**YOLO9000: Better, Faster, Stronger**

*Joseph Redmon, Ali Farhadi*

* We have huge amount of data for classification, but not so much for detection. Moreover, creating dataset for detection is quite time consuming because we need bounding boxes as well. Thus, authors propose a method that uses classification data along with detection data for training.
* YOLO v1 was improved to create YOLOv2. This v2 was trained to detect objects of 9000 classes, thus it is called YOLO9000
* Two problems with YOLOv1: localization errors and low recall. YOLO v2 addresses these problems.
* V2 uses BatchNormalization. This improves mAP by 2%. It also has regularization properties, so dropouts are removed in v2
* V1 trains the classification model first on 248\*248 resolution data and then increases the resolution to 448\*448 for detection.

In V2, a pre-trained classification network is fine-tuned on 448\*448 resolution ImageNet data for 10 epochs. This network is then fine-tuned on detection data.

* V1 predicts bounding boxes using FC layers on the conv. feature map.

V2 uses anchor boxes, similar to Faster R-CNN.

It uses 416\*416 input, removes one pool layer, and replace FC layers with conv. layers.

Since YOLO downsamples input by 32, for 416\*416 input, output will be of shape 13\*13

* In v1, we had one set of class probabilities per cell. In v2, we have a set of probabilities per bounding box.
* Using anchor boxes decreases mAP from 69.5 to 69.2 but increases recall from 81% to 88%
* The anchors are obtained by running k-means on the dataset.

Standard k-means uses Euclidean distance, but this doesn’t work well here because larger boxes generate more error than the smaller boxes. Since, we care about IoU, we use the below function:



Compare average IoU with closest centroid for various values of k. If k is large, computations will increase. So, there is a trade-off.

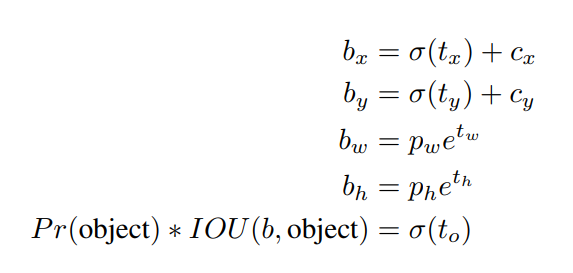
YOLO v2 uses k=5. So we have 5 bounding boxes per cell.

Using anchors instead of directly predicting bounding boxes helps training.

* V2 defines the bounding box coordinates relative to the grid cell. So, the top-left corner of each cell is (0, 0) and the bottom-right corner is (1, 1). Logistic function is used to ensure that values output by the network are in the range [0, 1].

The model predicts 5 bounding boxes, each specified by 5 numbers: , , , , and

If the grid cell is offset by , from the top-left corner of the input image and and are the height and width of the anchor associated with the bounding box under consideration, the bounding box becomes:



(Shouldn’t and be multiplied by and respectively?)

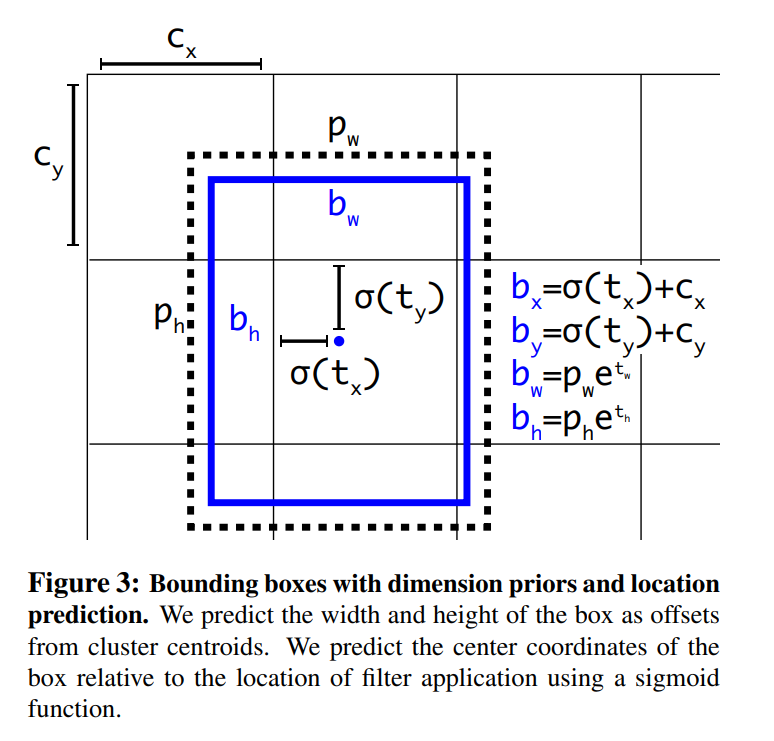
Another way to remember third and fourth equation is:

and

Basically, we are calculating bounding box width and height relative to the anchor’s width and height and using a log scale for the measurement.

**Note:** The network generates , , , and which are all relatively measured. To get the actual bounding box, we use the equations from the above image.

However, during training, you need to carry out the reverse procedure to get the ground-truth box details , , , and for each ground-truth box. Using these values as targets, we train the model to generate , , , and

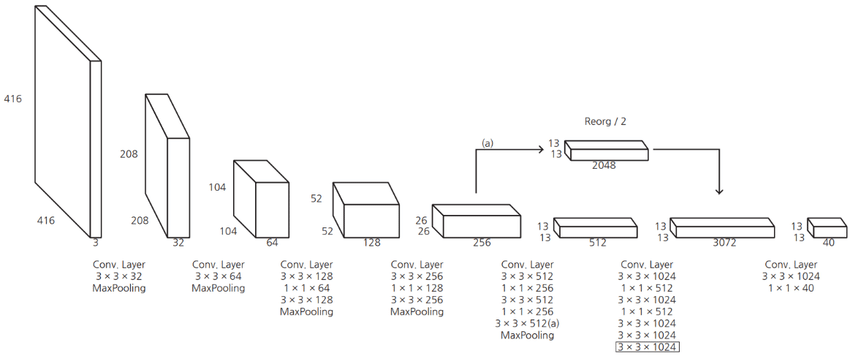


* YOLO v2 has 13\*13 conv. feature map from which we can detect objects. However, this will work for large objects but not for small objects. So, v2 uses a passthrough layer that brings the output of shape 26\*26 of a conv. layer and concatenates the 13\*13 conv. output with it along the channels to form a higher resolution feature map.

*So, the 26\*26\*256 output is “reorged” to 13\*13\*2048 and then concatenated with the 13\*13\*512 output to form a 13\*13\*(2048+512) output. The model now uses this feature map for detection.*

*Note: 2048+512 = 2560, but below image shows 3072, which is 512 more. So, this ‘reorg’ changes the no. of channels.*

<https://www.researchgate.net/figure/The-architecture-of-YOLOv2_fig4_336177198>



* YOLO v2 takes 416\*416 dimensional input, but to make the model work with images of various resolutions, training was done in a peculiar way.

The network is trained on batches of images of various resolutions. Since the model downsamples input by 32, resolutions tried were {320, 352, …, 608}. Thus, the lowest resolution was 288\*288 and the highest resolution was 608\*608. Depending on the resolution, network is modified for training.

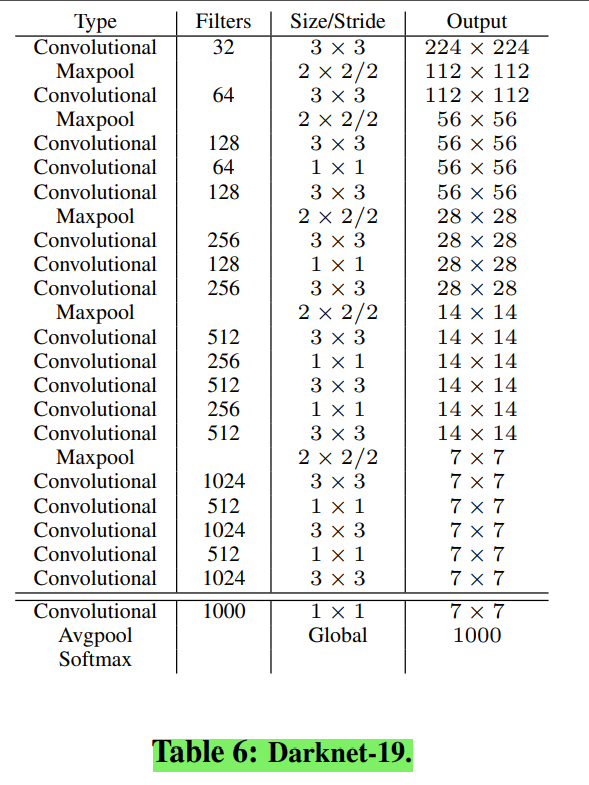
* The low resolution network is faster, so there is basically a tradeoff between accuracy and speed.

However, even at max resolution, the model is still real-time.

* Most of the detection frameworks rely on VGG-16 for feature extraction. It is highly accurate but costly. Thus, YOLO doesn’t use it; it uses a custom network **based** on GoogLeNet architecture. This network is faster than VGG-16 but is less accurate.
* This custom classification model used in YOLO is called Darknet-19. Similar to VGG, it has 3\*3 conv. layers and it doubles the filters as we go deeper in the network. The network uses 1\*1 conv. and global average pooling layer to reduce dimensions.

This model has 19 conv. layers and 5 max-pool layers.

Find Darknet architecture below:



* Training:

Trained the Darknet on ImageNet (224\*224 resolution) using SGD, initial learning rate 0.1, polynomial decay with a power of 4, weight decay 0.0005, and momentum 0.9

Then, fine-tuned the network on 448\*448 images for 10 epochs and initial learning rate 0.001

Now, modify the network for detection. Remove the last conv. layer and add three 3\*3 conv. layers, each having 1024 filters. Finally insert 1\*1 conv. layer with the no. of filters as per output requirements.

Train the model for 160 epochs with initial learning rate 0.001, momentum 0.9, and decay 0.0005

* Training using classification and detection dataset

Mix the two datasets. When the input to the network is a detection image, use the full loss function defined in the paper (given above in this file). When the input to the network is a classification image, use only classification related loss from the loss function and execute backpropagation accordingly.

* SKIPPED – how the two datasets are combined