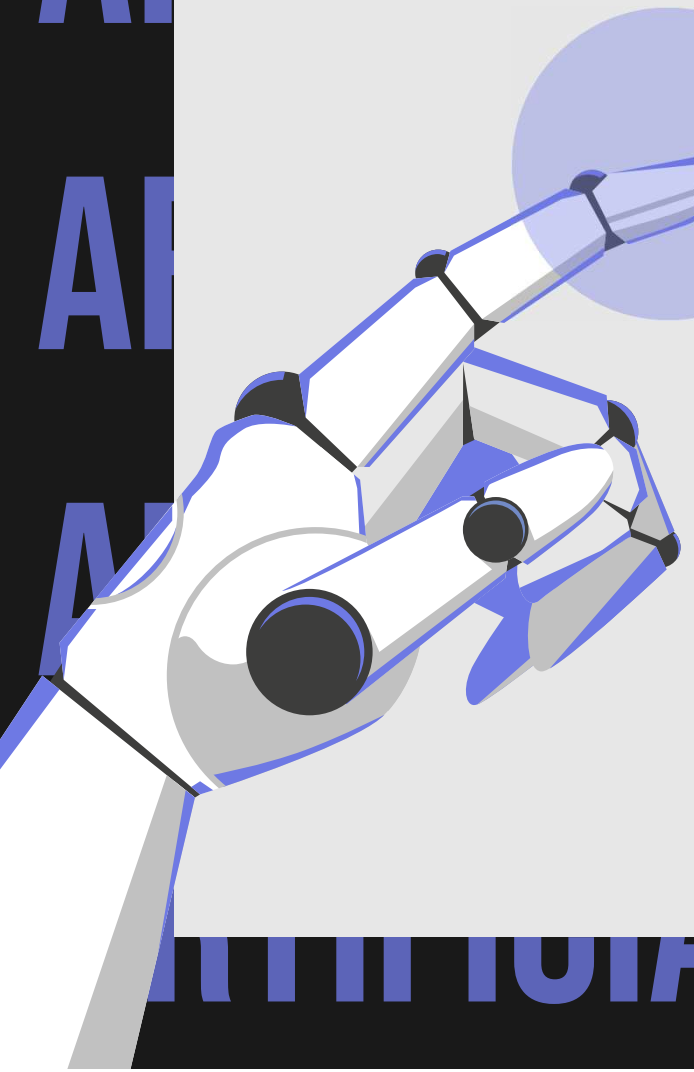


YOLO V2

학부연구생 2단계 과정 세미나

18011816 김동영

TABLE OF CONTENTS

ARTIFICIAL INTELLIGENCE (AI)

01.

INTRODUCTION

YOLO 9000 & YOLO v2

INTRODUCTION

We propose a new method to harness the large amount of classification data we already have and use it to expand the scope of current detection systems. Our method uses a hierarchical view of object classification that allows us to combine distinct datasets together.

We also propose a joint training algorithm that allows us to train object detectors on both detection and classification data. Our method leverages labeled detection images to learn to precisely localize objects while it uses classification images to increase its vocabulary and robustness.

Using this method we train YOLO9000, a real-time object detector that can detect over 9000 different object categories. First we improve upon the base YOLO detection system to produce YOLOv2, a state-of-the-art, real-time detector. Then we use our dataset combination method and joint training algorithm to train a model on more than 9000 classes from ImageNet as well as detection data from COCO.

전에는 Detection과 Classification을 따로 진행함

함께 학습시키는 방법을 제안

이를 통해 Detection Dataset에 존재하지 않는 Label들에 대해 예측이 가능, 여전히 빠른 실행 속도를 유지, 9000개 이상의 Classes에 대해 예측

/[AI]

02.

BETTER

Ideas to improve Performance

BETTER

Batch Normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization [7]. By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

High Resolution Classifier. All state-of-the-art detection methods use classifier pre-trained on ImageNet [16]. Starting with AlexNet most classifiers operate on input images smaller than 256×256 [8]. The original YOLO trains the classifier network at 224×224 and increases the resolution to 448 for detection. This means the network has to simultaneously switch to learning object detection and adjust to the new input resolution.

For YOLOv2 we first fine tune the classification network at the full 448×448 resolution for 10 epochs on ImageNet. This gives the network time to adjust its filters to work better on higher resolution input. We then fine tune the resulting network on detection. This high resolution classification network gives us an increase of almost 4% mAP.

Batch Normalization. 다른 Regularization의 필요성을 없앰, 신경망을 더 빠르게 수렴하도록 함

mAP를 2% Improve, Dropout 제거.

High Resolution Classifier.

기존 -> 224x224를 classifier network 학습 후 448x448로 detection network를 학습

문제점 -> object detection을 학습하면서 동시에 새로운 해상도에 적응해야 함

해결 -> ImageNet dataset에서 448x448 이미지로 fine tuning 함 -> mAP를 대략 4%정도 향상시킴

BETTER

We remove the fully connected layers from YOLO and use anchor boxes to predict bounding boxes. First we eliminate one pooling layer to make the output of the network's convolutional layers higher resolution. We also shrink the network to operate on 416 input images instead of 448×448. We do this because we want an odd number of locations in our feature map so there is a single center cell. Objects, especially large objects, tend to occupy the center of the image so it's good to have a single location right at the center to predict these objects instead of four locations that are all nearby. YOLO's convolutional layers downsample the image by a factor of 32 so by using an input image of 416 we get an output feature map of 13×13.

When we move to anchor boxes we also decouple the class prediction mechanism from the spatial location and instead predict class and objectness for every anchor box. Following YOLO, the objectness prediction still predicts the IOU of the ground truth and the proposed box and the class predictions predict the conditional probability of that class given that there is an object.

Using anchor boxes we get a small decrease in accuracy. YOLO only predicts 98 boxes per image but with anchor boxes our model predicts more than a thousand. Without anchor boxes our intermediate model gets 69.5 mAP with a recall of 81%. With anchor boxes our model gets 69.2 mAP with a recall of 88%. Even though the mAP decreases, the increase in recall means that our model has more room to improve.

Convolutional With Anchor Boxes.

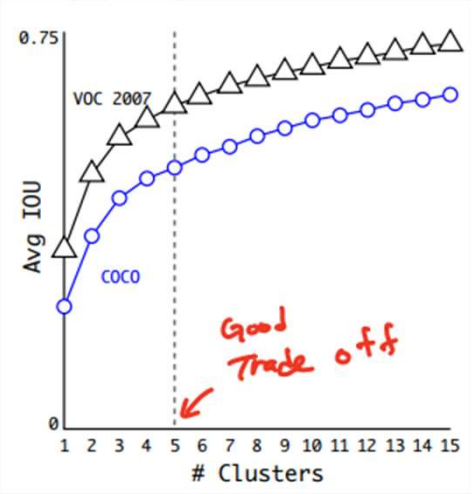
바운딩 박스의 좌표를 예측하는 것보다, 사전에 정의한 Anchor box에서 offset을 예측하는 것이 훨씬 간단.

(1) 한 개의 pooling layer를 제거 -> convolutional layer의 출력을 고 해상도로 만듦

(2) 입력 이미지 크기를 448에서 416으로 변경
416을 32배로 down sampling -> 13x13의 feature map
물체가 이미지의 중앙에 있는 경우가 많아, 최종 output feature map을 홀수x홀수로 지정해주는 것이 좋음

결과 -> mAP는 줄었지만 Recall이 증가해 개선 가능

BETTER



Dimension Clusters.

문제점 ->신경망은 Anchor box를 수정하면서 바운딩 박스를 예측함. 사전에 설정된 anchor box의 영향이 큼

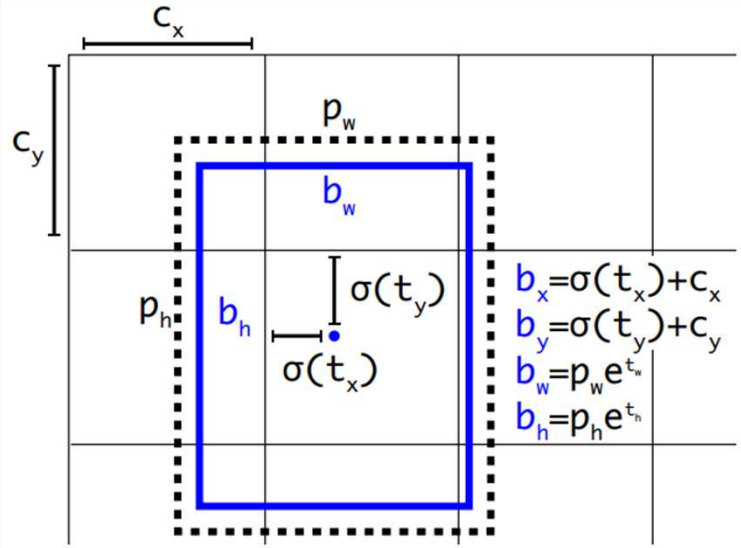
해결 -> training set의 바운딩 박스들에 k-means clustering을 사용하여 Anchor box를 가져옴
높은 k를 선택하면 높은 정확도를 얻을 수 있지만, 속도가 느려짐. 시간과 정확도를 어느정도 챙기는 5를 선택

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

hand-picked anchor box와 clustering 방법 비교 표

clustering 방법이 더 좋음 (60.9 <-> 67.2)

BETTER



c_x : 좌측 상단에서 grid cell의 x좌표
c_y : 좌측 상단에서 grid cell의 y좌표
p_w : 이전 바운딩 박스의 너비
p_h : 이전 바운딩 박스의 높이

Direct location prediction.

문제점 -> anchor box를 활용한 바운딩 박스 offset 예측법은 예측 값의 범위를 제한하지 않아 바운딩 박스가 이미지 어디에나 나타날 수 있어 불안정함

해결 -> sigmoid 함수를 활용하여 offset의 범위를 0에서 1로 제한함. 바운딩 박스 중심 좌표 예측값(tx,ty)을 시그모이드 함수로 감싸 바운딩박스의 위치를 제한하여 안정적으로 학습

결과 -> 성능 5% 향상

BETTER

Fine-Grained Features. This modified YOLO predicts detections on a 13×13 feature map. While this is sufficient for large objects, it may benefit from finer grained features for localizing smaller objects. Faster R-CNN and SSD both run their proposal networks at various feature maps in the network to get a range of resolutions. We take a different approach, simply adding a passthrough layer that brings features from an earlier layer at 26×26 resolution.

The passthrough layer concatenates the higher resolution features with the low resolution features by stacking adjacent features into different channels instead of spatial locations, similar to the identity mappings in ResNet. This turns the $26 \times 26 \times 512$ feature map into a $13 \times 13 \times 2048$ feature map, which can be concatenated with the original features. Our detector runs on top of this expanded feature map so that it has access to fine grained features. This gives a modest 1% performance increase.

Fine-Grained Features.

문제점 -> 13x13 feature map은 작은 물체 검출에 불충분

해결 -> passthrough layer를 추가하여 이전 layer의 26x26 feature map을 가져옴. 26x26x512의 feature map을 13x13x(512*4)의 feature map으로 변환하여 13x13 feature map에 이어 붙임. 26x26 크기의 feature map에 고 해상도 특징이 담겨 있어 작은 물체도 검출 잘됨.

결과 -> 성능 1% 향상

BETTER

Multi-Scale Training. The original YOLO uses an input resolution of 448×448. With the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size. Since our model downsamples by a factor of 32, we pull from the following multiples of 32: $\{320, 352, ..., 608\}$. Thus the smallest option is 320×320 and the largest is 608×608. We resize the network to that dimension and continue training.

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

Multi-Scale Training.

다른 크기의 이미지로부터 robust를 갖기 위해
매 10epoch마다 {320, 352, ..., 608} 크기로 학습함.

Input size에 32배 down sampling하기 때문에
간격을 32로 설정 -> 다양한 입력 크기에도 예측을
잘할 수 있음.

입력 크기를 변경하여 YOLO v2를 구동하면 속도와
정확도의 trade off를 조절할 수 있음.

작은 입력 크기 -> 빠르고 덜 정확하게 예측
높은 입력 크기 -> 느리고 정확하게 예측

BETTER

		Avg. Precision, IoU:			Avg. Precision, Area:			Avg. Recall, #Dets:			Avg. Recall, Area:		
		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast R-CNN [5]	train	19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast R-CNN[1]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster R-CNN[15]	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [1]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster R-CNN[10]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300 [11]	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512 [11]	trainval35k	**26.8**	**46.5**	**27.8**	**9.0**	**28.9**	**41.9**	**24.8**	**37.5**	**39.8**	**14.0**	**43.5**	**59.0**
YOLOv2 [11]	trainval35k	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4

COCO dataset을 사용해 학습한 결과를 나타낸 표

기준 IOU= 0.5 -> YOLOv2는 44.0mAP를 갖음, Faster R-CNN보다 좋고 SSD랑 비슷함

<Q-정확히 이해못함>

03.

FASTER

Ideas to make it Faster

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			

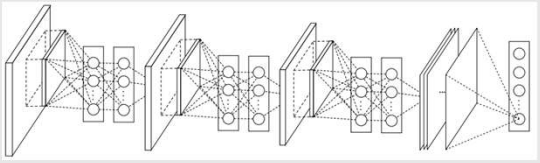
Table 6: Darknet-19.

Darknet-19.

문제점 -> 많은 detection 방법들이 VGG-16을 사용함.
강력하고 정확하지만 연산 량이 많음

해결 -> Darknet-19를 구축하여 사용함

Darknet-19는 19개 convolutional layer와 5개 maxpooling layer로 구성됨
VGG model에서 사용한 3x3 filter와 GoogLeNet에서 이용된 NIN(Network in Network) 기법을 사용.
1x1 filter를 3x3 filter 사이에 넣어 NIN을 통해 차원을 축소시키고 Fully Connected layer를 제거
-> 연산량이 줄어 속도가 빨라짐



ARTIFICIAL INTEL (AI)

THANKS A LOT FOR LISTENING

(AI)