YOLOv3

(You Only Look Once: Unified, Real-Time Object Detection)

First let's know about YOLO:

You Only Look Once: Unified, Real-Time Object Detection by Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi (2015).

It is a network for object detection. The task of object detection consist of determining the location and the objects in it, as well as classifying those objects. Previously there were other methods for classifying images such as R-CNN and its variations these methods have multiple steps and were very slow to run. They were also hard to optimize and need many GPUs. Because in R-CNN each individual components must be trained individually, which usually consumes a lot of time and resources. Whereas YOLO does it all with a single neural network.

So, you take an image as input, pass it through a neural network that looks similar to a normal CNN, and you get a vector of bounding boxes and class predictions in the output.

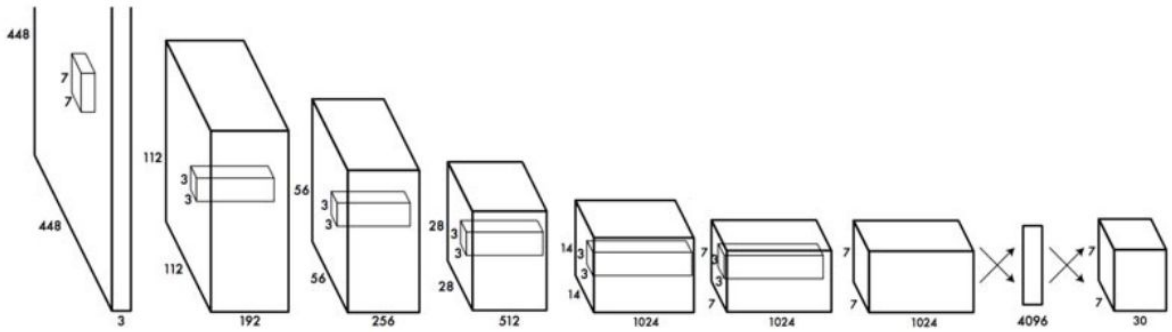
The Predictions Vector

The first step to understanding YOLO is how it encodes its output. The input image is divided into an $S \ x \ S$ grid of cells. For each object that is present on the image, one grid cell is said to be "responsible" for predicting it. That is the cell where the center of the object falls into.

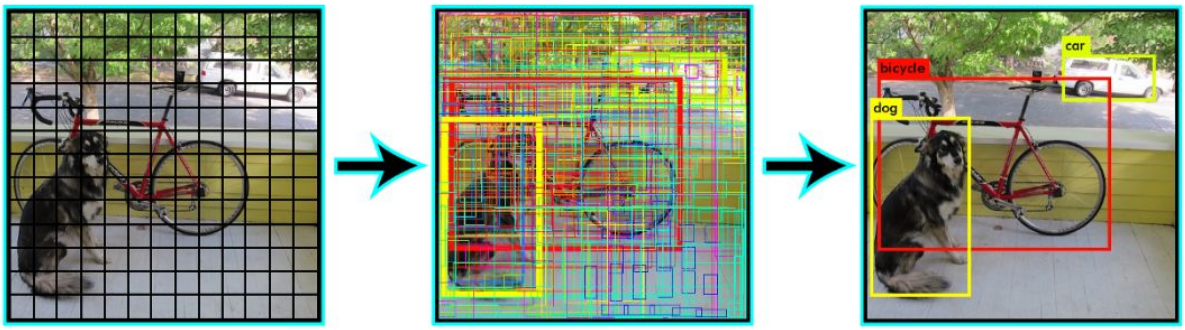
Each grid cell predicts B bounding boxes as well as C class probabilities. The bounding box prediction has 5 components: *(x, y, w, h, confidence)*. The *(x, y)*coordinates represent the center of the box, relative to the grid cell location (remember that, if the center of the box *does not* fall inside the grid cell, than this cell is not responsible for it). These coordinates are normalized to fall between 0 and 1. The *(w, h)* box dimensions are also normalized to [0, 1], relative to the image size.

There is still one more component in the bounding box prediction, which is the confidence score. Formally we define confidence as Pr(Object) * IOU(pred, truth) . If no object exists in that cell, the confidence score should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.

Each boundary box contains 5 elements: (x, y, w, h) and a **box confidence score**. The confidence score reflects how likely the box contains an object (**objectness**) and how accurate is the boundary box. We normalize the bounding box width w and height h by the image width and height. x and y are offsets to the corresponding cell. Hence, x, y, w and h are all between 0 and 1. Each cell has 20 conditional class probabilities. The **conditional class probability** is the probability that the detected object belongs to a particular class (one probability per category for each cell). So, YOLO's prediction has a shape of (S, S, B×5 + C) = (7, 7, 2×5 + 20) = (7, 7, 30).



The major concept of YOLO is to build a CNN network to predict a (7, 7, 30) tensor. It uses a CNN network to reduce the spatial dimension to 7×7 with 1024 output channels at each location. YOLO performs a linear regression using two fully connected layers to make 7×7×2 boundary box predictions (the middle picture below). To make a final prediction, we keep those with high box confidence scores (greater than 0.25) as our final predictions (the right picture).



The **class confidence score** for each prediction box is computed as:

$$\text{class confidence score} = \text{box confidence score} \times \text{conditional class probability}$$

The Loss Function:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2$$

This equation computes the loss related to the predicted bounding box position ***(x,y)***. Don't worry about ***λ*** for now, just consider it a given constant. The function computes a sum over each bounding box predictor ***(j = 0.. B)*** of each grid cell ***(i = 0 .. S^2)***. ***𝟙obj*** is defined as follows:

- 1, If an object is present in grid cell *i* and the *j*th bounding box predictor is "responsible" for that prediction
- 0, otherwise

YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding box predictor to be responsible for each object. We assign one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth.

second part:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2$$

This is the loss related to the predicted box width / height. The equation looks similar to the first one, except for the square root. Our error metric should reflect that small deviations in large boxes matter less than in small boxes. To partially address this we predict the square root of the bounding box width and height instead of the width and height directly.

third part:

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_i)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}(C_i-\hat{C}_i)^2$$

Here we compute the loss associated with the confidence score for each bounding box predictor. $\boldsymbol{C}$ is the confidence score and $\boldsymbol{\hat{C}}$ is the intersection over union of the predicted bounding box with the ground truth.$\boldsymbol{\mathbb{1}^{obj}}$ is equal to one when there is an object in the cell, and 0 otherwise. $\boldsymbol{\mathbb{1}^{noobj}}$ is the opposite.

The $\boldsymbol{\lambda}$ parameters that appear here and also in the first part are used to differently weight parts of the loss functions. This is necessary to increase model stability. The highest penalty is for coordinate predictions ($\boldsymbol{\lambda_{coord}} = 5$) and the lowest for confidence predictions when no object is present ($\boldsymbol{\lambda_{noobj}} = 0.5$).

last part of the loss function is the classification loss:

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

It looks similar to a normal sum-squared error for classification, except for the ***𝟙 obj*** term. This term is used because so we don't penalize classification error when no object is present on the cell

The Training

- First, pretrain the first 20 convolutional layers using the ImageNet 1000-class competition dataset, using a input size of 224x224
- Then, increase the input resolution to 448x448
- Train the full network for about 135 epochs using a batch size of 64, momentum of 0.9 and decay of 0.0005.
- Learning rate schedule: for the first epochs, the learning rate was slowly raised from 0.001 to 0.01. Train for about 75 epochs and then start decreasing it.
- Use data augmentation with random scaling and translations, and randomly adjusting exposure and saturation.

YOLOv2

SSD is a strong competitor for YOLO which at one point demonstrates higher accuracy for real-time processing. Comparing with region based detectors, YOLO has higher localization errors and the recall (measure how good to locate all objects) is lower. YOLOv2 is the second version of the YOLO with the objective of improving the accuracy significantly while making it faster.

Accuracy improvements

Batch normalization

Add batch normalization in convolution layers. This removes the need for dropouts and pushes mAP up 2%.

High-resolution classifier

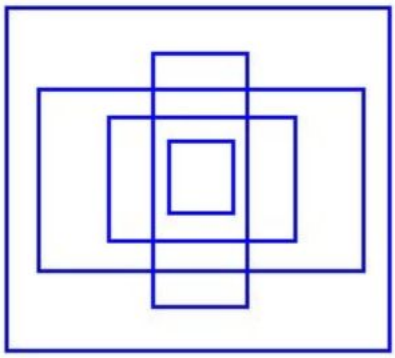
The YOLO training composes of 2 phases. First, we train a classifier network like VGG16. Then we replace the fully connected layers with a convolution layer and retrain it end-to-end for the object detection. YOLO trains the classifier with 224 × 224 pictures followed by 448 × 448 pictures for the object detection. YOLOv2 starts with 224 × 224 pictures for the classifier training but then retune the classifier again with 448 × 448 pictures using much fewer epochs. This makes the detector training easier and moves mAP up by 4%.

Convolutional with Anchor Boxes

As indicated in the YOLO, the early training is susceptible to unstable gradients. Initially, YOLO makes arbitrary guesses on the

boundary boxes. These guesses may work well for some objects but badly for others resulting in steep gradient changes. prediction can fighting with each other on what shapes to specialize on.

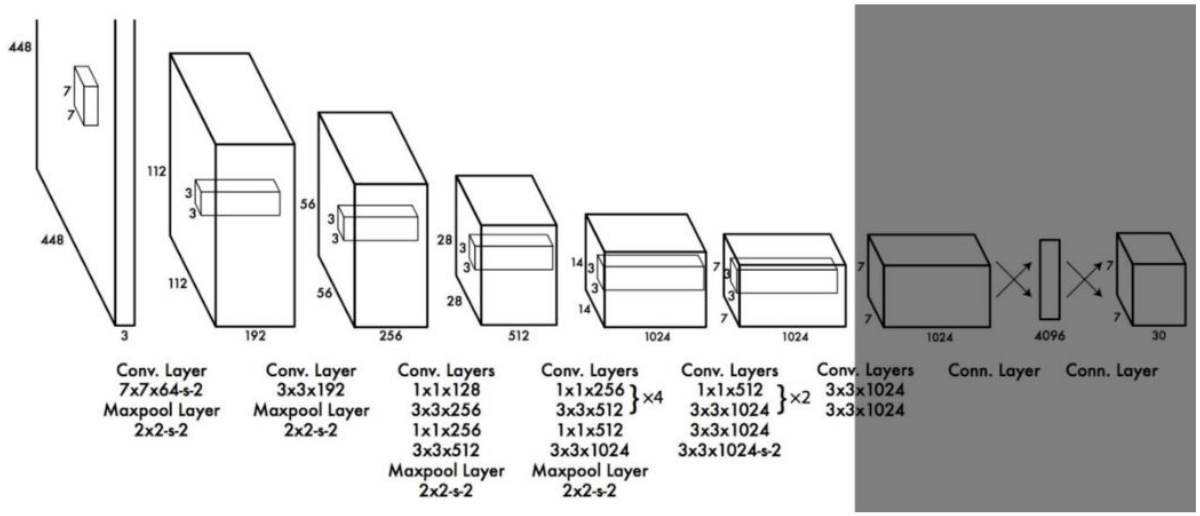
we can create 5 **anchor** boxes with the following shapes.



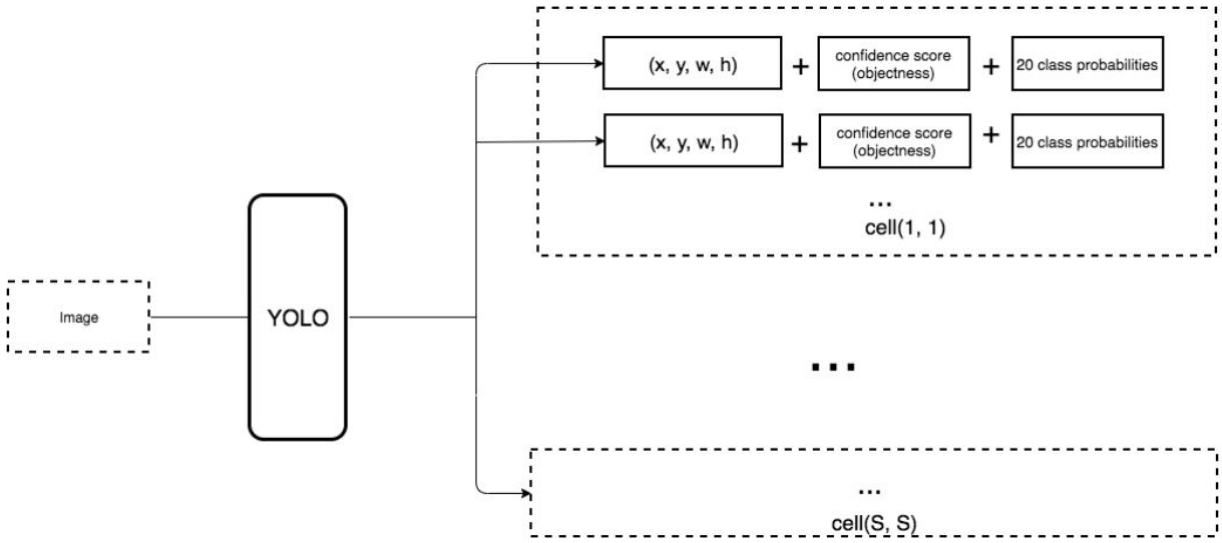
Instead of predicting 5 arbitrary boundary boxes, we predict offsets to each of the anchor boxes above. If we **constrain** the offset values, we can maintain the diversity of the predictions and have each prediction focuses on a specific shape. So the initial training will be more stable.

changes we make to the network:

- Remove the fully connected layers responsible for predicting the boundary box.



- We move the class prediction from the cell level to the boundary box level. Now, each prediction includes 4 parameters for the boundary box, 1 box confidence score (objectness) and 20 class probabilities. i.e. 5 boundary boxes with 25 parameters: 125 parameters per grid cell. Same as YOLO, the objectness prediction still predicts the IOU of the ground truth and the proposed box.

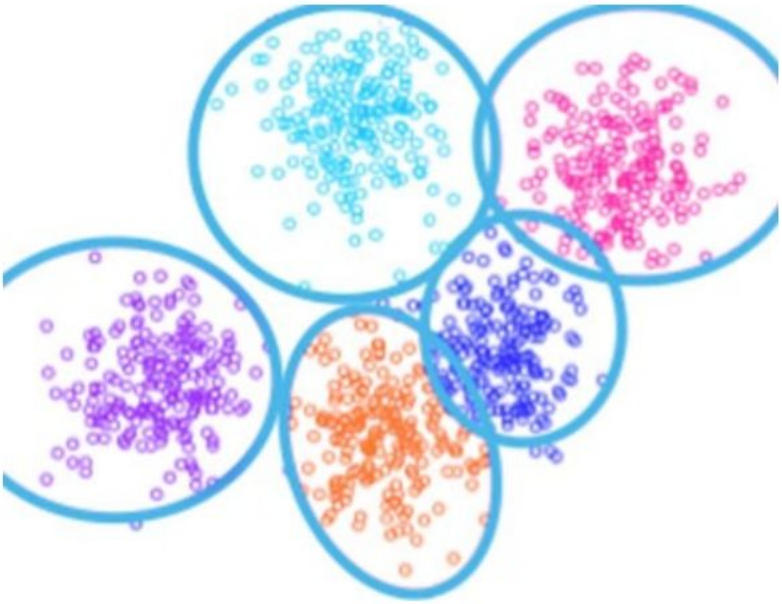


- To generate predictions with a shape of 7 × 7 × 125, we replace the last convolution layer with three 3 × 3 convolutional layers each outputting 1024 output channels. Then we apply a final 1 × 1 convolutional layer to convert the 7 × 7 × 1024 output into 7 × 7 × 125. Change the input image size from 448 × 448 to 416 × 416. This creates an odd number spatial dimension (7×7 v.s. 8×8 grid cell). The center of a picture is often occupied by a large object. With an odd number grid cell, it is more certain on where the object belongs. Remove one pooling layer to make the spatial output of the network to **13×13** (instead of 7×7).

Anchor boxes decrease mAP slightly from 69.5 to 69.2 but the recall improves from 81% to 88%. i.e. even the accuracy is slightly decreased but it increases the chances of detecting all the ground truth objects.

Dimension Clusters

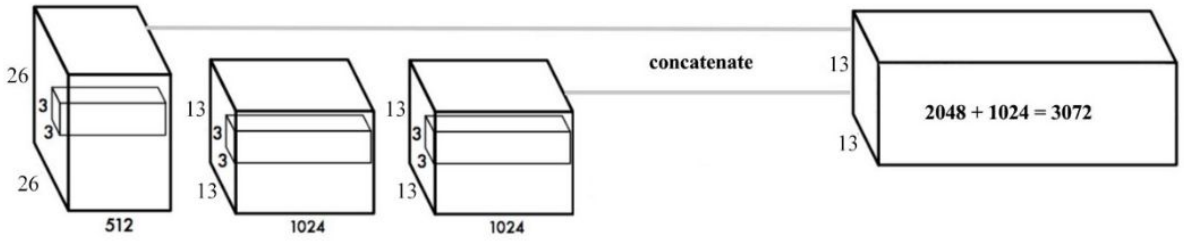
In many problem domains, the boundary boxes have strong patterns. For example, in the autonomous driving, the 2 most common boundary boxes will be cars and pedestrians at different distances. To identify the top-K boundary boxes that have the best coverage for the training data, we run K-means clustering on the training data to locate the centroids of the top-K clusters.



Fine-Grained Features

Convolution layers decrease the spatial dimension gradually. As the corresponding resolution decreases, it is harder to detect small objects. Other object detectors like SSD locate objects from different layers of feature maps. So each layer specializes at a different scale. YOLO adopts a different approach called passthrough. It reshapes the 26 × 26 × 512 layer to 13 × 13 × 2048. Then it concatenates with the original 13 × 13 ×1024 output

layer. Now we apply convolution filters on the new 13 × 13 × 3072 layer to make predictions.



Multi-Scale Training

After removing the fully connected layers, YOLO can take images of different sizes. If the width and height are doubled, we are just making 4x output grid cells and therefore 4x predictions. Since the YOLO network down samples the input by 32, we just need to make sure the width and height is a multiple of 32. During training, YOLO takes images of size 320×320, 352×352, … and 608×608 (with a step of 32). For every 10 batches, YOLOv2 randomly selects another image size to train the model. This acts as data augmentation and forces the network to predict well for different input image dimension and scale. In additional, we can use lower resolution images for object detection at the cost of accuracy. This can be a good tradeoff for speed on low GPU power devices. At 288 × 288 YOLO runs at more than 90 FPS with mAP almost as good as Fast R-CNN. At high-resolution YOLO achieves 78.6 mAP on VOC 2007.

Accuracy

Here is the accuracy improvements after applying the techniques so far:

	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**

Speed improvement

GoogLeNet

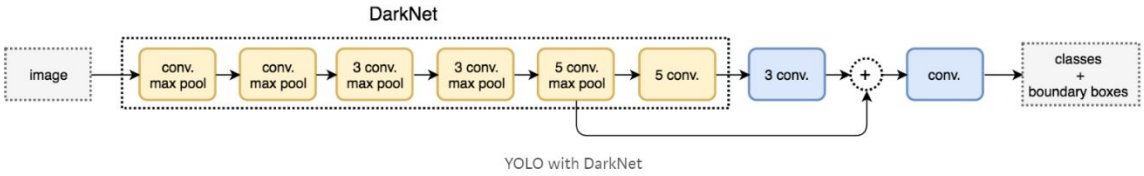
VGG16 requires 30.69 billion floating point operations for a single pass over a 224 × 224 image versus 8.52 billion operations for a customized GoogLeNet. We can replace the VGG16 with the customized GoogLeNet. However, YOLO pays a price on the top-5 accuracy for ImageNet: accuracy drops from 90.0% to 88.0%.

DarkNet

We can further simplify the backbone CNN used. Darknet requires 5.58 billion operations only. With DarkNet, YOLO achieves 72.9% top-1 accuracy and 91.2% top-5 accuracy on ImageNet. Darknet uses mostly 3 × 3 filters to extract features and 1 × 1 filters to reduce output channels. It also uses global average pooling to make predictions.

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			

We replace the last convolution layer (the cross-out section) with three 3×3 convolutional layers each outputting 1024 output channels. Then we apply a final 1×1 convolutional layer to convert the $7 \times 7 \times 1024$ output into $7 \times 7 \times 125$. (5 boundary boxes each with 4 parameters for the box, 1 objectness score and 20 conditional class probabilities)



YOLO with DarkNet

Training

YOLO is trained with the ImageNet 1000 class classification dataset in 160 epochs: using stochastic gradient descent with a starting learning rate of 0.1, polynomial rate decay with a power of 4, weight decay of 0.0005 and momentum of 0.9. In the initial training, YOLO uses 224 × 224 images, and then retune it with 448× 448 images for 10 epochs at a 10−3 learning rate. After the training, the classifier achieves a top-1 accuracy of 76.5% and a top-5 accuracy of 93.3%.

Then the fully connected layers and the last convolution layer is removed for a detector. YOLO adds three 3 × 3 convolutional layers with 1024 filters each followed by a final 1 × 1 convolutional layer with 125 output channels. (5 box predictions each with 25 parameters) YOLO also add a passthrough layer. YOLO trains the network for 160 epochs with a starting learning rate of 10−3 , dividing it by 10 at 60 and 90 epochs. YOLO uses a weight decay of 0.0005 and momentum of 0.9.

YOLOv3

YOLOv3 processes its images on Pascal Titan X at 30 FPS and has a mAP of 57.9% on COCO test-dev.

Class Prediction

Most classifiers assume output labels are mutually exclusive. It is true if the output are mutually exclusive object classes. Therefore, YOLO applies a softmax function to convert scores into probabilities that sum up to one. YOLOv3 uses multi-label classification. For example, the output labels may be “pedestrian” and “child” which are not non-exclusive. (the sum of output can be greater than 1 now.) YOLOv3 replaces the softmax function with independent logistic classifiers to calculate the likeliness of the input belongs to a specific label. Instead of using mean square error in calculating the classification loss, YOLOv3 uses binary cross-entropy loss for each label. This also reduces the computation complexity by avoiding the softmax function.

Bounding box prediction & cost function calculation

YOLOv3 predicts an objectness score for each bounding box using logistic regression. YOLOv3 changes the way in calculating the cost function. If the bounding box prior (anchor) overlaps a ground truth object more than others, the corresponding objectness score should be 1. For other priors with overlap greater than a predefined threshold (default 0.5), they incur no cost. Each ground truth object is associated with one boundary box prior only. If a bounding box prior is not assigned, it incurs no classification and localization lost, just confidence loss on objectness. We use tx and ty (instead of bx and by) to compute the loss.

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

Feature Pyramid Networks (FPN) like Feature Pyramid

YOLOv3 makes 3 predictions per location. Each prediction composes of a boundary box, a objectness and 80 class scores, i.e. N × N × [3 × (4 + 1 + 80)] predictions.

YOLOv3 makes predictions at 3 different scales (similar to the FPN):

1. In the last feature map layer.
2. Then it goes back 2 layers back and upsamples it by 2. YOLOv3 then takes a feature map with higher resolution and merge it with the upsampled feature map using element-wise addition. YOLOv3 apply convolutional filters on the merged map to make the second set of predictions.
3. Repeat 2 again so the resulted feature map layer has good high-level structure (semantic) information and good resolution spatial information on object locations.

To determine the priors, YOLOv3 applies k-means cluster. Then it pre-select 9 clusters. For COCO, the width and height of the anchors are (10×13),(16×30),(33×23),(30×61),(62×45),(59× 119),(116 × 90),(156 × 198),(373 × 326). These 9 priors are

grouped into 3 different groups according to their scale. Each group is assigned to a specific feature map above in detecting objects.

Feature extractor

A new 53-layer Darknet-53 is used to replace the Darknet-19 as the feature extractor. Darknet-53 mainly compose of 3 × 3 and 1× 1 filters with skip connections like the residual network in ResNet. Darknet-53 has less BFLOP (billion floating point operations) than ResNet-152, but achieves the same classification accuracy at 2x faster.

	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			

YOLOv3 performance

YOLOv3's COCO AP metric is on par with SSD but 3x faster. But YOLOv3's AP is still behind RetinaNet. In particular, AP@IoU=.75 drops significantly comparing with RetinaNet which suggests YOLOv3 has higher localization error. YOLOv3 also shows significant improvement in detecting small objects.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [3]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [6]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [4]	Inception-ResNet-v2 [19]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [18]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [13]	DarkNet-19 [13]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [9, 2]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [2]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [7]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [7]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

YOLOv3 performs very well in the fast detector category when speed is important.

References:

1. https://pjreddie.com/media/files/papers/yolo_1.pdf
2. https://hackernoon.com/understanding-yolo-f5a74bbc7967
3. https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088