Benchmark of mobile neural networks for object detection on a custom dataset

Ivan Chernukha
University of Trento
Mat. 188890
ivan.chernukha@studenti.unitn.it

ABSTRACT

This work presents overview of the current state-of-the-art object category detection neural networks optimized for running on mobile devices. We collect our own dataset, describe steps required to train efficiently a neural network specifically for this task, run experiments and demonstrate performance and speed efficiency for selected architectures.

1. INTRODUCTION

While deep neural networks have achieved a human level classification benchmark, for detecting objects there is still a room to continue development. Object detection (OD) is something harder than image classification, as it aims not only to classify instances of objects presented in an image, but also to find their location (localization), e.g. produce a bounding box around them (Figure 1).

There are many applications and real-world problems related to OD. The face detection problem gave the first motivation to work on this, as it is can be used in biometric systems for facial recognition, in cameras for stabilization, video surveillance to identify criminals. Manufacturing plants or shop malls may want to utilize counting for tracking goods or people. Another interesting example is Similar Look visual search at Pinterest [11] - it allows a user to click on an instance of object(e.g. shoes or bags), then it searches for similar products in different contexts, but in much more effective way than Google Image's search engine!

Now let's take a look at the challenges behind:

- *Unknown number of objects within an image.* In classic machine learning models we do have a fixed size output vector, thus we know exactly the number of outputs. For long time this was tackled by a very inefficient sliding window approach.

- *Size of objects is variable.* One needs different patterns to recognize a large object and the small one taking only dozen of pixels within an image.

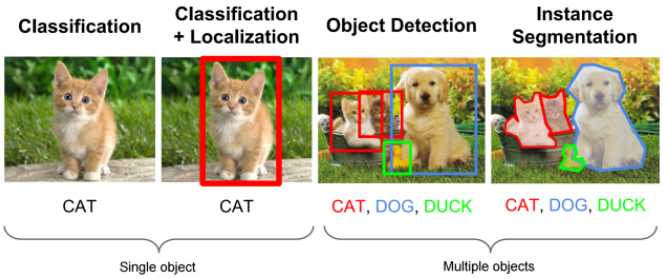


Figure 1: Classification vs Object Detection

- *A single model to solve two problems.* Ideally, we should have one single model that jointly does classification and bounding box labeling.

- *Occlusions, overlapping and other noise.* Dealing with one object in a frame might be straightforward, but that's not the case in most real-world problems, e.g. pedestrian detection in automotive.

- *Computational complexity.* We will see only some of the most recent works achieved real-time performance, but they still have limited power and/or needs a lot of GPU resources.

In Section 2 we will go through the history of OD from the first success to tackle this problem until the most recent work based on deep neural networks (NN). Just to mention, after NN breakthrough one of the major problems remained is a matter of computational power and efficiency. That's why in Section 4 we will dive into models that are very compact, efficient and can be run almost in real-time on mobile devices.

Unfortunately, due to large cost of human efforts to annotate objects with bounding boxes, there are not a lot of labeled datasets for evaluation and/or training.

The most popular one was introduced in Pascal Visual Objects Classes Challenge (VOC) [3] that have been running since 2005. The number of classes is equal to 20 (person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, tv/monitor) since 2007.

Other famous datasets are MS-COCO [13], ImageNet Object Detection (ILSVRC 2015) [18] and KITTI Vision [6]. See Table 1.

ACM ISBN .
DOI: 10.1145/1235

Name	Images	Classes	Last updated
ImageNet Detection	450k	200	2015
MS-COCO	123k	80	2014
PASCAL VOC (2010)	12k	20	2012
KITTI Vision	7k	3	2014

Table 1: Object Detection Datasets

Evaluation metric is the *mean average precision*. For example, *mAP@0.5* meaning it computes separate precision for each class and average over classes, while setting a threshold 0.5 on *intersection over union* (IOU) - an overlap between predicted and ground truth bounding boxes that must be greater this number for true positives.

2. RELATED WORK

Basically, there is a clear difference between classic computer vision algorithms, that exploits *hand-engineered* features, and deep neural networks, that *learn* features. We shall provide an overview of the most robust methods from the first set, then move to the state-of-the-art.

2.1 Classic computer vision approach

Absolute majority of approaches adopted the sliding-window paradigm, where a classifier is applied on an image grid.

The most known OD framework in 2000s was proposed by Paul Viola and Michael Jones [20], with father improvements for the first real-time face detector [21]. Briefly, it consists of four stages: (i) Haar feature selection - exploiting regularities of human face, e.g. nose area is brighter than eyes; (ii) creating an integral image(summed area table) - constant time calculation of features; (iii) AdaBoost training - as there will be too many classifiers, adaboost builds a linear combination of weighted weak classifiers; (iv) cascading classifiers - faces take only small area on an image, thus instead of applying all classifiers in row, group them and check one-by-one, discarding an image if it fails to pass the first stage.

Histogram of oriented gradients (HOG) applied for pedestrian detection by Navneet Dalal and Bill Triggs [2], is another great feature descriptor for OD. HOG utilizes gradient computation of vertical and horizontal gradients, then builds histograms of these gradients over patch of pixels and produce vector representation of an image, which feeds to SVM or NN classifier. Intuition behind is that it exploits edge and shape changes, thus "highlighting" areas useful for detecting an object.

Deformable part models proposed by Felzenszwalb [4] was based on HOG, transformation and rescaling of images to detect objects by matching to its parts, learning SVM with latent variables. At VOC'2007 it achieved 33.7 mAP.

2.2 Deep learning era

We here provide a brief overview of past works, as there are a lot of similar approaches with modifications and improvements. We will dive into details of the training process in Section 4. The first three stage architecture was Regions with CNN features (R-CNN) proposed by R. Girschick et al. [8], which were improved over years to Fast R-CNN [7] and Faster R-CNN [17]. R-CNN includes a few stages: (i) extract possible objects using a region proposal method; (ii) extract features from each region using a CNN and save them to disk; (iii) classify each region with SVMs. This is very inefficient, because for Pascal dataset features are 200GB for Pascal dataset, the inference is also very slow.

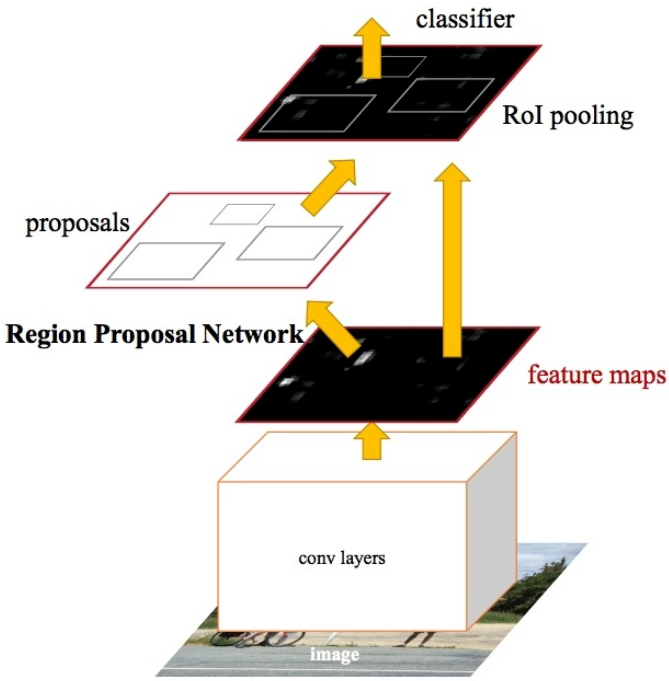


Figure 2: Faster R-CNN architecture.

Improvements in Fast R-CNN were instead of extracting all of region proposals independently and using SVM classifiers, it swaps the order and feed an image to CNN, then uses both Region of Interest (RoI) Pooling on the feature map with a final feed forward network for classification and regression.

Finally, Faster R-CNN is able to compute region proposals within the network, what makes this architecture end-to-end trainable with good performance and speed. See Figure 2.

Faster, real-time models do not inherit RPN, but instead treats detection as regression of bounding boxes in one stage or *single shot*. For example, YOLO by Joseph Redmon et al. [15] can process at 45 frames per second (FPS), Fast (Tiny) version - 155 FPS! We will dive into the architecture more in Section 4. A father improvement YOLO9000 was done by the same author [16]. See Table 2.

The last state-of-the-art model is SSD proposed by Wei Liu [14], which takes on YOLO network, by using multiple sized CNN. !!!!

There are a few recently published promising papers to be mentioned. Adopting new focal loss functions by Tsung-Yi Lin et al. [12], deconvolutional version of SSD (DSSD by Cheng-Yang Fu et al. [5], deeply supervised training without using pretrained networks by Zhiqiang Shen et al. [19].

3. DATA

In order to train a neural network for detecting own object categories or any other task, one firstly needs to collect and label own dataset. What may sound easy at the first

Model	MS-COCO	VOC2012
YOLO	-	57.9
Tiny-YOLO	-	-
YOLOv2	-	73.4
Faster R-CNN	-	70.4
Focal	-	-
DSOD	-	-
SSD512	-	68.4

Table 2: SOTA models benchmark

Class	Train	Val	Instances
ATM*	201	51	-
zebra road*	287	72	-
orange	418	149	-
trashbin*	490	128	-
sandwich	605	213	-
door*	736	198	-
bus	742	605	-
toilet	913	272	-
cup	920	352	-
traffic light	925	508	-
bed	995	294	-
cat	1100	380	-
sink	1149	424	-
dog	1157	363	-
train	1172	339	-
car	1177	962	-
couch	1244	202	-
chair	1432	408	-
bench**	1680	601	-
person	4325	2079	-
Total	11776	4313	–

Table 3: Dataset statistics

glance, indeed is a painful and tedious process, unless one can outsource this work. The raw image data can be either found on the web (though web data is *horrible*) or manually by taking pictures in the wild. For labeling one of the most very popular option is crowdsourcing.

For our purposes we exctracted images of 16 classes from MS-COCO validation dataset(40072 images), also collected 2415 images and labeled 4417 instances of 5 classes (total 20 classes). However, in order to eliminate class imbalance(original dataset contained 88344 instances of person!), we reduced the dataset to 11776 training images and left other 4313 for validation. Final train/val split is presented at Table 3 [1].

Collection was done using Google Image Search and Flickr, labeling with the tool LabelImg[2]. Labeled custom classes dataset is available online.[3]

4. STATE-OF-THE-ART UNDER THE HOOD

[1]NB: each image may contain several objects from different classes. Star * denotes manually labeled classes, ** is complemntary to COCO images

[2]https://github.com/tzutalin/labelImg

[3]https://goo.gl/KDJwFf

Which neural network provides the best trade-off between detection precision and speed during inference? Let's firstly take a closer look how the major players work!

4.1 YOLO

You Only Look Once (YOLO) network lags behind other networks in accuracy and presents errors in object localization, but it runs incredibly quickly.

There are key idea that makes this network work in one stage is dividing an image into a grid $S\ x\ S$ and associate each cell with a set of bouding boxes B with confidense crose and classification scores C, therefore the prediction is $S\ x\ S\ x\ (B \star 5 + C)$ tensor. Then it uses non-maximum suppression to eliminate 'bad' boxes. Basically, we have a standard CNN with the last fully connected layer, which is connected to this final prediction output. YOLO orininally proposed 24 layers network, but Tiny (Fast) model containing only 9 convolutional layers. The loss function is multi-part, which backpropagates erros for classification and bouding box values($x,\ y,\ w\ ,h$). Authors also that it performs badly for small objects. **Overall**, we can see that it was a good first attempt to provide end-to-end training, however in practice there are also a lot of parameters to tune the network and the detection is not precise.

Yolov2 improved a lot, having similar idea of Faster-R-CNN to have *anchors* and output *offset* bbox position wrt a cell instead of real values. Anchors are priors, predefined boxes for every grid cell. Authors propose to run k-means for a particular dataset in order to cluster and define an optimal number of predefined bboxes wrt to average IOU. Also it uses newer tricks, for instance batch normalization, hierarchical classification, which allows to detect *more than 9000 classes*(that's where the name comes from). **In the end,** we receive new tricks for detection part to be integrated with lighter versions of Yolo-like family networks.

4.2 SqueezeDet

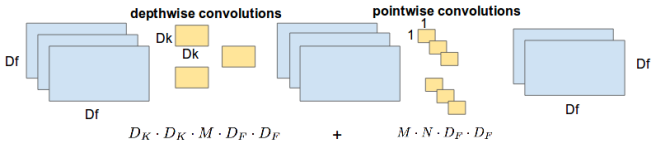
At the same time as Yolov2 was published, there was another work SqueezeDet [22] though not very known(only 6 citations vs 63 Yolo2). Basically, it uses the same ideas of anchors, but the backbone network for feature extraction is a very advanced efficient SqueezeNet [10] proposed by the same authors. With this network it is possible to achieve a 50X reduction in model size compared to AlexNet, while meeting or exceeding the top-1 and top-5 accuracy of AlexNet. The core idea of that is *Fire Module* which is a replacement for traditional 3x3 convolutional layer into 1x1 convolutional (squeeze) with later expand to mixture of 1x1 and 3x3,that should retrieve rich features of a compressed channel representation.

Unfortunately, authors reported only benchmark for KITTI dataset, but that's not sufficient to have a clear understanding of full capabilities of this architecture. As well as the comparison provided gives a parallel only to YOLOv1. **To conclude**, we'd say the idea of using SqueezeNet as a backbone CNN deserves to try with other setting.

4.3 Mobile-SSD

MobileNet by Andrew G. Howard et al. [9] as well focused on more efficient and light-weight convolutional type of layers that are able to achieve AlexNet accuracy. Big idea here is to

Figure 3: Depthwise and pointwise operations



Model	Top-1Acc.ImageNet	FLOPS[4]	Million param.
ShuffleNet x0.5	57.3	40	-
SqueezeNet	57.5	833	-
1.0-MobileNet224	70.6	569	4.2

Table 4: Mobile networks comparison

user *depthwise separable convolution*, then for example on top of it use detection pipeline from Faster-R-CNN or SSD [14]. Basically, it split standard convolution into two layers, for instance,The standard convolutional layer is parameterized by convolution kernel K of size $D_K x D_K x M x N$ where D_K is the kernel size assumed to be square and M is number of input channels and N is the number of output channels. Then, standard convolution has computational cost:

$$D_K \star D_K \star M \star N \star D_F \star D_F \quad (1)$$

where D_F is feature map size. In MobileNet this is replaced by depthwise and $1x1$ pointwise convolutionals, making up by sum of 2 terms. The final reduction will be:

$$\frac{D_K \star D_K \star M \star D_F \star D_F + \star M \star N \star D_F \star D_F}{D_K \star D_K \star M \star N \star D_F \star D_F} = \frac{1}{N} + \frac{1}{D_K^2} \quad (2)$$

While reported complexity of MobileNet Table 4 for classification on ImageNet is low, but in conjunction with SSD or even Faster for object detection it requires *one order of magnitude more operations*! [5]

4.4 ShuffleNet

Similar search for new convolutional layer were explored in yet the most recent paper of efficient networks - ShuffleNet [23], which claims to outperform MobileNet in both classification and detection tasks by accuracy. What refers to computational efficiency, it is not reported optimal trade-off between accuracy and cost, only comparable MobileNet model without stating which detection pipeline it uses!

Couple words about how it managed to beat(?) MobileNet. It turned out, that *shuffling* of channels(Section 4.4) gains the same level of accuracy with significant reduction of computation cost.

4.5 Summary

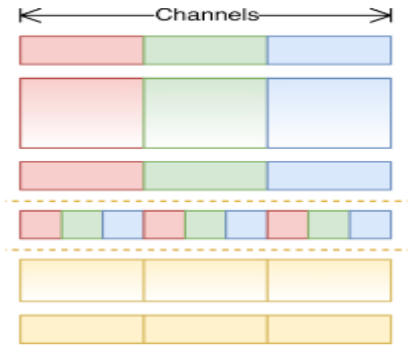
We have seen many different approaches, which are yet not comparable at every dimension, for instance, frozen model size is reported only for SqueezeDet. It is yet not clear how well models perform after compressing, as it's likely some are more sensitive to weight connection lose, while other may store redundant information. Interesting direction to explore is combining convolutional operations, using different detection pipeline to make the complexity not much different from classification one.

[5]Evaluated with FLOPs, i.e. the number of floating-point multiplication-adds

Model	mAP	MFLOPS
ShuffleNet x2 Faster-RCNN	24.8	524 + ???
1.0-MobileNet-224 Faster-RCNN	19.8	$30.5*10^3$
1.0-MobileNet-224 SSD	19.3	1200

Table 5: ShuffleNet vs MobileNet on MS-COCO

Figure 4: Channel shuffle operation



5. OPTIMIZING AND COMPRESSING MODELS

TODO:
It's logical to prune connection/weights and try to compress the model if one wants to achieve the fastest inference time possible. There are already some tools provided in Tensorflow, however that's maybe oout of scope of experiments. Also there are some things to try with training of the neural network, e.g. one work I found is named "Improving Object Detection With One Line of Code" [1] that presents an improvement of NMS(here's Soft-NMS) for bounding boxes.

6. EXPERIMENTS

> "Training a neural network is more like an art rather than a science."
> Unknown author.

6.1 Implementation notes

For training our model we used open-source implementations of YOLO[6] and SqueezeDet[7]. For converting labels of original COCO and/or VOC annotations the converter tool[8]. For our dataset we used anchors stated as for Pascal VOC dataset.

6.2 Results

For YOLO training we tied firstly to overfit on a small subset of our images, initializing the model with VOC pre-trained weights. But it turned out, that it the network

[6]https://github.com/thtrieu/darkflow
[7]https://github.com/BichenWuUCB/squeezeDet
[8]https://github.com/nghiattran/vod-converter

Model	mAP
Tiny-YOLO	19.4
SqueezeDet	-

Table 6: Benchmark of custom dataset

memorizes very well certain categories and rejects to recognize at all some others. Sample predictions available in appendix section.

7. CONCLUSION

This project enabled to get acquired with the state-of-the-art models in object detection and test some of them on a custom dataset. We went through all details of the modern approaches in creating efficient and powerful neural networks and adapting them accordingly to the needs of accuracy-efficiency trade-off. It's now clear where one should continue development in order to achieve real time performance on a mobile device.

8. REFERENCES

[1] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis. Improving object detection with one line of code. *CoRR*, abs/1704.04503, 2017.

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society.

[3] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, Jun 2010.

[4] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, Sept. 2010.

[5] C. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. DSSD : Deconvolutional single shot detector. *CoRR*, abs/1701.06659, 2017.

[6] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[7] R. Girshick. Fast R-CNN. *ArXiv e-prints*, Apr. 2015.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *ArXiv e-prints*, Nov. 2013.

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv e-prints*, Apr. 2017.

[10] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[11] Y. Jing, D. Liu, D. Kislyuk, A. Zhai, J. Xu, J. Donahue, and S. Tavel. Visual Search at Pinterest. *ArXiv e-prints*, May 2015.

(a) Zebraroad (b) Trashbin

[12] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. Focal Loss for Dense Object Detection. *ArXiv e-prints*, Aug. 2017.

[13] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common Objects in Context. *ArXiv e-prints*, May 2014.

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single Shot MultiBox Detector. *ArXiv e-prints*, Dec. 2015.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *ArXiv e-prints*, June 2015.

[16] J. Redmon and A. Farhadi. YOLO9000: Better, Faster, Stronger. *ArXiv e-prints*, Dec. 2016.

[17] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[19] Z. Shen, Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue. DSOD: Learning Deeply Supervised Object Detectors from Scratch. *ArXiv e-prints*, Aug. 2017.

[20] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.

[21] P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004.

[22] B. Wu, F. N. Iandola, P. H. Jin, and K. Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. *CoRR*, abs/1612.01051, 2016.

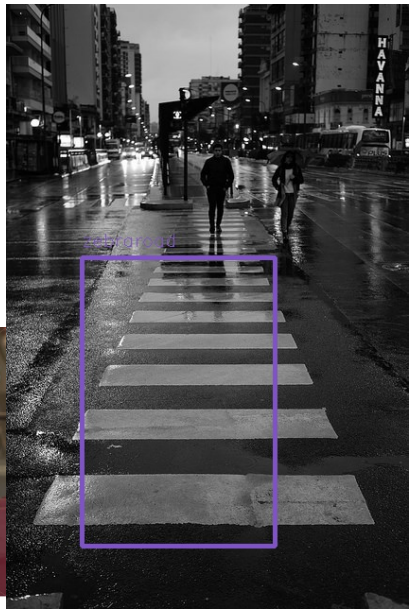
[23] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.

APPENDIX

Examples of predicted detections.

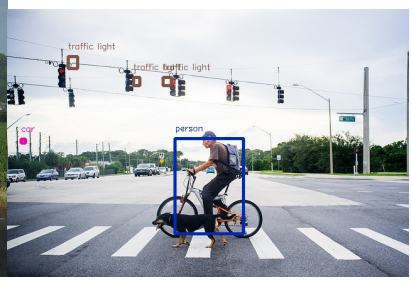
(a) Trashbin

(b) Atm



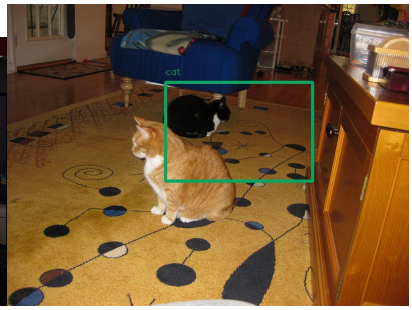
(c) Zebraroad

(a) Train



(b) Trafficlight

(a) Bed



(b) Cat