Abstract

1 Introduction

1.1 Motivation

The autonomous vehicles (AVs) is the evolutionary direction of automobiles due to its promising reliability and efficiency, as well as the underlying commercial profits. To gain a comprehensive perception of the driving environment is the very essential step for AVs. Object detection is one of the key challenges of perception. Thanks to the remarkable advancement of deep neural networks, great achievements have been make on 2D object detection, while 3D object detection is still underdeveloped. For example, according to the KITTI Object Detection Benchmark[36], the Average Precision (AP) of top 10 2D car detection algorithms is over 90% whereas best performance of 3D car detection is only 73.66%. This gap results from the difficulty of adding the third dimension and the orientation of the 3D bounding box.

For AVs, 3D information of the surrounding vehicles is indispensable because it expresses the vehicle dimensions, locations and orientation in the real 3D environment, which is essential for AVs to perform planning and decision making. To find a safe and efficient route, the information of actual and potential movement, dimensions, and location of other vehicles is necessary. In order to perceive the movement, 3D localization, orientation, and time are used to recover the velocity. The decision-making systems are more complicated and require more detailed 3D information, for example, the Bosch's Autonomous Emergency Braking systems (AEB) requires the distance to each part of a foregoing vehicle to decide whether or which level of brakes to apply.And it is common that some parts of the vehicle are occluded by other objects or truncated by the boundaries of the image. Thus, the exact location of each vehicle part and the visibility property of these parts are necessary.

Here we propose an approach that can simultaneously present the 3D dimensions, 3D localization, 3D bounding box orientation, 3D parts location, and parts visibility of a vehicle by giving a monocular image and all the 2D bounding boxes for vehicles. Figure 1 shows one output image example.

TBD

1.2 Contributions

The first contribution is that our approach can predict not only the 3D bounding box for each vehicle but also the 3D position of each vehicle part, even if these parts are occluded by other objects or truncated by the boundaries of the image. The foundation behind this is that vehicles are rigid objects and their geometric characteristics have a lot in common despite of the vehicle types. Therefore, these shared geometric characteristics serve as the priors which make it reasonable that each vehicle can to be expressed by a 3D artificial model along with a scaling vector. This 3D model is integrated by vehicle parts which are further encoded by characteristic points. Besides, we apply regression rather than detection to search and locate these characteristic points. By doing so, our approach can find all the rest of parts, as long as it ascertains the existence of the vehicle object and some parts. Therefore, even through some parts are invisible to the camera, our approach can still localize them.

The second contribution is the proposed semi-automatic labelling process which generates additional labels to create a new dataset for our approach. Deep neural networks are greedy for data. Manual labelling is time-consuming and error-prone, let alone it is infeasible to correctly label the very far vehicles or occluded parts in the image. Therefore, we eliminate the human labor to the extend where only the 3D models need to be labelled manually. Then the process automatically selects the best-matching model and projects this model into the real image to generate labels, *e.g.* visibility and characteristic points. Besides, this process can be easily generalized to other tasks, as long as they require 2D geometry information in the image coordinate and 3D geometry knowledge in the world coordinate.

The third contribution is the multi-task framework which includes a prediction neural network and a inference block. The network can simultaneously perform vehicle parts localization, visibility characterization, and template proximity prediction at high accuracy level and consequentially, the inference block performs the vehicle dimension estimation, 3D vehicle localization, and rotation recovery. All these tasks can be completed in real time s which makes it applicable to directly process the video sequence of the vehicle's front camera.

TBD

2 Related Work

2.1 Mono RGB Image Based Approaches

Most vehicles are equipped with monocular cameras and RGB images have detailed texture information with high resolution so that plenty of approaches are developed based on monocular RGB images. Images are mainly contain 2D information so most approaches in this category require other helps.

One way is to utilize the geometry information of the vehicles. In this category, a set of algorithms recover the 3D bounding box based on 3D car models. [98, 99] model the 3D geometry representation of objects with 3D wireframe models. And the 3D bounding box is estimated based on the geometry constraints of the vehicle in image and its corresponding 3D wireframe model. 3DVP [93] performs 3D detection based on 3D Voxel Patterns which are generated from the KITTI dataset [36] and a set of 3D CAD models to encode the geometric information of objects. Mono3D [23] introduces 3D proposals. It exhaustively places 3D bounding boxes on the ground-plane as proposals, then scores each proposal based on several hand-crafted geometric features, and applies a CNN to score the most promising candidates to generate the final 3D detections. Recently, CNNs are introduced to estimate some key features for 3D detection. Deep3DBox [62] applies two CNNs to estimate the orientation and dimensions respectively, makes use of the geometric criterion that a 3D box should fit tightly within the 2D box of the vehicle to estimate the translation, and finally integrates them together to perform 3D detection. Deep MANTA [22] generates 3D detection based on 2D detection. It applies a CNN to predict some 2D key points and the template similarity of the vehicles in image and recovers the corresponding 3D key points and 3D dimensions with some 3D CAD vehicle models, and finally perform 3D detection via 2D-3D matching. Our approach is similar to Deep MANTA in spirit.

Another way is to use temporal information. [29, 81] make use of motion structure and ground estimation to perform 3D detection from 2D bounding box.

These approaches are sensitive to the assumptions made and the parameter estimation accuracy so that they perform poorly on the 3D detection task compared to methods that use point cloud data . TBD

2.2 Stereo RGB Image Based Approaches

A pair of stereo RGB images can provide the depth information of the current scene. 3DOP [25] recovers depth from stereo images and generates 3D box proposals with the constraints from depth and other geometry characters, which are forwarded to modified Fast R-CNN [37] pipeline for object detection and pose estimation. [68] extends 3DOP by applying a separate CNN to extract features from depth and integrating them together for the final 3D detection.

2.3 Point Cloud Based Approaches

A LiDAR point cloud is a collection of 3D points acquired by a LiDAR laser scanner. It can represent 3D information of the surroundings but its resolution is lower than images'.

One set of approaches apply a point cloud by converting it into a 2D array and making use of the image-based methods to perform 3D detection tasks. This can alleviate the inherent problem that LiDAR point is often sparse and irregular in 3D voxel representation . VeloFCN [58] projects the point cloud to the front view to generate a 2D point map and then applies a fully convolutional neural network to estimate the 3D bounding boxes for vehicles on this 2D point map. [92, 24] also fall into this category. Besides, the pre-trained models based on RGB images can be used to initialize the CNNs, which has been proven to be beneficial [42]. However, the 2D representation of LiDAR cloud points suffers from object size variations because of its distance to the sensor and object overlapping.

Another category of approaches transforms the cloud point into a 3D voxel grid representation. Various quantities are used to encode voxels [83, 82, 57, 88, 32], For example, in [88, 32], the information in one non-empty cell is encoded with six quantities while empty cells contain no information. For the single-stage detectors, *e.g.* Sliding Shapes [83] and Vote3D [88], slide window across the 3D voxel space and apply SVM classifiers to perform 3D detection. To improve the performance, Vote3Deep [32] uses a voting strategy with sparse 3D convolution. And [57] feeds the point cloud voxel directly to a 3D FCN to generate 3D bounding boxes. For two-stage framework, *e.g.* Voxelnet, extends the 2D RPN [73] to 3D to generate 3D proposals and apply a refine network to score these proposals [97]. 3D CNNs boosts the 3D detection performance while the computation is very expensive and sparsity is to be explored.

2.4 Sensor Fusion Based Approaches

Images are of high resolution but lack depth, while LiDAR data has 3D information but are sparse and irregular. Therefore many works have investigated combining these complementary data sources together to build robust 3D object detectors.

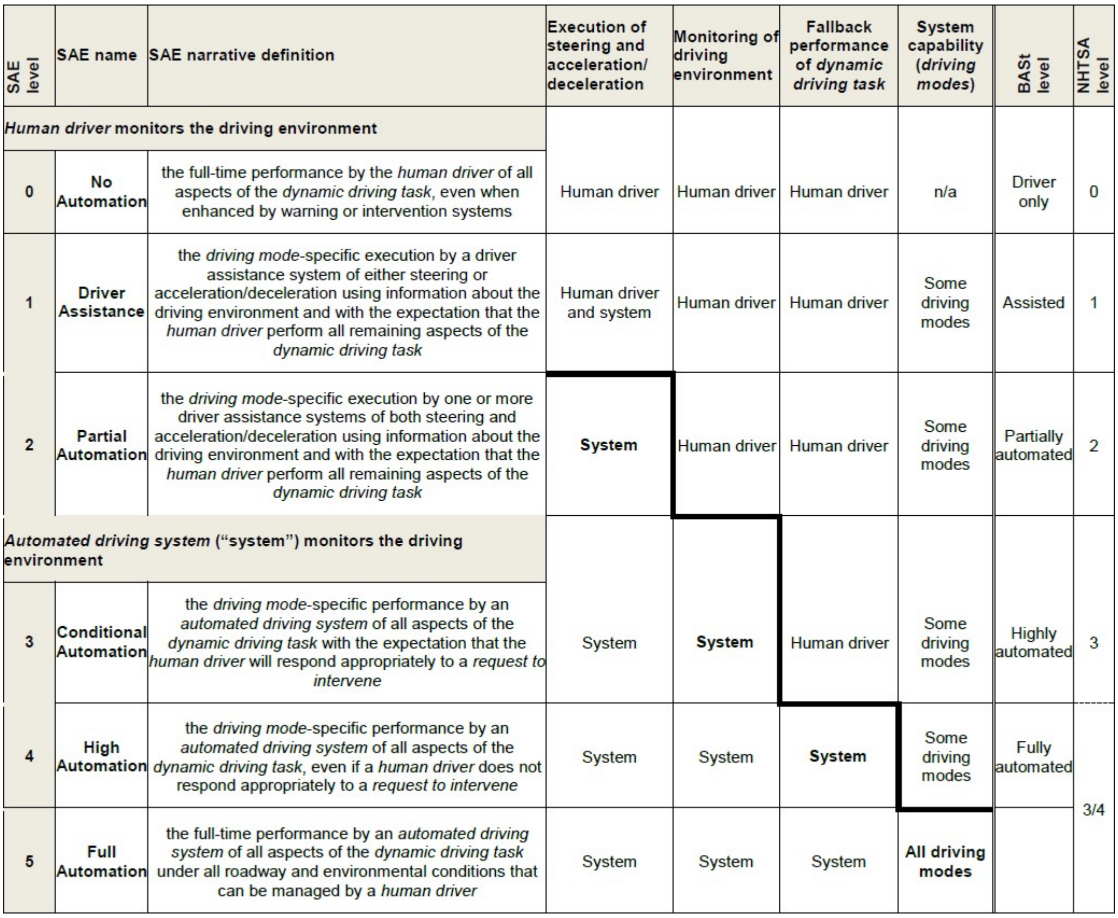
One investigated direction is to extract information from these two sources separately for sub-functions. Frustum PointNets [70] first generates frustum point cloud proposals based on 2D object detection, then performs 3D instance segmentation in each frustum point cloud and finally applies a box estimation net to estimate the final 3D bounding box for each object. Similarly, [30] first generates 2D bounding boxes and estimate vehicle dimensions for the target vehicles on images. Secondly, a model fitting algorithm estimates the 3D bounding box on a subsets of the point clouds which fall into the 2D bounding box after projection. Finally a refine 2D CNN performs the final 3D box regression and classification.

The other is to fuse these two kinds of data and process it as a whole. There are various ways to fuse data. One intuitive way is projection, *e.g.* [31] converts the point cloud to a dense depth image and then appends it as an additional channel of image, [77] extends [31] via converting the cloud point to a three-channel HHA map. Their 3D detection score is low mostly due to the lose of 3D information during projection. The more advanced way is to fuse their feature maps. MV3D [26] first generates 3D proposals in LiDAR's bird's eye view and projects these proposals to the features maps of the image and the bird's eye view and front view of LIDAR data. Then a deep network fuses three ROI pooling regions and performs 3D detection based on the fused features. PointFusion[94] applies a ResNet [44] to extract appearance and geometry features from image crops defined by the 2D bounding boxes and a modified PointNet [71] to process the raw point cloud simultaneously, and finally uses a novel fusion network to integrate both features and estimate 3D bounding box. Recently, Zining *et al.* [90] applies an innovative sparse non-homogeneous pooling layer to fuse the features extracted from bird's eye view of LiDAR data and front view camera images by two separate CNNs before region proposal stage. And then a single-stage detector adapted from [60] is designed to perform 3D detection on these fused data without RoI pooling, which is 6 times faster then MV3D. AVOD [54] applies double feature fusions. It uses two identical CNN to extract features from images and LiDAR data respectively and exploits a Multimodal Fusion Region Proposal Network to generate vehicle proposals from the fused feature crops, projection regions of anchors in the feature maps. Finally it applies a Multiview Detection Network to perform 3D detection on the fused feature crops corresponding to the proposals. It achieves the best performance on KITTI 3D vehicle detection up till now.

3 Background

3.1 A Short History of Autonomous Vehicles

Autonomous Vehicles (a.k.a. Automated/Driverless/Unmanned/Robotic/Self-driving Vehicles) is a relatively vague concept to the public. Actually, it covers a continuum from traditional fully human-driving automobiles to fully self-driving vehicles, as SAE classified in the Table 1.



SAE level	SAE name	SAE narrative definition	Execution of steering and acceleration/ deceleration	Monitoring of driving environment	Fallback performance of *dynamic driving task*	System capability (*driving modes*)	BASt level	NHTSA level
Human driver **monitors the driving environment**								
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a	Driver only	0
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes	Assisted	1
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes	Partially automated	2
Automated driving system **("system") monitors the driving environment**								
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the *dynamic driving task* with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes	Highly automated	3
4	**High Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes	Fully automated	3/4
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**		

Table 1: Summary of levels of driving automation[7]

Contrary to the intuition, the idea of Autonomous Vehicles has a long history. The experiments of this fictional idea can be tracked back to 1920s and the technology behind it was radio control [6].

But the truly autonomous cars did not show up until 1980s, even though they could only move slowly on clear streets and required large human intervention.

Mercedes-Benz demonstrated a robotic van based on saccadic vision [78]. The Autonomous Land Vehicle (ALV) project funded by The Defense Advanced Research Projects Agency (DARPA) cultivated automatic vehicles based on Computer Vision, LIDAR, and autonomous robotic control [52]. Carnegie Mellon University initiatively applied neural network to control the vehicle [69].

In 1990s, huge progress was made.The VaMP from Daimler-Benz drove more than 1000 km, achieving the maximum speed of 130 km/h on a normal Pairs highway semi-autonomously [78]. The Navlab project in Carnegie Mellon University achieved a 5000 km journey cross America with only 1.8% human interventions [3]. The ParkShuttle in Netherlands could autonomously navigate itself on a dedicated lane as an automated people mover [66].In this decade, the experiments were mainly carried out in highway scenarios rather than urban scenes.

In 2000s, competitions promoted this technology a lot. One of the most famous is The DARPA Grand Challenge in U.S. who offered $ 1 million for the first prize. In 2004, no vehicle completed the 241-km journey autonomously while 5 teams achieved this goal in 2005 [50]. And in Grand Challenge III 2007 , known as Urban Challenge, 6 vehicles finished the event which was a 96 km urban route involving with traffic regulations and other vehicles [51].

In 2010s, Autonomous Driving Technology starts to take off. Numerous events and projects have been carried out and considerable companies, universities, and research centres have engaged into this field. Based on the progress made before, many autonomous vehicle systems are being tested or even brought into production. Notable events covers the VisLab Intercontinental Autonomous Challenge in 2010 [19] and the Intelligent Vehicle Future Challenges from 2009 to 2013 [63]. In industry, Tesla Motor released AutoPilot that is able to perform automated parking and lane control with autonomous driving, braking and speed adjustment in 2014, and Audi started the first production car, A8, reaching Level 3 of Automation in 2017 [2].

Based of the efforts in the last decades, Autonomous Driving gradually transform from dream to reality. Its bonus covers safety guarantee, congestion reduction, land use efficiency, energy and emission reduction, economic benefits and so on. However, the Level 5 vehicles are so far from maturity that the research and development are in high demand.

3.2 Deep Learning Technology

Recently, Deep Learning (DL) has shown its impressive power in a variety of tasks, especially in some complex tasks that can not be explicitly programmed by hand, such as anomaly detection and online advertising. DL delivers the solutions by learning from data automatically via a general learning procedure which dominantly makes use of backpropagation and optimization algorithms, *e.g.* gradient descent.

It attracts the world-wide attention mainly by outperforming other classic machine learning algorithms, *e.g.* Support Vector Machine (SVM), in many competitions involved with image classification, object detection, or nature language processing. This is because DL applys a deep automatic feature learning architecture, *i.e.*, a artificial neural network model with multiple hidden layers, to learn deep distributed hierarchical nonlinear representations which yield better performance for learning tasks, *e.g.* in terms of classification accuracy. Such representations bring many good properties, such as feature reuse, parameter sharing, multiple levels of progressive abstraction, and invariance to local changes of the raw inputs, and thus provide better predictive power than classic machine learning algorithms [13].

Convolutional neural networks (CNNs) is one of the major branches of artificial neural networks. A CNN is a feed-forward multi-layer neural network, typically consisting of one or more convolutional layers, interleaved by some pooling layers, and finally followed by some fully-connected layers. This modern framework of CNNs is established by LeCun *et al.* when he proposed the handwritten digit classifier LeNet-5 [55]. Afterwards, more and more deeper architectures are explored such as AlexNet [76], VGGNet [79], GoogleNet [85] and ResNet [45]. In general, deeper architectures, bring better feature representations, and closer approximations to the target function, as well as more complex models which are more difficult to train and easier to be overfitting. Therefore many approaches have been proposed to address such problems.

CNNs are specifically designed to work with problems taking images as inputs, such as image classification, object detection and pose estimation. The abovementioned CNNs examples all made great performance in their respective tasks, *e.g.* ResNet won the first prize in ILSVRC 2015.

The core building block, convolutional layers, is used to compute feature maps with convolution kernels. Each neuron in a convolutional layer is connected to a local region in the previous layer, called receptive field, and computes an output by performing an convolution operation (element-wise matrix multiplication) between its weights (kernel) and the connected region followed by an non-linear activation function. A nice property of CNNs is that the kernel is shared by all receptive fields in the preceding layer when computing the corresponding feature map. Thus, each feature map is used to capture exactly the same feature but at different locations. And this characteristic can substantially reduce the number of parameters in CNNs, which will lead to faster training [65].

The pooling layers perform a downsampling operation, typically max pooling [18] and average pooling [89], along the spatial dimensions of feature maps to achieve shift-invariance.

Normally, successive convolution layers detect more abstract features, *e.g.* wheels, than the preceding ones that tend to detect low-level features, *e.g.* curves and edges. Therefore, after the operations performed by several convolutional and pooling layers, progressively higher-level features can be obtained to feed a fully-connected layer whose goal is to perform high-level reasoning to generate the global semantic information [79, 46].

For classification problems, the output layer of CNNs usually deploys the softmax operator [76] or the SVM method [87] to output discrete results. While regression tasks require continuous-valued predictions so that the output layer should have a linear activation function, *e.g.* weighted sum, along with a proper cost function, *e.g.* mean squared error [96].

The training for CNNs is a global optimization problem by minimizing the defined loss function. Normally, CNNs can be trained end-to-end efficiently with back-propagation together with an optimization algorithm, such as stochastic gradient descent [91]. The mechanism behind it is that gradient of the loss function w.r.t. all parameters is calculated and then used to update the parameters to the direction of minimizing the loss function based on iterations over the full batch or mini batches of the training dataset.

3.3 Computer Vision

Computer vision is an interdisciplinary field that enables computers to interpret images just as what we human can do. Its goal is about automatic extraction, analysis, and understanding of the information from images [5]. There are a huge variety of ways to process images and a marvellous diversity of applications in this board field, ranging from replicating human visual abilities to creating nonhuman visual capabilities [41]. Some real-world applications include object recognition or detection, 3D model building, motion capture, etc [86].

Despite all the progress achieved, computer vision systems are still so underdeveloped that they can't match the visual ability of a 5 year-old child. Vision is natural and effortless for humans but factitious and demanding for computers [10]. The difficulty in part results from that vision is a inverse problem where we attempt to recover the unknowns, *e.g.* shape and illumination, based on insufficient information of their causes, *e.g.* models [86]. Therefore, it is hard to conclude the clear rules that can help get a solution explicitly, especially for complex problems, *e.g.* 3D object detection. But currently, Deep Learning seems to find a way out and pushes a big step forward.

Next in this section, we introduce some basic knowledge in computer vision related to our approach.

3.3.1 Image Formation

Imaging systems or cameras are some devices that allow the projection of light from 3D points to some 2D medium that records the light pattern. Figure 1 (a) shows a pinhole imaging model which is able to capture the light pattern, the inverted candle image, by allowing a very tiny cone of rays issued at every point of the source, the real candle, to project on the image plane. This is called pinhole perspective projection which is primitive but provides an acceptable approximation of the imaging process [35].

The projection equation can be derived from Figure 1 (b), where (O, i, j, k) is the pinhole camera coordinate system and the origin O is at the pinhole, $p = (u, v, d)^T$ is a point in image, and $P = (x, y, z)^T$ is a point denotes the source. As light travels straight in the same medium, p, O and P are collinear, and therefore we can deduce

$$\begin{cases} u = d\frac{x}{z}, \\ v = d\frac{y}{z} \end{cases} \tag{1}$$

Modern cameras are built on lenses that can gather more light to make the image more bright while maintaining its sharpness. But the imaging process is very similar to the pinhole camera. Lenses can also introduce some aberrations, *e.g.* spherical aberration, radial distortion, and chromatic aberration [35]. Therefore, correction is necessary.

3.3.2 Intrinsic and Extrinsic Parameters

To project a 3D point in the world coordinate system, first we have to transform this point to the camera coordinate system and then transform it to the image plane. The first transformation depends on the extrinsic parameters while the second rely on intrinsic parameters.

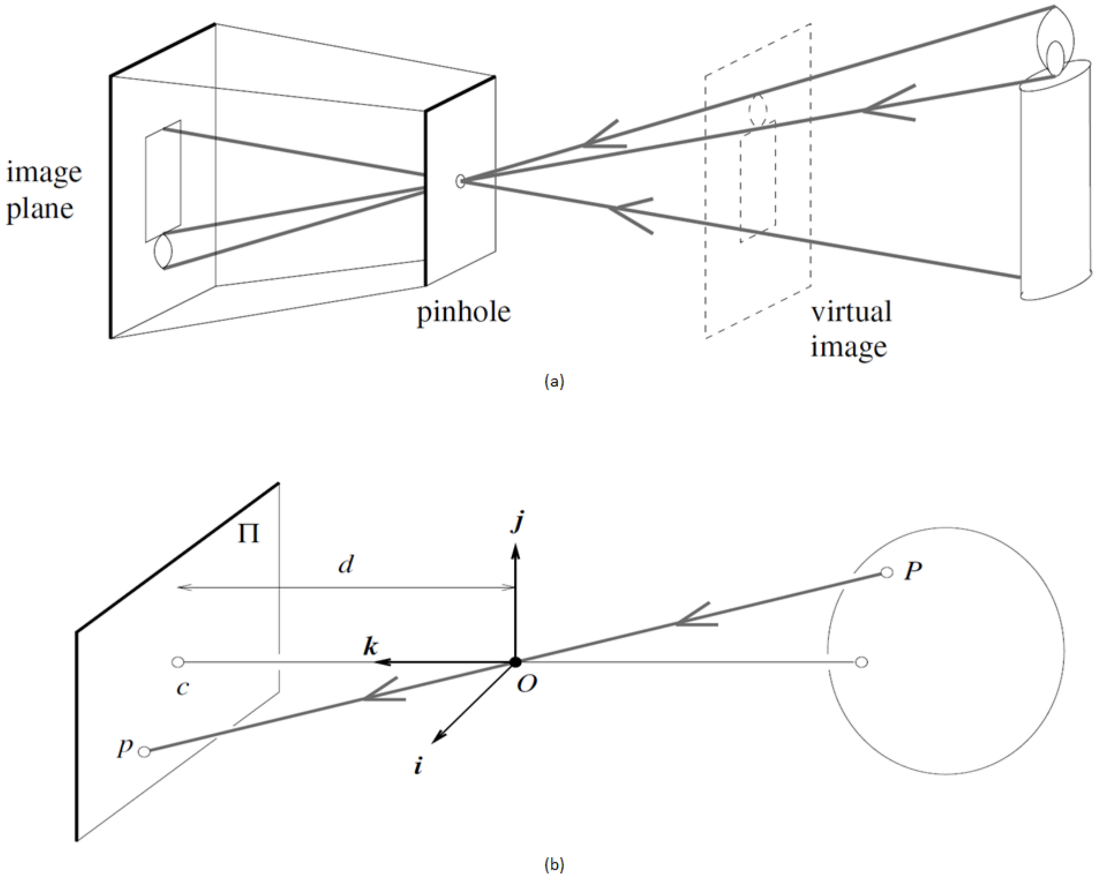


Figure 1: (a). the pinhole imaging model; (b). geometric model for perspective projection. [35]

Intrinsic parameters include the focal length f just like the d in Figure 1 (b), the image coordinates origin (u_0, v_0), and the skewed angle θ of two image axes. Coordinates in image plane are usually expressed in pixel which can be square or rectangular. So let's assume α and β are the value expressed f with horizontal and vertical pixel-meter scales. Therefore, we can finally transform Eq. 1 into

$$\begin{cases} u = \alpha \frac{x}{z} - \alpha \cot \theta + u_0, \\ v = \frac{\beta}{\sin \theta} \frac{y}{z} + v_0 \end{cases} \tag{2}$$

When it is written in matrix form as Eq 3, the 3×3 matrix K is the intrinsic matrix of the camerac. Note that P is expressed in camera coordinate system and p is in image coordinate system.

$$p = KP \Leftrightarrow \frac{1}{z}\binom{u}{v} = \begin{pmatrix} \alpha & -\alpha\cot\theta & x_0 \\ 0 & \frac{\beta}{\sin\theta} & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{3}$$

Extrinsic parameters defines a rigid transformation from world coordinate system to camera frame. A rigid transformation has six degree of freedom, including three Euler angles expressed in a 3 rotation matrix R and three translation components along each axis expressed in a 1×3 translation vector t. A point expressed in homogeneous coordinates is $P = (x, y, z, 1)^T$. Homogeneous coordinates can simplify various geometric transformations into matrix multiplication [35]. Therefore, this rigid transformation expressed in homogeneous coordinates is

$$P^C = T_W^C P^W, where\ \ T_W^C = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \tag{4}$$

P^C and P^W are coordinates of the same point expressed in world coordinate system and camera coordinate system respectively. T_W^C is the extrinsic matrix of the camera.

To put it all together, the projection equation in homogeneous coordinates is

$$p = \frac{1}{z} M P^W = \frac{1}{z} K \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} P^W \tag{5}$$

where M is the **perspective projection matrix**.

4 My Work - Technical Detail

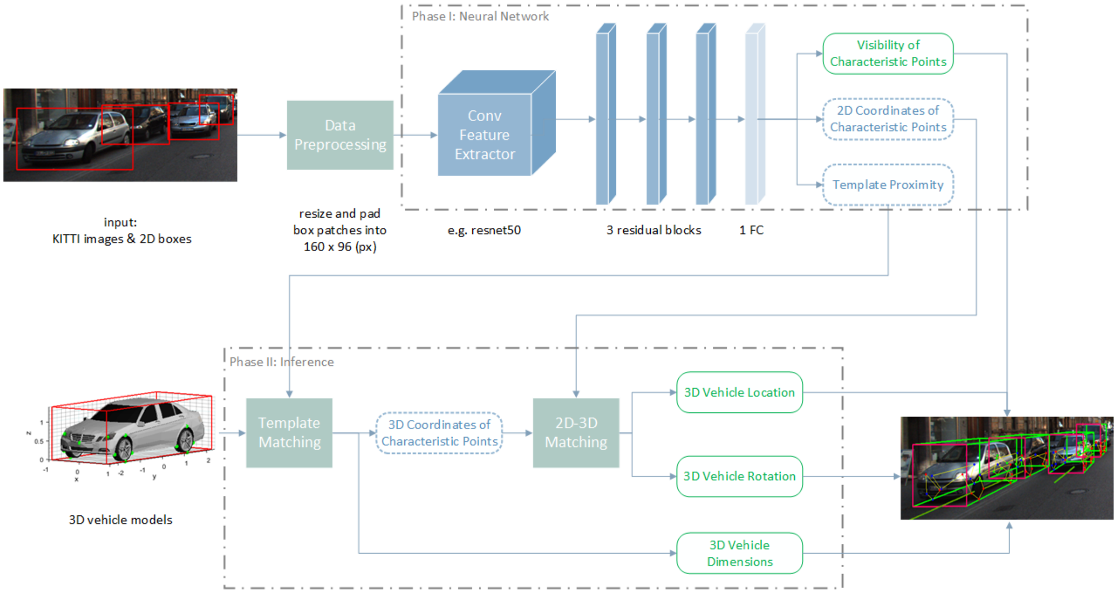


Figure 2: The Architecture of our Approach

In this section, we elaborate our approach for 3D pose estimation of vehicles in monocular images. Our approach is based on 2D vehicle detection, *i.e.*, the 2D bounding boxes of vehicles are given as prior, since 2D object detection techniques, such as YOLO [72], Faster R-CNN [73], SSD [61], R-FCN [28], and FPN [59], have achieve very high accuracy while end-to-end 3D object detection performance is still primitive. The overall architecture of the approach is illustrated in Figure 2. It consists of two main phases. The first one is the CVT Network which takes the KITTI images and the 2D bounding box as inputs and predicts the visibility property and 2D coordinates of all characteristic points, as well as the template proximity. The details are presented in section 4.2. The other phase estimates the 3D location, rotation, and dimension of each vehicle based on the 3D vehicle dataset and the outputs of the CVT Network. Section 4.3 demonstrated this phase in detail. The KITTI dataset and the 3D vehicle dataset are described in section 4.1.

4.1 Data and Labelling

4.1.1 KITTI Dataset

The KITTI 2D/3D object detection challenge is dedicated to autonomous driving and releases a dataset containing 7481 images for each one of 4 cameras and their associated labels and calibrations [36]. The dataset covers scenarios of city, residential, road, campus, and person. We only use the images taken by the left colour camera. The example image is shown in Figure 3. The associated labels are show in Table 2. The calibration is given as transformation matrix.

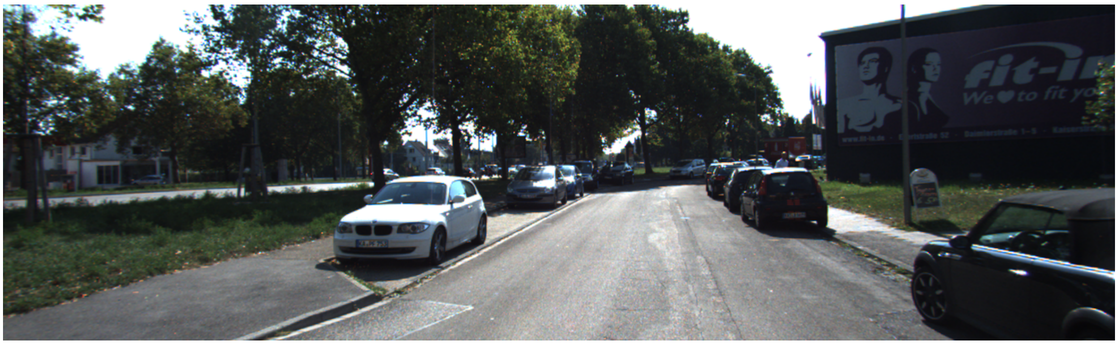


Figure 3: Example image from KITTI dataset captured by the left colour camera

#Values	Name	Description
1	type	Type of object: Car, Van, Truck, Cyclist, Tram, Pedestrian, Person_sitting, Misc or DontCare
1	truncated	Float from 0 (non-truncated) to 1, indicating the extent of the object out of the image
1	occluded	Integer indicating occlusion state: 0 = fully visible, 1 = partly occluded, 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-pi..pi]
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates (x_1, y_1, x_2, y_2)
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x,y,z in camera coordinates (in meters)
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]

Table 2: Label specification of KITTI dataset

4.1.2 3D Vehicle Dataset

This dataset is intended to encode the diversity of vehicles by means of type, dimension, and chassis shape. It is created based on a selected subset of synthetic 3D CAD models[34]. We only include the well-aligned and symmetrical models, and also take the dimension and type into consideration. We classify all the models into six categories: Mini, Hatchback, Sedan, SUV, Wagon, and Van. And for each category, subcategories are marked out according to their dimensions and shapes in order to make a refined selection. The statistic of dimensions is well distributed and each existing car can be classified into one category exclusively.

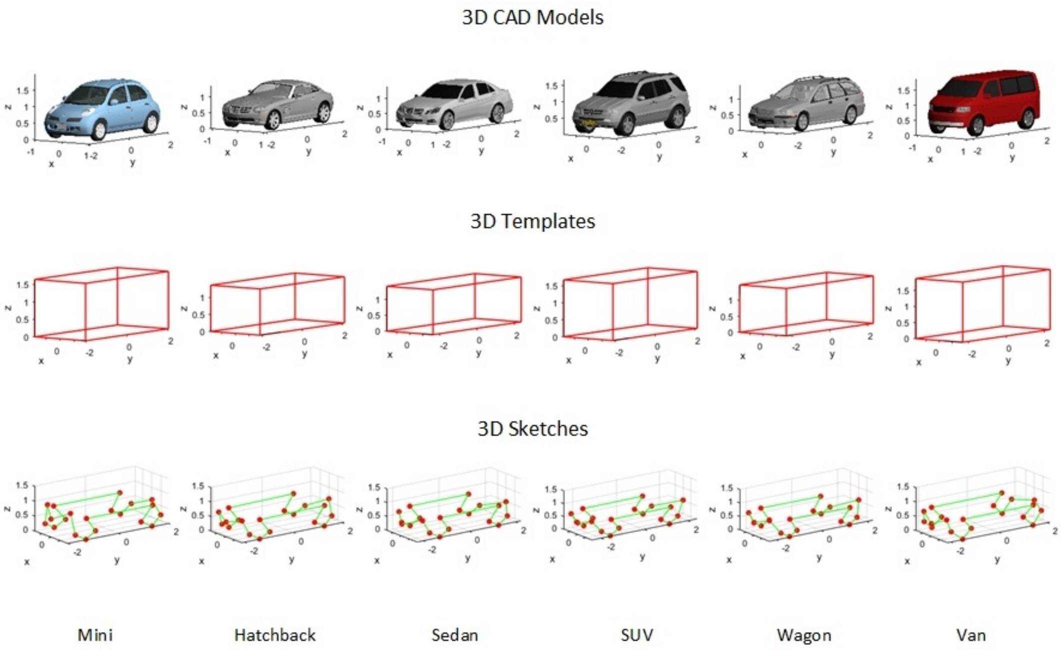


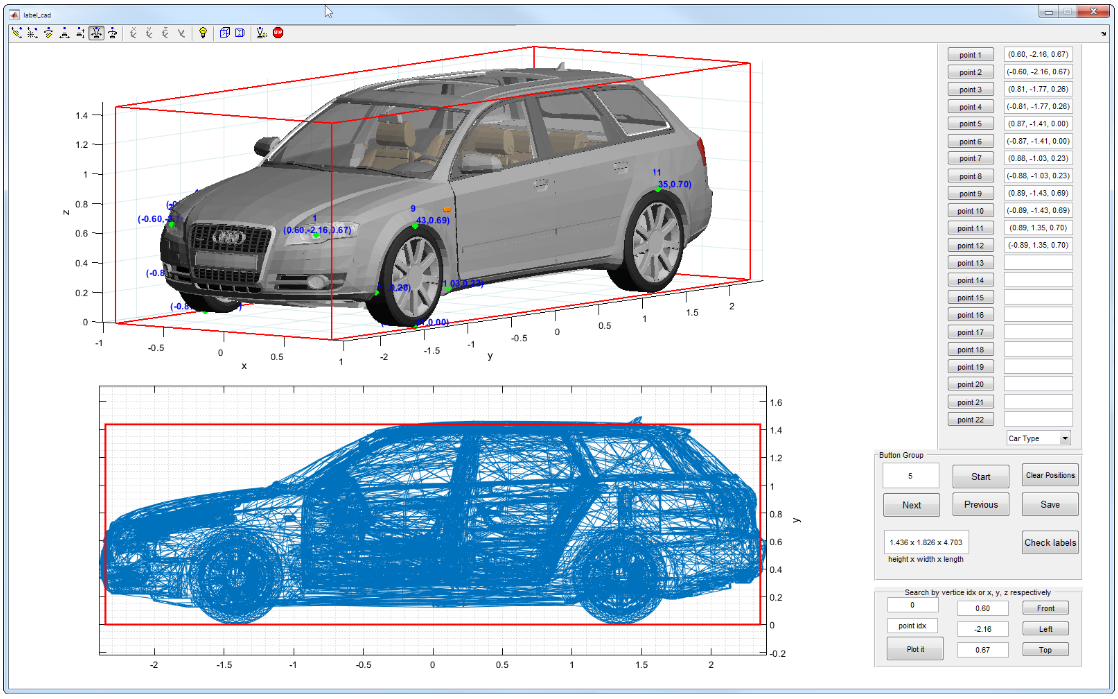
Figure 4: Six vehicle categories of the 3D vehicle dataset. Each vehicle model is associated to a 3D CAD model, a 3D template, and a 3D sketch.

Our final dataset consists of 54 valid vehicle models. Each vehicle model has a 3D CAD model, a 3D template, and a 3D sketch correspondingly, as shown in Figure 4. All these three representation are aligned in canonical view and share an identical object coordinate system. The CAD models are created based on real vehicles and have all the geometry information with them. The 3D templates represent the vehicles' dimensions. The 3D template associated to the 3D model k is denoted as $t_k = (h_k, w_k, l_k)$ where h_k, w_k and l_k represent the height, width, and length of the model respectively.

The 3D sketches indicate the chassis shapes. Each 3D sketch consists of 20 characteristic points around the chassis with each point denoting one part of the vehicle. So the k^{th} sketch is denoted as $S_k^{3d} = (p_1, p_2, ...p_{20})$, where $p_i = (x_i, y_i, z_i)$. The reason why we choose the points around chassis as feature points is that their geometric relationship is more stable than points in other places. Because the chassis part is more about functionality than appearance attraction compared to other parts of the vehicle, *e.g.* the upper part. In this way, the sketches can be more universal so that they can represent a wider range of vehicles, which reduces the number of models required and further increases the computational efficiency.

In order to create the 3D sketches, we implement a labelling tool, shown in Figure 5. It is capable to label the 3D coordinates for all the characteristic points associated to each vehicle.[should it be elaborated in detail]

TBD



point 1	(0.60, -2.16, 0.67)
point 2	(-0.60, -2.16, 0.67)
point 3	(0.81, -1.77, 0.26)
point 4	(-0.81, -1.77, 0.26)
point 5	(0.87, -1.41, 0.00)
point 6	(-0.87, -1.41, 0.00)
point 7	(0.88, -1.03, 0.23)
point 8	(-0.88, -1.03, 0.23)
point 9	(0.89, -1.43, 0.69)
point 10	(-0.89, -1.43, 0.69)
point 11	(0.89, 1.35, 0.70)
point 12	(-0.89, 1.35, 0.70)
point 13	
point 14	
point 15	
point 16	
point 17	
point 18	
point 19	
point 20	
point 21	
point 22	

5

1.436 x 1.826 x 4.703
height x width x length

0
point idx
0.60
-2.16
0.67

Figure 5: Labelling tool for 3D sketches

4.1.3 Labelling for KITTI Dataset

Our approach requires three extra kinds of labels to training the CVT Network, *i.e.*, 2D coordinates and visibility property of characteristic points and template proximity, as shown in Table 3. Labelling is never a trivial task. Due to the demanding workload of manual labelling and the cases where it is almost impossible to label the small vehicles in image manually, we propose an automatic label generation method. It is able to make use of the 3D vehicle dataset and the KITTI dataset to generate these three additional kinds of ground truth.

#Values	Name	Description
2x20	2D coordinate	(x, y), location of 20 characteristic points in image coordinates
1x20	visibility	Integer indicating visibility property of each point: 0 = visible, 1 = occluded, 2 = self-occluded, 3 = truncated
3x54	template proximity	A vector T_i represents the dimension ratios between each model and the vehicle

Table 3: Specification of three additional labels

4.1.3.1 2D Coordinates

2D coordinates of key points of each vehicle in image coordinates is generated by projecting the 3D sketch of this vehicle to the image. The vehicle's 3D sketch is selected via template-matching, *i.e.*, the best-matching sketch is the one whose associated template is closest to the vehicle's dimensions. The 3D-2D projection is performed based on the given intrinsic and extrinsic parameters, as shown in Figure 6. The intrinsic parameters are given as calibration by KITTI and the extrinsic parameters are given as 3D object dimensions and rotation ry in KITTI labels. KITTI makes a simplified assumption here that the vehicle only rotates around the yaw axis but not roll or pitch axis. One example of the labelled 2D coordinates is shown in Figure 7.

4.1.3.2 Visibility

As an example of the labelled visibility shows in Figure 7, the visibility property of characteristic points is classified into four scenarios:

i. Visible if the point can be seen directly;
ii. Occluded if the point is occluded by other objects;
iii. Self-occluded if the point is blocked out by the vehicle itself;
iv. Truncated if the point exceeds the boundaries of the image.

The visibility of each point is determined by its position, *i.e.*, the 2D coordinate in image. Being visible or self-occluded is distinguished by means of rotation ry of each vehicle. As the right plan sketch in Figure 8 shows, the rotation ry can

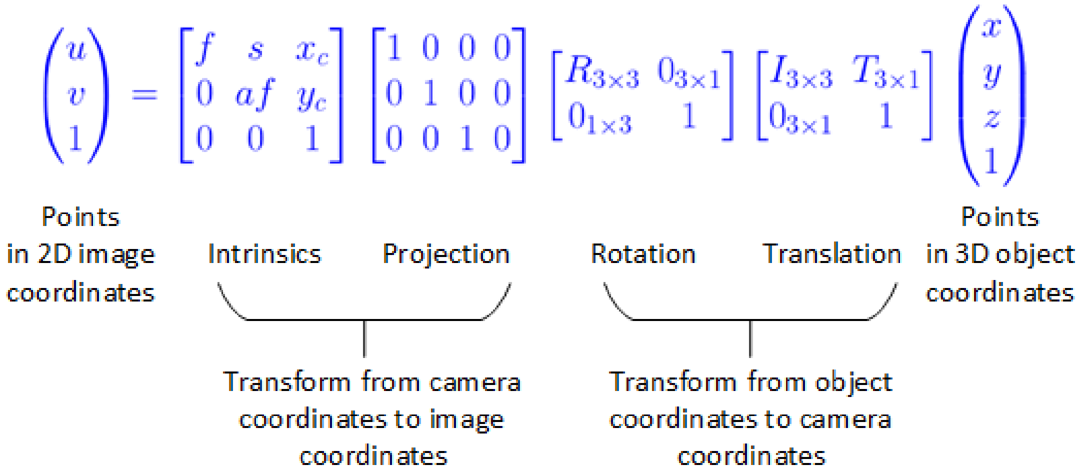

Points in 2D image coordinates
Intrinsics
Projection
Rotation
Translation
Points in 3D object coordinates
Transform from camera coordinates to image coordinates
Transform from object coordinates to camera coordinates

Figure 6: 3D-2D projection

indicate unambiguously which faces of the vehicle are facing to or away from the camera. If the points are on the observable faces, they are labelled as visible, otherwise they are classified as self-occluded. Occluded case happens when the 2D coordinate of the point falls into the 2D bounding box of a former object. The object is defined as former when it locates in the region enclosed by the axes and two solid blue line in the left schematic plot of Figure 8 if the vehicle is on the first quartile. And when the vehicle is on the second quartile, it's just a mirrored case. Truncated is defined that its 2D coordinate exceeds the boundaries of the image. The size of KITTI images are determined which makes it easy to implement. The truncated property has the highest priority, the occluded underlies , the self-occluded or the visible is considered at last.

4.1.3.3 Template Proximity

Template proximity of one vehicle is encoded as a vector T_i which is defined as $T_i = \{r_k\}_{k \in \{1,..,K\}}$, where K denotes the number of 3D vehicle models and $r_k = (r_h, r_w, r_l)$ corresponds to three scaling ratios between the dimensions (*i.e.*, height, width, and length) of each model and the vehicle's respectively. The vector T_i represents the similarity between each model and the vehicle. The most similar model is the one whose r_k is closest to $(1, 1, 1)$.

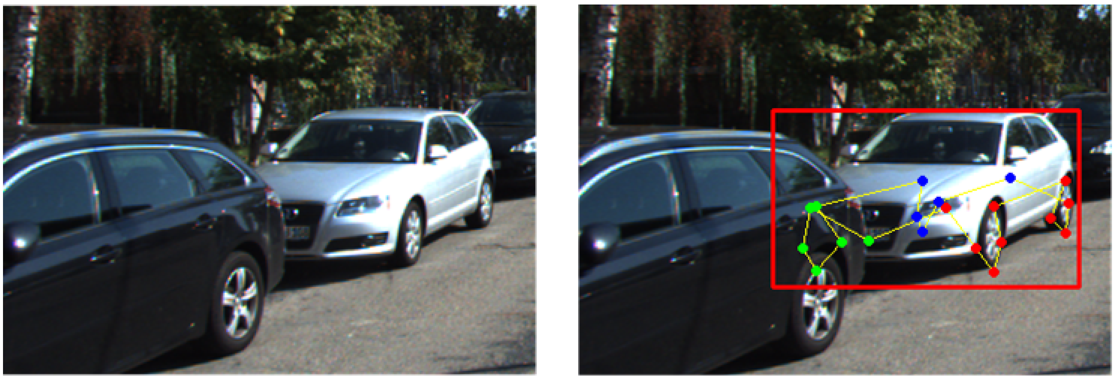


Figure 7: Example of 2D coordinates and visibility of one vehicle. The left image is one patch of KITTI image. The left image is the one after labelling. The points indicate the 2D coordinates and the color indicates the types of visibility: red for visible, green for occluded, and blue for self-occluded.

4.1.4 Notations

In sum, based on the KITTI image and 3D vehicle models, each vehicle can be defined by seven critical attributes:

$$\{D, B^{2d},\ B^{3d},\ C^{2d},\ C^{3d},\ V,\ T\}$$

$D = (h, w, l)$ represents the dimensions of the vehicle. $B^{2d} = (c_u, c_v, w, h)$ defines the 2D bounding box in image with (c_u, c_v) denoting its center, w for its width and h for its height. $B^{3d} = (o, \theta, d)$ where $o = (c_x, c_y, c_z)$ is the center, θ is the rotation ry around the yaw axis, and $d = (w, h, l)$ is its dimensions. $C^{2d} = \{(u_i, v_i)\}_{i\in\{1,\ldots,20\}}$ represents the 2D part coordinates in image plane, while $C^{3d} = \{(x_i, y_i, z_i)\}_{i\in\{1,\ldots,20\}}$ denotes the 3D part coordinates in world coordinate system. $V = \{v_i\}_{i\in\{1,\ldots,20\}}$ is the visibility for all the characteristic points in the vehicle. $T = \{(r_h, r_w, r_l)_k\}_{k\in\{1,\ldots,54\}}$ is the template proximity vector.

4.2 Deep CVT Network

This subsection describes the details of the Deep CVT Network, including its architecture, implementation, and training.

4.2.1 Architecture

The neural network is to learn a map from an N-dimensional input space to an M-dimension output space. The map consists of several stages, called layers of the

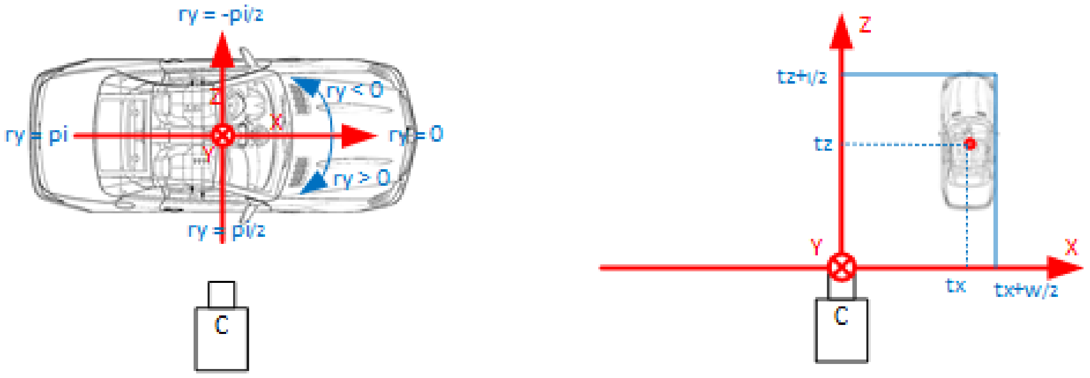


Figure 8: Visibility labelling mechanism. The left image shows the rotation ry in the camera coordinate system, which helps distinguish visible or self-occluded case. The left image shows the location of the vehicle in the camera coordinate system, which helps defined the occluded situation.

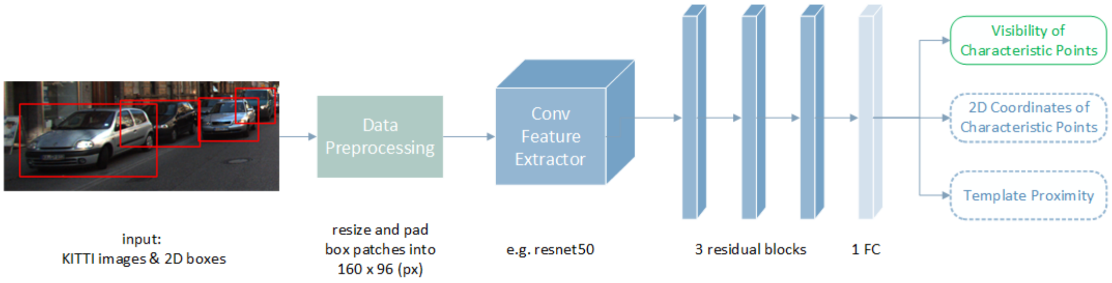


Figure 9: The Architecture of Deep CVT Network

network, written as

$$Y = l_L(...l_2(l_1(X))), where\ Y \in \mathbb{R}^M,\ \ X \in \mathbb{R}^N \tag{6}$$

Even through there are various layers, most layers is composed of neurons, the basic computation element and most neurons are realized by linear and non-linear operations as

$$a_{ij} = \sigma(W_{ij}^T X_i + b_j) \tag{7}$$

where a_{ij} is the result of the i_th neuron in the j_th layer, σ is the activation function and W_ij and b_j are the weights vector and bias applied to input vector X_i. The pooling layer is a special case which performs downsampling operations, such as max pooling [18] and average pooling [89].

As shown in Figure 9, our network follows the standard CNN architecture established in LeNet-5 [55]. It stacks a set of convolutional layers and pooling layers as a feature extractor, one fully-connected layers followed, and three output layers in the end. The goal of this network is to learn the 2D coordinates and visibility property of 20 interest points and the template proximity of the vehicle, given the RGB images captured by a monocular camera.

TBD

4.2.1.1 Input Layer

The input layer accepts the input RGB images and feeds them into the network. In theory, images with arbitrary size can be accepted, but to make it more efficient, we use images with fixed resolution (96 x 160). In this way, multiple images can be processed in one batch in order to reduce the variance of parameter updates and make the best use of highly optimized matrix optimizations during training [74].

Since our approach is based on 2D object detection, each image contains at least one whole vehicle . To obtain these images, we first crop the patches defined by the 2D bounding box, then resize them to match one predefined dimension (96 or 160), and finally put the resized images in the center of the 96 x 160 canvas and padded with zeros for other pixel positions. We choose 96 x 160 as the fixed size because they are the means of sizes of all original patches so that we don't have to rescale the patch too much. Some examples are shown in Figure 10.

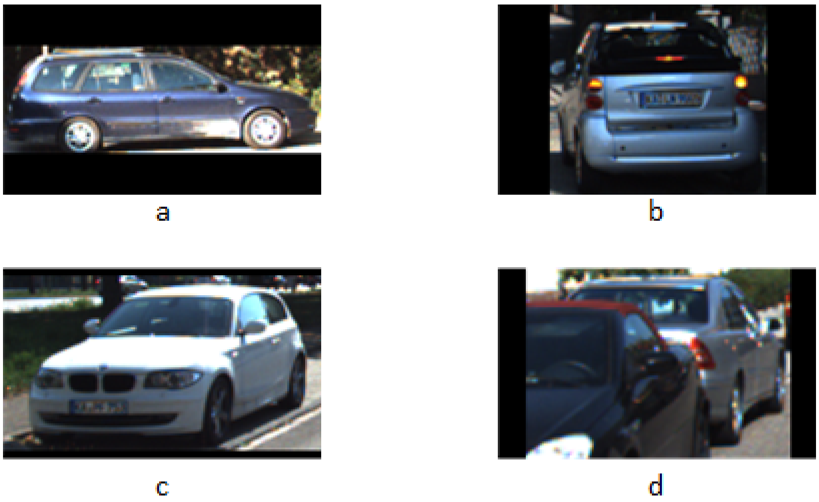


Figure 10: Examples of input images: a. the side view, b. the back view, c. the front view, d. the occluded case

4.2.1.2 Hidden Layers

The hidden layers in out network consist of a feature extractor, 3 residual blocks [44], and a fully-connected layer. Both the feature extractor and residual blocks are composed of convolutional layers and pooling layers.

The feature extractor is actually one of available neural network models built in the Keras library [27], *i.e.*, VGG16, VGG19, Xception, ResNet50, *etc.*It is trained to learn deep distributed hierarchical nonlinear representations of the input images.

ResNet50 [44] is chosen to be the benchmark model because it can ease the training and accelerate the convergence, especially for the fine tuning. It is well known that there are two main obstacles that hamper the training of deep neural networks: the vanishing/exploding gradient problem [14, 39] and the degradation problem that as the network is built deeper, the accuracy gets saturated or even degrades sharply [43]. Residual nets are designed to address these two problems. Instead of learning the direct mapping from input to output, it learns a residual mapping first and then add the input to it, which just as Figure 11 (a) shows. The paper validates that it is easier to learn the residual mapping than the direct one and the gradient can always pass through along the shortcut connections. Figure 11 (b) and (c) shows two building blocks in our network. (b) is an identity block where the dimensions of input and output matches, while (c) is a linear projection block where the right branch performs 1 × 1 convolution to match the dimensions because the final addition is performed element-wisely.

The last three residual blocks and a fully-connected layer is designed to learn nonlinear combinations of the extracted features and then forward them to the output layers.

For all hidden layers, we all apply ReLU as the activation function. The function is

$$f(z) = \max\{0, z\} \quad where\ z = W^T X + b \tag{8}$$

This is a piecewise linear function consisting of two linear pieces, which makes the gradients through a rectified linear unit stay large and consistent whenever the unit is active. Papers [40, 64, 48] have validated that networks activated by ReLU can achieve much better performance than others. ReLU preserves many properties that make the model easy to optimize with gradient-based methods and generalize well, while introducing non-linearity into the model [41].

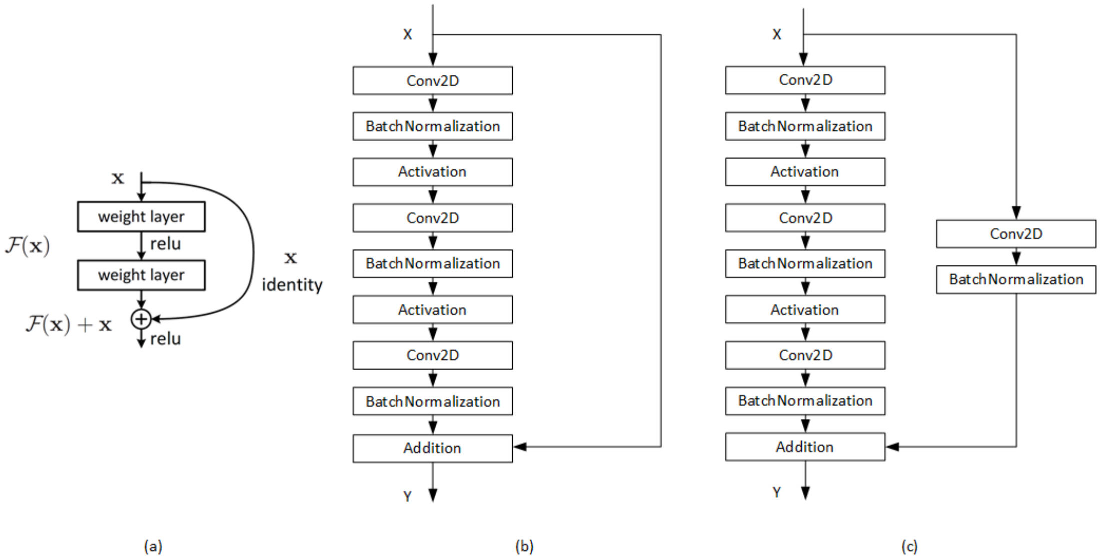


Figure 11: (a). a residual block [44], (b). an identity residual block in our network, (c). a linear projection residual block in our network

4.2.1.3 Output Layers

Our network has three kinds of output layer for the three tasks respectively. For 2D coordinates regression, a 40-neuron layer with linear activation function is provided. Each neuron's output indicates one coordinate value, *i.e.*, x_i or y_i. 20 coordinates are arranged in an ascending order. For visibility characterization, we implement 20 independent quaternary classifiers for 20 points. Each classifier is associated with a softmax activation function [17] which outputs four values representing the probabilities of each target class over all possible classes. Template proximity is equipped with a 154-neuron linear layer for regression. Each neuron indicates one ratio element of the template proximity vector. Eq.9 is a linear activation function while Eq.10 is a softmax activation function.

$$out_i = W_i^T X_i + b_i \tag{9}$$

$$out_i = \frac{e^{W_i^T X_i + b_i}}{\sum_{k=1}^{K} e^{W_k^T X_k + b_k}} \tag{10}$$

4.2.2 Training

4.2.2.1 Multi-task Learning

As mentioned in Section 4.2.1.3, our network has totally 22 output layers. This implementation makes use of multi-task learning (MTL) where multiple learning tasks are trained parallel based on the shared features. Caruana has validated that tasks trained in MTL have better generalization performance than trained in a single-task learning mode [20]. And he summarizes that this improvement results from leveraging the domain-specific information contained in the training signals of other related tasks [20]. Therefore, we follow this paradigm to design our network in order to achieve better performance.

4.2.2.2 Loss Functions

The learning of 2D coordinates and template proximity are regression tasks so that a robust smooth L_1 loss function [37] is chosen for them. We modify it as

$$smooth_{L_1}(x) = \begin{cases} 0.5x^2 & if \ \ |x| < 0.5 \\ |x| - 0.25 & otherwise \end{cases} \quad where\ x = \widehat{y} - y \tag{11}$$

Compared to L_2 loss, it is less sensitive to outliers and no special attention is required to pay in order to prevent gradient exploding problem[37]. Besides, when applied with gradient-based optimization, L_1 loss and L_2 loss often result in poor performance [41].

For visibility characterization, we develop a model for probabilistic classification so that categorical cross-entropy is used as the loss function. It is also known as the negative log-likelihood [41], defined as:

$$L_{cce} = \frac{1}{N}\sum_{n=1}^{N} H(y, \widehat{y}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{c} y_{n,i} \log \widehat{y}_{n,i} \tag{12}$$

where $H(y, \widehat{y})$ denotes the cross-entropy between the ground truth probability distribution y and the predicted $\widehat{y}$, and $y_{n,i}$ represents the true probability of i_{th} class for the n_{th} data example $\widehat{y}_{n,i}$ is the estimated. It is a continuous convex loss function which measures the discrepancy between the true and estimated distributions multi-class classification tasks [12] which means it can measure the degree of correctness, *i.e.*, it can distinguish between "nearly correct" and "totally wrong" cases. Therefore, it outperforms other loss functions in classification tasks and then becomes ubiquitous in deep learning nowadays.

Therefore the total loss for all tasks is

$$L_{total} = \lambda_{coord} L_{coord} + \lambda_{temp} L_{temp} + \lambda_{visib} L_{visib} \tag{13}$$

where λ_{coord}, λ_{temp}, and λ_{visib} are loss weights for these three kinds of tasks respectively. Loss with higher weights has more impact on the gradients and thus tunes the parameters perform better for its corresponding task.

4.2.2.3 Normalization

Normalization is an important pre-processing step in deep learning which ensures that each feature has a similar data distribution. This is usually done by restricting the features in certain range or standardizing their ranges. It can enhance the learning capability of the network and speed up the convergence substantially because it can reduce the bias among features and decrease the low and high frequency noisy in data [49]. The speed-up mechanism can be concluded from Figure 12. If we initialize the network at point A in Figure 12 (a), the gradient-descent update routine will oscillate along the long axis of the eclipse, which definitely takes more time to reach the optimum than any arbitrary initial point in Figure 12 (b) where the update trajectory is almost a straight line for any starting point to the minimum. Besides, this also helps the performance because the update trajectory oscillates less around the minimum for case (b) than case (a) so that it is more possible for case (b) to reach the optimum.

Our network takes RGB images as input so that pixel intensity is the input feature. The technique we use to normalize the image is channel mean subtraction, used in [80, 38], which centers all the features around the origin along each dimension.As mentioned in CS231N [1], mean channel subtraction is enough for CNNs and the original range of pixel values is determined, *i.e.*, $[0, 255]$.

The range of ground truth influences the loss. In order to make the three labels impact similarly on the loss, we normalize them. For visibility, we use one-hot-encoding [4] to encode its classes so that it only has value 0 or 1. The value of 2D coordinates $C^{2d} = \{p_1, p_2, ..., p_{20}\}$varies a lot so that we normalize them w.r.t. the associated 2D bounding box $B^{2d} = (c_u, c_v, w, h)$ [22]. The normalized 2D coordinates $\overline{C}^{2d} = \{\overline{p}_1, \overline{p}_2, ..., \overline{p}_{20}\}$ ranges $[-1, 1]$, where

$$\overline{p}_i = (\frac{u_i - c_u}{w}, \frac{v_i - c_v}{h}) \tag{14}$$

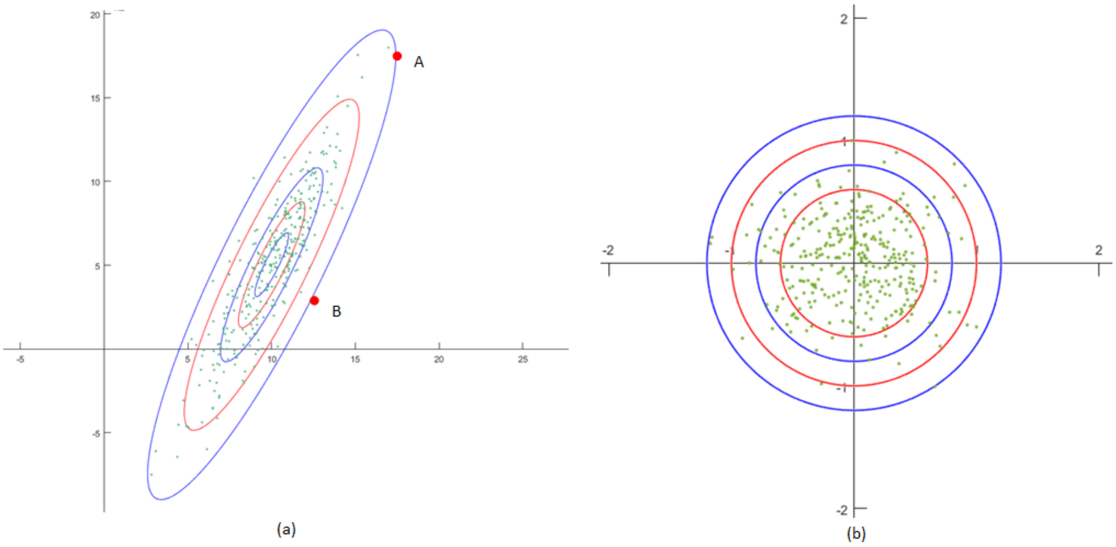


Figure 12: (a). data distribution before normalization, (b). data distribution after normalization

The template proximity vector T is normalized with log function element-wisely [22], resulting in $\overline{T}$ so that the range falls into $[-1, 1]$, too.

Moreover, as Sergey *et al.* [47]states the variation of each layer's input distribution during the training makes the training complicated and hard to converge quickly so that we apply Batch Normalization (BN) to alleviate the internal covariate shift phenomenon. BN normalizes the summed activations of each layer to a distribution of zero mean and unit variance. The mean and variance of each activation is computed on each mini-batch. By doing so, much larger learning rate can be used for training and no special attention has to pay on parameter initialization.

4.2.2.4 Regularization

Regularization techniques are used to address the overfitting problem in order to make an algorithm not only perform well on the training data but also on the test data, the previously unseen data [41]. Overfitting results from either that the algorithm is too complicated for the data or that the sampled data is not able to represent the internal pattern. Normally, we design an algorithm to model the data pattern exhaustively first and then apply regularization techniques to make the algorithm generalize well.

In the data aspect, we use data augmentation as a regularization. The best way to address overfitting is to train the model on more data, while the amount of data is restricted for one dataset. Therefore we generate some synthetic data. Since our network involves with classification and regression tasks, we mainly enlarge the dataset via flipping, scaling, and cropping .

TBD

In the architecture aspect, Batch Normalization [47] is used as the regularization in each layer. BN enables the network to obtain the information of the training sample and the others in mini-batch simultaneously so that it does not generate deterministic dependency on this training sample. Therefore BN can replace Dropout [84] to be the regularization for our convolutional network.

Besides, Multitask Learning is another technique we used to improve generalization performance. In our network, the learning representation is shared across all tasks so that the parameters shared are constrained to bias towards one specific task, *i.e.*, only the representation that is useful for more than one can be kept [41]. Therefore statistical strength of the parameters is highly enhanced [11].

During the training, early stopping is applied to prevent the model being trained too complex to fit the test data. As Figure 13 shows, we halt the training when the validation error stops decreasing. The stopping point is when the learned model can represent the pattern of validation data most. It regards the number of training epochs as a hyperparameter and can effectively tune it to be optimal [15], because early stopping can restrict the global learning capacity of a large network to fit simpler dataset without affecting the backpropagation to control the learning capacity locally [21].

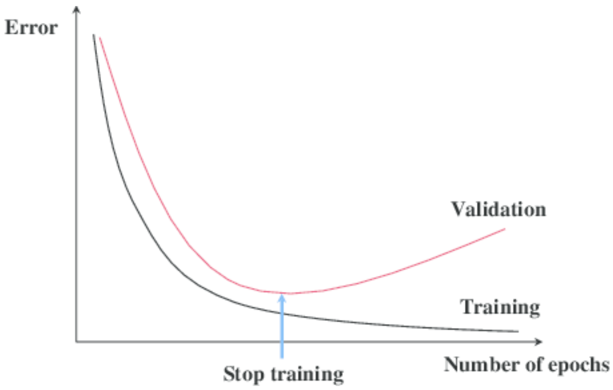


Figure 13: Early stopping

4.2.2.5 Parameter Initialization

Transfer learning is proposed to transfer the representation learned from one task to another related task [67]. As is known that CNNs learn more abstract features in a deeper layer. It is not surprising to find that, for visual tasks, shallow layers learn low-level features, *e.g.* edges, corners, changes in lighting, *etc.*, which are shared across datasets and tasks. Moreover, Yosinski *et al.* validate [95] that initializing a network with transferred features can improve the generalization capability hugely and Yoshua *et al.* [16] confirm that this initialization put the start point closer to a local minimum than random initializations, resulting in accelerating convergence.

Therefore we initialize our feature extractor with parameters learned on ImageNet dataset [76] and the three residual blocks with parameters trained on the KITTI dataset. The fully-connected layer is initialized from a zero-mean Gaussian distribution with standard deviation 0.01 and all the output layers are initialized with zeros.

4.2.2.6 Learning Algorithms / Optimization

Optimization is a task of finding the value x in order to minimize or maximize some objective function $f(x)$. One most powerful optimization technique category in deep learning is gradient-based.

As is known, the derivative $f'(x) = \frac{\mathrm{d}y}{\mathrm{d}x}$ specifies how to make a small change ϵ of the input x to get the corresponding change in the function:

$$f(x+\epsilon) \;\approx\; f(x) + \epsilon f'(x). \tag{15}$$

Thus, the derivative tells the direction to minimize a function, i.e. it can point out how to change x to make a small update. One popular algorithm, gradient descent [9], makes use of the derivatives and update the input x directly as:

$$x = x - \alpha \nabla_x f(x) \tag{16}$$

where α is the learning rate, a positive real value to decide the size of an update step. The examples in Figure 14 (a) show how the update works. A deep network often consists of many layers so that back-propagation algorithm [75] is used to compute the gradient for each parameter in different layers of the objective function based on the chain rule of calculus.

In deep learning, the input to the objective function is often multidimensional so that it probably has many local minima and saddle points, which renders huge difficulties to optimization. Therefore we usually take in a compromise scenario where the value x makes f really low but not necessarily globally minimal [41].

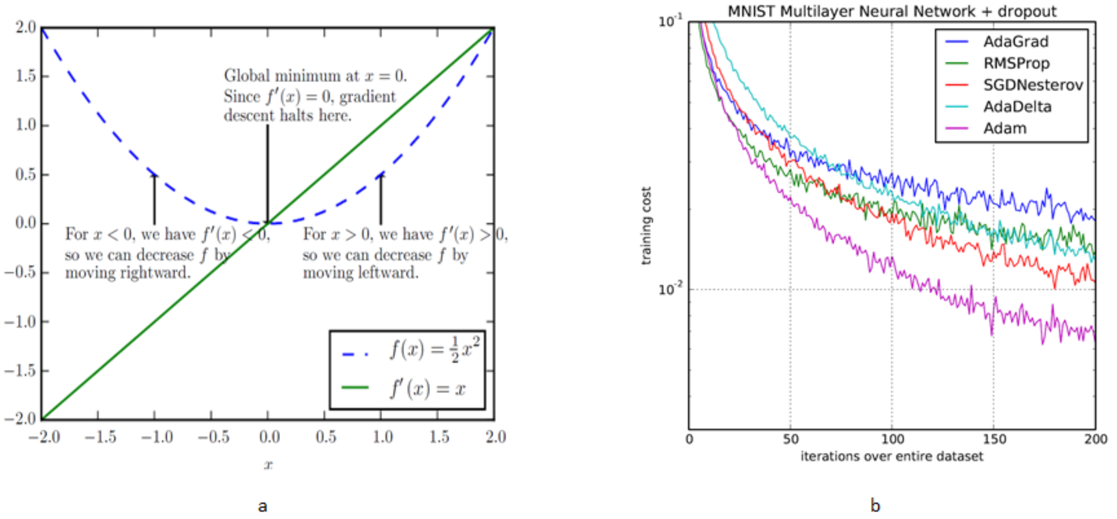


Figure 14: (a). examples to show how gradient decent makes use of derivatives to reach a minimum [41], (b). performance of optimization algorithms in the same setting [53]

The optimization algorithm we used is Adam [53] which is robust and efficient in memory and computation for the optimization of stochastic objectives with high-dimensional parameters spaces. It makes use of both the gradient and its momentum to update parameters as:

```
while $x_t$ not converged do
    $t = t + 1$;
    $g_t = \nabla_x f(x_{t-1})$;
    $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$ ;
    $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$;
    $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$;
    $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$;
    $x_t = x_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$;
end
```

where t denotes the current iteration, α is the learning rate, β_1 and β_2 are two exponential decay rates, and ϵ is a small scalar to prevent zero-division.

From the Figure 14 (b), we can see that Adam can make the learning task converge relatively faster and has lower training error so that we follow this guide to apply Adam in our optimization of CVT Network.

4.3 Inference

As shown in Figure 15, he inference phase consists of two main steps: template matching to obtain the 3D coordinates of characteristic points and 3D dimensions for the objective vehicle and 2D-3D matching to recover the 3D vehicle location and rotation.

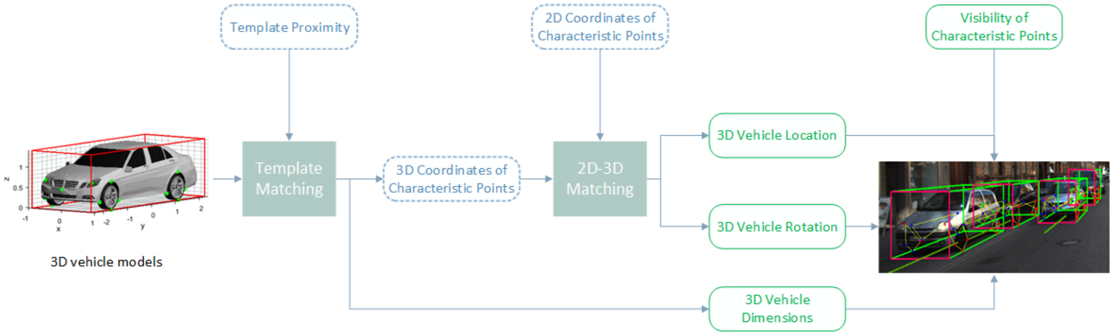


Figure 15: The Architecture of our Approach

4.3.1 Template Matching

Template matching is based on the 3D template dataset and the learned template proximity for the objective vehicle in image. As defined in section 4.1.3.3, template proximity vector, $T = \{(r_h, r_w, r_l)_k\}_{k\in\{1,..,K\}}$, measures the dimension similarity between the target vehicle and 3D vehicle models. The best matching model is the one whose dimensions (h, w, l) has the least distance to the target vehicle's dimension $(\bar{h}, \bar{w}, \bar{l})$, *i.e.*, the corresponding scaling ratios, (r_h, r_w, r_l), is closest to $(1, 1, 1)$.

Let's denote the best matching template for the target vehicle m as t_j and the dimension ratio between the target vehicle and the best matching model as $r_j = (r_h, r_w, r_l)$. Then its corresponding 3D sketch is $S_j^{3d} = (p_1, p_2, ...p_{20})$. To get the target vehicle's dimension D_m, we apply the scaling ratios, (r_h, r_w, r_l), to t_i as

$$D_m = t_j \cdot r_j = (h_j, w_j, l_j) \cdot (r_h, r_w, r_l) = (h^m, w^m, l^m) \tag{17}$$

In the same way, we can get the 3D coordinates of the interest points of the objective vehicle,C_m^{3d}, as

$$\begin{aligned} C_m^{3d} &= S_j^{3d} \cdot r_j \\ &= \{(x_i, y_i, z_i)\}_{i\in\{1,...,20\}} \cdot (r_h, r_w, r_l) \\ &= \{(x_i^m, y_i^m, z_i^m)\}_{i\in\{1,...,20\}} \end{aligned} \tag{18}$$

4.3.2 2D-3D Matching

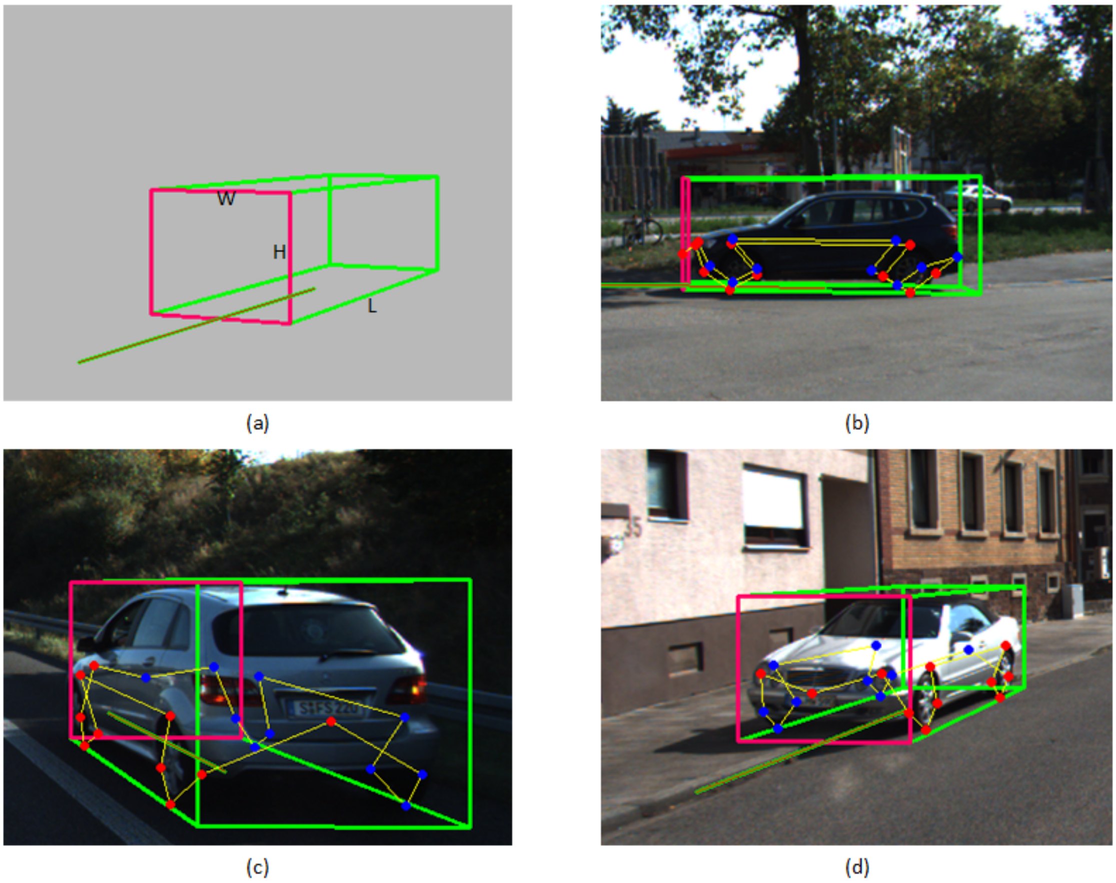
Based on the projection mechanism described in section 3.3.2, the 2D coordinate of one point in image coordinate system can be generated via projecting the 3D point in the world coordinate system with the perspective projection matrix to the image plane. This process is described in Figure 6. Now the CVT network predicts the 2D coordinates,C_m^{2d} of the 20 interest points of the target vehicle m and the template matching process provides the corresponding 3D coordinates, C_m^{3d}. Then the 3D-2D projection equation in homogeneous coordinates becomes:

$$C_m^{2d} = K_{3\times4} \begin{pmatrix} R_{3\times3} & t_{3\times1} \\ 0_{1\times3} & 1 \end{pmatrix} C_m^{3d} \tag{19}$$

where $K_{3\times4}$ is the given camera calibration matrix, $R_{3\times3}$ and $t_{3\times1}$ are the rotation and translation to be computed. It is easy to compute the rotation and translation of the target vehicle in the camera coordinate system with the standard 2D-3D matching algorithm, EPnP [56]. The KITTI data simplifies the rotation to one dimension, r_y, so that we follow this convention. Translation is the 3D coordinate of the origin of the vehicle so that it can represent the location of the vehicle in camera coordinate system.

4.3.3 Visualization

To visualize the 3D bounding box, we use the vehicle's dimensions to determine the eight corners of the cuboid and then project this cuboid into the image with the projection matrix which composes of the recovered rotation and translation. And in order to show the rotation clearly, project the line on the ground to show the direction of the vehicle. This visualization mechanism is shown in Figure 16 (a) where the front face of vehicle is coloured with magenta. Figure 16 also shows three typical examples in side view, back view, and front view.



(a)

(b)

(c)

(d)

Figure 16: Visualization of 3D bounding box and visibility. (a). visualization mechanism, (b). side view of visualization, (c). back view of visualization, (d). front view of visualization

5 Experiment and Evaluation

In this section, we describe the experiment setup, discuss the most crucial design choices for our approach, evaluate our approach and its variations , and compare our approach with the start-of-the-art methods on monocular pose estimation of vehicles.

template Experiments are divided into three parts1. First we compare with state-of-the-art methods for 3D object detection on KITTI [10] and SUN-RGBD [33] (Sec 5.1). Second, we provide in-depth analysis to validate our design choices (Sec 5.2). Last, we show qualitative results and discuss the strengths and limitations of our methods (Sec 5.3).

TBD

5.1 Experiment Setup

We have presented the approach in Section 4.2 and 4.3 which is served as a benchmark of our approach. Some other design choices are made based on it. The benchmark network use ResNet50 [44] as feature extractor, initialized with pre-trained weights trained on ImageNet [76], and is trained with Adam [53]. The network is implemented on Keras [27] using TensorFlow [8] as backend. Keras supports running both on GPU and CPU.

We evaluate our approach and its variations on the dataset created based on KITTI 3D object detection benchmark [36], described in Section 4.1. Because the KITTI only releases the ground truth for 7481 training images, we split them into train and validation set for training and validation respectively. We follow difficulty division policy of KITTI and extend to more detailed levels . We use 54 vehicle CAD models [34] for semi-automatic labelling and template matching. Each model is encoded with 20 points for its corresponding 3D sketch.

TBD

We evaluate six tasks: 3D vehicle detection, orientation estimation, 3D localization, 3D dimension estimation, 2D part localization, and 2D part visibility prediction. 3D vehicle detection is the ultimate task, representing the localization, orientation, and dimension of 3D bounding box, which is therefore used to evaluated all the design choices.

5.2 Evaluation Metrics

We use intersection over union (IoU) to measure the performance of 3D vehicle detection. IoU used to measure the similarity of two 2D bounding boxes in various

2D object detection challenges, *e.g.* Pascal VOC [33] and ILSVRC [76]. Recently it is extended to measure 3D object detection with the formula:

$$IoU(b_1, b_2) = \frac{V(b_{pre} \cap b_{gt})}{V(b_{pre} \cup b_{gt})} \tag{20}$$

where $V(\cdot)$ indicates the volume and b_{pre} denotes the predicted 3D bounding box while b_{gt} is the ground truth. KITTI judges a 3D object detection is correct, if $IoU \geq 0.7$. But in our experiment, we lower the criterion to $IoU \geq 0.5$. If multiple bounding boxes are predicted for one vehicle, they are considered as false predictions.

For orientation estimation, we consider the estimation correct if its difference from the ground truth is less than 0.2 rad. For other tasks, We follow the metrics set by [22]. A 3D localization is considered correct if its distance to the ground truth is less than a threshold. Two thresholds, 1 meter and 2 meters, are chosen. 2D part localization are measured the same way and the threshold is 20 pixels. 3D dimension estimation is correct if the predicted dimensions (h, w, l) satisfies the equation

$$\left|\frac{h - h_{gt}}{h_{gt}}\right| < 0.2 \;\; \& \;\; \left|\frac{w - w_{gt}}{w_{gt}}\right| < 0.2 \;\; \& \;\; \left|\frac{l - l_{gt}}{l_{gt}}\right| < 0.2 \tag{21}$$

where (h_{gt}, w_{gt}, l_{gt}) is the ground truth. 2D part visibility prediction is a pure classification problem so that the measure is the accuracy over 4 classes.

Mean distance error (MDE) TBD

Task	Metric
3D vehicle detection	$IoU \geq 0.5$
Orientation estimation	$\lvert r_{pre} - r_{gt} \rvert < 0.2$ rad
3D localization	$\lVert \bar{\mathbf{t}}_{\mathbf{pre}} - \bar{\mathbf{t}}_{\mathbf{gt}} \rVert < 1/2$ meters
3D dimension estimation	$\left\lvert \frac{d - d_{gt}}{d_{gt}} \right\rvert_{d=\{h,w,l\}} < 20\%$
2D part localization	$\lVert \bar{\mathbf{p}}_{\mathbf{pre}} - \bar{\mathbf{p}}_{\mathbf{gt}} \rVert < 20$ pixels
2D part visibility	$V_{pre} = V_{gt}$

Table 4: Metrics for six tasks

6 Conclusion and Discussion

6.1 Discussion

6.1.1 Deficiency of Our Approach

6.1.2 Deficiency of Labelling

6.2 Conclusion

References

[1] Data preprocessing.

[2] History of autonomous cars. `https://en.wikipedia.org/wiki/History_of_autonomous_cars#cite_note-106l`.

[3] No hands across america home page. `http://www.cs.cmu.edu/afs/cs/usr/tjochem/www/nhaa/nhaa_home_page.html`.

[4] One-hot-encoding.

[5] What is computer vision.

[6] Phantom auto will tour city. *The Milwaukee Sentinel. Google News Archive*, Dec 1926.

[7] *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*, Jan 2014.

[8] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[9] Cauchy Augustin-Louis. Méthode générale pour la résolution des systèmes d'équations simultanées. *Compte rendu des séances de l'académie des sciences*, abs/1411.1792:536–538, 1847.

[10] Dana H. Ballard, Geoffrey E. Hinton, and Terrence J. Sejnowski. Parallel visual computation. *Nature*, 306:21–26, 1983.

[11] Jonathan Baxter. Learning internal representations. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, COLT '95, pages 311–320, New York, NY, USA, 1995. ACM.

[12] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *CoRR*, abs/1802.09941, 2018.

[13] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013.

[14] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994.

[15] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

[16] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

[17] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[18] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 111–118, USA, 2010. Omnipress.

[19] Alberto Broggi, Pietro Cerri, Mirko Felisa, Maria Chiara Laghi, Luca Mazzei, and Pier Paolo Porta. The vislab intercontinental autonomous challenge: an extensive test for a platoon of intelligent vehicles. *International Journal of Vehicle Autonomous Systems*, 10(3):147–164, 2012.

[20] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997.

[21] Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, pages 381–387, Cambridge, MA, USA, 2000. MIT Press.

[22] 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. *CoRR*, abs/1703.07570, 2017.

[23] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *IEEE CVPR*, 2016.

[24] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate

object class detection. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 424–432, Cambridge, MA, USA, 2015. MIT Press.

[25] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals using stereo imagery for accurate object class detection. *CoRR*, abs/1608.07711, 2016.

[26] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. *CoRR*, abs/1611.07759, 2016.

[27] François Chollet et al. Keras. `https://keras.io`, 2015.

[28] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: object detection via region-based fully convolutional networks. *CoRR*, abs/1605.06409, 2016.

[29] V. Dhiman, Q. H. Tran, J. J. Corso, and M. Chandraker. A continuous occlusion model for road scene understanding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4331–4339, June 2016.

[30] Xinxin Du, Marcelo H. Ang Jr., Sertac Karaman, and Daniela Rus. A general pipeline for 3d detection of vehicles. *CoRR*, abs/1803.00387, 2018.

[31] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin A. Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust RGB-D object recognition. *CoRR*, abs/1507.06821, 2015.

[32] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. *CoRR*, abs/1609.06666, 2016.

[33] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[34] Sanja Fidler, Sven Dickinson, and Raquel Urtasun. 3d object detection and viewpoint estimation with a deformable 3d cuboid model. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 611–619. Curran Associates, Inc., 2012.

[35] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach.* Prentice Hall Professional Technical Reference, 2002.

[36] 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.

[37] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[38] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[39] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[40] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[42] Saurabh Gupta, Ross B. Girshick, Pablo Arbelaez, and Jitendra Malik. Learning rich features from RGB-D images for object detection and segmentation. *CoRR*, abs/1407.5736, 2014.

[43] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. *CoRR*, abs/1412.1710, 2014.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[46] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[47] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[48] Kevin Jarrett, Koray Kavukcuoglu, Karol Gregor, and Yann LeCun. What is the best feature learning procedure in hierarchical recognition architectures? *CoRR*, abs/1606.01535, 2016.

[49] T Jayalakshmi and Santhakumaran A. Statistical normalization and back propagation for classification. 3:89–93, 01 2011.

[50] S. Buehler M. Iagnemma K., Singh. *The 2005 DARPAvGrand Challenge: The Great Robot Race*, volume 36. Springer, 1st. edition, 2007.

[51] S. Buehler M. Iagnemma K., Singh. *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, volume 56. Springer, 1st. edition, 2009.

[52] Takeo Kanade, Chuck Thorpe, and William Whittaker. Autonomous land vehicle project at cmu. In *Proceedings of the 1986 ACM Fourteenth Annual Conference on Computer Science*, CSC '86, pages 71–80, New York, NY, USA, 1986. ACM.

[53] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[54] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Lake Waslander. Joint 3d proposal generation and object detection from view aggregation. *CoRR*, abs/1712.02294, 2017.

[55] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[56] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81(2):155, Jul 2008.

[57] Bo Li. 3d fully convolutional network for vehicle detection in point cloud. *CoRR*, abs/1611.08069, 2016.

[58] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3d lidar using fully convolutional network. *CoRR*, abs/1608.07916, 2016.

[59] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.

[60] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[61] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[62] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry. *CoRR*, abs/1612.00496, 2016.

[63] Xin C. Wang Z. Zhang N. Zheng, J. *China future challenge: Beyond the intelligent vehicle*, volume 16, pages 8–10. IEEE Intell. Transp. Syst. Soc. Newslett, 2014.

[64] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[65] Michael Nielsen. Neural networks and deep learning, Dec. 2017.

[66] University of Washington. Park shuttle automated driverless vehicle. `http://faculty.washington.edu/jbs/itrans/parkshut.htm`, 2009.

[67] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, October 2010.

[68] Cuong Cao Pham and Jae Wook Jeon. Robust object proposals re-ranking for object detection in autonomous driving using convolutional neural networks. *Signal Processing: Image Communication*, 53:110 – 122, 2017.

[69] Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 305–313. Morgan-Kaufmann, 1989.

[70] Charles Ruizhongtai Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. *CoRR*, abs/1711.08488, 2017.

[71] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[72] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[73] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[74] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[75] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[76] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[77] J. Schlosser, C. K. Chow, and Z. Kira. Fusing lidar and images for pedestrian detection using convolutional neural networks. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2198–2205, May 2016.

[78] Jürgen Schmidhuber. Prof. schmidhuber's highlights of robot car history. `http://people.idsia.ch/~juergen/robotcars.html`, 2009.

[79] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[80] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[81] S. Song and M. Chandraker. Joint sfm and detection cues for monocular 3d localization in road scenes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3734–3742, June 2015.

[82] S. Song and J. Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 808–816, June 2016.

[83] Shuran Song and Jianxiong Xiao. Sliding shapes for 3d object detection in depth images. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 634–651, Cham, 2014. Springer International Publishing.

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

[85] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[86] Richard Szeliski. Computer vision algorithms and applications, 2011.

[87] Yichuan Tang. Deep learning using support vector machines. *CoRR*, abs/1306.0239, 2013.

[88] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[89] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng. End-to-end text recognition with convolutional neural networks. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3304–3308, Nov 2012.

[90] Zining Wang, Wei Zhan, and Masayoshi Tomizuka. Fusing bird view LIDAR point cloud and front view camera image for deep object detection. *CoRR*, abs/1711.06703, 2017.

[91] R. G. J. Wijnhoven and P. H. N. de With. Fast training of object detection using stochastic gradient descent. In *2010 20th International Conference on Pattern Recognition*, pages 424–427, Aug 2010.

[92] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent CRF for real-time road-object segmentation from 3d lidar point cloud. *CoRR*, abs/1710.07368, 2017.

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

[94] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. Pointfusion: Deep sensor fusion for 3d bounding box estimation. *CoRR*, abs/1711.10871, 2017.

[95] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[96] Jing Zhou, Xiaopeng Hong, Fei Su, and Guoying Zhao. Recurrent convolutional neural network regression for continuous pain intensity estimation in video. *CoRR*, abs/1605.00894, 2016.

[97] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017.

[98] M.Zeeshan Zia, Michael Stark, and Konrad Schindler. Are cars just 3d boxes? - jointly estimating the 3d shape of multiple objects. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[99] Zeeshan Zia, Michael Stark, Bernt Schiele, and Konrad Schindler. Detailed 3d representations for object recognition and modeling. *IEEE Transactions on Patterm Analysis and Machine Intelligence (PAMI)*, 35(11):2608 – 2623, 2013.