Intelligent Methods for Solving Real Problems

Object Detection for driving assistance

1. Introduction

Object detection is a subfield of computer vision and image processing that deals with detecting instances of semantic objects in images and videos. Detection means drawing corresponding bounding boxes for objects of interest, in our case: people and cars.

2. General Approach

State of the art in object detection is achieved by deep learning-based methods. Our focus is to study and apply techniques that would best solve our problem. Our approach is based on the idea of a Single Shot Detector [1], namely an SSDLite Convolutional Network [2] with priors adapted to closely represent cars and people, trained with focal loss [3] to alleviate the problem of class imbalance.

2.1. Model Description

The SSDLite model uses a MobileNetV2 [2] as a feature extractor while also replacing standard convolutional layers with depth wise separable ones, thus highly focusing on speed and efficiency.

Concretely, the model computes offsets and class IDs for a predetermined set of anchors (also called priors, see Figure 0) that try to cover all, or most of the possible locations where objects might be found. To achieve this, the model uses a set of low resolution feature map cells, specifically in our case the 10x10, 5x5, 3x3, 2x2 and 1x1 grids, the idea being that the receptive fields [4] of the cells are large enough to incorporate valuable spatial information to correctly predict the type of the object and its bounding box.

As an example, for an image, the model would compute two large arrays: A x 4 and A x 2, where A is the total number of anchors. Specifically, the job of each anchor is to say whether an object is located at the area it encompasses, hence the 4 bounding box coordinates, and the 2 confidence scores for each class.

2.2. Technologies used

Python, PyTorch, CUDA, Numpy, OpenCV

Figure 0: Sparsely plotted anchors

`

3. The Dataset - COCO [6]

3.1. Description

COCO is a large-scale object detection, segmentation, and c

aptioning dataset with 330K images and 1.5 million object instances.

3.2. Format and Benchmarks

Each training image has associated with it a list of bounding boxes and class IDs for the object instances it contains, Fig. 1. There are 12 metrics used for characterizing the performance of an object detector on COCO, Fig. 2

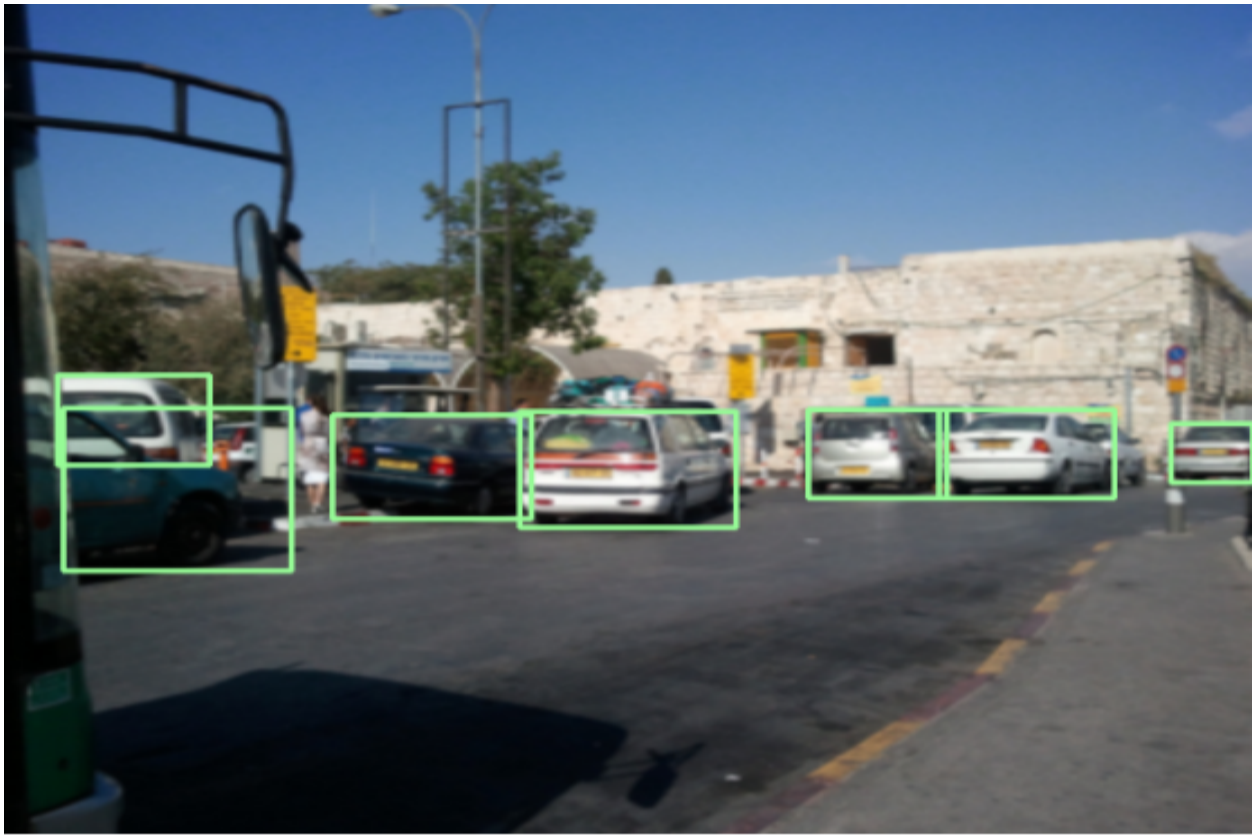

Figure 1: Sample labeled image from the COCO validation set

```
Average Precision (AP):
  AP                  % AP at IoU=.50:.05:.95 (primary challenge metric)
  AP^IoU=.50          % AP at IoU=.50 (PASCAL VOC metric)
  AP^IoU=.75          % AP at IoU=.75 (strict metric)
AP Across Scales:
  AP^small            % AP for small objects: area < 32^2
  AP^medium           % AP for medium objects: 32^2 < area < 96^2
  AP^large            % AP for large objects: area > 96^2
Average Recall (AR):
  AR^max=1            % AR given 1 detection per image
  AR^max=10           % AR given 10 detections per image
  AR^max=100          % AR given 100 detections per image
AR Across Scales:
  AR^small            % AR for small objects: area < 32^2
  AR^medium           % AR for medium objects: 32^2 < area < 96^2
  AR^large            % AR for large objects: area > 96^2
```

Figure 2: AP and AR are averaged over multiple Intersection over Union (IoU) values. Specifically there are 10 IoU thresholds of .50:.05:.95.

4. Pre-processing steps

Before commencing training, the data must be brought in the necessary format for the model to be able to learn efficiently.

4.1. Selecting our data

As we are interested in the person and car classes, we have only kept images containing instances of these two objects. Furthermore, since these classes generally have large instances (except if they are located far from the observing point in the image), we have also stripped small instances (segmentation area <32*32 pixels).

4.2. Constructing mini batches and Augmenting Data

Each mini batch consists of a number of images with similar structure (320x320 resolution, RGB) and the associated ground truth labels that are then fed to the model. As for data augmentation, we employ horizontal flipping and adding noise in the brightness, saturation, contrast and hue of the images, see Figure 3.

4.3. The matching problem

It is not immediately clear how to construct a loss function for the model. It would be necessary to "bind" the correct feature map cells to bounding boxes in their respective locations in the image. Otherwise, if feature map cells whose receptive fields have no information about the bounding box/class they try to predict, learning would not be possible. However, note that we can easily bind anchors to feature map cells, so if we are able to match the anchors to the ground truth bounding boxes, the problem is solved. This, in turn, is done simply by iterating over all of the ground truth bounding boxes in an image and assigning them to anchors that they have a

high enough IoU with. Usually the mapping threshold is a hyperparameter whose value ranges between [0.4, 0.6], see Figure 4.

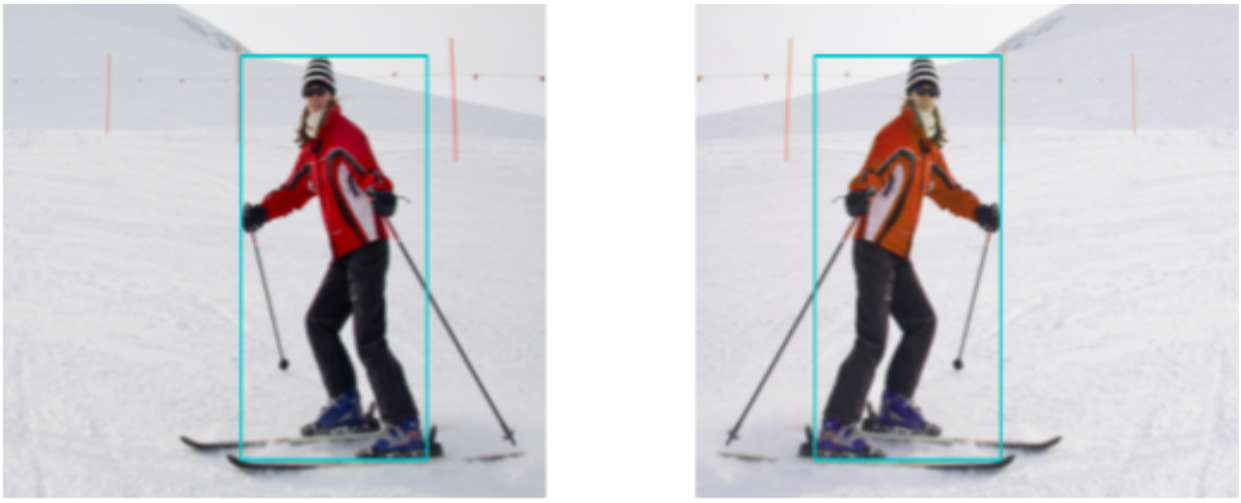

Figure 3: Comparison of original and augmented images

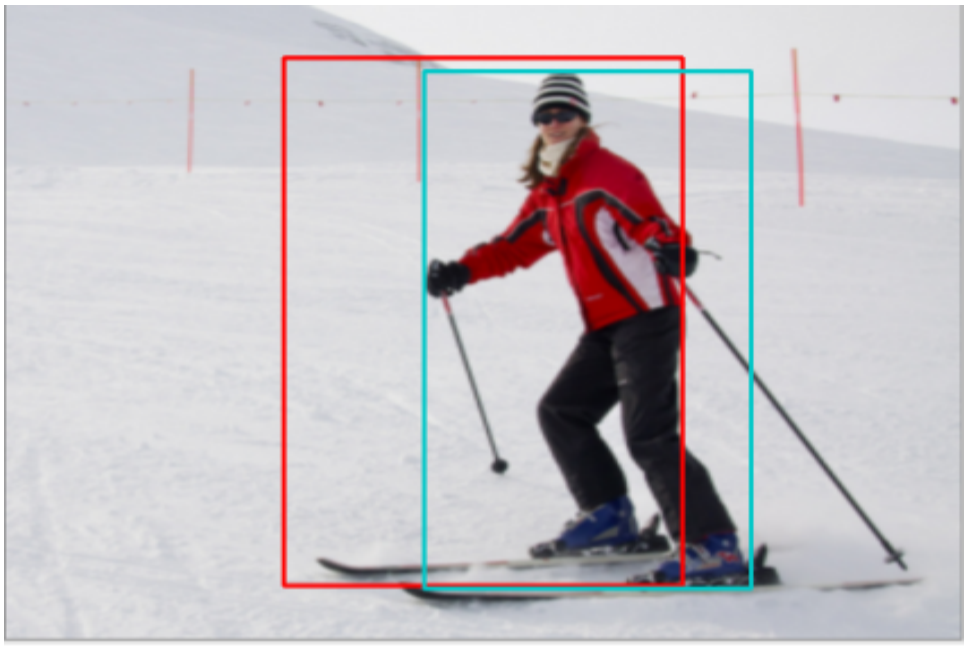

Figure 4: Anchor/GT Bounding Box with IoU = 0.53

5. Training

5.1. Transfer learning

Given the difficulty of training DCNNs, it is almost always a good idea to make use of transfer learning. PyTorch provides numerous pretrained models on ImageNet [5], and we chose MobileNetV2 as our backbone.

5.2. Handling a pretrained model

A pretrained model is extremely sensitive in the sense that a single wave of large gradients can completely ruin the weights and render the precisely trained model useless. This can happen, for example, if the learning rate used when training with the model as a backbone is larger than the one it has been trained with previously. This is why it is extremely important to choose a suitable learning rate, because in the scenario above the pretrained weights cannot be recovered. Concretely, we freeze the backbone concretely for the first few epochs, and then progressively unfreeze layers with a learning rate an order of magnitude lower than the one of the head.

5.3. Loss

The loss is computed as the sum of two components: the localization loss and the classification loss. The localization loss is simply the L1 difference between the ground truth bounding boxes and those predicted by the model, whereas the classification loss consists of a Binary Cross Entropy taken between the ground truth values and the confidence scores given by the model. It is worth noting that only the feature map cells that have mapped to ground truth instances have the localization loss accounted for. However, for classification, all feature map cells need to correctly return the scores, because in the case of no object there is still background. Given the heavy class imbalance that we face, especially since we elected to only train on two classes, we also employ focal loss [3] to alleviate this issue.

5.4. Learning rate decay

We begin training with a learning rate of 0.01, and then proceed to lower it by a factor of 10 at epochs 10, 20, and 35. We train for a total of 45 epochs.

5.5. Tensorboard

Tensorboard is a useful tool for keeping track of the training process through logging of various statistics, see Figures 5 and 5.1

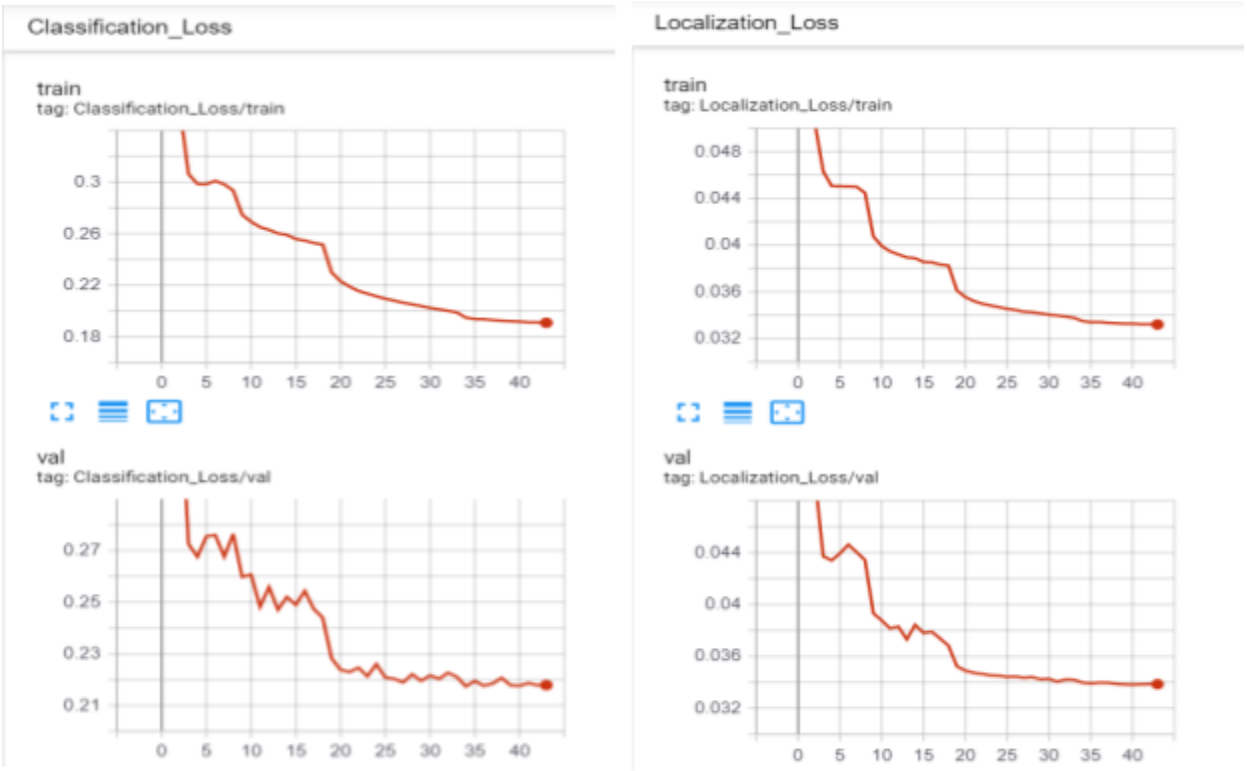

Figure 5: Localization and classification losses on the train and validation sets.

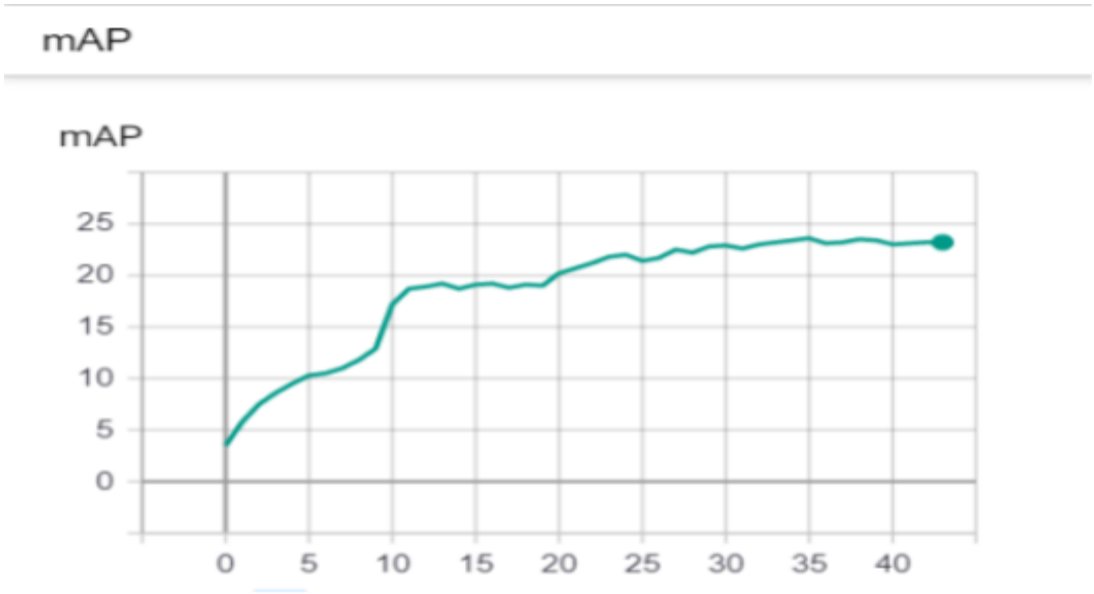

Figure 5.1: mAP on the validation set

5.6. Post-processing

We have to keep in mind that more than one anchor may map to a single ground truth object, and in that case the model may output multiple bounding boxes for a single instance, which is undesirable. Thus, we use the non maximum suppression algorithm to only keep the best matching and highest confidence score bounding box for each object, see Figure 5. Two hyperparameters are important when it comes to NMS, the confidence score for which the prediction is considered an object, and the suppression threshold specifying what IoU two bounding boxes have to have to be considered as predicting the same object. Both of these hyperparameters can vary quite a bit, so we tested out multiple values and picked the best performing on the validation set.

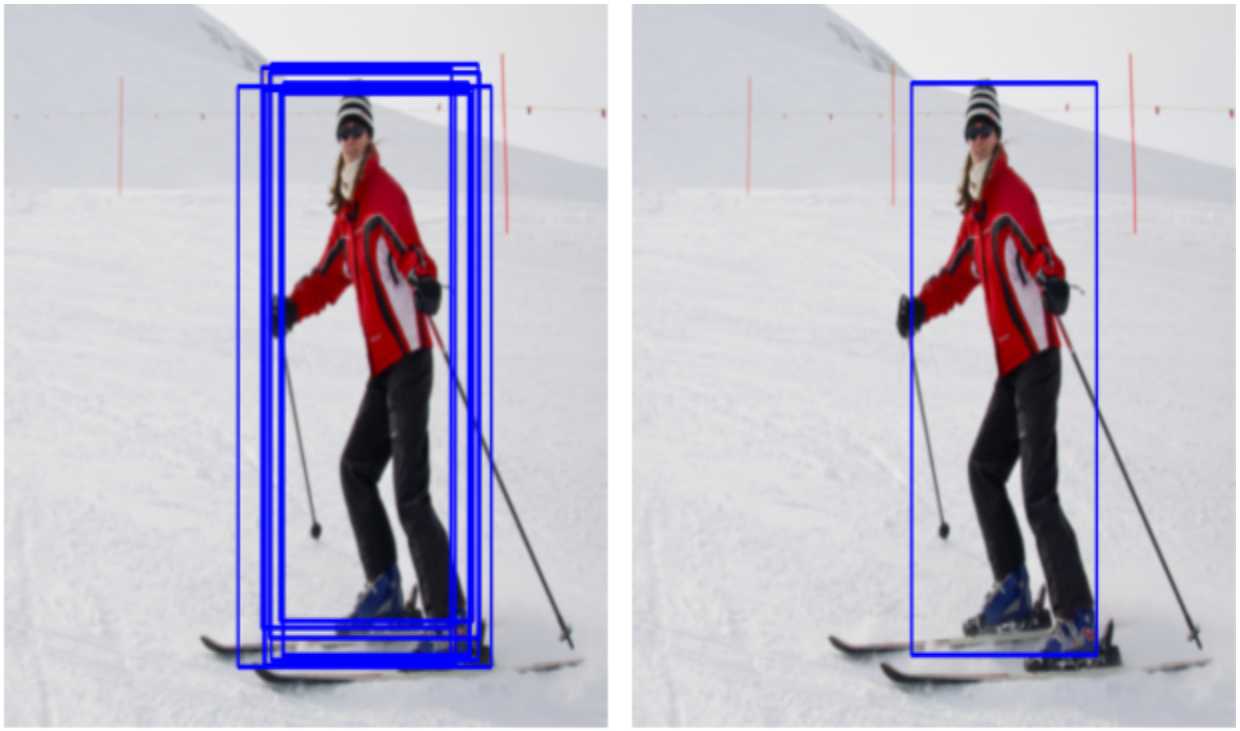

Figure 5: Before and after NMS

5.7. Performance

We currently achieve 56% AP50 on the person and car classes on the validation set, on medium and large instances. Although per category results are not usually reported, upon consulting the COCO leaderboard, models with similar mAP as SSDLite (22.1% reported in the paper) achieve 56% AP50 on people and 34% on cars respectively on the COCO test set. However, this also includes small objects, which considerably decrease the overall score. So we currently fall short of reproducing the results reported in the paper, several reasons for this and how we are trying to address the issues are stated in 7 and 8.

6. Inference

- The inference is pretty good even on CPU, the model takes only 200ms to compute predictions on a mobile phone

- Low parameter count: only 3.1M for two classes

- JAAD

7. Improving

7.1. Mistakes

- directly stripping off all images without desired properties left the dataset with only about half of the initial images, which without data augmentation led to overfitting issues.

- using a pretrained model without careful adjustments of the learning rate led to bad training and waste of time

- poor anchor choices

- not establishing a good baseline first and building upon that, instead we tried lots of heuristics on an uncertain basis which led us to very little improvement.

7.2. Addressing mistakes

- data augmentation: apply horizontal flipping and color channel jittering

- implement methods to progressively unfreeze layers and set per layer learning rates

- implement careful anchor inspection

- implement and train the standard SSD model -> implement and train SSDLite -> add focal loss

8. Training on only people and cars: Was it the right choice?

- there is the possibility that a deep model might benefit generally more from training on a large number of classes, we need more time to test and train to confirm this.

References:

[1] SSD: Single Shot MultiBox Detector

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg arXiv:1512.02325

[2] MobileNetV2: Inverted Residuals and Linear Bottlenecks

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen

arXiv:1801.04381

[3] Focal Loss for Dense Object Detection

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár

arXiv:1708.02002

[4] Understanding the Effective Receptive Field in Deep Convolutional Neural Networks

Wenjie Luo, Yujia Li, Raquel Urtasun, Richard Zemel

arXiv:1701.04128

[5] ImageNet Large Scale Visual Recognition Challenge

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei

arXiv:1409.0575

[6] Microsoft COCO: Common Objects in Context

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár

arXiv:1405.0312