You Only Look Once: Unified, Real-Time Object Detection

1 Citation

Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

`https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf`

2 Abstract

We solve object detection with a single neural network that takes an input image and predicts bounding boxes and class probabilities. It can handle 45 frames per second (there's a fast version that does 155 frames per second) and gets near state-of-the-art. Compared to state of the art, it messes up localization more often but is better at avoiding false positives on the background class.

3 Introduction

R-CNN has several steps (1) propose regions (2) classify/bound regions (3) non-maxima suppression, which makes it slow.

Our system, YOLO, frames detection as a regression problem and solves it with a single neural network. It's blazing fast, can use entire image when making prediction (avoids background false positives), and generalizes to other datasets better. The downside is that it isn't state of the art because it can't localize as well as other systems.

4 Unified Detection

Divide the image into an $S \times S$ grid (we set $S = 7$). If an object's center falls into a grid cell, that grid cell must detect the object. Each cell predicts $B = 2$ bounding boxes, each with a $Pr(Object)$ confidence. The (x, y) of the bounding box are relative to bounds of grid cell and the (w, h) are relative to image size. Each grid cell also predicts $Pr(Class_i|Object)$.

Our network has 24 (the fast version has 9) convolutional layers and 2 fully connected layers. It resembles GoogLeNet, but instead of Inception modules, we just use 1×1 reductions followed by 3×3 convolutions. See paper for network diagram.

We first pretrain on ImageNet. To do this, we remove the last 4 conv layers, and add an average pooling layer and fully connected layer. We get 12% top-5 error (similar to GoogLeNet). To convert this to object detection, we remove the average pooling + FC layer and add 4 conv layers + 2 FC layers. We also increase the input size from 224×224 to 448×448 (makes it easier to detect small objects).

The last layer has linear activation and all other layers have leaky ReLu:

$$\phi(x) = \mathbb{I}(x > 0)x + \mathbb{I}(x \leq 0)0.1x$$

Our loss function is:

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

where $\mathbb{I}_i^{obj}$ indicates whether an object appears in cell i and $\mathbb{I}_{ij}^{obj}$ indicates that bounding box j in cell i is "responsible" for making the prediction.

Let's examine this loss function. First, notice that we use sum of squared losses because it's easy to optimize. It has two issues though (1) it weights a classification mistake the same as a localization mistake (2) since most grid cells don't contain an object, it will encourage the network to push their confidences to zero. To fix this, we introduce $\lambda_{coord} = 5$ to punish localization errors more aggressively and $\lambda_{noobj} = 0.5$ to be relaxed when the grid cell has no object. Another issue is that we want our network to weight a small error more heavily if it is in a small box rather than a big box. Taking the square root of the heights and widths partially achieves this. To encourage each bounding box in a cell to specialize, we designate only one box in the cell as responsible for the object (this is the box with highest intersection-over-union with the ground truth box).

We train for 135 epochs on PASCAL VOC 2007 and 2012 with a minibatch size of 64, momentum of 0.9, and weight decay of 0.0005. We start with a small learning rate (10^{-3}), raise it (10^{-2}), and drop it again (10^{-3}, 10^{-4}) .

We use Dropout on FC layers with rate = 0.5. We augment the data with random scaling, translation, exposure shift, and saturation shift.

If you want, you can use non-maxima suppression of the bounding boxes (gives 2- 3% mAP).

Since we have two bounding boxes per cell, YOLO fails if a cell contains many objects (e.g. flock of birds in the distance). We also can't handle weird aspect ratios (objects in PASCAL VOC don't have such aspect ratios). We are not fully able to encode the preference that a small error is worse in a small box than a big box, so we get localization errors.

5 Comparison to Other Systems

The Deformable Parts Model slides a window over the image and extracts static features. It then classifies them and applies non-maximum suppression.

R-CNN uses selective search to propose regions, extracts features with a CNN, and classifies with SVMs. Fast R-CNN and Faster R-CNN speed it up, but are not realtime ($\geq$ 30 frames per second).

Deep Multibox uses a CNN to predict regions and can detect a single object class. It can't do multi-class detection.

OverFeat use CNNs for localization and detection, but they optimize for localization and don't incorporate global information.

Multigrasp detects the region of an image where a robot should grasp an object. This is simpler than multi-class object detection.

6 Experiments

We compare against Fast R-CNN and DPM. We measure mAP.

YOLO is twice as accurate as previous realtime detectors.

We tried replacing GoogLeNet with VGG and got worse performance.

19% of our mistakes are localization errors (compared to 8.6% for Fast R-CNN). It's our biggest error source. However, for background false positives, it is 4.75% of our error but 13.5% Fast R-CNN's error.

If you ensemble YOLO and Fast R-CNN, you get state of the art on object detection. Using two Fast R-CNN models does not get this performance.

We tried appling YOLO to two artwork datasets and it beats Fast R-CNN and DPM because Fast R-CNN's selective search is tuned for natural images and DPM is not perfect for artwork either.

7 Realtime Detection in the Wild

We connected YOLO to a webcam and got realtime performance.

8 Conclusion

YOLO is blazing fast because it's a single neural network that solves object detection. It generalizes well to other datasets and can ensemble with Fast R-CNN.