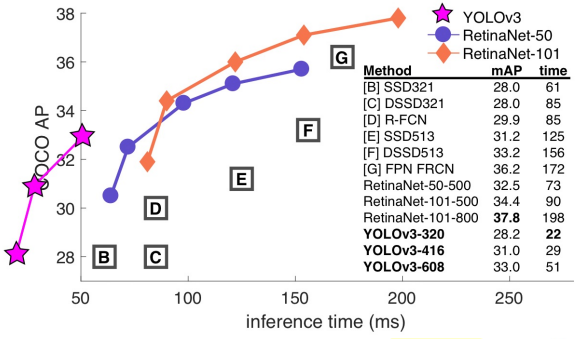
YOLOv3: An Incremental Improvement

Joseph Redmon Ali Farhadi
University of Washington

Abstract

We present 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 320 × 320 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 AP_{50} in 51 ms on a Titan X, compared to 57.5 AP_{50} in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at https://pjreddie.com/yolo/.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

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.

YOLO v3의 성능비교!

1. Introduction

Sometimes you just kinda phone it in for a year, you know? I didn't do a whole lot of research this year. Spent a lot of time on Twitter. Played around with GANs a little. I had a little momentum left over from last year [12] [1]; I managed to make some improvements to YOLO. But, honestly, nothing like super interesting, just a bunch of small changes that make it better. I also helped out with other people's research a little.

Actually, that's what brings us here today. We have a camera-ready deadline [4] and we need to cite some of the random updates I made to YOLO but we don't have a source. So get ready for a TECH REPORT!

The great thing about tech reports is that they don't need intros, y'all know why we're here. So the end of this introduction will signpost for the rest of the paper. First we'll tell you what the deal is with YOLOv3. Then we'll tell you how we do. We'll also tell you about some things we tried that didn't work. Finally we'll contemplate what this all means.

2. The Deal

So here's the deal with YOLOv3: We mostly took good ideas from other people. We also trained a new classifier network that's better than the other ones. We'll just take you through the whole system from scratch so you can understand it all.

2.1. Bounding Box Prediction

Following YOLO9000 our system predicts bounding boxes using dimension clusters as anchor boxes [15]. The network predicts 4 coordinates for each bounding box, t_x, t_y, t_w, t_h. If the cell is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width and height p_w, p_h, then the predictions correspond to:

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

During training we use sum of squared error loss. If the ground truth for some coordinate prediction is $\hat{t}_*$ our gradient is the ground truth value (computed from the ground truth box) minus our prediction: $\hat{t}_* - t_*$. This ground truth value can be easily computed by inverting the equations above.

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. If the bounding box prior is not the best but does overlap a ground truth object by more than some threshold we ignore the prediction, following [17]. We use the threshold of .5. Unlike [17] our system only assigns one bounding box prior for each ground truth object. If a bounding box prior is not assigned to a ground truth object it incurs no loss for coordinate or class predictions, only objectness.

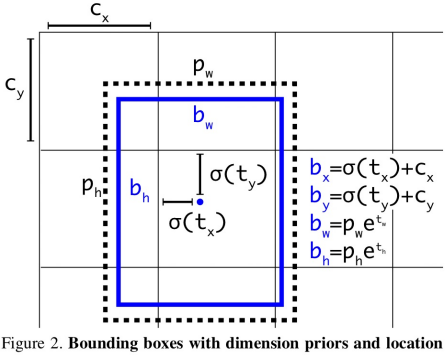


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [15].

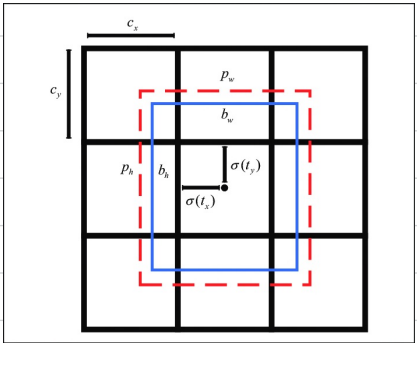
YOLO v2는 bound box를 예측할 때 t_x, t_y, t_w, t_h를 구한 후 위의 그림과 b_x, b_y, b_w, b_h로 변형한 후 L2 Loss를 통해 학습시켰습니다. 여기서 c_x, c_y는 grid cell의 좌상단 좌표 입니다.

하지만 YOLO v3는 ground truth 좌표를 위의 공식을 거꾸로 적용시켜 t_*로 변형 시킨 후 t_x와 직접 L1 Loss를 통해 학습시키는 방식을 선택합니다.

이때 ground truth의 좌표는 위의 공식을 통해 t_*가 됩니다.

예측한 bbox 마다 objectness score는 logistic 함수를 적용하여 계산합니다. 또한 prior box (anchor box)와 ground truth box와의 IoU 값이 가장 높은 box만 대응시킵니다.

ground truth box에 할당되지 못한 bounding box는 bounding box regression loss를 유발하지 않고, 오직 objectness score에 대한 loss 만 발생시킵니다.



$$
\begin{aligned}
b_* &= \sigma(t_*) + c_x \\
\sigma(t_*) &= b_* - c_* \\
t_* &= log(b_* - c_*)
\end{aligned}
$$

2.2. Class Prediction

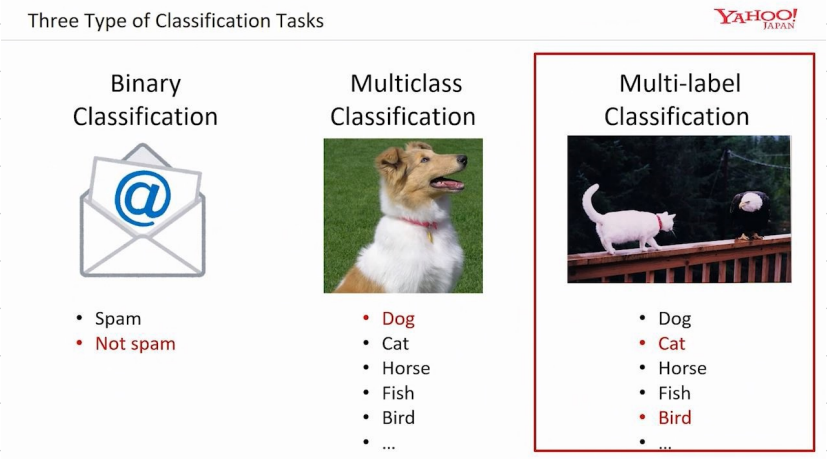
Each box predicts the classes the bounding box may contain using multilabel classification. We do not use a softmax as we have found it is unnecessary for good performance, instead we simply use independent logistic classifiers. During training we use binary cross-entropy loss for the class predictions.

This formulation helps when we move to more complex domains like the Open Images Dataset [7]. In this dataset there are many overlapping labels (i.e. Woman and Person). Using a softmax imposes the assumption that each box has exactly one class which is often not the case. A multilabel approach better models the data.

각각의 bbox는 multi-label classification을 수행합니다.

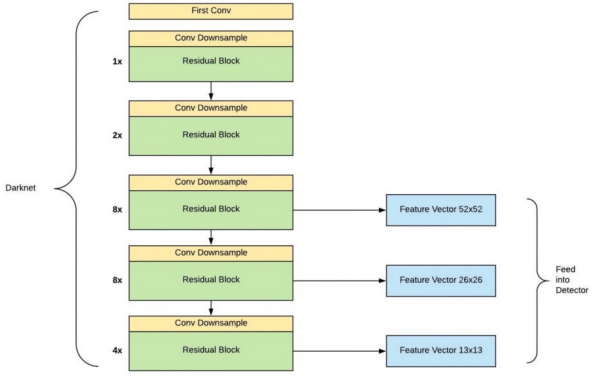
여기서 softmax 함수를 사용하여 class를 예측하면 성능면에서 좋지 않기 때문에 binary cross-entropy를 사용!

가령 하나의 box 안에 복수의 객체가 존재하는 경우 softmax 함수를 이용하여 class를 예측하면 적절하게 객체를 포착하지 못한다는 문제가 있습니다. 따라서 box에서 각 class가 존재하는 여부를 확인하는 binary-cross entropy가 보다 적절하다고 할 수 있습니다.



2.3. Predictions Across Scales

YOLOv3 predicts boxes at 3 different scales. Our system extracts features from those scales using a similar concept to feature pyramid networks [8]. From our base feature extractor we add several convolutional layers. The last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions. In our experiments with COCO [10] we predict 3 boxes at each scale so the tensor is $N \times N \times [3 * (4 + 1 + 80)]$ for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.



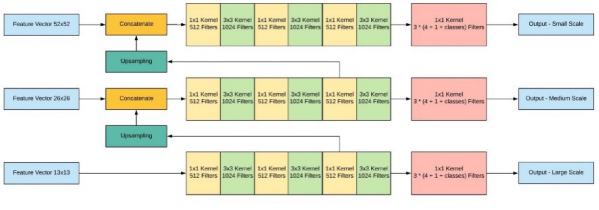
Next we take the feature map from 2 layers previous and upsample it by $2\times$. We also take a feature map from earlier in the network and merge it with our upsampled features using concatenation. This method allows us to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map. We then add a few more convolutional layers to process this combined feature map, and eventually predict a similar tensor, although now twice the size.

We perform the same design one more time to predict boxes for the final scale. Thus our predictions for the 3rd scale benefit from all the prior computation as well as fine-grained features from early on in the network.

We still use k-means clustering to determine our bounding box priors. We 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: $(10 \times 13), (16 \times 30), (33 \times 23), (30 \times 61), (62 \times 45), (59 \times 119), (116 \times 90), (156 \times 198), (373 \times 326)$.

YOLO v3는 서로 다른 3개의 scale을 사용하여 최종 결과를 예측합니다. 여기서 multi-scale feature map을 얻는 방법은 FPN과 유사합니다.

416 x 416크기의 이미지를 input으로 feature map의 크기가 52x52, 26x26, 13x13 이 되도록 feature map을 추출합니다.



가장 높은 level, 즉 해상도가 가장 낮은 feature map은 1x1, 3x3 conv layer로 구성된 작은 FCN에 입력!
이후 FCN의 output channel이 되기가 직는 지점에서 feature map을 추출한 뒤 2배로 upsampling을 수행!
바로 아래에 있는 level의 feature map과 concatenate 해줌! 이후 merged feature map을 FCN에 입력하고 다음 level에서도 반복! 이를 통해 3개의 scale을 가진 feature map을 얻을 수 있음!

이때 각 scale의 feature map의 output channel 수가 $[3 \times (4+1+80)] = 255$이 되도록 마지막 1x1 conv layer의 channel수를 조절!

Class
anchor box
bbox offset
objectness score

2.4. Feature Extractor

We use a new network for performing feature extraction. Our new network is a hybrid approach between the network used in YOLOv2, Darknet-19, and that newfangled residual network stuff. Our network uses successive 3×3 and 1×1 convolutional layers but now has some shortcut connections as well and is significantly larger. It has 53 convolutional layers so we call it.... wait for it..... Darknet-53!

	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			

Table 1. **Darknet-53.**

This new network is much more powerful than Darknet-19 but still more efficient than ResNet-101 or ResNet-152. Here are some ImageNet results:

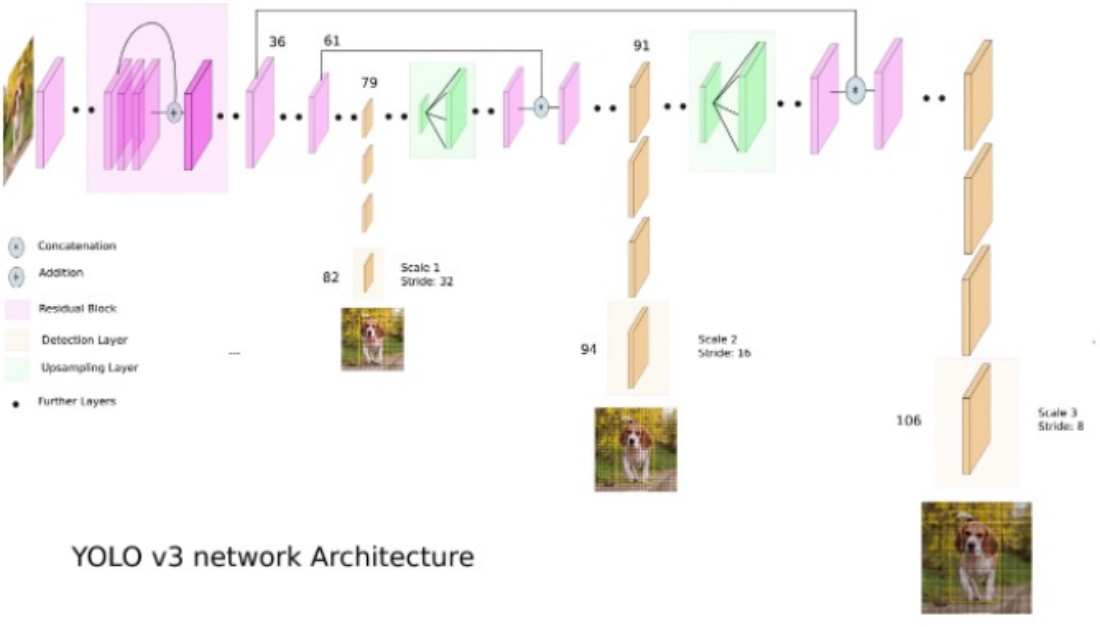
Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Table 2. **Comparison of backbones.** Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

Each network is trained with identical settings and tested at 256×256, single crop accuracy. Run times are measured on a Titan X at 256×256. Thus Darknet-53 performs on par with state-of-the-art classifiers but with fewer floating point operations and more speed. Darknet-53 is better than ResNet-101 and $1.5\times$ faster. Darknet-53 has similar performance to ResNet-152 and is $2\times$ 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. That's mostly because ResNets have just way too many layers and aren't very efficient.

YOLO v3 에서는 shortcut connection이 추가되어 53개의 layer를 가지는 DarkNet-53을 backbone 모델로 사용!



YOLO v3 network Architecture

2.5. Training

We still train on full images with no hard negative mining or any of that stuff. We use multi-scale training, lots of data augmentation, batch normalization, all the standard stuff. We use the Darknet neural network framework for training and testing [14].

3. How We Do

YOLOv3 is pretty good! See table 3. In terms of COCOs weird average mean AP metric it is on par with the SSD variants but is 3× faster. 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= .5 (or AP_{50} in the chart) YOLOv3 is very strong. It is almost on par with RetinaNet and far above the SSD variants. This indicates that YOLOv3 is a very strong detector that excels at producing decent boxes for objects. However, performance drops significantly as the IOU threshold increases indicating YOLOv3 struggles to get the boxes perfectly aligned with the object.

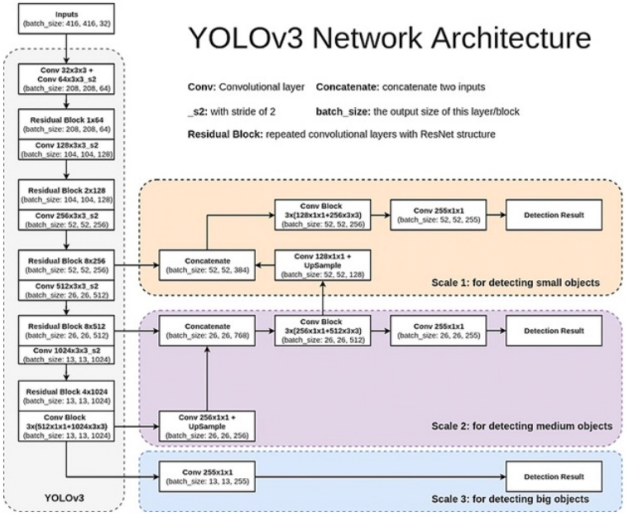
In the past YOLO struggled with small objects. However, now we see a reversal in that trend. With the new multi-scale predictions we see YOLOv3 has relatively high AP_S performance. However, it has comparatively worse performance on medium and larger size objects. More investigation is needed to get to the bottom of this.

When we plot accuracy vs speed on the AP_{50} metric (see figure 5) we see YOLOv3 has significant benefits over other detection systems. Namely, it's faster and better.

1) feature map by DarkNet

이미지를 입력하여 지정한 layer 에서 52x52, 26x26, 13x13 크기의 feature map을 추출합니다.

- input : 416x416 size image
- process : extract feature map
- Output : 52x52, 26x26, 13x13 size feature map



4. Things We Tried That Didn't Work

We tried lots of stuff while we were working on YOLOv3. A lot of it didn't work. Here's the stuff we can remember.

Anchor box x, y offset predictions. We tried using the normal anchor box prediction mechanism where you predict the x, y offset as a multiple of the box width or height using a linear activation. We found this formulation decreased model stability and didn't work very well.

Linear x, y predictions instead of logistic. We tried using a linear activation to directly predict the x, y offset instead of the logistic activation. This led to a couple point drop in mAP.

Focal loss. We tried using focal loss. It dropped our mAP about 2 points. YOLOv3 may already be robust to the problem focal loss is trying to solve because it has separate objectness predictions and conditional class predictions. Thus for most examples there is no loss from the class predictions? Or something? We aren't totally sure.

2. building feature pyramid by FCN

앞서 얻은 3개의 서로 다른 scale을 가진

feature map을 1x1, 3x3 conv로 구성된 FPN에 입력하여 feature pyramid를 설계함!

- input : 52x52, 26x26, 13x13 sized feature map
- process : building feature pyramid by FCN
- output : 52x52 (x255), 26x26 (x255), 13x13 (x255) sized feature map.

3. Train YOLO v3 by loss function

앞서 얻은 multi-scale feature maps을 loss function을 통해 학습시킵니다.

loss funtion은 4개의 항으로 구성되어 있습니다.

1) bound box offset의 MSE
2) 객체를 예측하도록 할당된 bbox의 objectness score의 BCE
3) 객체를 예측하도록 할당되지 않은 bbox의 no objectness score의 BCE
4) bbox의 multi-class BCE