Faculty of Computer Science and Mathematics

Department of Computer Science

Project Report

Object Distance Estimation

supervised by

Prof. Dr. Gemma Roig

authored by

Carla Frenzel (5602924)
Tim Rosenkranz (6929884)
Benedikt Schröter (7487868)
Nils Möbus (6466745)

17. February 2022

Contents

1 Introduction **1**
1.1 Motivation 1

2 Research Papers **1**
2.1 Paper I: Learning Object-Specific Distance From a Monocular Image 1
2.2 Paper II: Representation Based Regression for Object Distance Estimation . . . 1

3 Evaluating Other Approaches **2**
3.1 Relational Reasoning & Relation Networks 2
3.2 Space-Time Region Graphs & Graph Convolutional Networks 2

4 Model Modification **3**
4.1 Recreating Results of Paper II 3
4.2 Modifying Original Model 3

5 Interobject Distance Estimation **4**
5.1 Background 4
5.2 Requirements 4
5.3 Data Acquisition 4
5.4 Data Processing 5
5.5 Prototype 6
5.6 Altered Prototype 6
5.7 Performance evaluation 7
5.7.1 Prototype Performance 7
5.7.2 Altered Prototype Performance 8
5.8 Experiments 8

6 Conclusion **9**
6.0.1 Future Work 9

1 Introduction

Imagine you are driving in a self-driving car and a long street with other cars lay in front of you. To regulate the speed reliably, the self-driving car needs to know how far away the car right in front of it is. In our project, we look at the state-of-the-art approach to estimating the distance of an object to the camera. Our work with this approach can be divided into following parts: Reseach, reproducing the results of the research paper, modifying the original model and augmenting the model with our own enhancements.

1.1 Motivation

Traditionally, the distance to an object could be calculated with following formula.

$$d_{istance} = f_{ocal\,width} \cdot \frac{O_{bject\,dimension}}{P_{rojection\,dimension}} \quad (1)$$

But the problem in real-world scenarios is that we do not know the object dimension. To find out the distance nevertheless, modern self-driving cars use LiDAR sensors which are based on the reflection time of laser beams. These sensors are highly accurate but very expensive (around 500 to 8.000 $ per unit). Monocular Cameras on the other hand are less accurate but a low-cost alternative (around 125 $ per unit). Stakeholders can benefit from their high availability, color capturing, reliability in bad weather and simplicity to integrate them into the product. The state-of-the-art for finding out the distance of objects with monocular cameras is using Supervised Deep Neural Networks. First, features are extracted and object classification is applied. Then, the Neural Network is trained to estimate distances which results in a depth map or an explicit distance as an output. In some cases, Deep Reinforcement Learning also is used. We have identified the following constraints and challenges for estimating the distance: The object is unknown, we have no prior knowledge about the scene, we only have a single RGB image and we aim for low computational complexity after training, i. e. to directly calculate it in a car fast.

2 Research Papers

After we defined and examined the problem we want to work on, our team searched for the state-of-the-art solution. We found a paper that explicitly only works with data from a single RGB camera and a second paper that improves the results.

2.1 Paper I: Learning Object-Specific Distance From a Monocular Image

This paper was published in 2019 in ICCV [1] and it represents the first learning-based model to directly predict distances for objects. The model architecture (Figure 1) consists of these components: The feature extractor (e. g. VGG16 or Res50) generates a feature map for the whole RGB image. Then, Region of Interest (ROI) Max-Pooling is applied. The distance regressor predicts the distance from the object specific ROI feature. In parallel, the multiclass classifier predicts the category from the ROI feature.

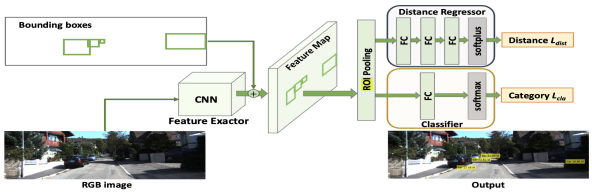


Figure 1: Paper I: Framework of Base Model

2.2 Paper II: Representation Based Regression for Object Distance Estimation

In this 2021 published paper [2], the concept of Paper I is developed further. The new framework is based on Convolutional Support Estimator Networks (CSEN) introduced in [3]. CSEN compute a direct mapping for the Support Estimation (SE) part in a representation-based classification problem. In this paper the architecture is further

developed to suit a regression problem (Figure 2).
Figure (Figure 2) shows the model archhitecture. The input to the model are images with objects. First, the objects are cropped out and features extraced. For this pretrained Keras models are used. This will be further discussed in section 4.2. After cropping, classifying and feature extraction, a representatitve dictionary D is build. In the inference phase it is attempted to solve equation 2 for x

$$y = D * x \tag{2}$$

where x corresponds to a vector of representation coefficients and D is the representative dictionary. In this paper and in our work the Collaborative Representation-based Classification (CRC) approach is used. This means that the l_2 solution is used.

$$\hat{x} = (D^T * D + \lambda * I)^{-1} * D^T * y \tag{3}$$

With λ being a regularization parameter. Now, the calculated support estimations (SE) can be used in the regression to learn object distances. Another modification to [3] is the CL-CSEN model. The main relevant difference to the above descibed CSEN model is the use compressive learning to optimize the both support estimation and the proxy mapping. With the CSEN model the proxy mapping can not be further optimized. It is treated as a constant. In the CL-CSEN two dense layers are added to finetune the mapping before it is used for the regression model.

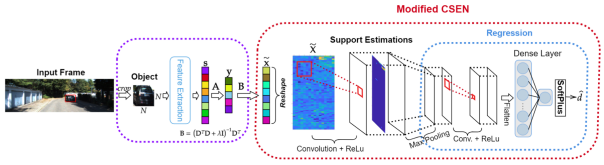


Figure 2: Paper II: CSEN Framework

3 Evaluating Other Approaches

As the distance estimation of objects has a wide range of applications, there are multiple solutions to this problem. Together with our tutor, we looked out for papers that also deal with relationships between objects. Although we did not have the capacity to apply every approach, we want to discuss the possible impact on our project.

3.1 Relational Reasoning & Relation Networks

Neural networks oftentimes have problems with rational questions about relational reasoning because they do not have the general capacity. One example question could be: „Are there any blue objects that have the same size as the yellow object?“. This relational question requires explicit reasoning about the relations between all objects in the image. In this paper, the solution is introduced to extend a deep learning architecture with Relation Networks (RNs) [4]. Then, its performance on visual question answering, text-based question answering and complex reasoning about dynamic physical systems is examined. RNs are optimized for learning to infer relations, being data efficient and takes a set of objects as the input (order invariant and versatile). Since RNs are developed to compare attributes of objects, we came to the conclusion that they would not be suitable for being trained to find out the exact distance between objects in a scene. Furthermore, it is hard to adapt them to provide regression, which would be extremely helpful in our case.

3.2 Space-Time Region Graphs & Graph Convolutional Networks

This paper examines the recognition of simple actions that happen in a video sequence [5]. For this purpose, a video is represented by a space-time region graph. The nodes in this graph represent the object regions from different frames in the video. The edges in this graph either describe temporal changes of shapes or functional relationships between objects (human-object and object-object). Then, reasoning with the help of Graph Convolutional Networks (GCNs) is performed

on the graph. GCNs were originally developed for applications in Natural Language Processing. The GCN in the paper is set up with multiple stacked layers of graph convolutions and average pooling over all the object nodes afterwards. The GCN's output are updated features for each object node, which is then used for performing classification. Our evaluation on this method is that we could not modify it to help us with finding out the distance between models. Space-Time Region Graphs are optimized to work with the relationship between objects and not the exact distance.

4 Model Modification

The first step in our project is to get a understanding of the papers in terms of code. For that we investigated the code of the CSEN model and conducted a few experiments to deepen our understanding on the topic.

4.1 Recreating Results of Paper II

Paper II features a public implementation that can be found on GitHub (https://github.com/meteahishali/CSENDistance). In the following, this will serve as a basis for our work on the code. The code works with the „KITTI 3D Object" dataset, which contains 80.000 labeled objects in 15.000 images. It provides 3D or 2D bounding boxes for various object types (cars, vans, trucks, pedestrians, cyclists and more) and the corresponding distance to the camera (http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d). To start our work with the code, every team member executed the code on their machine as a trial. For the CSEN method we obtained very similar results as published in the paper.

4.2 Modifying Original Model

For the modification of the original script, each team member was assigned a different deep learning model to apply. So everyone could gain a deep understanding of the code structure. We used the already suggested models VGG19 [6], DenseNet121[7], and ResNet50 [8] and added the Xception [9] and InceptionV3 [10] models. When the Keras model is instantiated in the script, the pre-trained weights are downloaded automatically. All Keras models used are pretained for image classification on the ImageNet dataset. The models are also built fitting to the image data format (i. e. height, width, depth) in the Keras configuration file. The accuracy of each model is listed in Table 1a below.

The Xception model, which stands for "Exreme Inception", shows the best statistical results. The results are comparable to the ResNet50 model but still show a slight improvement, especially in the variation and deviation of the results. The InceptionV3 model shows results that are comparabel to the VGG19 model. InceptionV3 shows slight improvement of results compared to the VGG19 model in all statiscal measurements except the ARD value. The worst performance in the original CSEN model can be observed for the DenseNet121 model with all result below the results of all other introduced models. It is important to mention that for the Xception and InceptionV3 model, input image size has been adjusted. The Xception model requires a minimum image size of (71,71,3) and the InceptionV3 of (75,75,3). The image size has thus been adjusted during feature extraction and also during the regression to fit the kernel size to the image size. Results may have been influenced by the larger image size. We choose an image size of (128,128,3) for the Xception and InceptionV3 model opposed to the image size of (64,64,3) for VGG19, ResNet50, and DenseNet121 models. Results thus can be influenced by the doubled image size. Reasons for choosing the Xception and InceptionV3 model include the ease of implementation as both are available as pretrained Keras models on the same dataset as the VGG19, ResNet50, and DenseNet121 models. This makes comparison between models easier and fair. Another reason is the relative small model

(a) Statistical performance for the CSEN method

	ARD ↓	SRD ↓	RMSE ↓	RMSE_{log} ↓	# params
DenseNet121	0.3308 ± 0.036	3.0487 ± 0.534	6.9950 ± 0.367	0.5370 ± 0.217	3,326
VGG19	0.3401 ± 0.039	3.1667 ± 0.563	7.2027 ± 0.331	0.6763 ± 0.264	3,326
ResNet50	0.2835 ± 0.035	2.2479 ± 0.437	6.2142 ± 0.333	0.5074 ± 0.155	3,326
Xception	0.272 ± 0.005	2.152 ± 0.073	6.229 ± 0.026	0.390 ± 0.013	3,326
InceptionV3	0.320 ± 0.004	2.880 ± 0.059	6.838 ± 0.029	0.507 ± 0.017	3,326

and parameter size. With a model size of 88MM and 92MB respectively both models are rather compact. Parameter size is 22,910,480 and 23,851,784 respectively. Most importantly, both models show high accuracy ranking on the keras application board.[1]

5 Interobject Distance Estimation

In an extend to the original model we propose the estimation of the distance between two objects in a camera view (or frame). This concept can be realised through two approaches:

1. Geometric approach, using the existing Model to determine the distance from the camera to the objects of interest as well as the coordinates in the two dimensional projection plane. The coordinates and the distance can be stacked into a three-dimensional vector to compute the euclidean - real world - distance.

2. Data driven approach, modifying the model discussed in the papers is adapted to our proposal.

Our focus is on the second, the data driven appraoch.

5.1 Background

As mentioned in the beginning of this report we aim to develop a distance estimator for inter-object distances on signle RGB images. Such an estimator can find application in e.g. autonomous vehicles, i.e. to gain information about objects around the vehicle, or surveillance cameras, i.e. to monitor people to follow the COVID-19 rules.

5.2 Requirements

The requirements for our project are the mainly the code for the original CSEN-Distance project [2] and the kitti-dataset [11] which is also used in the original project. As mentioned before the Code for the CSEN-Project is available on GitHub, so we forked the repository to use as foundation of our project[2].
This results in our requirements being the same as those of the original project. All additions and changes are fitted into the existing structure and dependencies which are

- Python including Tensorflow and OpenCV for image processing and feature extraction
- MATLAB for processing the input features (splitting, pre-predicitions)
- Data to train the model

5.3 Data Acquisition

As already stated we use the kitti-dataset [11], speicifically the set for three-dimensional object detection which has been described in section 2.2. Further the dataset includes information about the exact 2D (bounding boxes) and 3D (real-world coordinates) locations of the objects.
The original paper comes with several scripts to process the downloaded dataset into a csv

[1] as obtained on 13.02.2022
[2] URL: https://github.com/Tim-orius/CSENDistance

(a) Excerpt of the original csv with added IDs

filename	class	truncated	occluded	observation angle	xmin	ymin	xmax	ymax	height	width	length	xloc	yloc	zloc	rot_y	id
000000.txt	Pedestrian	0	0	-0.2	712.4	143	810.73	307.92	1.89	0.48	1.2	1.84	1.47	8.41	0.01	0
000001.txt	Truck	0	0	-1.57	599.41	156.4	629.75	189.25	2.85	2.63	12.34	0.47	1.49	69.44	-1.56	1
000001.txt	Car	0	0	1.85	387.63	181.54	423.81	203.12	1.67	1.87	3.69	-16.53	2.39	58.49	1.57	2
000001.txt	Cyclist	0	3	-1.65	676.6	163.95	688.98	193.93	1.86	0.6	2.02	4.59	1.32	45.84	-1.55	3
000002.txt	Misc	0	0	-1.82	804.79	167.34	995.43	327.94	1.63	1.48	2.37	3.23	1.59	8.55	-1.47	4
000002.txt	Car	0	0	-1.67	657.39	190.13	700.07	223.39	1.41	1.58	4.36	3.18	2.27	34.38	-1.58	5
000003.txt	Car	0	0	1.55	614.24	181.78	727.31	284.77	1.57	1.73	4.15	1	1.75	13.22	1.62	6
000004.txt	Car	0	0	1.96	280.38	185.1	344.9	215.59	1.49	1.76	4.01	-15.71	2.16	38.26	1.57	7
000004.txt	Car	0	0	1.88	365.14	184.54	406.11	205.2	1.38	1.8	3.41	-15.89	2.23	51.17	1.58	8
000005.txt	Pedestrian	0	0	1.94	330.06	178.74	360.77	238.64	1.87	0.96	0.65	-8.5	2.07	23.02	1.59	9
000006.txt	Car	0	2	-1.55	548	171.33	572.4	194.42	1.48	1.56	3.62	-2.72	0.82	48.22	-1.62	10
000006.txt	Car	0	0	-1.21	505.25	168.37	575.44	209.18	1.67	1.64	4.32	-2.61	1.13	31.73	-1.3	11
000006.txt	Car	0	0	0.15	49.7	185.65	227.42	246.96	1.5	1.62	3.88	-12.54	1.64	19.72	-0.42	12
000006.txt	Car	0	1	2.05	328.67	170.65	397.24	204.16	1.68	1.67	4.29	-12.66	1.13	38.44	1.73	13
000007.txt	Car	0	0	-1.56	564.62	174.59	616.43	224.74	1.61	1.66	3.2	-0.69	1.69	25.01	-1.59	14
000007.txt	Car	0	0	1.71	481.59	180.09	512.55	202.42	1.4	1.51	3.7	-7.43	1.88	47.55	1.55	15
000007.txt	Car	0	0	1.64	542.05	175.55	565.27	193.79	1.46	1.66	4.05	-4.71	1.71	60.52	1.56	16
000007.txt	Cyclist	0	0	1.89	330.6	176.09	355.61	213.6	1.72	0.5	1.95	-12.63	1.88	34.09	1.54	17
000008.txt	Car	0.88	3	-0.69	0	192.37	402.31	374	1.6	1.57	3.23	-2.7	1.74	3.68	-1.29	18

(b) Excerpt of the resulting data

filename	object_ids	x1	y1	z1	x2	y2	z2	obj distance 2D	obj distance 3D
000001.txt	(1, 2)	0.47	1.49	69.44	-16.53	2.39	58.49	20.22133774012	20.2413561798611
000001.txt	(1, 3)	0.47	1.49	69.44	4.59	1.32	45.84	23.9569280167554	23.9575311749771
000001.txt	(2, 3)	-16.53	2.39	58.49	4.59	1.32	45.84	24.6186291251158	24.6418708705325
000002.txt	(4, 5)	3.23	1.59	8.55	3.18	2.27	34.38	25.8300483932957	25.838997658578
000004.txt	(7, 8)	-15.71	2.16	38.26	-15.89	2.23	51.17	12.9112547802295	12.9114445357597
000006.txt	(10, 11)	-2.72	0.82	48.22	-2.61	1.13	31.73	16.4903668849422	16.4932804499287
000006.txt	(10, 12)	-2.72	0.82	48.22	-12.54	1.64	19.72	30.1443593396841	30.1555102759015
000006.txt	(10, 13)	-2.72	0.82	48.22	-12.66	1.13	38.44	13.944604691421	13.9480500429272
000006.txt	(11, 12)	-2.61	1.13	31.73	-12.54	1.64	19.72	15.5834848477483	15.5918279877633
000006.txt	(11, 13)	-2.61	1.13	31.73	-12.66	1.13	38.44	12.0841466392956	12.0841466392956
000006.txt	(12, 13)	-12.54	1.64	19.72	-12.66	1.13	38.44	18.7203846114336	18.7273302955867
000007.txt	(14, 15)	-0.69	1.69	25.01	-7.43	1.88	47.55	23.5261386546964	23.5269058739138
000007.txt	(14, 16)	-0.69	1.69	25.01	-4.71	1.71	60.52	35.7368227462935	35.7368283427615
000007.txt	(14, 17)	-0.69	1.69	25.01	-12.63	1.88	34.09	15.0003333296297	15.0015365879633
000007.txt	(15, 16)	-7.43	1.88	47.55	-4.71	1.71	60.52	13.2521432228904	13.2532335676996
000007.txt	(15, 17)	-7.43	1.88	47.55	-12.63	1.88	34.09	14.42953914718	14.42953914718
000007.txt	(16, 17)	-4.71	1.71	60.52	-12.63	1.88	34.09	27.5911453187431	27.5916690325178
000008.txt	(18, 19)	-2.7	1.74	3.68	-1.17	1.65	7.86	4.45121331773709	4.45212308904415
000008.txt	(18, 20)	-2.7	1.74	3.68	3.81	1.64	6.15	6.96282988446508	6.96354794626992

Table 2: Snippets of the processed data.

and then use the csv to compute the network input features. We adapt and extend these scripts for our needs. In total we use three python scripts with the first being the original `generate-csv.py` that is used to transform the donwloaded data into a csv file. This file is then expanded with our own script `give-ids.py` to add an ID to every object, the result is shwon in table 2a. The ID is a counter, basically the number of row the data is stored, that is added in a column. The resulting csv data is now used by a second script `generate-data-interdist` we developed to iteratively pair two objects from the same image together and compute the actual distance between them given by the data in the kitti dataset; the data is shown in table 2b. We chose to compute the distance between two objects in a three-dimensional space and in a two-dimensional space. The reason for this is the usage of the objects center for our calculations. However when one object is significantly larger, i.e. higher, the distance will increase proportionally. To avoid this issue we measure the distance on a flat plane and erase the coordinate specifiying the height of the objects, namely the Y-Coordinate. The resulting values are labeled *obj distance 2D* and *obj distance 3D* in table 2b respectively. Whatsoever the suspected difference is so small that it presumably can be neglected.

5.4 Data Processing

Following the data acquisition and preparation it is required to preprocess the data to feed into the network. The processing is done with two scripts, `feature_extraction.py` and `processFeatures.m`, with the latter being a MATLAB-script. Both scripts came with the original model but a series of adaptions had to be implemented for our application. The most changes have been made to the python

script that is used to load the respective images from the dataset and process them through the selected model (VGG, ResNet, DenseNet).[3] The computed features are then stored in a `.mat` file to be further processed in the MATLAB-script.
The finally computed data, the interdistance data, consists of approximately 140,000 entrys, i.e. 140,000 object pairs. The processing of such a large batch of data takes a lot of time so we decided to add the functionality to randomly sample a desired number of samples from the created dataset. It is important to not choose a number of samples too low because too few samples can lead to numerical issues in the following processes. The default sample size has been chosen as 20,000 - half of the sample size used in the original model - for faster computation and easier execution.
The stored features are loaded in the MATLAB-script and further split for a training of five runs with 100 epochs each. As we conducted a series of experiments regarding the processing, further analysis can be found in section 5.8.

5.5 Prototype

With the adapted processed we have created a model for distance estimation between two selected objects in an image. We named the basic model with just the necessary changes the ”Prototype”.
In the protoype we decided to perform the data processing per object and stack the features at the end. So the procedure is the following (Figure 3):

1. For a chosen image, the objects in the frame are marked with bounding boxes and their locations (coordinates) are detected. In the kitti-dataset this is already done.
2. The objects are then taken as pairs. Both objects are cropped separately and fed into the backbone networks to extract the features.
3. The features are stacked into an input vector. Further the labels are saved too.

Afterwards the data is split as described before.

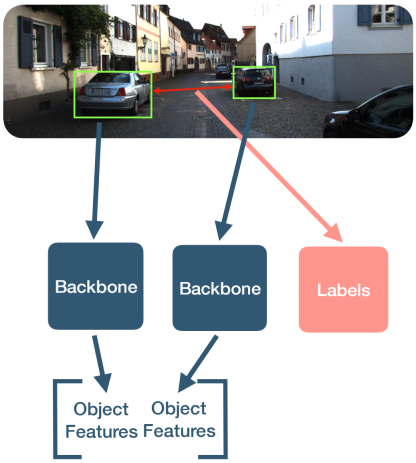


Figure 3: Prototype: Data processing

5.6 Altered Prototype

Building on the prototype architecture explained above, we experimented with another approach to get better results. The underlying idea of the modified prototype was to keep as much spatial information in the input data as possible. For this purpose, we have altered the data processing as follows: While in the first approach the bounding boxes of the objects whose inter object distance is to be estimated are fed individually into the backbone network, in this approach we proceed by computing the bounding box that includes both objects and then feeding it into the backbone nerwork. Once again, the true distances between the two objects serve as labels. All further steps of the model pipeline then remain unchanged. The differences in the data processing are visualized in figure 4. In summary, the merits of this approach are that, at least in theory, more spatial information is preserved while the computational effort remains comparable and the implementation is straight forward.

[3] The changes on the code can be investigated in the source code but are also further discussed in the upcoming sections

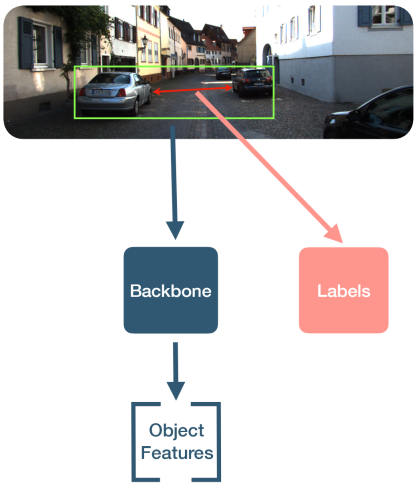


Figure 4: Altered prototype: Data processing

5.7 Performance evaluation

As explained before the training is split into five runs with a hundred epochs each. For every run several metrics are computed to qualitatively compare and discuss the results. First the Absolute Relative Distance (ARD) and the Squared Relative Distance (SRD) are computed as follows

$$ARD = \frac{1}{N}\sum_{i=1}^{N}(|\hat{d}_i - d_i|/d_i) \tag{4}$$

$$SRD = \frac{1}{N}\sum_{i=1}^{N}((\hat{d}_i - d_i)^2/d_i) \tag{5}$$

with d_i and $\hat{d}_i$ denoting the actual and predicted distances and N the number of samples in the split.
Additionally the Root of Mean Squared Error ($RMSE$) and the Root of the Mean squared logarithmic Error ($RMSE_{log}$) are defined as

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{d}_i - d_i)^2} \tag{6}$$

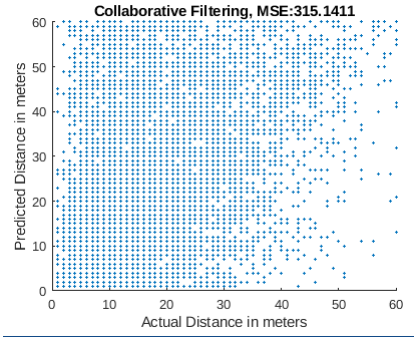
$$RMSE_{log} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(log\,\hat{d}_i - log\,d_i)^2} \tag{7}$$

We then take the median and standard deviation of the computed metrics of the five runs per backbone.

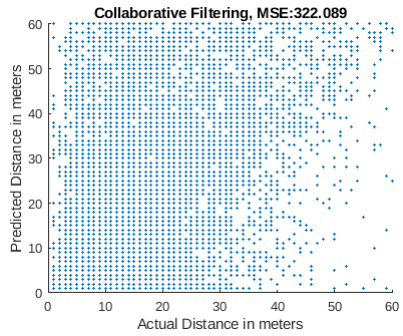
$$\varnothing R = \frac{\sum x_i}{n} \tag{8}$$

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} = \sqrt{\frac{\sum(x_i - \varnothing x)^2}{n-1}} \cdot \frac{1}{\sqrt{n}} \tag{9}$$

5.7.1 Prototype Performance



(a) *ResNet50*, 20k samples



(b) *DenseNet121*, 20k samples

Figure 5: Predicted vs. actual distances computed by the base CRC model [2]

The performance of our proposed inter-object distance estimation prototype is shown in table 3. The *Root of Mean squared Error* (Equation 6) is the most interesting metric to look at as we can directly interpret the value. For the *CSEN* method we can see a deviation of about nine to ten meters while the *CL-CSEN* performs slightly better with deviations of seven to eight meters from the actual distance. These results indicate a decent if not good model, however when we plot the predicted and the actual distances (figure 5) we see a bulk of predicitions that are wrong for smaller actual distances. The question arises what causes these wrong predicitions. We speculate that the reasons

(a) Statistical performance for the CSEN method

Backbone	ARD ↓	SRD ↓	RMSE ↓	RMSE_{log} ↓	# params
DenseNet121	0.713 ± 0.012	7.936 ± 0.204	9.732 ± 0.016	0.657 ± 0.002	3,326
VGG19	0.738 ± 0.009	8.669 ± 0.147	10.086 ± 0.03	0.684 ± 0.01	3,326
ResNet50	0.719 ± 0.006	8.148 ± 0.115	9.566 ± 0.033	0.654 ± 0.002	3,326

(b) Statistical performance for the CL-CSEN method

Backbone	ARD ↓	SRD ↓	RMSE ↓	RMSE_{log} ↓	# params
DenseNet121	0.653 ± 0.006	6.836 ± 0.105	8.208 ± 0.022	0.592 ± 0.002	1,233,326
VGG19	0.55 ± 0.004	5.032 ± 0.053	7.294 ± 0.032	0.533 ± 0.003	618,926
ResNet50	0.678 ± 0.006	7.204 ± 0.116	8.517 ± 0.043	0.605 ± 0.002	2,462,126

Table 3: Statistical performances for CSEN and CL-CSEN methods on the three backbones of the CSEN project [2] for our prototype

could potentially be occluded or truncated objects but there might also be other sources for problems as the methods are originally designed for distances to a single object, not two different objects.

5.7.2 Altered Prototype Performance

To assess the quality of our modified prototype, we compare the statistical performance, measured by the metrics already explained, of our two approaches (Table 3 and 4). Interestingly, it can be seen that in most cases the normal prototype scores better. Only in the combination of CSEN method and ARD or SRD metric does the altered prototype perform better. Another interesting aspect is that the alternative prototype works best in all combinations with ResNet50 as the backbone. The normal prototype, on the other hand, does not show such a clear preference. Only when using the CL-CSEN method it appears to achieve the best results with VGG19.
All in all it can be denoted that the performance of both approaches is very similar. The seen fluctuations can be caused by the random bacthes of data as we sample 20 000 out of 140 000 data points.

5.8 Experiments

To get a better understanding of the influence of the dictionary size on the performance, we varied it experimentally using the CSEN model. Dictionary size is determined by two parameters: the number of support sets and the samples per set. The prototype works with support sets whose samples differ in their distance by only one meter. If this criterion is modified so that the samples of a set may only differ by up to 2 meters, for example, the number of sets is halved. For our experiment with the prototype and the altered prototype we considered support sets with a tolerance of 1,2,5 and 7.5 meters. At the same time we tested 10,15,20 and 25 samples per support set. The results of the different combinations are shown in figure 6 and figure 7 measured with the ARD.

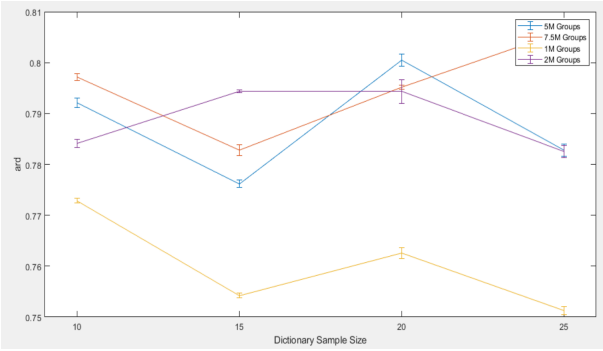


Figure 6: Prototype: Variation in dictionary size

(a) Altered Prototype: Statistical performance for the CSEN-1D method

Backbone	ARD ↓	SRD ↓	RMSE ↓	RMSE_{log} ↓	# params
DenseNet121	0.624 ± 0.0006	6.5175 ± 0.095	10.201 ± 0.02	1.2 ± 0.3	3,326
VGG19	0.6239 ± 0.0003	6.562 ± 0.084	10.2 ± 0.016	1.052 ± 0.135	3,326
ResNet50	0.581 ± 0.00003	5.767 ± 0.013	9.721 ± 0.021	0.782 ± 0.029	3,326

(b) Altered Prototype - Statistical performance for the CL-CSEN-1D method

Backbone	ARD ↓	SRD ↓	RMSE ↓	RMSE_{log} ↓	# params
DenseNet121	0.663 ± 0.002	7.034 ± 0.367	9.251 ± 0.012	0.708 ± 0.005	618,926
VGG19	0.630 ± 0.0002	6.7152 ± 0.015	9.23 ± 0.027	0.777 ± 0.025	311,726
ResNet50	0.587 ± 0.001	5.73 ± 0.129	8.893 ± 0.017	0.585 ± 0.0004	1,233,326

Table 4: Altered Prototype - Statistical performances for CSEN and CL-CSEN methods on the backbones for the altered prototype

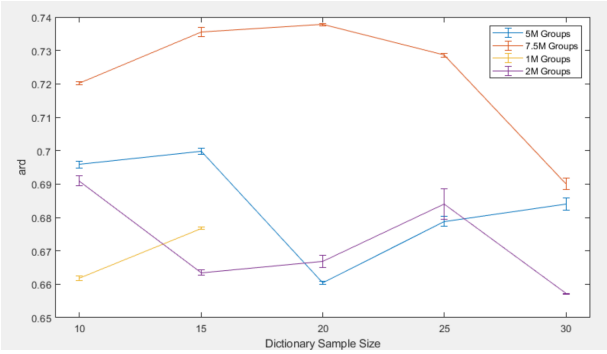


Figure 7: Altered prototype: Variation in dictionary size

From the plots, it can be seen that the alternative prototype performs overall a bit better when measured with ARD. In addition, a high number of support sets seems to be advantageous, which can be seen for example in the comparison of the combinations with 2 meter support sets and 7.5 meter support sets. There also appears to be a slight trend towards a greater number of samples per support set being beneficial.
In summary, the overall influence of Dictionary Size is clearly present, but not all that great. Furthermore, the number of support sets seems to be considerably more relevant than the actual samples per support set.

6 Conclusion

We conclude that we have succesfully created a inter-object distance stimator based on the CSEN-regressor model and thus achieved what we aimed for with reasonably good results. Further we gained extensive intel about the CSEN model within our conducted experiments.

We also claim that the use of a regressor model is better suited for estimation tasks of continuous metrics than a classifier.

6.0.1 Future Work

We have seen that the performance of our developed models is reasonable but there is still room for improvements. This can be the topic for future work.

One possibility to improve the estimator is additionally use the angle between objects within the network, which is contained in the kitti dataset. We even computed the inter-object angle, however there was no computation involving this information in the original CSEN method and we decided to keep our approaches rather vanilla.[4]

The angle contains some spatial information so it could be an interesting idea to see whether the angle alone can be used to estimate the distance between the objects. Apart from that the angle can be fed into our network with the object features to increase the input information for the network.

[4]Mainly due to complexity and to better compare our model with the original CSEN

References

[1] Jing Zhu et al. Learning Object-specific Distance from a Monocular Image. 2019. arXiv: 1909.04182 [cs.CV].

[2] Mete Ahishali et al. Representation Based Regression for Object Distance Estimation. 2021. arXiv: 2106.14208 [cs.CV].

[3] Mehmet yamaç et al.
Convolutional Sparse Support Estimator Network (CSEN) From energy efficient support estimation t
Mar. 2020.

[4] Adam Santoro et al. A simple neural network module for relational reasoning. 2017. arXiv: 1706.01427 [cs.CL].

[5] Xiaolong Wang and Abhinav Gupta. Videos as Space-Time Region Graphs. 2018. arXiv: 1806.01810 [cs.CV].

[6] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: CoRR abs/1409.1556 (2015).

[7] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: CoRR abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[8] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: CoRR abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[9] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: CoRR abs/1610.02357 (2016). arXiv: 1610.02357. URL: http://arxiv.org/abs/1610.02357.

[10] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: CoRR abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[11] 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.

Appendix

Authors and Contributors

- **Carla**: Research Papers, Model Modification, Prototype
- **Tim**: Data acquisition & processing, Prototype & Performance(s), Conclusion
- **Benedikt**: Data Processing, Prototype, Altered Prototype, Experiments
- **Nils**: Introduction, Research Papers, Evaluating other Appraches, Model Modification