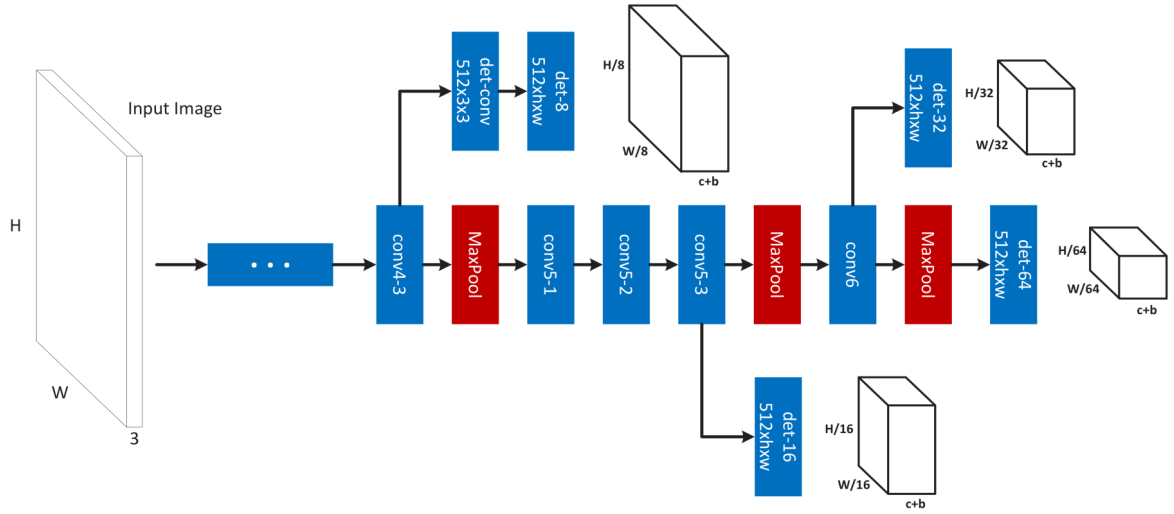

Figure 1:

Summary of "A Unified Multi-scale Deep CNN for Fast Object Detection"

Zhaowei Cai, Quanfu Fan, Rogerio S. Feris, and Nuno Vasconcelos

Overview

This paper focuses on the problem of detecting objects that may appear anywhere within a large range of scale. Distant objects appear small and close objects appear large. Specific features learned by object detection Neural Networks work best over a limited range of scale so standard networks tend to perform poorly with objects that differ widely from the one learned scale.

Like many object detectors this solution is split into two CNN tasks: A Proposal Network and a Detection Network. The Proposal Network is trained at multiple scales and generates proposals that consist of a classification (car, pedestrian, bicycle, etc) and a bounding box. The classification portion of these propsals could be used for final output and the results would be okay but the detection quality is much improved by adding a Detection Network. The Detection Network takes the proposals and performs 7x7 ROI pooling which feeds a fully connected layer. This FC layer outputs class probability and a bounding box.

The two tasks (Proposal and Detection) share the first four layers of the network. This reduces the computational load at inference time and allows for shared learning between the two tasks. Since this paper deals with the final layers of the network you could use any architecture for the earlier layers but in this case they used VGG-Net.

Proposal Network

The Proposal Network (image above) acheives multi-scale detection by branching off of the CNN at multiple layers. Each branch corresponds to a different spatial extent because pooling layers change the size of the feature maps. For the leftmost branch, each proposal feature covers a box on the input image of 8x8 pixels. The next branch each features covers 16x16 input image pixels and so on up to the final branch which covers 64x64 input image pixels.

Note that the leftmost branch includes an extra convolutional layer with no pooling. This was done as a buffer to keep the first branch losses from dominating the learning done in the earlier layers. The proposal outputs (white boxes at the end of each branch) are shown with a depth of "c+b". Here "c" is the number of classes and the "b" is the number of values used to describe the bounding box. In this case b is 4 (BoundingBox_x, BoundingBox_y, BoundingBox_height, and BoundingBox_width).

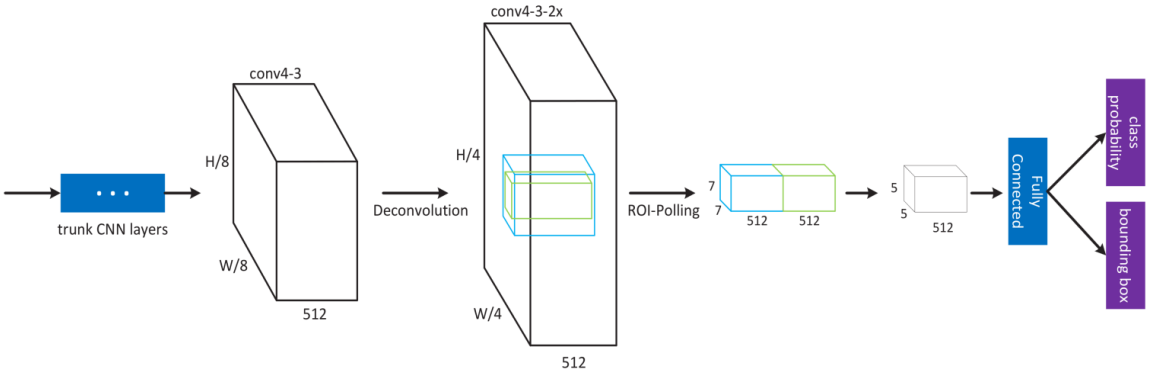

Figure 2:

Detection Network

The Detection Network shown above branches from the base network at the same "conv4-3" layer as the first proposal layer. The first step in the detection pipeline is a deconvolutional layer that resizes the feature map to twice the original height and width using bilinear interpolation. The authors found that approximating a larger feature map through interpolation improved the results. Inside this "conv4-3-2x layer (above) there are green and blue boxes. The green box is the proposed bounding box (taken from the Proposal Network output) and the blue box is called the "context" box. This blue box is always set to 1.5 times larger than the bounding box.

The green and blue boxes are each ROI pooled down to 7x7x512 and concatenated yeilding a 7x7x1024 feature map. This is subjected to a 3x3 non-padded convolution which generates a 5x5x512 set of features. This convolutional step compresses redundant object and context features and reduces the number of model parameters. The final step is a fully connected layer that outputs class probabilities and bounding box coordinates.

Training

The authors go to great lengths to ensure that the training set is balanced along the positive/negative, hard/easy, and small-object/large-object feature axes. Training occurs in two stages. For the first 10,000 iterations the data is sampled at random from dataset and the learning rate is set to 0.00005. For the next 25,000 iterations the balanced dataset is used and the learning rate decays 10X every 10,000 cycles.

Implementation

The authors have code available online at (https://github.com/zhaoweicai/mscnn). It's written in C++ with Caffe bindings. Trained weights are provided as is a Docker image.

Opinion

On the plus side, this technique provides solid KITTI object detection performance (#23 on the leaderboard) and a trained implementation is available online. On the minus side the C++/Caffe implementation may be more challenging to incorporate into our pipeline.

This is the highest ranking object detector with a link to code (and trained weights) so I'm inclined to use it unless we have interface difficulties.