Master's thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Vehicle Detection and Pose Estimation for Autonomous Driving

Bc. Libor Novák

May 2017
Supervisor: prof. Ing. Jiří Matas, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student:	Bc. Libor N o v á k
Study programme:	Open Informatics
Specialisation:	Computer Vision and Image Processing
Title of Diploma Thesis:	Vehicle Detection and Pose Estimation for Autonomous Driving

Guidelines:

1. The 3D bounding box is a compact, yet powerful representation of vehicle state. Review the literature on 3D bounding box detection and estimation of its parameters from images or videos.
2. Select a standard deep neural network (DNN) method for 3D bounding box estimation. Train and test the selected method on datasets for 3D bounding box detection of vehicles in traffic in any rotation with respect to the camera.
3. Suggest and implement a modification of an existing algorithm or propose a new 3D bounding box detector.
4. Evaluate the proposed detector on standard datasets.

Bibliography/Sources:

[1] Chen, Xiaozhi, et al. "Monocular 3D object detection for autonomous driving." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.
[2] L. Huang, Y. Yang, Y. Deng, and Y. Yu, "Densebox: Unifying landmark localization with end to end object detection," arXiv preprint arXiv:1509.04874, 2015.
[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed, "SSD: Single shot multibox detector," arXiv preprint arXiv:1512.02325, 2015.
[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

Diploma Thesis Supervisor: prof. Ing. Jiří Matas, Ph.D.

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
Head of Department	**Dean**

Prague, January 6, 2017

Acknowledgement / Declaration

I would like to thank my family for all the support they provided me during the endless time of my studies. Special thanks belong to my supervisor as he made it possible for me to work on a very interesting topic and to Jiri Trefny for providing me with an image labeling tool and the comparison to his WaldBoost detector.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 23. 5. 2017

. . . .

Anotace / Abstract

Tato diplomová práce představuje plně konvoluční síť pro detekci 2D a 3D bounding boxů aut z obrázků, se speciálním zaměřením na využití v autonomním řízení vozidel. Oproti předcházejícím metodám, které používají neuronové sítě pro detekci 3D bouniding boxů, síť představená v této práci je trénovatelná tzv. end-to-end a umí detekovat objekty v různých velikostech během jediného zpracování.

Je uvedena nová reprezentace 3D bounding boxů, která je nezávislá na matici kamery (kameře použité pro snímání obrázků). Tato vlastnost umožňuje, aby byl detektor trénován na několika různých datasetech najednou a zároveň mohl detekovat 3D bounding boxy na úplně jiných datasetech, než byl trénován.

Prezentovaná síť dokáže zpracovávat 0.5 MPx obrázky z KITTI datasetu v rychlosti 10 snímků za sekundu, což je přibližně o řád rychleji, než nejrychlejší síť, která má lepší výsledky detekce. Z tohoto důvodu může být aplikovaná v autonomním řízení.

Klíčová slova: detekce automobilů, neuronové sítě, strojové učení, zpracování obrazu.

The thesis presents a fully convolutional neural network for 2D and 3D bounding box detection of cars from monocular images intended for autonomous driving applications. In contrast with previous deep neural network methods applied to 3D bounding box detection, the introduced network is end-to-end trainable and detects objects at multiple scales in a single pass.

We introduce a novel 3D bounding box representation, which is independent of the image projection matrix (camera used to take the images). Therefore, the detector may be trained on several different datasets at a time and also detect 3D bounding boxes on completely different datasets than it was trained on.

The presented multi-scale end-to-end network is capable of processing 0.5MPx KITTI images in 10fps, which makes it about an order of magnitude faster than the closest competitor that has superior detection results. Therefore, it is possible to be used in autonomous driving scenarios.

Keywords: car detection, 3D bounding box, deep neural networks, deep learning, machine learning, image processing.

Contents /

Chapter 1
Introduction

The demand for driverless vehicles is rising in both the public and the commercial sectors. In the public sector, people demand safer and less time consuming means of transportation and on-demand services, which would be available within minutes. In the private sector the motivation comes mainly from the effort to increase the reliability and utilization of transportation vehicles. For example, an autonomous truck can be driven nearly 24 hours a day, whereas human-driven vehicles have to stop for the driver to rest.

Achieving fully autonomous driving in urban environment is a very challenging task and, today, it is one of the main drivers of the development of a broad range of new technologies. Achieving this goal can be compared to the space race in the 1960s. Undoubtedly, space race lead to many inventions and technologies used not only for space space exploration, but also in industry, health care, etc. The development of autonomous driving systems has had a similar impact.

Object detection belongs to the core abilities of an autonomous systems as they are required to perceive the surrounding environment. In the academic field, general object detection and classification dominates. It has been a very extensively studied problem and many classical computer vision approaches exist, e.g. [52, 49, 18, 16]. Recently, deep learning and deep neural networks stole the show. In 2012, Krizhevsky et al. [29] managed to beat all classical computer vision methods in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [13].

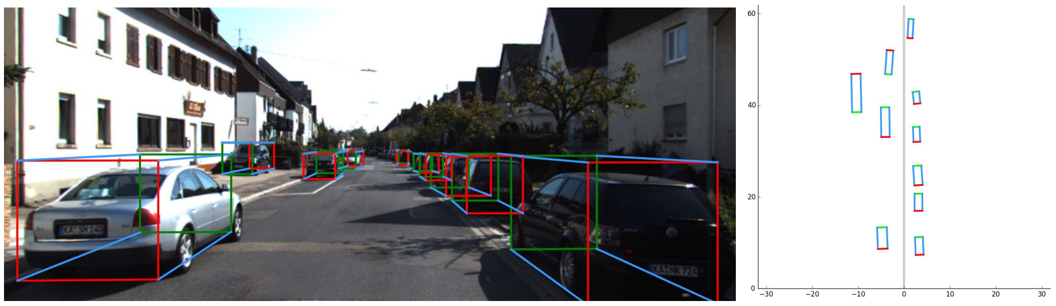


Figure 1.1. 3D bounding boxes (left) and their top view (right) detected by the proposed method. Note the imprecision in the top view when the detected cars do not lie exactly on the ground plane. The front sides of 3D bounding boxes are depicted in green, the rear sides in red.

Currently, systems based on deep neural networks are the state-of-the-art in image classification and object detection [41, 39]. Since the impressive achievement of Krizhevsky et al. [29], deep neural networks have been successfully applied to various kinds of problems, for example [26, 58, 47, 53]. This proves that they are a very powerful tool, which may be easily and effectively adapted to tackle a large variety of tasks.

The topic of this work is detection of cars and representing their pose with 3D bounding boxes. The 3D bounding box (Fig. 1.2 right) is a very convenient and, for many applications, sufficient representation of the objects in the 3D world. We, in line with general practice, define the 3D bounding box as a tight rectangular cuboid around an object that has 9 degrees of freedom (3 for position, 3 for rotation, and 3 for dimensions). This information is sufficient to determine the position, orientation, and size of the object in the 3D world, which can be used especially for path planning of an autonomous car.

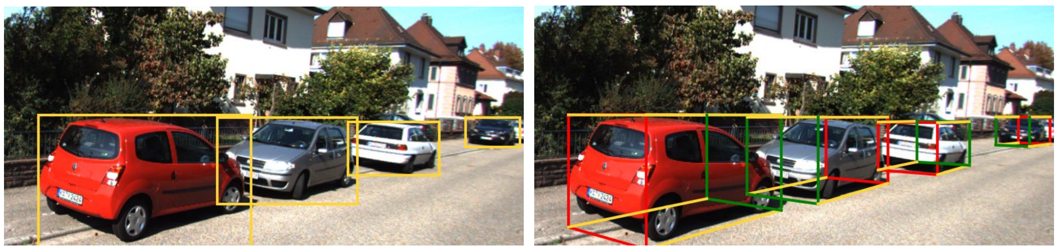


Figure 1.2. 2D bounding box (left) and 3D bounding box (right) annotation.

The vast majority of existing object detectors focuses on finding 2D bounding boxes ([21, 46, 40, 33] among others), which provide sufficient information for basic reasoning about object positions (Fig. 1.2 left). However, it is insufficient for autonomous driving applications, where finding poses of objects in the 3D world is desired. This problem has been tackled by a new voxel representation with more extensive preprocessing [55], 3D bounding box proposal generation [9] or estimation of 3D bounding boxes from detected 2D bounding boxes [35]. These methods employ multi-step approaches, which makes them slow and more difficult to train. Instead, we took a different approach.

We chose to implement an end-to-end system for 3D bounding box detection from a mono camera (Fig. 1.1) by combining the current state-of-the-art methods for 2D bounding box detection [40, 33, 27]. We opted for a DenseBox-like [27] approach, which we adapted to regress the positions of 3D bounding boxes. On top of that, the network architecture was changed to detect objects of several sizes in a single pass instead of building an image pyramid. The whole network can be trained end-to-end from images to detect projections of 3D bounding boxes of objects of different sizes from a single image (monocular system). The projections can then be reconstructed in the 3D world providing that the ground plane is known.

1.1 Contributions

The idea of DenseBox [27] was applied to the detection of 3D bounding boxes in a single (mono) image. A novel network architecture inspired by SSD [33] and MS-CNN [5] was introduced to directly accommodate detection of objects at multiple scales. Specifically, we used the idea of extraction of results from multiple network layers at the same time.

We suggested a new, compact representation of 3D bounding boxes, which is independent of the camera (image projection matrix). It uses the projections of the 3D bounding box corners into the image instead of storing real-world parameters. This makes the detector capable of detecting 3D bounding boxes in images from any dataset regardless of the used imaging system. The 3D world bounding boxes can then be reconstructed when the correct image projection matrix and the ground plane equation is provided.

The thesis makes contributions to the Jura vehicle test set, which consists of challenging images of cars in various poses from roundabouts in Prague and Brussels. We refined and completed the 2D bounding box annotation to include cars of all sizes and occlusions, which the labeler was able to recognize just by looking at a single image. This was important in order to carry out correct evaluations of the method.

1.2 Overview of the Work

First, we carry out a thorough review of the state-of-the-art methods in deep learning applied to object detection and we provide justifications for selecting DenseBox as our base method, which is further extended to directly support multi-scale detection as done in SSD and MS-CNN.

The next chapter provides an insight into the theory behind deep neural networks applied to our method. We describe the input and output representation used in our network and provide the derivation of back-propagation for the used loss function. At the end, the network architectures used in this work are shown.

The network description is followed by the description of the used datasets and data representations. The BBTXT and BB3TXT label formats are introduced and the 3D bounding box reconstruction procedure is explained in detail.

In Chapter 5, we justify the choices made while designing our neural network and provide evaluation of the best 2D and 3D detection networks.

Finally, we discuss the problems of the proposed method and suggest their solutions and further enhancements.

Chapter 2
Related Work

The chapter reviews the recent advances in deep learning approaches to object detection and points out the ones, which relate to car detection in urban scenarios, i.e. autonomous driving scenarios. Also, we justify the selection of the used DNN method.

2.1 Overview

The new era of deep neural networks started in 2012, when Krizhevsky et al. published their results [29] on the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [13]. Their deep neural network (DNN) with 5 convolutional layers followed by 2 fully-connected layers beat their closest competitor based on SIFT [45] by approximately 10% in classification error rate. This moment was a huge breakthrough for DNNs as they showed to be superior to classical computer vision methods. There were multiple factors that led to this achievement, for example large amount of data available for training the network, more powerful hardware able to carry out orders of magnitude more training iterations than previously possible, new training algorithms and network architecture with convolutional layers, and other.

Since then, many DNN methods have been developed for image classification and object detection. In the next section we review those methods. The existing methods can be divided into two groups: region-proposal methods and end-to-end-learning methods. The former consist of a region proposal step and a DNN classifier of the proposed regions, the latter is a DNN taking raw images and directly outputting information about detected objects.

2.1.1 Region-proposal Methods

Krizhevsky et al. [29] showed that DNNs are a powerful tool for image classification. A conversion of the classification task to the detection task can be achieved by classifying every possible sub-window of an image. However, this approach requires enormous computational power as a classifier must be run thousands of times per image. The idea of region proposals addresses the issue. It aims to reduce the number of evaluated windows (runs of a classifier) per single image by suggesting promising locations using an algorithm that would be orders of magnitude faster than the classifier. Take selective search [51] or edge boxes [59] as examples of region proposal methods.

R-CNN of Girshick et al. [21] makes use of selective search. It is a three-stage object detector with selective region proposal generator, further processed by a DNN (same as was used by Krizhevsky et al. [29]) into a 4096 feature vector, which is then classified by an SVM. This approach performed well, but was quite slow as it took about 18s per image on a Tesla K20 GPU.

Spatial pyramid pooling (SPPNet) was introduced by He et al. [24] to speed up the computation of R-CNN. It avoids repeated evaluation of the whole DNN on each proposal window by running the convolutional layers on the whole image and then pooling the feature maps - creating a fixed-dimensional grid (1×1, 2×2, 3×3 and 6×6) across the feature map. The fixed size feature vector (4096 elements) for each object proposal is then created by concatenation of the pooled vectors from the grid cells.

SPPNet inspired the authors of R-CNN to create the Fast R-CNN [20] version of their algorithm. It speeds up computation the same way as in SPPNet, however only one pyramid layer (7×7) was used. Also, the SVM on the output is replaced by fully connected layers, which predict class probabilities, and interestingly also bounding box coordinates directly.

In the meantime, Simonyan et al. were experimenting with very deep convolutional networks [48], where they used only 3×3 convolution and created the very successful VGGNet, which has up to 19 layers and is used in many following papers, for example [43, 10].

One of them was Ren et al., who introduced Faster R-CNN [43] that make use of the VGG-16 net architecture. On top of that, they replace selective search with a region proposal network (RPN), which shares convolutional layers with the classification DNN. The RPN is basically a sliding window on the feature map, which outputs objectness score and 9 bounding boxes (for 9 anchor boxes - 3 scales, 3 aspect ratios) per window position. These proposals are then classified as in Fast R-CNN. This algorithm is able to achieve about 15fps and performs very well.

An attempt was made by Lenc et al. [32] to drop the region proposal part from the Faster R-CNN framework, however the results show that the effort was not successful.

The Faster R-CNN framework was even further improved by He et al. [25] and their ResNet. It adds residual connections - shortcuts to the network, which make the network learn only the difference between the input and output of building blocks. This allows for training extremely deep networks with SGD because their design does not suffer so much from gradient decay. Despite being so deep, the networks are less complex than VGG-16 [48]. This is one of the latest state-of-the-art performing networks on ImageNet.

3D Bounding Box Detection for Autonomous Driving Applications

A very sophisticated detection framework 3DOP was introduced by Chen et al. [10]. It uses stereo cameras to create a 3D point cloud, which is then used to find the ground plane and score 3D box proposals (for cars, pedestrians, and cyclists), which are generated on the ground plane (2 orientations ($0°$ and $90°$) and 3 box templates per class) in the places, where the point cloud is dense. The proposals are scored with energy, which is based on measures from the point cloud. They use 3D integral image and voxels to compute the emptiness of the space regions. The VGG-16 [48] architecture is adapted to score the bounding boxes and also regress the orientation of the objects. This is one of the current published state-of-the-art on the KITTI dataset [19].

Mono3D [9], an adaptation of 3DOP, was made to exclude the need for stereo images. No point cloud is build, however the knowledge of the position of the ground plane is assumed. Also, fully convolutional networks (FCNs) from external sources are used to propose bounding boxes from object and class segmentations. Its performance is slightly worse than in the stereo case [10] and the pipeline is even more complicated.

MS-CNN of Cai et al. [5] adapts an alternative approach. Instead of resizing the image on the input or the feature layers they train a set of object proposal networks, which are attached to different depths of the convolution. This results in multi-scale region proposals, which are then converted to feature vectors by ROI pooling and evaluated by a fully connected network as in Fast R-CNN. This is also one of the current published state-of-the-art on the KITTI dataset [19]. Our method was inspired by their way of extracting detections from different layers of the network.

An interesting work called SubCNN was published by Xiang et al. [55]. It makes use of their novel voxel-pattern-based representation [54], which represent cars of different viewpoints, occlusions, and truncations. Groups of such patterns represent subcategories, which are detected on the output of their SubCNN network. Interestingly, since the voxel patterns define spatial models, 3D segmentation and bounding box is obtainable from the prototype database. However, the net is still just a classifier, which requires the region-proposal stage.

Mousavian et al. [35] took a different approach to 3D bounding box detection in their Deep3DBox. From 2D bounding boxes provided by [55] they estimated the 3D bounding boxes. The projection of the 3D bounding box is constrained by the 2D bounding box as it has to tightly fit inside. They use DNNs to estimate the orientation and size of the 3D bounding box from the 2D one and then solve for the rest of the unknowns. This is very interesting, since the dimensions of the 2D bounding box relate not only to the distance of the car from the camera and its orientation, but also to the real-world size of the car (SUV, sedan, etc.). The network thus has to learn the relation of a type of a car to its real-world dimensions.

2.1.2 End-to-end Systems

Another approach was taken by Sermanet et al. in OverFeat [46]. They avoided the use of object proposal generator by using a fully convolutional network (FCN) to predict objects for each grid cell of a width/12 $\times$ height/12 grid and then fused the predictions to yield the output locations. This approach showed to be faster, but worse than R-CNN [21].

A region proposal-free approach was suggested by Redmon et al. [40]. Their YOLO network divides the input image into a 7×7 grid of cells, which are responsible for detecting objects with bounding box centers inside them. It has several convolutional layers followed by 2 fully connected ones. This makes it possible for the network to learn priors on the object positions within the input image. Also, very importantly, it is an end-to-end trained neural network.

Other proposal-free networks are based on the fully convolutional framework (FCN), where a prediction is extracted for each part of the input image. Such networks are mainly used for image segmentation [34, 7, 38, 8], however, when carefully designed, they can be used for object detection as well.

Recently, YOLO9000 [41] was introduced, which builds on YOLO, however the network is converted into a fully convolutional scheme. The output of the network was changed to a 13×13 grid for a 416×416 input image and the bounding box coordinates are not predicted directly, but anchor boxes [43] were incorporated into the output layer. The network can run in 40fps and performs on par with SSD [33] and ResNet [25].

SSD, another end-to-end approach was introduced by Liu et al. [33]. The architecture of this network is almost the same as in YOLO9000 as it is also a fully convolutional network, which outputs class confidences and bounding box differences from anchor boxes [43]. The difference is that YOLO9000 concatenates features from 2 feature maps of different sizes and trains a convolutional layer on top of these concatenated vectors, whereas SSD trains a convolutional filter (layer) for each scale separately. Therefore, SSD has more detection windows on the output. The performance of SSD is on the state-of-the-art level on the Pascal VOC dataset [14].

Another well performing FCN framework on KITTI is DenseBox [27]. They use a fully convolutional network to get a map of probabilities and bounding box coordinates of objects. The probabilistic map is basically a Hough Map [3], where the pixels that are in the center of the object are responsible for detecting it. It works very well on the KITTI dataset [19] and therefore can be used for our purposes.

2.2 DenseBox - The Selected Base Method

A car detector for autonomous vehicles has certain requirements, which have to be met in order for the detector to be usable. It has to be able to detect cars in all possible viewpoints, deal very well with occlusion, which in the case of urban areas may be very extensive. Lighting conditions also vary from very dark to very bright, including the case when sun shines into the camera. Also, the detector must be reasonably fast and robust and, finally, provide the autonomous car with the 3D poses of the surrounding cars.

The current state-of-the-art 3D bounding box detectors based on DNN [35, 55, 9–10] can deal with the aforementioned problems really well except for the time time to detection. In Tab. 2.1 we see the comparison of the times it takes for each method to process one image with dimensions 1240×375 (0.5MPx) from the KITTI dataset on a GPU. All times exceed 1s, which is very unsatisfactory for an application in a real time system such as an autonomous car.

Method	Description	TTD
Deep3DBox [35]	SubCNN RPN, DNN for pose estimation	1.5s
SubCNN [55]	RPN, DNN for subcategory classification	2.0s
3DOP [10]	Point cloud 3DBB proposal, DNN refinement	3.0s
Mono3D [9]	Segmentation 3DBB proposal, DNN refinement	4.2s

Table 2.1. Comparison of the time to detection (TTD) of the best performing 3D-bounding-box-detecting DNN methods on the KITTI dataset. Data taken from the KITTI scoreboard. RPN: Region Proposal Network, 3DBB: 3D bounding box.

If we look at the state-of-the-art end-to-end systems for 2D bounding box detection, such as YOLO [40–41] or SSD [33], they are all able to run in a frame rate larger than 20fps on an image with dimensions about 500×500 (0.25MPx). This is at least 15 times faster than the previously mentioned detectors, therefore, it makes the end-to-end methods obvious candidates for our purpose.

Currently, one of the best performing published end-to-end systems on the KITTI dataset is DenseBox [27]. It uses a very natural structure of the output as it estimates a *probabilistic map* of object centers across the whole image, see Fig. 2.1 for illustration. One can compare its nature to the Hough accumulator [3], used for example in [18].

The detected objects are then extracted as the maxima from this probabilistic map. This can be done for an image of arbitrary size as it is a fully convolutional network. When thinking about this output representation, one can notice similarities to YOLO or OverFeat [46], which can be thought of as having more coarse grids of responses.

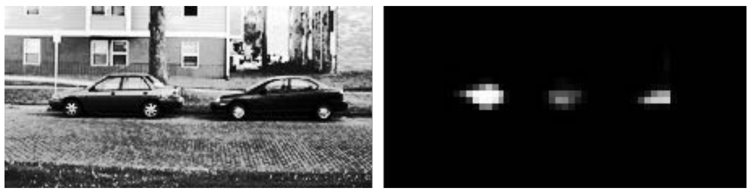


Figure 2.1. Sample image (left) and a probabilistic map of object centers (right).

DenseBox also suffers from the detection speed problem, but that is because it is not such a mature and well crafted system as YOLO or SSD. YOLO and SSD process each image in one pass, whereas DenseBox processed an image pyramid. The network can be changed to work in a similar manner and with a similar number of parameters it should achieve comparable speeds. This shows that DenseBox has a lot of potential.

Considering that the output of DenseBox can be easily adapted to predict the image projections of the corners of 3D bounding boxes and its loss function is way simpler to compute than in the case of YOLO or SSD, we chose to use a DenseBox-like output structure. It allows us to easily train, test our progress, and tune parameters on a single scale detector, which we then extend to a multi-scale one-pass detector. The architecture of the multi-scale network is inspired by SSD and MS-CNN [5] in that we extract probabilistic maps for different scales from different levels of the detection network as illustrated in Fig. 3.2.

NOTE In general, we could have chosen any of the three (YOLO, SSD, DenseBox) methods and change its output to generate 3D bounding boxes, however, another argument in favor of DenseBox was that it is much easier to implement and test on a small scale, which was very helpful while taking the first steps.

Chapter 3
Bounding Box Detection using DNN

This chapter contains a detailed description of the proposed network. We present the chosen input and output representations, the loss function, and a discussion of the network design choices and describe the resulting network designs.

3.1 Quick Overview of the Method

An artificial neural network is a non-linear projection from an N-dimensional space $X^N \subset \mathbb{R}^N$ to an M-dimensional space $Y^M \subset \mathbb{R}^M$, which can be described as a composition of linear and non-linear functions $l_i : X^{N_i} \rightarrow Y^{M_i}$ called layers. Formally, an artificial neural network with L layers may be written as

$$\mathbf{y} = (l_L \circ ... \circ l_2 \circ l_1)(\mathbf{x}), \qquad \mathbf{y} \in Y^M, \mathbf{x} \in X^N.$$

The layers (except for the pooling layers) consist of neurons - elementary units performing the following operation

$$l_{ij}(\mathbf{x_i}) = \phi(\mathbf{x_i} \cdot \mathbf{w_{ij}} + b_j),$$

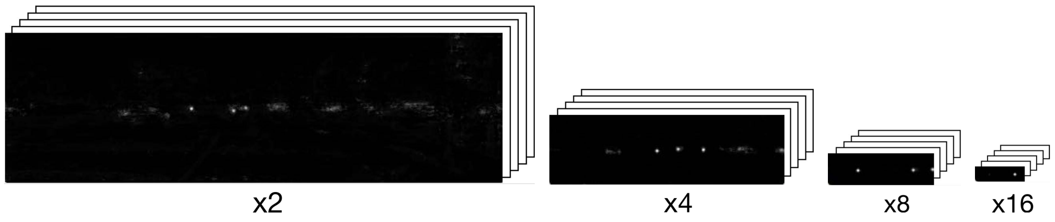
where $\mathbf{w_{ij}}$ represents the weights applied to the elements of the layer input vector $\mathbf{x_i}$, b_j is the bias, and ϕ is the activation function, which introduces non-linearity to the network. This is done for every neuron j in the layer i, therefore again producing a vector $\mathbf{y_i}$ of output values. Many types of layers and activation functions exist. As artificial neural networks are a widely studied topic, we refer to [30, 23, 37] for further details on the theoretical part and instead describe the specifics of our design.

Our artificial neural network has been designed to detect 2D and 3D bounding boxes of cars from RGB images of any resolution captured by a monocular camera. We used the very well known fully convolutional design in combination with the output representation introduced by DenseBox [27]. This allows us to detect cars of various sizes, including very small ones (from 20px of the longer side of their 2D bounding box), and run the detector on an image of any resolution, providing that the network fits into the GPU memory.

In short, one can think of our fully convolutional network (FCN) as a large window sliding on the input image, which outputs the *probability*[1]) of that window having a car in its center for each window position on the input image (Fig. 3.1b). Another point of view can be that the *pixels* of the output response map, which correspond to the position of an object center (center of its 2D bounding box) are responsible for detecting that object. Intuitively, the response maps in Fig. 3.1b assimilate the very well known

[1]) We will be calling this probability, however it is not probability in the statistical sense as it only measures the strength of the response of the network, more like confidence.

a) A busy intersection with cars coming from all directions.



b) Object center *probability* response maps (black) and response maps with 2D bounding box coordinates (white) in scales denoted by xs, where s is the down-sampling factor with respect to the input image. The larger the scale s, the larger objects the response map detects.

Figure 3.1. Sample input image (a) and the corresponding response of a network for 2D bounding box detection (b). In b) we show several scales of response maps down-sampled by factor s, which are used to perform multi-scale detection.

Hough accumulators [3] from the Hough-transform-based object detectors. Yet another point of view may be that the output response map is an extended grid (more dense) of responses used in YOLO [41] or SSD [33].

Fig. 3.1b contains several scales of the mentioned response maps. This is because we are performing multi-scale car detection, which may be achieved either by building an image pyramid from the input image as it is done in DenseBox or, as we did it, extracting the response maps from several different layers of the network. This idea was previously used in SSD [33] and MS-CNN [5] and is illustrated in Fig. 3.2. It is a very convenient approach because it performs multi-scale detection in one pass through the network as opposed to the repeated evaluation in the image pyramid case. For this reason it is way faster. It also allows the network to learn more and more precise description for larger and larger objects.

3.2 Input Layer

The network takes 3-channel RGB images on the input. For detection, the images can be of arbitrary resolution[1]), in training, however, we use images of fixed dimensions (for example 128×256) in order to be able to process multiple images in a single batch.

The intensities of the image pixels $[0, 255]$ are converted to span the interval $[-1, 1]$ in the following way:

$$\frac{v - 128}{128},$$

[1]) Providing the network fits into the GPU memory.

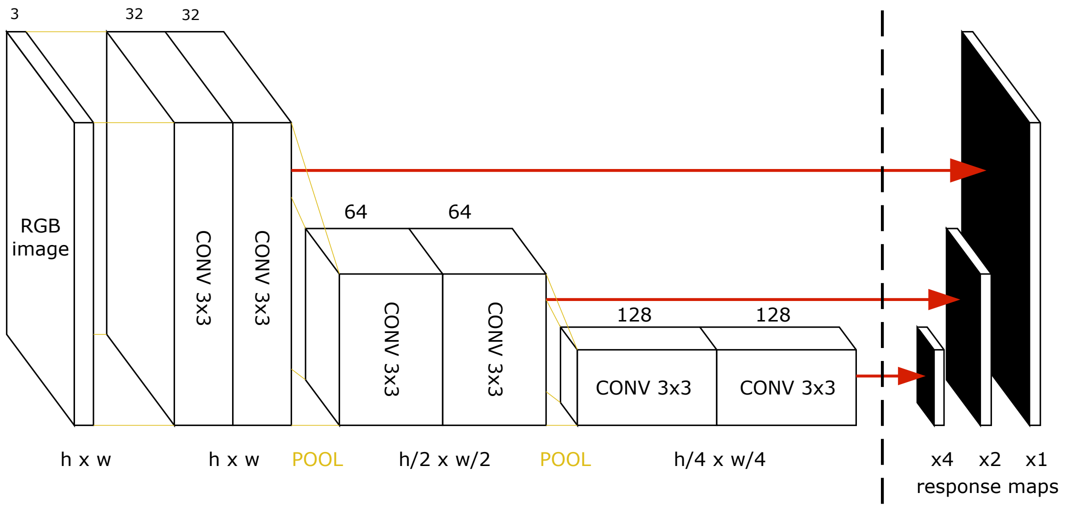


Figure 3.2. Extraction of response maps for multi-scale detection. The red arrows show where the output response maps are extracted, their scale is denoted xs (i.e. x1, x2, x4). The scales s used on the output may be chosen arbitrarily based on the sizes of the objects we want to detect. The extraction is carried out using 1×1 convolution with 5 filters in the case of 2D bounding box and with 8 filters in the case of 3D bounding box detection.

where v stands for the original value, since values between -1 and 1 are more suitable for neural networks. In training, the dataset is randomly shuffled and the images are randomly augmented with exposure and hue shifts and color noise.

3.2.1 Bounding Box Sampling

The detector has to learn to detect cars (bounding boxes) of various sizes. The original sizes in the training set would not be enough, and hence we perform cropping of bounding boxes to random scales.

We want to make sure that we have at least one bounding box in each training image. Therefore, we randomly select a bounding box from the training set. The bounding box size $size_{orig} = \max(w, h)$, where w and h stand for the width and height of the bounding box respectively. Then we randomly select a size $size_{new}$ (uniformly from a user-defined interval) in which the bounding box will be cropped and rescale the whole image so that the bounding box matches this new size $size_{new}$. Then, a random crop of the dimensions of the net input is extracted from the rescaled image. The random crop is however restricted such that it has to contain the whole selected bounding box. Examples of the sampled images with highlighted bounding boxes are shown in Fig. 3.3.

When training a multi-scale detection network, one has to adjust the size $size_{new}$ sampling distribution. Intuitively, the length of the interval of sizes grows for increasing scales s (Fig. 3.7). If we sampled uniformly from the interval of all sizes, we would end up sampling more bounding boxes for larger scales. We require all scales to have the same amount of training data (bounding boxes), therefore we have to convert the $size$ sampling distribution. We want for our new distribution to hold that

$$f(2kx) - f(kx) = const, \qquad k \in \mathbb{N}$$

because the interval of sizes doubles for each two consecutive scales $s = 2^i, 2^{i+1}$. For example the interval for $s = 2$ might be $[30, 60]$ (length 30) and for $s = 4$ $[60, 120]$

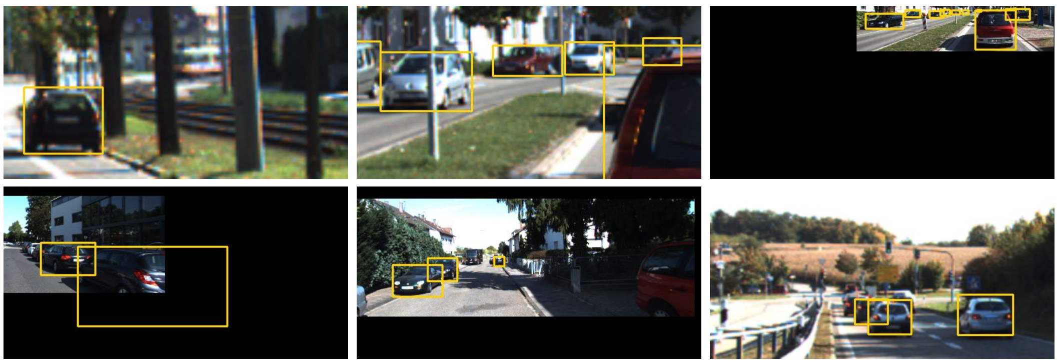


Figure 3.3. Examples of sampled training images from the KITTI dataset with highlighted ground truth.

(length 60). Selecting $f(\cdot) = \log(\cdot)$ satisfies the requirement

$$\log(2kx) - \log(kx) = \log\left(\frac{2kx}{kx}\right) = \log 2 = const.$$

Uniformly sampling values from the $\log(size)$ space gives us the required property.

3.3 Hidden Layers

Our network uses two very well known types of hidden layers - convolutional layers and pooling layers. We do not use any fully connected layers as our design is fully convolutional.

3.3.1 Convolutional Layer

Convolution is an operator used in signal processing, which takes two functions f and g and produces a third function $(f * g)$ defined as

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m],$$

where n and m are integers (this is the discrete version of the definition of convolution). We can interpret this as sliding a reversed linear filter across a vector. This definition can be extended into two dimensions, where the notion translates to sliding a flipped 2D linear filter (matrix) across a matrix. In convolutional layers of neural networks we omit the flipping since the parameters of the filter are learned and therefore the flip is irrelevant.

Following the above definition, the convolution operator produces a matrix, which is on each side extended by the half of its size, i.e. convolutional operator 3×3 produces a matrix of size $(h+2) \times (w+2)$ (with non-zero values). However, we drop the extension and keep the size of the matrix the same. This is convenient to keep the design of the network organized.

When designing a network, one requires the output neurons to have a certain field of view (the size of the part of the input image from which the input is taken). The field of view (FOV) of the neurons - convolutional operators can be extended by enlarging the filter (e.g. to 7×7) or by stacking several convolutional layers on top of each other. In the latter case, each input neuron of a convolutional layer already has a certain FOV, which enlarges the FOV of the neurons (operators) in the next layer (see Sec. 3.6 for details on how to compute the FOV). This is the approach we used as it was shown in [48] that it uses much less parameters and there is no drop in performance. It also saves the memory required to compute the convolutional layer output.

The FOV of a convolutional operator can be increased even more by using so called dilated convolution or *atrous* convolution, described in [57]. Dilated convolution introduces holes in between the operator elements and therefore spans the operator over a larger area of the input matrix. Conveniently, when stacked, these convolutions can cover a very large area without reducing the resolution of the output matrix. See Fig. 3.4 for illustration. Using the dilated convolutions is a design change over the DenseBox architecture, where they achieve larger FOV by adding extra pooling layers and deconvolving the result back to the required resolution. We argue that dilated convolution is superior to their approach as it does not perform any extra resolution reduction.

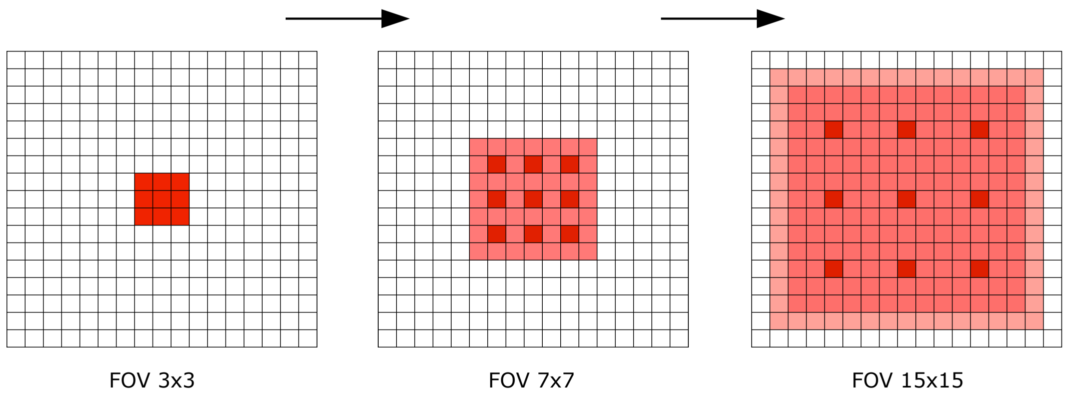


Figure 3.4. The expanding FOV of repeatedly applied dilated convolution. (left) is classical 1-dilated 3×3 convolution; (middle) is 2-dilated convolution applied on the (left) result; (right) is 4-dilated convolution applied on the result of (middle).

During training, we look at convolutional layers as weight-sharing layers. It is because the same convolutional operator with the same learned weights is applied many times on the input matrix. When updating its weights, the gradient from all positions, where the operator was applied is summed up - i.e. shared. This, however, introduces a problem. If a large input matrix is used for training, the sum will have more operands and the update will be larger because of this. It is a known problem[1]) and one has to keep in mind that the learning rate needs to be adjusted accordingly.

ReLU Following current trends, after each convolutional layer, except the output ones, we apply the rectified linear unit activation function $\max(0, x)$, evaluated in [12].

3.3.2 Pooling Layer

Pooling layer provides an informed dimension reduction of the input matrix. We use max-pooling layers with kernel size 2×2 and stride 2 in our network to scale down

[1]) `https://github.com/BVLC/caffe/issues/3242`

the input matrix exactly by the factor of 2. Using pooling layers is arguably superior to using convolutional layers with stride 2 to do dimension reduction because such a reduction is not informed and just blindly skips every other input position.

3.4 Output Layer(s)

As already mentioned, we use the output representation taken from DenseBox [27]. We have either 5 (for 2D bounding box) or 8 (for 3D bounding box) channels in the output layer (see Fig. 3.5), whose dimensions are down-sampled with respect to the input image by the scale factor s. In DenseBox, the factor $s = 4$ is used, however, in our case we are performing multi-scale detection directly in one pass through the network (Fig. 3.2), hence we have several scales s on the output, usually $2, 4, 8$ or $2, 4, 8, 16$, depending on what sizes of objects we aim to detect.

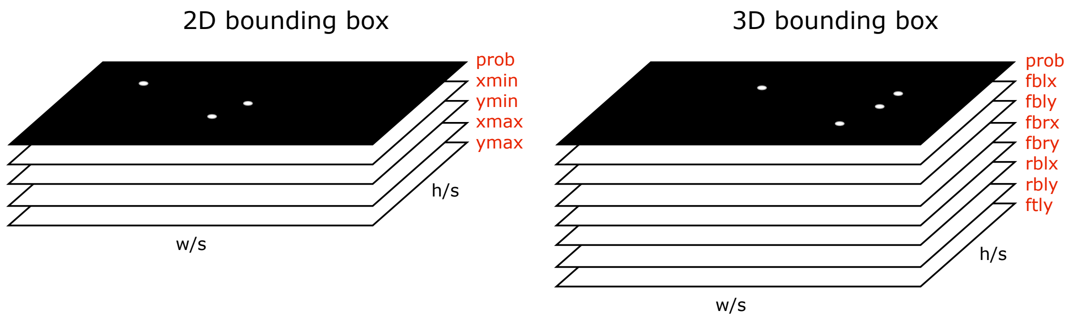
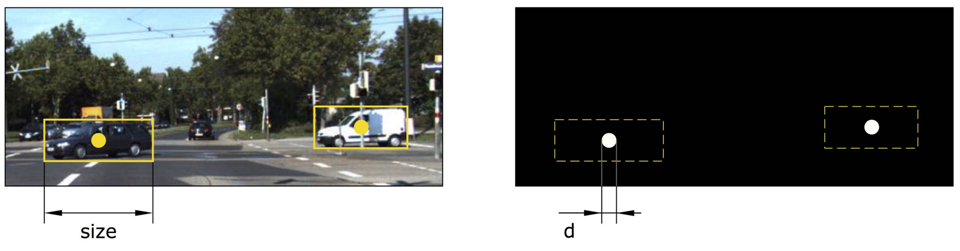


Figure 3.5. Illustration of the representation of the 2D and 3D bounding boxes on the network output in scale s.

3.4.1 Target Representation

The pixels in the response map within some given radius $r = d/2$ from an object center (center of its 2D bounding box) are responsible for detecting that object and regressing the coordinates of its 2D or 3D bounding box (Fig. 3.6). In the case of a 2D bounding box we regress the coordinates of the top-left and bottom-right corners of the 2D bounding box. In the case of a 3D bounding box we regress the position of the projections of the rear-bottom-left, front-bottom-left, and front-bottom-right corners and the y-coordinate of the front-top-left corner. For a detailed description of the 2D and 3D bounding box representation see Sec. 4.2.1 and Sec. 4.2.2.



a) Annotated image.

b) Target response map.

Figure 3.6. 2D bounding box annotation and the corresponding ground truth *probability* response map required on the output of the network (*black* = 0, *white* = 1). The center of each object in the current scale must be detected. The affiliation of each object to a certain scale s is determined by its size (see Fig. 3.7).

The choice of the radius $r = d/2$ (in Fig. 3.6) of the circle of pixels responsible for detecting an object is arbitrary. However, we relate it to the $size = \max(w, h)$ of the object 2D bounding box. Providing the circle ratio

$$cr = \frac{2 * r + 1}{size} \cdot s$$

and selecting r directly determines the $size$ of objects that should be detected in the response map of the scale s. For example, choosing $cr = 0.25$ and $r = 2$ gives $size = 80$ for scale $s = 4$, which means that the response map of scale $s = 4$ should detect objects of size around 80 pixels. Notably, this provides the **ideal size** of object, which should be detected in the response map of the scale s, however, in reality, we need to provide a span of sizes of objects that will be detected by a certain scale s.

The scales of response maps in our network are a sequence of $2^i, i \in \mathbb{N}$ numbers. This comes from the fact that each pooling layer shrinks the input matrix by $1/2$ on each side. We chose to make each scale s_i responsible for detecting objects in the following span (interval) of sizes

$$\left[\frac{size_i + size_{i-1}}{2} - o_i^L, \frac{size_i + size_{i+1}}{2} + o_i^R\right],$$

where o_i^L and o_i^R is the left and right overlap of the bounds respectively. The overlap is provided in order to smoothen the boundary between two neighboring scales. It also means that some objects may be detected in two response maps of different scales. Fig. 3.7 illustrates this.

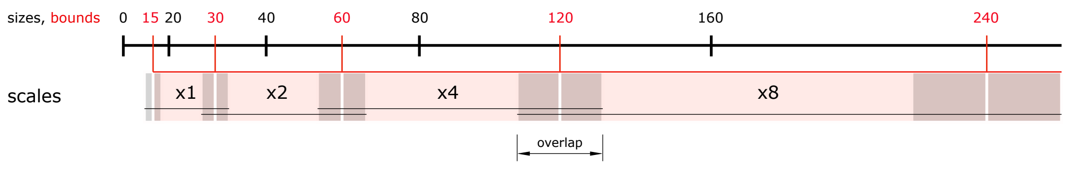


Figure 3.7. The distribution of object sizes into different response map scales xs ($cr = 0.25$, $r = 2$). Each response map is responsible for detecting objects within the given bounds plus the overlap with neighboring scales.

Gaussian response In our target *probability* response map we require either 0 for the pixels out of the object centers or 1 for the pixels in the object centers. This results in a steep change in the error on the edge between a correct and incorrect detection when the network output is slightly misplaced (Fig. 3.8). Using a Gaussian instead guides the learning algorithm better towards the required position. Therefore, we filter the ground truth *probability* response map with a 3×3 Gaussian filter with $\sigma = 1$. This results in having Gaussian blobs in the positions of the circles and smoother transitions (smaller error) for small misplacements in the response maps. For $r = 2$ the Gaussian looks as shown in Fig. 3.9.

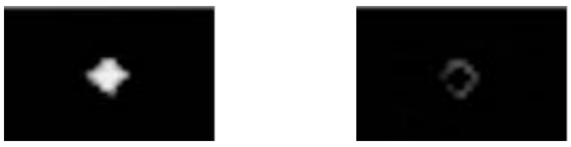


Figure 3.8. The response of a network trained on a binary response map (left) and its error when compared to ground truth (right). The *ring* in the left image corresponds to the steep change in the error, which is undesired.

0	0	0.075	0.123	0.075	0	0
0	0.075	0.322	0.478	0.322	0.075	0
0.075	0.322	0.677	0.849	0.677	0.322	0.075
0.123	0.478	0.849	1	0.849	0.478	0.123
0.075	0.322	0.677	0.849	0.677	0.322	0.075
0	0.075	0.322	0.478	0.322	0.075	0
0	0	0.075	0.123	0.075	0	0

Figure 3.9. The Gaussian blob used instead of the binary circle with $r = 2$ in the *probability* response map.

Coordinate response maps So far, we have been describing the *probability* response map channel, however, the output contains other 4 (or 7) channels (Fig. 3.5) that regress the 2D (or 3D) bounding box coordinates. The coordinates are necessarily relative to the coordinate map pixel position in which they are regressed as we are using a FCN. They are regressed in each pixel (understand pixel as a sample with 5 or 8 channels), which has a value > 0 in the *probability* channel. This is because each positive pixel (sample) in the net output represents a bounding box.

The relative coordinate values are scaled to approximately $(0, 1)$, which is more suitable for training of the network - provides gradients of similar magnitude to the ones from the *probability* channel. The relative coordinate value v is converted to the value v' in a coordinate response map in the scale s_i as follows

$$v' = \frac{v}{size_i} + 0.5,$$

where $size_i$ is the ideal object size for the scale s_i. This assures that a 2D bounding box with the dimensions $size_i \times size_i$ will have coordinates in the relative coordinate maps $x'_{min} = 0$, $y'_{min} = 0$, $x'_{max} = 1$, and $y'_{max} = 1$. Other bounding box dimensions will have values around 0 and 1.

3.4.2 Loss Function

Ground truth *probability* and coordinate maps are created as described above. Our loss, inspired by DenseBox, is based on the widely used squared Euclidean loss function

$$E = \frac{1}{2} \sum_{i=1}^{N} (t_i - y_i)^2,$$

where t_i is the target value (ground truth) and y_i is the network output, i iterates over all N output layer neurons.

For simplicity we will be describing the loss for one image in the batch and for a single-scale detection network, that is a network with only one output response map. The channel $c = 0$ is the *probability* response map, the rest are coordinate response maps.

Lets denote t_i^c and y_i^c the target ground truth value and network output value of the pixel i and the channel c in the response map ($c \in \{0, ..., 4\}$ for 2D and $c \in \{0, ..., 7\}$ for 3D bounding box, i is just one dimensional pixel index for a simpler notation). The loss function we compute is

$$E = \frac{1}{2N} \sum_{i=1}^{N} (t_i^0 - y_i^0)^2 + \frac{1}{2N_P(C-1)} \sum_{i=1}^{N} \sum_{c=1}^{C-1} t_i^0 (t_i^c - y_i^c)^2, \tag{1}$$

where $N_P = \sum_{i=1}^{N} [\![t_i^0 \neq 0]\!]$ is the number of positive pixels in the target *probability* response map, $[\![\cdot]\!]$ denotes the Iverson bracket, and C is the number of output map channels (either 5 or 8). Note that we multiply the coordinate part of the loss with t_i^0, which in the case of using Gaussian blob in the *probability* response map decreases the gradient from the pixels, which are further from the object center.

The above loss function is only used for display during training and is shown on all plots in the evaluation section. The problem of the loss function is the biased target response map. The number of positive pixels (samples) $N_P << N_N$, where $N_N = N - N_P$, i.e. the number of negative pixels (see Fig. 3.6b). The gradient from the positive samples would be overweighted by the gradient from the negative samples, therefore we introduced a weight factor $\alpha > 1$ to increase the significance of the positive samples. The resulting loss function is

$$E = \frac{1}{2N} \sum_{i=1}^{N} \left(1 + [\![t_i^0 \neq 0]\!](\alpha - 1)\right) (t_i^0 - y_i^0)^2 + \frac{1}{2N_P(C-1)} \sum_{i=1}^{N} \sum_{c=1}^{C-1} \alpha t_i^0 (t_i^c - y_i^c)^2, \quad (2)$$

where $[\![\cdot]\!]$ denotes the Iverson bracket.

3.4.3 Gradient Computation

Back-propagation [31, 44] is the most frequently used algorithm for learning hidden variables in neural networks. It is a method based on gradient descent, and hence we are required to find the gradient (derivative) of the loss function with respect to each hidden variable of the network. The value of each hidden variable is then updated by the value of the gradient (3). The update is performed based on the outputs of the network on a training set. It is important to note that instead of updating the values after processing the whole dataset, we use a modified version of the learning algorithm called stochastic gradient decent (SGD). This version performs the updates after processing a certain number B (batch) of randomly selected training images.

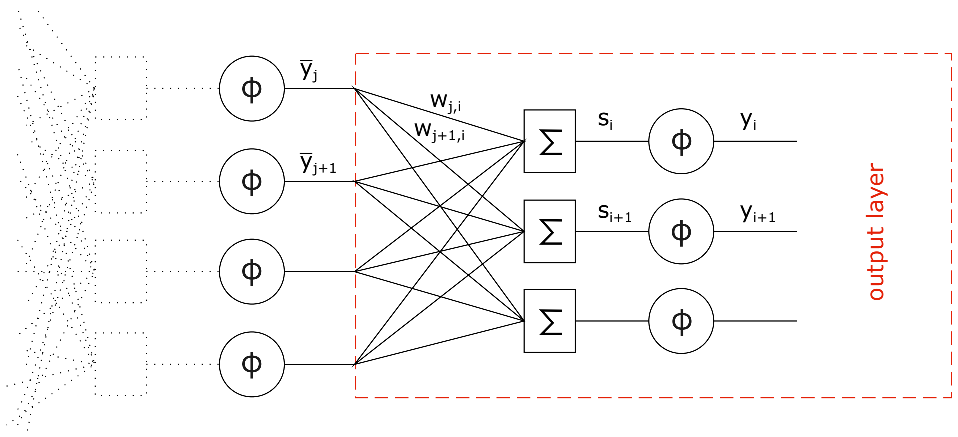


Figure 3.10. Illustration to support the derivation of the weight update $\Delta w_{j,i}$.

We now derive the computation of the weight updates $\Delta w_{j,i}$ for the connections in the output layer (see supporting illustration of the problem in Fig. 3.10). For simplicity, lets consider $B = 1$ for the derivation. We derive the version without biases in the neurons (5) because the computation of updates for them is similar.

Following the notation from Fig. 3.10, the weight update is defined as

$$\Delta w_{j,i} = -\eta \frac{\partial E}{\partial w_{j,i}}, \tag{3}$$

where η is the learning rate. We use chain rule to compute the derivative of the loss function (2) as follows

$$\frac{\partial E}{\partial w_{j,i}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial w_{j,i}}, \tag{4}$$

$$y_i = \phi\left(s_i\right),$$

$$s_i = \sum_k w_{k,i} \bar{y}_k. \tag{5}$$

For the first term of (4) we have to distinguish between the weights on the connections $w_{j,i}^0$ belonging to the *probability* response map y_i^0 and the weights $w_{j,i}^c$ on the connections to the coordinate response maps y_i^c, where $c \in \{1, ..., C-1\}$, which yields two different equations

$$\frac{\partial E}{\partial y_i^0} = -\frac{1}{N}\left(1 + [\![t_i^0 \neq 0]\!](\alpha - 1)\right)(t_i^0 - y_i^0), \tag{6}$$

$$\frac{\partial E}{\partial y_i^c} = -\frac{1}{N_P(C-1)} \alpha t_i^0 (t_i^c - y_i^c). \tag{7}$$

The derivation of the second term of (4) is common for both cases and is easily solvable using chain rule

$$\frac{\partial y_i}{\partial w_{j,i}} = \frac{\partial y_i}{\partial s_i} \frac{\partial s_i}{\partial w_{j,i}} = \phi' \frac{\partial s_i}{\partial w_{j,i}}, \tag{8}$$

$$\frac{\partial s_i}{\partial w_{j,i}} = \frac{\partial\left(\sum_k w_{k,i} \bar{y}_k\right)}{\partial w_{j,i}} = \bar{y}_j, \tag{9}$$

where ϕ' is the derivative of the activation function and it depends on the chosen activation function. If we plug (6), (7), (8), and (9) into (4) and then (3) we get two final expressions for the weight updates of the connections of the neurons in the output layer

$$\Delta w_{j,i}^c = \begin{cases} \eta \frac{1}{N}\left(1 + [\![t_i^0 \neq 0]\!](\alpha - 1)\right)(t_i^0 - y_i^0)\phi' \bar{y}_j & \text{c} = 0; \\ \eta \frac{1}{N_P(C-1)} \alpha t_i^0 (t_i^c - y_i^c)\phi' \bar{y}_j & \text{c} \neq 0. \end{cases} \tag{10}$$

The equation (10) describes the weight update used in the output layer connections. The derivation of the updates of the weights in the hidden layers will not be shown here because they are carried out in the standard way. For details on back-propagation see [31, 44] or a modern on-line book [37].

NOTE We made an important enhancement to the learning procedure. When learning a multi-scale network, the gradient from response maps, which do not contain any object (any positive pixel) is nullified. In Sec. 5.3.9 we show that it improves the performance of the network. However, the reason why is questionable. The motivation behind this change was to reduce the gradient from negative pixels of the response map, and hence increase the influence of the gradient from the positive pixels, which is inevitably happening. On the other hand, it reduces the ability of the network to learn not to detect objects in scales, where they should not be detected. Apparently, this is not happening too much and the advantages overweight the disadvantages.

3.5 Detection Extraction

When an image passes through our detection network, it produces a multi-channel response map (or a set of multi-channel response maps in the case of multi-scale detection). Each pixel in such a response map represents one detected bounding box (2D or 3D) and the *probability* of the bounding box being an object. A sample *probability* response map is shown in Fig. 3.11. In general, one object will be detected by several pixels surrounding the center of its 2D bounding box. We need to find only the one most appropriate pixel and output its response as a detection.

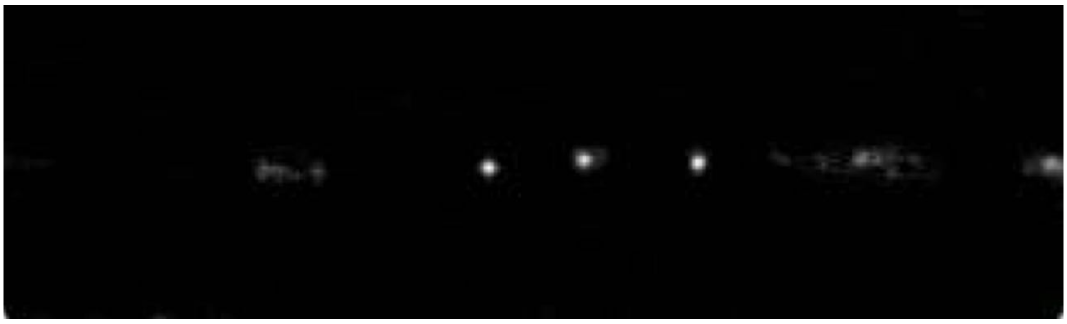


Figure 3.11. Example of the 0^{th} output channel - *probability* response map. Detected bounding box centers are at the positions of the maxima in the map.

A non-maxima suppression (NMS) algorithm needs to be used to reject all insignificant and repeated detections. We use a two-stage solution, where we approach the *probability* response map as a Hough accumulator [3]. First, local maxima on a 3×3 neighborhood are found. Then, a classical non-maxima suppression algorithm discards all detections, which have lower confidence (in our case the *probability* response) than some other detection with intersection over union greater than 0.5.

Using the two-stage approach is crucial in order to achieve reasonable speed of the NMS algorithm. Extracting all detections from the response maps leads to the need to suppress many more detections and therefore it is slower.

3.6 Computing FOV of Convolution

Before we introduce our network designs, it is important to explain how the field of view (FOV) of a fully convolutional network and its layers is computed because the designs are derived from how large FOV is needed in each of the layers.

A fully convolutional network produces an output value for each $s \times s$ pixels of the input image. The down-sampling s of the network is given by the number of pooling layers and the strides of the used convolutions. The FOV of the network is the size of the window on the input image, which influences one pixel in the output layer.

Stacking convolutional layers increases the FOV. A single convolutional layer can have a FOV for example 3×3, however, when two such layers are stacked on top of each other, the FOV expands to 5×5. It is caused by the fact that each pixel in the second layer input already contains information from its 3×3 neighborhood, see Fig. 3.12.

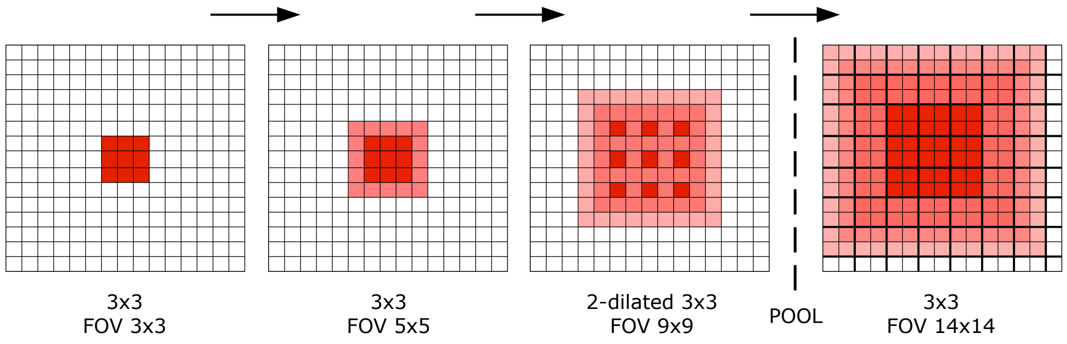


Figure 3.12. Example of how FOV broadens while stacking convolutional and pooling layers.

It is very important to know how to compute the FOV of a convolutional network (and each convolutional layer) because it determines the size of the object that the network can detect, i.e. we want the object to be fully contained within its FOV. We extract output response maps from different levels of the network (Fig. 3.2) because we want to be able to detect objects of different sizes.

Let's denote fov_i the FOV of the convolutional layer i. The down-sampling (scale) of the layer i is represented by s_i. Kernel size is k_i and d_i stands for the dilation factor, where $d = 1$ is regular convolution. The FOV of a layer is given by the recurrent equation

$$fov_i = s_i(d_i(k_i - 1) + 1) - s_{i-1} + fov_{i-1}. \tag{11}$$

Applying equation (11) on our example from Fig. 3.12 gives us

$$\begin{aligned}
&fov_0 = 0, s_0 = 0,\\
&fov_1 = 1(1(3-1)+1) - 0 + 0 = 3,\\
&fov_2 = 1(1(3-1)+1) - 1 + 3 = 5,\\
&fov_3 = 1(2(3-1)+1) - 1 + 5 = 9,\\
&fov_4 = 2(1(3-1)+1) - 1 + 9 = 14,
\end{aligned}$$

which computes the correct FOV of the layers. Pooling layers and stride larger than 1 is represented by increasing scale s (down-sampling factor).

3.7 Used Architectures

When designing our network, we aimed for an object to be detected with a convolutional layer with FOV two times larger than the object $size$. That is, the convolutional layer must have FOV at least $2 \times size$, where $size$ is the **ideal size** of an object.

We mainly used two different networks, which are described in Tab. 3.1 and 3.2. The former network, code name `r2_x4`, detects only single-scale objects and was used in the beginning for making design choices because it was way faster to train. In order to make it detect multi-scale objects we used image pyramid as done in DenseBox. The latter, code name `r2_x2_to_x16_s2` is a new design for multi-scale detection in a single pass. This network was the target of our work and is used in the final evaluations.

For every scale, where the ideal $size$ is noted in Tab. 3.1 and 3.2 a 5 or 8 channel response map is created. The response map is generated by an additional convolutional

Kernel	Stride	Filters	FOV	Ideal *size*	*size* span
3×3	1	64	3×3		
3×3, $d = 2$	1	64	9×9		
MAXPOOL	2				
3×3	1	128	14×14		
3×3, $d = 2$	1	128	26×26		
MAXPOOL	2				
3×3	1	256	36×36		
3×3, $d = 1$	1	256	52×52		
3×3, $d = 3$	1	256	84×84		
3×3, $d = 7$	1	256	148×148	80×80	$[72, 96]$

Table 3.1. The single-scale detection network `r2_x4` with $s = 4$, $cr = 0.25$, $r = 2$. We use this network for initial tests on an image pyramid. Note that the *size* span is smaller than usual because we were using more scales in the image pyramid.

Kernel	Stride	Filters	FOV	Ideal *size*	*size* span
3×3	1	64	3×3		
3×3, $d = 2$	2	64	9×9		
3×3	1	128	14×14		
3×3, $d = 1$	1	128	22×22		
3×3, $d = 3$	1	128	38×38		
3×3, $d = 6$	1	128	66×66	33×33	$[22.5, 55.5]$
MAXPOOL	2				
3×3	1	256	76×76		
3×3, $d = 1$	1	256	92×92		
3×3, $d = 3$	1	256	124×124	66×66	$[44.5, 111]$
MAXPOOL	2				
3×3	1	512	144×144		
3×3, $d = 1$	1	512	176×176		
3×3, $d = 3$	1	512	240×240	133×133	$[89, 222]$
MAXPOOL	2				
3×3	1	512	280×280		
3×3, $d = 1$	1	512	344×344		
3×3, $d = 3$	1	512	472×472	266×266	$[178, 444]$

Table 3.2. The multi-scale detection network `r2_x2_to_x16_s2` with $s = 2, 4, 8, 16$, $cr = 0.3$, $r = 2$. The final design of the detection network. The same setup is used for 2D and 3D bounding box detection. It detects objects of sizes from 22.5 to 444 pixels, which is sufficient for most applications.

layer (not shown in the tables) with 1×1 kernel and 5 or 8 output channels, attached to the last layer in the scale.

One can notice the missing maxpool layer after the first two convolutional layers in the `r2_x2_to_x16_s2` design (Tab. 3.2). The GPU memory required to store the parameters and data of a network is very high. One of the main reasons for this is the computation of convolution, which needs to unfold the input matrix and copy it several times[1]). The size of the unfolded matrix is dependent on the number of outputs of the layer. Using $stride = 2$ reduces the number of outputs of the layer drastically. Also it removes the need for the maxpool layer as the output will be down-sampled by the factor of 2. This seemingly little change saved us about 1/5 of the required GPU memory for storing the network.

The last three convolutional layers in `r2_x2_to_x16_s2` have only 512 filters. Normally, the number of filters is doubled with decreasing the scale by 2. The reason for choosing only 512 filters is the number of parameters of the network. If 1024 filters were used, the network would have about 130MB of parameters. With 512, the network has only 60MB of parameters. Our tests showed that the extra 70MB of parameters are not necessary. We would need much longer training and those parameters would only influence the last detection scale $s = 16$, which is not very abundant in our test images.

Note that the scale x4 ($s = 4$) in Tab. 3.1 has ideal object $size = 80$, whereas in the second network (Tab. 3.2) the ideal object $size = 66$ for the scale x4 ($s = 4$). It is caused by different circle to object size ratio cr used in the two networks. A larger circle ratio (in our case 0.25 vs. 0.3) requires smaller objects when the circle radius r is kept. This choice was made in order for the `r2_x2_to_x16_s2` network to be able to detect smaller objects.

[1]) Details on how convolution is computed in Caffe can be found at `http://caffe.berkeleyvision.org/tutorial/convolution.html`.

Chapter 4
Data and Labels

In this chapter we introduce the requirements on ground truth labels, followed by the description of the used representation, which introduces two new file formats created to store those labels. Four datasets used in this work are described and their comparison is shown. Finally, we show how a 3D bounding box can be reprojected from image coordinates into the world coordinates using the flat ground plane assumption.

4.1 Ground Truth Specification

In this work we use two different ground truth labels - the 2D bounding box and the 3D bounding box. Here we describe the required properties of these two types of annotations.

In both cases, we require all objects (cars), which the annotator can distinguish in the image to be annotated. The annotator should label cars of all sizes, occluded up to 90%, and truncated up to 75%. For the annotators this essentially means to label all objects, which they can recognize from the single image without having any extra knowledge about the scene. See Fig. 4.1 for details.

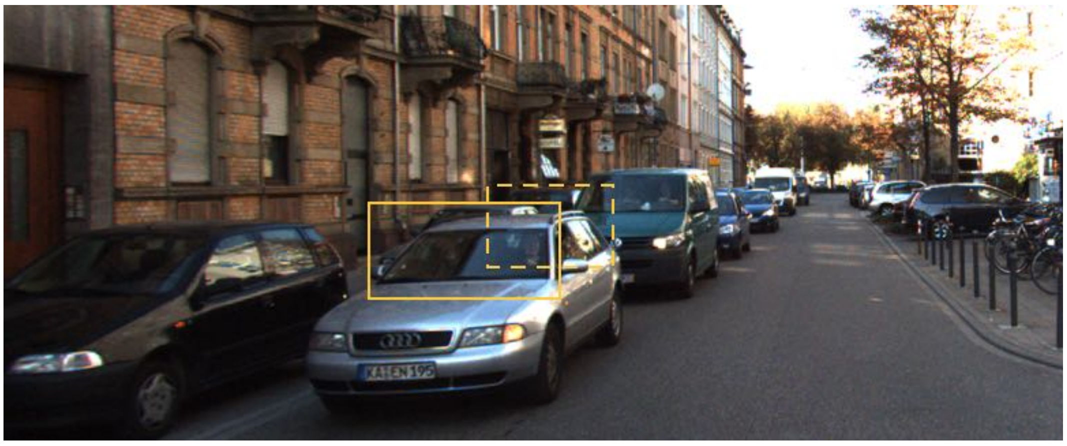
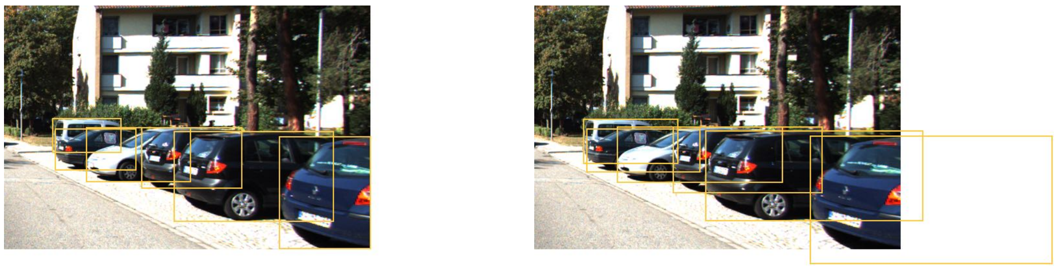


Figure 4.1. An object that according to the methodology should be labeled (solid) vs. object that should not be labeled (dashed). A car from which we would only see a small part of the roof (dashed) would be labeled as `don't know` since the labeler cannot distinguish whether it actually is a car or some obstacle next to the road. This applies even if the labeler has a prior knowledge from a sequence of images that it actually is a roof of a car.

4.1.1 2D Bounding Box

2D bounding boxes are minimum enclosing image-axis-aligned rectangles, which are required to encompass the whole object - in our case the car. However, they are only useful if we can reason from them about the object size or even pose. This leads to the labeler having to make up the hidden (occluded or truncated) parts of the objects such as demonstrated in Fig. 4.2.

Looking at Fig. 4.2, we see that the incorrect bounding boxes only contain the visible parts of the objects (Fig. 4.2a), however, this annotation is useless if we want to, hypothetically, reason about the distance of the object from the camera as, in this case, it has no relation to the actual size of the object. Also, in our algorithm, we create peaks in the response map in the centers of the bounding boxes. If the incorrect bounding boxes were used, the centers would be in different positions with respect to the object for each bounding box, which is not desired.



a) Incorrect annotation.

b) Correct annotation.

Figure 4.2. Examples of incorrect and correct 2D bounding box annotations. See text of Sec. 4.1.1 for explanation.

4.1.2 3D Bounding Box

3D bounding boxes are minimum enclosing rectangular cuboids of objects in the real 3D world. In the case of a 3D bounding box, occlusion and truncation translates to 3D as missing parts (missing parts of a point cloud created from a stereo image pair) of the objects. Again, the labeler needs to think up these missing parts and draw a tight rectangular cuboid encompassing the whole object in the 3D world. On top of that, each 3D bounding box has a distinguished front and rear side, which determine the pose of the object, see Fig. 4.3.

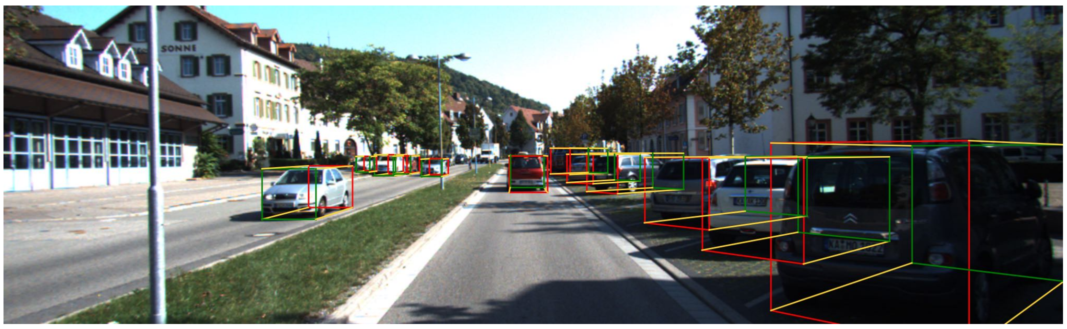


Figure 4.3. Sample image from the KITTI dataset with 3D bounding box annotations. Front sides are displayed in green, rear sides in red.

Very importantly, a 3D bounding box is required to be placed on the ground plane. As we are performing car detection, we can safely assume that cars will be pinned to the ground plane until flying cars will have been developed. We use this assumption also in our representation of the 3D bounding box, which is described further in this chapter.

4.2 Representation

In order to work with the 2D and 3D bounding boxes we had to select the most suitable representation for each and we created our own annotation file formats, which keep the annotations in the specified forms.

4.2.1 2D Bounding Box - BBTXT

The 2D bounding boxes are represented by the coordinates of their top-left (x_{min}, y_{min}) and bottom-right (x_{max}, y_{max}) corners. We created a special file format, which can be used to store ground truth 2D bounding boxes as well as detections. The format of the file is shown below, each line contains one bounding box. In the case of ground truth we simply set `confidence` to 1.

```
filename label confidence xmin ymin xmax ymax
filename label confidence xmin ymin xmax ymax
...
```

4.2.2 3D Bounding Box - BB3TXT

For 3D bounding boxes we store the coordinates of the projections of the bounding box corners into the image. This representation is very convenient as it makes the coordinates independent of the used camera (and its image projection matrix). We can make use of it in our network output as it allows us to train a single network and then use it on images taken by any other camera (for example a different dataset). Importantly, we can use the coordinates of the projected points because we assume that the bounding boxes lie on the ground plane and thus we are able to reconstruct the 3D pose of the bounding box if the equation of the ground plane is known (Sec. 4.4).

If coordinates of all projected corners $(8 + 8$ in total) were stored we would have redundant information. A bounding box in 3D has 9 degrees of freedom (DOF) - 3 for position, 3 rotations, and 3 dimensions. By assuming the objects (cars) are pinned to the ground plane we set **`roll`** and **`pitch`** angles to 0 with respect to the ground plane and the bottom side is a subset of the ground plane, which removes one more degree of freedom. The 3D bounding box therefore has 6 DOF.

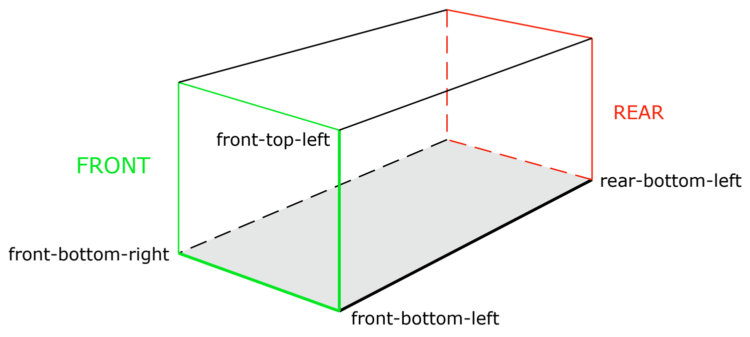


Figure 4.4. 3D bounding box corners explained.

Regarding this analysis, we selected to store the coordinates of the projected rear-bottom-left, front-bottom-left, and front-bottom-right corners and the y-coordinate of the front-top-left corner (Fig. 4.4). This is 7 numbers in total, which is to provide redundancy on the perpendicularity of the front and left sides of the bounding box, which will show to be beneficial in the reconstruction phase (see Sec. 4.4). Also it allows us to draw 3 lines - front-bottom, left-bottom, and front-left into images without doing any kind of reconstruction, which provide sufficient insight into the bounding box's position (see Fig. 4.5). Note that in order to draw the front-left line we assume for simplicity that its projection to the image is vertical.

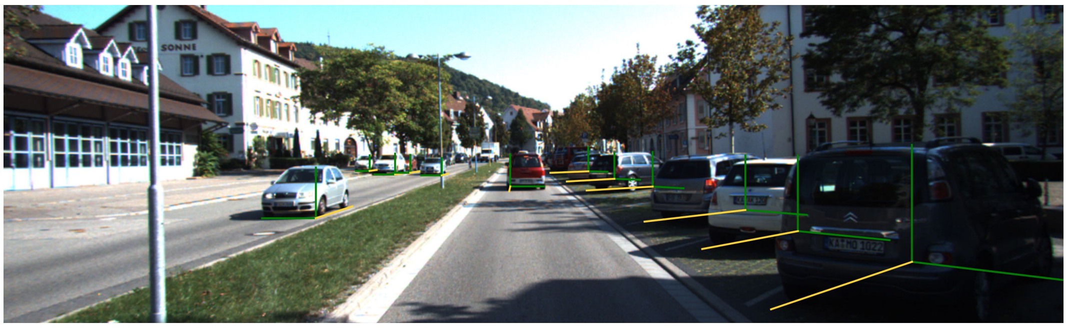


Figure 4.5. Information stored about 3D bounding boxes. Each bounding box is defined by 7 coordinates (3 lines) - front-bottom (green), left-bottom (yellow), front-left (green).

We created a BB3TXT file format to store this information, which is shown below. Conveniently, we also store the 2D bounding box, because it is needed to find the center of the object. Also, we can then load this file into our evaluation script for 2D bounding boxes.

```
filename label confidence xmin ymin xmax ymax fblx fbly fbrx fbry rblx rbly ftly
filename label confidence xmin ymin xmax ymax fblx fbly fbrx fbry rblx rbly ftly
...
```

The `confidence` is, again, in the case of ground truth set to 1. The variable names for the projected bounding box corners in the above code translate in the following way: `fblx` means x-coordinate of the front-bottom-left corner, the other variable names follow this pattern.

4.3 Datasets

Throughout this work we used four different datasets - the UIUC [2], Jura, KITTI [19], and Pascal3D+ [56]. A quick overview of the sizes of the datasets is shown in Tab. 4.1, the UIUC dataset is left out as we only use it to do first experiments but no actual results are presented on it. We now describe label extraction, advantages and disadvantages of each dataset.

4.3.1 UIUC

UIUC is a very old and small dataset consisting of gray-scale images of side view cars. The training set contains crops of single-scale cars, which make it useless for training FCN. However, we needed a simple dataset, where we could initially test our network, therefore, we wrote a script, which takes the crops and randomly places them on randomly cropped background images, adds some noise and exposure shift (Fig. 4.6). This way we created a training set, which was then used for quick benchmarking of the initial enhancements. For testing we used the single-scale version of the test dataset.

	Jura		KITTI		Pascal3D+
Subset	train	test	train	test	all
Number of images	33937	559	7481	7518	6551
Number of cars	74250	2393	30938	-	7465
Width median [px]	39	111	79	-	396
Height median [px]	37	56	47	-	191

Table 4.1. Comparison of the sizes of the used datasets.

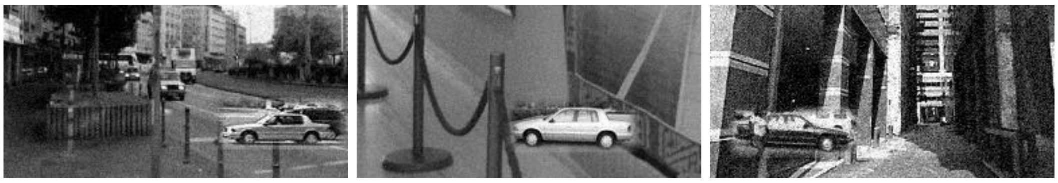


Figure 4.6. Sample images from the artificially created UIUC training dataset.

4.3.2 Jura

The Jura dataset is a collection of images from Prague and Brussels roundabouts and highways. Therefore, it contains cars in a very wide variety of viewpoints, occlusions, and lighting conditions - dark shadow or direct sunlight is abundant in this dataset, see examples in Fig. 4.7. Most of the images are taken with a high-resolution camera, either 1280×1024 or 1208×720. This resolution is sufficient to capture even very distant cars and provide a lot of detail about the close cars.

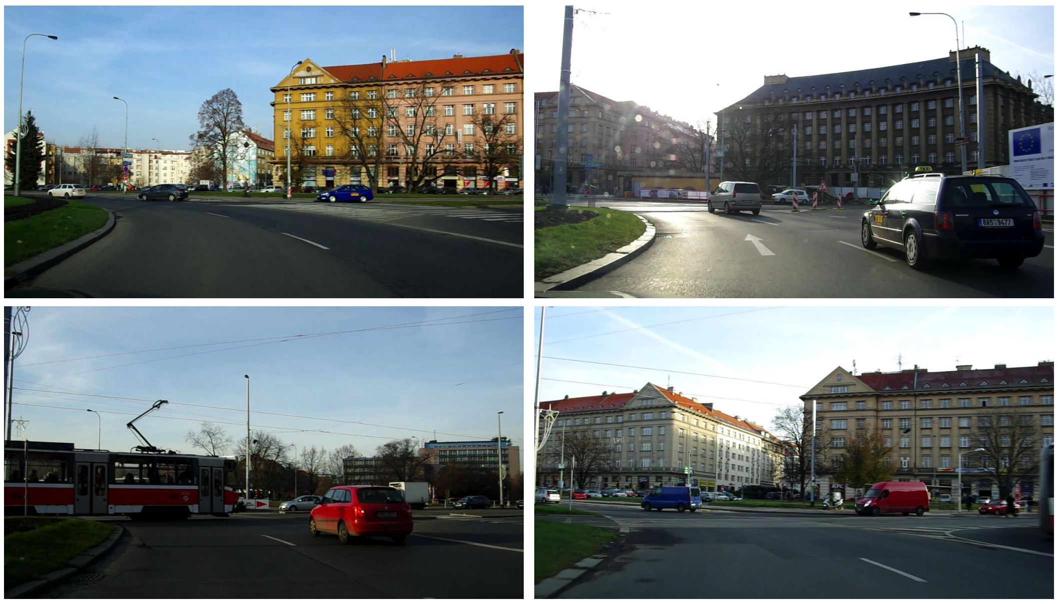


Figure 4.7. Sample images from the Jura test dataset.

The test set consists of 559 images, which we precisely re-labeled with 2D bounding boxes that comply with our labeling requirements (Sec. 4.1). The training set, even though it contains plenty of cars, is not labeled precisely enough and many cars are missing. Often only the fully visible cars are annotated, which is not sufficient for our purpose. Also, the bounding boxes are cropped to the image boundaries, which does not comply with our labeling requirements. We carried out a few tests using the Jura training set, however, the trained detectors were not satisfactory.

Variations Apart from just labeling 2D bounding boxes we also marked the boxes as required or not required and we marked don't-care regions. The bounding boxes labeled as not required are poorly visible or largely occluded cars or cars, where it is hard even for a human labeler to recognize that it is a car. Don't-care regions usually contain parking lots of cars, where it is basically impossible to label all cars correctly. Detections from such regions are then ignored during evaluation.

The Jura test set was used as the only validation and test set throughout our experiments. We use 4 variations of the dataset shown in Fig. 4.8. The first version simply takes all cars without considering the other marks. The second also takes all cars, however removes detections, which are casted in the don't-care regions (i.e. 2D bounding boxes, which have more than 0.75 intersection with a don't-care region). The third only considers required cars. The fourth version considers only required cars and also removes detections from the don't-care regions. The different variants are essentially ordered by difficulty from the hardest to the easiest.

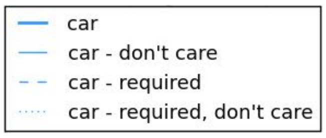


Figure 4.8. The legend of the precision/recall plots in the evaluation. Variations of the Jura dataset.

4.3.3 KITTI

The KITTI dataset was introduced by Karlsruhe Institute of Technology in their 2012 publication [19]. It consist of images extracted from video sequences recorded in the streets of Karlsruhe. It contains very precise annotations from point clouds, which provide 3D bounding boxes needed for training our 3D detection network. The dataset is split into a training (7481 images) and test (7518 images) sets by the authors. Only the labels for the training set are provided as there is an active competition on the test set on the KITTI website¹).

The images have a very high width to height ratio 1240×375, which makes this dataset unique. Large lighting variations and especially extensive occlusions are present (Fig. 4.9). Image projection matrices are available for both the training and test set, thus 3D reconstruction may be carried out.

The labels contain 2D and 3D bounding boxes for several classes of objects, from which we only extract the cars and vans into our training set. As the 2D bounding boxes are cropped to the image boundaries, we had to extract a new set of 2D bounding boxes from the projections of the 3D ones in order for the bounding boxes to comply with our ground truth requirements. KITTI labels provide the 3D position of the center of the bottom side of the bounding box, its rotation around y axis, and width, height, and length - Fig. 4.10.

¹) `http://www.cvlibs.net/datasets/kitti/eval_object.php`

Figure 4.9. Sample images from the KITTI dataset.

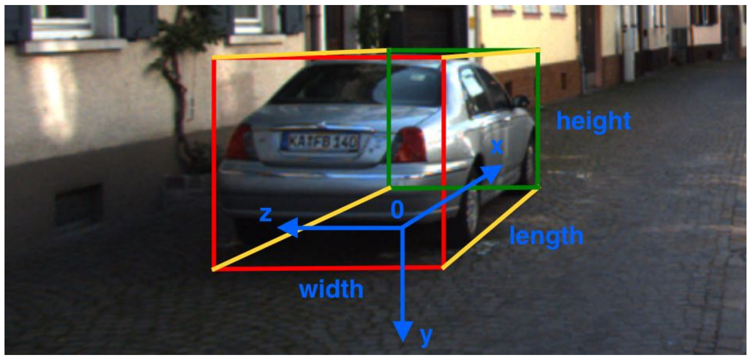


Figure 4.10. Annotated KITTI 3D bounding box. The displayed vertex **0** and axes x, y, z define the reference frame of the car. KITTI labels provide the position of the vertex **0** (center of the bottom bounding box side), the rotation around y axis, and the width, height, and length of the 3D bounding box.

From the labeled dimensions we construct the 3D bounding box in the car reference frame and convert it to the world reference frame by rotation around y and translation. We can conveniently use matrix representation to carry out this operation

$$\begin{bmatrix} \mathbf{X}_{\mathbf{world}} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{\mathbf{y}} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\mathbf{car}} \\ 1 \end{bmatrix},$$

where $\mathbf{R}_{\mathbf{y}}$ represents the rotation matrix around y axis, $\mathbf{t}$ the translation vector from the world origin to the vertex **0**, and $\mathbf{X}_{\mathbf{car}}$ is a 3×8 matrix, where each column represents one corner of the 3D bounding box in the car reference frame. These vertices are then projected to the image with the provided image projection matrix $\mathbf{P}$.

In the case of 2D bounding box extraction we find the maxima and minima of the x and y coordinates in the image, in the case of 3D bounding box extraction we just save the coordinates of the projected rear-bottom-left, front-bottom-left, and front-bottom-right corners and the y-coordinate of the front-top-left corner. It is important to note that we remove bounding boxes with truncation larger than 75%.

The KITTI dataset is the only dataset we used for training the 3D detection networks and the main dataset for training the 2D detection networks.

4.3.4 Pascal3D+

The dataset used in the Pascal VOC [14] was designed to test classification and detection of various objects. The main focus, however, was put on the classification task and, therefore, the images usually contain only one object and the object can appear in very different poses.

Recently, a part of the dataset was extended with 3D annotations [56]. The annotators fit predefined CAD models onto the objects, which provide 3D pose labels of the objects. However, as mentioned before, the objects are in very wide variety of poses (Fig. 4.11) and that is not desired for our 3D detection task since we require the car bounding boxes to have vertical projections of the vertical 3D bounding box edges (horizontal vanishing line of the ground plane).

We use the Pascal3D+ dataset only for training with 2D bounding boxes. The 3D annotations were used to extract 2D bounding boxes, which comply with our ground truth specification (Sec. 4.1). The reason why we could not use the provided 2D bounding boxes is that they were cropped to fit the image dimensions. It is the same problem as in the case of KITTI.

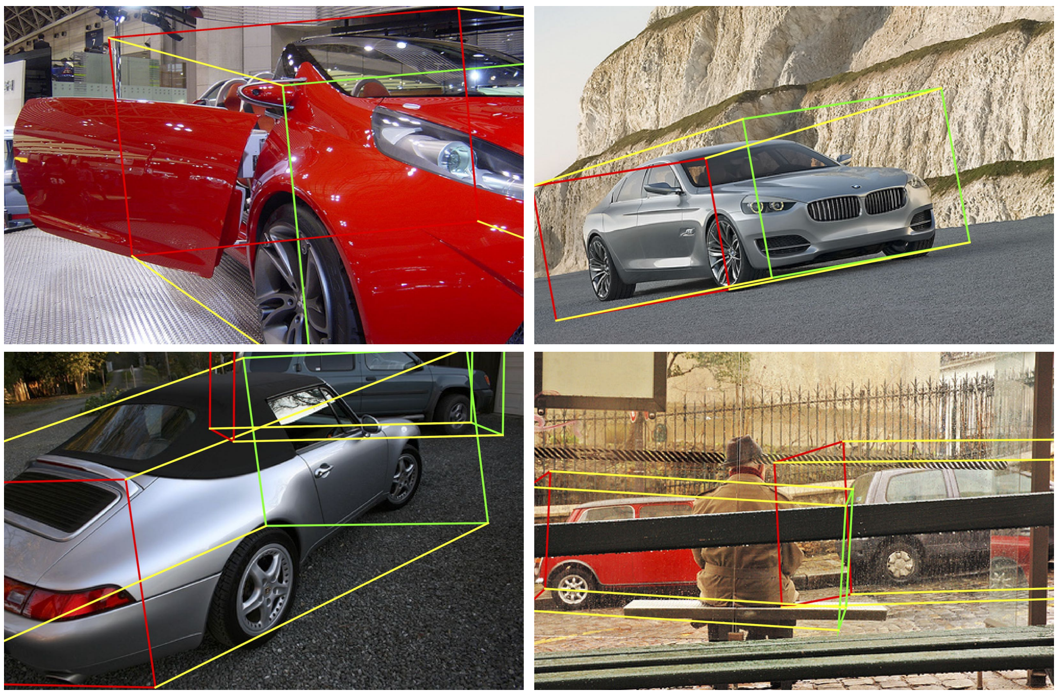


Figure 4.11. Sample images with 3D bounding boxes from the Pascal3D+ dataset. The 3D bounding boxes could not be used because they are too imprecise and we also require the ground plane to have horizontal vanishing line.

4.4 3D Bounding Box Reconstruction

The aim of this work is to detect 3D bounding boxes of cars from a monocular camera. This task is obviously impossible without further knowledge of the scene (triangulation requires stereo camera information in order to perceive depth of the scene). Instead of using stereo imagery we opt for a known flat ground plane assumption. That, together with the requirement of all bounding boxes being pinned to the ground plane provides sufficient amount of information to reconstruct the 3D world positions of the 3D bounding boxes. We now describe the steps needed to carry out this reconstruction.

We assume a known ground plane with equation $ax + by + cz + d = 0$ (Sec. 4.5) with the normal vector $\mathbf{n} = [a, b, c]^T$. Our representation of a 3D bounding box stores 7 image coordinates of projected bounding box corners (rear-bottom-left, front-bottom-left, front-bottom-right, and the y-coordinate of the front-top-left corner as mentioned in Sec. 4.2.2, Fig. 4.4) of which only 6 are needed to carry out the reconstruction considering 3D bounding box is a rectangular cuboid (has right angles among all neighboring sides). We explain why we use this redundant information further in Sec. 4.4.2. On top of that, the 3×4 image projection matrix $\mathbf{P} = \mathbf{KR}[\mathbf{I}| - \mathbf{C}]$, where $\mathbf{I}$ is the identity matrix, needs to be known.

4.4.1 Inverse Projection

Let's have a vertex $\mathbf{X}$ and the camera position $\mathbf{C}$ in the 3D world defined as 3×1 vectors. Projecting the point $\mathbf{X}$ into the camera with center in $\mathbf{C}$, rotation $\mathbf{R}$, and calibration $\mathbf{K}$ gives

$$\lambda\mathbf{x} = \mathbf{KR}(\mathbf{X} - \mathbf{C}) = [\mathbf{KR}|-\mathbf{KRC}]\begin{bmatrix}\mathbf{X}\\1\end{bmatrix},$$

where $\lambda \in \mathbb{R}$. This is because the point $\mathbf{x} = [u, v, 1]^T$ in the image in fact represents the whole ray from $\mathbf{C}$ with the direction $\mathbf{X} - \mathbf{C}$. Considering the inverse projection, we can only reconstruct the ray direction, but we need additional information to determine the exact position of vertex $\mathbf{X}$, resp. λ. We can write

$$\begin{aligned}\lambda\mathbf{x} &= \mathbf{KRX} - \mathbf{KRC},\\ \mathbf{KRC} + \lambda\mathbf{x} &= \mathbf{KRX},\\ \mathbf{C} + \lambda(\mathbf{KR})^{-1}\mathbf{x} &= \mathbf{X},\end{aligned}$$

which tells us that to get the direction of the ray through $\mathbf{X}$ we need to compute the inverse of $\mathbf{KR}$ and the camera position $\mathbf{C}$. This is easily achievable from the image projection matrix $\mathbf{P} = \mathbf{KR}[\mathbf{I}| - \mathbf{C}]$, which we have available. The inverse of $\mathbf{KR}$ is trivial to compute and the camera center can be computed as follows

$$\mathbf{C} = -(\mathbf{KR})^{-1}\mathbf{p_4},$$

where $\mathbf{p_4}$ is the fourth column of $\mathbf{P}$.

4.4.2 Reconstruction of the Bottom Side

We use the ground plane equation $ax + by + cz + d = 0$ ($\mathbf{n} = [a, b, c]^T$) and the inverse projected rays to determine the position of the bottom side of the 3D bounding box in the world. We can reconstruct the rear-bottom-left, front-bottom-left, and front-bottom-right vertices using this information.

For simplicity of the explanation we will just use an artificial vertex $\mathbf{X}$, which can be substituted with the aforementioned vertices. From the inverse projection of image point $\mathbf{x}$ we obtain the ray $\mathbf{l_X} = \mathbf{C} + (\mathbf{KR})^{-1}\mathbf{x}$ passing through $\mathbf{C}$ and $\mathbf{X}$. If $\mathbf{X}$ lies in the ground plane it must hold $\mathbf{n} \cdot \mathbf{X} + d = 0$. Plugging in for $\mathbf{X}$ and solving for λ gives us

$$\begin{aligned}
\mathbf{n} \cdot (\mathbf{C} + \lambda \mathbf{l_X}) + d &= 0, \\
\mathbf{n} \cdot \mathbf{C} + \lambda \mathbf{n} \cdot \mathbf{l_X} + d &= 0, \\
\lambda &= -\frac{\mathbf{n} \cdot \mathbf{C} + d}{\mathbf{n} \cdot \mathbf{l_X}}.
\end{aligned}$$

This way we can reconstruct all three points $\mathbf{X_{fbl}}$, $\mathbf{X_{fbr}}$, and $\mathbf{X_{rbl}}$ on the ground plane and then compute $\mathbf{X_{rbr}} = \mathbf{X_{fbr}} + (\mathbf{X_{rbl}} - \mathbf{X_{fbl}})$. However, since we have the re-projection of three points we will obtain a parallelogram in the ground plane instead of a rectangle (Fig. 4.12 solid). We ignored the fact that a 3D bounding box is a rectangular cuboid and all neighboring sides should be perpendicular.

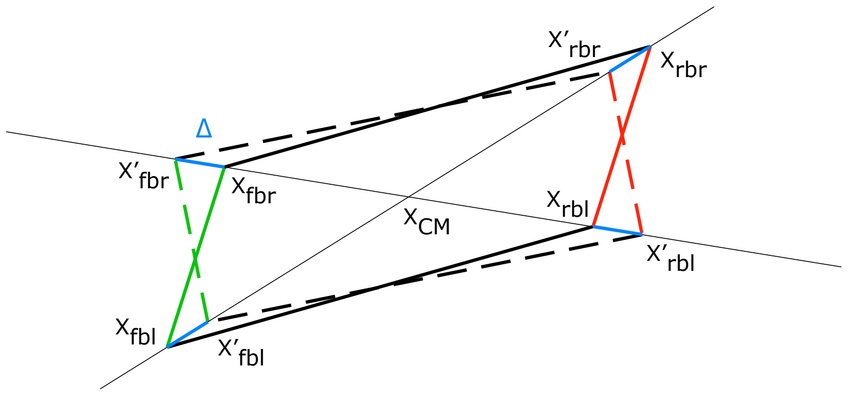


Figure 4.12. Rectification of a parallelogram (solid) to a rectangle (dashed). Projection of a 3D bounding box to the ground plane.

We can make use of the redundant information to improve the accuracy of the 3D bounding box. The precision of the coordinates for each corner differ depending on the pose of the car, therefore, combining the information may reduce the imprecision in the corners' position. If we decided to only predict 6 coordinates in total, we would have large imprecisions in certain poses of the car.

An illustration of how a parallelogram can be rectified into a rectangle is shown in Fig. 4.12. We require that the center of mass $\mathbf{X_{CM}}$ stays the same. A rectangle has diagonals of equal length as opposed to a parallelogram. We can use this information to rectify the parallelogram in a way that we keep the directions of the diagonals and only adjust their lengths. Obviously, this may be done in many ways. We chose to use the mean length of the lengths of the two diagonals. Using the notation in Fig. 4.12, this is how it may be achieved:

$$\begin{aligned}
\mathbf{d_1} &= \mathbf{X_{fbl}} - \mathbf{X_{CM}}, \qquad \mathbf{d_2} = \mathbf{X_{fbr}} - \mathbf{X_{CM}}, \\
\Delta &= \frac{\|\mathbf{d_1}\| - \|\mathbf{d_2}\|}{2}, \\
\mathbf{X'_{fbl}} &= \mathbf{X_{CM}} + \mathbf{d_1}\left(1 - \frac{\Delta}{\|\mathbf{d_1}\|}\right), \\
\mathbf{X'_{fbr}} &= \mathbf{X_{CM}} + \mathbf{d_2}\left(1 + \frac{\Delta}{\|\mathbf{d_2}\|}\right).
\end{aligned}$$

The new corners $\mathbf{X}'_{\mathsf{rbl}}$ and $\mathbf{X}'_{\mathsf{rbr}}$ are obtained in a similar way. Thus, we made use of the redundant information that the representation provides and made the reconstruction more robust.

4.4.3 Reconstruction of the Top Side

Once we have reconstructed the bottom side, in order to reconstruct the top side of the 3D bounding box we only need one piece of information - the height of the bounding box. That is why we have the y-coordinate of the projection of the front-top-left corner $\mathbf{x}_{\mathsf{ftl}}$ in the 3D bounding box description.

Given that the rotation matrix $\mathbf{R}$ of the used cameras is identity (the ground plane has horizontal vanishing line), the x-coordinate of the projection of the front-top-left corner $\mathbf{x}_{ftl}$ is equal to the x-coordinate of the projection of the front-bottom-left corner $\mathbf{x}_{fbl}$. That is, the projection of the front-left line to the image is vertical (see Fig. 4.4).

The front-left line lies in a plane perpendicular to the bottom-left line. We can therefore use the direction of the bottom-left line as the normal vector of the frontal plane $\mathbf{n}_{\mathsf{F}} = [a_F, b_F, c_F]$ and place the front-bottom-left point in the frontal plane to get

$$\begin{aligned} \mathbf{n}_{\mathsf{F}} &= \mathbf{X}_{\mathsf{fbl}} - \mathbf{X}_{\mathsf{rbl}}, \\ d_F &= -(\mathbf{n}_{\mathsf{F}} \cdot \mathbf{X}_{\mathsf{fbl}}), \end{aligned}$$

which determines the frontal plane $a_F x + b_F y + c_F z + d_F = 0$. Finding the intersection of the frontal plane and the ray $\mathbf{l}_{\mathsf{ftl}} = \mathbf{C} + (\mathbf{KR})^{-1}\mathbf{x}_{\mathsf{ftl}}$, in the same way as previously described with the ground plane, gives us the position of the vertex $\mathbf{X}_{\mathsf{ftl}}$ in the 3D world. From this we can determine the height of the bounding box and reconstruct the top side by adding this height to the rectified bottom plane corners.

4.5 Ground Plane Extraction

The aforementioned 3D bounding box representation requires the ground plane equation to be known in order to be able to reconstruct the bounding boxes in the 3D world. Therefore, for each image, we need the equation of the ground plane. This information is unknown in the datasets we use, however we can estimate it for the KITTI dataset. Unfortunately, for the Jura dataset we just have to guess. We assume that the ground plane equation is shared in all KITTI images and estimate it from the training set.

For each 3D bounding box there are 4 vertices, which lie in the ground plane. This gives us 158388 vertices if we accumulate the bounding boxes from the whole training set. We use RANSAC [17] to find the ground plane, which suits those vertices the best.

A pseudo-code of our RANSAC algorithm is shown in Fig. 4.13. Given the number of iterations N and a set of vertices G in the ground plane we can run the RANSAC estimation. A plane is defined by 3 vertices $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ and can be computed as follows:

$$\begin{aligned} \mathbf{l}_1 &= \mathbf{p}_2 - \mathbf{p}_1, \quad \mathbf{l}_2 = \mathbf{p}_3 - \mathbf{p}_1, \\ \mathbf{n} &= \mathbf{l}_1 \times \mathbf{l}_2, \\ d &= -(\mathbf{n} \cdot \mathbf{p}_1), \end{aligned}$$

```
Given: $N$, $G = \{[x_1, y_1, z_1]^T, ..., [x_m, y_m, z_m]^T\}$.
Initialize: $dst_{min} = \infty$.
For $i = 1, ..., N$:
    ▪ Randomly sample 3 vertices from $G$.
    ▪ Extract the coefficients of $ax + by + cz + d = 0$ plane defined by the sampled vertices.
    ▪ Compute the sum of squared distances $dst$ of all vertices in $G$ from the plane.
    ▪ If $dst < dst_{min}$, then $dst_{min} = dst$, store $a$, $b$, $c$, $d$.
Output the best fitting $ax + by + cz + d = 0$ plane coefficients:
    $a, b, c, d.$
```

Figure 4.13. The RANSAC algorithm for ground plane estimation.

where $\mathbf{n} = [a, b, c]^T$ is the normal vector of the plane $ax + by + cz + d = 0$. The sum of squared distances from the estimated plane of the points in G, which we aim to minimize is

$$dst = \sum_{i=1}^{|G|} \left([a, b, c, d][x_i, y_i, z_i, 1]^T\right)^2 .$$

After running the RANSAC algorithm for 10000 iterations we obtained a ground plane in the 3D world with the equation

$$y - 1.49 = 0.$$

Images, which show the ground plane in the context of the points are in Fig. 4.14. Note that the directions of the x and y axes are the same as image axes and the z axis points forward from the camera. Since the world reference frame's units are meters, the result tells us that the ground plane is 1.49m below the camera mount.

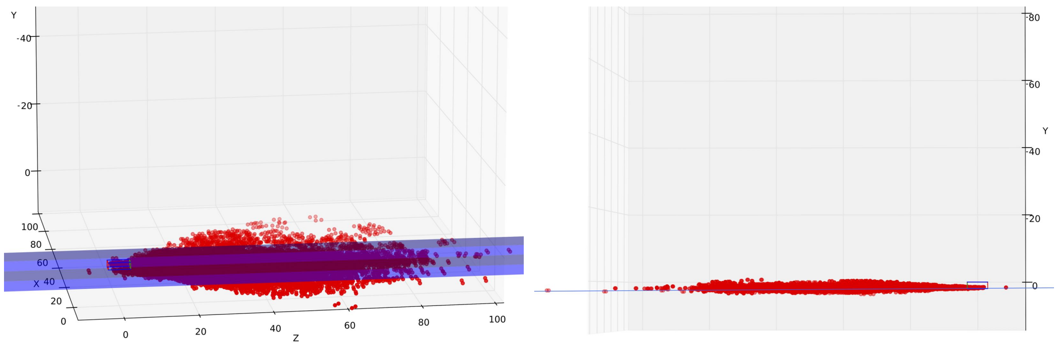


Figure 4.14. The extracted ground plane $y - 1.49 = 0$ from the KITTI dataset (blue). The ground truth points (red), the reference car pose is shown with the blue-red-green 3D bounding box.

PGP Files In order to be able to use this information for 3D bounding box reconstruction, we created a new file format called PGP, which contains the 3×4 image projection matrix $\mathbf{P}$ and the coefficients of the ground plane equation $ax + by + cz + d = 0$ for each image. The file looks as follows:

```
filename1 p00 p01 p02 p03 p10 p11 p12 p13 p20 p21 p22 p23 a b c d
filename2 p00 p01 p02 p03 p10 p11 p12 p13 p20 p21 p22 p23 a b c d
...
```

The PGP file is loaded during test time and serves for extraction of the whole 3D bounding box and the 2D bounding box, which is enclosing all corners of the 3D bounding box (Sec. 4.4).

NOTE The flat ground plane assumption is obviously very coarse. As shown in Fig. 4.14, the points do not always lie on the same plane. It is caused by the fact that sometimes the road goes uphill or downhill or the suspension of the car reacts to unevenness of the road. As a result, the error of the reconstruction, when using such an assumption may be quite large. Employing stereo information would be more suitable for this purpose.

Chapter 5
Evaluation

Designing a neural network requires many choices, whose impact may only be evaluated by performing extensive testing. For this reason, in this chapter we first provide a thorough analysis of the most important design choices we made during our work. The impact of the changes is always evaluated on a 2D bounding box detection network, trained on the KITTI training set and tested on the Jura test set. At the end, we present the best results of our method on 2D and 3D bounding box detection.

5.1 Implementation

Many frameworks for implementing neural networks exist [4, 11, 1, 28], out of which we chose to use the Caffe [28] framework. Caffe provides implementation for forward and backward propagation of all widely used layers, and also allows for implementing our own layer types. The default Caffe layers support both CPU and CUDA capable GPU during runtime, which makes it convenient for benchmarking on both GPU-equipped and other computers.

For training, we implemented our own data reading layers, which accepted the BBTXT and BB3TXT (Sec. 4.2.1 and 4.2.2) file format. These layers first read the whole ground truth label file and then randomly sample bounding boxes from the images in the label file, create batches of image crops and send them for further processing. They are programmed to use multiple threads in order to avoid the data-reading bottleneck.

On top of that, custom loss layers were also implemented. They take care of the creation of the target response maps and the computation of the gradient. As opposed to other frameworks [4, 11, 1], in Caffe, one is required to program also the computation of the gradient. This was one of the reasons why we chose to use Caffe because the user has a larger amount of control over the training phase. It makes it possible to keep the computation of the loss which is displayed to the user separate from the loss used for the weight updates. Thus, we can conveniently compare training runs with different training parameters because the computation of the loss presented to the user stays the same.

The scripts for plotting the results, browsing the detection files and data manipulation were programmed in Python. Apart from other, we programmed scripts for plotting the learning curves, precision/recall curves of the results, browsing the BBTXT and BB3TXT detection files (including reconstruction of 3D bounding boxes), and dataset format translators.

The whole implementation is thoroughly documented and uses Git version control. It is available on GitHub[1]).

[1]) `https://github.com/libornovax/master_thesis_code`

5.2 Measures

We introduce performance measures, which are used throughout the whole evaluation. They are all very well known so we only provide a short description.

Intersection over union (IOU) measures the similarity of two bounding boxes. It is used to compare detected bounding boxes and ground truth bounding boxes. We only use the 2D version, even though it can be easily converted to 3D. It is computed as

$$IOU(b_1, b_2) = \frac{A(b_1 \cap b_2)}{A(b_1 \cup b_2)},$$

where $A(\cdot)$ represents area and b_* are bounding boxes. For our evaluation we require $IOU \geq 0.5$ to accept a detection as a correct one, in the KITTI benchmark they use $IOU \geq 0.7$ for the car detection task.

Precision is the ratio of the detected objects $TP + FP$, which are detected correctly. It tells us how many false positive FP detections the detector produces. It is defined as follows

$$Precision = \frac{TP}{TP + FP}.$$

Recall tells us the ratio of the ground truth objects $TP + FN$, which are detected by the detector. It is defined as

$$Recall = \frac{TP}{TP + FN}.$$

Average precision (AP) is used in the KITTI benchmark. It samples the precision/recall curve on 11 places, interpolates it and computes the arithmetic mean. Therefore, this measure gives insight into how the precision/recall curve changes. For details see [15].

Mean distance error (MDE) introduced by [35] gives an insight into the performance of a 3D bounding box detector. The detections are matched to the ground truth using IOU of 2D bounding boxes. After that, the distance of the detection - ground truth pairs is computed in the 3D world. An MDE plot is created by splitting the pairs into groups by the distance from the camera. This provides an insight on how the error in position changes with respect to the distance of the bounding boxes from the camera.

5.3 Design Choices

This section is a walk through the experiments carried out in order to justify the design choices we made during the network development.

All evaluations presented as precision/recall plots were carried out on the Jura test set labeled with 2D bounding boxes. The networks were trained to detect 2D bounding boxes only since this problem is simpler to train than the 3D bounding boxes (it has less parameters on the output) and, therefore, may prevent us from having misleading results caused by insufficient training. It is also less prone to wrongly setting the training parameters.

NOTE As the presented results are from a continuous process of the development of the algorithm, the plots from different sections cannot be objectively compared. The difference of the algorithms between the sections might not be just the presented enhancements, but also some smaller changes, which are not explained. Nevertheless, the plots proving the impact of the described enhancement are always put side by side.

5.3.1 Dilation vs. Up-sampling

The original DenseBox network does not use dilated convolution. Instead, the network down-samples the image using pooling layers to scales $s > 4$ and then up-samples them back using up-sampling convolutional layers. This seems like an excessive work to do, and also it reduces the resolution of the output image. Whereas, if layers with dilated convolution are used, we do not need the up-sampling layers and the resolution of the output is kept.

We carried out a test on the UIUC dataset, which shows artifacts from the excessive down-sampling and up-sampling in the output *probability* response map, which are shown in Fig. 5.1 (middle) or also Fig. 5.2 (middle). One can notice a repeated structure in the response map, which is a mistake as we require a smooth response. Using dilated convolution removes this problem, see Fig. 5.1 (right). Also, using dilated convolution reduces the number of layers needed to obtain a response map with the same resolution as in the case of up-sampling as the up-sampling layers are simply omitted.

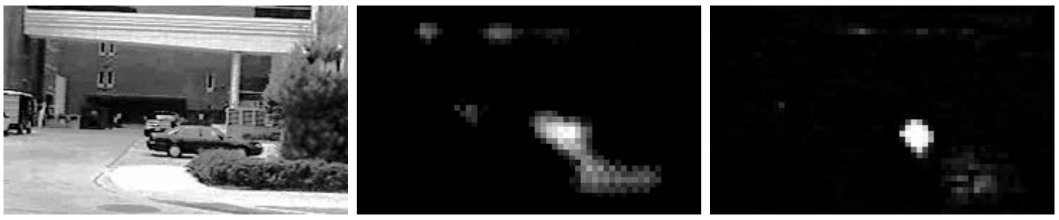


Figure 5.1. Test image (left), response map of a network with up-sampling (middle), network with dilation (right). Note the repeated artifacts in the bottom-right of the middle response map.

5.3.2 Gradient Scaling

The positive and negative pixels in the target response map have a very uneven ratio. The negative pixels cover about 95% of the output response map. This is a problem for training because it makes it hard for the training algorithm to take into account the errors on the positive pixels.

DenseBox addresses this by randomly sampling negative pixels until the ratio of positive and negative pixels is $1 : 1$ and uses only gradient from these randomly sampled pixels for training.

Their strategy, however, slows down training a lot. If positive pixels were 5% of the response map it would mean that we only sample another 5% to represent the negative pixels. This means that 90% of the response map remains unused and we are making a lot of useless computation during training. Instead, we opted for weighting the gradient from the positive pixels by a user-defined coefficient α (see Sec. 3.4.2). Thus, the gradient from all computed response map pixels is used and we do not waste any computation.

Fig. 5.2 shows the difference between random sampling of the negative pixels and weighting the positive pixels. The network was trained for the same number of iterations and with the same setup. Clearly, the *probability* response map of the network trained with weighting the positive pixel gradients is superior as it suppresses background much better.

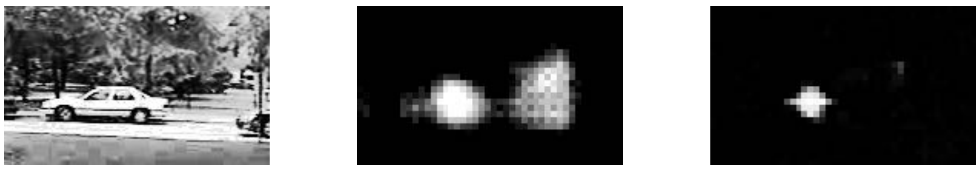


Figure 5.2. Test image (left), response map of a network with random negative pixel selection training (middle), network trained with weighted positive pixel gradient (right).

Setting α: Of course the choice of scaling the positive pixel gradient adds another parameter α to be set for training. We carried out a quick test with setting $\alpha_1 = 1/10$ and $\alpha_2 = 1/50$ and examples of *probability* response maps obtained by these networks can be seen in Fig. 5.3. Clearly, the network trained with $\alpha = 10$ produces more biased responses, whereas the other more assimilate Gaussian blobs. An important take from this is also the fact that larger α produces false positives with a higher *probability* response, therefore, making them harder to be separated by confidence thresholding. We chose to use $\alpha = 1/30$, which provides a good balance between the two properties.

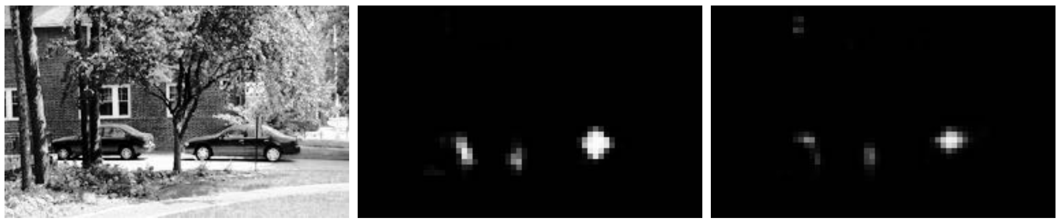


Figure 5.3. Test image (left), response map of a network trained with $\alpha = 1/10$ (middle), network trained with $\alpha = 1/50$ (right).

5.3.3 Learning Rate

The choice of the appropriate learning rate is very important. Using a low learning rate may lead to very slow convergence of the learning, whereas using a learning rate which is too high may make the learning diverge and the network would never learn.

For these tests we used the `r2_x4` network trained on the KITTI dataset to detect 2D bounding boxes in combination with an image pyramid with scales $\{1.0, 0.66, 0.44, 0.29, 0.19\}$. The ideal bounding box *size* of this network is 80px. Therefore, we were unable to detect bounding boxes smaller than approximately 60px. We could not use a scale larger than 1.0 of the input image because the network did not fit into the memory of the GPU.

It is very often suggested to use a large learning rate in the beginning because we are very far from the optimum and we want to make larger steps in order to bring the solution closer. Then, the further the training goes, the lower the learning rate should be since we are already close to the optimum and we want to take smaller steps.

Initially, we employed this learning rate schedule as well, however, as Fig. 5.4 suggests, the results were not very pleasing. We started with $lr = 0.0001$ and we lowered it by the factor of 10 each 5000 iterations. However, lowering the learning rate only lead to essentially stopping the network from learning.

We argue that this learning rate schedule does not make too much sense. If we look at how the magnitude of the gradient evolves during learning, necessarily, the gradient is very large in the beginning as the network gives completely random outputs, however, the further we are in the training process, the lower the gradients are because the network keeps learning the problem. The standard learning rate schedule thus increases the gradient even if it is large already (in the beginning) and then makes it even smaller as it gets smaller (the later stages). What we want is a lower learning rate in the beginning to overcome the early unstable gradient stages, then increase the learning rate once the learning is stable enough. Such a strategy was already successfully used in [40].

Following this observation, first we tried to employ the uniform learning rate $lr = 0.0001$ during the whole learning. Fig. 5.4 (top-middle) shows learning with such setup. It is obvious that the training keeps converging during a much longer period. Also the precision/recall curves suggest improvement on the test set.

Continuing with the intuition, we modified the learning rate schedule as follows. Starting with $lr = 0.0001$ and then raising the learning rate to $lr = 0.0005$ after 500 iterations. The results in Fig. 5.4 confirm that the change improved the performance.

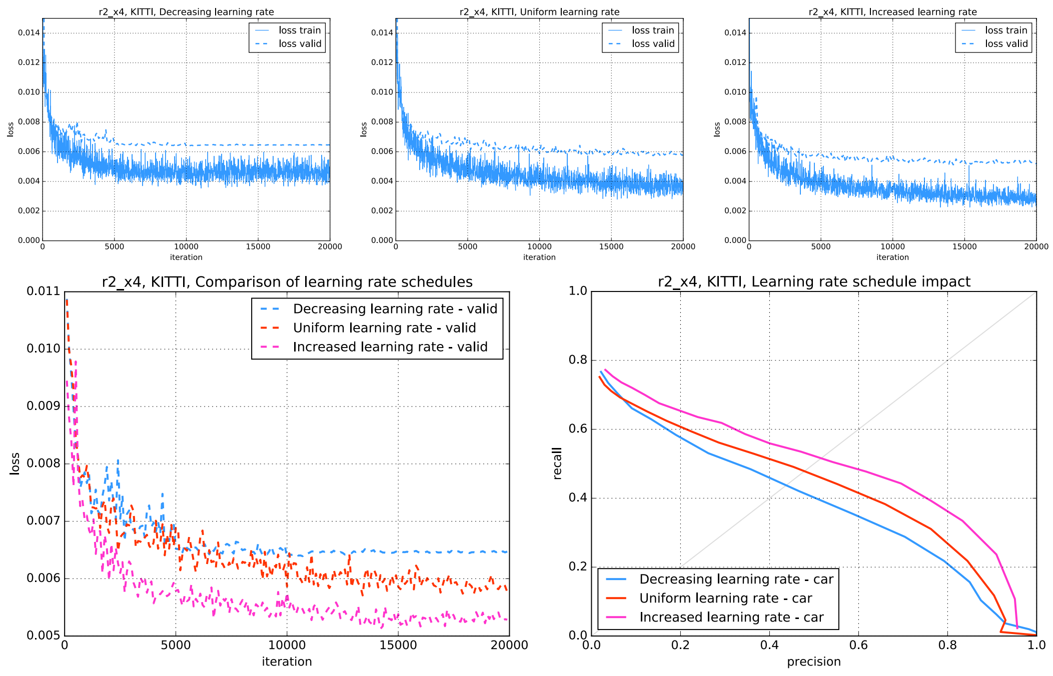


Figure 5.4. Comparison of different learning rate schedules' influence on the network performance. The top row shows the evolution of the training and validation error of a schedule with a decreasing learning rate (left), uniform learning rate (middle), and increased learning rate (right). The bottom row compares the validation loss learning curves (left) and displays corresponding precision/recall curves on the Jura test set (right).

NOTE For training the multi-scale network `r2_x2_to_x16_s2` we had to change this schedule even more as the gradient was unstable for a longer period of time and the used learning rate was on the edge between successful training and divergence. Since we were also using more training iterations (100000) we raised the learning rate once 2.5× at iteration 10000 and again 2.5× at iteration 20000. This resulted in even higher learning rate in the end, but it was used in the stable part of the training, which was beneficial.

NOTE As we are using FCN, the gradient of the loss function with respect to each weight depends on the number of the pixels of the output response map because the convolutional operators introduce weight sharing (this problem is mentioned in Sec. 3.3.1). The aforementioned networks are trained on 256×128 images, which means that the output response map has 64×32 pixels. If we train a network on a larger image, say 256×256 (response map 64×64), we need to scale down the learning rate accordingly. In this example case we would have to use $lr = 0.0001/2 = 0.00005$ to achieve the same training behavior.

5.3.4 Leaky ReLU

A leaky rectified linear unit (ReLU) is a generalization of ReLU, where instead of using $\phi(s_i) = \max(0, s_i)$ the negative part is multiplied by some very small number, for example

$$\phi(s_i) = \begin{cases} s_i & \mathrm{s}_i \geq 0; \\ 0.01 s_i & \mathrm{s}_i < 0. \end{cases}$$

The advantage of this activation function is that it keeps information from the negative numbers as well as the positive ones. On the other hand, its computation is slower.

We compare learning with standard ReLU and leaky ReLU in Fig. 5.5. Since the leaky version of ReLU did not bring any improvement of the results, we chose not to use it in our network.

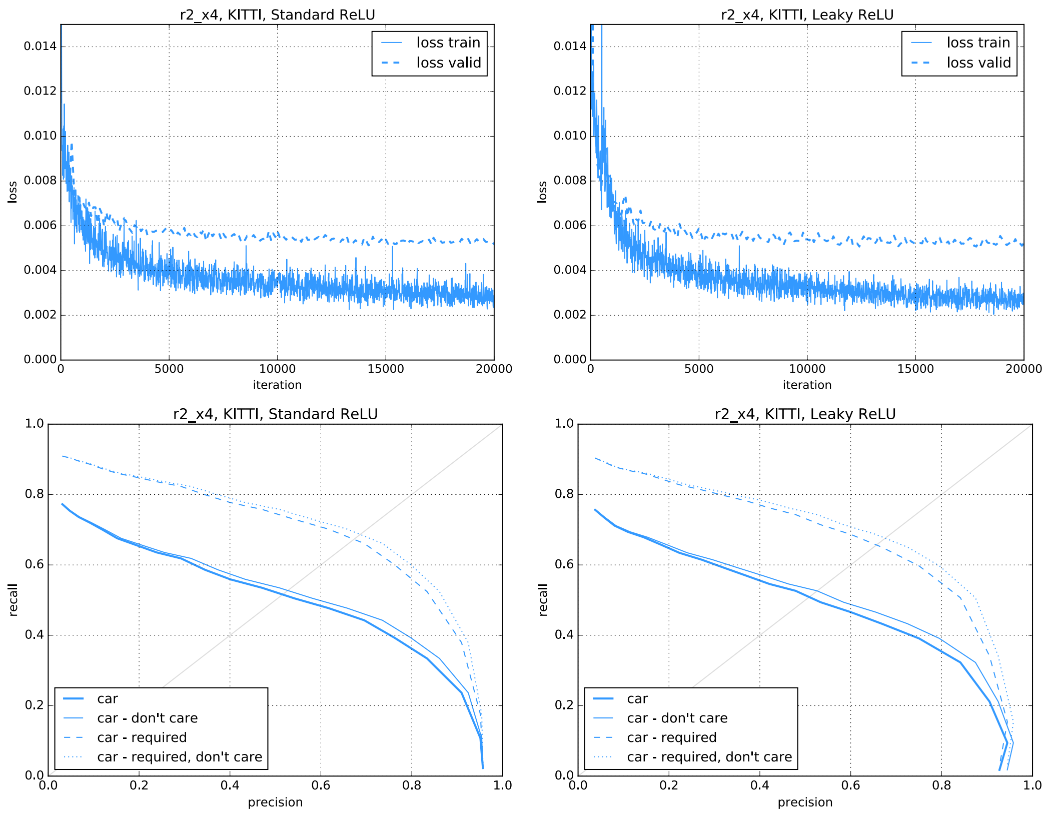


Figure 5.5. A network trained with standard ReLU (left) vs. leaky ReLU (right).

5.3.5 Reference Object *size* Span

In general, the network `r2_x4` can be trained to detect objects of arbitrary sizes. It only depends what we supply as the training data. We made a choice to compute the reference object size from the size of the circle drawn to the response map and a given circle ratio as explained in Sec. 3.4.1. However, not just one size but a whole interval of randomly sampled sizes is used to train the network to provide some generalization.

The `r2_x4` with the chosen $cr = 0.25$ detects objects with the *size* of 80px. In the beginning, we chose the span of sizes with which the network is trained rather small $[76, 88]$, but later we decided to change it to $[72, 96]$, which improves the performance as shown in Fig. 5.6. The precision/recall curve moved slightly up, which signifies an improvement in the detection performance.

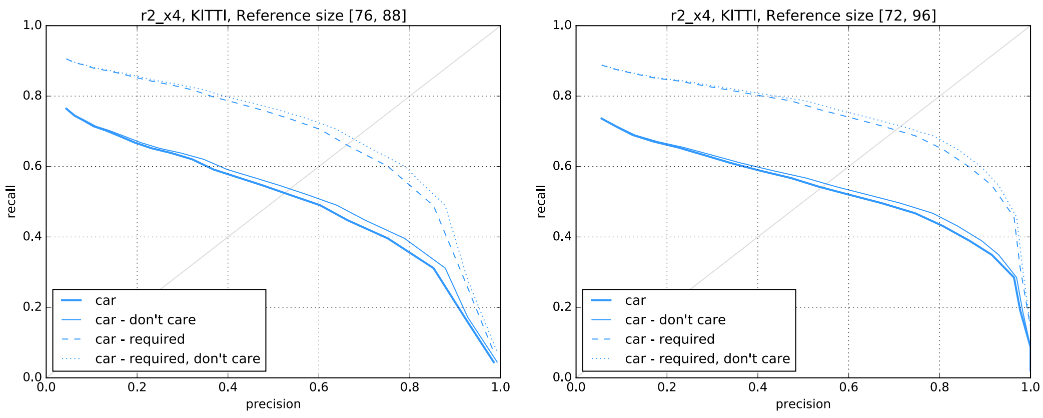


Figure 5.6. Comparison of the networks trained with the reference *size* span $[76, 88]$ (left) and $[72, 96]$ (right).

5.3.6 KITTI Dataset Filtering

The KITTI dataset includes, apart from other, labels for very occluded cars. The problem is that many of those cars are barely visible because only about 5% of the car is visible in the image. Unfortunately, this creates ambiguous situations because our network is not able to spot such subtle differences. On top of that, our Jura test set does not contain cars with very extensive occlusion.

We extracted a filtered dataset, where the cars with occlusion label ≥ 2 (see [19]) are excluded and also very truncated cars ≥ 0.75 are excluded. Fig. 5.7 shows the comparison of the training performed on the dataset with original labels and with the filtered labels. The network trained on the KITTI filtered dataset shows much better performance on the test set (compare the precision/recall curves). We therefore used the filtered version of the dataset for further experiments and for the final evaluation.

5.3.7 Gaussian vs. Binary Response

We chose to train our network to detect Gaussians in the positions of the centers of the object in the *probability* response maps. The used methodology is described in detail in Sec. 3.4.1. We opted for this solution mainly because it converted the sharp binary response into a smoother response, which then resulted in the *probability* response map

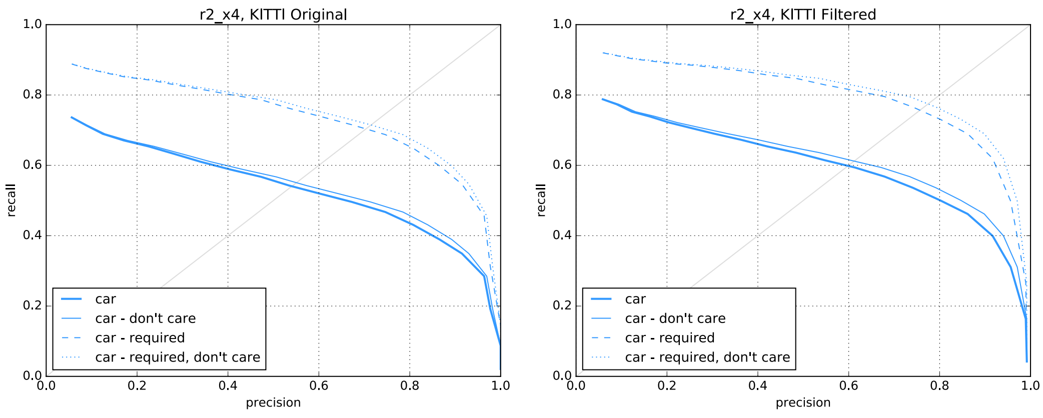


Figure 5.7. The networks trained on the original KITTI labels (left) and the filtered KITTI labels (right).

to act more like confidence. The reason is that in the case of a binary response, the values in between have unclear meaning, whereas if we train the network to predict (regress) those numbers as well, then the values may be used better as a measure of confidence in the prediction.

Fig. 5.8 illustrates this difference. It is important to look at the high-precision part of the precision/recall curves. The difference is very subtle, but the binary-response-trained network does not reach precision 1.0, whereas the version with the Gaussian response does. It is because the responses of the former network are more random around the max value (in our case 1), for that reason it produces more false detections with high confidence, whereas the Gaussian smooths the responses around the max value and gives it a better and less random meaning.

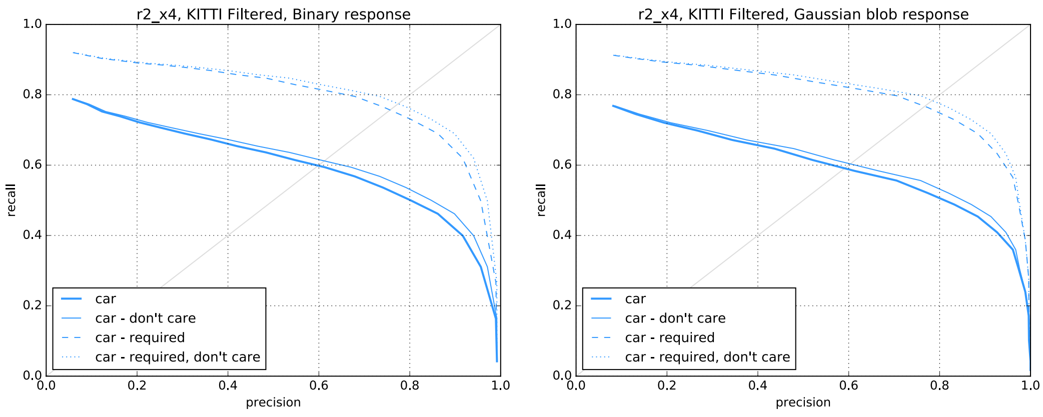


Figure 5.8. The difference in performance between a binary-response-trained network (left) and a Gaussian-response-trained network (right).

5.3.8 Image Pyramid vs. Multi-scale Network

Conversion of the single-scale network to a multi-scale was a crucial step. The problem of the single-scale network is that it is unable to detect smaller objects than for which it is trained because the input image would have to be up-sampled. This is not possible because the network then does not fit into the GPU memory, which was in our case limited to 12GB (already the latest high-end GPU). On top of that, performing multi-scale detection with an image pyramid is slow as the image has to be re-sampled several times and passed several times through the network. And, finally, using a single network for all scales cannot properly exploit all information available in the image because the image is down-sampled several times.

Having in mind the networks from SSD [33] and MS-CNN [5], we designed our own network `r2_x2_to_x16_s2`, which performs multi-scale detection in a single pass (see Sec. 3.7). The design tackles all the aforementioned problems of the single-scale network running on an image pyramid. It is trained to detect objects of sizes from 22.5 to 444 pixels in one pass using 4 response map scales $s = 2, 4, 8, 16$. The comparison of the performance of the `r2_x4` and `r2_x2_to_x16_s2` networks is shown in Fig. 5.9. The multi-scale version clearly outperforms the single-scale version.

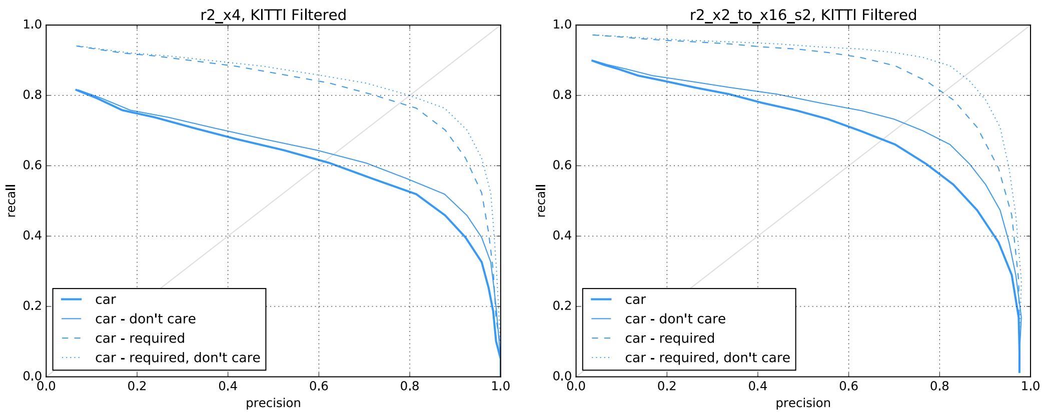


Figure 5.9. A single-scale network with image pyramid (left) and a multi-scale network (right).

The multi-scale version also outperforms the image pyramid in terms of time. We compare the times to detection for an image pyramid detector and a multi-scale network in Tab. 5.1. The multi-scale network takes about 2/3 of the time needed by the single-scale network with image pyramid, which is a significant difference.

Method	Image 1280×1024	Image 1240×375
Single-scale + image pyramid	326 ± 49	134 ± 20
Multi-scale network	222 ± 34	95 ± 3

Table 5.1. Comparison of the times to detection (in milliseconds).

Gradient computation In the transition to a multi-scale network we have to cope once again with the problem that comes from summing up gradients in fully convolutional networks (mentioned in Sec. 3.3.1). The problem is that the response map has different dimensions for each scale, i.e. different number of pixels, and hence a different magnitude of the sum of the gradients coming from the response map. The further in the network we go, the lower the sum of gradients is. Since the scales in our network are powers of 2, we can compensate for this by multiplying the gradient by s^2, where s is the scale (down-sampling factor).

5.3.9 Gradient Nullifying in Multi-scale Networks

When training a multi-scale network, the gradient gets back-propagated from the output response maps of all scales at the same time. However, not all scales always contain objects, i.e. some response maps contain only negative pixels. This may lead to the gradient from the negative pixels overweighting the gradient from the positive pixels. We have already addressed the problem of overweighting the gradient from positive pixels by introducing the coefficient α, but this is a new problem, which arises in the multi-scale setup.

We applied a simple solution. The gradient, which comes from response maps, which do not contain any positive pixels is set to zero. Therefore, the gradient has no influence on learning and is not overweighting the gradient coming from the positive pixels.

The difference in performance of a network trained with nullifying the gradients from the response maps without positive pixels compared to regular training is shown in Fig. 5.10. Note that the test is carried out on a network code name `r2_x2_to_x8`, which we used for quick benchmarking instead of the more complicated `r2_x2_to_x16_s2`. Nevertheless, the fact that the presented change helps is clear.

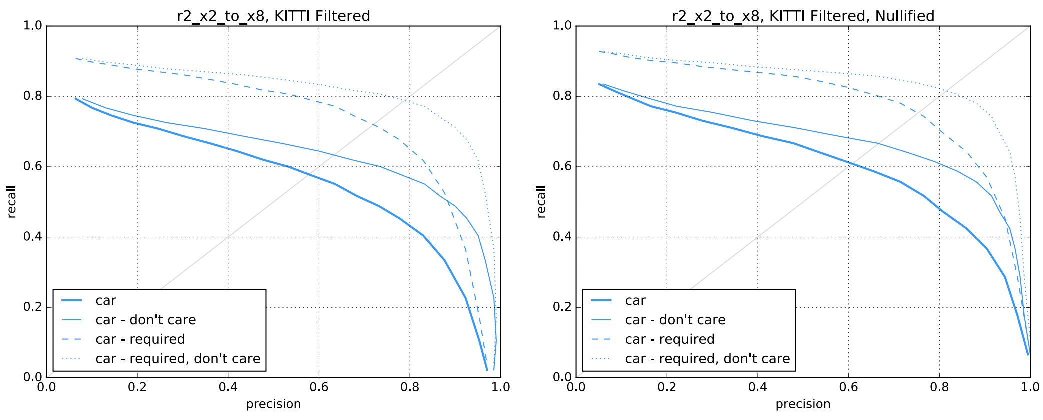


Figure 5.10. A network trained in a regular fashion (left) and a network trained with nullifying the gradient from negative-only response maps (right).

5.3.10 Enhanced Confidence

In our current setup, confidence is taken only from the *probability* channel of the response map. Since the network outputs 5 or 8 values for each bounding box, we were curious if the coordinates of the bounding box could contain additional information with which we could improve the confidence prediction.

To examine the related information contained in the bounding box coordinates' channels we created plots of $Pr(car|BB2D)$, which is the probability of an object being a car given the dimensions of its 2D bounding box $BB2D$. They are shown in Fig. 5.11. Note that the plots are specific for a given network, therefore they need to be generated for each network separately. They were created by evaluating a network on the KITTI training set (on which the network was trained) and taking the bounding boxes, which corresponded to the ground truth car positions in the response maps as positive (car) and the rest as negative (background). We show both $Pr(car|BB2D)$ and $Pr(\neg car|BB2D)$ because the ratios that are 0 in both plots were never predicted by

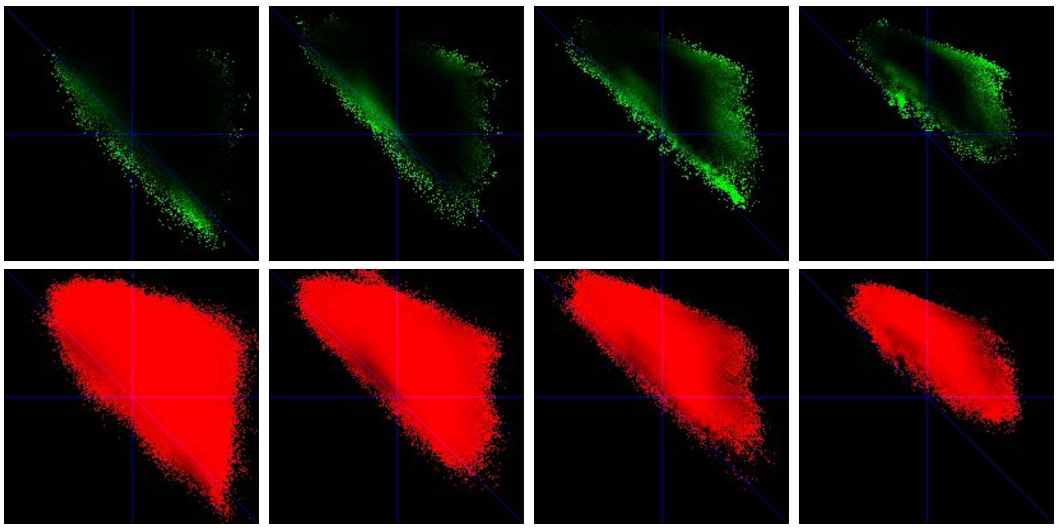


Figure 5.11. $Pr(car|BB2D)$ (top row), $Pr(\neg car|BB2D)$ (bottom row) for different response map scales xs, where 2D bounding box $BB2D$ is defined by its width (x-axis) and height (y-axis). The dimensions are computed in the normalized network response space (see the *Coordinate response maps* part of Sec. 3.4.1), therefore, the ideal object size corresponds to coordinates $(1,1)$. The axes of the plots are $[0,2] \times [0,2]$ with reversed y-axis, i.e. the diagonal blue line corresponds square bounding boxes.

the network (mostly the bottom-left part, which corresponds to tall narrow bounding boxes).

The plots in Fig. 5.11 show that little information about $Pr(car|BB2D)$ which would be useful is stored in the coordinate responses. In the middle area of the top-right section (short wide bounding boxes) the $\neg car$ dominates, but the reason is that there are much more $\neg car$ responses than car responses. Essentially, the $\neg car$ responses are all over the place. The car responses dominate only on the edges of the *trapezoid*. This is because the network is not taught to predict bounding box coordinates for the $\neg car$ responses, and thus they predict dimensions that are most common. It means that they will never predict something completely wrong so the confidence in a prediction cannot be degraded or upgraded by this information. The probability of car for such bounding box shapes is in deed high, but the network would not predict $\neg car$ on them anyway.

We carried out a test of the enhanced confidence defined as $c_0 + Pr(car|BB2D)$, where c_0 is the *probability* channel. The $Pr(car|BB2D)$ was discretized into 50×50 bins spanning $[0,2] \times [0,2]$ space of the width and height response. The result of the test in Fig. 5.12 justifies our observation. The enhanced confidence was not a helpful step to make and we will not use it in our network.

5.3.11 Training Set Choice

It is widely known that having more training data is very helpful while training DNNs. So far we have been using only the KITTI training set to train our 2D bounding box detection network, but what if we used also other datasets (Jura and Pascal3D+) or even combined them? To explore these options we first carried out trainings on the three datasets separately and only then combined them into one.

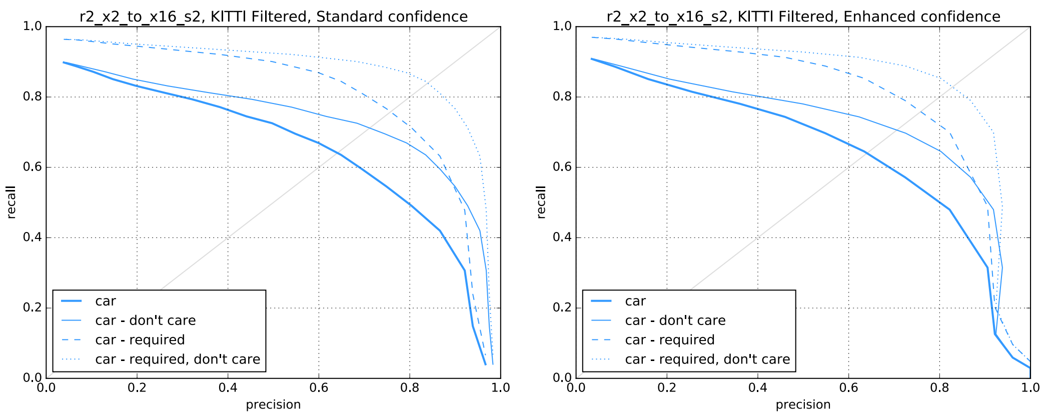


Figure 5.12. The same network evaluated with the standard confidence from the *probability* channel (left) and the enhanced confidence (right).

The sizes of the datasets (Tab. 4.1) suggest that the Jura dataset should have an advantage as it has more than double the number of cars than the KITTI dataset. However, the results in Fig. 5.13 suggest that it is not so. We blame this mostly on the fact that not all cars are labeled in the Jura and Pascal3D+ training datasets, thus resulting in the network being forced to learn contradictory information. We also made an attempt to combine all the datasets together, which resulted in good performance (Fig. 5.13 right), however, not superior to using the KITTI dataset only.

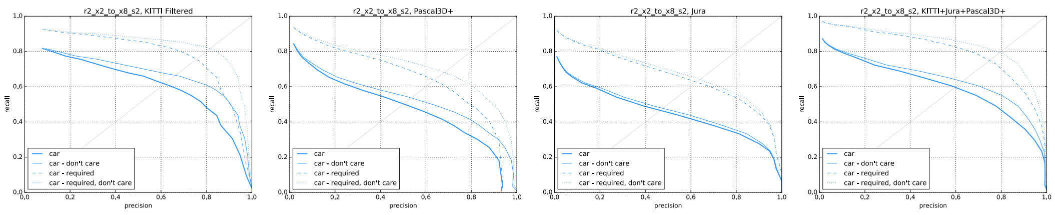


Figure 5.13. The same network trained on 4 different datasets. Left to right: KITTI, Pascal3D+, Jura, and the combination of all.

Following these results, we chose not to use other data than the KITTI training set for learning our networks.

5.4 Car Detection Results

In the following sections, we present the best results we achieved with our network. First, we describe the parameters and the setup used for training the networks, then we compare our network to other detectors on the Jura and KITTI datasets.

5.4.1 2D Bounding Box Detection

The `r2_x2_to_x16_s2` network for 2D bounding box detection was trained on the KITTI Filtered dataset. We used batch size 32. Input image dimensions were 512×512, therefore we had to scale down the learning rate to 0.00000078125, which was then raised $2.5\times$ after 10000 iterations and again after 20000 iterations. We used stochastic gradient descent (SGD) optimization with momentum 0.9 and weight decay 0.0005.

Since the whole batch 32 did not fit into the GPU memory, we used the batch splitting implemented in Caffe into the chunks of 8 images. The network was trained on the object reference *size* from 23 to 440 pixels.

Each training image is randomly augmented, i.e. horizontally flipped with the probability 0.5 and perturbed to make the network learn as robust representation as possible. We perform hue shifts for each color channel in the RGB space separately, exposure changes to account for various lighting conditions, and add white noise to each color channel to simulate sensor noise. The extent of the augmentations is noted in Tab. 5.2.

Augmentation	Min	Max
Hue shift	-30	30
Exposure change	-40	50
White noise σ	0	25

Table 5.2. The image augmentation parameters. The hue shift and exposure change are sampled from a uniform distribution, whereas the white noise is generated from a normal distribution.

The training ran for 100000 iterations, which took about 10 days on a GeForce GTX TITAN X. A snapshot was taken each 10000 iterations and the network tested on the Jura test set. We selected the best result shown in Fig. 5.14 (sample images in Fig. 5.15) based on precision/recall curves, which corresponded to the iteration 40000. Further training seemed to overfit the network to the training data.

It is interesting to note the increasing amount of sudden high peaks in the training error in Fig. 5.14 (left) as the network is getting better. We attribute these peak changes to the outliers and hard positives in the training set. Perhaps, lowering the learning rate in the later stages of the training would reduce these peaks since in our current setup we keep the learning rate constant starting iteration 20000 till the end.

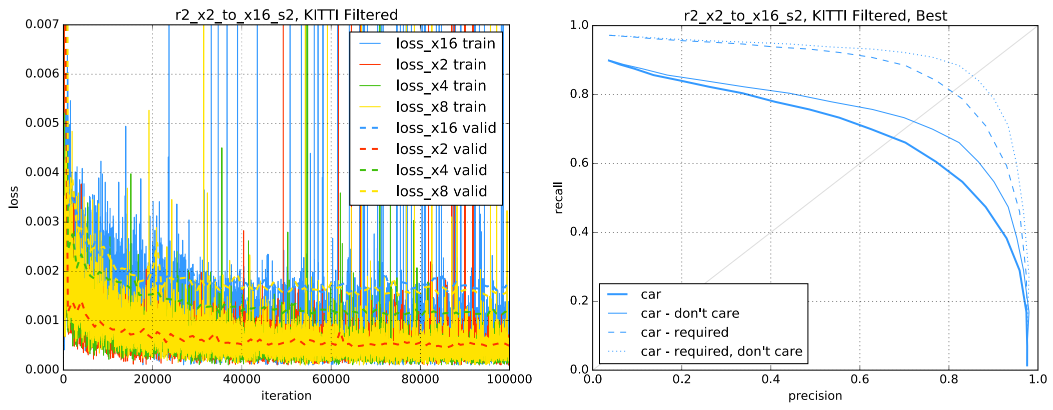


Figure 5.14. The best `r2_x2_to_x16_s2` network for 2D bounding box detection trained on the KITTI Filtered dataset (left) and evaluated on the Jura test set (right).

Comparison We were able to obtain results on the Jura test set for three other methods. Our benchmark has been the WaldBoost [49] detector combined with a CNN refinement on the output. It was trained on the Jura training set specifically. The two other methods are YOLO9000 [41] and SSD [33]. We took their weights trained on the data

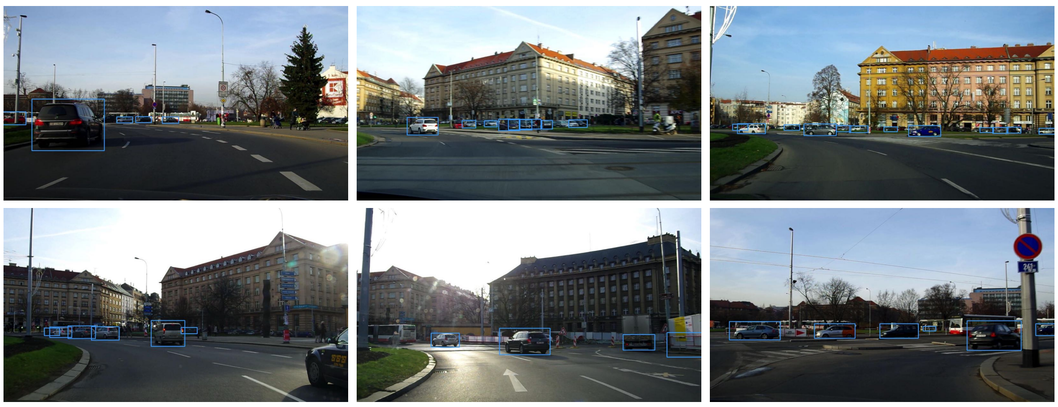


Figure 5.15. Sample images from the Jura test set with the 2D bounding boxes from our best `r2_x2_to_x16_s2` network. Only detections with confidence ≥ 0.7.

from the Pascal VOC challenge. Fig. 5.16 shows the comparison. Our method gives superior results to all the others, however, the comparison is not very fair since each method was trained on a different dataset. Nevertheless, the WaldBoost detector should be fitted more specifically to the Jura dataset, but its performance is inferior to our FCN.

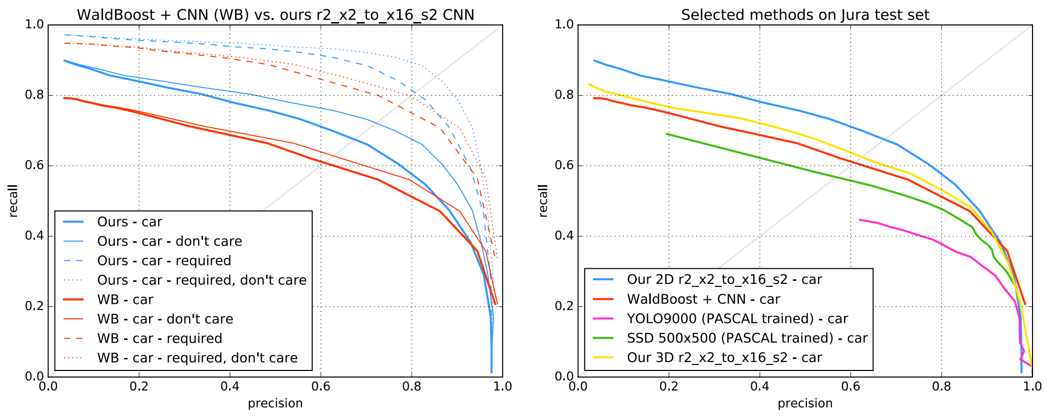


Figure 5.16. Comparison of our `r2_x2_to_x16_s2` network to WaldBoost, YOLO9000, and SSD. The left plot shows a thorough comparison of our method to WaldBoost, which is our closest competitor on the Jura test set. Note the result of our 3D network in the right plot (explained in Sec. 5.4.2).

KITTI Scoreboard We compared our network to the state-of-the-art detectors on the KITTI test set. The only way to do this is to submit results to the KITTI on-line scoreboard[1]) through a web interface. The labels of the KITTI test set are not known to the public, making it a fair comparison of the algorithms. The evaluation measure is average precision (AP), described in [15].

Since the KITTI test set has similar scenes as the training set, we decided to extract a little different training set then we were using for evaluation on the Jura test set. We needed to also include very occluded cars because, for example, situations where cars appear parked along the road are abundant in the test set. Therefore, a dataset with all

[1]) `http://www.cvlibs.net/datasets/kitti/eval_object.php`

occlusion labels (only cars with truncation ≥ 0.75 were excluded) was used for training the network used for submission to the scoreboard. Otherwise, all training parameters were exactly the same.

Again, the network after 40000 iterations was selected for the test. Tab. 5.3 shows our position in the KITTI scoreboard among other methods and Fig. 5.17 sample images with detections. Unfortunately, our method did not appear among the best scoring ones but around the middle of the scoreboard (52/96). There are, however, positive observations we can make. Our method is way faster than all published methods, which appear in the scoreboard in front of us. We beat the original DenseBox on the easy variant of the dataset. Also, given that our method is in the early stages of its development, there is still a lot of room for improvement. We discuss possible enhancements in Chap. 6.

#	Method	Moderate	Easy	Hard	Time [s]
6	RRC [42]	90.22%	90.61%	87.44%	3.6
10	Deep MANTA [6]	90.03%	97.25%	80.62%	0.7
22	SubCNN [55]	88.86%	90.75%	79.24%	2
24	MS-CNN [5]	88.83%	90.46%	74.76%	0.4
32	Mono3D [9]	87.86%	90.27%	78.09%	4.2
39	DenseBox [27]	85.07%	82.33%	76.27%	> 1
45	Faster R-CNN [43]	79.11%	87.90%	70.19%	2
52	Ours	75.76%	85.60%	66.99%	0.1
63	YOLO9000 [41]	69.01%	86.40%	59.57%	0.03

Table 5.3. 2D bounding box car detection KITTI Scoreboard as of May 2017, selection of published methods only. RRC [42] and Deep MANTA [6] are new methods published in CVPR 2017.

5.4.2 3D Bounding Box Detection

Exactly the same training procedure as in the 2D bounding box case was used to train the `r2_x2_to_x16_s2` network for 3D bounding box detection. The network was also trained to detect objects of the *size* from 23 to 440 pixels. The only difference is that the 8-channel output response map was used.

The SGD training also ran for 100000 iterations (Fig. 5.18 left), from which we chose the best network based on its 2D bounding box detection results on the Jura test set. The best precision/recall curves shown in Fig. 5.18 (right) were extracted on the network corresponding to the iteration 80000. This suggests that the 3D detection network needed double the training time than the 2D detection network to learn the problem. The precision/recall curve on the whole Jura test set is shown in the 2D bounding box detection comparison plot in Fig. 5.16 (right). See also sample images with detected 3D bounding boxes on the Jura test set (Fig. 5.19) and the KITTI test set (Fig. 5.20).

2D Bounding Box Extraction In order to plot the precision/recall curves of the 2D bounding box performance of a 3D detection network we first had to extract the 2D bounding boxes. To achieve this, we reconstructed each bounding box in the 3D world (Sec. 4.4), then projected it back to the image and, finally, found the extremes of the corners' image coordinates. The extremes determine the enclosing 2D bounding box.

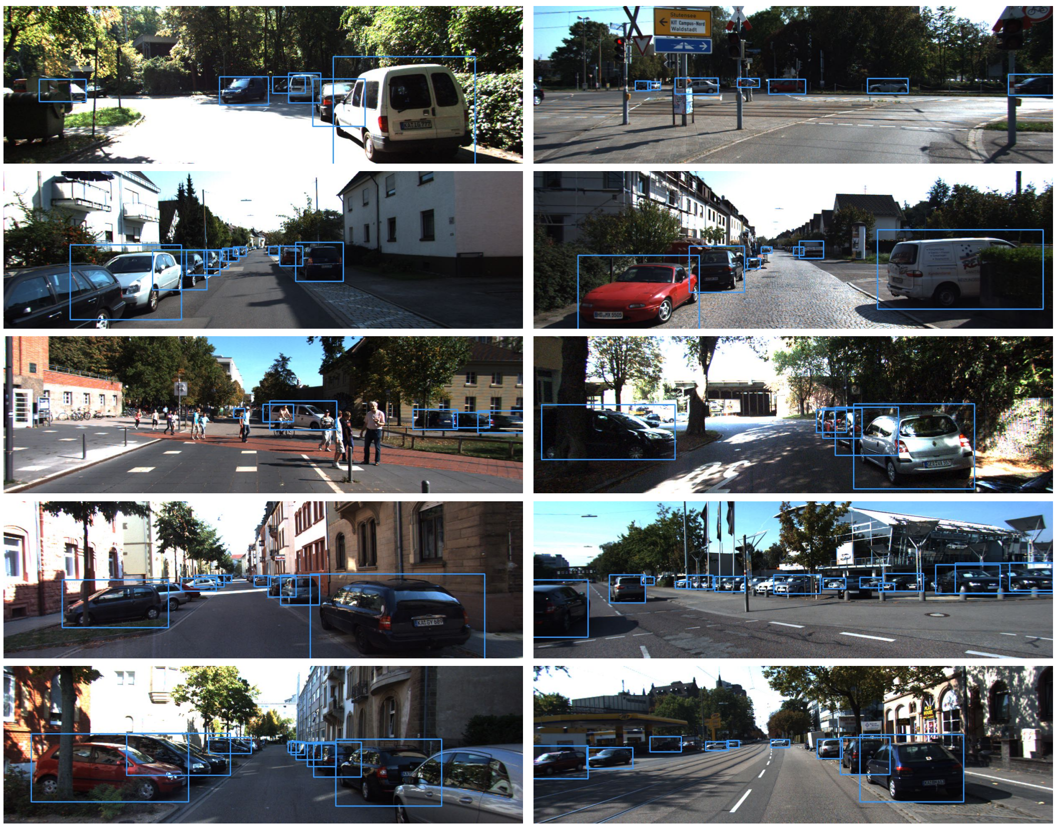
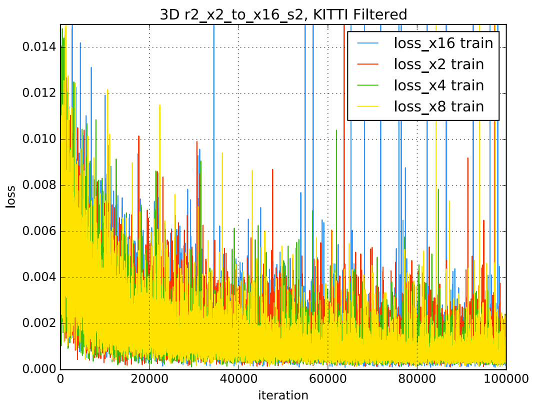


Figure 5.17. Sample images from the KITTI test set with the 2D bounding boxes from our best `r2_x2_to_x16_s2` network. Only detections with confidence ≥ 0.6.



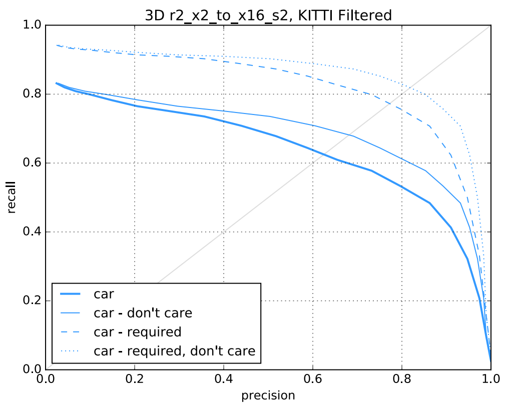


Figure 5.18. The training (left) and the performance of the 3D `r2_x2_to_x16_s2` detection network on the 2D bounding box Jura test set (right). No validation loss is shown among the learning curves (left) because we did not use any validation dataset for 3D bounding box training. Also, see text on how the 2D bounding boxes were extracted from the 3D detections to plot the precision/recall curves (right).

Obviously, this procedure is dependent on the ground plane equation and thus may lead to imprecisions caused by the incorrect positioning of the ground plane. Nevertheless, as Fig. 5.18 (right) shows, the 2D detection performance is still reasonable.

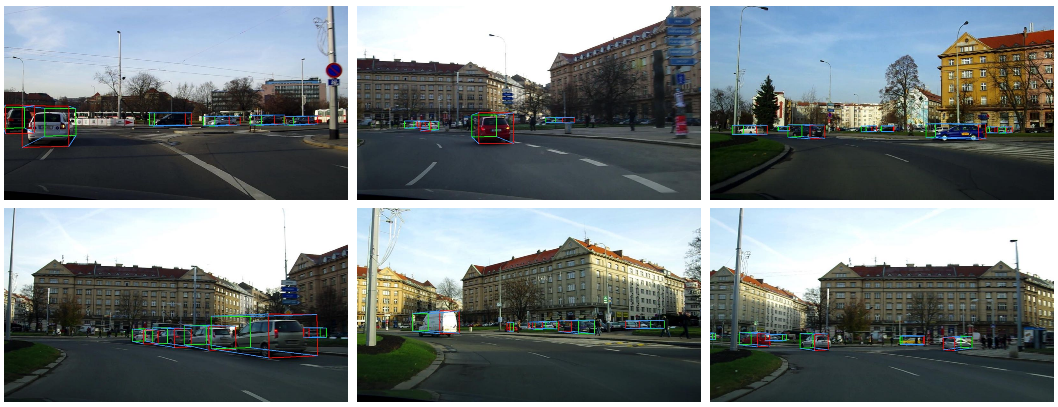


Figure 5.19. Sample images from the Jura test set with the 3D bounding boxes from our best `r2_x2_to_x16_s2` network. Only detections with confidence ≥ 0.7.

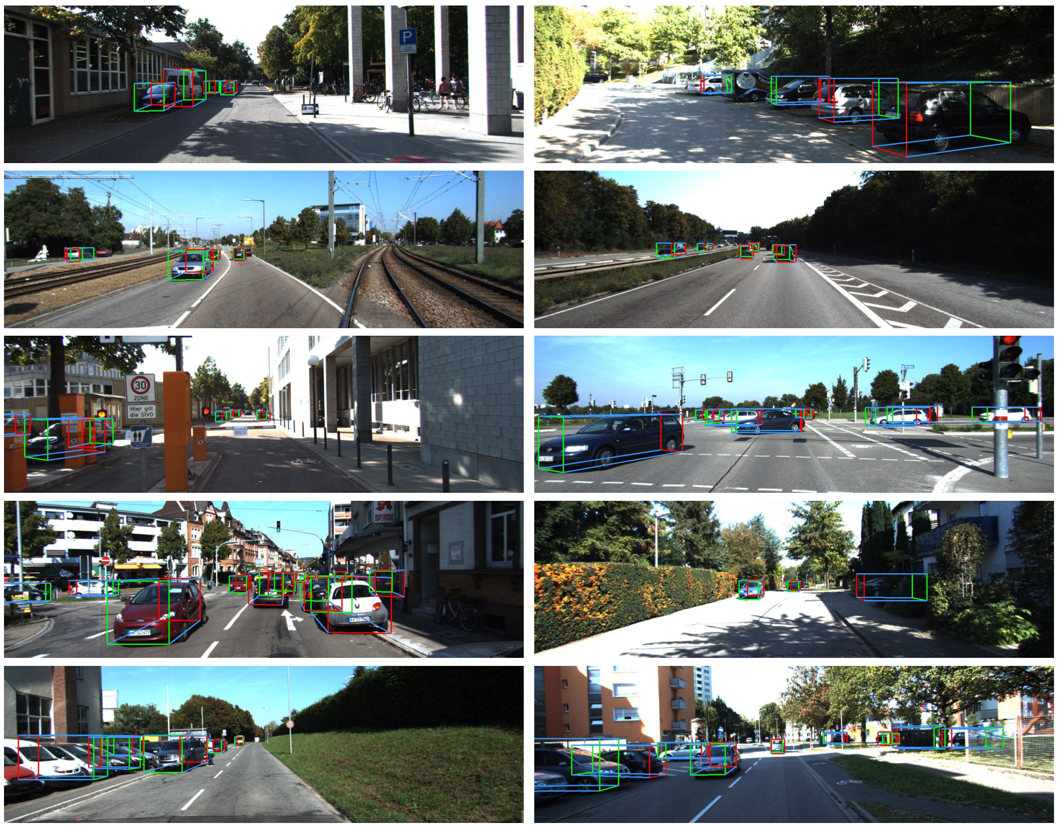


Figure 5.20. Sample images from the KITTI test set with the 3D bounding boxes from our best `r2_x2_to_x16_s2` network. Only detections with confidence ≥ 0.6. See additional images in Appendix A.

3D Detection Evaluation As the original KITTI evaluation does not contain a 3D detection benchmark (except for orientation similarity) we chose to compare the 3D detection performance on the mean distance error (MDE) introduced by [35]. This measure tells us how far are our detected bounding boxes from the ground truth for different distances from the camera.

Because the KITTI test set labels are not available to the public, to carry out this evaluation we split the training set to the two disjoint sets introduced in [10][1]). Roughly, a half of the images is put in a training set and the other half into a validation set. We denote these splits as *KITTI XChen Train* and *KITTI XChen Val*. A network was trained on the reduced training set and then evaluated on the validation set.

The results in Fig. 5.21 show the detection performance of the network on the 2D Jura test set. We see that the results are worse than in Fig. 5.18 (right) as the network is trained on less training data. More interesting, however, is the plot on the right showing mean distance error of the detected bounding boxes to the ground truth. Our detector has a very low mean value of the errors, but quite a large standard deviation suggesting that very large imprecisions are common.

NOTE As in SubCNN [35], we took the detections with 2D bounding box IOU ≥ 0.7 and computed their distance in the 3D world to their matched ground truth. It is very important to note that the positions of the ground truth 3D bounding boxes were reconstructed from the BB3TXT annotation and the ground plane position, using the same approach as for reconstructing the detected 3D bounding boxes! This means that the 3D positions of the ground truth 3D bounding boxes we were evaluating against were different than the original 3D bounding box annotations provided by the KITTI annotators because they were affected by the imprecise position of the ground plane. However, we argue that this is a fair comparison as the reconstruction of the detections is affected by the imprecision in the ground truth position as well. The detector would just be punished for not having a precise ground plane, which has nothing to do with the actual precision of the detector predictions.

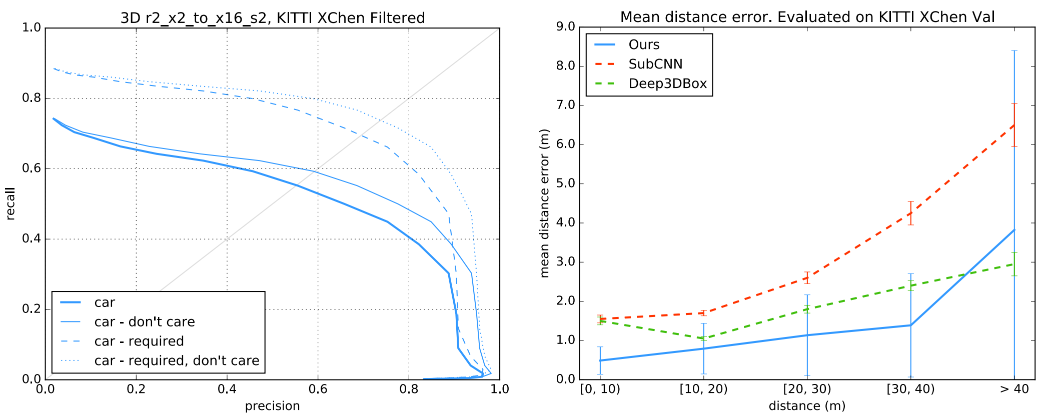


Figure 5.21. The performance of the 3D `r2_x2_to_x16_s2` detection network trained on the KITTI XChen training subset on 2D bounding box detection (left) and a comparison of the mean distance error (MDE) on the KITTI XChen Val dataset to SubCNN [55] and Deep3DBox [35] (right). The results for SubCNN and Deep3DBox are taken from [35]. See the note in the text about how we computed the MDE for our detector.

[1]) The splits may be downloaded at `https://xiaozhichen.github.io`

Chapter 6

Discussion and Analysis

The evaluation showed that we designed a new promising method based on the combination of ideas from the DenseBox, SSD, and MS-CNN algorithms, which is capable of detecting 3D bounding boxes of cars from monocular images. However, as all methods, it has several problems. We now asses these problems and suggest how to tackle them.

Truncated Cars A particular example of where our method lacks behind its competitors is the detection of very truncated cars. Examples of such annotated cars are shown in Fig. 6.1 in the bottom-right and bottom-left corner. This limitation comes from the chosen representation on the output. In the current setup, our method requires the center of the 2D bounding box of a car to be inside of the image in order to be able to detect it.

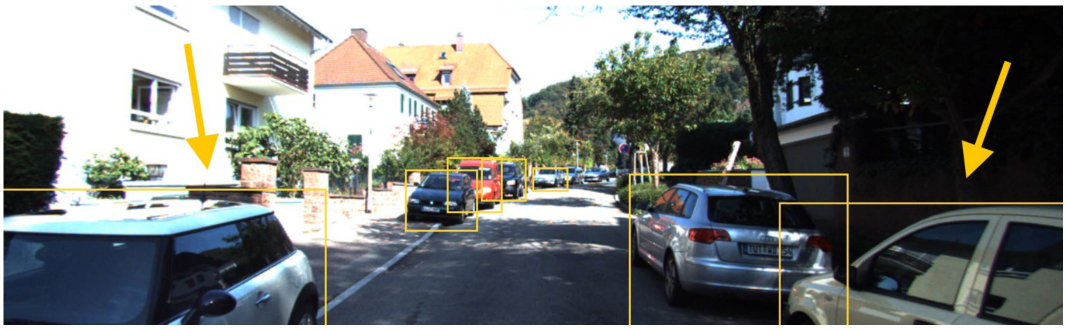
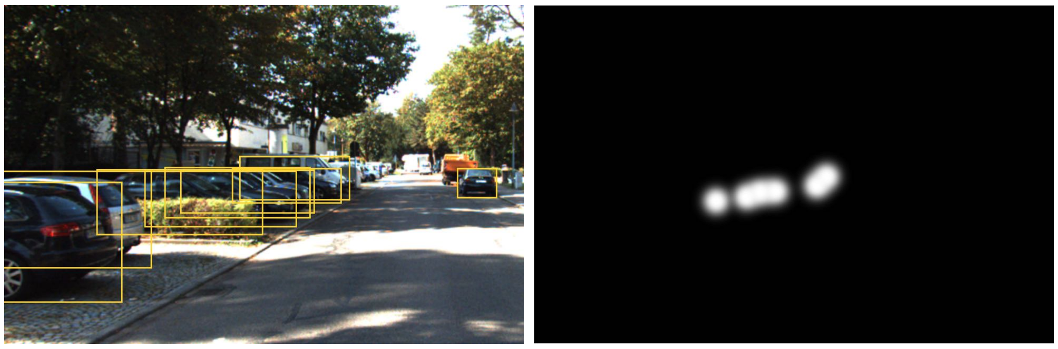


Figure 6.1. Examples of very truncated cars in the KITTI dataset in the bottom-right and bottom-left corner.

We argue that in practice detection of such cars from a camera is mostly not necessary. Such cars are very close to the camera, therefore, easily detectable by a different kind of sensor (radar or lidar). Also, an autonomous car is equipped with multiple cameras, and thus these cars can be easily detected on different cameras. Finally, since the scene is changing and the autonomous car is moving each of the close cars must have appeared further (smaller in the camera) in some previous point in time, therefore our car would have already had the information about the position of the car from the previous video frames.

If one would require detection of very truncated cars anyway, the input image can be padded with a black border, and thus provide room for the responses in the centers of the truncated cars.

Extensively Overlapping Cars The detection of cars with large overlap is a problem in general, which the presented detector shares as well. In our case, the problem is that the circles drawn in the center of the 2D bounding boxes of the cars are overlapping if the cars are in the same scale (response map) and occlude each other a lot. An



a) 2D bounding box annotated image. b) *Probability* response map.

Figure 6.2. Image with ground truth annotations and the corresponding ground truth *probability* response map. The response map is for the middle-scale cars and, therefore, shows 6 different Gaussian blobs corresponding to the 6 different bounding boxes in the middle of the image.

example of such a case is in Fig. 6.2. It is difficult even for a human eye to recognize the 6 different bounding boxes in the middle of Fig. 6.2a.

The problem of our representation is that the circles of very overlapping cars are merged together into a blur (Fig. 6.2b). It is even a bigger problem for the bounding box coordinate response maps because there we have to decide which bounding box we are regressing in each pixel. Since, in this case, one pixel in the response map corresponds to multiple bounding boxes, it is a problem.

Similar problem appears for example in the representation of YOLO [40], where they address it by allowing a single cell to detect 2 different bounding boxes.

3D Pose Estimation Another problem is the regression of the coordinates of the projected 3D bounding boxes. We directly predict the coordinates with respect to the response map pixel position regardless of the car orientation. However, our observations suggest that the network has a problem distinguishing the orientation of side-view cars. As can bee seen in Fig. 6.3, instead of learning the correct coordinates of the projections the network learns to average the coordinates of the left and right facing cars. This results in a completely incorrect car pose, either facing the camera or the other way around, or predicting some small bounding box in the middle.

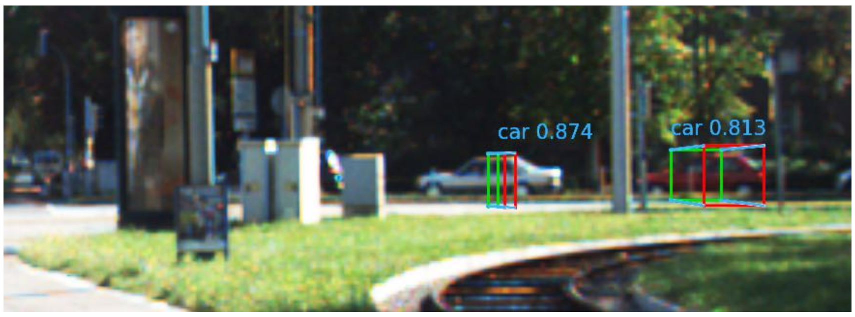


Figure 6.3. 3D bounding box detection with unsure pose. The detector is unsure whether the cars are facing left or right, and hence outputs the average coordinates of the 3D bounding box which is a nonsense.

The solution to this problem may be to use sub-categories divided by the orientation of the car with respect to the camera. We want to avoid the network to learn to average the coordinates of the left and right facing cars. Creating for example 4 orientation sub-categories (frontal, rear, left, and right view) would prevent this from happening as the coordinates would only be regressed in some given region.

Splitting the output of a neural network into multiple sub-categories has shown to be a well-working approach. Anchor boxes, introduced in Faster R-CNN [43] and then also used for example in SSD [33] and YOLO9000 [41], separate the bounding boxes on the output into several distinct shapes. 3D voxel patterns (3DVP) [54] in SubCNN [55] bring this to an extreme as they use hundredths of sub-categories based on orientation and occlusion of the cars. SubCNN is currently one of the best performing methods on KITTI. Training of these methods is carried out with discrete-continuous loss [35], which splits the objects into sub-categories, and then regresses the bounding box coordinates for each sub-category. Using such an approach is possible in our scenario as well and we believe it would bring a radical improvement to the 3D bounding box pose estimation.

Double Detection Associated with the previous problem is that the detector sometimes tends to predict multiple bounding boxes for a single car. Fig. 6.4 shows a case when a detector predicted a side-view bounding box and a rear-view bounding box on the same car. The fact that this is happening makes sense since the two bounding boxes have centers in different places and, therefore, do not interfere with each other in the *probability* response maps.

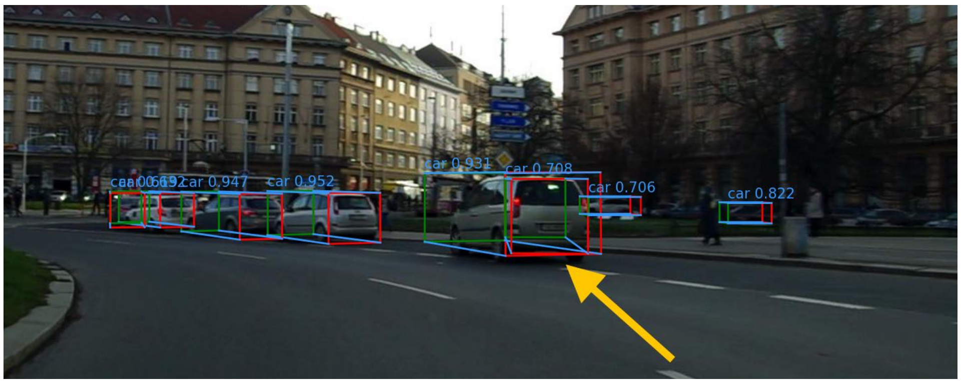


Figure 6.4. Double detection on a single car. The detector predicts both the side view and the rear view bounding box.

It is a problem since the network does not learn to completely suppress the irrelevant view by itself. A thorough solution requires a complex analysis of the detected bounding boxes. One such approach could be non-maxima suppression (NMS) in the 3D world. The two predicted bounding boxes significantly overlap in 3D, and thus filtering them in 3D could bring the suppression of the incorrect view. We elaborate about 3D NMS further in this chapter.

Hard Negatives It is a well known problem that neural networks are easily fooled, as described for example in [36]. Very often, they produce detections with high confidence that are in fact incorrect (false positives). Examples of such detections from our 2D bounding box detector are in Fig. 6.5. Without a very thorough examination of the learned network those false detections are inexplicable.

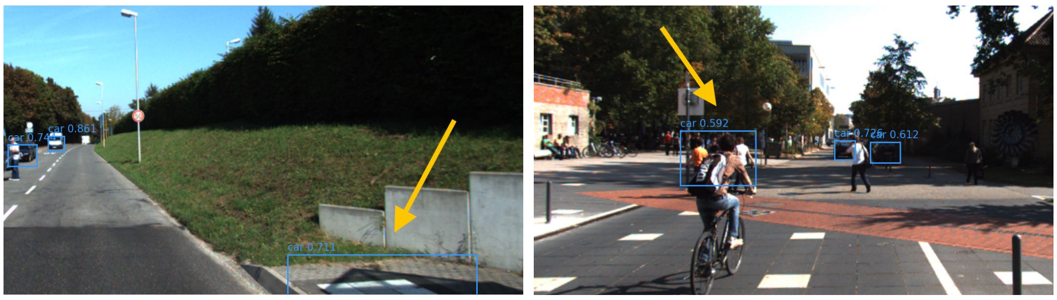


Figure 6.5. Examples of false positives with high confidence. High confidence prediction on sidewalk (left) and on a flock of people (right).

These detections may be caused for example by over-fitting to outliers in the training set, which we can reduce using dropout layers [50]. We did not use this technique because it prolongs training and training our network already took a significant amount of time (5 days for 2D and 10 days for 3D bounding boxes), however it might be a possible upgrade, which would reduce the false positive rate.

Another way to reduce the amount of false positives is hard negative mining during training. Such approach is used in the original DenseBox, however since we changed to positive/negative pixel weighting we did not re-implement it here. Hard negative mining is, however, a known working technique for reducing false positives.

Ground Plane The reconstructed 3D world position of the detected bounding boxes is heavily dependent on the position of the ground plane. The problem is that the requirement of a flat ground plane does not hold for arbitrary scenes. Even in KITTI, where the ground plane is very flat, we can sometimes see huge reconstruction errors (Fig. 6.6). Arguably, this is the biggest drawback of our method. A better solution would be to obtain a more precise ground plane equation from other sensors in the car or use stereo images.

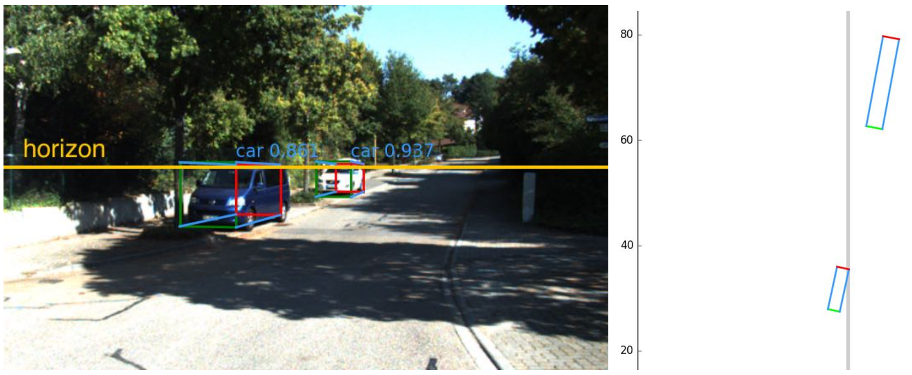


Figure 6.6. An example of a scene, where the road goes a bit down hill (see the used ground plane horizon line). The top view of the scene (right) shows a huge error in the 3D bounding box reconstruction (the y-axis is the distance from the camera in meters). The farther bounding box is estimated to be about 15 meters long.

3D Bounding Box Non-maxima Suppression Currently, we are performing non-maxima suppression of 3D bounding boxes on their 2D projection into the image (the same way as we do it for 2D bounding boxes). This approach is not ideal because it can filter out cars with IOU of their 2D bounding boxes even though they are actually correct detections. Providing we have more precise 3D reconstruction, the 3D bounding boxes could be filtered directly in the 3D world. The requirement of low or none IOU makes perfect sense when used for 3D bounding boxes as no two cars can intersect each other.

Weight Initialization At the current training setup we always initialize the weights randomly from scratch using the Glorot uniform initialization [22]. However, it was shown that using pre-trained weights for initializing the neural network will most likely lead to a superior performance. It would be interesting to try to initialize the weights of the network for example with the VGG [48] pre-trained weights.

Unsure Ground Truth Coordinates In our ground truth specification (Sec. 4.1) we require the labeler to make up the part of the bounding box from the occluded or truncated part of the car. It is important since we need to extract the centers of the object during training. However, we are teaching the network to precisely predict the coordinates of the bounding box corners, which in this case may come from imprecisely labeled coordinates since the labeler is just imagining where the car has its boundaries.

If the unsure coordinates were marked in the training labels as in Fig. 6.7 we could make use of this information during training. The gradient from the unsure coordinates would simply be nullified, therefore not taking part in training of the network. This way, the network would learn only on the precise labels.

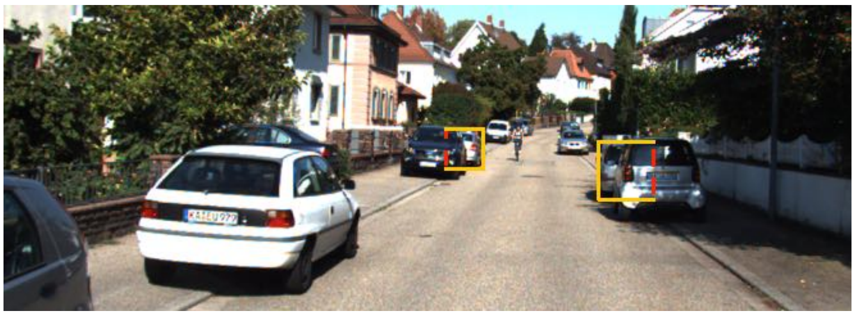


Figure 6.7. Ground truth labels with marked unsure sides (dashed red).

Conclusion

This work tackles the problem of the detection of 3D bounding boxes from monocular images using deep neural networks with the focus on autonomous driving applications. The proposed algorithm uses the output representation of DenseBox and the multi-scale network architecture inspired by SSD and MS-CNN, which were selected based on a thorough review of the existing DNN object detection methods.

We presented a fully convolutional neural network (FCN) adapted to detect 2D or 3D bounding boxes of cars from monocular images. The network was designed to detect cars of various sizes in a single pass without the need for an image pyramid. In contrast with previous deep neural network approaches to 3D bounding box detection, it is an end-to-end trained network that does not utilize an object proposal phase.

The used design alterations were thoroughly evaluated on a 2D bounding box detection network, first, in the single scale setting with image pyramid, and later in the multi-scale setting. When compared to original DenseBox, our 2D bounding box detection network has a superior performance on the KITTI easy car detection benchmark and it is also more than an order of magnitude faster. It is able to process the KITTI images in 10fps, thus being faster than all published methods, which have a superior performance on the KITTI car detection benchmark.

We created a novel representation of the 3D bounding box, which stores the image projections of the corners of the 3D bounding box. The proposed representation is very convenient because it does not depend on the image projection matrix. A detector using such representation on the output can, therefore, be used with no problems on a completely different dataset than it was trained on.

Finally, we present results of the 3D detection network and elaborate about the problems of the method. Possible solutions to the problems are suggested together with the ideas on further improvements of the method.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Shivani Agarwal, Aatif Awan, and Dan Roth. Learning to detect objects in images via a sparse, part-based representation. *IEEE transactions on pattern analysis and machine intelligence*, 26(11):1475–1490, 2004.

[3] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.

[4] James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*, volume 3. Citeseer, 2011.

[5] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *European Conference on Computer Vision*, pages 354–370. Springer, 2016.

[6] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Céline Teulière, and Thierry Chateau. Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. *CVPR*, 2017.

[7] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[8] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016.

[9] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2156, 2016.

[10] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *Advances in Neural Information Processing Systems*, pages 424–432, 2015.

[11] François Chollet et al. Keras: Deep learning library for theano and tensorflow. *URL: https://keras. io/k*, 2015.

[12] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[14] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[15] Mark Everingham, Andrew Zisserman, Christopher KI Williams, and Luc Van Gool. The pascal visual object classes challenge 2006 (voc2006) results, 2006.

[16] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.

[17] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[18] Juergen Gall and Victor Lempitsky. Class-specific hough forests for object detection. In *Decision forests for computer vision and medical image analysis*, pages 143–157. Springer, 2013.

[19] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[20] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[21] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[22] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[23] Kevin Gurney. *An introduction to neural networks*. CRC press, 1997.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision*, pages 346–361. Springer, 2014.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[26] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[27] Lichao Huang, Yi Yang, Yafeng Deng, and Yinan Yu. Densebox: Unifying landmark localization with end to end object detection. *arXiv preprint arXiv:1509.04874*, 2015.

[28] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[30] Ben Kröse, Ben Krose, Patrick van der Smagt, and Patrick Smagt. An introduction to neural networks. 1993.

[31] Yann Le Cun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *The Connectionist Models Summer School*, volume 1, pages 21–28, 1988.

[32] Karel Lenc and Andrea Vedaldi. R-cnn minus r. *arXiv preprint arXiv:1506.06981*, 2015.

[33] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, and Scott Reed. Ssd: Single shot multibox detector. *arXiv preprint arXiv:1512.02325*, 2015.

[34] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[35] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry. *arXiv preprint arXiv:1612.00496*, 2016.

[36] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

[37] Michael A Nielsen. Neural networks and deep learning. *URL: http://neuralnetworksanddeeplearning.com*, 2015.

[38] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.

[39] Patrick Poirson, Phil Ammirato, Cheng-Yang Fu, Wei Liu, Jana Kosecka, and Alexander C Berg. Fast single shot detection and pose estimation. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 676–684. IEEE, 2016.

[40] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640*, 2015.

[41] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[42] Jimmy Ren, Xiaohao Chen, Jianbo Liu, Wenxiu Sun, Jiahao Pang, Qiong Yan, Yu-Wing Tai, and Li Xu. Accurate single stage detector using recurrent rolling convolution. *CVPR*, 2017.

[43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[44] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[45] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1665–1672. IEEE, 2011.

[46] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[47] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[48] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[49] Jan Sochman and Jiri Matas. Waldboost-learning for time constrained sequential detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 150–156. IEEE, 2005.

[50] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[51] Jasper RR Uijlings, Koen EA van de Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[52] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[53] Naiyan Wang and Dit-Yan Yeung. Learning a deep compact image representation for visual tracking. In *Advances in neural information processing systems*, pages 809–817, 2013.

[54] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Data-driven 3d voxel patterns for object category recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1903–1911, 2015.

[55] Yu Xiang, Wongun Choi, Yuanqing Lin, and Silvio Savarese. Subcategory-aware convolutional neural networks for object proposals and detection. *arXiv preprint arXiv:1604.04693*, 2016.

[56] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2014.

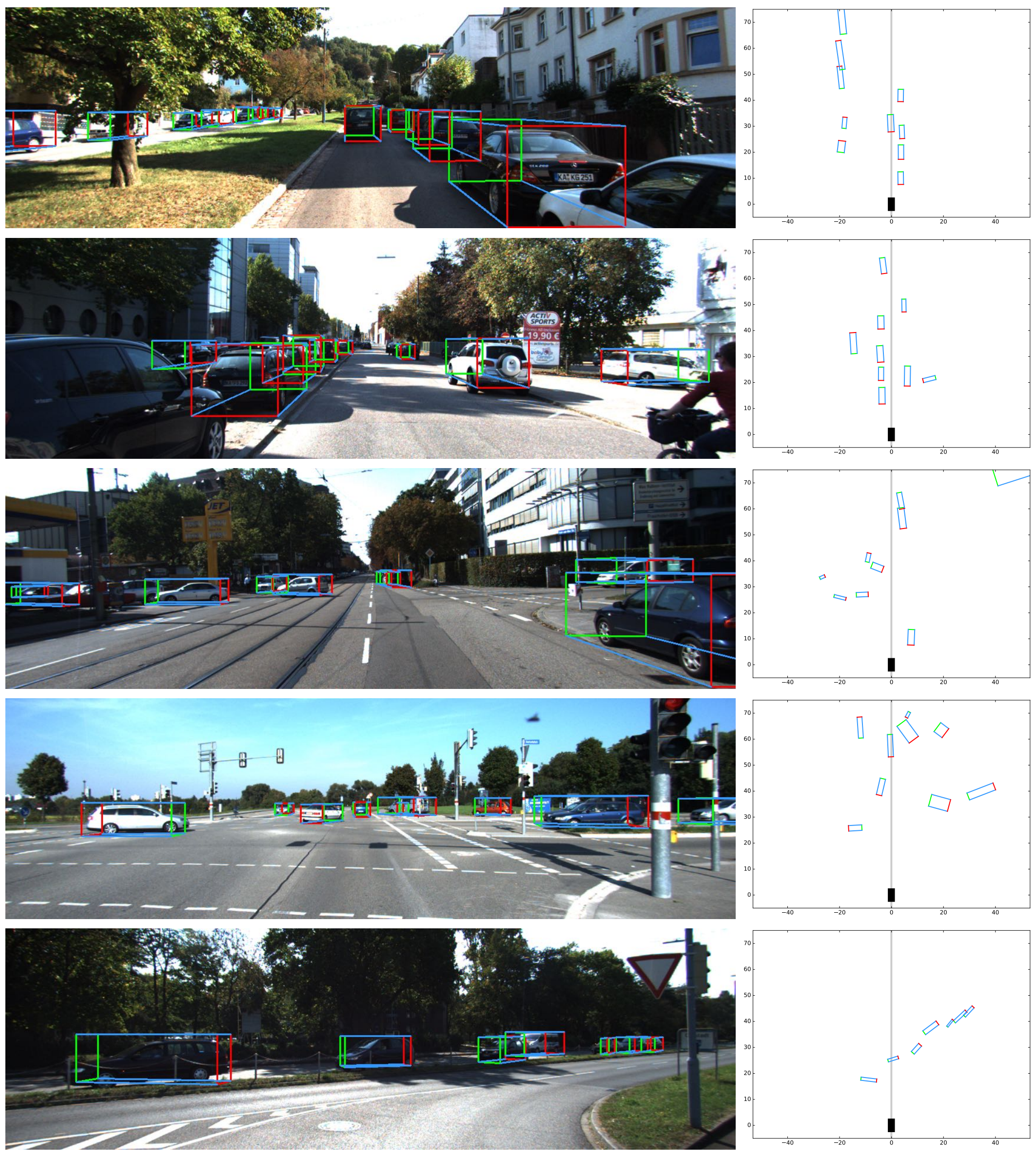
[57] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

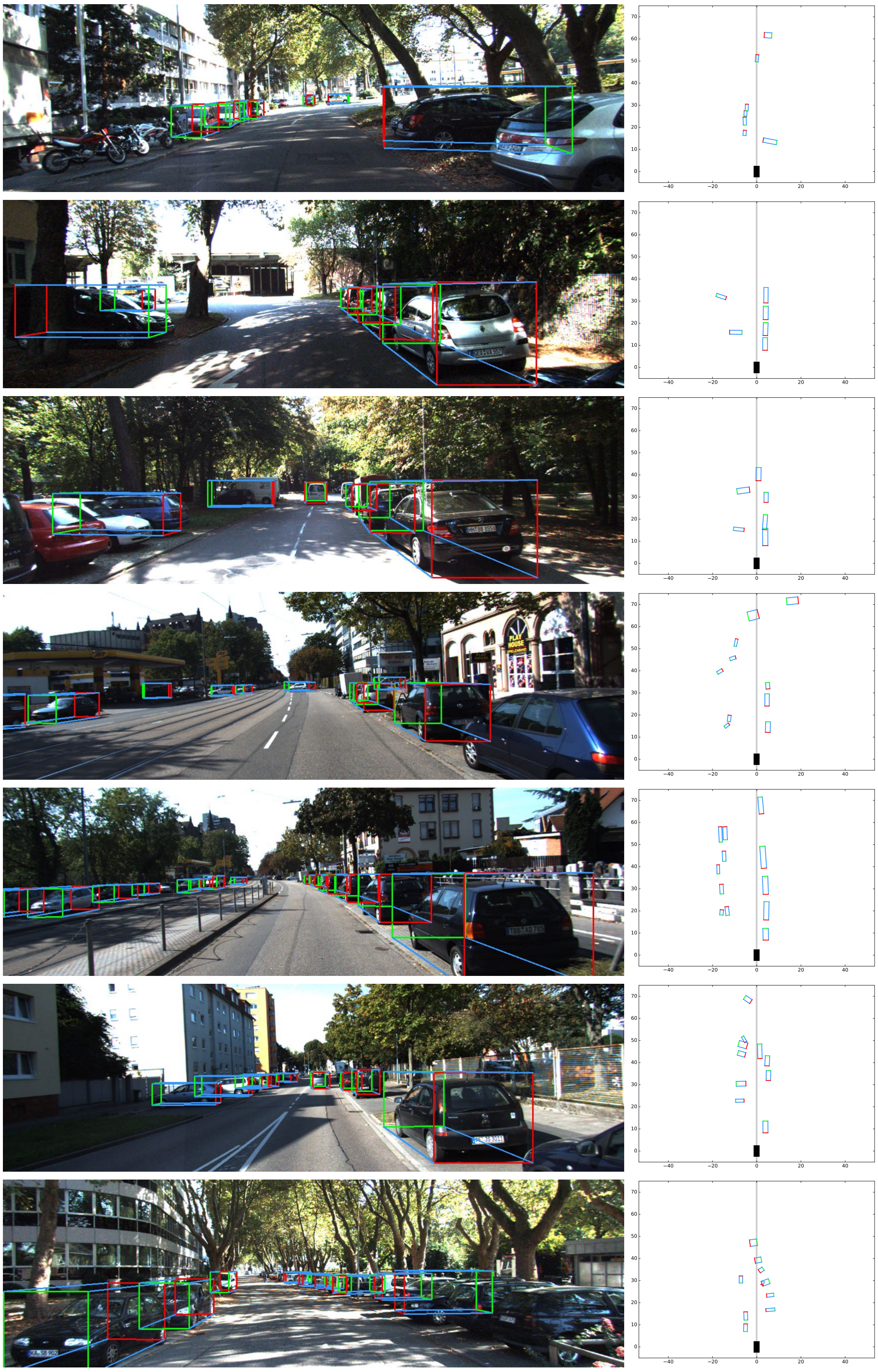
[58] Heiga Ze, Andrew Senior, and Mike Schuster. Statistical parametric speech synthesis using deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7962–7966. IEEE, 2013.

[59] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *European Conference on Computer Vision*, pages 391–405. Springer, 2014.

Appendix A
3D Bounding Box Detections

Additional sample images from the KITTI test set with the detections from our best 3D bounding box detector described in Sec. 5.4.2, including the top view (xz-plane) of the scene. Only detections with confidence ≥ 0.6 are displayed. Note the missing cars and the distortions in the cars' dimensions in the top view when the scene does not have a flat ground plane.





Appendix B
Contents of the Attached CD

Path	Description
/Master's thesis.pdf	This PDF file.
/Master's thesis.mp4	Video with 2D and 3D detections on unseen data.
/datasets/	The BBTXT, BB3TXT, and PGP annotations.
/nets/	Trained networks and training scripts.
/results/	BBTXT and BB3TXT detections and PR curves.
/software/	The Python and C++ source code.