Blood Cell Detection with YOLO

10801128 陳俊鴻

Implementation Details

PL	Python 3.8
API	Tensorflow 2.4.0
Packages	os, numpy, matplotlib, parser-libraries3.6, bs4 4.10.1, tf2_YOLO
GPU	M1
Github	None

Dataset

Name	BCCD Dataset
Content	Photos of red blood cell, white blood cell, and platelets
Source	https://public.roboflow.com/object-detection/bccd
Size	640x480 pixels, channel number=3
Classes	RBC, WBC, platelet
Distribution	Train: 50, valid: 50

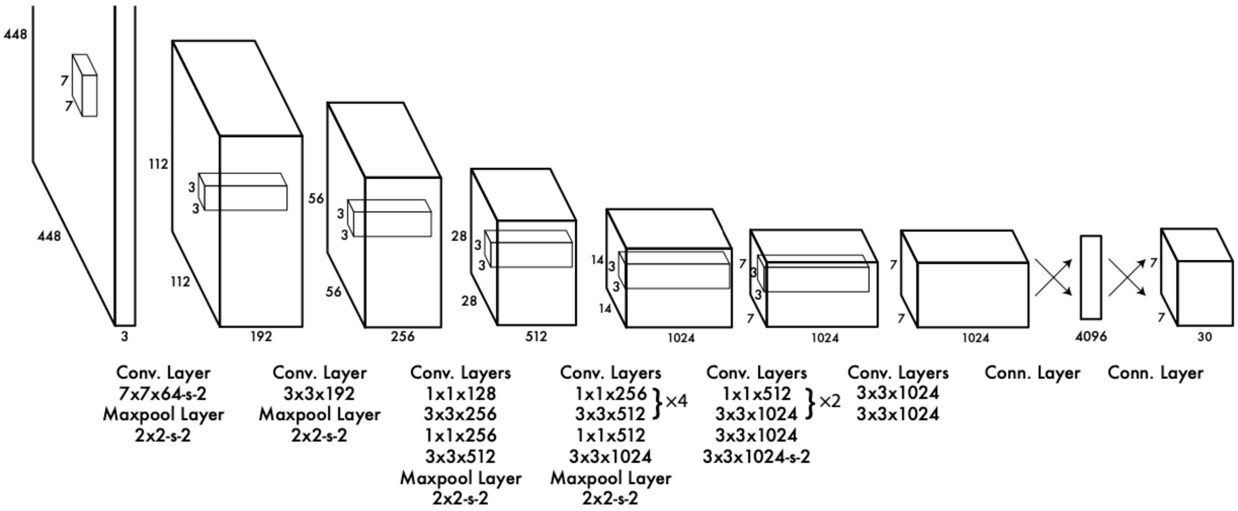
Experimental Design

You only look once(YOLO) is a model for object detection. Different from traditional object detection, YOLO conducting recognition of class and box(classification and regression) simultaneously to improve efficiency but trade accuracy off, while compared to R-CNN.

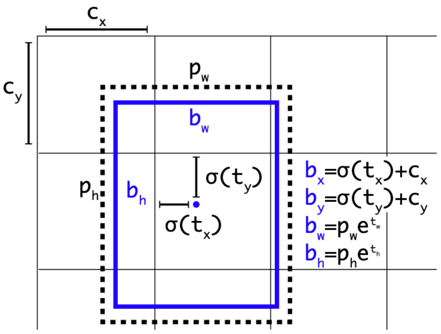
YOLOv2 is improved from YOLO and trained by VOC, COCO, and ImageNet datasets; hence, number of predicted classes increased over 9000, the efficiency and accuracy also surged. Furthermore, YOLOv2 added pre-set anchor box avoiding unnecessary computation and increase recall significantly. Much more modification are shown in the following figures.

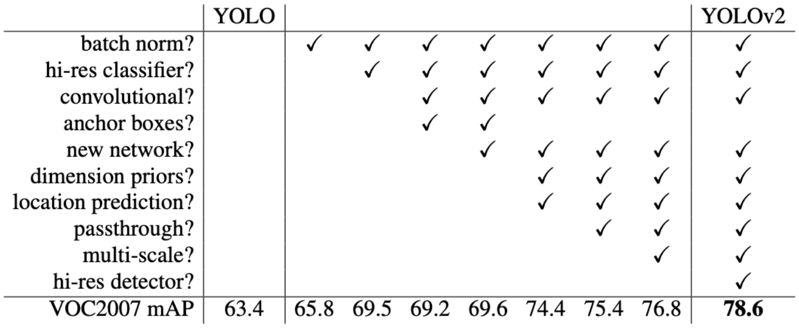
YOLOv3 is improved from YOLOv2 but only changed architecture to DarkNet-53 and added a full connection layer.

YOLO

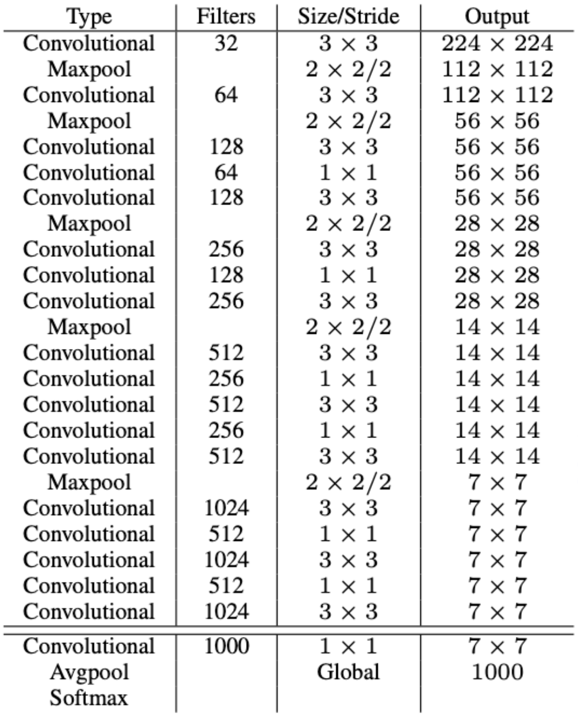


YOLOv2

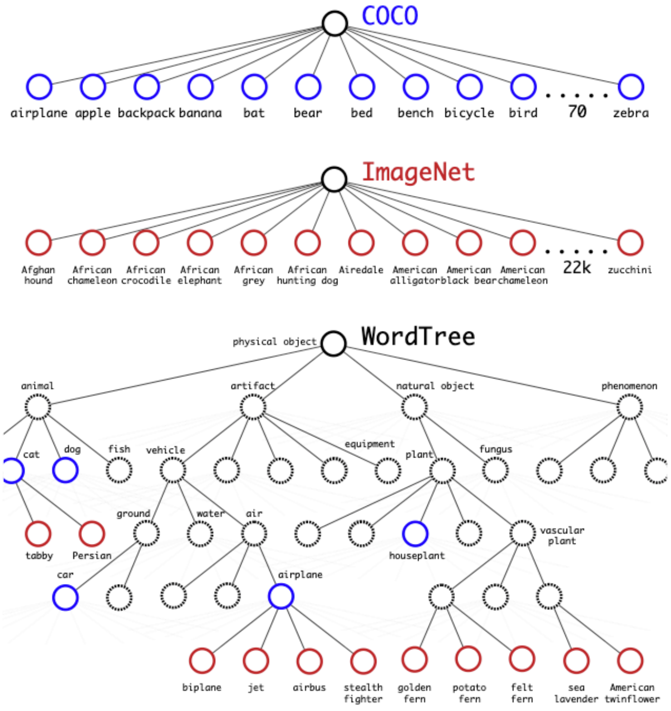




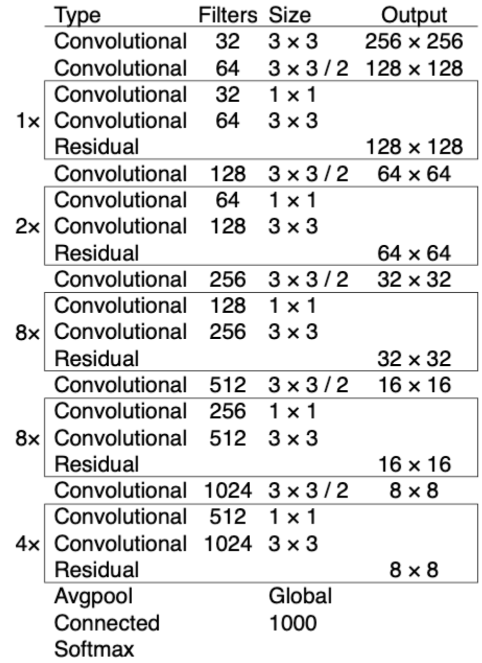
	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**



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



YOLOv3

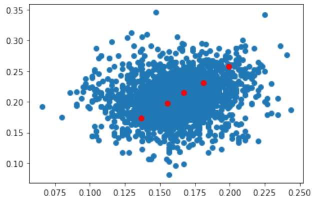
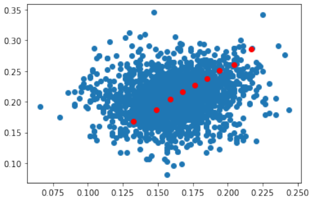


	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3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

Modified Parameters

To have an insight into the cross-effect of input size and loss function to U-Net, I implemented the following four trials altering input size and loss function. Otherwise, with the interest in the performance of DeepLabv3, it also be tried to compare with U-Net.

	YOLOv1	YOLOv2	YOLOv3
Epoch	100	100	100
Batch size	5	5	5
n_cluster	N/A	5	9
stop_dist	N/A	1E-05	1E-05
Optimizer	Adam	Adam	Adam
Learning rate	1E-04	1E-04	1E-04
binary_weight	0.27757511	0.6834027	[0.06834027], [0.01627721], [0.00402176]
ignore_thresh	N/A	N/A	0.7
loss_weight-xy	5	1	5
loss_weight-wh	5	1	5

loss_weight-conf	1	5	1
loss_weight-prob	1	1	1
conf_threshold	0.3	0.3	0.5
nms_mode	N/A	N/A	2
nms_threshold	0.5	0.5	0.5
Anchor	N/A	[[0.19903696, 0.25725892], [0.18114652, 0.23104765], [0.16697793, 0.21561809], [0.1551797 , 0.19698958], [0.13641098, 0.17319174]]	[[0.21696429, 0.2863095], [0.20478724, 0.2611259], [0.1937181 , 0.25125426], [0.1853782 , 0.23777226], [0.17648502, 0.22706038], [0.16771634, 0.2157492], [0.15889142, 0.20403494], [0.14906909, 0.18735233], [0.13243665, 0.16764748]]
			

Results

Architecture	n_cluster	thread_num(train, valid)	Precision	Recall	F1-score
YOLOv1			0.0716845	0.830275	0.131975
YOLOv2	5	50, 50	0.709855	0.847737	0.772693
		50, 10	0.721783	0.826337	0.770530
	9	50, 50	0.644133	0.831276	0.725835
		50, 10	0.613897	0.705350	0.656453
YOLOv3	9	50, 50	0.447563	0.941756	0.606765
		50, 10	0.77038	0.930271	0.842809
	6	50, 50	0.761462	0.940115	0.841410
		50, 10	0.757655	0.954061	0.844590

Reference

1. https://github.com/samson6460/tf2_YOLO.git
2. YOLOv1: You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/abs/1506.02640
3. YOLOv2(YOLO9000): Better, Faster, Stronger. https://arxiv.org/abs/1612.08242
4. YOLOv3: An Incremental Improvement. https://arxiv.org/abs/1804.02767