentation Mosaic** and **Self Adversarial Training (SAT)**
- Select optimal hyper parameters while applying genetic algorithms
- Modify some existing methods for efficient training and detection
 - ✓ Modified SAM
 - ✓ Modified PAN
 - ✓ Cross mini Batch Normalization (CmBN)

➤ Data augmentation Mosaic

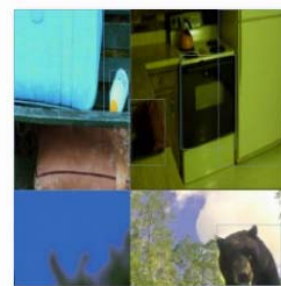
- Mixes 4 training images → allows detection of objects outside their normal context
- BN calculates activation statistics from 4 different images on each layer → reduce the need for large batch size



aug_-319215602_0_-238783579.jpg



aug_-1271888501_0_-749611674.jpg



aug_1462167959_0_-1659206634.jpg



aug_1474493600_0_-45389312.jpg



aug_1715045541_0_603913529.jpg



aug_1779424844_0_-589696888.jpg

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

Yolov4 : Additional Improvements

➤ Self Adversarial Training (SAT)

- New data augmentation technique that operates in 2 forward backward stages
- In the 1 st stage, the NN alters the original image instead of the network weights
→ NN executes an adversarial attack on itself
- In the 2 nd stage, the NN is trained to detect an object on this modified image
- But.. There are no experimental result of SAT

❖ Self-adversarial training - data augmentation #5117

```
[net]
adversarial_lr=1
#attention=1 # just to show attention
```

Note for **Classifier**: it seems it makes training unstable for high learning rate, so you should train 50 of iterations the model as usual, then add `adversarial_lr=0.05` and continue training.

Explanation: If we run attention-algorithm or adversarial-attack-algorithm, then we find that network looks only at rare areas of the object, since it considers them to be the most important, but often the network makes mistakes - these parts of the object are not the most important or do not belong to the object at all, and the network does not notice other details of the object.

Our goal: to make the network take into account a large area

A way to achieve the goal: during training, for every second iteration, the network conducts an Adversarial attack on itself:


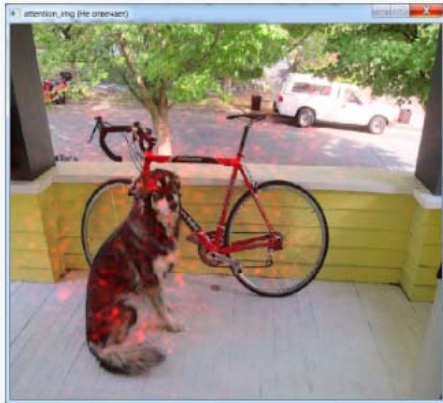
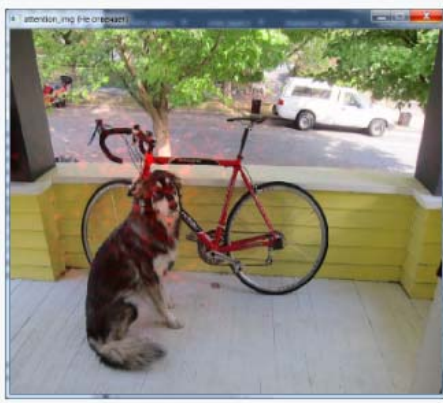
1. In the first forward-backward pass, the network tries to remove from the image all the details that relate to the objects, and makes it believe itself that there is not a single object in the image.
2. In the second run forward-backward, the network teaches its weights that there are still objects here, despite the fact that it seems to it that it is not here.

<https://github.com/AlexeyAB/darknet/issues/5117>

Yolov4 : Additional Improvements

➤ Self Adversarial Training (SAT)

- New data augmentation technique that operates in 2 forward backward stages
- In the 1 st stage, the NN alters the original image instead of the network weights
 - NN executes an adversarial attack on itself
- In the 2 nd stage, the NN is trained to detect an object on this modified image
- But.. There are no experimental result of SAT

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	<code>[net] adversarial_lr=0.05 attention=1</code> the network sees dog/bicycle/car	<code>[net] adversarial_lr=0.05 attention=1</code> (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.

<https://github.com/AlexeyAB/darknet/issues/5117>

Yolov4 : Additional Improvements

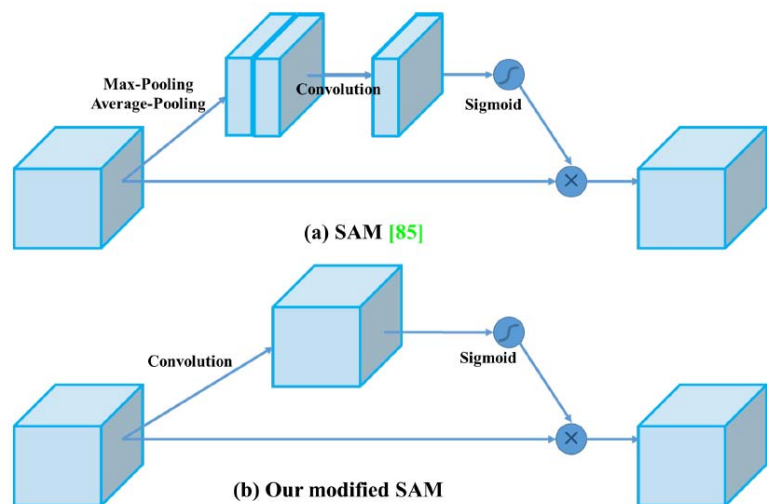


Figure 5: Modified SAM.

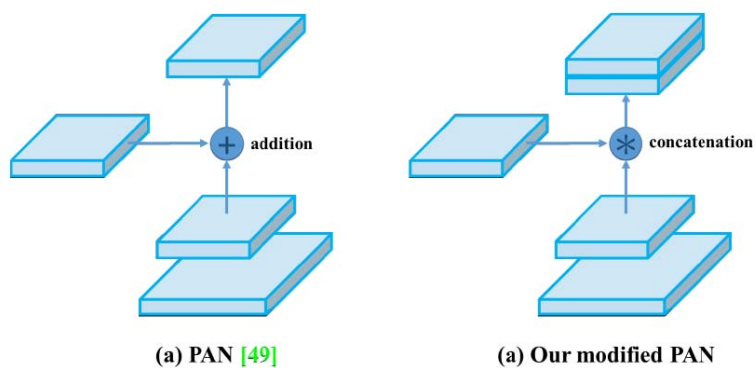


Figure 6: Modified PAN.

- Modify some existing methods for efficient training and detection

- ✓ Modified SAM
- ✓ Modified PAN
- ✓ Cross Mini-batch Normalization (CmBN)

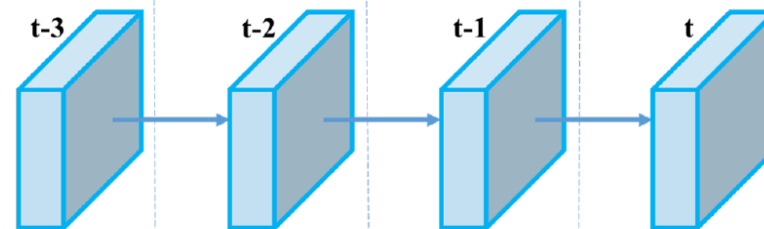
BN [32] – assume a batch contains four mini-batches

accumulate $W^{(t-3)}$ calculate $BN^{(t-3)}$ normalize BN	accumulate $W^{(t-3 \sim t-2)}$ calculate $BN^{(t-2)}$ normalize BN	accumulate $W^{(t-3 \sim t-1)}$ calculate $BN^{(t-1)}$ normalize BN	accumulate $W^{(t-3 \sim t)}$ calculate $BN^{(t)}$ normalize BN update W , ScaleShift
--	---	---	--

CBN [89] – assume cross four iterations

update $W^{(t-3)}$ accumulate $BN^{(t-3 \sim t-6)}$ normalize BN update ScaleShift	update $W^{(t-2)}$ accumulate $BN^{(t-2 \sim t-5)}$ normalize BN update ScaleShift	update $W^{(t-1)}$ accumulate $BN^{(t-1 \sim t-4)}$ normalize BN update ScaleShift	update $W^{(t)}$ accumulate $BN^{(t \sim t-3)}$ normalize BN update ScaleShift
---	---	---	---

Lets:
Bias, scale – ScaleShift
Mean, variance – BN
Weights – W



CmBN – assume a batch contains four mini-batches

accumulate $W^{(t-3)}$ accumulate $BN^{(t-3)}$ normalize BN	accumulate $W^{(t-3 \sim t-2)}$ accumulate $BN^{(t-3 \sim t-2)}$ normalize BN	accumulate $W^{(t-3 \sim t-1)}$ accumulate $BN^{(t-3 \sim t-1)}$ normalize BN	accumulate $W^{(t-3 \sim t)}$ accumulate $BN^{(t-3 \sim t)}$ normalize BN update W , ScaleShift
---	---	---	--

Figure 4: Cross mini-Batch Normalization

Experimental Setup

➤ ImageNet for classification

- Default hyper-parameters are as follows:
 - ✓ training steps is 8,000,000;
 - ✓ the batch size and the mini-batch size are 128 and 32, respectively;
 - ✓ the polynomial decay learning rate scheduling strategy is adopted with initial learning rate 0.1;
 - ✓ the warm-up steps is 1000;
 - ✓ the momentum and weight decay are respectively set as 0.9 and 0.005.
- All of our BoS experiments use the same hyper-parameter as the default setting
- In the BoF experiments, we add an additional 50% training steps and verify MixUp, CutMix, Mosaic, Blurring data augmentation, and label smoothing regularization methods.
- In the BoS experiments, we compared the effects of LReLU, Swish, and Mish activation function.
- All experiments are trained with a 1080 Ti or 2080 Ti GPU.

➤ MS COCO for object detection

- Default hyper-parameters are as follows: the training steps is 500,500; the step decay learning rate scheduling strategy is adopted with initial learning rate 0.01 and multiply with a factor 0.1 at the 400,000 steps and the 450,000 steps, respectively; The momentum and weight decay are respectively set as 0.9 and 0.0005. All architectures use a single GPU to execute multi-scale training in the batch size of 64 while mini-batch size is 8 or 4 depend on the architectures and GPU memory limitation. Except for using genetic algorithm for hyper-parameter search experiments, all other experiments use default setting. Genetic algorithm used YOLOv3-SPP to train with GloU loss and search 300 epochs for min-val 5k sets. We adopt searched learning rate 0.00261, momentum 0.949, IoU threshold for assigning ground truth 0.213, and loss normalizer 0.07 for genetic algorithm experiments. We have verified a large number of BoF, including 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, different kind of IoU losses. We also conduct experiments on various BoS, including Mish, SPP, SAM, RFB, BiFPN, and Gaussian YOLO [8]. For all experiments, we only use one GPU for training, so techniques such as syncBN that optimizes multiple GPUs are not used.

Experimental : Influence of BoF and Mish on classifier training

- CutMix, Mosaic data augmentation, Label smoothing → improved!
- Mish activation → Good!

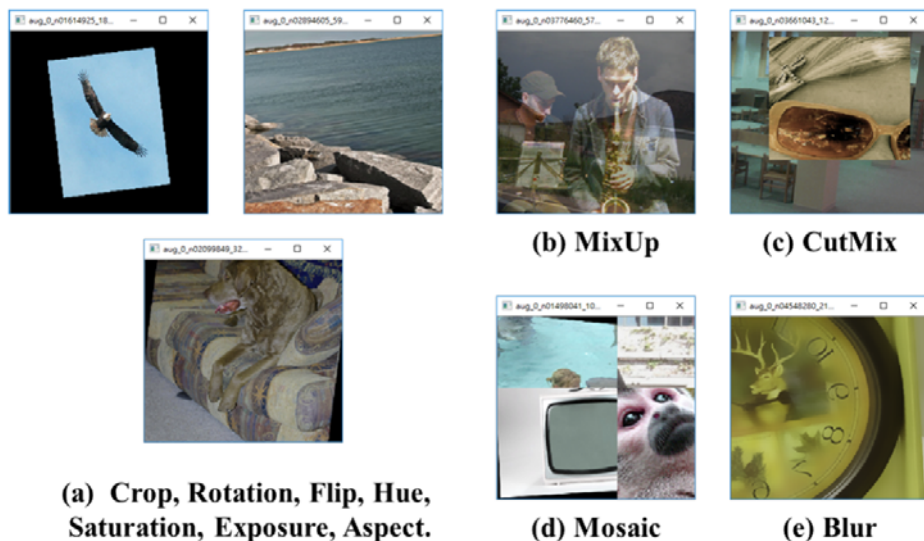


Figure 7: Various method of data augmentation

Mish activation

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

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

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

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

Experimental : Influence of BoF and Mish on object detect training

- 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.
→ Worse performance
- M: Mosaic data augmentation - using the 4-image mosaic during training instead of single image
- IT: IoU threshold - using multiple anchors for a single ground truth IoU (truth, anchor) > IoU_threshold
→ Worse performance
- GA: Genetic algorithms - using genetic algorithms for selecting the optimal hyperparameters during network training on the first 10% of time periods
- LS: Class label smoothing - using class label smoothing for sigmoid activation
→ Worse performance
- CBN: CmBN - using Cross mini-Batch Normalization for collecting statistics inside the entire batch, instead of collecting statistics inside a single mini-batch
- CA: Cosine annealing scheduler - altering the learning rate during sinusoid training
- DM: Dynamic mini-batch size - automatic increase of mini-batch size during small resolution training by using Random training shapes
→ Worse performance
- OA: Optimized Anchors - using the optimized anchors for training with the 512x512 network resolution
- GIoU, CIoU, DIoU, MSE - using different loss algorithms for bounded box regression

Experimental : Influence of **BoF** and **Mish** on object detect training

- PAN + SPP + SAM → Better performance!
- RFP, Gaussian YOLO(G), ASFF → Worse performance

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%

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%

Experimental : Influence of **different backbones** on object detector training

- Although classification accuracy of CSPResNeXt is higher compared to CSPDarknet, CSPDarknet model shows higher accuracy in terms of object detection

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

Experimental : Influence of **different mini-batch size** on object detector training

- After adding BoF and BoS, the mini batch size has almost no effect on the detector's performance

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

Experiments

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%

Table 8 lists the frame rate comparison results of using Maxwell GPU, and it can be GTX Titan X (Maxwell) or Tesla M40 GPU.

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%

Experiments

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%

Table 9 lists the frame rate comparison results of using Pascal GPU, and it can be Titan X (Pascal), Titan Xp, GTX 1080 Ti, or Tesla P100 GPU.

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%

Conclusion

- Offer SOTA detector which is faster (FPS) and more accurate
- YOLOv4 can be trained and used on single conventional GPU with 8 GB VRAM
- Verified a large number of features, and selected for use such of them for improving the accuracy of both the classifier and the detector