Merchandise Detection System
-Hierarchy of yolov4 & implementation observation and comparison with yolov2

0612020 吳峻陞 0612128 林俊德

Part I
-Hierarchy and Comparison

Why we choose these two models?

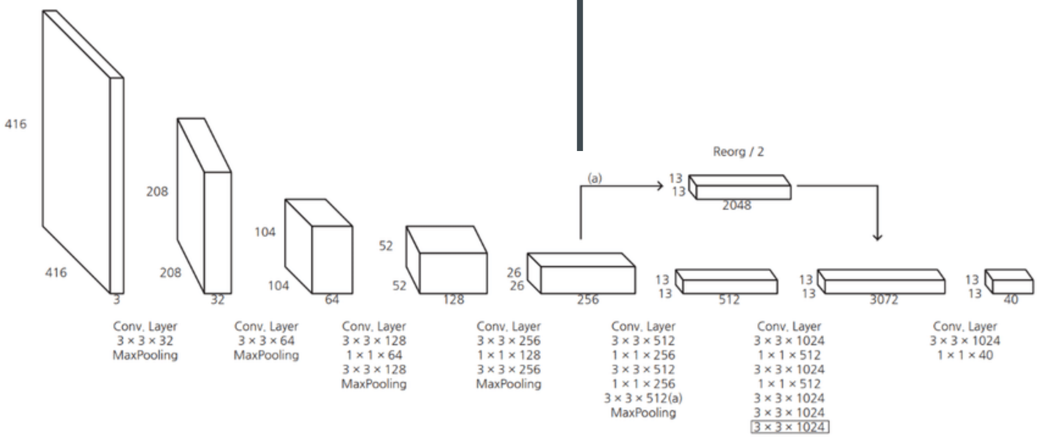
- Most two models have significant performance difference relative to its last one

- Many open resources, like tiny model cfg, coco and voc datasheets

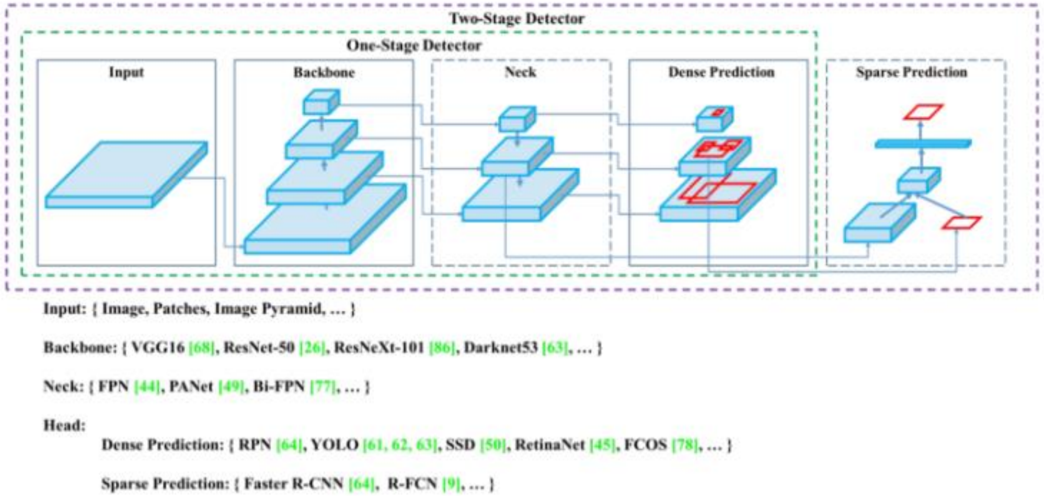
- We use v2 to train firstly, but the performance is unsatisfied. We want to adopt the newcomer yolov4 to fix it and see how much it accelerates

Structural Difference

FastRNN, Batch Normalization

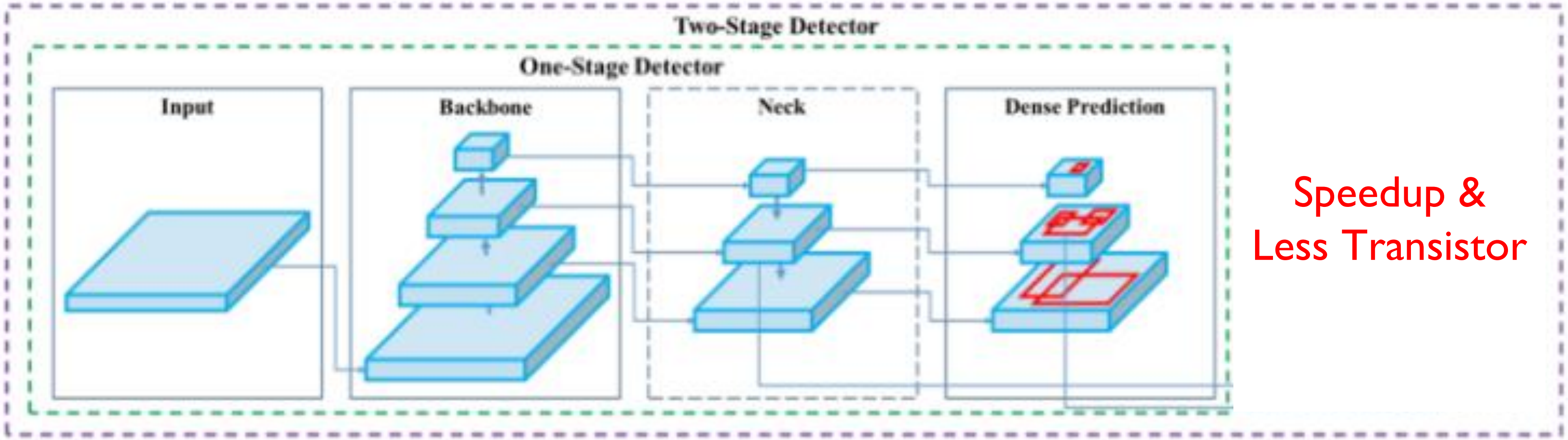


yolov2



yolov4

Well-Performed One-Stage Architecture



Speedup &
Less Transistor

Input: { Image, Patches, Image Pyramid, ... }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], ... }

Neck: { FPN [44], PANet [49], Bi-FPN [77], ... }

Head:

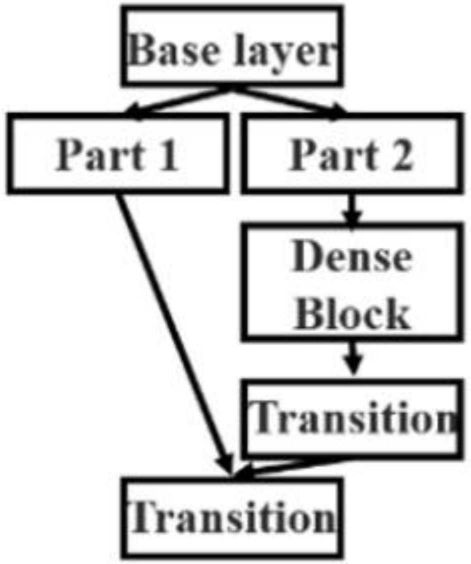
Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], ... }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], ... }

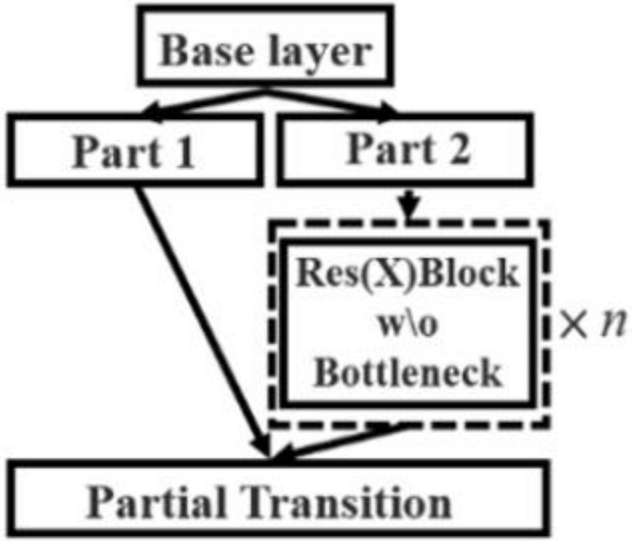
Inside the Architecture

- Backbone : Network, CSP
- Activation : ReLU, Leaky ReLU, Mish
- Dropout : Drop Connect, Drop Path, Special Dropout, DropBlock
- Batch Normalization, Batch Renormalization, CmBN
- Mosaic
- NAS-FPN BiFPN, Modified PAN (Concatenation)
- SPP, ASPP, RFB
- MiWRC
- Attention Module : SE, SAM, Modified SAM, SFAM, ASFF
- Loss : Focal Loss, Loss Smoothing, Grid Sensitivity, LRN
- Loss Function on IOU : GIOU, DIOU, CIOU, DIOU-NMS
- SAT

CSP (Cross Stage Patial Connections)



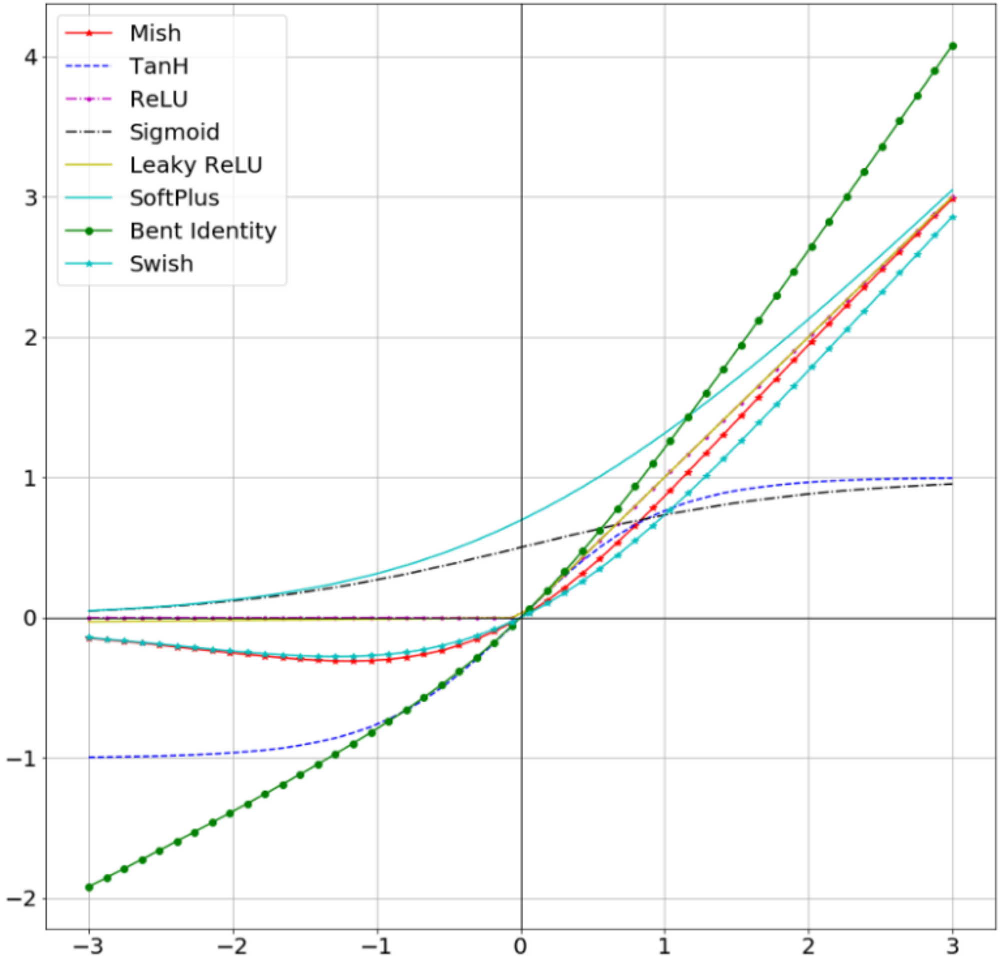
For DenseNet



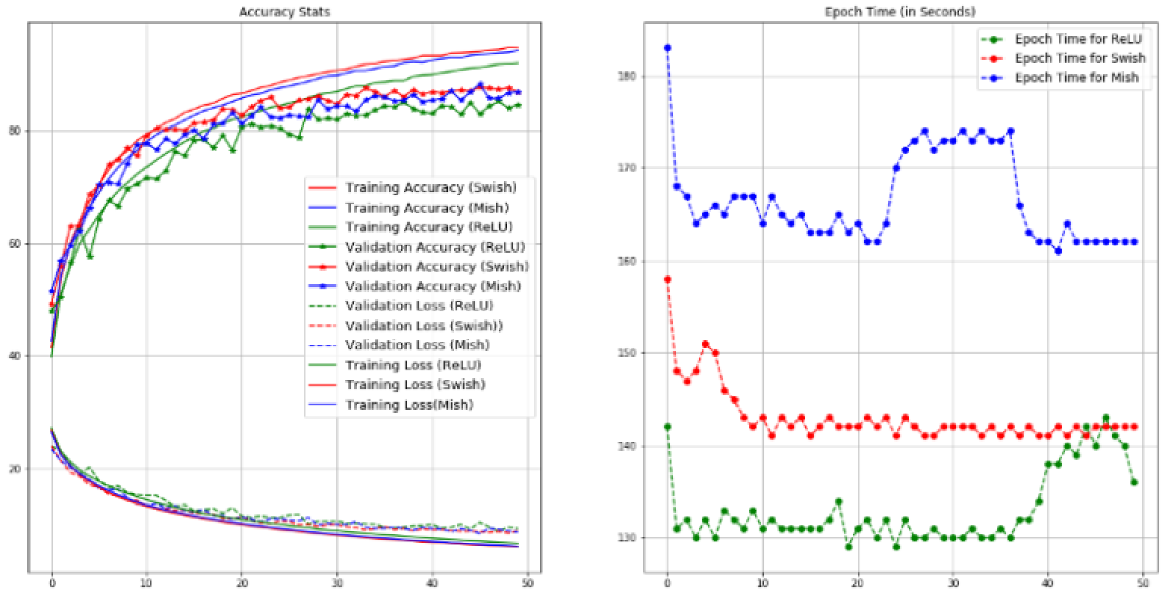
For ResNe(X)t

Split -> Bypass -> Merge

Mish Most Suitable Function to Predict !



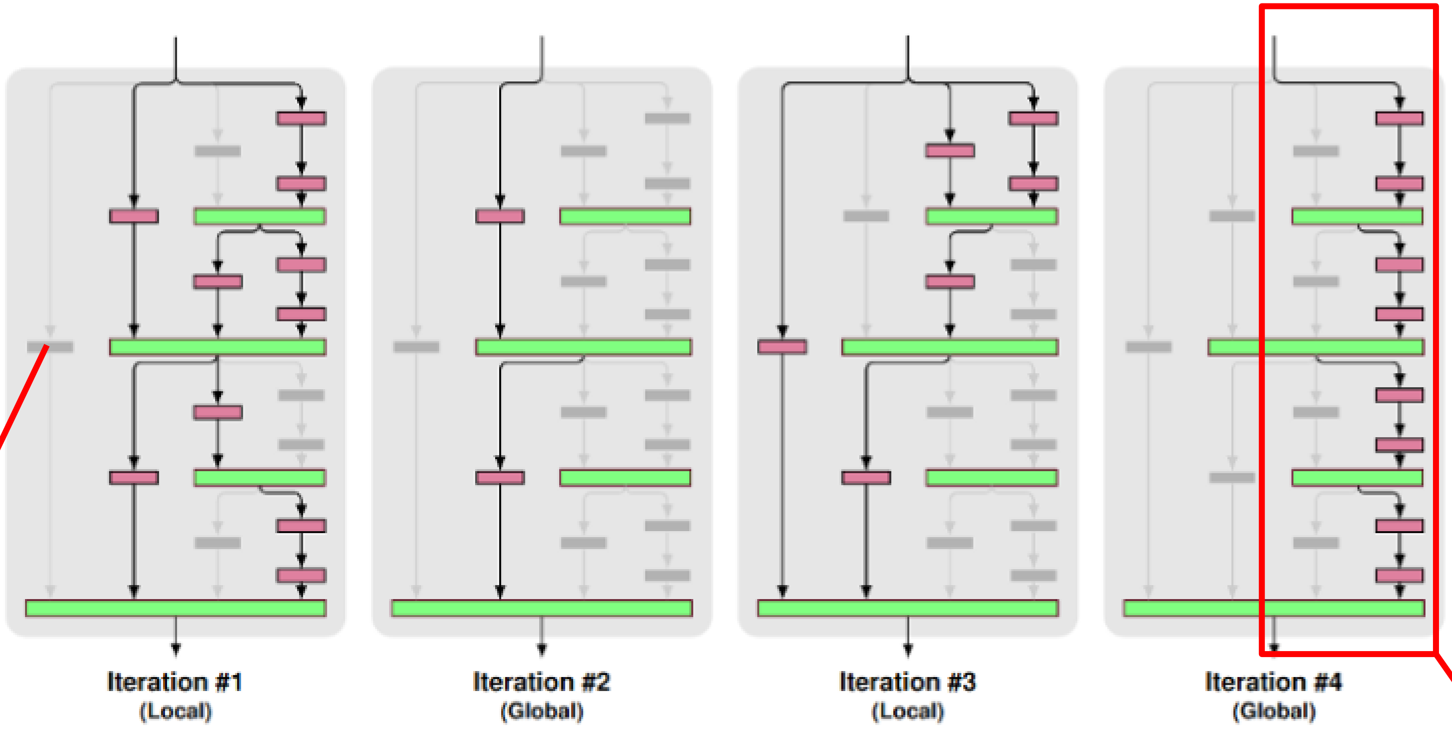
$$f(x) = x \tanh(\ln(1 + e^x))$$



ResNet v2−56 on CIFAR-10 for 50 epochs

Mish has the best performance on ResNet-like training

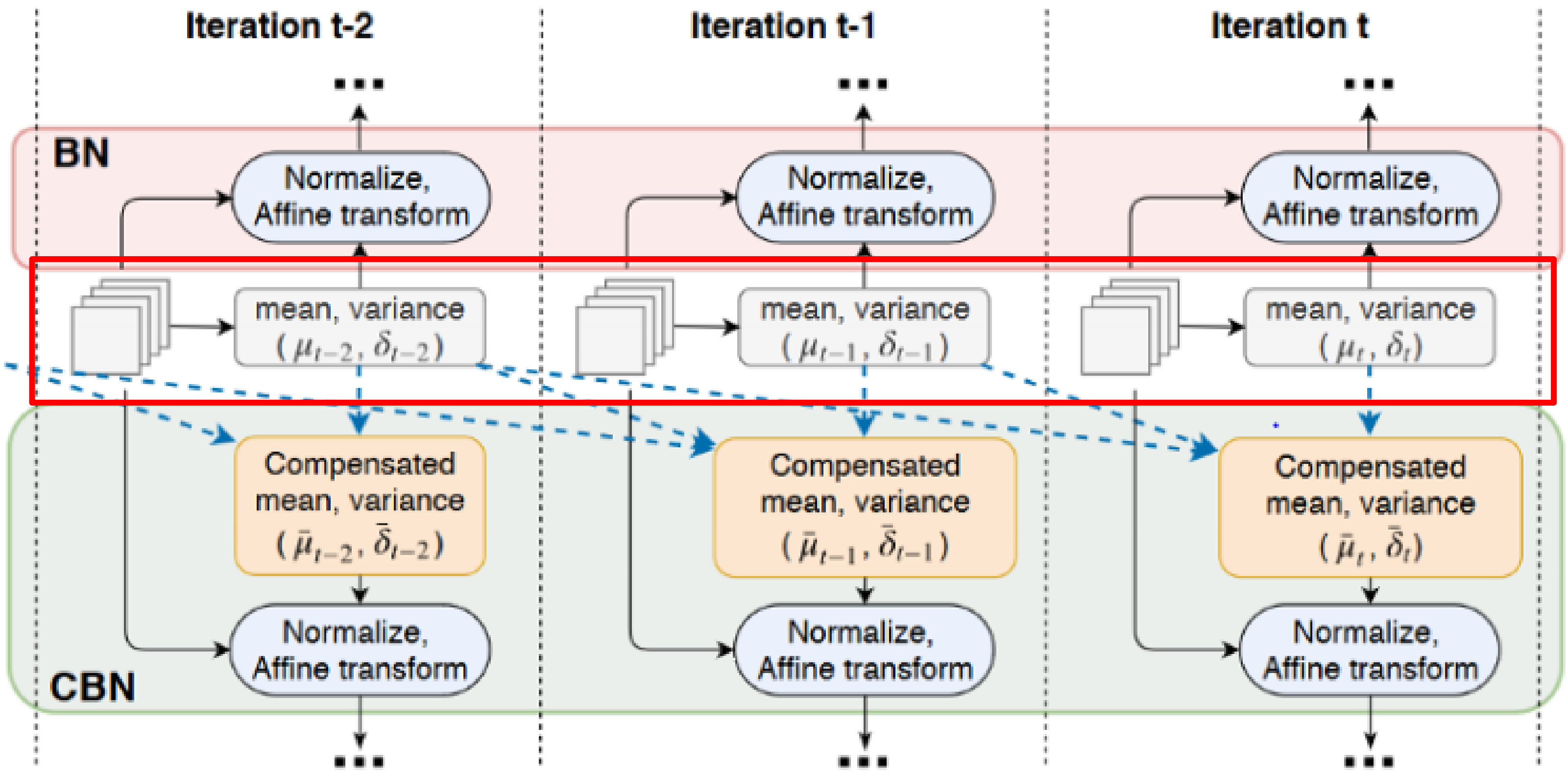
Drop Path



Dropout a whole CNN layer (Local)

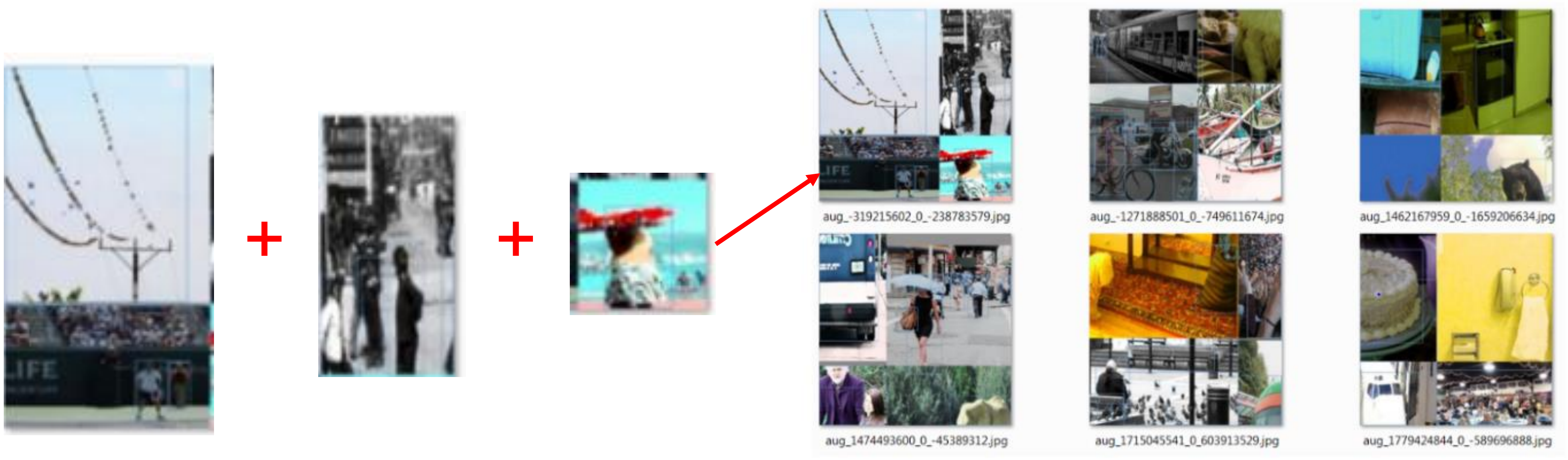
Only train one path to see other paths' function (Global)

CmBN (Cross mini Batch Normalization)



Split into mini-batches and preserve (k-1) iterations to use linear function to predict mean value and deviation

Mosaic



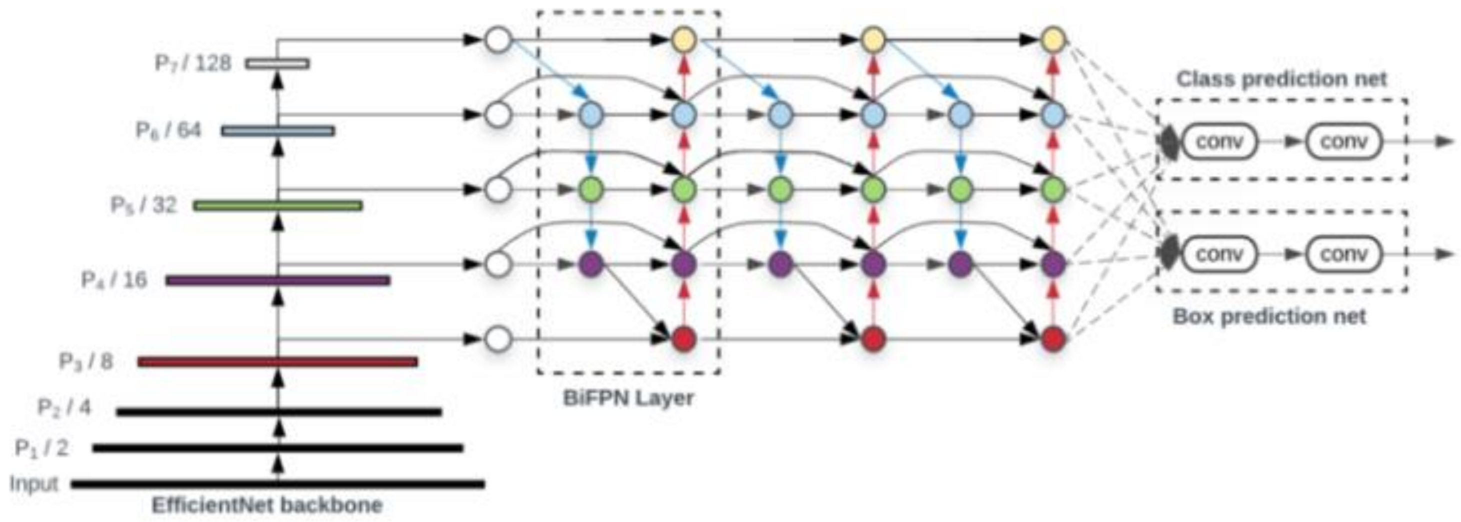
Part of training pictures

Reassembled pictures to train

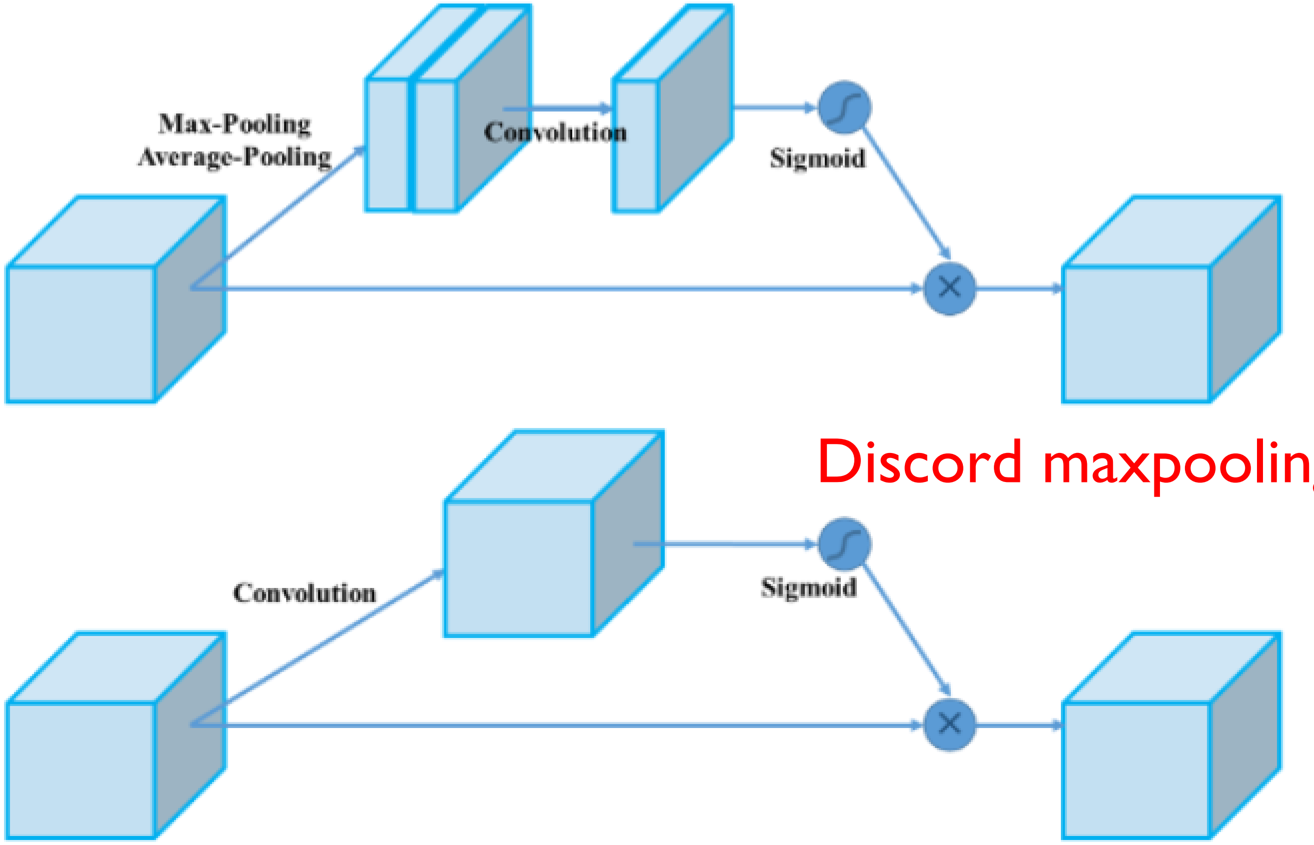
NAS-FPN BiFPN (Deep Feature Pyramid Network)

Stack various size of pictures to form a single feature pyramid network

xN times

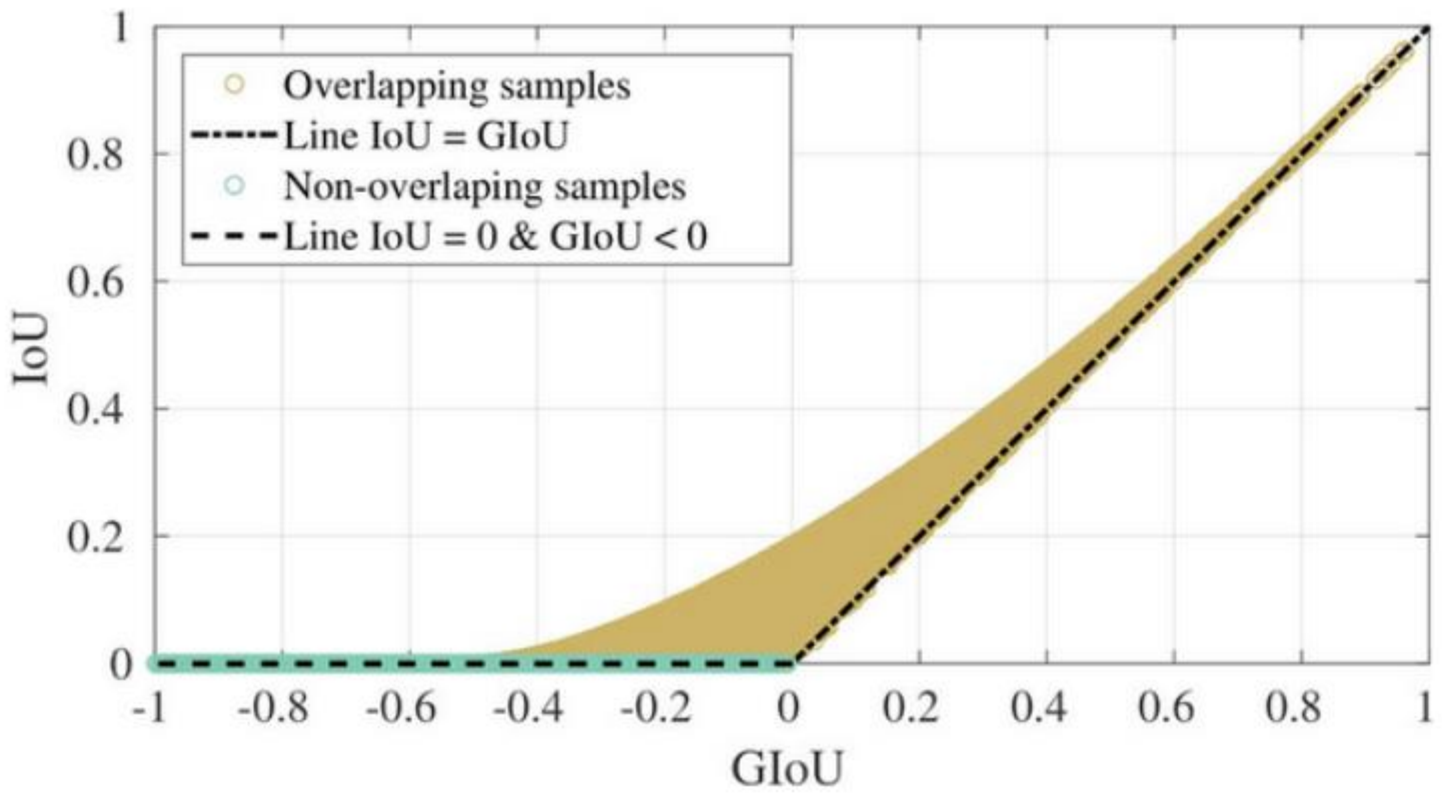


Modified SAM (Spatial Attention Module)



Discord maxpooling lest information lost

CIOU (Complete IoU)

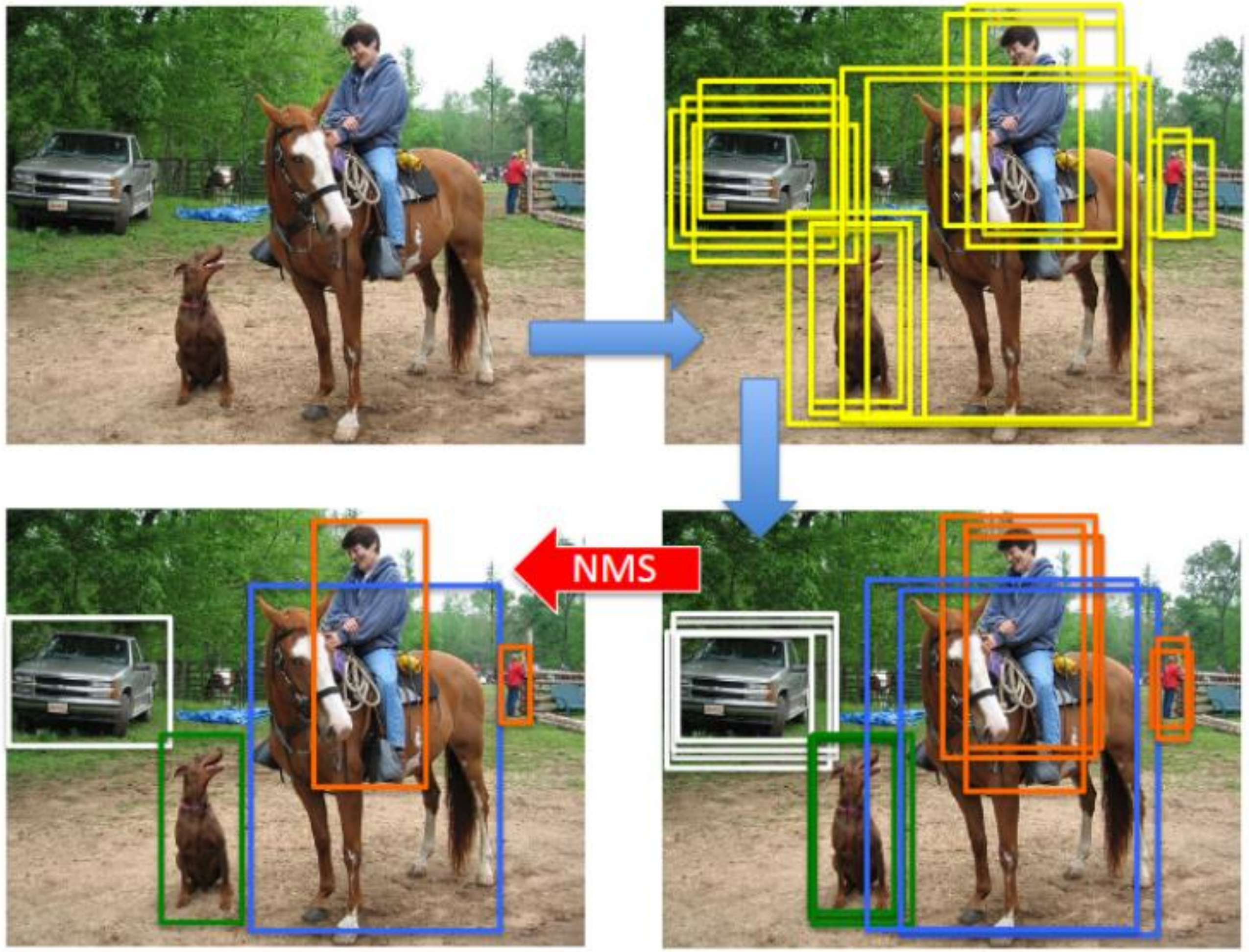


$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v.$$

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

Put height and weight in consideration

DIOU-NMS (Distance IoU- Non-Max Suppression)

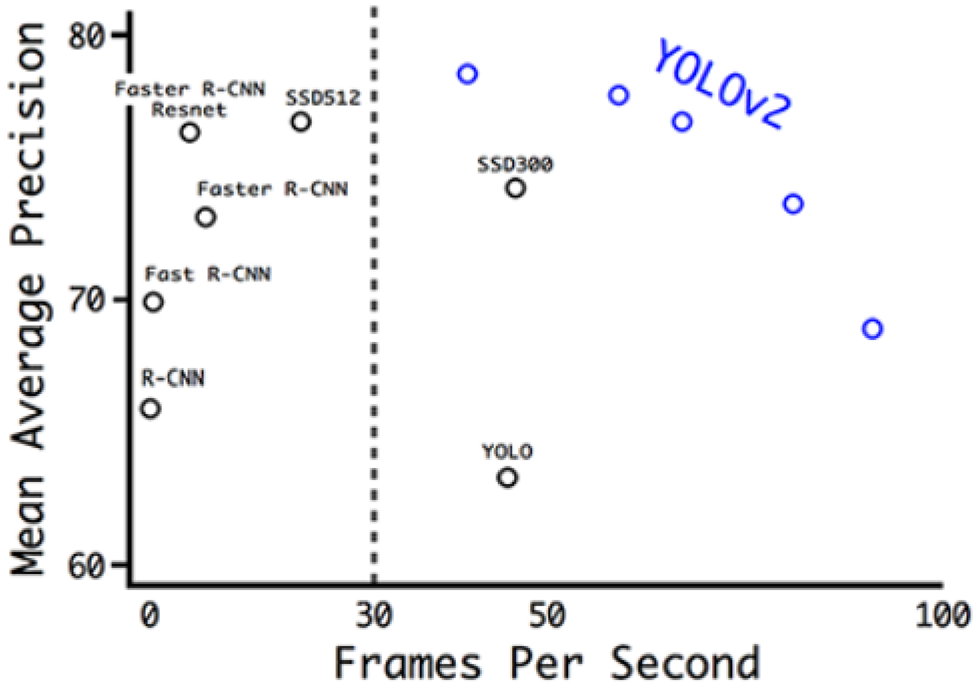


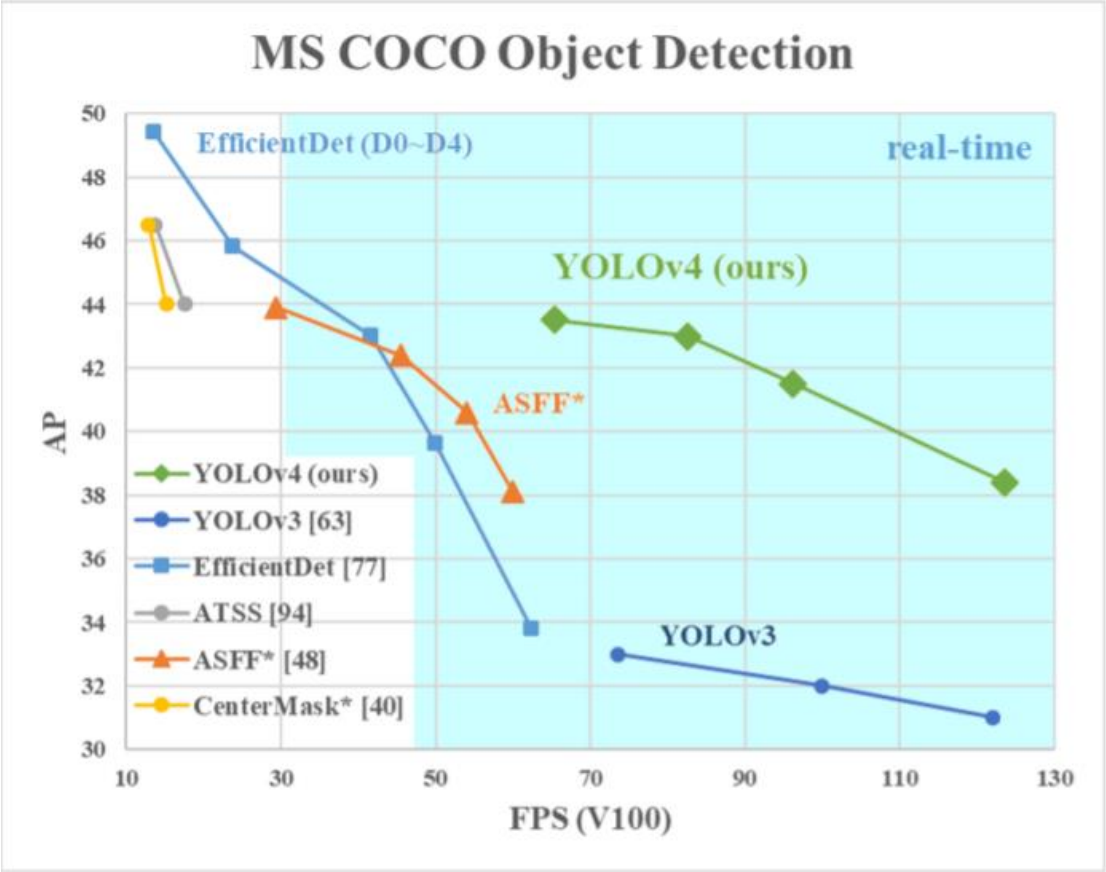
$$\mathcal{L}_{DIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}.$$

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

Consider duplicated bounding box and select one overlapping the target most

Theoretical Performance





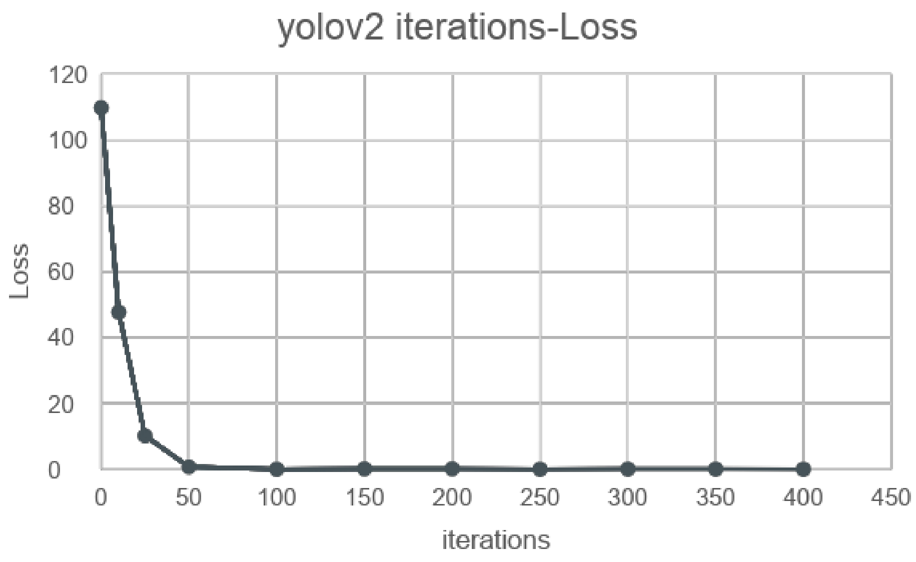
- According to COCO datasheet published in 2018, yolov3 outperforms yolov2 in average 27% mAP (18% and 36% in two kind of FPS conditions)
- In paper of yolov4, it raises 27.6% accuracy versus yolov3 for 416x416
- Based on these two conditions, yolov4 must have generally 62% stronger than yolov2.

Training Data Selection

1. Random background pictures in 3 types, which is nature, market & room, and pure color
2. Random 8 sites for objects in every background picture
3. Same object size
4. 3 labels, 240 pictures each.

Experimental Results (yolov2)

With Batch = 16. OpenCV = 1,GPU = 1,input size = 416x416



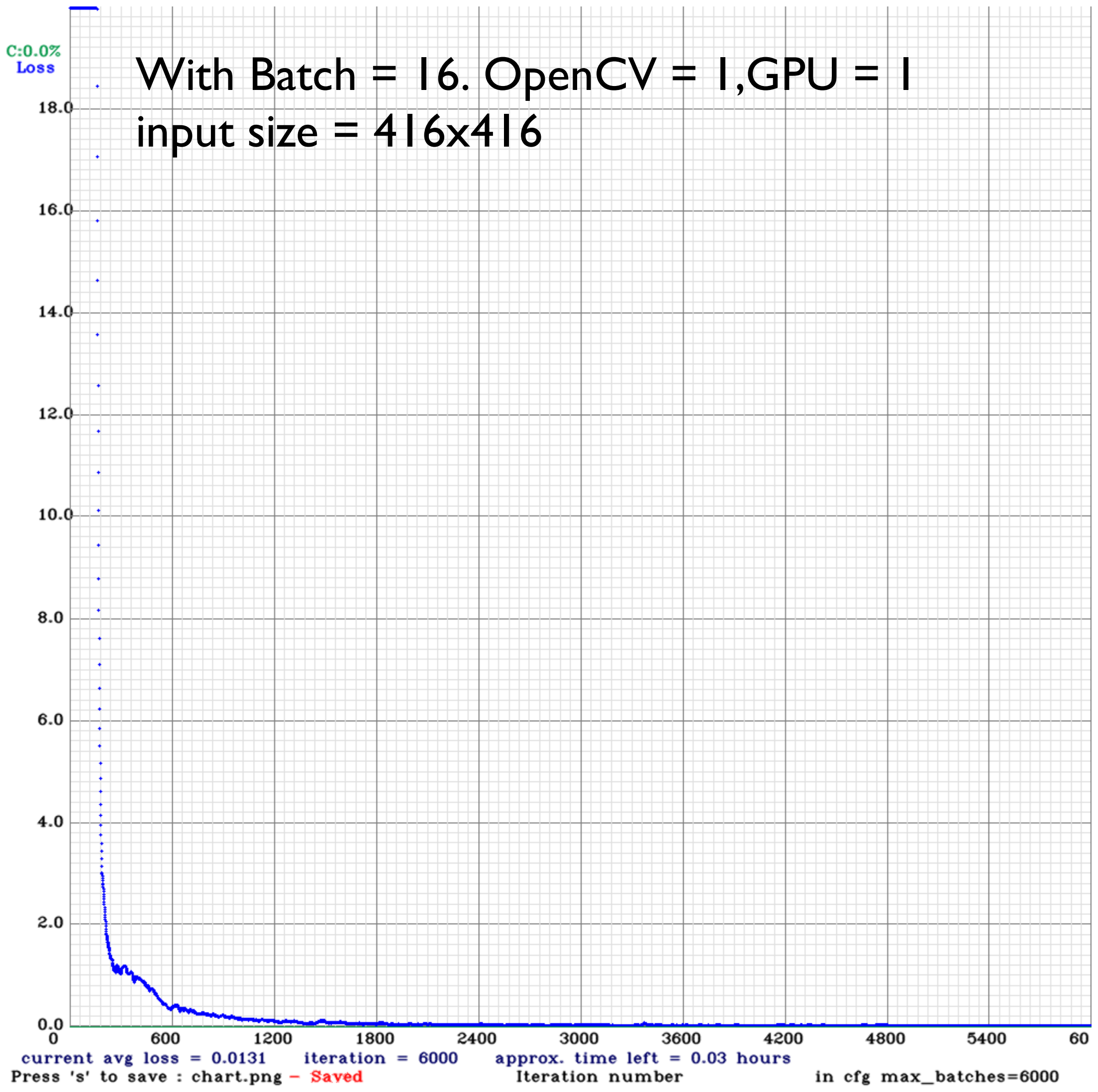
batch	iterations	epoch	loss
16	0	0	109.8
16	150	10	47.72
16	375	25	10.176
16	750	50	0.73
16	1500	100	0.038
16	2250	150	0.1495
16	3000	200	0.1694
16	3750	250	0.036
16	4500	300	0.0727
16	5250	350	0.0714
16	6000	400	0.0395

Overfitting occurs!

Experimental Results (yolov4)

By every 1000 iterations

With Batch = 16. OpenCV = 1,GPU = 1
input size = 416x416




```
for conf_thresh = 0.25, precision = 0.87, recall = 0.90, F1-score = 0.89
for conf_thresh = 0.25, TP = 217, FP = 32, FN = 24, average IoU = 59.11 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.938121, or 93.81 %
Total Detection Time: 3 Seconds
```

```
for conf_thresh = 0.25, precision = 1.00, recall = 1.00, F1-score = 1.00
for conf_thresh = 0.25, TP = 241, FP = 0, FN = 0, average IoU = 88.47 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 1.000000, or 100.00 %
Total Detection Time: 2 Seconds
```

```
for conf_thresh = 0.25, precision = 1.00, recall = 1.00, F1-score = 1.00
for conf_thresh = 0.25, TP = 241, FP = 0, FN = 0, average IoU = 86.37 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 1.000000, or 100.00 %
Total Detection Time: 2 Seconds
```

```
for conf_thresh = 0.25, precision = 1.00, recall = 1.00, F1-score = 1.00
for conf_thresh = 0.25, TP = 241, FP = 0, FN = 0, average IoU = 91.41 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 1.000000, or 100.00 %
Total Detection Time: 2 Seconds
```

```
for conf_thresh = 0.25, precision = 1.00, recall = 1.00, F1-score = 1.00
for conf_thresh = 0.25, TP = 241, FP = 0, FN = 0, average IoU = 94.94 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 1.000000, or 100.00 %
Total Detection Time: 2 Seconds
```

```
for conf_thresh = 0.25, precision = 1.00, recall = 1.00, F1-score = 1.00
for conf_thresh = 0.25, TP = 241, FP = 0, FN = 0, average IoU = 95.22 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 1.000000, or 100.00 %
Total Detection Time: 2 Seconds
```

IoU and mAP comparisons

Yolov2 (by lucky-ing)

IoU @ 2250 iterations : 61.23

mAP @ 2250 iterations : 1.00

Best performance

IoU @ 6000 iterations : 59.67

mAP @ 6000 iterations : 0.98

Yolov4

IoU @ 6000 iterations : 95.22

mAP @ 6000 iterations : 1.00

IoU surpasses 55.5% @ 2250 iterations

59.57% @ 6000 iterations

mAP draws a little impact because of small training database

Possible Reasons of Relatively Small Enhancement

Yolov4 is freaking phenomonal !

Part II
-Implementation

Testing Observations in yolov4

Testing rules : pictures with various size, resolution, and background each label
Hitting standard : with IoU above 75%

	Nature	Restaurant & Room	Pure color
High resolution	16/20	15/20	19/20
Low resolution	14/20	14/20	19/20
Various object size	7/20	7/20	11/20

High contrast and diversed pixel color

Training in pure color obtains better testing performance than others

- Resolution of background doesn’t play a vital role on both testing and training (little effect)

★Yolov4 has the ability to various size of objects

Ability to learn various object sizes

75% kala

76% kala

Use part of feature to predict by feature pyramid deep network

Accidentally Observation

If I put two objects in one background

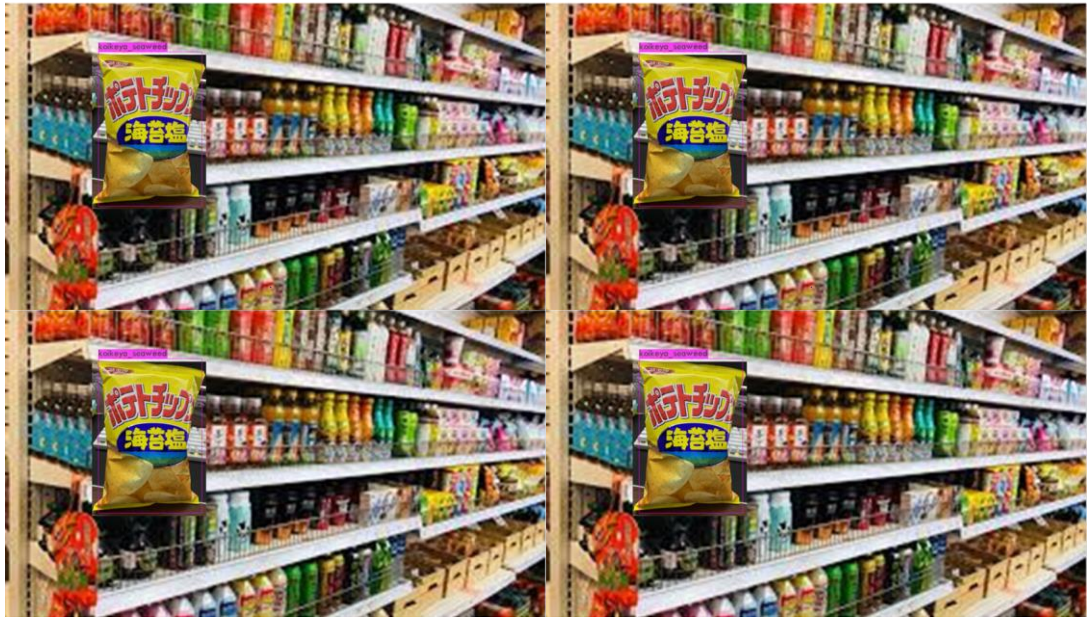
The data becomes…

IoU 68%

IoU 97% and 77%

Shrink input size to 0.25x and compose a new picture

IoU 68%



IoU 78%, 73%, 61%, 60%

IoU 92%

IoU 93%, 90%, 86%, 79%

- Have more data (IoU) to determine label
- Raise feature amount to test with tolerable time consumption

Conclusions

- Yolov4 gets obvious upgrade due to architecture enhancement
- Pure color backgrounds have a great success on testing
- Resolution is trivial than thoughts
- Folded input pictures can make performance better

Thanks
For
Watching

Citations

- https://github.com/AlexeyAB/darknet
- https://github.com/lucky-ing/voc_eval
- https://arxiv.org/abs/2004.10934
- https://towardsdatascience.com/mish-8283934a72df
- https://reurl.cc/oLag1M
- https://reurl.cc/Wd23re
- https://reurl.cc/O1y047