Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

1 Citation

Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems. 2015.

`http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf`

2 Abstract

Fast R-CNN and SPPNet have made object detection fast, but now the bottleneck is the proposal of regions. We speed it up by making a region proposal network (RPN) that shares its convolutional layers with the detection network. We get state of the art on PASCAL VOC (70.4% mAP) at 5 frames per second.

3 Introduction

Region proposal techniques like selection search are slow because they run on the CPU instead of the GPU. We make a region proposal network (RPN) that runs on GPU and shares convolutional layers with the detection network (this shares computation). The RPN has convolutional layers, a fully-connected layer that extracts a vector that then gets fed into a layer that predicts k bounding boxes and another layer that predicts the probability that the each of the k bounding boxes has an object in it.

4 Related Work

OverFeat predicts a bounding box. Multibox produces region proposals.

5 Region Proposal Networks

We need a base CNN. We use Zeiler and Fergus (ZF) network or VGG. Keep only the convolutional layers and make sure they are pretrained on ImageNet.

Slide the CNN over the image. We get d $n \times n$ feature maps at each pixel. Feed each pixel's maps through a fully-connected layer to get a $256d$ length feature vector. We then feed this through two sibling fully connected layers. The first layer, called the regression layer, considerd k anchor bounding boxes centered at the pixel (we use three scales and three aspect ratios, so that's $k = 3 \times 3 = 9$). We predict a bounding box (x, y, w, h) for each anchor (so the regression network outputs a $4k$-length vector). The second layer, called the classification layer, predicts a pair of numbers, a object-ness score and non-object-ness score for each anchor (so $2k$ length output).

For a $W \times H$ image, we have WHk anchors. Each is labeled with a binary object-ness value. If it has the highest (or ≥ 0.7) intersection-over-union (IoU) with a ground truth bounding box, it gets labeled as 1. If the highest IoU is ≤ 0.3, then it's labeled as 0. Our loss then becomes:

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

where anchors are index by i, p_i is object-ness probability, p_i^* is ground truth object-ness binary label, t_i is the predicted bounding box, and t_i^* is the ground truth bounding box. We parametrize bounding boxes as follows:

$$t = (\frac{x - x_a}{w_a}, \frac{y - y_a}{h_a}, \log \frac{w}{w_a}, \log \frac{h}{h_a})$$

where the a subscript refers to the anchor.

We can train the RPN with gradient descent. We use 256-anchor minibatches. We pretrain our CNN layers on ImageNet and use those to initialize the CNN layers (the new layers have initial weights sampled from a zero-mean Gaussian).

How do we make the detection network and RPN share convolutional layers?

1. Train RPN (with CNN layers pretrained on ImageNet)
2. Generate proposals and train detection network (with CNN layers pretrained on Imagenet)
3. Now copy the CNN layers from detection network to RPN. Fine-tune only the RPN-specific layers, keeping CNN layers fixed.
4. Copy CNN layers from RPN to detection network and fine-tune only detection-specific layers.

Scale images so shorter side has length $s = 600$. For anchors, we use three scales: 128^2, 256^2, 512^2 at three aspect ratios: 1:1, 1:2, 2:1. During training, we ignore anchors that go outside the image boundary. During testing, we just clip them to the image boundary.

We apply non-maxima suppression based on object-ness score with an IoU threshold of 0.7. We then feed the top-N proposals to the detection network.

6 Experiments

If you don't share conv layers between RPN and detection networks, mAP drops.

Replacing the RPN with SS after both RPN + detection network are trained, mAP drops.

If you ignore the object-ness scores at test time and just randomly sample proposals, mAP drops (slightly if $N = 1000$ and hugely if $N = 100$).

If you ignore the regressed bounding boxes and just use the anchor, mAP drops moderately.

VGG has moderate improvement in mAP than ZF net.

We can handle 5 frames per second with VGG and 17 fps with ZF.

Proposal recall drops gracefully as you decrease N from 2000 to 100.

OverFeat is one-stage. We are two-stage (RPN + detection network). We simulate a one-stage network and find that our two-stage approach gives moderage gains to mAP.

7 Conclusion

The RPN is basically zero computation cost because it shares conv layers with the detection network. Our system runs at 5 frames per second and beats Fast R-CNN.