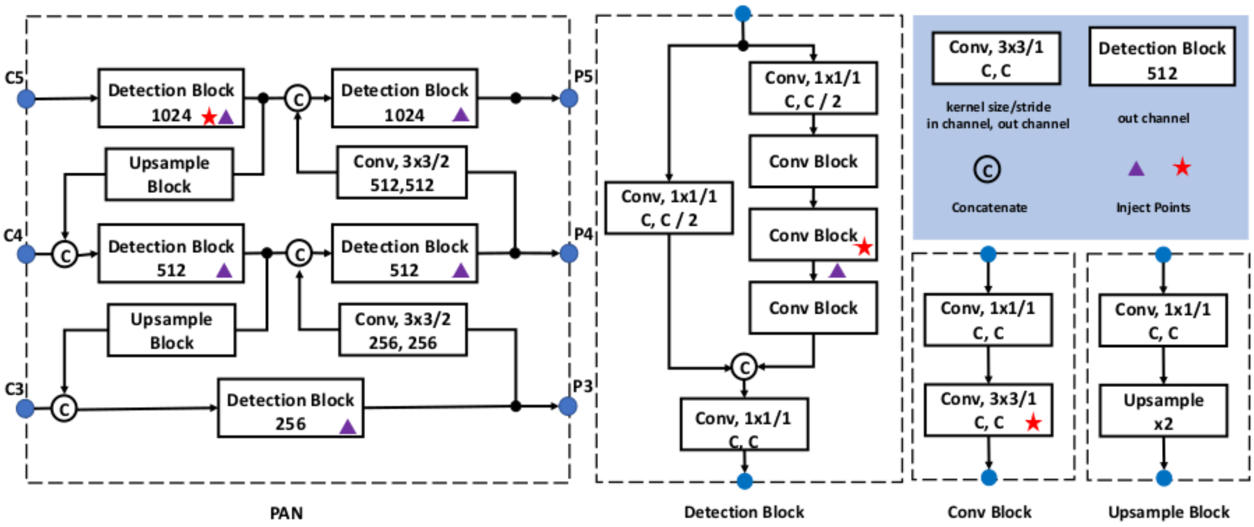
Name: Panneerselvam N
Role : Research Intern – Inspect
Date : 05/06/2021

Paddle Paddle You Only Look Once v2(PP-YOLO v2)

PP-Yolo is a improved version of PP-YOLO v2 and give better performance than Yolov3 and Yolo v4.

Architecture :



Above image shows the detection and neck of PP-Yolo v2.

Backbone :

- In PP-Yolo v1, ResNet-50 as used as Backbone , but in PP-Yolo v2 ResNet-101 is used as Backbone.
- ResNet-101 is pre-trained with COCO dataset.

Neck :

- Path Aggregation Network(PAN) is used as Neck part of PP-Yolo v2. Path Aggregation Network is improved version of PP-Yolov2.

Detection Part:

- Detection part is designed in the structure of Residual Blocks which has 3x3 and 1x1 convolutional blocks to produce detection results.
- The output channel of each final prediction is 3(K + 5), where K is number of classes.

Optimizations :
Apart from PP-Yolo v1 optimizations , v2 has introduced lots of optimization techniques.

Path Aggregation Network:

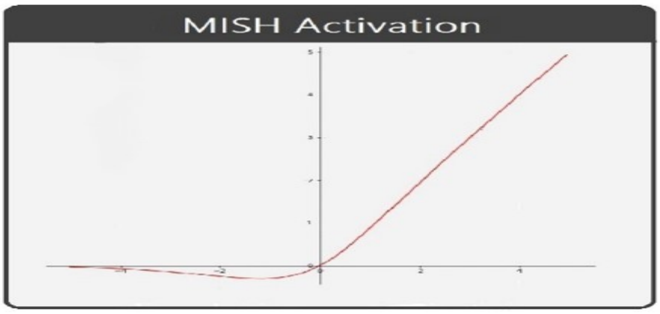
- PP-Yolo v2 uses PAN as Neck part of Architecture. PAN is advanced version Feature Pyramid Network.

- Normal FPN merges the high level features to low level features in Top-Bottom opration. So the features of middel and low level are modified . This givesdifficulties for detect small objects.

- For overcome above problem, the PANet comes with additional layer pyramid of Bottom-top , this gives very efficient extraction low and middle level features.

Mish Activation Function:
In Yolo v4 and v5, Mish activation function gives good results in accuracy, so that PP-Yolo v2 also uses Mish Activation Function.

Mish Activation function is only used in Path Aggregation Network(PAN) in PP-Yolo v2.



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

Large Input Size:

- Increase input size is one of key technique to improve model accuracy. But increasing image size will take large memory space .

- Deceasing batch size is one of the solution for above problem, image size is incrased from 608 to 768 and bactch size is decreased from24 to 12.

IoU Aware Lose:

In PP-Yolo v2 modified IoU Loss is introduced for better result in bounding box prediction.

$$loss = -t * \log(\sigma(p)) - (1-t) * \log(1-\sigma(p)) \qquad (1)$$

Optimization technique that didn’t work:

Author tried some other optimization also, but that didn’t improve performance. The are some of that techniques,

- Cosine Learning Rate Decay
- Backbone Parameter Freezing
- SiLU activation function

Performance :

Author gives some Ablation study regarding optimizations,

	Methods	mAP	Parameters	GFLOPs	infer time	FPS
A	PP-YOLO	45.1	45 M	45.1	13.7 ms†	72.9
B	A + PAN + MISH	47.1	54 M	52.0	14.0 ms	71.4
C	B + input size 640	47.7	54 M	57.6	14.5 ms	68.9
D	C + Larger input size	48.3	54 M	57.6	14.5 ms	68.9
E	D + IoU Aware Branch	49.1	54 M	57.6	14.5 ms	68.9

Method	Backbone	Size	FPS (V100)		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
			w/o TRT	with TRT						
YOLOv3 + ASFF* [13]	Darknet-53	320	60	-	38.1%	57.4%	42.1%	16.1%	41.6%	53.6%
YOLOv3 + ASFF* [13]	Darknet-53	416	54	-	40.6%	60.6%	45.1%	20.3%	44.2%	54.1%
YOLOv3 + ASFF* [13]	Darknet-53	608	45.5	-	42.4%	63.0%	47.4%	25.5%	45.7%	52.3%
YOLOv3 + ASFF* [13]	Darknet-53	800	29.4	-	43.9%	64.1%	49.2%	27.0%	46.6%	53.4%
EfficientDet-D0 [24]	Efficient-B0	512	98.0+	-	33.8%	52.2%	35.8%	12.0%	38.3%	51.2%
EfficientDet-D1 [24]	Efficient-B1	640	74.1+	-	39.6%	58.6%	42.3%	17.9%	44.3%	56.0%
EfficientDet-D2 [24]	Efficient-B2	768	56.5+	-	43.0%	62.3%	46.2%	22.5%	47.0%	58.4%
EfficientDet-D2 [24]	Efficient-B3	896	34.5+	-	45.8%	65.0%	49.3%	26.6%	49.4%	59.8%
YOLOv4 [1]	CSPDarknet-53	416	96	164.0*	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4 [1]	CSPDarknet-53	512	83	138.4*	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4 [1]	CSPDarknet-53	608	62	105.5*	43.5%	65.7%	47.3%	26.7%	46.7%	53.3%
YOLOv4-CSP [26]	Modified CSPDarknet53	512	97	-	46.2%	64.8%	50.2%	24.6%	50.4%	61.9%
YOLOv4-CSP [26]	Modified CSPDarknet53	640	73	-	47.5%	66.2%	51.7%	28.2%	51.2%	59.8%
YOLOv5s	-	640	113*	-	36.7%	55.4%	-	-	-	-
YOLOv5m	-	640	88.2*	-	44.5%	63.1%	-	-	-	-
YOLOv5l	-	640	69.8*	-	48.2%	66.9%	-	-	-	-
YOLOv5x	-	640	43.4*	-	50.4%	68.8%	-	-	-	-
PP-YOLO [16]	ResNet50-vd-dcn	320	132.2†	242.2†	39.3%	59.3%	42.7%	16.7%	41.4%	57.8%
PP-YOLO [16]	ResNet50-vd-dcn	416	109.6†	215.4†	42.5%	62.8%	46.5%	21.2%	45.2%	58.2%
PP-YOLO [16]	ResNet50-vd-dcn	512	89.9†	188.4†	44.4%	64.6%	48.8%	24.4%	47.1%	58.2%
PP-YOLO [16]	ResNet50-vd-dcn	608	72.9†	155.6†	45.9%	65.2%	49.9%	26.3%	47.8%	57.2%
PP-YOLOv2	ResNet50-vd-dcn	320	123.3	152.9	43.1%	61.7%	46.5%	19.7%	46.3%	61.8%
PP-YOLOv2	ResNet50-vd-dcn	416	102	145.1	46.3%	65.1%	50.3%	23.9%	50.2%	62.2%
PP-YOLOv2	ResNet50-vd-dcn	512	93.4	141.2	48.2%	67.1%	52.7%	27.7%	52.1%	62.1%
PP-YOLOv2	ResNet50-vd-dcn	608	72.1	109.9	49.2%	68.0%	54.1%	29.9%	52.8%	61.5%
PP-YOLOv2	ResNet50-vd-dcn	640	68.9	106.5	49.5%	68.2%	54.4%	30.7%	52.9%	61.2%
PP-YOLOv2	ResNet101-vd-dcn	512	69.8	116.8	49.0%	67.8%	53.8%	28.7%	53.0%	63.5%
PP-YOLOv2	ResNet101-vd-dcn	640	50.3	87.0	50.3%	69.0%	55.3%	31.6%	53.9%	62.4%

Table 2. Comparison of the speed and accuracy of different object detectors on the MS-COCO (test-dev 2017). We compare the results with batch size = 1, without tensorRT (w/o TRT) or with tensorRT(with TRT). Results marked by "+" are updated results from the corresponding official code base. Results marked by "*" are test in our environment using official code and model. "†" indicates the result includes bounding box decode time(1~2ms). The backbone of YOLOv5 has not been named yet, so we leave it blank.

Conclusion :

Finally PP-YOLOv2 runs in 68.9FPS at 640x640 input size. Paddle inference engine with TensorRT, FP16-precision and batch size = 1 further improves PP-YOLOv2’s infer speed, which achieves 106.5 FPS. MAP also gives as 45.5.