Jonathan Hui Follow
Deep Learning
Mar 18 · 18 min read

Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3

You only look once (YOLO) is an object detection system targeted for real-time processing. We will introduce YOLO, YOLOv2 and YOLO9000 in this article. For those only interested in YOLOv3, please forward to the bottom of the article. Here is the accuracy and speed comparison provided by the YOLO web site.

Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	-	-	7.07 Bn	200
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35
YOLOv3-416	COCO trainval	test-dev	57.9	140.69 Bn	20

Source

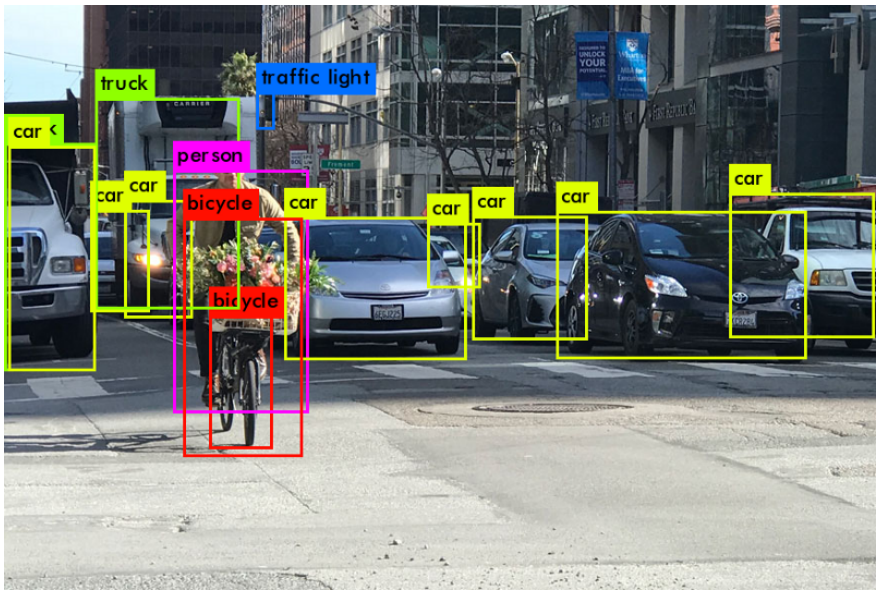
A demonstration from the YOLOv2.

Object detection in real-time

Let’s start with our own testing image below.

Testing image

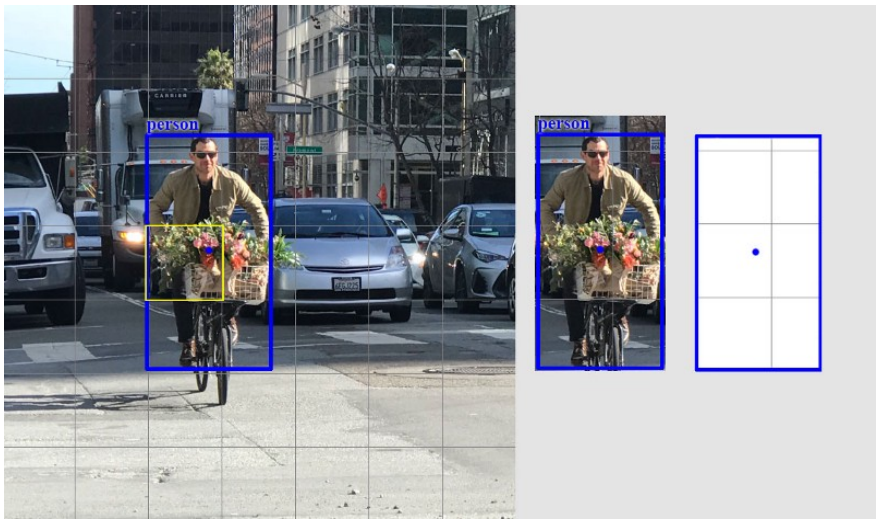
The objects detected by YOLO:



Objects detected by YOLO.

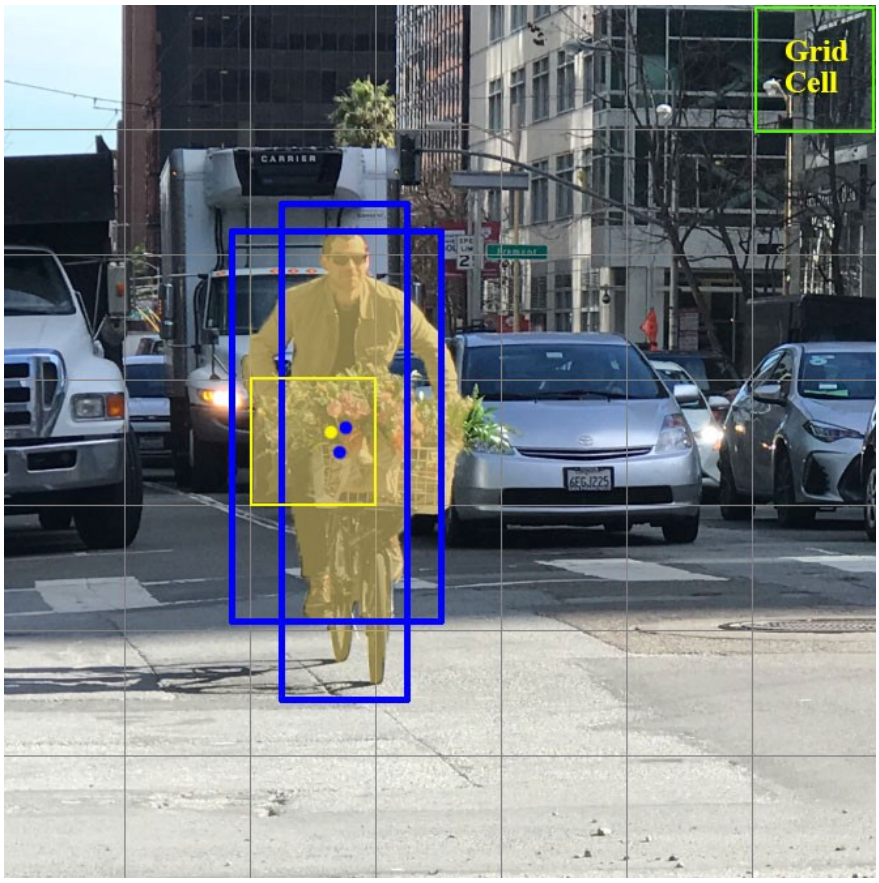
Grid cell

For our discussion, we crop our original photo. YOLO divides the input image into an **S**×**S** grid. Each grid cell predicts only **one** object. For example, the yellow grid cell below tries to predict the "person" object whose center (the blue dot) falls inside the grid cell.



Each grid cell detects only one object.

Each grid cell predicts a fixed number of boundary boxes. In this example, the yellow grid cell makes two boundary box predictions (blue boxes) to locate where the person is.



Each grid cell make a fixed number of boundary box guesses for the object.

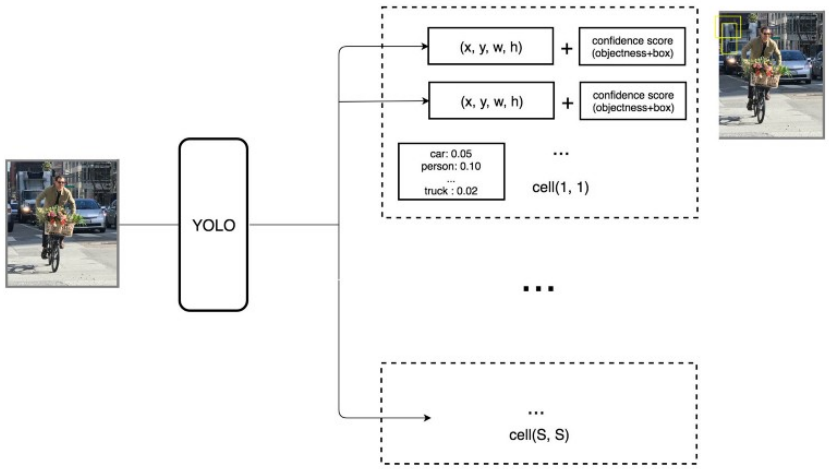
However, the one-object rule limits how close detected objects can be. For that, YOLO does have some limitations on how close objects can be. For the picture below, there are 9 Santas in the lower left corner but YOLO can detect 5 only.

YOLO may miss objects that are too close.

For each grid cell,

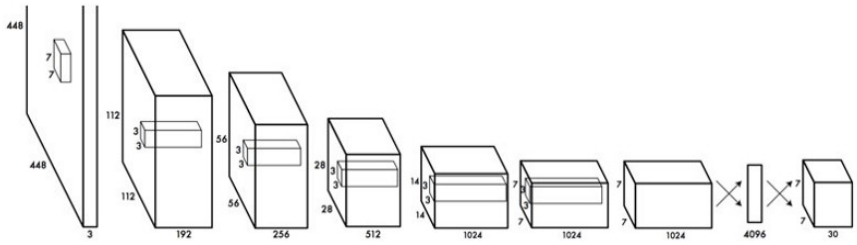
- it predicts **B** boundary boxes and each box has one **box confidence score**,
- it detects **one** object only regardless of the number of boxes B,
- it predicts **C conditional class probabilities** (one per class for the likeliness of the object class).

To evaluate PASCAL VOC, YOLO uses 7×7 grids (S×S), 2 boundary boxes (B) and 20 classes (C).



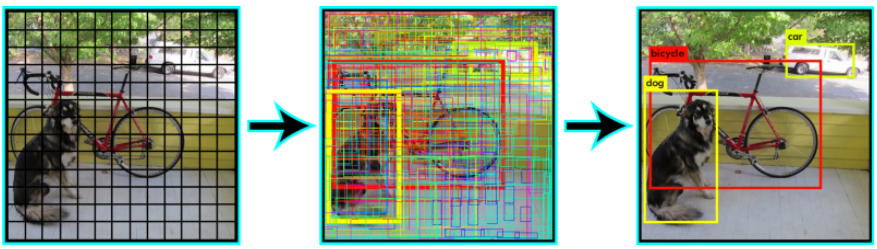
YOYO make SxS predictions with B boundary boxes.

Let's get into more details. 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).



Source

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).



Source

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

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

It measures the confidence on both the classification and the **localization** (where an object is located).

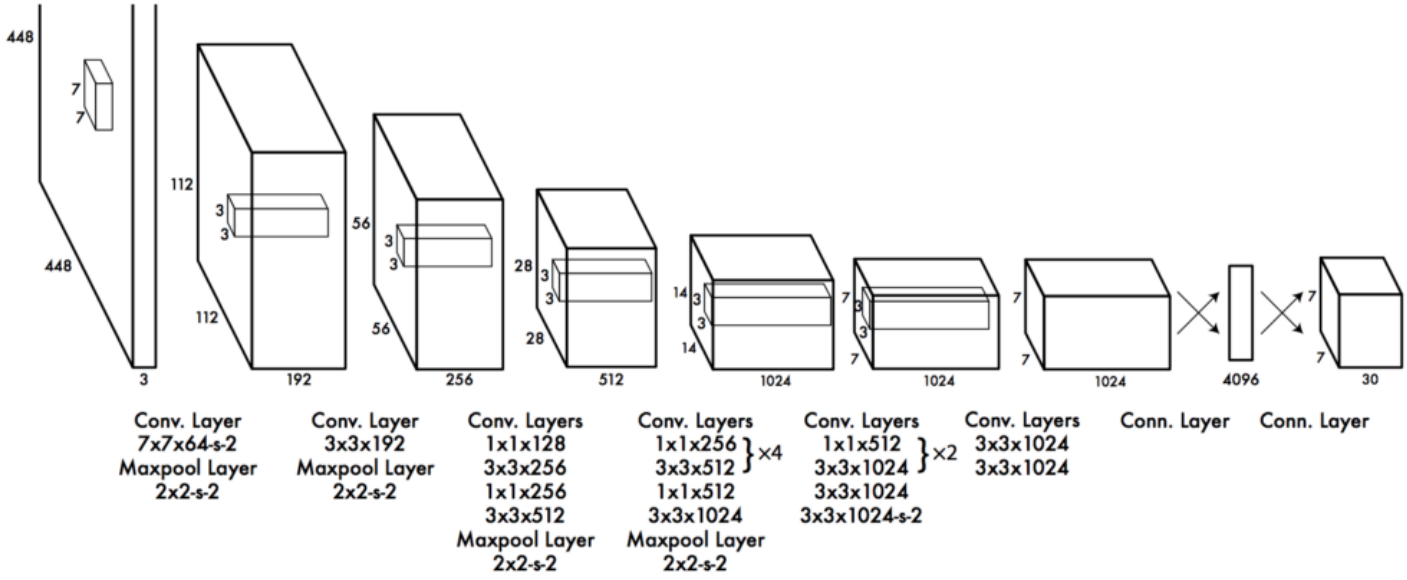
We may mix up those scoring and probability terms easily. Here are the mathematical definitions for your future reference.

$$\begin{aligned} \text{box confidence score} &\equiv P_r(object) \cdot IoU \\ \text{conditional class probability} &\equiv P_r(class_i|object) \\ \text{class confidence score} &\equiv P_r(class_i) \cdot IoU \\ &= \text{box confidence score} \times \text{conditional class probability} \end{aligned}$$

where

$P_r(object)$ is the probability the box contains an object.
IoU is the IoU (intersection over union) between the predicted box and the ground truth.
$P_r(class_i|object)$ is the probability the object belongs to $class_i$ given an object is presence.
$P_r(class_i)$ is the probability the object belongs to $class_i$

Network design



Source

YOLO has 24 convolutional layers followed by 2 fully connected layers (FC). Some convolution layers use 1×1 reduction layers alternatively to reduce the depth of the features maps. For the last convolution layer, it outputs a tensor with shape (7, 7, 1024). The tensor is then flattened. Using 2 fully connected layers as a form of linear regression, it outputs $7 \times 7 \times 30$ parameters and then reshapes to (7, 7, 30), i.e. 2 boundary box predictions per location.

A faster but less accurate version of YOLO, called Fast YOLO, uses only 9 convolutional layers with shallower feature maps.

Loss function

YOLO predicts multiple bounding boxes per grid cell. To compute the loss for the true positive, we only want one of them to be **responsible** for the object. For this purpose, we select the one with the highest IoU (intersection over union) with the ground truth. This strategy leads to specialization among the bounding box predictions. Each prediction gets better at predicting certain sizes and aspect ratios.

YOLO uses sum-squared error between the predictions and the ground truth to calculate loss. The loss function composes of:

- the **classification loss**.

- the **localization loss** (errors between the predicted boundary box and the ground truth).
- the **confidence loss** (the objectness of the box).

Classification loss

If *an object is detected*, the classification loss at each cell is the squared error of the class conditional probabilities for each class:

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

where

$\mathbb{1}_i^{obj} = 1$ if an object appears in cell i, otherwise 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i.

Localization loss

The localization loss measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

where

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

λ_{coord} increase the weight for the loss in the boundary box coordinates.

We do not want to weight absolute errors in large boxes and small boxes equally. i.e. a 2-pixel error in a large box is the same for a small box. To partially address this, YOLO predicts the square root of the bounding box width and height instead of the width and height. In addition, to put more emphasis on the boundary box accuracy, we multiply the loss by $\lambda coord$ (default: 5).

Confidence loss

If *an object is detected in the box*, the confidence loss (measuring the objectness of the box) is:

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

where

$\hat{C}_i$ is the box confidence score of the box j in cell i.

$\mathbb{1}_{ij}^{obj}=1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

If *an object is not detected in the box*, the confidence loss is:

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

where

$\mathbb{1}_{ij}^{noobj}$ is the complement of $\mathbb{1}_{ij}^{obj}$.

$\hat{C}_i$ is the box confidence score of the box j in cell i.

λ_{noobj} weights down the loss when detecting background.

Most boxes do not contain any objects. This causes a class imbalance problem, i.e. we train the model to detect background more frequently than detecting objects. To remedy this, we weight this loss down by a factor λ_{noobj} (default: 0.5).

Loss

The final loss adds localization, confidence and classification losses together.

$$\begin{aligned}
&\lambda_{\textbf{coord}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]\\
&+\lambda_{\textbf{coord}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]\\
&+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(C_i-\hat{C}_i\right)^2\\
&+\lambda_{\text{noobj}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{noobj}}\left(C_i-\hat{C}_i\right)^2\\
&+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{\text{obj}}\sum_{c\in\text{classes}}\left(p_i(c)-\hat{p}_i(c)\right)^2
\end{aligned}$$

Source

Inference: Non-maximal suppression

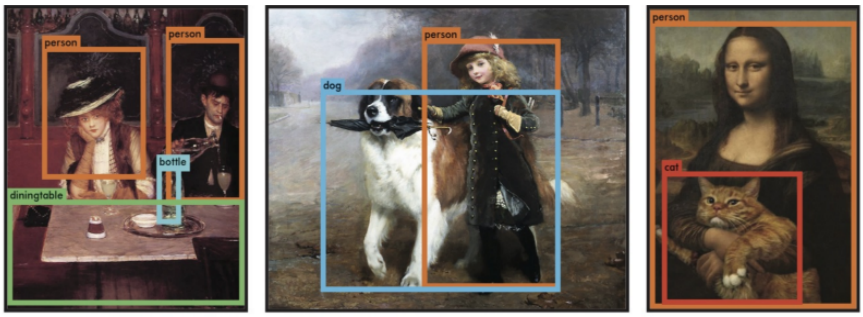
YOLO can make duplicate detections for the same object. To fix this, YOLO applies non-maximal suppression to remove duplications with lower confidence. Non-maximal suppression adds 2- 3% in mAP.

Here is one of the possible non-maximal suppression implementation:

1. Sort the predictions by the confidence scores.
2. Start from the top scores, ignore any current prediction if we find any previous predictions that have the same class and IoU > 0.5 with the current prediction.
3. Repeat step 2 until all predictions are checked.

Benefits of YOLO

- Fast. Good for real-time processing.
- Predictions (object locations and classes) are made from one single network. Can be trained end-to-end to improve accuracy.
- YOLO is more generalized. It outperforms other methods when generalizing from natural images to other domains like artwork.



- Region proposal methods limit the classifier to the specific region. YOLO accesses to the whole image in predicting boundaries. With the additional context, YOLO demonstrates fewer false positives in background areas.
- YOLO detects one object per grid cell. It enforces spatial diversity in making predictions.

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

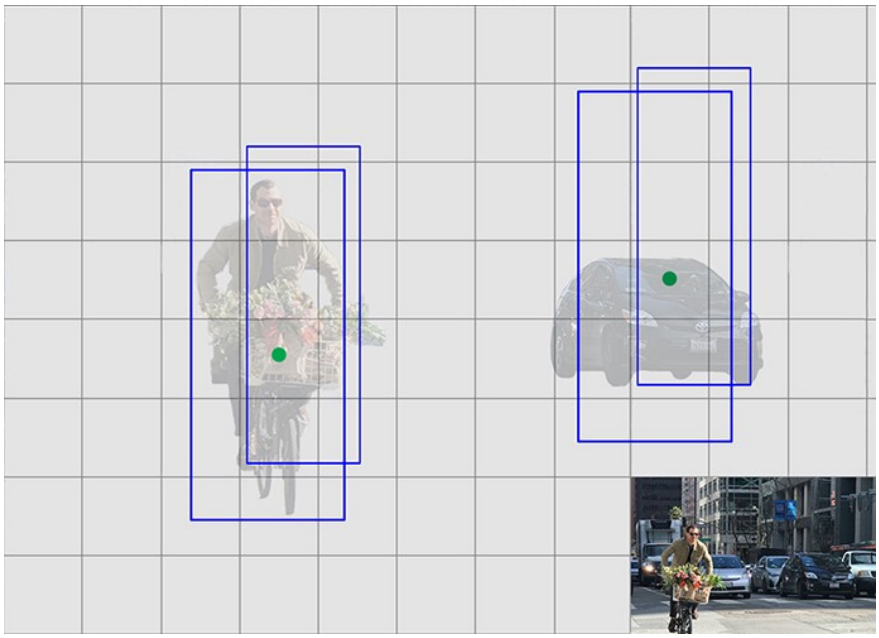
Add batch normalization in convolution layers. This removes the need for dropouts and pushes mAP up 2%.

High-resolution classifier

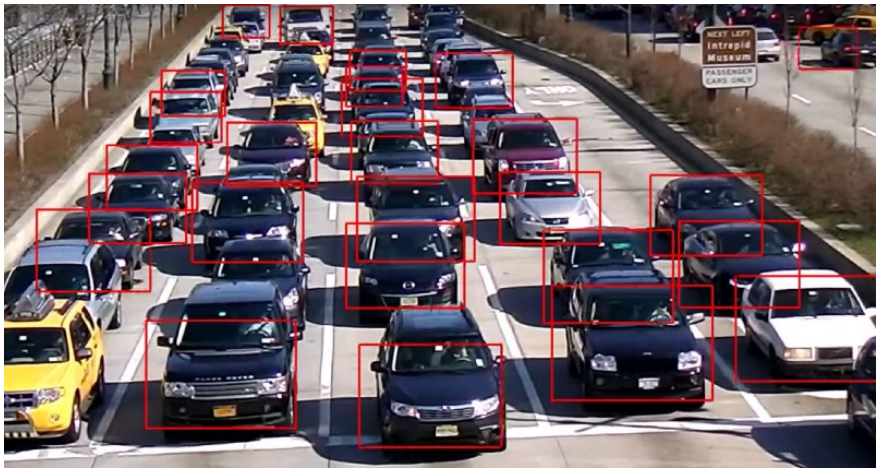
The YOLO training composes of 2 phases. First, we train a classifier network like VGG16. Then we replace the fully connected layers with a convolution layer and retrain it end-to-end for the object detection. YOLO trains the classifier with 224×224 pictures followed by 448×448 pictures for the object 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 paper, the early training is susceptible to unstable gradients. Initially, YOHO makes arbitrary guesses on the boundary boxes. These guesses may work well for some objects but badly for others resulting in steep gradient changes. In early training, predictions are fighting with each other on what shapes to specialize on.

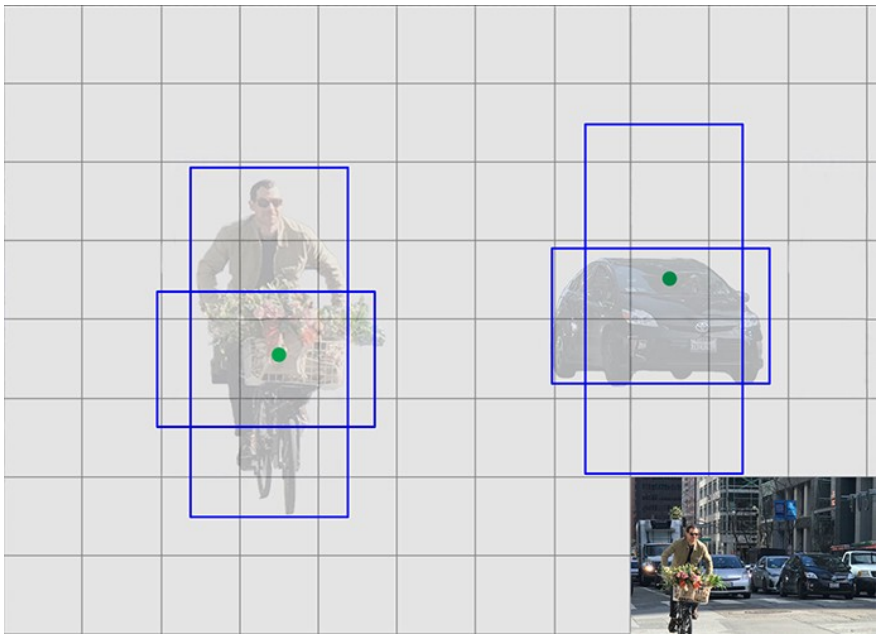


In the real-life domain, the boundary boxes are not arbitrary. Cars have very similar shapes and pedestrians have an approximate aspect ratio of 0.41.



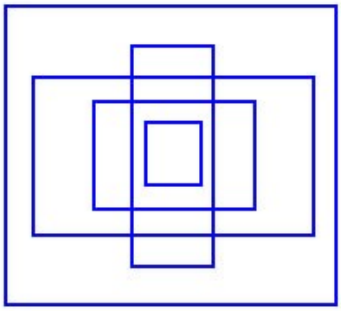
Source

Since we only need one guess to be right, the initial training will be more stable if we start with diverse guesses that are common for real-life objects.



More diverse predictions

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



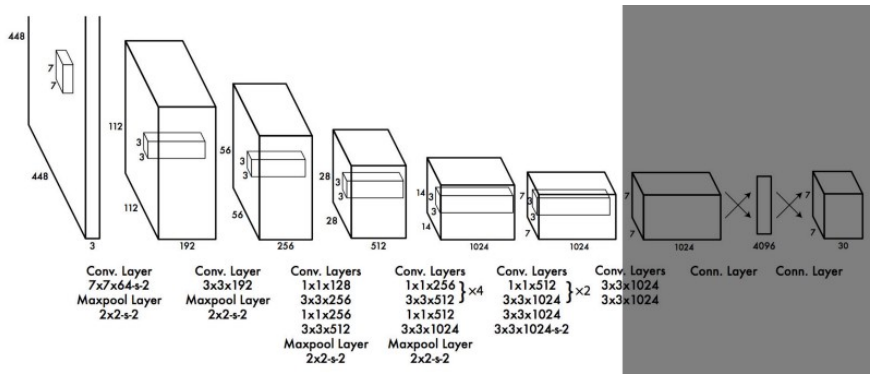
5 anchor boxes

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.

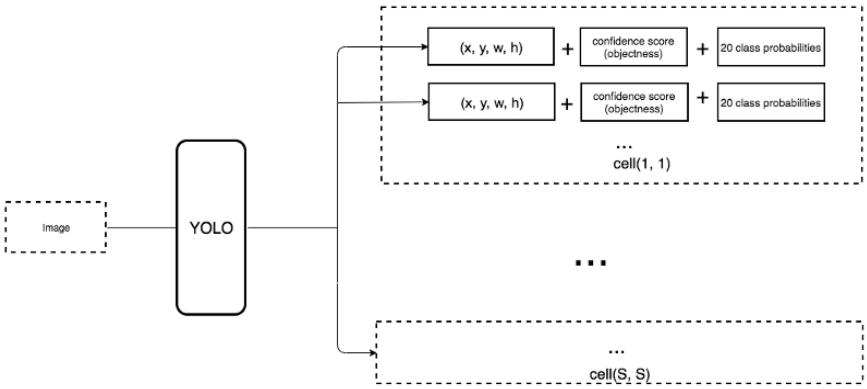
> *In the paper, anchors are also called **priors**.*

Here are the changes we make to the network:

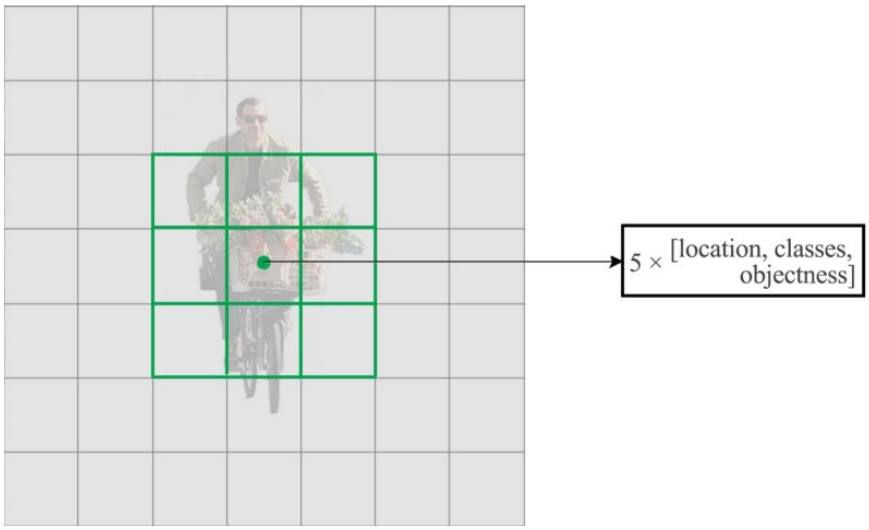
- Remove the fully connected layers responsible for predicting the boundary box.



- 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.

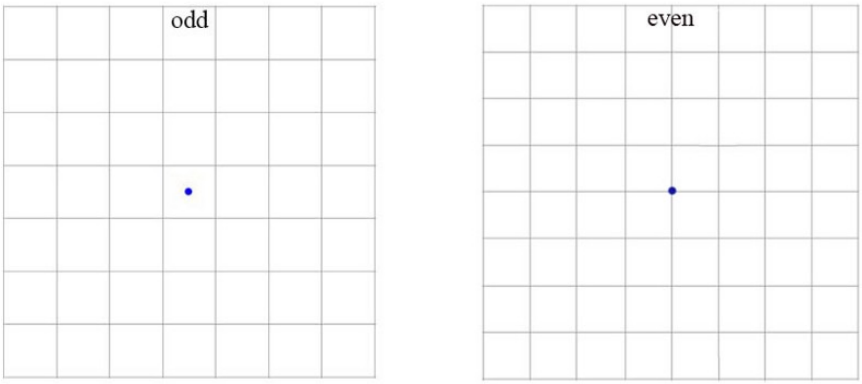


- To generate predictions with a shape of $7 \times 7 \times 125$, we replace the last convolution layer with three 3×3 convolutional layers each outputting 1024 output channels. Then we apply a final 1×1 convolutional layer to convert the $7 \times 7 \times 1024$ output into $7 \times 7 \times 125$. (See the section on DarkNet for the details.)



Using convolution filters to make predictions.

- 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.



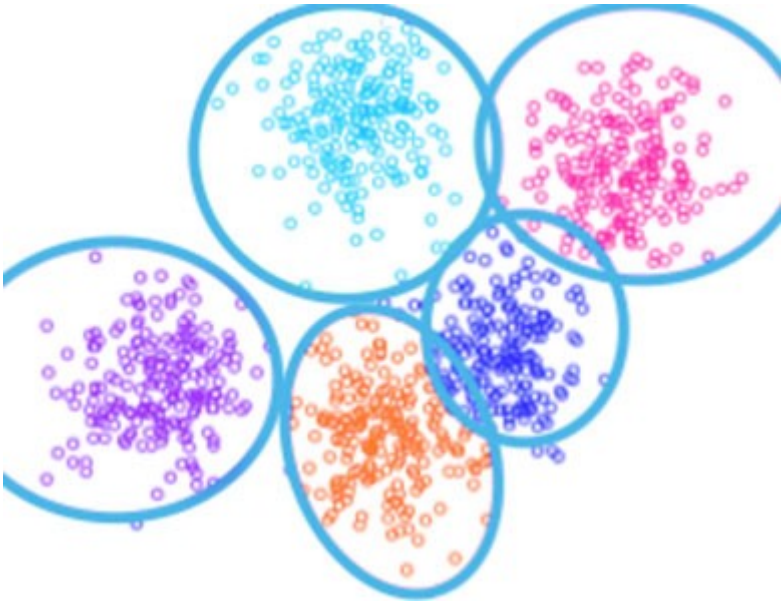
- Remove one pooling layer to make the spatial output of the network to **13×13** (instead of 7×7).

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

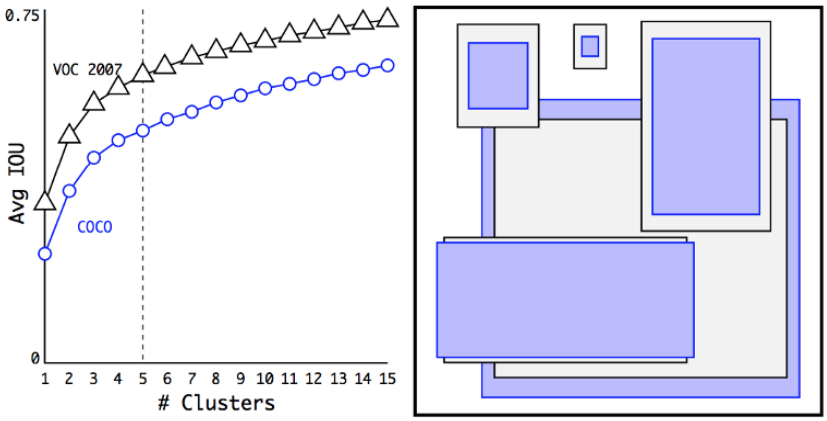
In many problem domains, the boundary boxes have strong patterns. For example, in the autonomous driving, the 2 most common boundary boxes will be cars and pedestrians at different distances. To identify the top-K boundary boxes that have the best coverage for the training data,

we run K-means clustering on the training data to locate the centroids of the top-K clusters.



(Image modified form a k-means cluster)

Since we are dealing with boundary boxes rather than points, we cannot use the regular spatial distance to measure datapoint distances. No surprise, we use IoU.



Source

On the left, we plot the average IoU between the anchors and the ground truth boxes using different numbers of clusters (anchors). As the number of anchors increases, the accuracy improvement plateaus. For the best return, YOLO settles down with 5 anchors. On the right, it displays the 5 anchors' shapes. The purplish-blue rectangles are selected from the COCO dataset while the black border rectangles are

selected from the VOC2007. In both cases, we have more thin and tall anchors indicating that real-life boundary boxes are not arbitrary.

> *Unless we are comparing YOLO and YOLOv2, we will reference YOLOv2 as YOLO for now.*

Direct location prediction

We make predictions on the offsets to the anchors. Nevertheless, if it is unconstrained, our guesses will be randomized again. YOLO predicts 5 parameters (t*x*, t*y*, t*w*, t*h*, and t*o*) and applies the sigma function to constraint its possible offset range.

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h} \\
Pr(\text{object}) * IOU(b, \text{object}) &= \sigma(t_o)
\end{aligned}
$$

where

t_x, t_y, t_w, t_h are predictions made by YOLO.

c_x, c_y is the top left corner of the anchor.

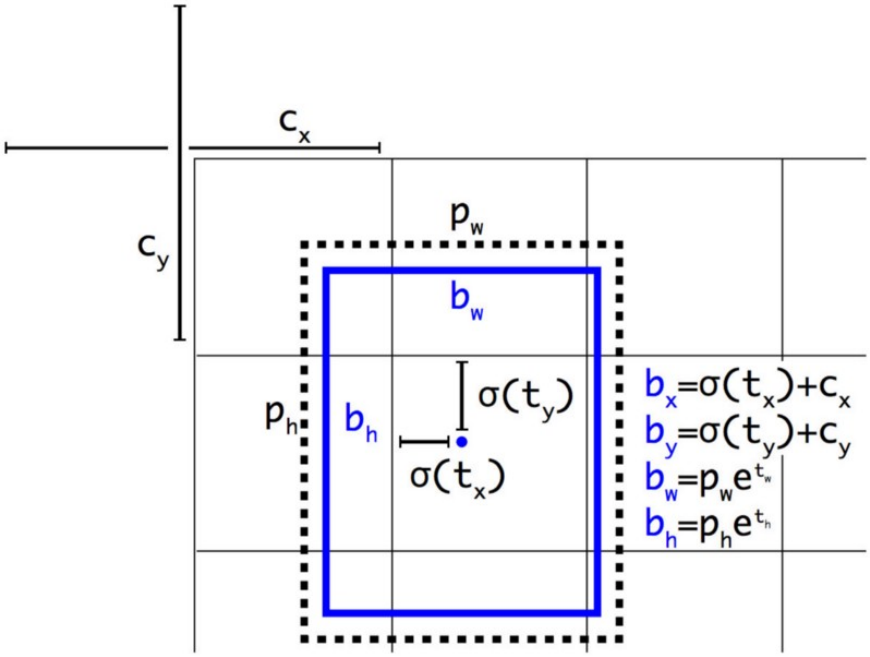
c_w, c_h are the width and height of the anchor.

c_x, c_y, c_w, c_h are normalized by the image width and height.

b_x, b_y, b_w, b_h are the predicted boundary box.

$\sigma(t_o)$ is the box confidence score.

Here is the visualization. The blue box below is the predicted boundary box and the dotted rectangle is the anchor.

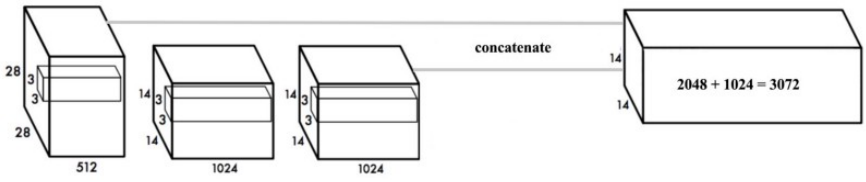


Modified from the paper.

With the use of k-means clustering (dimension clusters) and the improvement mentioned in this section, mAP increases 5%.

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 28 × 28 × 512 layer to 14 × 14 × 2048. Then it concatenates with the original 14 × 14 ×1024 output layer. Now we apply convolution filters on the new 14 × 14 × 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 downsamples 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.

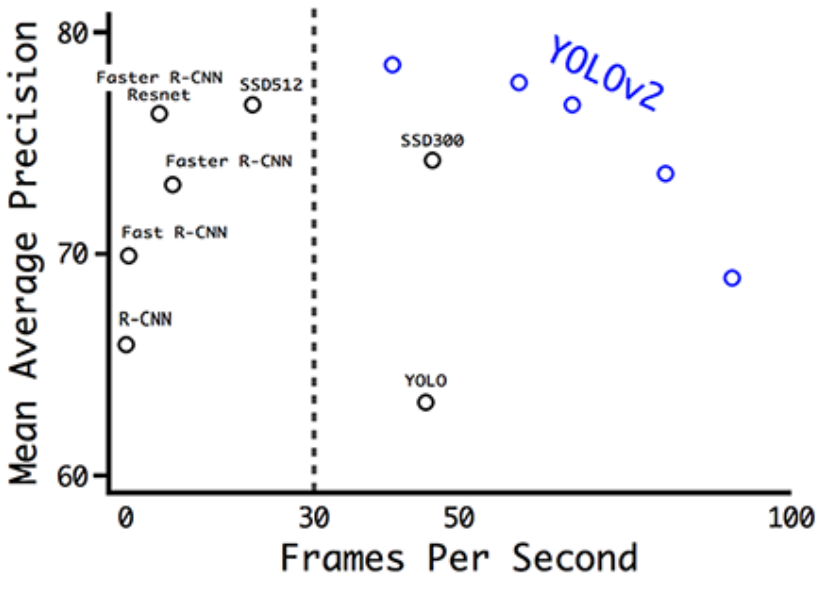
Accuracy

Here is the accuracy improvements after applying the techniques discussed 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**

Source

Accuracy comparison for different detectors:



Source

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%.

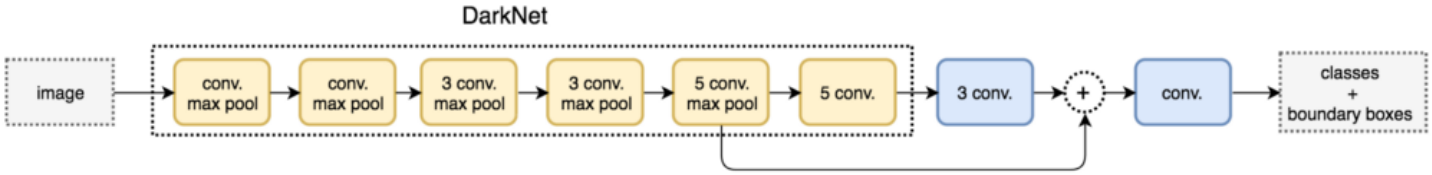
DarkNet

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. Here is the detail network description:

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~~			

Modified from source

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 $\times$ 224 images, and then retune it with 448$\times$ 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.

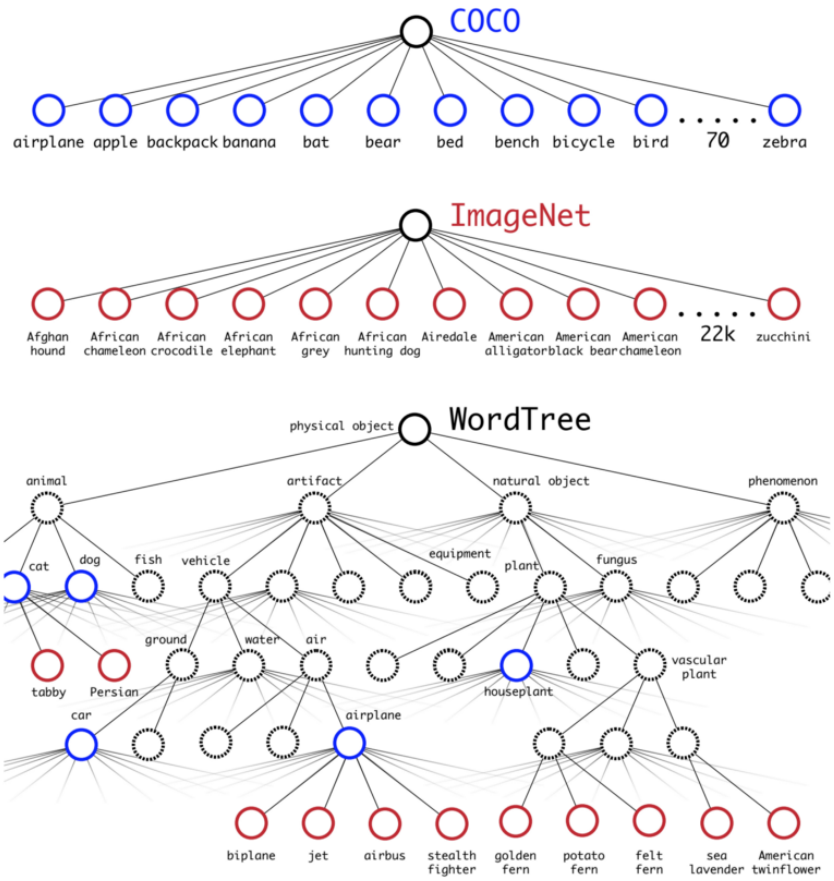
Classification

Datasets for object detection have far fewer class categories than those for classification. To expand the classes that YOLO can detect, YOLO proposes a method to mix images from both detection and classification datasets during training. It trains the end-to-end network with the object detection samples while backpropagates the classification loss from the classification samples to train the classifier path. This approach encounters a few challenges:

- How do we merge class labels from different datasets? In particular, object detection datasets and different classification datasets uses different labels.
- Any merged labels may not be mutually exclusive, for example, *Norfolk terrier* in ImageNet and *dog* in COCO. Since it is not mutually exclusive, we can not use softmax to compute the probability.

Hierarchical classification

Without going into details, YOLO combines labels in different datasets to form a tree-like structure **WordTree**. The children form an is-a relationship with its parent like biplane is a plane. But the merged labels are now not mutually exclusive.



Combining COCO and ImageNet labels to a hierarchical WordTree (source)

Let's simplify the discussion using the 1000 class ImageNet. Instead of predicting 1000 labels in a flat structure, we create the corresponding WordTree which has 1000 leave nodes for the original labels and 369 nodes for their parent classes. Originally, YOLO predicts the class score for the biplane. But with the WordTree, it now predicts the score for the biplane given it is an airplane.

$$score(biplane|airplane)$$

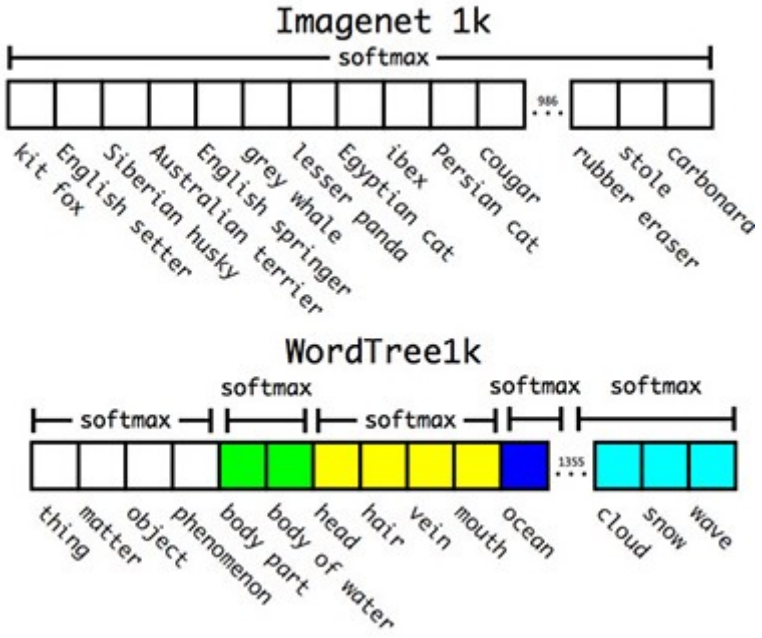
Since

$$P_r(\text{biplane}|\text{airplane}) + P_r(\text{jet}|\text{airplane}) + \cdots + P_r(\text{stealth fighter}|\text{airplane}) = 1$$

we can apply a softmax function to compute the probability

$$P_r(class_i|class_{parent})$$

from the scores of its own and the siblings. The difference is, instead of one softmax operations, YOLO performs multiple softmax operations for each parent's children.



source

The class probability is then computed from the YOLO predictions by going up the WordTree.

$$\begin{aligned} P_r(\text{biplane}) = & P_r(\text{biplane}|\text{airplane}) \cdot P_r(\text{airplane}|\text{air}) \\ & \cdot P_r(\text{air}|\text{vehicle}) \cdot P_r(\text{vehicle}|\text{artifact}) \\ & \cdot P_r(\text{artifact}|\text{physical object}) \\ & \cdot P_r(\text{physical object}) \end{aligned}$$

For classification, we assume an object is already detected and therefore Pr(physical object) = 1.

One benefit of the hierarchy classification is that when YOLO cannot distinguish the type of airplane, it gives a high score to the airplane instead of forcing it into one of the sub-categories.

When YOLO sees a classification image, it only backpropagates classification loss to train the classifier. YOLO finds the bounding box that predicts the highest probability for that class and it computes the classification loss as well as those from the parents. (If an object is labeled as a biplane, it is also considered to be labeled as airplane, air, vehicle…) This encourages the model to extract features common to them. So even we have never trained a specific class of objects for

object detection, we can still make such predictions by generalizing predictions from related objects.

In object detection, we set Pr(physical object) equals to the box confidence score which measures whether the box has an object. YOLO traverses down the tree, taking the highest confidence path at every split until it reaches some threshold and YOLO predicts that object class.

YOLO9000

YOLO9000 extends YOLO to detect objects over 9000 classes using hierarchical classification with a 9418 node WordTree. It combines samples from COCO and the top 9000 classes from the ImageNet. YOLO samples four ImageNet data for every COCO data. It learns to find objects using the detection data in COCO and to classify these objects with ImageNet samples.

During the evaluation, YOLO test images on categories that it knows how to classify but not trained directly to locate them, i.e. categories that do not exist in COCO. YOLO9000 evaluates its result from the ImageNet object detection dataset which has 200 categories. It shares about 44 categories with COOC. Therefore, the dataset contains 156 categories that have never been trained directly on how to locate them. YOLO extracts similar features for related object types. Hence, we can detect those 156 categories by simply from the feature values.

YOLO9000 gets 19.7 mAP overall with 16.0 mAP on those 156 categories. YOLO9000 performs well with new species of animals not found in COCO because their shapes can be generalized easily from their parent classes. However, COCO does not have bounding box labels for any type of clothing so the test struggles with categories like “sunglasses”.

YOLOv3

A quote from the YOLO web site on YOLOv3:

> *On a Pascal Titan X it processes images 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.

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

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			

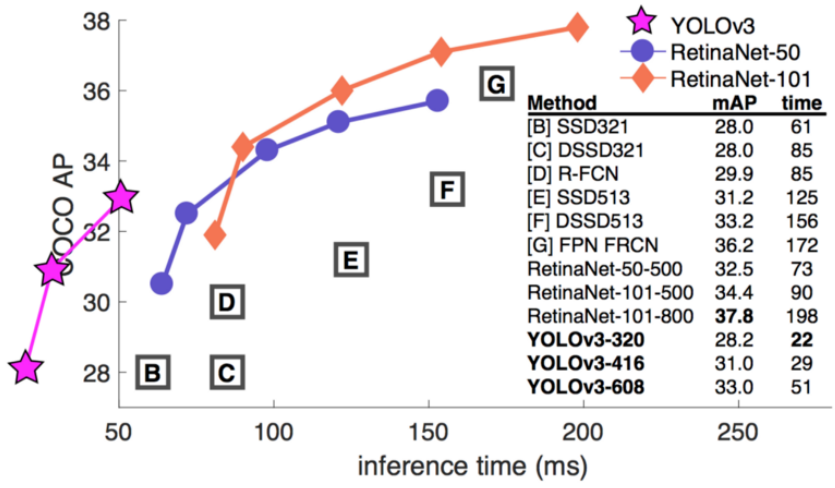
Darknet-53

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.



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

Source

Resources

The original paper on YOLO.

Paper on YOLOv2 and YOLO9000.

DarkNet implementation.