YOLO FAMILY

YOLOv2

Agenda

Topics Covered

Why YOLOv2?

How does YOLOv2 Work?

Comparison with other methods

YOLO 9000: Better, Faster, Stronger

YOLOv2

Why YOLOv2?

Why YOLOv2?

Clearly, YOLOv1 performed a lot faster compared to the other methods. But, it's detections suffered by ~10 mAP compared to Faster R-CNN VGG-16.

YOLOv1 makes a significant number of localization errors. Furthermore, YOLOv1 has a relatively low recall.

So, the authors focused mainly on improving recall and localization while maintaining classification accuracy.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18

Why YOLOv2?

This was one of the motive behind the 2nd version of YOLO, which was introduced in late 2016.

YOLOv2 outperforms all the other methods in both speed and detection.

At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP (Mean Average Precision).

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18

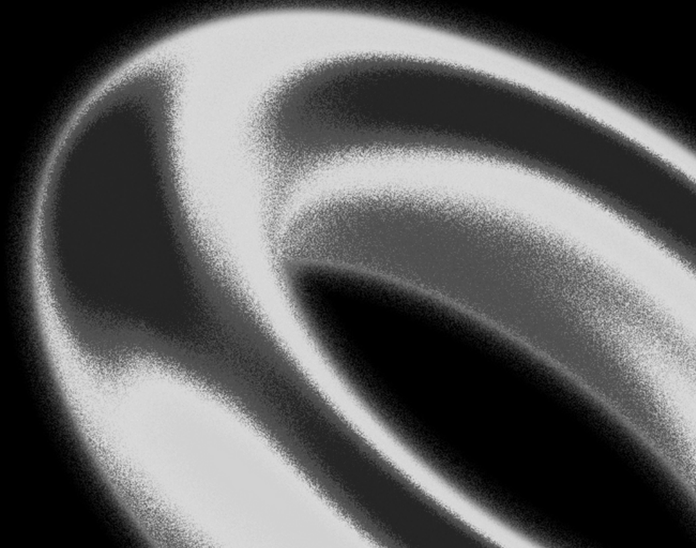
YOLOv2

How does YOLOv2 work?

How does YOLOv2 Work?

Computer vision generally trends towards larger, deeper networks. Better performance often hinges on training larger networks or ensembling multiple models together.

However, with YOLOv2, they wanted a more accurate detector that is still fast. Instead of scaling up the network, they simplified the network to make the representation easier to learn. Here's how they improved YOLO's performance:



Batch Normalization

They added batch normalization after every convolutional layer in YOLOv1.

This itself resulted in 2% increment in mAP.

Batch Normalization improves the model convergence while regularising it.

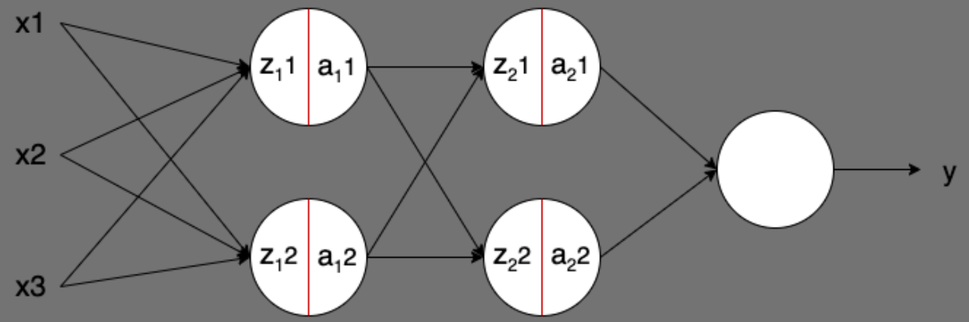
Due to this, they removed the Dropout layer which was used in YOLOv1.

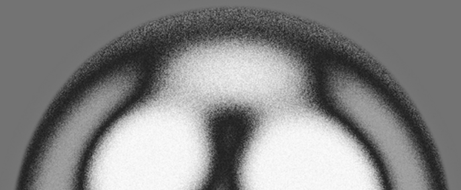
Batch Normalization

Batch Norm is a normalization technique done between the layers of a Neural Network instead of in the raw data.

It is done along mini-batches instead of the full data set.

Batch Norm — in the image represented with a red line — is applied to the neurons' output just before applying the activation function.



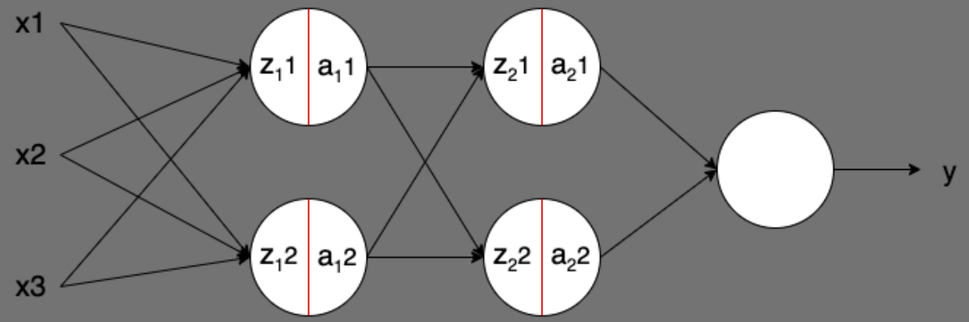


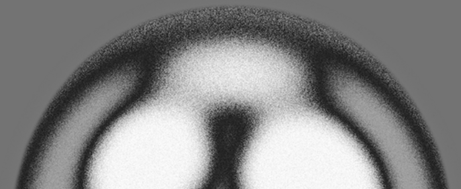
Batch Normalization

Usually, a neuron without Batch Norm would be computed as follows:

$$z = g(w, x) + b; \qquad a = f(z)$$

g() the linear transformation of the neuron, w the weights of the neuron, b the bias of the neurons, and f() the activation function.



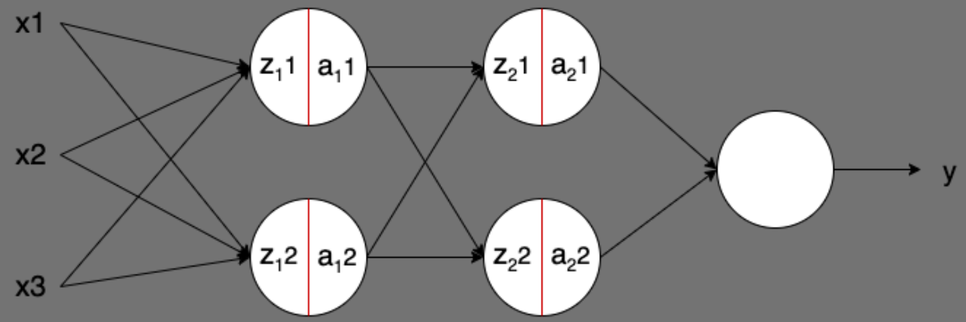


Batch Normalization

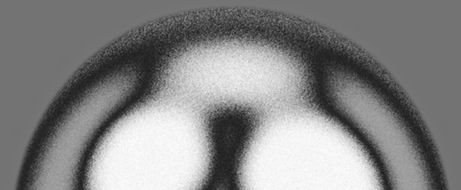
Adding Batch Norm, it looks as:

$$z = g(w, x); \qquad z^N = \left(\frac{z - m_z}{s_z}\right) \cdot \gamma + \beta; \qquad a = f(z^N)$$

z^N the output of Batch Norm, **m_z** the mean of the neurons' output, **s_z** the standard deviation of the output of the neurons, and **gamma** and **beta** learning parameters of Batch Norm.



Note that the bias of the neurons (b) is removed. This is because as we subtract the mean m_z, any constant over the values of z – such as b – can be ignored as it will be subtracted by itself.

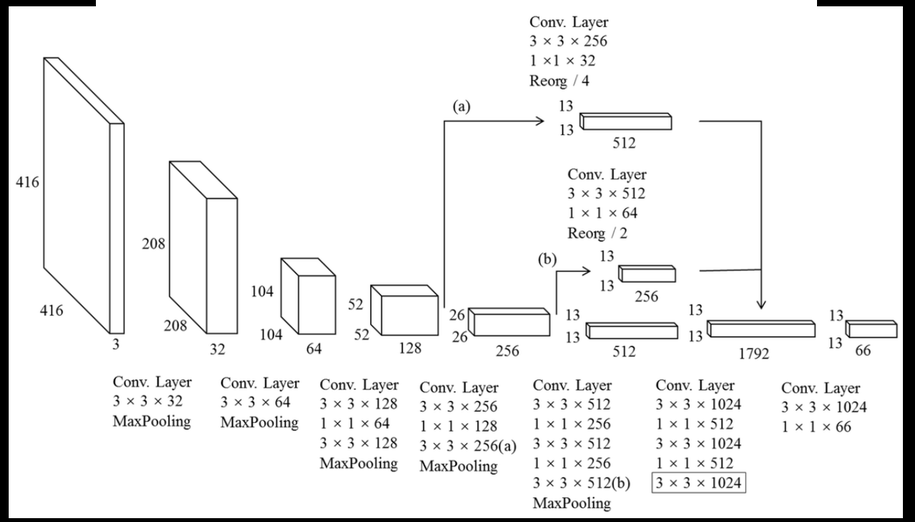


Architecture

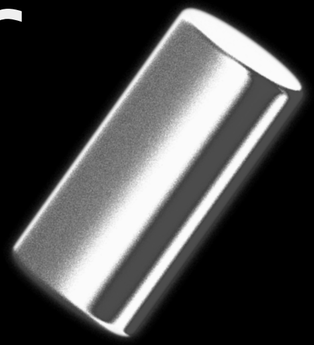
To solve the problems of complexity and accuracy the authors propose a new classification model called **Darknet-19** to be used as a backbone for YOLOv2.

Darknet-19 has 19 convolutional layers and 5 max-pooling layers with 11 more layers for object detection. It achieved 91.2% top-5 accuracy on ImageNet which is better than VGG (90%) and YOLO network(88%) **for classification**

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			



High Resolution Classifier



General way in which object detection works is, the model is pretrained on imagenet for classification.

Then for detection, the network is resized to higher resolution especially to detect smaller objects in a scene.

The original YOLO was trained as follow:

i-They trained the classifier network at 224×224 input size.

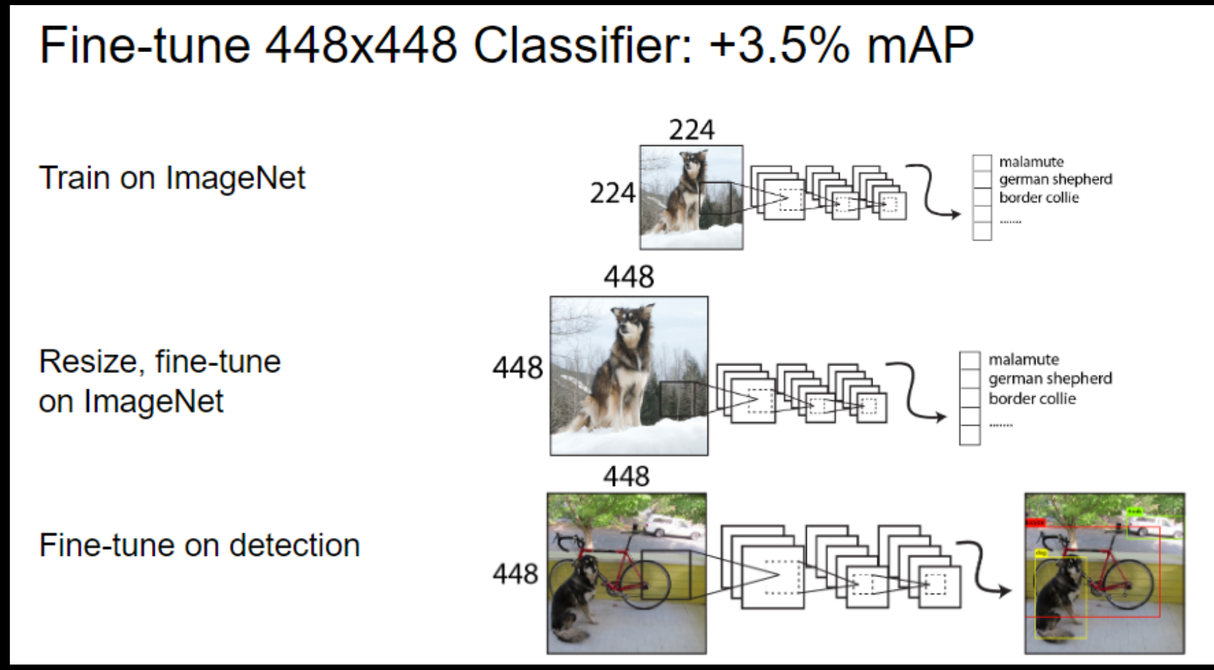
ii-Then they increased the resolution to 448 for detection.

High Resolution Classifier

This means when switching to detection the network has to simultaneously switch to learning object detection and adjust to the new input resolution.

While for YOLOv2 they initially trained the model on images at 224×224, then they fine tune the classification network at the full 448×448 resolution for 10 epochs on ImageNet before training for detection.

This gives the network time to adjust its filters to work better on higher resolution input. This high-resolution classification network gives an increase of almost 4% mAP.

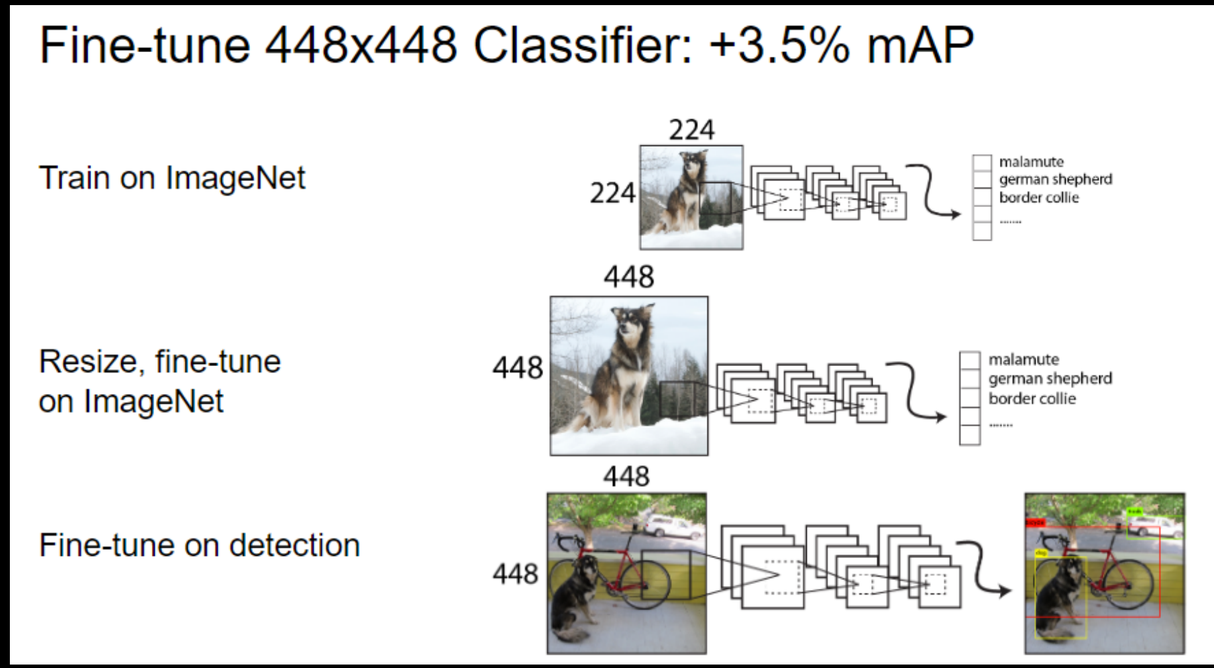


Training

The model was first trained for classification then it was trained for detection.

1 - **Classification**: they trained Darknet-19 network on the standard ImageNet 1000 class classification dataset with input shape 224x224 for 160 epochs.

After that, they fine-tune the network at a large input size 448x448 for 10 epochs. This gives a top-1 accuracy of 76.5% and a top-5 accuracy of 93.3%



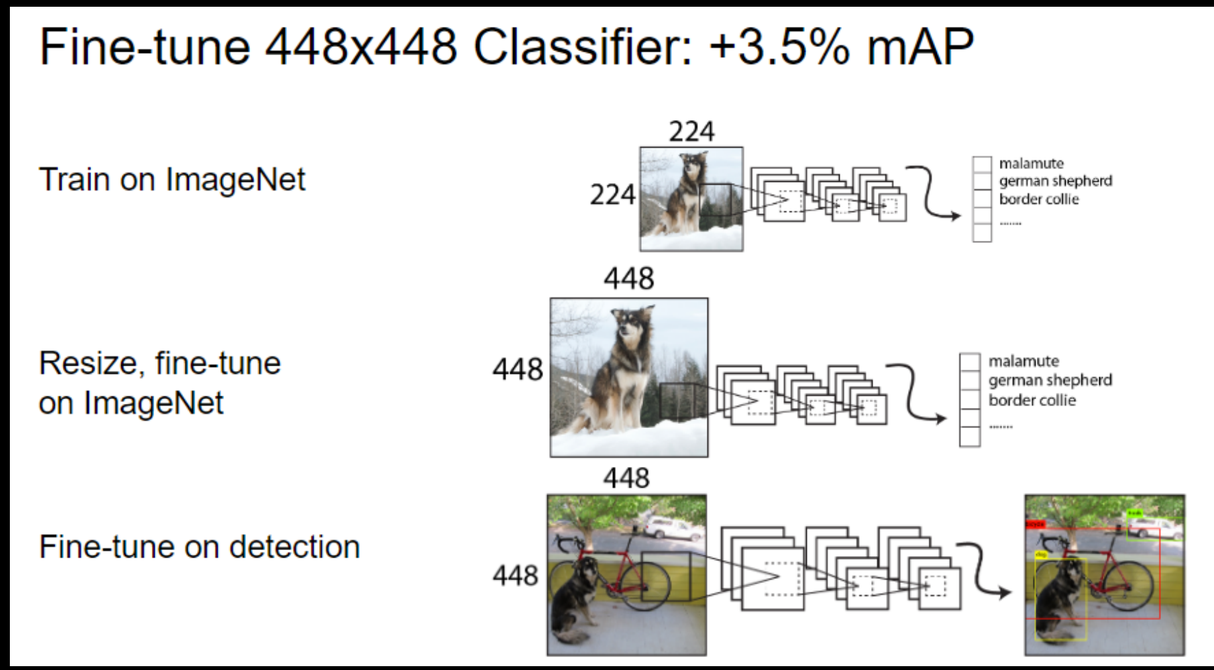
Training

The model was first trained for classification then it was trained for detection.

2 - Detection: After training for classification they removed the last convolutional layer from Darknet-19 and instead they added three 3 × 3 convolutional layers and a 1x1 convolutional layer with the number of outputs they need for detection(13x13x125).

Also, a passthrough layer was added so that their model can use **fine grain features** from previous layers.

Then they trained the network for 160 epochs on detection datasets (VOC and COCO datasets).



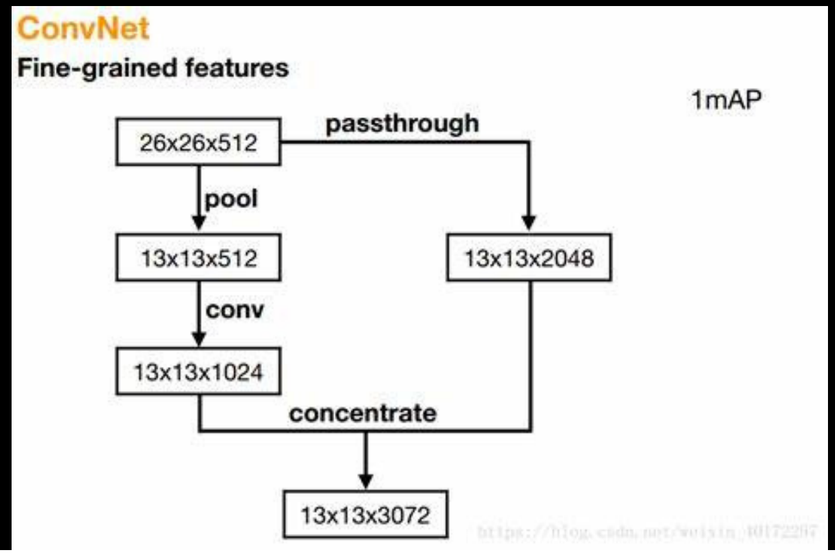
Fine Grained Features

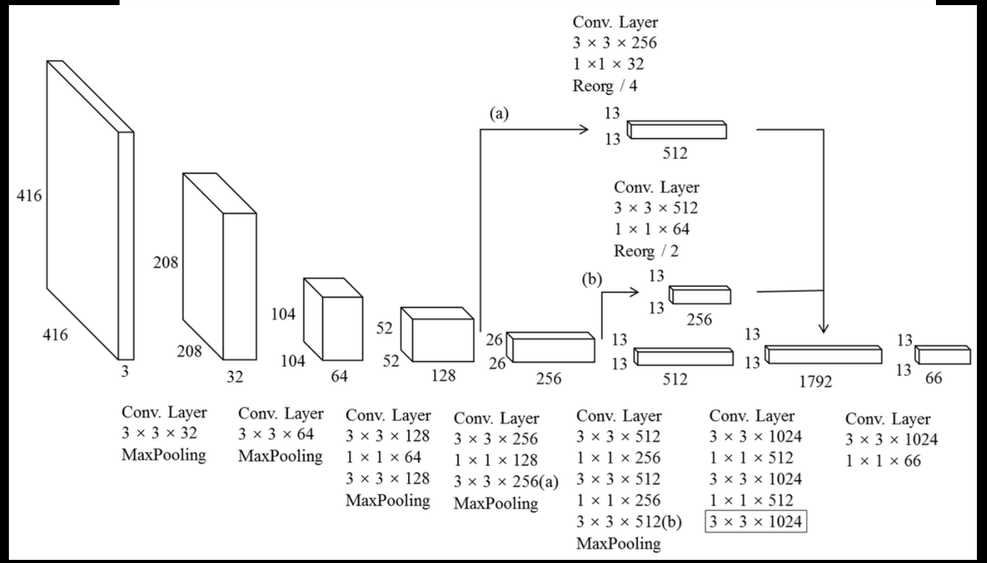
YOLOv2 predicts the detections using the 13 x 13 feature map.

This is sufficient for identifying large objects but not smaller objects.

To better localize smaller objects, a passthrough layer that takes features from an earlier layer at 26 x 26 resolution is concatenated with the lower resolution features.

This gives a 1% increase in performance.





Multiscale Training

To make YOLOv2 robust to running on images of different sizes they trained the model for ,different input sizes

Since the model uses only convolutional and pooling layers
.the input can be resized on the fly.

YOLOv1 used 448 x 448 resolution for the input. In YOLOv2, they resize the input image randomly to different resolutions between 320 x 320 to 608 x 608 (the resolution is always a multiple of 32).

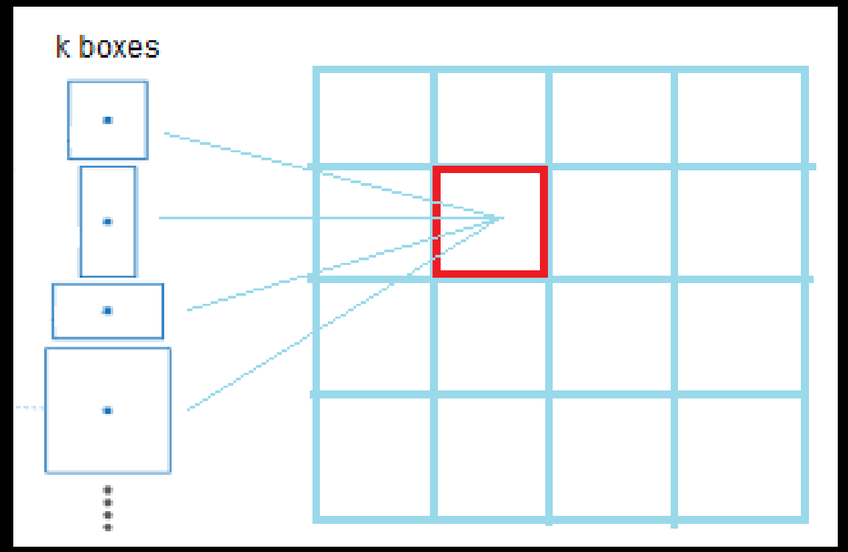
This multi-scale training can be thought of like augmentation, it forces the network to learn to predict well across a variety of input dimensions. This increased the mAP by 1.5%.

Anchor Boxes

YOLO (v1) tries to assign the object to the grid cell that contains the middle of the object.

Using this idea the red cell in the image to the right must detect both the man and his necktie, but since any grid cell can only detect one object, a problem will arise here.

To solve this, the authors tried to allow the grid cell to detect more than one object using k bounding box.



Anchor Boxes

There are two ways of predicting the bounding boxes-

1. Directly predicting the bounding box of the object
2. Using a set of pre-defined bounding boxes (Anchor box) to predict the actual bounding box of the object.

YOLOv1 predicts the coordinates of bounding boxes directly using fully connected layers on top of the convolutional feature extractor.

But, it makes a significant amount of localisation error. It is easier to predict the offset based on anchor boxes than to predict the co-ordinates directly.

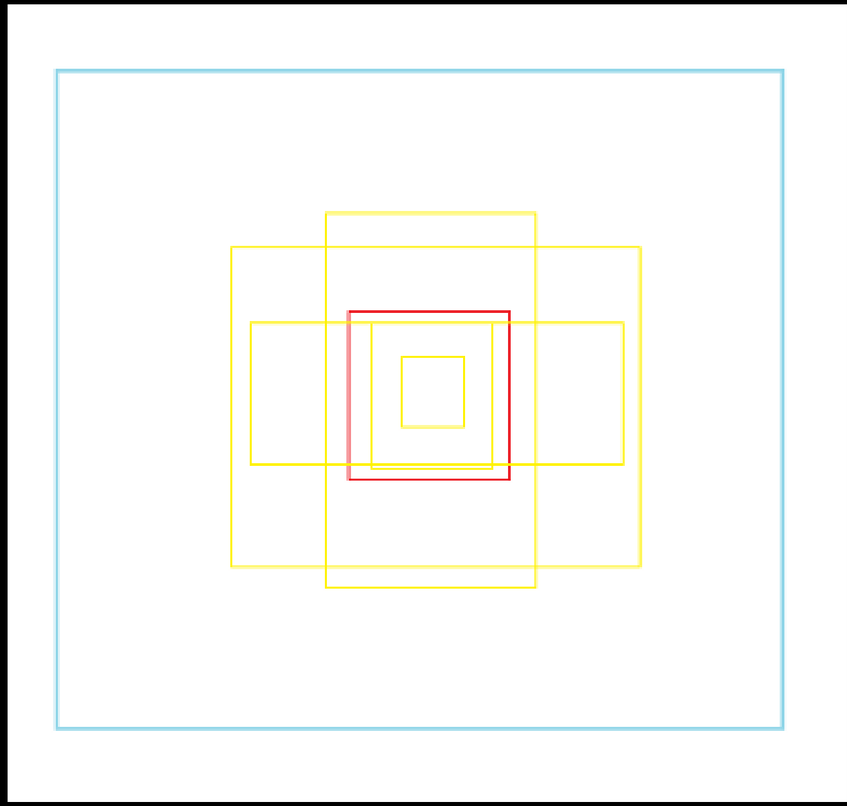
Anchor Boxes

In this image, we have a grid cell(red) and 5 anchor boxes(yellow) with different shapes (aspect ratios).

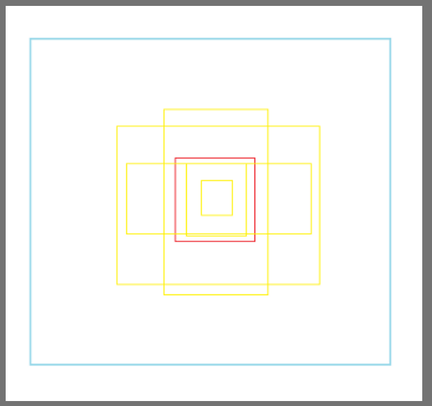
In the paper they called the anchor box a (prior box)

YOLOv2 tries to use the idea of anchor boxes but instead of picking the k anchor boxes by hand, it tries to find the best anchor box shapes to make it easier for the network to learn detection.

We can predict the bounding box relative to the anchor box instead of predicting the box relative to the image. Using this idea it will be easier for the network to learn.



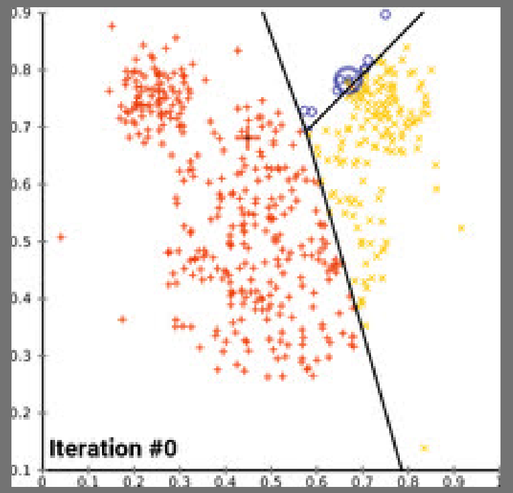
Anchor Boxes



In this image, the 5 red boxes represent the average dimensions and locations of objects in VOC 2007 dataset.

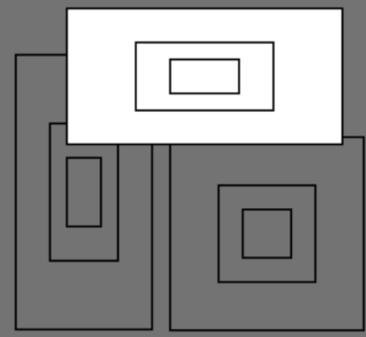
Someone may ask how and why they chose these 5 boxes?

Instead of using predefined anchor boxes, they looked at the bounding boxes in training data (VOC 2007, COCO) and run k-means clustering on the **training set bounding boxes** for **various values of k** and **plot the average IOU** with the closest centroid.

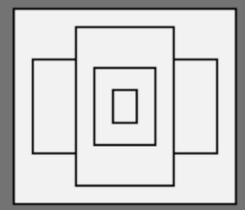


But instead of using Euclidean distance, they used IOU between the bounding box and the centroid. This resulted in Dimension clusters

Anchor Boxes



Dimension Clusters

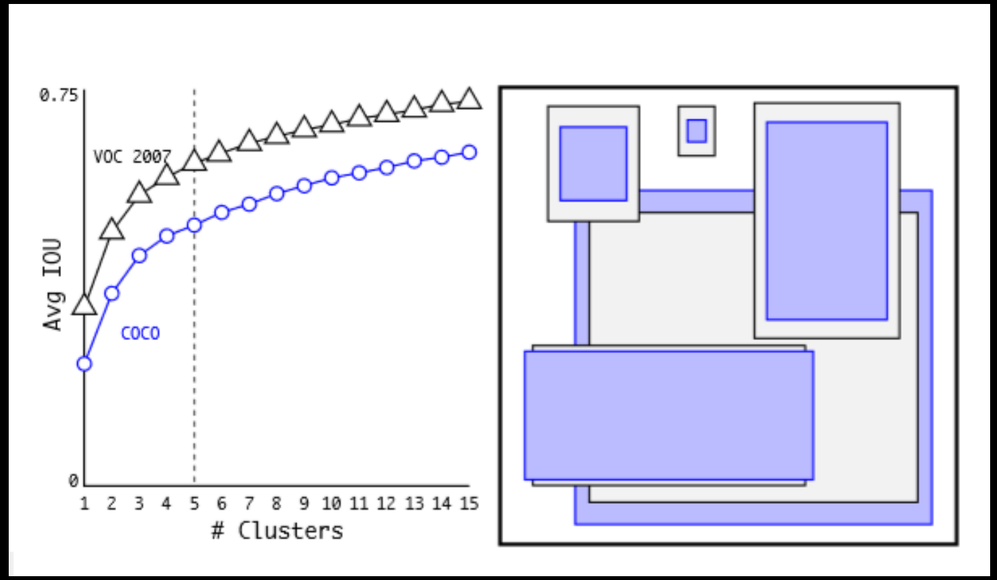


Anchor Boxes

They chose k = 5 as a good trade-off between model complexity and high recall.

When they ran K-means clustering on VOC 2008 and COCO training data, this is what they obtained the image to the right.

The graph on the right shows how much the Dimension clusters overlap with the training data's bounding boxes without any manipulation. Using these cluster centers helped increase mAP by 5%.



Left: Avg. IOU vs Number of clusters.

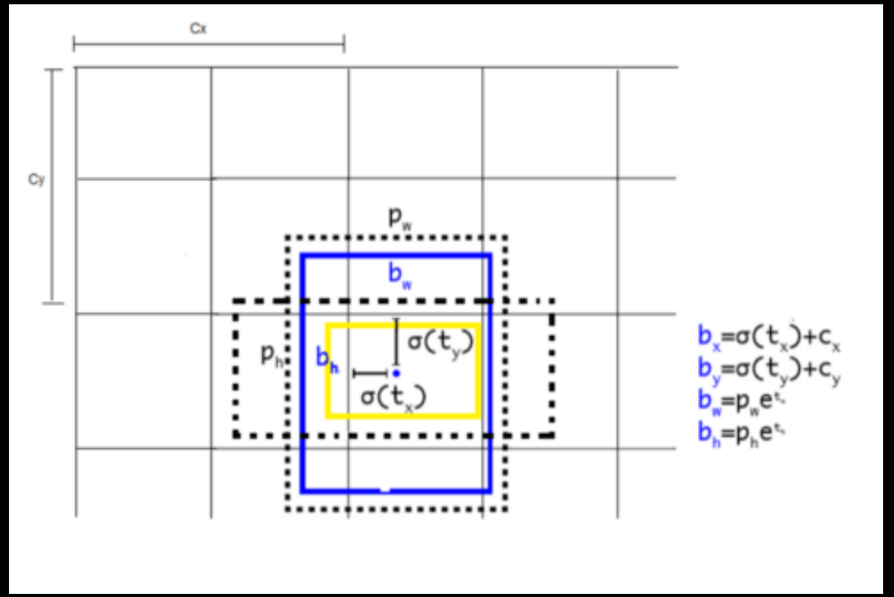
Right: Dimension clusters obtained from the train images.

Direct location prediction

When they used **anchor boxes** with **YOLO**, they encountered the issue of **model instabilities** especially during **initial iterations**.

Instead of predicting offsets, YOLOv2 follows the approach of YOLOv1 and predicts location coordinates relative to the location of the grid cell.

This bounds the ground truth to fall between 0 and 1. The network predicts 5 bounding boxes for each cell. It predicts 5 coordinates for each bounding box, tx, ty, tw, th, and to.

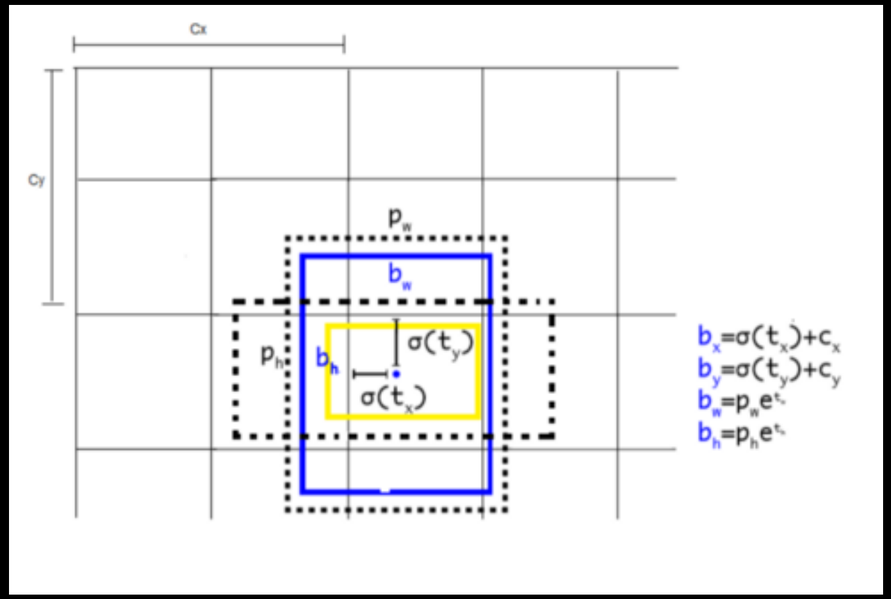


$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

Direct location prediction

If the cell is offset from the top left corner of the image by (cx,cy) and the bounding box prior(anchor box) has width and height pw, ph, and the predictions correspond to b_x, b_y, b_w, b_h.

For example, if we use 2 anchor boxes the grid cell (2,2) in the image will output 2 boxes (the blue and the yellow boxes). Let the black dotted boxes represent the 2 anchor boxes for that cell.

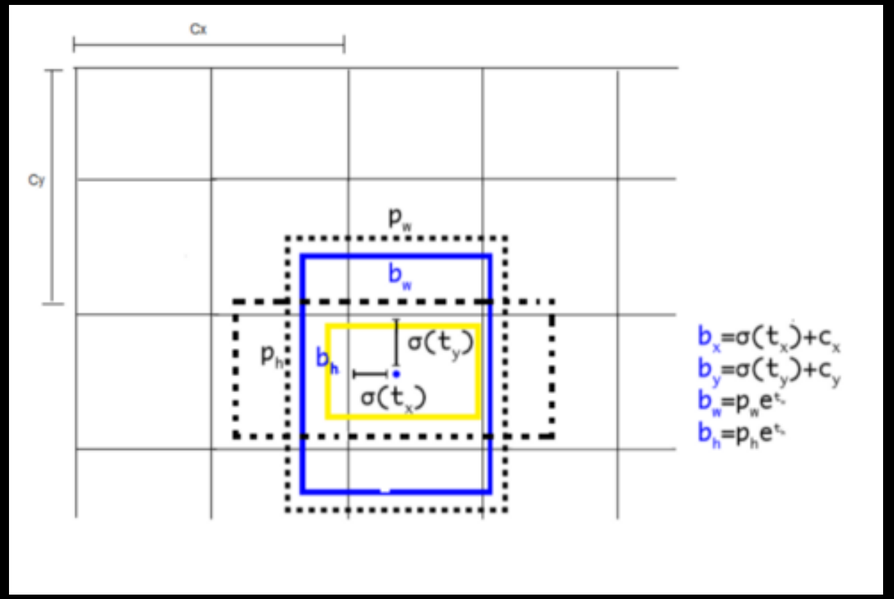


$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

Direct location prediction

Now consider only the blue box, instead of assigning the predicted blue box to the grid cell only as in YOLOv1, YOLOv2 assigns the blue box not only to the grid cell but also to one of the anchor boxes and that will be the one that has the highest IOU with the ground truth box.

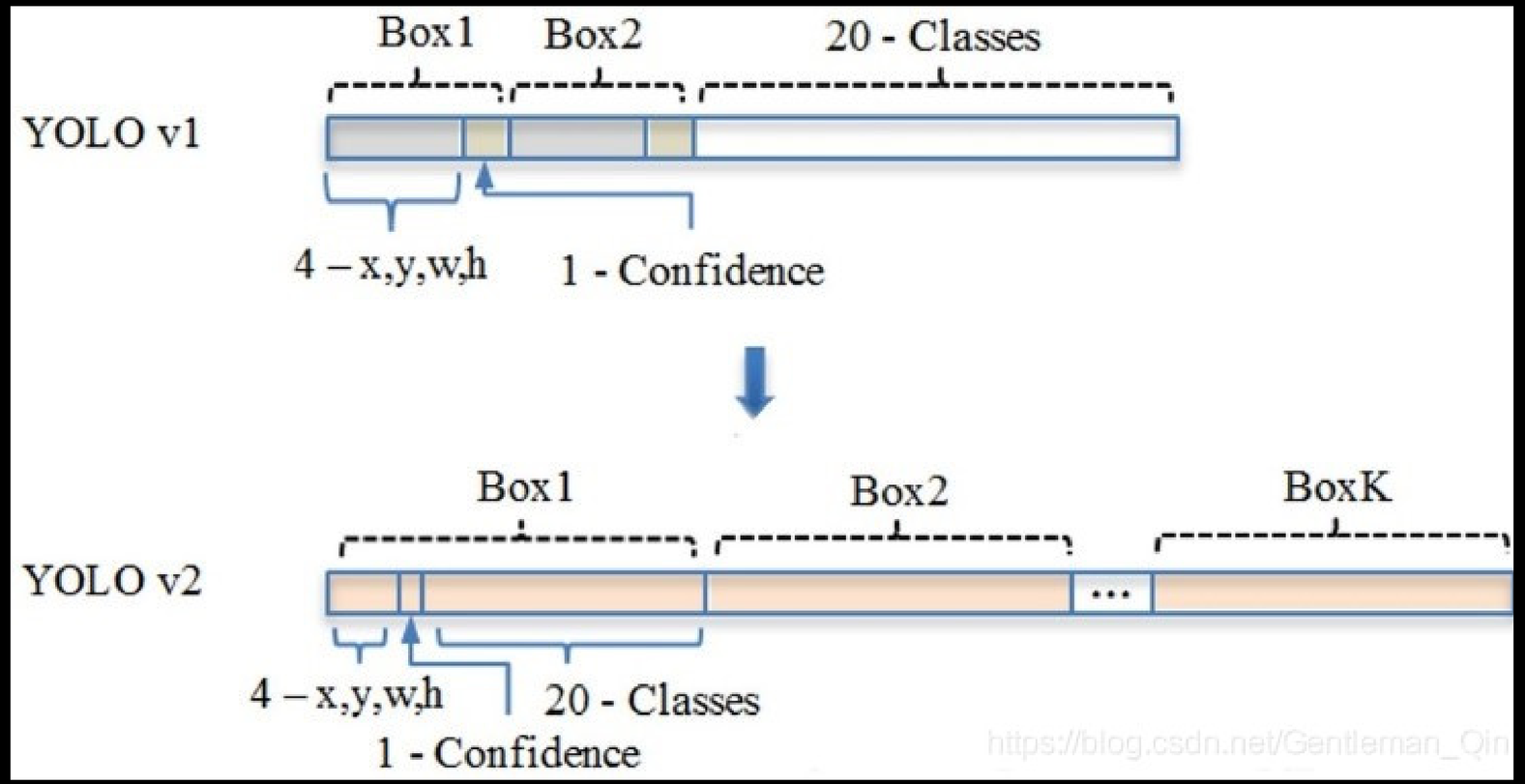
YOLOv2 uses the equations on the right to assign the blue box to the grid cell and the anchor box.



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

Output shape

YOLOv2 output shape is 13x13x(kx(1+4+20)) where k is the number of anchor boxes, and 20 is the number of classes. For k=5 the output shape will be 13x13x125.



Loss Function

This function defines the loss function for an **iteration t**.

If a **bounding box** doesn't have any object then its confidence of objectness need to be reduced and it is represented as first loss term.

As the bounding box coordinates prediction need to align with our prior information, a loss term reducing the difference between prior and the predicted is added for few iterations (t < 12800).

If a bounding box k is responsible for a truth box, then the predictions need to be aligned with the truth values which is represented as the third loss term. The values are the pre-defined weightages for each of the loss terms.

$$
\begin{aligned}
loss_t = \sum_{i=0}^{W}\sum_{j=0}^{H}\sum_{k=0}^{A} \quad & 1_{Max\ IOU < Thresh}\ \lambda_{noobj} * \left(-b_{ijk}^{o}\right)^2 \\
& + 1_{\ t<12800}\ \lambda_{prior} * \sum_{r\ \epsilon\ (x,y,w,h)} \left(prior_k^r - b_{ijk}^r\right)^2 \\
& + 1_k^{truth} (\lambda_{coord} * \sum_{r\ \epsilon\ (x,y,w,h)} \left(truth^r - b_{ijk}^r\right)^2 \\
& + \lambda_{obj} * \left(IOU_{truth}^k - b_{ijk}^o\right)^2 \\
& + \lambda_{class} * \left(\sum_{c=1}^{C} \left(truth^c - b_{ijk}^c\right)^2\right))
\end{aligned}
$$

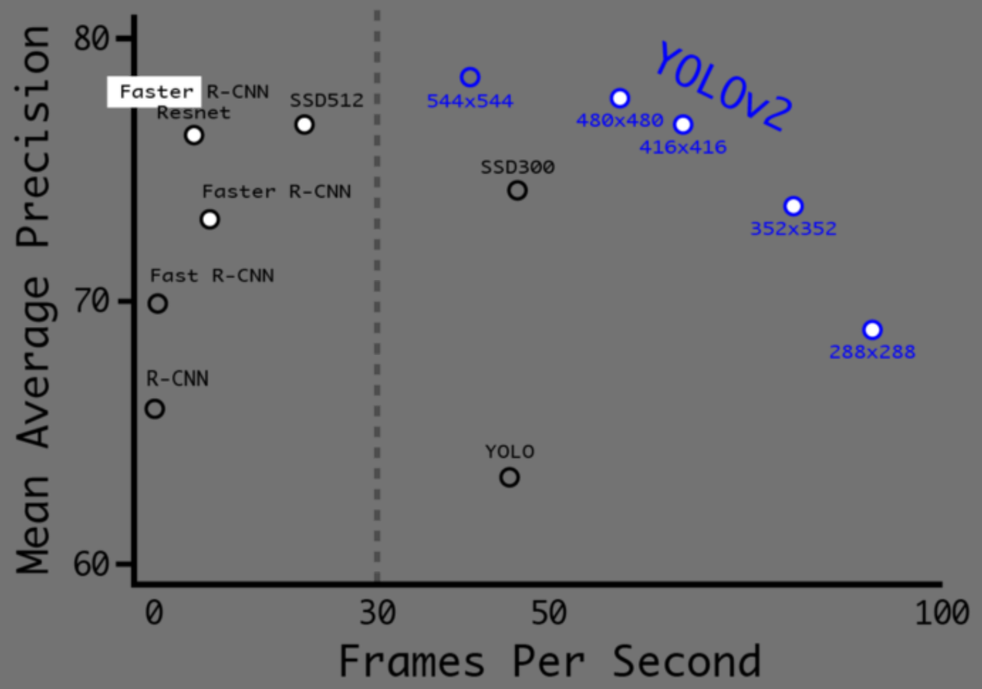
YOLOv2

Comparison to Other Detection Systems

Comparison to Other Detection Systems

YOLOv2 is **state-of-the-art and faster** than other detection systems across **a variety of detection datasets**. Furthermore, it can be run at a **variety of image sizes** to provide a **smooth trade-off between speed and accuracy**.

The graph and table below show how different methods perform with respect to **precision and speed** on **VOC 2007 dataset** and **VOC 2007 + 2012 dataset** respectively.



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

YOLO9000

Comparison to Other Detection Systems

YOLO9000

Sometimes we need a model that can detect **more than 20 classes**, and that is what **YOLO9000** does. It is a **real-time framework** for detecting more than **9000 object categories** by jointly **optimizing detection and classification**.

As we mentioned previously, **YOLOv2** was trained for **classification** then for **detection**. This is because the **dataset for classification** –**which contains one object**– **is different from the dataset for detection**. In **YOLOv2**, the authors propose a mechanism for **jointly training on classification and detection data**.

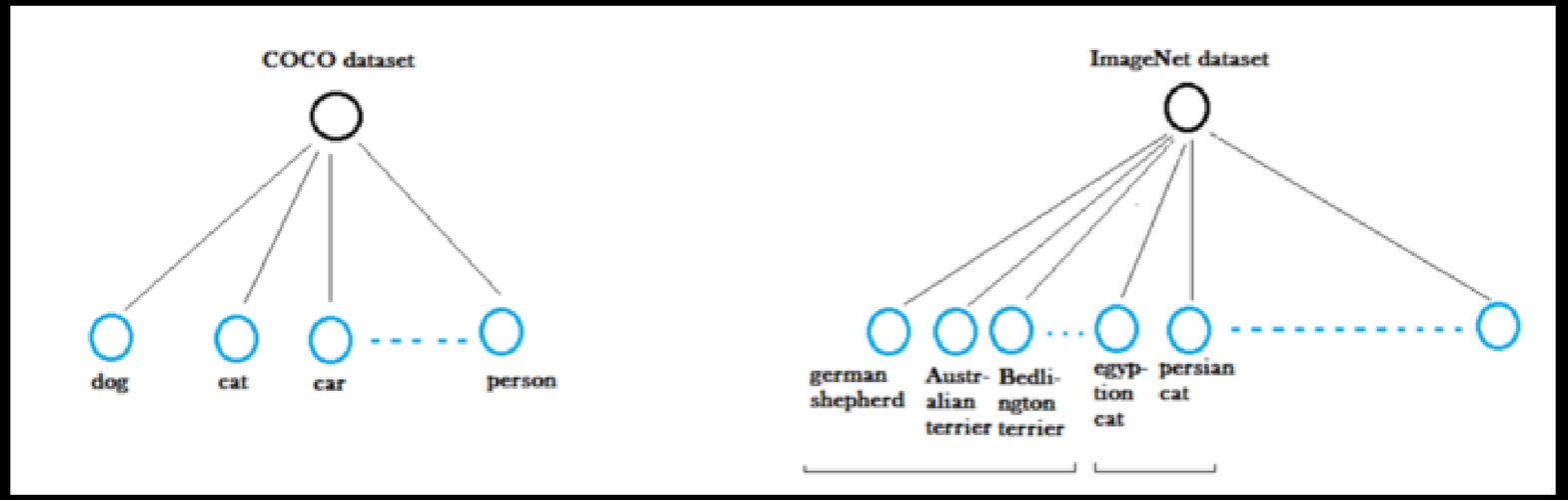
During training, they **mix images from both detection and classification datasets**. When the network sees an **image labeled for detection**, we can **backpropagate** based on **the full YOLOv2 loss function**. When it sees a **classification image** we only **backpropagate loss** from the **classification specific parts** of the architecture.

YOLO9000

The idea of mixing detection and classification data faces a few challenges:

1- **Detection datasets** are **small** compared to **classification datasets**.

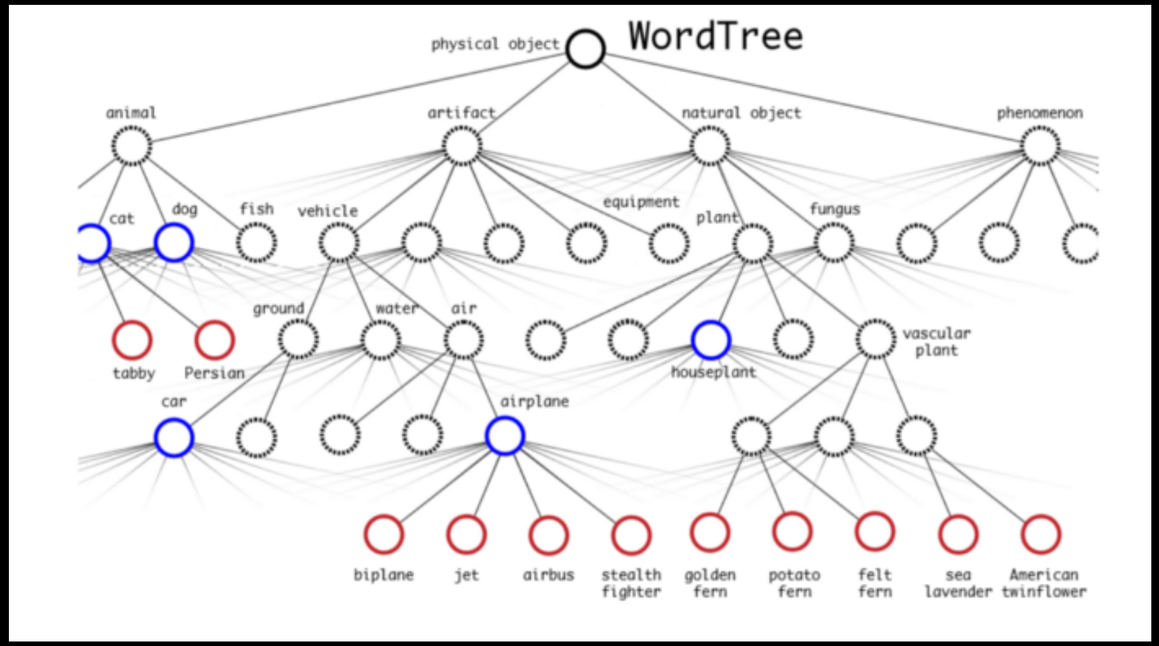
2- **Detection datasets** have only common **objects** and **general labels**, like **"dog"** or **"boat"**, while **Classification datasets** have a much **wider and deeper range of labels**. For example, **ImageNet dataset** has more than a **hundred breeds of dogs** like **"german shepherd"** and **"Bedlington terrier."**



YOLO9000

To merge these two datasets the authors created a **hierarchical model of visual concepts** and called it **WordTree**.

As we see, all the classes are under the root **(physical object)**. They trained the **Darknet-19** model on **WordTree** .They extracted the **1000 classes** of **ImageNet dataset** from **WordTree** and added to it all the **intermediate nodes**, which expands the label space from **1000 to 1369** and called it **WordTree1k**. Now the **size of the output layer** of **darknet-19** became **1369** instead of **1000**.

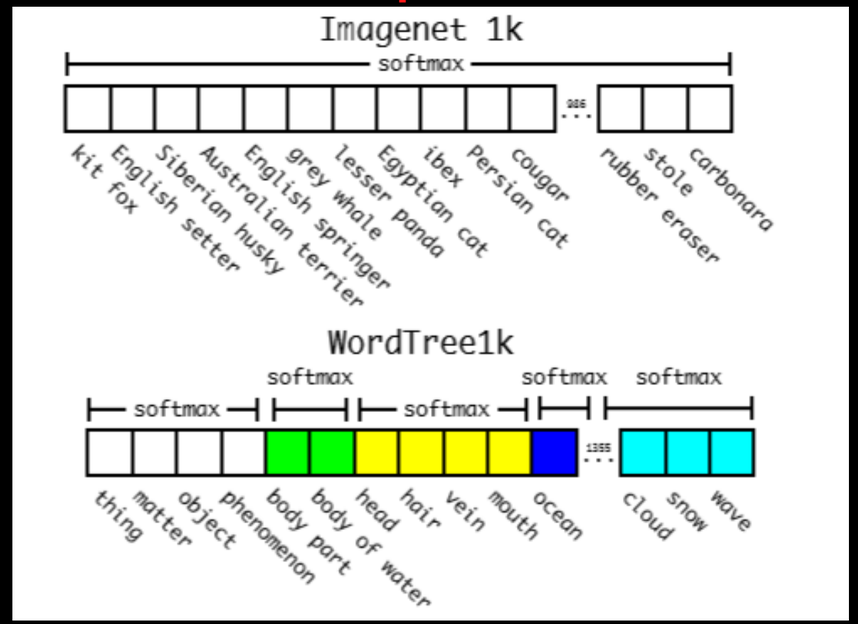


YOLO9000

For these **1369 predictions**, we don't compute **one softmax**, but we compute **a separate softmax** overall **synsets** that are **hyponyms** of the **same concept**.

Despite adding **369 additional concepts** Darknet-19 achieves **71.9% top-1 accuracy** and **90.4% top-5 accuracy**.

The detector predicts a **bounding box** and the **tree of probabilities**, but since we use **more than one softmax** we need to **traverse the tree to find the predicted class**.

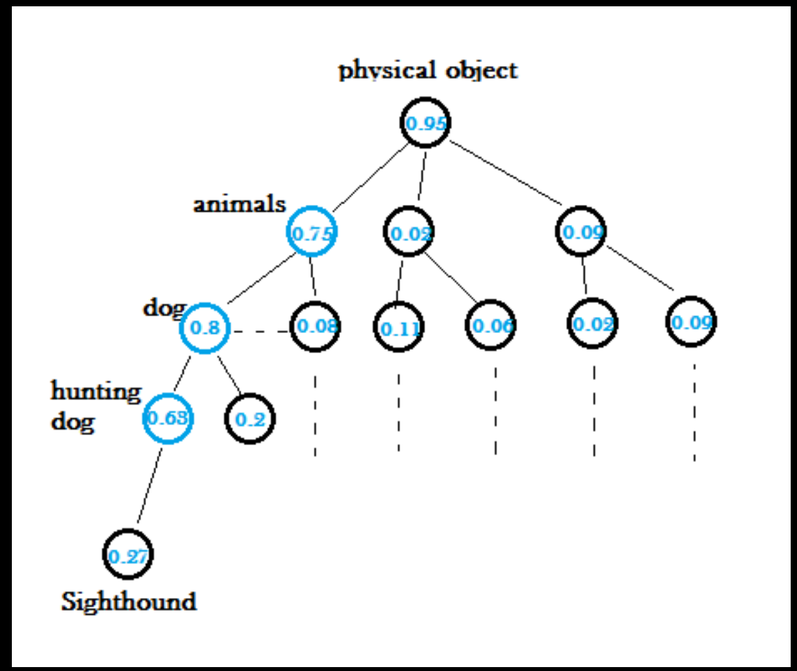


YOLO9000

We traverse the tree from **top to down**, taking the **highest confidence path** at **every split** until we reach a node with probability < threshold-probability then we predict that object class.

For example, if the input image contains a **dog**, the tree of probabilities will be like this tree below.

Instead of assuming **every image has an object**, we use **YOLOv2's objectness predictor** to give us the value of Pr(physical object), which is the **root of the tree**.

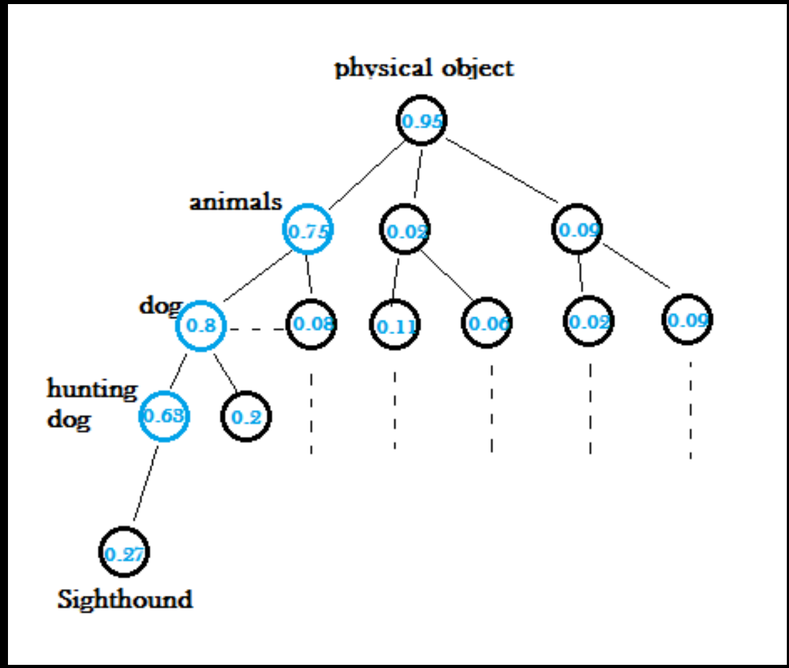


YOLO9000

The model outputs a **softmax for each branch level**. We choose the **node with the highest probability (if it is higher than a threshold value)** as we move from **top to down**. The **prediction** will be the node where we **stop**.

In the tree above, the model will go through **physical object => dog=>hunting dog**. It will stop at **'hunting dog'** and do not go down to **sighthound** (a type of hunting dogs) because its **confidence** is less than the confidence threshold value, so the model will predict **hunting dog** not **sighthound**.

Performing **classification** in this manner also has some benefits. Performance degrades gracefully on **new or unknown object categories**. For example, if the network sees a **picture of a dog** but it is uncertain **which type of dog it is**, it will **stop at the dog** with **high confidence and the output will be (dog)**.



YOLO9000

The combined dataset was created using the **COCO detection dataset** and **the top 9000 classes from the full ImageNet release**. **YOLO9000** uses **three priors(anchor boxes) instead of 5** to limit the **output size**. It learns to find objects in images using the **detection data in the COCO dataset** and it learns to **classify a wide variety of these objects** using **data from the ImageNet dataset**.

When the network sees a **detection image**, we **backpropagate loss as normal**. When it sees a **classification image** we only **backpropagate classification loss**. **YOLO9000** uses the base **YOLOv2 architecture** but only **3 priors instead of 5** to **limit the output size**.

Since **COCO** does not have a **bounding box label** for many categories, **YOLO9000** struggles to model some categories like **“sunglasses”** or **“swimming trunks.”** **YOLO9000** gets **19.7 mAP** overall with **16.0 mAP** on the **disjoint 156 object classes** that it has never seen any labelled detection data for when evaluated on the **ImageNet detection task**.