YOLO9000: Better, Faster, Stronger

1 Citation

Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." arXiv preprint (2017).

`https://arxiv.org/pdf/1612.08242.pdf`

2 Abstract

YOLO9000 is state of the art object detection that runs in realtime. You can trade-off speed with mAP (e.g. 67 fps is 76.8 mAP and 40 fps is 78.6 mAP). We jointly train classification and detection and use both ImageNet and COCO. To do this, we predict a hierarchical set of concepts (taken from WordNet). This means we can predict over 9000 classes.

3 Introduction

Labeling detection data is hard, so detection datasets are smaller than classification datasets. How can we better harness the classification dataset?

4 Better

YOLO makes more localization errors than Fast R-CNN and has less region recall. We try to fix this in YOLOv2. We can't use a much deeper network because we still need realtime performance.

We use batch normalization on each convolutional layers and remove Dropout. This gives 2% gain in mAP.

YOLO trains on 224×224 images from ImageNet and then suddenly gets fine-tuned on $448 \times 448 images$. For YOLOv2, we train on 448×448 ImageNet images for 10 epochs before switching to COCO. This gives the network more time to adjust to the new size and gives 4% mAP gain.

Instead of predicting coordinates, let's predict offsets from anchor boxes (Faster R-CNN benefits from this). We remove the FC layers and use anchor boxes instead. To do this, we remove a pooling layer to get a higher resolution output. Then, we shrink the network to operate on 416×416 images instead of 448×448 - this gives a 13×13 feature map, which has a clear center cell (we anchor the box here). Next, we predict a class distribution and objectness score for every anchor box. This approach hurts our mAP (69.5% to 69.2%) but boosts our region recall (81% to 88%).

Instead of running anchor box sizes and aspect ratios by hand, we compute them by doing K-means clustering on the training data. Our distance function is:

$$d(box, centroid) = 1 - IOU(box, centroid)$$

We set $k = 5$.

With the improvements so far, we find that the model is unstable at predicting (x, y) coordinates for boxes. This is because the (x, y) coordinates are parameterized as:

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

which is unconstrained. So, we re-parametrize to predict coordinates relative to the grid cell instead of offsets and use logistic activation to constrain them between 0 and 1. That is, we predict $(t_x, t_y, t_w, t_h, t_o)$. If the cell has offset (c_x, c_y) and the anchor box has size p_w, p_h, then we have:

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

This gives 5% mAP boost.

Another improvement is to add a pass-through layer that combines our 13×13 feature maps with 26×26 feature maps from a previous layer. To shrink the 26×26 feature maps, we stack adjacent spatial values into multiple channels instead of keeping them spatially separate. This turns our $26 \times 26 \times 512$ feature map into a $26 \times 26 \times 2048$ feature map. Concatenate this with the 13×13 feature maps and get a 1% mAP boost.

We also try multi-scale training. Every 10 batches, we sample from a multiple of 32 between 320 to 608 and scale the input 416 image to that size and modify our network architecture so it can process and image of that size. We pick 32 because that's our network's total downsampling ratio. At test time, you can pick one input scale to trade off speed and performance.

We get 78.6% mAP on PASCAL VOC, which is state of the art.

5 Faster

To make our network fast, we create our own classification architecture called Darknet-19 instead of using VGG (which is slow). We have 8.6 billion multiply-adds instead of the 30.7 billion of VGG. We get 88% top-5 accuracy while VGG gets 90% top-5 accuracy.

Darknet-19 (see paper for architecture) uses mainly 3×3 convolutions. We double the number of channels after each pooling. We use global average pooling. We compress the feature maps with 1×1 convolution before applying 3×3 convolution. We use batch normalization. We have 19 conv layers and 5 max-pooling layers.

We train for 160 epochs on ImageNet classification with SGD with weight decay and momentum. We augment data with crops, rotations, hue shift, saturation shift, and exposure shift. We initially train on 224×224 images. Then, we fine tune on 448×448 images.

To convert the network for detection, we remove the last conv layer, add three 3×3 conv layers (each with 1024 filters), and add a 1×1 conv layer where the number of channels is equal to the number of outputs we need for detection. We add our pass through layer for fine-tuning. We train for 160 epochs with SGD with weight decay, momentum, and shrinking learning rate. We augment the data with crops, color shifts, etc.

6 Stronger

Now with a trained Darknet-19, we want to jointly train on classification and detection. So, we take a training image from detection or classification and feed it through the network. If it's a classification image, we backpropagate loss only from the classification piece of the network. The problem with this approach is that detection labels tend to be more coarse (e.g. dog) than classification labels (e.g. Yorkshire terrier, Norfolk terrier).

To mitigate this issue, we use hierarchical classification. We build our hierarchy by looking at WordNet, a language database that links concepts. We consider each class's paths to the "physical object" node in WordNet and pick the path that grows our tree as little as possible. Thus, we incrementally build our tree that we will use for hierarchical classification. To compute the probability of a node, we multiply probabilities along the path to that node, assuming that $Pr(\text{physical object}) = 1$ (our root node).

This lets us combine datasets. If we combine every ImageNet dataset and COCO, we get a tree with 9000 concepts. For a given label, we also treat every ancestor node in the tree as also being labeled.

This enables us to train on COCO and ImageNet. We oversample from COCO so that ImageNet is only larger by 4:1. Using this tree approach and multiple datasets, we turn YOLOv2 into YOLO9000. We also set the number of anchor boxes to 3 instead of 5 for compute efficiency.

For backpropagation, we backpropagate detection normally, but for classification we only backpropagate loss through the classification pieces (assuming the true object location matches the location of the bounding box with highest confidence for that class and assuming the IOU is 0.3). For classification, we also don't backpropagate to more fine-grained labels. That is, if the label is "dog" we don't backpropagate to "Yorkshire terrier" or "Norfolk terrier".

We find YOLO9000 does well on learning new animals, but poorly on equipment and clothing.

7 Conclusion

YOLOv2 is fast and state-of-the-art on detection.

YOLO9000 is flexible enough to train on many different datasets thanks to our hierarchical WordNet-based classification.