Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL *VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.*

1. Introduction

General purpose object detection should be fast, accurate, and able to recognize a wide variety of objects. Since the introduction of neural networks, detection frameworks have become increasingly fast and accurate. However, most detection methods are still constrained to a small set of objects. → 문제점

Current object detection datasets are limited compared to datasets for other tasks like classification and tagging. The most common detection datasets contain thousands to hundreds of thousands of images with dozens to hundreds of tags [3] [10] [2]. Classification datasets have millions of images with tens or hundreds of thousands of categories [20] [2].

We would like detection to scale to level of object classification. However, labelling images for detection is far more expensive than labelling for classification or tagging (tags are often user-supplied for free). Thus we are unlikely to see detection datasets on the same scale as classification datasets in the near future.

We propose a new method to harness the large amount of classification data we already have and use it to expand the scope of current detection systems. Our method uses a hierarchical view of object classification that allows us to combine distinct datasets together.

We also propose a joint training algorithm that allows us to train object detectors on both detection and classification data. Our method leverages labeled detection images to learn to precisely localize objects while it uses classification images to increase its vocabulary and robustness.

Using this method we train YOLO9000, a real-time object detector that can detect over 9000 different object categories. First we improve upon the base YOLO detection system to produce YOLOv2, a state-of-the-art, real-time detector. Then we use our dataset combination method and joint training algorithm to train a model on more than 9000 classes from ImageNet as well as detection data from COCO.

All of our code and pre-trained models are available online at http://pjreddie.com/yolo9000/.

2. Better

YOLO suffers from a variety of shortcomings relative to state-of-the-art detection systems. Error analysis of YOLO compared to Fast R-CNN shows that YOLO makes a significant number of localization errors. Furthermore, YOLO has relatively low recall compared to region proposal-based methods. Thus we focus mainly on improving recall and localization while maintaining classification accuracy.

Computer vision generally trends towards larger, deeper networks [6] [18] [17]. Better performance often hinges on training larger networks or ensembling multiple models together. However, with YOLOv2 we want a more accurate detector that is still fast. Instead of scaling up our network, we simplify the network and then make the representation easier to learn. We pool a variety of ideas from past work with our own novel concepts to improve YOLO's performance. A summary of results can be found in Table 2.

Batch Normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization [7]. By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

High Resolution Classifier. All state-of-the-art detection methods use classifier pre-trained on ImageNet [16]. Starting with AlexNet most classifiers operate on input images smaller than 256×256 [8]. The original YOLO trains the classifier network at 224×224 and increases the resolution to 448 for detection. This means the network has to simultaneously switch to learning object detection and adjust to the new input resolution.

For YOLOv2 we first fine tune the classification network at the full 448×448 resolution for 10 epochs on ImageNet. This gives the network time to adjust its filters to work better on higher resolution input. We then fine tune the resulting network on detection. This high resolution classification network gives us an increase of almost 4% mAP.

(1) localization error가 많다.
→ bbox 위치를 제대로 포착하지 못함.

(2) recall이 낮다.
→ region proposal 방법보다 recall이 낮음!

더 나은 성능을 얻기 위해 신경망의 크기는 키우거나 모델을 앙상블 하는 방법이 주로 이용됨!

저자는 detector가 빠르길 원함!

1. Batch Normalization

Batch Normalization은 다른 regularization의 필요성을 없애고 신경망을 더 빠르게 수렴하도록 함.
→ BN을 활용하여 mAP 2% 상승, Drop out 제거

2. High Resolution Classifier

YOLO v1은 224×224 학습 → 448 detection = 성능저하
YOLO v2은 448×448로 fine tuning
→ 고해상도 학습 = 4% mAP 증가

Convolutional With Anchor Boxes. YOLO predicts the coordinates of bounding boxes directly using fully connected layers on top of the convolutional feature extractor. Instead of predicting coordinates directly Faster R-CNN predicts bounding boxes using hand-picked priors [15]. Using only convolutional layers the region proposal network (RPN) in Faster R-CNN predicts offsets and confidences for anchor boxes. Since the prediction layer is convolutional, the RPN predicts these offsets at every location in a feature map. Predicting offsets instead of coordinates simplifies the problem and makes it easier for the network to learn.

We remove the fully connected layers from YOLO and use anchor boxes to predict bounding boxes. First we eliminate one pooling layer to make the output of the network's convolutional layers higher resolution. We also shrink the network to operate on 416 input images instead of 448×448. We do this because we want an odd number of locations in our feature map so there is a single center cell. Objects, especially large objects, tend to occupy the center of the image so it's good to have a single location right at the center to predict these objects instead of four locations that are all nearby. YOLO's convolutional layers downsample the image by a factor of 32 so by using an input image of 416 we get an output feature map of 13×13.

When we move to anchor boxes we also decouple the class prediction mechanism from the spatial location and instead predict class and objectness for every anchor box. Following YOLO, the objectness prediction still predicts the IOU of the ground truth and the proposed box and the class predictions predict the conditional probability of that class given that there is an object.

Using anchor boxes we get a small decrease in accuracy. YOLO only predicts 98 boxes per image but with anchor boxes our model predicts more than a thousand. Without anchor boxes our intermediate model gets 69.5 mAP with a recall of 81%. With anchor boxes our model gets 69.2 mAP with a recall of 88%. Even though the mAP decreases, the increase in recall means that our model has more room to improve.

Dimension Clusters. We encounter two issues with an

3. Convolutional with Anchor Box

YOLO v2는 FC Layer를 제거하고 anchor Box를 활용하여 bbox를 예측.

→ 1×1 conv Layer로 예측.

Bbox를 예측하는 것 보다 사전에 정의한 Anchor box에서 offset을 예측하는 것이 간단.

한 개의 pooling Layer를 제거.

→ Conv Layer의 출력을 고해상도로 만듦.

input image를 448→416 변경.

448 → downsampling → 14×14

416 → downsampling → 13×13

output이 홀수×홀수 인게 좋음!

→ why? 14×14인 경우 중앙의 근접한 grid cell이 4개이기 때문에! 13×13로 중앙에 하나의 grid cell이 오도록!

anchor box를 이용하여 mAP 감소. recall 증가.

without anchor box → 69.5 mAP, 81% recall

with anchor box → 69.2 mAP, 88% recall

object detection task에서 recall이 높다는 것은 모델이 실제 객체의 위치를 예측한 비율이 높음을 의미함. YOLO v1이 recall 값이 낮은 이유는 region proposal 기반의 모델에 비해 이미지 당 상대적으로 적은 수의 bbox를 예측하기 때문. 하지만 V2에서 anchor box를 통해 더 많은 수의 bbox를 예측하면서 실제 객체의 위치를 보다 잘 커버하게 되고 이로 인해서 recall 상승!

Dimension Clusters. We encounter two issues with anchor boxes when using them with YOLO. The first is that the box dimensions are hand picked. The network can learn to adjust the boxes appropriately but if we pick better priors for the network to start with we can make it easier for the network to learn to predict good detections.

Instead of choosing priors by hand, we run k-means clustering on the training set bounding boxes to automatically find good priors. If we use standard k-means with Euclidean distance larger boxes generate more error than smaller boxes. However, what we really want are priors that lead to good IOU scores, which is independent of the size of the box. Thus for our distance metric we use:

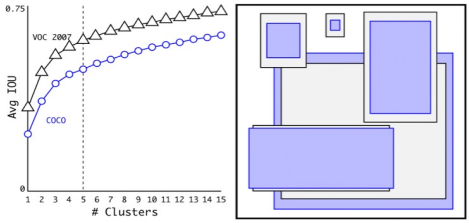


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

$$d(\text{box}, \text{centroid}) = 1 - \text{IOU}(\text{box}, \text{centroid})$$

We run k-means for various values of k and plot the average IOU with closest centroid, see Figure 2. We choose $k = 5$ as a good tradeoff between model complexity and high recall. The cluster centroids are significantly different than hand-picked anchor boxes. There are fewer short, wide boxes and more tall, thin boxes.

We compare the average IOU to closest prior of our clustering strategy and the hand-picked anchor boxes in Table 1. At only 5 priors the centroids perform similarly to 9 anchor boxes with an average IOU of 61.0 compared to 60.9. If we use 9 centroids we see a much higher average IOU. This indicates that using k-means to generate our bounding box starts the model off with a better representation and makes the task easier to learn.

Box Generation	#	Avg IOU
Cluster SSE	5	58.7
Cluster IOU	5	61.0
Anchor Boxes [15]	9	60.9
Cluster IOU	9	67.2

Table 1: Average IOU of boxes to closest priors on VOC 2007. The average IOU of objects on VOC 2007 to their closest, unmodified prior using different generation methods. Clustering gives much better results than using hand-picked priors.

신경망은 Anchor box를 수정해서 bbox를 예측.
따라서 사전에 정의된 anchor box의 영향이 큰 수밖에 없고, 신중하게 anchor box를 선정해야 함.
→ 해결책: Anchor box를 training set의 바운딩 박스에 k-means clustering을 사용하여 선정.

일반적인 k-means clustering은 Euclidean distance를 이용함!

YOLO v2에서 Euclidean distance를 사용한 k-mean clustering을 이용하면 문제가 생김.
실제 bbox와 높은 IOU를 가진 anchor box를 선정해야 하는데 중심 좌표의 거리가 가장 짧은 것을 기준으로 anchor box를 선정하면 IOU가 낮은 anchor box를 선정할 수 있다는 것!
이를 해결하기 위해 IOU를 기준으로 k-mean clustering을 사용!

$$d(\text{box}, \text{centroid}) = 1 - \text{IOU}(\text{box}, \text{centroid})$$

Direct location prediction. When using anchor boxes with YOLO we encounter a second issue: model instability, especially during early iterations. Most of the instability comes from predicting the (x, y) locations for the box. In region proposal networks the network predicts values t_x and t_y and the (x, y) center coordinates are calculated as:

$$x = (t_x * w_a) - x_a$$
$$y = (t_y * h_a) - y_a$$

For example, a prediction of $t_x = 1$ would shift the box to the right by the width of the anchor box, a prediction of $t_x = -1$ would shift it to the left by the same amount.

This formulation is unconstrained so any anchor box can end up at any point in the image, regardless of what location predicted the box. With random initialization the model takes a long time to stabilize to predicting sensible offsets.

Instead of predicting offsets we follow the approach of YOLO and predict location coordinates relative to the location of the grid cell. This bounds the ground truth to fall between 0 and 1. We use a logistic activation to constrain the network's predictions to fall in this range.

The network predicts 5 bounding boxes at each cell in the output feature map. The network predicts 5 coordinates for each bounding box, t_x, t_y, t_w, t_h, and t_o. If the cell is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width and height p_w, p_h, then the predictions correspond to:

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

Since we constrain the location prediction the parametrization is easier to learn, making the network more stable. Using dimension clusters along with directly predicting the bounding box center location improves YOLO by almost 5% over the version with anchor boxes.

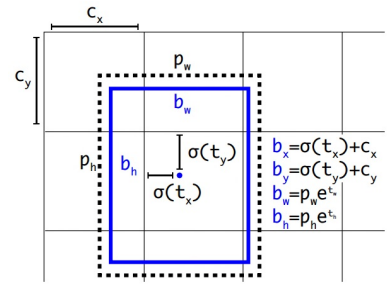


Figure 3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

anchor box를 활용한 bbox offset 예측법은 bbox의 위치를 제한하지 않아서 초기 학습시 불안정하다는 단점이 있음.

예측값에 범위를 두지 않으면 bbox가 이미지 어디에도 나타날 수 있음! 안정적으로 bbox를 예측하기 까지 많은 시간이 걸림

→ sigmoid function을 이용하여 offset의 범위를 0에서 1로 제한함.

모델은 각 bbox마다 5개의 값 $(t_x, t_y, t_w, t_h, t_o)$를 예측.

바운딩 박스 중심 좌표 예측값 (t_x, t_y)을 시그모이드 함수로 감싸 0~1 사이로 제한하여 안정적으로 학습.

이 방법과 bbox clustering을 활용하여 성능 5% 향상

Fine-Grained Features. This modified YOLO predicts detections on a 13×13 feature map. While this is sufficient for large objects, it may benefit from finer grained features for localizing smaller objects. Faster R-CNN and SSD both run their proposal networks at various feature maps in the network to get a range of resolutions. We take a different approach, simply adding a passthrough layer that brings features from an earlier layer at 26×26 resolution.

The passthrough layer concatenates the higher resolution features with the low resolution features by stacking adjacent features into different channels instead of spatial locations, similar to the identity mappings in ResNet. This turns the $26 \times 26 \times 512$ feature map into a $13 \times 13 \times 2048$ feature map, which can be concatenated with the original features. Our detector runs on top of this expanded feature map so that it has access to fine grained features. This gives a modest 1% performance increase.

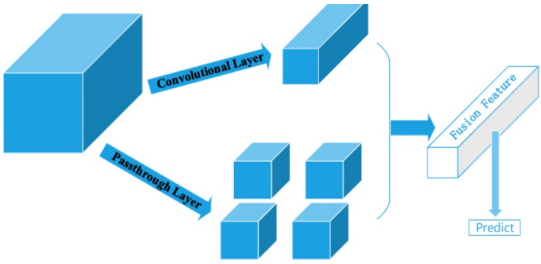


Figure 3. Passthrough Layer

YOLO v2는 13×13 feature map을 출력.

13×13의 크기는 큰 물체를 검출할때는 충분하지만, 작은 물체는 불충분!

Faster R-CNN과 SSD는 다양한 크기의 feature map에서 영역을 제안하여 문제를 해결.

YOLO v2. → passthrough layer를 추가하여 이전 layer의 26×26 feature map을 가져옴.

feature map의 크기가 다르므로 그냥 이어붙힐 수 없음.

26×26×512 ⟶ 13×13×(512×4)로 변환하여 붙힘! (figure 3.)

→ 26×26크기의 feature map의 고해상도 특징이 당겨짐으로 이 정보를 활용.

Multi-Scale Training. The original YOLO uses an input resolution of 448×448. With the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses a new image dimension size. Since our model downsamples by a factor of 32, we pull from the following multiples of 32: $\{320, 352, ..., 608\}$. Thus the smallest option is 320×320 and the largest is 608×608. We resize the network to that dimension and continue training.

YOLO v2는 다른 크기의 이미지로 부터 Robust를 갖기 위해 다양한 크기로 학습!

매 epoch 마다 {320, 352, ⋯, 608} 크기로 학습.

∴ 다양한 입력크기에도 예측을 잘함!

입력크기를 변경하여 구동해준 속도와 정확도를 trade off 할 수 있음!

3. Faster

We want detection to be accurate but we also want it to be fast. Most applications for detection, like robotics or self-

maximize performance we design YOLOv2 to be fast from the ground up.

Most detection frameworks rely on VGG-16 as the base feature extractor [17]. VGG-16 is a powerful, accurate classification network but it is needlessly complex. The convolutional layers of VGG-16 require 30.69 billion floating point operations for a single pass over a single image at 224×224 resolution.

The YOLO framework uses a custom network based on the Googlenet architecture [19]. This network is faster than VGG-16, only using 8.52 billion operations for a forward pass. However, it's accuracy is slightly worse than VGG-16. For single-crop, top-5 accuracy at 224×224, YOLO's custom model gets 88.0% ImageNet compared to 90.0% for VGG-16.

Darknet-19. We propose a new classification model to be used as the base of YOLOv2. Our model builds off of prior work on network design as well as common knowledge in the field. Similar to the VGG models we use mostly 3×3 filters and double the number of channels after every pooling step [17]. Following the work on Network in Network (NIN) we use global average pooling to make predictions as well as 1×1 filters to compress the feature representation between 3×3 convolutions [9]. We use batch normalization to stabilize training, speed up convergence, and regularize the model [7].

Our final model, called Darknet-19, has 19 convolutional layers and 5 maxpooling layers. For a full description see

Table 6. Darknet-19 only requires 5.58 billion operations to process an image yet achieves 72.9% top-1 accuracy and 91.2% top-5 accuracy on ImageNet.

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			

Table 6: Darknet-19.

Training for classification. We train the network on the standard ImageNet 1000 class classification dataset for 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 using the Darknet neural network framework [13]. During training we use standard data augmentation tricks including random crops, rotations, and hue, saturation, and exposure shifts.

As discussed above, after our initial training on images at 224×224 we fine tune our network at a larger size, 448. For this fine tuning we train with the above parameters but for only 10 epochs and starting at a learning rate of 10^{-3}. At this higher resolution our network achieves a top-1 accuracy of 76.5% and a top-5 accuracy of 93.3%.

Training for detection. We modify this network for detection by removing the last convolutional layer and instead adding on three 3×3 convolutional layers with 1024 filters each followed by a final 1×1 convolutional layer with the number of outputs we need for detection. For VOC we predict 5 boxes with 5 coordinates each and 20 classes per box so 125 filters. We also add a passthrough layer from the final $3 \times 3 \times 512$ layer to the second to last convolutional layer so that our model can use fine grain features.

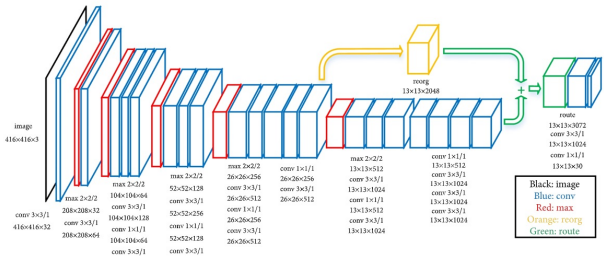
We train the network for 160 epochs with a starting learning rate of 10^{-3}, dividing it by 10 at 60 and 90 epochs.

We use a weight decay of 0.0005 and momentum of 0.9. We use a similar data augmentation to YOLO and SSD with random crops, color shifting, etc. We use the same training strategy on COCO and VOC.

빠른 속도를 지향하는 YOLO v2 에서는 새로운 신경망 Dark Net-19 사용!

VGG Model 에서 사용한 3x3 filter와 GoogLeNet에서 이용된 NIN (Network in Network) 기법을 사용!

3x3 filter 사이에 1x1 filter로 채널 축소와 FC Layer를 제거하여 연산량을 줄임!



4. Stronger

We propose a mechanism for jointly training on classification and detection data. Our method uses images labelled for detection to learn detection-specific information like bounding box coordinate prediction and objectness as well as how to classify common objects. It uses images with only class labels to expand the number of categories it can detect.

During training we mix images from both detection and classification datasets. When our network sees an image labelled 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.

This approach presents a few challenges. Detection datasets have only common objects and general labels, like "dog" or "boat". Classification datasets have a much wider and deeper range of labels. ImageNet has more than a hundred breeds of dog, including "Norfolk terrier", "Yorkshire terrier", and "Bedlington terrier". If we want to train on both datasets we need a coherent way to merge these labels.

Most approaches to classification use a softmax layer across all the possible categories to compute the final probability distribution. Using a softmax assumes the classes are mutually exclusive. This presents problems for combining datasets, for example you would not want to combine ImageNet and COCO using this model because the classes "Norfolk terrier" and "dog" are not mutually exclusive.

We could instead use a multi-label model to combine the datasets which does not assume mutual exclusion. This approach ignores all the structure we do know about the data, for example that all of the COCO classes are mutually exclusive.

YOLO 9000은 classification data와 detection data로 둘에서 학습시킨 것!

detection data로 bbox의 좌표와 objectness 같은 정보를 학습하고 classification data로 class label을 학습하여 검출할 수 있는 카테고리의 수를 확장함.

classification data가 입력되면 classification loss만 계산하고 detection data가 입력되면 원래 loss를 계산.

→ 어려움?

1. Detection dataset과 Classification dataset의 라벨이 다름.
   detection data set은 일반적이고 넓은 범위의 label (ex: dog, boat, cat ····)

2. 최종 class 확률을 구할때, 각 class가 상호 배타적으로 가정하면, 두 data set을 결합하면 detection data label인 dog와 classification data label인 Norfolk terrier는 상호 배타적이지 않음!
   ∴ multi-label model을 사용하여 상호배타를 가정하지 않음!

어떤 방법으로 두 data set을 결합하고 학습시켰는지?

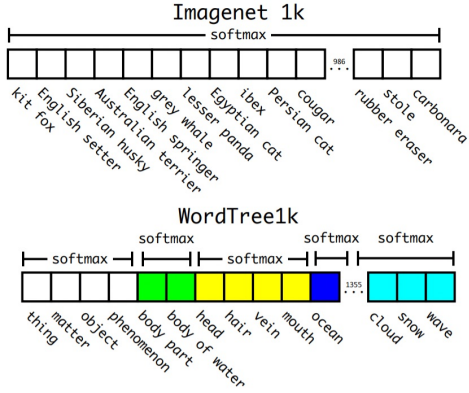
→

Hierarchical classification. ImageNet labels are pulled from WordNet, a language database that structures concepts and how they relate [12]. In WordNet, "Norfolk terrier" and "Yorkshire terrier" are both hyponyms of "terrier" which is a type of "hunting dog", which is a type of "dog", which is a "canine", etc. Most approaches to classification assume a flat structure to the labels however for combining datasets, structure is exactly what we need.

WordNet is structured as a directed graph, not a tree, because language is complex. For example a "dog" is both a type of "canine" and a type of "domestic animal" which are both synsets in WordNet. Instead of using the full graph structure, we simplify the problem by building a hierarchical tree from the concepts in ImageNet.

To build this tree we examine the visual nouns in ImageNet and look at their paths through the WordNet graph to the root node, in this case "physical object". Many synsets only have one path through the graph so first we add all of those paths to our tree. Then we iteratively examine the concepts we have left and add the paths that grow the tree by as little as possible. So if a concept has two paths to the root and one path would add three edges to our tree and the other would only add one edge, we choose the shorter path.

논문에서는 이러한 문제를 해결하기위해 ImageNet label 로부터 계층적인 트리(Hierarchical tree)인 WordTree를 구성하는 방법을 제안. WordTree에서 각 노드는 범주를 의미하고 하위 범주는 자식노드가 되는 구조를 가짐!



$$Pr(Yorkshireterrier) = Pr(Yorkshierterrier|terrier) * Pr(terrier|huntingdog) *\ldots* Pr(mammal|animal) * Pr(animal|physicalobject)$$

이러한 트리에서 특정 범주에 속할 확률은 루트 노드로부터 해당 범주의 노드까지의 조건부 확률의 곱으로 표현할 수 있음! 가령 임의로 들어온 이미지가 보더 테리어일 확률은 위와 같음.

The final result is WordTree, a hierarchical model of visual concepts. To perform classification with WordTree we predict conditional probabilities at every node for the probability of each hyponym of that synset given that synset. For example, at the "terrier" node we predict:

$$Pr(\text{Norfolk terrier}|\text{terrier})$$
$$Pr(\text{Yorkshire terrier}|\text{terrier})$$
$$Pr(\text{Bedlington terrier}|\text{terrier})$$
$$\ldots$$

If we want to compute the absolute probability for a particular node we simply follow the path through the tree to the root node and multiply to conditional probabilities. So if we want to know if a picture is of a Norfolk terrier we compute:

$$Pr(\text{Norfolk terrier}) = Pr(\text{Norfolk terrier}|\text{terrier}) * Pr(\text{terrier}|\text{hunting dog}) *\ldots* *Pr(\text{mammal}|Pr(\text{animal}) * Pr(\text{animal}|\text{physical object})$$

For classification purposes we assume that the the image contains an object: $Pr(\text{physical object}) = 1$.

To validate this approach we train the Darknet-19 model on WordTree built using the 1000 class ImageNet. To build WordTree1k we add in all of the intermediate nodes which expands the label space from 1000 to 1369. During training we propagate ground truth labels up the tree so that if an image is labelled as a "Norfolk terrier" it also gets labelled as a "dog" and a "mammal", etc. To compute the conditional probabilities our model predicts a vector of 1369 values and we compute the softmax over all sysnsets that are hyponyms of the same concept, see Figure 5.

Using the same training parameters as before, our hierarchical Darknet-19 achieves 71.9% top-1 accuracy and 90.4% top-5 accuracy. Despite adding 369 additional concepts and having our network predict a tree structure our accuracy only drops marginally. 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 is uncertain what type of dog it is, it will still predict "dog" with high confidence but have lower confidences spread out among the hyponyms.

This formulation also works for detection. Now, instead of assuming every image has an object, we use YOLOv2's objectness predictor to give us the value of $Pr(\text{physical object})$. The detector predicts a bounding box and the tree of probabilities. We traverse the tree down, taking the highest confidence path at every split until we reach some threshold and we predict that object class.

Dataset combination with WordTree. We can use WordTree to combine multiple datasets together in a sensible fashion. We simply map the categories in the datasets to synsets in the tree. Figure 6 shows an example of using WordTree to combine the labels from ImageNet and COCO. WordNet is extremely diverse so we can use this technique with most datasets.

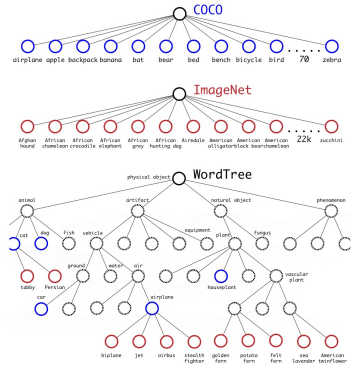
Joint classification and detection. Now that we can combine datasets using WordTree we can train our joint model on classification and detection. We want to train an extremely large scale detector so we create our combined dataset using the COCO detection dataset and the top 9000 classes from the full ImageNet release. We also need to evaluate our method so we add in any classes from the ImageNet detection challenge that were not already included. The corresponding WordTree for this dataset has 9418 classes. ImageNet is a much larger dataset so we balance the dataset by oversampling COCO so that ImageNet is only larger by a factor of 4:1.

Using this dataset we train YOLO9000. We use the base YOLOv2 architecture but only 3 priors instead of 5 to limit the output size. When our network sees a detection image we backpropagate loss as normal. For classification loss, we only backpropagate loss at or above the corresponding level of the label. For example, if the label is "dog" we do assign any error to predictions further down in the tree, "German Shepherd" versus "Golden Retriever", because we do not have that information.

이 같은 방법을 통해 ImageNet 데이터와 CoCo 데이터를 합쳐 word tree를 구성!

논문에서는 CoCo dataset과 ImageNet 데이터셋을 합쳐 9418개의 범주를 가지는 Word Tree를 구성!

(ImageNet : CoCo = 4 : 1 비율)



네트워크가 detection 데이터셋의 이미지를 보면 detection loss는 평소와 같이 loss를 backward pass 하며, classification loss의 경우에는 특정 범주와 상위 범주에 대해서만 loss를 계산!

만약 네트워크가 classification의 이미지를 본다면 오직 classification에 대해서만 backward pass를 수행. 이때 ground truth box 와의 IOU값이 0.3 이상인 경우에만 역전파를 수행!

이 같은 joint training 방식을 통해 YOLO 9000모델은 CoCo dataset을 활용하여 이미지 내에서 객체는 존재는 detection task와 imageNet을 통해 보다 넓은 범주의 객체를 분류하도록 학습!