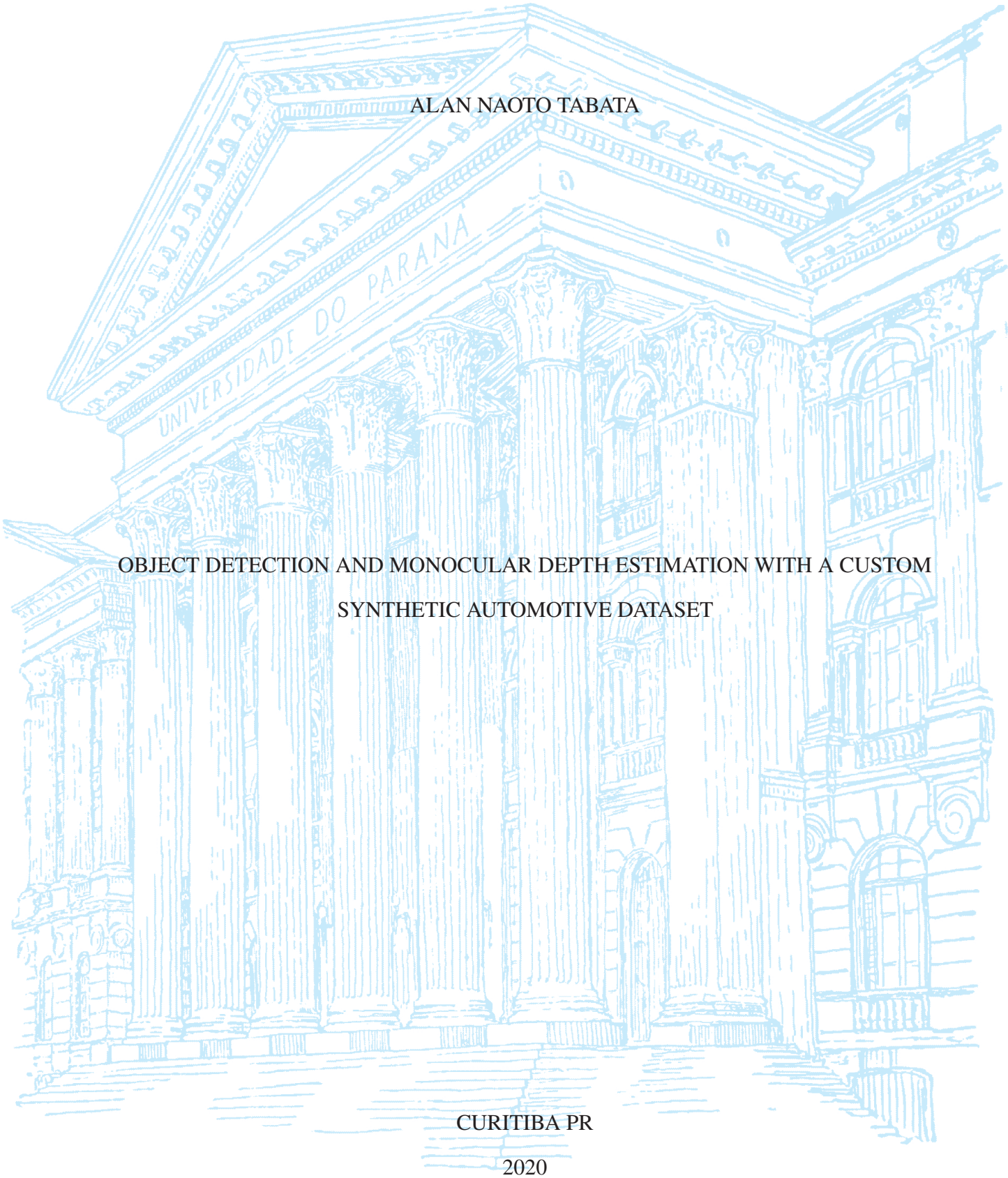
UNIVERSIDADE FEDERAL DO PARANÁ

ALAN NAOTO TABATA



OBJECT DETECTION AND MONOCULAR DEPTH ESTIMATION WITH A CUSTOM SYNTHETIC AUTOMOTIVE DATASET

CURITIBA PR

2020

ALAN NAOTO TABATA

OBJECT DETECTION AND MONOCULAR DEPTH ESTIMATION WITH A CUSTOM SYNTHETIC AUTOMOTIVE DATASET

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Engenharia Elétrica no Programa de Pós-Graduação em Engenharia Elétrica, Setor de Tecnologia, da Universidade Federal do Paraná.

Área de concentração: *Sistemas Eletrônicos*.

Orientador: Prof. Dr. Leandro dos Santos Coelho.

Coorientador: Prof. Dr. Alessandro Zimmer.

CURITIBA PR

2020

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

T112o Tabata, Alan Naoto
Object detection and monocular depth estimation with a custom synthetic automotive dataset [recurso eletrônico] / Alan Naoto Tabata – Curitiba, 2020.

Dissertação - Universidade Federal do Paraná, Setor de Tecnologia, Programa de Pós-graduação em Engenharia Elétrica.

Orientador: Prof. Dr. Leandro dos Santos Coelho
Coorientador: Prof. Dr. Alessandro Zimmer

1. Redes neurais. 2. Base de dados sintética. 3. Detecção de pedestres e veículos. 4. Veículos autônomos. I. Universidade Federal do Paraná. II Coelho, Leandro dos Santos. III. Zimmer, Alessandro. IV. Título.

CDD: 006.32

Bibliotecária: Roseny Rivelini Morciani CRB-9/1585

MINISTÉRIO DA EDUCAÇÃO
SETOR DE TECNOLOGIA
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO ENGENHARIA ELÉTRICA - 40001016043P4

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em ENGENHARIA ELÉTRICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **ALAN NAOTO TABATA** intitulada: **Object detection and monocular depth estimation with a custom synthetic automotive dataset**, sob orientação do Prof. Dr. LEANDRO DOS SANTOS COELHO, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 30 de Julho de 2020.

Assinatura Eletrônica
30/07/2020 20:02:31.0
LEANDRO DOS SANTOS COELHO
Presidente da Banca Examinadora

Assinatura Eletrônica
30/07/2020 20:00:11.0
VIVIANA COCCO MARIANI
Avaliador Externo (PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ)

Assinatura Eletrônica
31/07/2020 10:39:04.0
GIDEON VILLAR LEANDRO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
03/08/2020 22:15:07.0
ROBERTO ZANETTI FREIRE
Avaliador Externo (PONTIFICA UNIVERSIDADE CATóLICA DO PARANá)

Assinatura Eletrônica
31/07/2020 16:31:37.0
ELIZETE MARIA LOURENÇO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

AGRADECIMENTOS

I would like to thank my parents, Wilson and Maria, as well as my brother, Cesar, for supporting and encouraging me during my studies.

To prof. Leandro dos Santos Coelho and prof. Alessandro Zimmer for guiding my research and providing the opportunity of studying and working abroad.

To my colleagues at CARISSMA and Lactec for providing helpful insights to my work and career.

To Lactec, TP Norte Matrinchã Transmissora de Energia S.A., TP Sul Guaraciaba Transmissora de Energia, Paranaíba Transmissora de Energia, Luziânia-Niquelândia Transmissora with the Brazilian Electricity Regulatory Agency (ANEEL) R&D program Project PD 08106-0002/2017 SISTEMA ROBOTIZADO PARA INSPEÇÃO DE SUBESTAÇÕES DE ENERGIA ELÉTRICA, for providing financial aid during the early stages of this work.

To the Bayerisches Hochschulzentrum für Lateinamerika (BAYLAT) for providing financial aid during the latter stages of this work.

RESUMO

Na indústria automotiva, o conceito de veículos autônomos vem se aproximando da realidade, com empresas disputando para serem as pioneiras em alcançar pelo menos o nível 3 de direção autônomo. Contudo, antes de implementar veículos autônomos em larga escala, pesquisas e testes devem ser realizados de forma a avaliar a segurança e confiabilidade dos veículos. Como uma das formas pelo qual veículos autônomos percebem seu entorno é por meio de câmeras, então uma abordagem para promover a segurança humana é a pesquisa em técnicas de visão computacional que podem ajudar o veículo a assimilar melhor o contexto em que ele se situa. Logo, nesse trabalho um algoritmo capaz de detectar pedestres e veículos, e a distância deles em relação à câmera será desenvolvido, de forma que trabalhos futuros possam aplicar técnicas de correção de trajetória com antecedência. As principais contribuições são a aplicação e validação de tais técnicas em um contexto diferente daqueles que já foram extensivamente testados na literatura. Nessa dissertação, isso é feito ao criar uma base de dados própria baseada no CARLA e avaliando a capacidade de transferência de conhecimento de algoritmos de visão computacional para outra base de dados real, o Waymo Open. O propósito de uma base de dados sintéticos é possibilitar a geração de grandes quantidades de dados à vontade, um requisito para parametrizar modelos de visão computacional baseados em redes neurais convolucionais profundas. O Faster R-CNN com a ResNet 50 de suporte é avaliada para a tarefa de reconhecimento de objetos, e para a estimativa de profundidade monocular o modelo monodepth2 com a U-Net e ResNet 18 de suporte foram avaliados. Na parte de detecção de objetos, foi notado que a injeção de dados sintéticos não auxiliou na generalização do modelo, com um decréscimo de 12% nas métricas de performance quando comparado com o modelo treinado do zero na base de dados Waymo skip 10. Para a estimativa de profundidade monocular, no entanto, os modelos com melhor desempenho provaram ser a combinação de dados sintéticos e reais, melhorando as métricas de performance em média 5% na base dados do Waymo. No geral, foi notada a importância da diversidade de dados para ambos algoritmos, com a iteração da base de dados sintética atual sendo benéfica para o monodepth2, mas não para o Faster R-CNN, o que sugere que ainda há espaço para melhorias. Essas observações levam a conclusão de que as características que impactam positivamente o modelo para criar uma base de dados diferem de acordo com o propósito do algoritmo, e portanto a criação de uma base de dados de propósito geral provavelmente não é ideal.

Palavras-chave: Detecção de pedestres e veículos. Estimativa de profundidade monocular. Redes neurais convolucionais profundas. Veículos autônomos. Base de dados sintética.

ABSTRACT

In the automotive industry, the concept of autonomous vehicles is becoming closer to reality, with companies disputing to be the pioneers on reaching at least a level 3 on driving automation. However, before implementing autonomous vehicles on a large scale, research and testing should be performed to assess its safety and reliability. Since one of the ways autonomous vehicles sense its surrounding is through cameras, then one approach to promote human safety is by researching computer vision techniques that may help the vehicle to better understand the context it is in. Therefore, on this work algorithms capable of detecting pedestrians and vehicles, and their distance to the camera are evaluated, in a way that future works can apply corrective trajectory procedures in advance. The main contributions of this work are application and validation of such techniques in a context different from those of which have already been extensively tested on the literature. In this dissertation, this is done by creating a custom CARLA-based synthetic dataset and evaluating its knowledge transfer capability with computer vision algorithms to a real-world dataset, Waymo Open. The purpose of a synthetic dataset is the possibility of generating huge amounts of data at will, a requirement for parametrizing state-of-the-art computer vision models based on deep convolutional neural networks. The Faster R-CNN with a ResNet 50 as backbone was evaluated for the bounding box task, and for monocular depth estimation, the monodepth2 model with a U-Net and ResNet 18 as backbone was evaluated. On the object detection part, it was noted that the injection of synthetic data did not aid in model generalization, with 12% performance decrease when compared to training from scratch on the Waymo skip 10 dataset. For monocular depth estimation, however, the best performing models proved to be different combinations of both synthetic and real-world data, with them improving the performance metrics on average 5% on the Waymo dataset. Overall, it is noted the importance of data variety for both algorithms, with the current synthetic dataset iteration being beneficial for monodepth2 but not Faster R-CNN, which suggests that there is still room for improvement. These observations lead to the conclusion that features which impact positively the model for creating a dataset differ according to the algorithm's purpose, and as such the creation of an all-purpose dataset is probably not ideal.

Keywords: Pedestrians and vehicles detection. Monocular depth estimation. Deep convolutional neural networks. Autonomous vehicles. Synthetic dataset.

LISTA DE FIGURAS

1.1 Forecast of autonomous vehicles integration in Europe 16

2.1 Chair examples 19
2.2 Image filter basic functionality 20
2.3 Machine learning general pipeline for each learning approach. 21
2.4 Model training behavior 22
2.5 k-fold cross validation with k=5 23
2.6 Perceptron. 24
2.7 Fully connected feedforward ANN 24
2.8 Activation functions. 25
2.9 Stride with value 1 27
2.10 Stride and padding with value 1 27
2.11 Stride and dilation rate with value 1 28
2.12 Convolution operation exemplified. (a) Convolution step 1; (b) Step 2 29
2.13 Pooling operations illustrated. 29
2.14 Transfer learning categories. 30
2.15 Single-class confusion matrix structure. 31
2.16 Multiclass confusion matrix structure 31
2.17 Areas of intersection and union example 32
2.18 Precision x recall chart example 34
2.19 Precision x recall smoothed chart example 34
2.20 Precision x recall chart example 35

3.1 Synthetic datasets and simulators. 38
3.2 Synthia dataset sample on all four weather seasons. (a) Autumn; (b) Winter; (c) Spring; (d) Summer 39
3.3 Sim 10k dataset sample. (a) Late night; (b) Night; (c) Day; (d) Midday. 39
3.4 Airsim interface. 40
3.5 Synscapes samples. (a) Street way; (b) Cyclist; (c) Street way variant (d) Dense pedestrians population 41
3.6 PreSIL sample. 41
3.7 (a) KITTI, (b) Virtual KITTI, and (c) Virtual KITTI 2 42
3.8 CARLA sample. (a) Cloudy; (b) Rainy; (c) Afternoon; (d) Midday 43
3.9 real-world automotive datasets 45
3.10 KITTI sample. (a) Street way 1 (b) Street way 2 (c) Comercial area (d) Highway 45

3.11 Cityscapes sample. 46
3.12 ApolloScapes sample (a) Semantic segmentation; (b) Instance segmentation; (c) Depth; (d) Lane markings. 46
3.13 BDD100k sample on two dashboard cameras. (a) Dashboard cmaera 1; (b) Dashboard camera 2 47
3.14 NuScenes sample. (a) Early day; (b) Night; (c) Midday; (d) Night alternate . . . 47
3.15 Waymo Open sample. (a) Midday; (b) Morning; (c) Rainy; (d) Clear weather . . 48
3.16 Relevance analysis on bounding box algorithms 50
3.17 R-CNN workflow 51
3.18 Fast R-CNN workflow 51
3.19 Faster R-CNN workflow 52
3.20 Relevance analysis on depth estimation algorithms 55
3.21 Inference samples of Godard et al. on KITTI. 56
3.22 By column from left to right: image, ground-truth, gaussian model, laplacian model 56
3.23 Multi-Scale high level. 57
3.24 Multi-Scale inference on KITTI. From top to bottom: input image, coarse prediction, refined output, and ground-truth. 57
3.25 Multi-Scale Convolutional Architecture's network on a high level. 58

4.1 High-level view methodology flowchart 63
4.2 Waymo Open - sensor setup 65
4.3 Waymo Open - LIDAR calibrated on RGB. 66
4.4 Waymo Open - Metadata database file 67
4.5 Waymo Open - Sample 1 67
4.6 Waymo Open - Sample 2 68
4.7 Waymo Open - Sample 3 68
4.8 Bounding boxes location heatmap for vehicles (a) and pedestrians (b) on Waymo v1.0.0 frontal camera 69
4.9 RGB (a) and grayscale (b) histograms for Waymo v1.0.0 69
4.10 RGB frame (left) and dense depth (right). For easier visualization, depth is shown in log scale 70
4.11 CARLA Town 01 - Residential area with plenty of green areas 71
4.12 CARLA Town 02 - Residential area of a medium-sized city. 71
4.13 CARLA Town 03 - Residential area with skyscrappers and tunnels 72
4.14 CARLA Town 04 - Sparse area with highways and a small residential area. . . . 72
4.15 CARLA Town 05 - Urban area with highways and skyscrappers 73
4.16 Vehicles and pedestrians - sample 1 74
4.17 Vehicles and pedestrians - sample 2 74

4.18 Custom weathers developed for CARLA . . . 75
4.19 Data capture flowchart for CARLA . . . 77
4.20 CARLA HDF5 timestamps structure . . . 77
4.21 CARLA HDF5 bounding box annotations structure . . . 78
4.22 Bounding boxes location heatmap for vehicles (a) and pedestrians (b) . . . 79
4.23 RGB (a) and grayscale (b) histograms for the synthetic dataset . . . 79

5.1 Object detection - Town holdout on synthetic data . . . 81
5.2 Object detection - Shuffled holdout on synthetic data . . . 82
5.3 Object detection - Kfold 1 on synthetic data . . . 82
5.4 Object detection - Kfold 2 on synthetic data . . . 83
5.5 Object detection - Kfold 3 on synthetic data . . . 83
5.6 Object detection - Skip 2 on synthetic data . . . 84
5.7 Object detection - Skip 5 on synthetic data . . . 84
5.8 Object detection - Skip 10 on synthetic data . . . 84
5.9 Object detection - Native split of real-world data . . . 85
5.10 Object detection - Shuffled real-world data . . . 86
5.11 Object detection - Skip 2 on the real-world data . . . 86
5.12 Object detection - Skip 5 frame on the real-world data . . . 87
5.13 Object detection - Skip 10 on the real-world data . . . 87
5.14 Categories balancing workflow . . . 88
5.15 Object detection - Oversampling Waymo skip 10 dataset . . . 89
5.16 Object detection - transfer learning from CARLA skip 10 to Waymo skip 10 . . . 90
5.17 Object detection - training on mixed CARLA skip 10 and Waymo skip 10, and evaluating against Waymo skip 10 data . . . 90
5.18 Object detection - training on CARLA skip 10 with pretrained weights from COCO . . . 91
5.19 Object detection - training on Waymo skip 10 with pretrained weights from COCO 91
5.20 Object detection - training on Waymo skip 10 with pretrained weights from CARLA and COCO . . . 92
5.21 Object detection - first fine-tuning on CARLA skip 10 without night frames with pretrained weights from COCO and then fine-tuning on Waymo skip 10 . . . 92
5.22 Object detection - training on mixed CARLA skip 10 and Waymo skip 10, with pretrained weights from COCO . . . 93
5.23 Vehicle (left) and pedestrian (right) AP on the test split from the model fine-tuned on Waymo skip 10 from COCO . . . 96
5.24 Inference sample 1 from the model fine-tuned on Waymo skip 10 from COCO . . . 96
5.25 Inference sample 2 from the model fine-tuned on Waymo skip 10 from COCO . . . 97
5.26 Inference sample 3 from the model fine-tuned on Waymo skip 10 from COCO . . . 97

5.27 Inference sample 4 from the model fine-tuned on Waymo skip 10 from COCO . . 98
5.28 Monocular depth estimation - training on CARLA from scratch. 99
5.29 Monocular depth estimation - training on Waymo from scratch 100
5.30 Monocular depth estimation - training on waymo from scratch without nocturne frames 101
5.31 Monocular depth estimation - training on Waymo from CARLA 102
5.32 Monocular depth estimation - training on mixed data from scratch 103
5.33 Monocular depth estimation - fine-tuning on CARLA from KITTI 104
5.34 Monocular depth estimation - fine-tuning on Waymo from KITTI. 105
5.35 Monocular depth estimation - fine-tuning on mixed data from KITTI 106
5.36 Monocular depth estimation - sample 1, standard view 109
5.37 Monocular depth estimation - sample 2, sunlight on camera. 109
5.38 Monocular depth estimation - sample 3, rain droplets on camera 110
5.39 Monocular depth estimation - sample 4; rain droplets on camera 110
5.40 Monocular depth estimation and object detection - sample 1 111
5.41 Monocular depth estimation and object detection - sample 2 111
5.42 Monocular depth estimation and object detection - sample 3 111

LISTA DE TABELAS

2.1 Precision x recall table 33

3.1 Comparison on synthetic datasets and simulators 44
3.2 Comparison on real-world datasets 49
3.3 mAP from 0.50 IoU to 0.95 IoU at 0.05 step on COCO test set 52
3.4 mAP for detections' difficulty easy, medium and hard on KITTI test set 53
3.5 Self-supervised and unsupervised algorithms performance on Eigen's split of KITTI 58

4.1 Hardware specification 63
4.2 Algorithms hyperparameters and requirements. 64
4.3 Objects count per town 76
4.4 Synthetic dataset annotations distribution 78

5.1 Algorithms fine-tuned from COCO and evaluated on Waymo skip 10. 93
5.2 Overall detection performance according to different data sources and targets . . 95
5.3 Overall metrics performance for monodepth2 on CARLA and WAYMO 108

LISTA DE ACRÔNIMOS

ANEEL	Brazilian Electricity Regulatory Agency
BAYLAT	Bayerisches Hochschulzentrum für Lateinamerika
CARISSMA	Center of Automotive Research on Integrated Safety Systems and Measurement Area
CARLA	Car Learning to Act
COPEL	Companhia Paranaense de Energia
DELT	Departamento de Engenharia Elétrica
FAIR	Facebook AI Research
FoV	Field of View
HOG	Histogram of Oriented Gradients
LIDAR	Light Detection and Ranging
RADAR	Radio Detection And Ranging
ReLU	Rectified linear unit
RoI	Regions of Interest
Pixel	Picture Element
SOA	State-of-the-art
SURF	Speeded-Up Robust Features
YOLO	You Only Look Once

LISTA DE SÍMBOLOS

α	A constant value
Abs rel	Absolute average error
Sq rel	Squared average error
AP	Average Precision
API	Application Program Interface
AR	Average Recall
ANN	Artificial Neural Network
BB	Bounding box
BDDV	Berkley DeepDrive Video
CNN	Convolutional Neural Networks
δ	Depth accuracy
d_i	Depth value for a specified pixel on the predicted data
d_i^*	Depth value for a specified pixel on the ground-truth data
D_t	Dense depth map
FN	False Negative
FP	False Positive
FPS	Frames per Second
GPS	Global Positioning Systems
GPU	Graphics Processing Unit
HDF5	Hierarchical Data Format
i	Selected pixel
$I_{t'}$	Source image's pose
I_t	Target image's pose
$I_{t'\rightarrow t}$	Relative pose between the source and target image's pose
IMU	Inertial Measurement Unit
IoU	Intersection over Union
K	Intrinsic matrix of the camera
L_p	Photometric Reprojection Error
LBP	Local Binary Patterns
mAP	mean Average Precision
mAR	mean Average Recall
N	Amount of depth pixels
PDF	Probability Density Function
pe	Photometric reconstruction error
PPGEE	Programa de Pós-Graduação em Engenharia Elétrica

$proj$	2D coordinates of projected depths D_t in $I_{t'}$
PWC	PricewaterhouseCoopers
R-CNN	Regions with CNN features
RGB	Red, Green and Blue
$RMSE$	Root-mean-square error
$RMSE\ log$	Log form of root-mean-square error at base 10
RPN	Region Proposal Networks
SGD	Stochastic Gradient Descent
SIFT	Scale-Invariant Feature Transform
SQL	Structured Query Language
SSD	Single shot detector
$SSIM$	Structural SIMilarity index
SVM	Support Vector Machine
t'	Previous and posterior frames from target frame
t	Target frame
THI	Technische Hochschule Ingolstadt
TN	True Negative
TP	True Positive
UAVs	Unmanned Aerial Vehicles
UFPR	Universidade Federal do Paraná
w	Neuron weight
x	Neuron input

SUMÁRIO

1 INTRODUCTION 16
1.1 OBJECTIVES 18
1.2 MOTIVATION 18
1.3 DISSERTATION STRUCTURE 18
2 THEORETICAL FOUNDATION 19
2.1 COMPUTER VISION 19
2.1.1 Digital image processing 19
2.2 MACHINE LEARNING 21
2.2.1 Model and data validation 22
2.3 ARTIFICIAL NEURAL NETWORKS 23
2.3.1 Convolutional neural networks 26
2.3.2 Transfer learning 29
2.4 METRICS 30
3 LITERATURE REVIEW 37
3.1 DATASETS 37
3.1.1 Synthetic 37
3.1.2 Real-world 44
3.2 COMPUTER VISION 49
3.2.1 Object detection 50
3.2.2 Monocular depth estimation 54
3.2.3 Transfer learning 60
4 MATERIALS AND METHODS 63
4.1 METHODOLOGY 63
4.2 MATERIALS 63
4.2.1 real-world and synthetic data 64
5 RESULTS 81
5.1 OBJECT DETECTION 81
5.1.1 Synthetic-based CARLA 81
5.1.2 Waymo 85
5.1.3 Transfer learning 89
5.2 MONOCULAR DEPTH ESTIMATION 98
5.2.1 Synthetic-based CARLA 98
5.2.2 Waymo Open 100
5.2.3 Transfer learning 101

6 CONCLUSIONS AND FUTURE WORKS 112
6.1 CONCLUSIONS 112
6.2 FUTURE WORKS 113
REFERÊNCIAS 114

1 INTRODUCTION

The automotive industry is going under constant transformations, and according to PricewaterhouseCoopers' (PWC) report [1], five points contribute the most to it: electrified, autonomous, shared, connected, and yearly updated. Electrified refers to the adoption of electrical energy as the main energy source for the vehicle, thus being more environmental-friendly since it emits less harmful substances, and is more silent. Autonomous refers to autonomous vehicles, i.e., self-driving which requires no human interference, being possible due to advances in the fields of artificial intelligence. Shared is attributed to the fact that car-sharing facilities will become economically viable with autonomous vehicles, enabling an on-demand service. Connected indicates the concept of Connected Car, i.e., Car2Car and Car2X communication, enabling the vehicle to share information with other vehicles or even transit devices. Lastly, yearly updated refers to the fact that with faster development of the previously mentioned topics, an off-the-shelf car with model cycles lasting from five to eight years will lag behind upcoming updates. Since customers won't buy new cars every year, then regular upgrades of shared vehicles will be an option for solving this. In addition to this, PWC's report calculates that, by 2030, approximately 40% of the mileage driven in Europe would be run by autonomous vehicles. This is further urged with PWC's trend for the forecasted penetration of autonomous vehicles shown in Figure 1.1 with an exponential increase of autonomous vehicles over the years. For clarity, the term "self-driven"on the chart implies a manually piloted vehicle and the ordinate axis represents the total percentage share of vehicles in Europe.

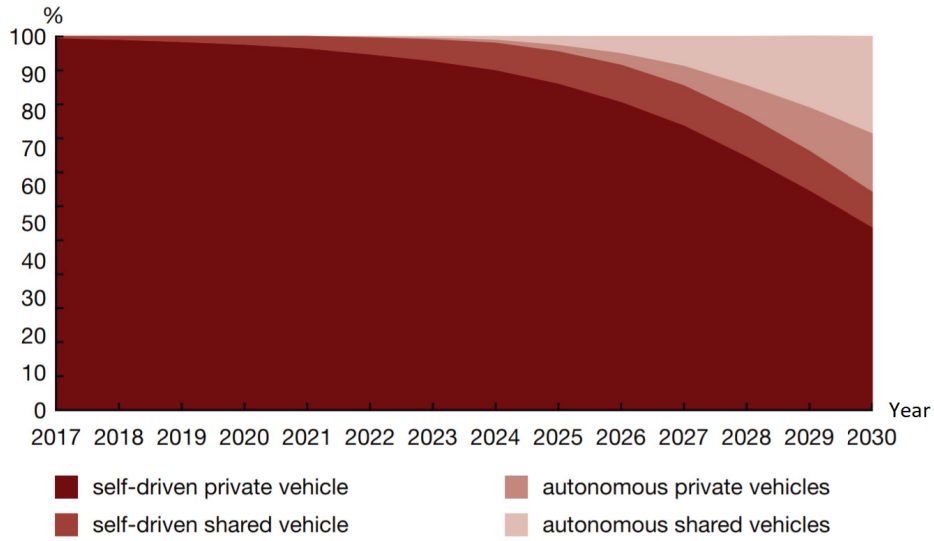


Figura 1.1: Forecast of autonomous vehicles integration in Europe
Source: [1]

In the context of autonomous vehicles, monitoring and understanding of the ambient that the vehicle is situated is a core concept, be it for navigation or safety purposes. In a safety context, having a vehicle with a system capable of identifying elements presents in a scene turns possible early adjustments or trigger on automobile safety systems, which are divided between active and passive ones [2]. Active safety systems account for those whose functionality are continuous during the vehicle's activity to avoid or lessen the risks of accidents, with examples

of it being maneuver correcting devices such as adaptive cruise control, anti-lock braking, and electronic braking systems. On the other hand, passive safety systems only activate when a vehicle crash is imminent and as such are deployed to lessen damage to the human parties involved. A few examples of implementable devices for this purpose are airbags, seat belts, and in-vehicle emergency calls [3, 4]. Correct activation of such devices is crucial since if done in a false positive or false negative-case situation, could act in favor of causing fatalities instead of diminishing them.

Common sensors deployed on vehicles for data collection and posterior algorithms development are Global Positioning Systems (GPS) and Inertial Measurement Unit (IMU) for localization and mapping, ultrasound sensors for parking, Light Detection and Ranging (LIDAR), Radio And Ranging (Radar) and cameras for navigation [3, 5, 6, 7]. Hence, one possible approach for detecting objects of interest on an ambient could be through inference on one sensor data or by performing data fusion with two or more different data source types.

On the context of images as a data source type, varied noise conditions, luminosity, contrast, occlusion and observation perspectives turn the object detection task into a complex objective, with classical computer vision techniques presenting a deficiency in performance on harsh scenarios when compared against approaches that couple computer vision with machine learning techniques. These approaches are capable of reaching state-of-the-art results on complex datasets, such as COCO [8], KITTI [6] and Cityscapes [9], thus hinting at a development opportunity on this field.

There are a plethora of state-of-the-art techniques on computer vision coupled with machine learning, such as algorithms on bounding boxes detection and image segmentation, with the state of the art techniques differentiating between themselves on the neural network architecture and methodology. Examples of bounding box detection algorithms are the Single Shot MultiBox Detector [10], Faster R-CNN [11], and YOLO [12]; for algorithms on image segmentation, Fully Convolutional Networks for Semantic Segmentation [13], U-NET [14], DenseNet [15], and Mask R-CNN [16]. For depth estimation, [17] presents an unsupervised method for monocular imaging and [18] a semi-supervised method.

The main obstacle to implement state-of-the-art techniques, which regards deep learning neural networks, is the need for a huge amount of data to correctly parametrize the model. When dealing with real-world data acquisition, a costly data collection pipeline becomes the biggest problem. As an alternative to it, the creation of synthetic data with simulators is an active line of research, which besides shifting the labor costs to software development and hardware, also enables large-scale data creation at will.

On this dissertation work, efforts are going to be concentrated on developing a methodology and algorithms to perform and further improve object detection and depth estimation with the usage of only Red, Green, and Blue (RGB) images as data for inference, but with the possibility of using additional data types to train the model. Techniques that can estimate this information, as previously shown, already exist, but still have room for improvement on the autonomous automotive field through the development and/or combination of models to increase reliability and accuracy. As an application focal point, the test ambient will be imaging captured by a dashboard camera mounted on a moving vehicle. Other approaches that can be derived from the proposed work include those aiming at people's safety by monitoring outdoor and indoor ambients, such as parks, malls and factories, and also applications in robotics.

1.1 OBJECTIVES

The main objective of the dissertation is to verify the knowledge transferability between synthetic and real-world automotive datasets, with a focus on object detection and monocular depth estimation algorithms.

Specific objectives are:

- Assembling of synthetic and real-world automotive datasets;
- Analysis of object detection and depth estimation state-of-the-art (SOA) approaches performed on data within the automotive context;
- Performance analysis of transfer learning methods in both synthetic and real-world data.

1.2 MOTIVATION

Human life holds immeasurable value. Unfortunately, many are still forfeit on road traffic fatalities; in 2019, 22800 cases were present on 27 Europe Union member states [19]. When new technologies are introduced to society such as autonomous vehicles, risks to humans are an inherent part of the transitioning process, be it due to unforeseen technical challenges or difficulties. Since one of the ways autonomous vehicles sense its surrounding is through cameras, then one approach to promote human safety is by researching computer vision techniques that may help the vehicle to better understand the context it is in. For that, SOA methods rely on huge amounts of data, which are often costly to acquire, with an alternative being the addition of synthetic data. As such, in this dissertation, the main contribution is an exploratory analysis on which factors influence object detection and monocular depth estimation algorithms when synthetic and real-world data are compounded. Along with it, the created synthetic dataset collection tool is made open-source to the community.

1.3 DISSERTATION STRUCTURE

The dissertation is structured as follows. In the first section, an introduction to the work was presented, with its context and main objectives. In the second section, fundamental concepts on computer vision and machine learning are presented to grasp the ideas of the algorithms applied in the dissertation. In the third section, a literature review on datasets and computer vision techniques coupled with machine learning is presented. In the fourth section, the materials and methodology to replicate this dissertation's work are presented, which include the data treatment and algorithms hyperparameters. In the fifth section, simulation results are presented. Finally, in the sixth section, the dissertation is concluded, with additional suggestions for future works.

2 THEORETICAL FOUNDATION

In this section, basic computer vision and machine learning concepts are shown, which help on the overall understanding of the algorithms applied to this dissertation work.

2.1 COMPUTER VISION

Computer vision is the field of study where image related tasks seek to be automated. Even though humans can effortlessly understand the context of a three-dimensional world based on two-dimensional pictures, automating this perception is not trivial since it often relies on insufficient information to perform inference [20]. Even though this field of study is not yet fully solved in every area, it is applied in many applications, such as optical character recognition, machine parts inspection, retail, photogrammetry, medical imaging, and automotive safety [20]. Common issues which rise the difficulty in solving this task are variability in illumination, pose, intra-class, and presence of occlusion, with examples of high diversity of chairs presented in Figure 2.1.

Figura 2.1: Chair examples
Source: The author (2020), with base images from pixelsquid.com

2.1.1 Digital image processing

Digital image processing refers to actions capable of being performed on two-dimensional data arrays, with common uses being image representation and modeling, enhancement, restoration, reconstruction, data compression, and image analysis. The first concerns on discerning what type of information does each pixel (picture element) or group of pixels on the image represents. Normally, this information is of object luminance, but could also be of radar cross-section, temperature and others. Regarding image enhancement, the objective is to emphasize certain image features, such as edges sharpening and color distribution handling for further image analysis. Image restoration is the process of minimizing data degradation, generally caused due to sensor and ambient restrictions. Image reconstruction deals with assembling a two or higher dimensional array from multiple one-dimensional data, such as on

x-ray projections. Another topic is on data compression, which cares for the occupied data volume and the related methods to reduce the number of bits to represent the same image with the less possible loss. Lastly, image analysis's focus is on investigating the image and inferring quantitative measurements from it through extraction and classification of image features [21] and is therefore the context of the present dissertation work.

The object detection task is divided into two parts: first finding image patches that contain possible objects and then classifying them. In image processing, features can be either global or local, with the former extracting information from the image as a whole and the latter identifying possible points and regions of interest on the image [22].

Classical feature detectors include, but are not limited to, LBP (Local Binary Patterns), HOG (Histogram of Oriented Gradients), SIFT (Scale-Invariant Feature Transform) and Speeded-Up Robust Features (SURF). All of them include some kind of image filter, with its general functionality shown in Figure 2.2. The main idea is to pass a kernel on top of the image which performs element-wise multiplications and then returns the sum of it to a pixel. Certain values and sizes of the filter highlight/modify different features of the image, such as borders, texture, noise shrinkage, and blurring.

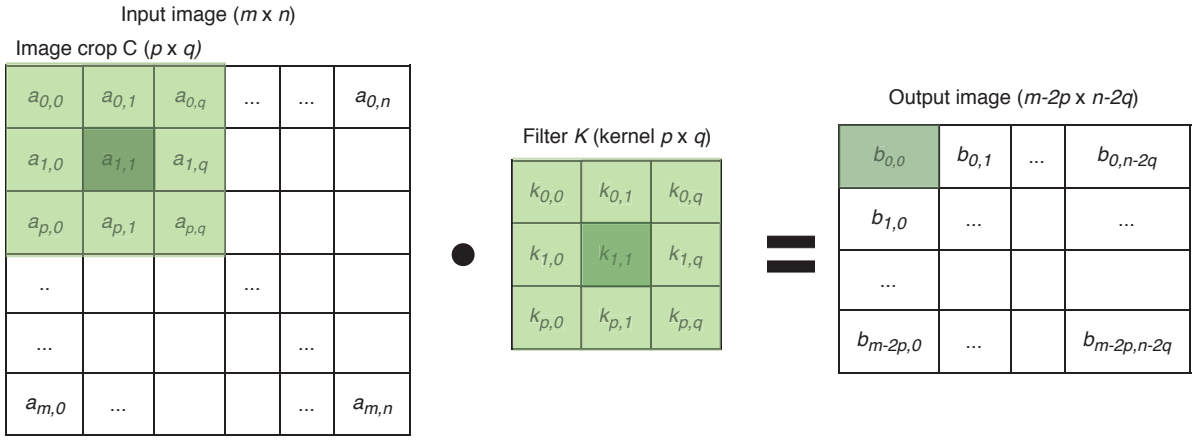


Figura 2.2: Image filter basic functionality
Source: The author (2020)

LBP is a local feature descriptor with, to the best of the author's knowledge, earliest appearance on 1990 by [23], and later made popular by [24] in 2002. The main idea with this approach is to identify texture patterns by passing a kernel on top of the image with activation locations being defined as a symmetric circular region around its center. When the border value is greater than the center value, then that specific filter location is activated. After computing it for every position on the circular region, then the resulting pixel value is calculated by multiplying them by a fixed pattern. The output filtered image serves as a feature descriptor, which can be used as input for other classifiers.

SIFT [25] is a local feature descriptor proposed by David Lowe in 1999, which is based mainly on analyzing the gradient between pixels, and is therefore invariant to scaling, translation, and rotation. In short, it generates SIFT key features for the whole image, groups similar keys through a Hough transform to generate a knowledge base with common keys for a certain class. Akin to SIFT, SURF [26] also works with a method invariant to scale and rotation by applying a simplified second-order derivative on the image pixels through the usage of 9x9 box filters. This way, since no derivatives are performed, this algorithm presents a higher processing speed than SIFT.

One type of global feature descriptor that is widely applied in this context is the Histogram of Oriented Gradients (HOG). Its concept was first patented by Robert McConnell [27] in 1982 and later made popular by Dalal and Triggs [28] in 2005. The main idea behind HOG is on relying the object classification task not only on the contour feature of the object but also on its direction. This is done by computing gradients of pixel color variation over patches on the image and then accounting histograms with these data. Afterward, this feature is fed into a classifier which decides if this histogram set belongs to a person or not.

2.2 MACHINE LEARNING

One difficulty that arises with manually fine-tuning global and local feature descriptors is on their capacity of generalization. To aid with this, state-of-the-art techniques rely heavily on machine learning methods, which when given enough training data, are capable of generalizing to a wide variety of contexts. In the context of computer vision, instead of manually handcrafting adequate filters as previously mentioned, the algorithm adapts those parameters according to patterns found in the data.

Even though various sources cite machine learning's definition with Samuel's quote as the *"Field of study that gives computers the ability to learn without being explicitly programmed"*, to the best of the author's knowledge this exact citation has not appeared in any paper of his. The machine learning term itself had it's first appearances back in 1959 with Arthur Samuel's checkers game paper [29], where the program adapted itself to play the game by challenging other human players. Since then, the field has been expanding, with applications ranging from the pharmaceutical industry to self-driving cars, and others.

Machine learning consists of three main approaches: supervised, unsupervised, and reinforcement learning, with a simple representation of each shown in Figure 2.3.

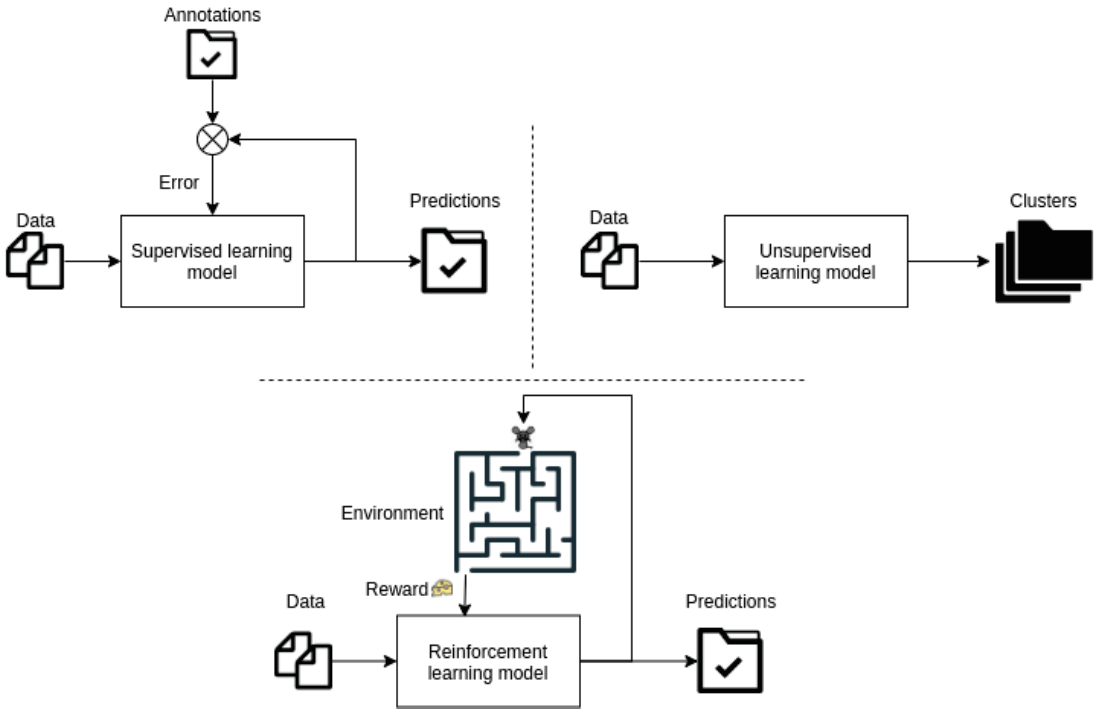


Figura 2.3: Machine learning general pipeline for each learning approach
Source: The author (2020)

The main difference between each approach is how the data is interpreted by the algorithm. For supervised learning, the data is annotated and the algorithm learns how to map the input data to the annotations, whereas in unsupervised learning the data is not annotated, so the algorithm learns to map groups according to its distribution. On reinforcement learning, the data is also not annotated, but the algorithm's environment manages to assign rewards according to its performance, and so it learns how to take optimal decisions by trial and error [30].

2.2.1 Model and data validation

A fundamental step in the pipeline of supervised learning when dealing with data consists of splitting it into three distinct sets - training, validation, and testing. The main idea is to enable the algorithm to adapt it's parameters according to the training data, while at the same time being able to generalize for cases other than those in the training set. For this, the validation set is not used to change the algorithm's parameters, but to continuously be evaluated alongside the training set. This way, a hint on the model's generalization for external data can be assumed should it improve on both the training and validation sets. This elucidation is presented in Figure 2.4, where the model is underfitting before the dashed vertical line and overfitting after it. Finally, the test set serves as the last performance assessment, being the most unbiased possible, and is usually the metric reported in the literature. It is to be noted, though, that depending on the difficulty of assembling a large amount of labeled data on a specific field, it is not unusual for the results to be reported only on the validation set.

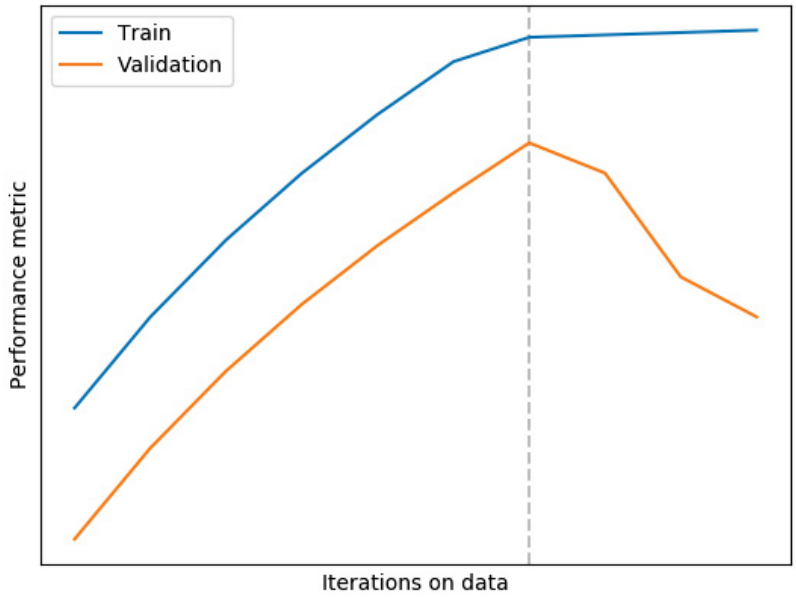


Figura 2.4: Model training behavior
Source: The author (2020)

This method of evaluation is named hold-out and is one of the many cross-validation methods in the literature, which include leave-one-out, leave-p-out, v-fold (also called k-fold), Monte-Carlo, and others [31]. On leave-one-out, each data sample is separated one at a time for validation against a model trained on the rest of the data until every sample has been used for

validation. Leave-p-out does the same but with a set of p (2 or more) validation data points. One problem that arises with these exhaustive methods is the extensive computation requirement, since a new training session must be done for each iteration, thus leading it to infeasibility on most cases, where in non-exhaustive methods not all combination of the data splits are analyzed, which presents then a computational advantage. On this line, in k-fold the dataset is split into k equal partitions, where $k-1$ splits are used for training and one for validation, with an example shown in Figure 2.5 where the number of folds k is 5. In this example, the model is trained 5 times with each fold splitting the data in 80-20% splits, where the validation is always unique between folds. This way, the model is evaluated k times against every partition of the dataset and the performance is accounted as the average between validation of all folds. Lastly, Monte Carlo is a method similar to k-fold, but the folding procedure is repeated for random partitions of the data several times.

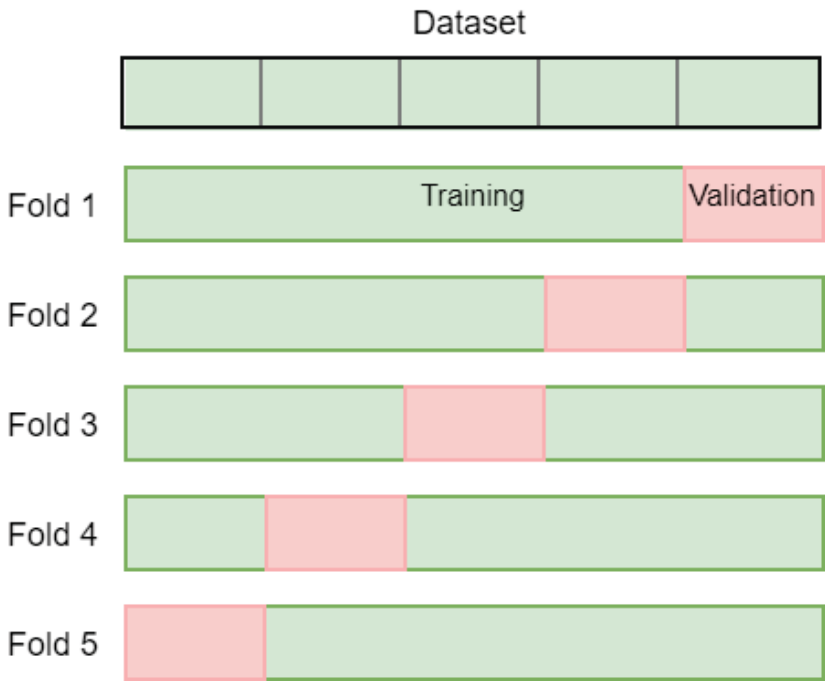


Figura 2.5: k-fold cross validation with k=5
Source: The author (2020)

2.3 ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANNs) are one of the basic foundations for state-of-the-art applications in machine learning and computer vision since they have shown their potential on assembling models capable of solving difficult tasks. For the sake of simplicity, the term neural networks is used interchangeably with ANNs on this dissertation. At its core, ANNs are a set of connected perceptrons, also called neurons, which enable a series of sum, product and activation function calculations. By feeding data in this model, it is capable of grasping its pattern and adjusting its hyperparameters accordingly. With this concept in mind, different arrangements of neurons in ANNs are possible, with denser ones being capable of solving more complex problems, although being at more risk of overfitting. Examples of a perceptron and a feedforward fully connected ANN are presented in Figures 2.6 and 2.7. For simplification, x_1 to x_4 represent the input data and x_5 to x_7 the perceptron model from Figure 2.6.

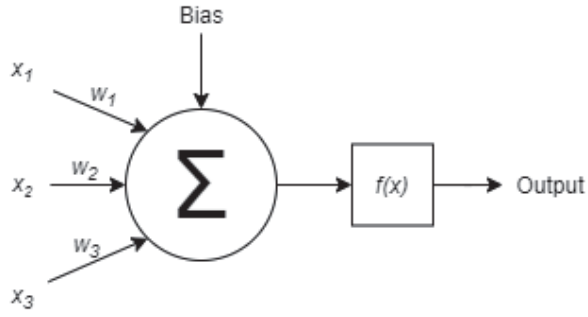


Figura 2.6: Perceptron
Source: Adapted from [32]

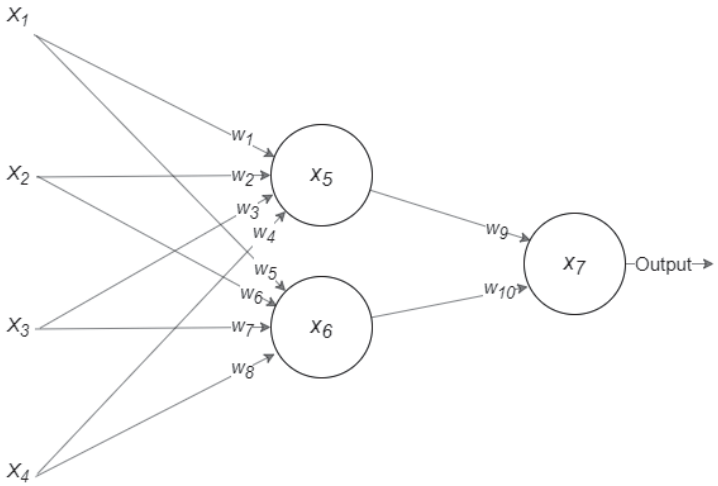
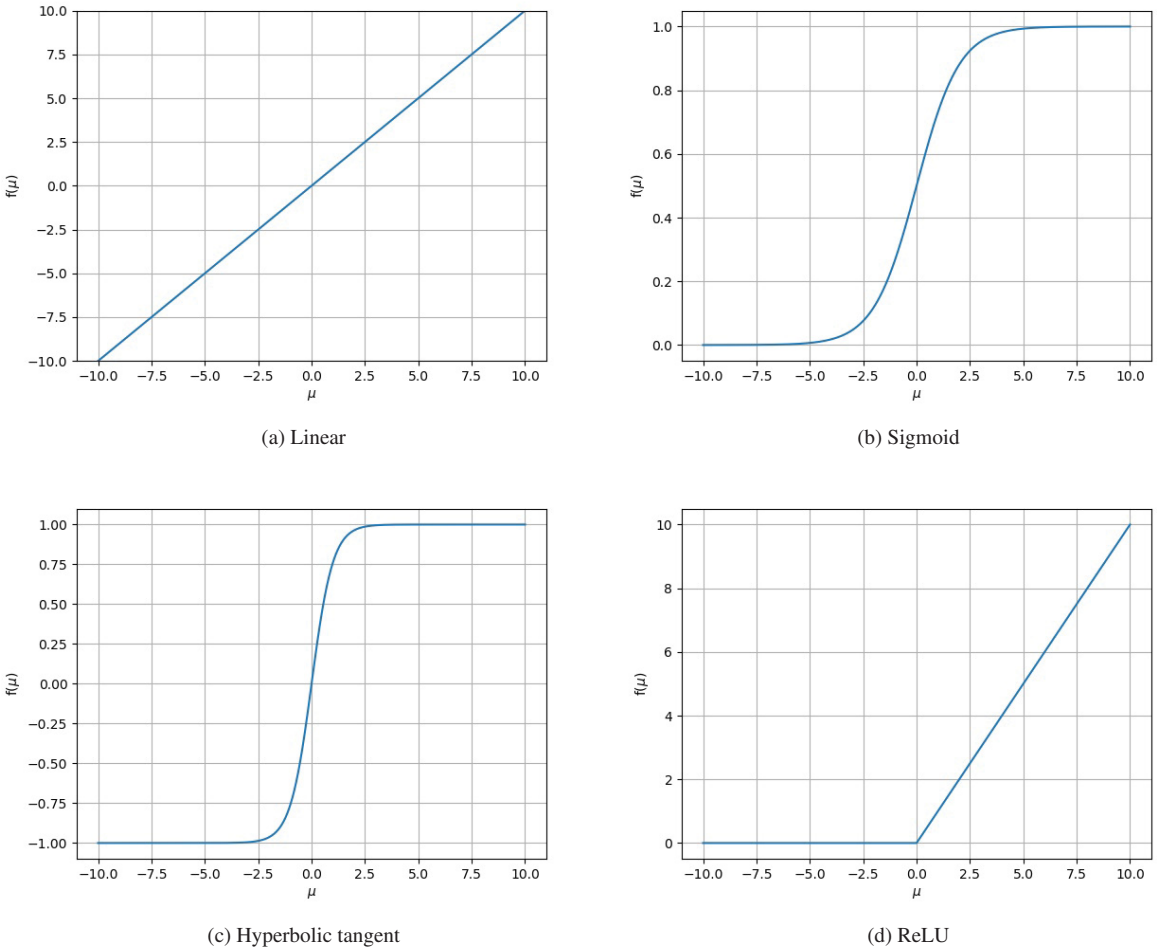


Figura 2.7: Fully connected feedforward ANN
Source: The author (2020)

On a high level, a perceptron can be interpreted as a function block where a set of one or more numerical input is transformed into a single numerical output. The general equation for it is presented on Equation 2.1, where x_i is the i input to the neuron, w_i the weight for that respective connection, *bias* an offset value, $f(x)$ the activation function for the neuron and *output* the output value for the neuron. The main idea for a neuron is for it to have a fixed activation function $f(x)$ and weights w_i and *bias* with adjustable values according to an error calculation when the real desired output value is known.

$$output = f(\sum_{i=1}^{n} x_i * w_i + bias) \tag{2.1}$$

Common activation functions are presented in Figure 2.8, which include the linear, sigmoid, hyperbolic tangent, and rectified linear unit (ReLU) activation functions. The main differences between each activation function are on their output range being negative or not, presence of non-linearity, and profile near the zero mark. In the computer vision context, ReLU and variants of it are the ones most commonly applied due to its computational efficiency and capability of performing derivatives. One last function that is also worth mentioning is the softmax, commonly applied on the last layer of classifier ANNs to infer score confidences for each class, such that the sum of all scores is 1.



(a) Linear

(b) Sigmoid

(c) Hyperbolic tangent

(d) ReLU

Figura 2.8: Activation functions
Source: The author (2020)

An ANN is divided into layers, which constitutes visually of neurons allocated in the same column. In Figure 2.7 there are three layers present: the first is the input layer, the middle one is the hidden layer and the last is the output layer. The connections between the neurons can be as of fully connected, where each neuron in the i layer is connected to the next $i+1$ layer, which results in more dense and computationally expensive calculations. There are operations, such as dropout [33], which tries to deal with this by randomly excluding neurons and connections during training since it has been observed that in a dense ANN the presence of too many connections has little positive impact on the model's inference, thus causing calculation overhead.

In ANNs, neuron weights and biases are the core values that are adjusted so that the model reacts as expected from it, and for that to happen, loss functions have to be designed, which vary according to each specific problem scope. The loss is the difference between the expected and predicted outcome, whose value is given as a feedback to each neuron in the model for it to adjust itself. At its heart, this whole process is one kind of indirect optimization [34] since the cost function is not the final goal itself, but it is reduced so that another performance measured is improved. For this to happen efficiently, backpropagation is needed.

Error backpropagation is a technique made famous in 1986 [35] and is a series of chain rule derivatives performed from inputting the error to the activation functions of the output layer up to the first hidden layer. The general equation for it is presented on Eq. 2.2 [36], where

θ are the model's set of weights and biases; t the training iteration; α the learning rate, and E each respective error function. Both the training iteration and learning rate are considered hyperparameters for the model, with optimal values being assigned by experimentation. Other optimizers include the Stochastic Gradient Descent (SGD) and Adam, with the main differences between them being on applying tricks to help convergence, such as varying the learning rate according to the training stage, accumulating the gradients, and use of momentum.

$$\theta^{t+1} = \theta^t - \alpha * \frac{\delta E(X, \theta^t)}{\delta \theta} \tag{2.2}$$

As a quick addendum, when two or more hidden layers are present, the ANN is commonly referred to as deep ANN, where its operations and the model itself starts to become harder to trace and understand the bigger they get, in other words, a black box. On that, the deep learning topic is arisen, which according to Goodfellow [34], is the concept of enabling a computer to learn based on a knowledge hierarchy, i.e., to build a complex concept based on a series of simpler ones.

Another relevant topic of deep learning is the transfer learning aspect. According to Pan [37], common sense dictates that models trained on a particular dataset should be feasible only for data similar to its domain. However, by "transferring"knowledge from one model to another, even if its application domain is slightly different, the necessary amount of epochs to achieve an ideal configuration of weights and biases might be reduced. Another side advantage of performing transfer learning is that less training data would be needed. The concept of transfer learning is further explained in subsection 2.3.2.

2.3.1 Convolutional neural networks

When dealing with computer vision challenges, input data are usually two-dimensional integer arrays with 3 color channels, which implies in heavy algorithm computation were conventional dense ANNs applied. To make it more efficient, CNNs are used, which rely on switching ANN's dense layers for convolutional ones, whose behavior is similar to that of the filters on section 2.1.1 of this dissertation, where a kernel traverses and multiplies the whole input image, thus resulting in a convolved feature. Besides easing computation, CNNs present three main characteristics/advantages: sparse connectivity, parameter sharing, and equivariant representations [34].

The convolution operation is formally expressed by Equations 2.3 [38] and 2.4 [34]:

$$s(t) = \int_{-\infty}^{\infty} x(a)y(t-a)da \tag{2.3}$$

$$s(t) = (x * y)(t) \tag{2.4}$$

Where x and y are functions dependent of t and a, and $*$ the convolutional operator. In the context of image processing, x is the input image, y a kernel, and $s(t)$ the feature map with two or more dimensions [34].

Sparse connectivity implies the kernel having a smaller size than the input image, thus not all connections of one layer to another are connected. On parameter sharing, the idea is that since the same kernel is traversed through the image, then these weights are applied multiple times, contrary to ANNs where each weight is multiplied only for its specific neuron. Since the kernel output is several times smaller than the input image, then memory storage requirements also become smaller. On equivariant representations, since the convolution function itself is

equivariant, then any translation operations in the input image result directly on to the output image [34].

An example of the convolution operation performed in images is presented in Figure 2.12. Basically, on its first layer, for each channel that the input data has, a kernel goes through the whole array and outputs the cumulative results to the convolved features, which are not necessarily limited to the same number of channels from the input data. Common hyperparameters for CNNs include stride, which represents the displacement length of the kernel, padding, which is basically how many/if out of bound pixels are to be considered, and dilation, which dictates how sparse should the pixels acquisition from the input image be. An example of the convolution operation with stride 1 is shown in Figure 2.9.

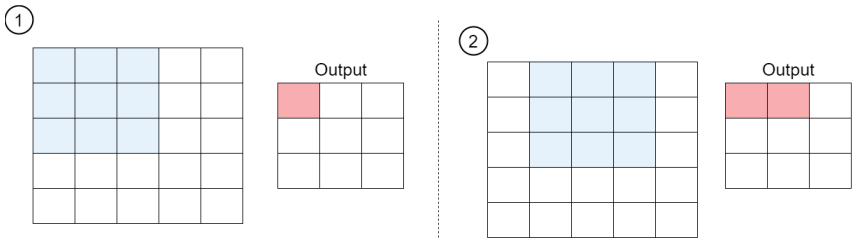


Figura 2.9: Stride with value 1
Source: Adapted from [39]

It is to be noted in this example that the center of the kernel does not hop in every cell of the input array, resulting in smaller output size. At first, it might not be a problem, but after performing several convolutions, then the array size might be too small to carry information. To prevent this, the padding operation is one solution, as shown in Figure 2.10, where the semi-transparent cells represent the added rows and columns. The main benefits are the perpetuation of the input size and consideration of information on the borders as well.

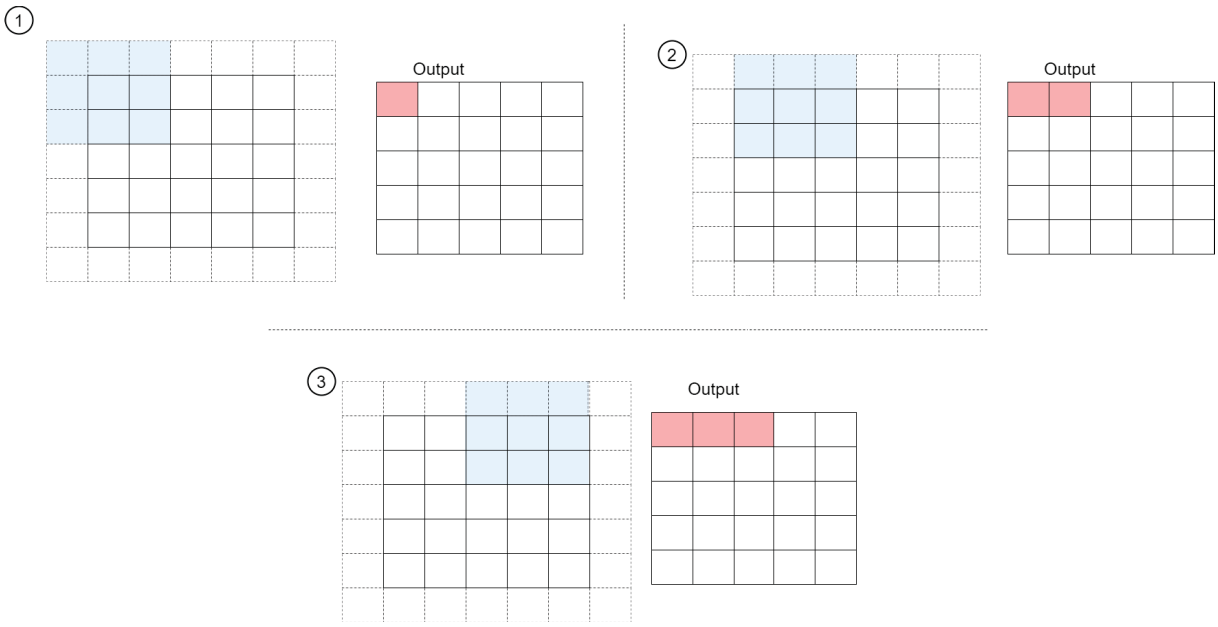


Figura 2.10: Stride and padding with value 1
Source: Adapted from [39]

The third operation, dilation, is used mainly for saving computation resources since it manages to sparsely navigate a bigger input array while maintaining a smaller output array size. An operation with a dilation rate of value 2 is shown in Figure 2.11.

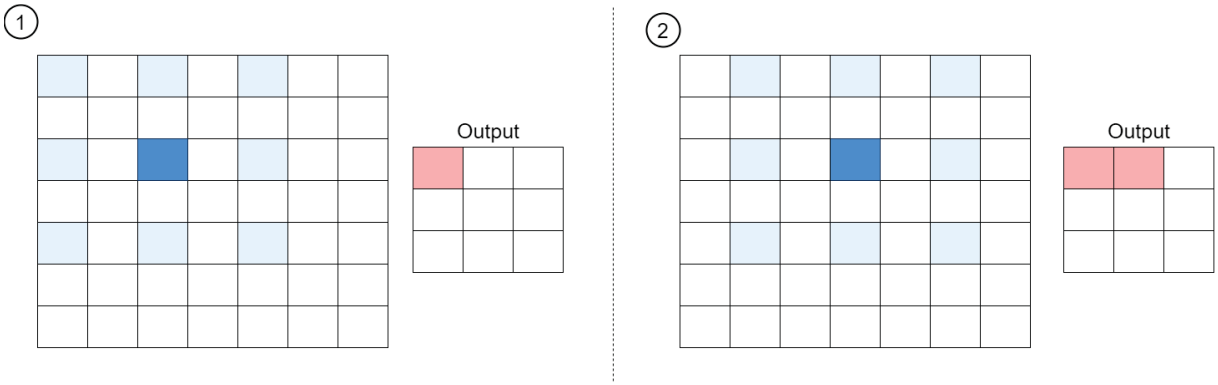
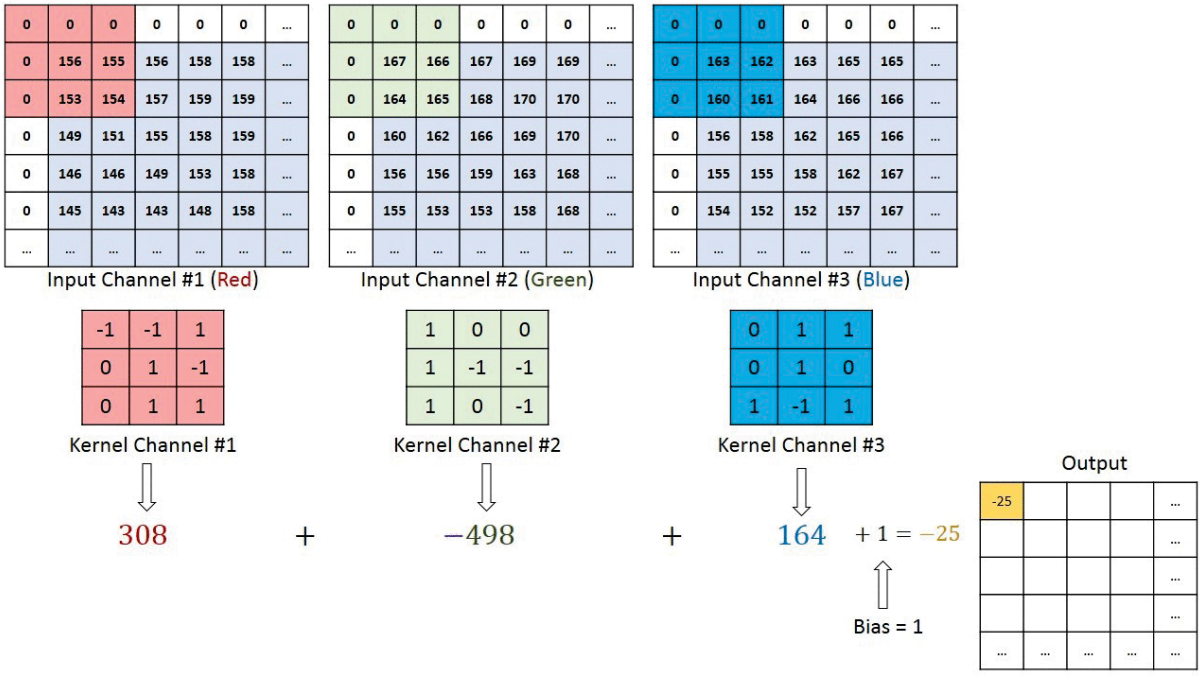
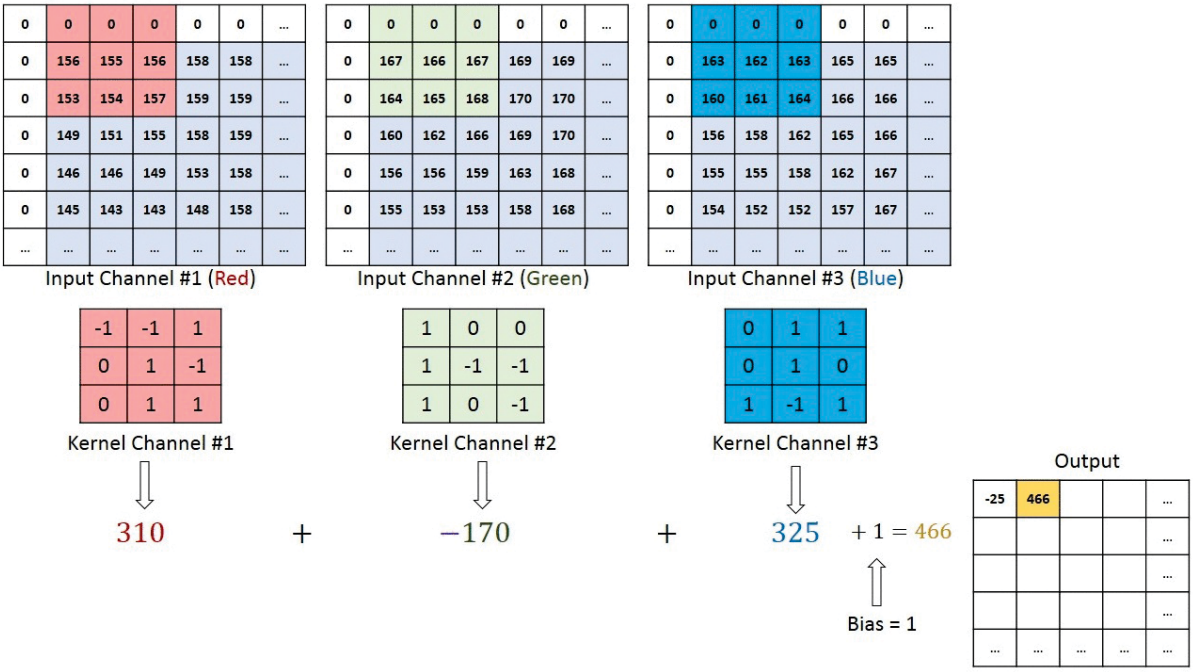


Figura 2.11: Stride and dilation rate with value 1
Source: Adapted from [39]

Finally, examples of the convolution operation on an RGB image are shown in Figure 2.12, with stride, padding, and dilation rate with a value of 1.



(a)



(b)

Figura 2.12: Convolution operation exemplified. (a) Convolution step 1; (b) Step 2
Source: [40]

One last thing worth of citation is the operation of pooling, which is, once again, the act of passing a kernel through the image. After applying the convolutional operations, various types of pooling (e.g. max pooling, average pooling) are often performed to reduce the convolved array's size, extract it's most relevant features and also make it invariant to translations [34, 40]. Visual examples of the operations of max-pooling and average pooling are shown in Figure 2.13.

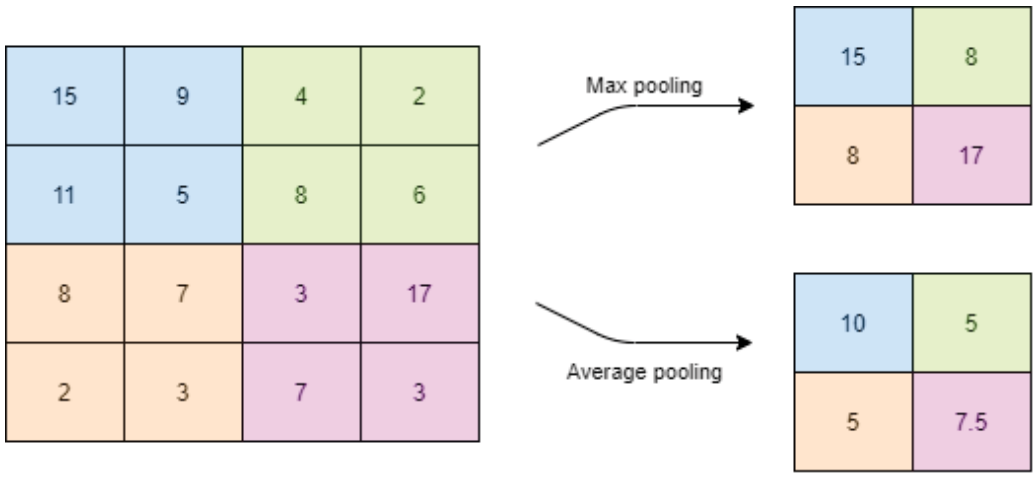


Figura 2.13: Pooling operations illustrated
Source: Adapted from [41]

2.3.2 Transfer learning

Transfer learning, according to Pan et al. [42], is defined as a technique which helps improve the learning task of a target predictive function situated in the target domain, based on knowledge from another source domain and learning task. To further explain its concept,

transfer learning can be categorized as inductive, transductive or unsupervised, according to how the source/target learning task and domains relate as shown in Figure 2.14 with the last rows in Figure 2.14 showing the areas in which each case of transfer learning falls into. According to [42], there is no conclusive theory on how to predict if applying transfer learning will benefit or not, with most works relying on empirical methods to test it, by evaluating the similarity between the source and target tasks and by grouping tasks which could have similar parameters. Therefore, it's relevant to note that there is always the negative transfer possibility, where the source domain data and task reduce performance in the target domain.

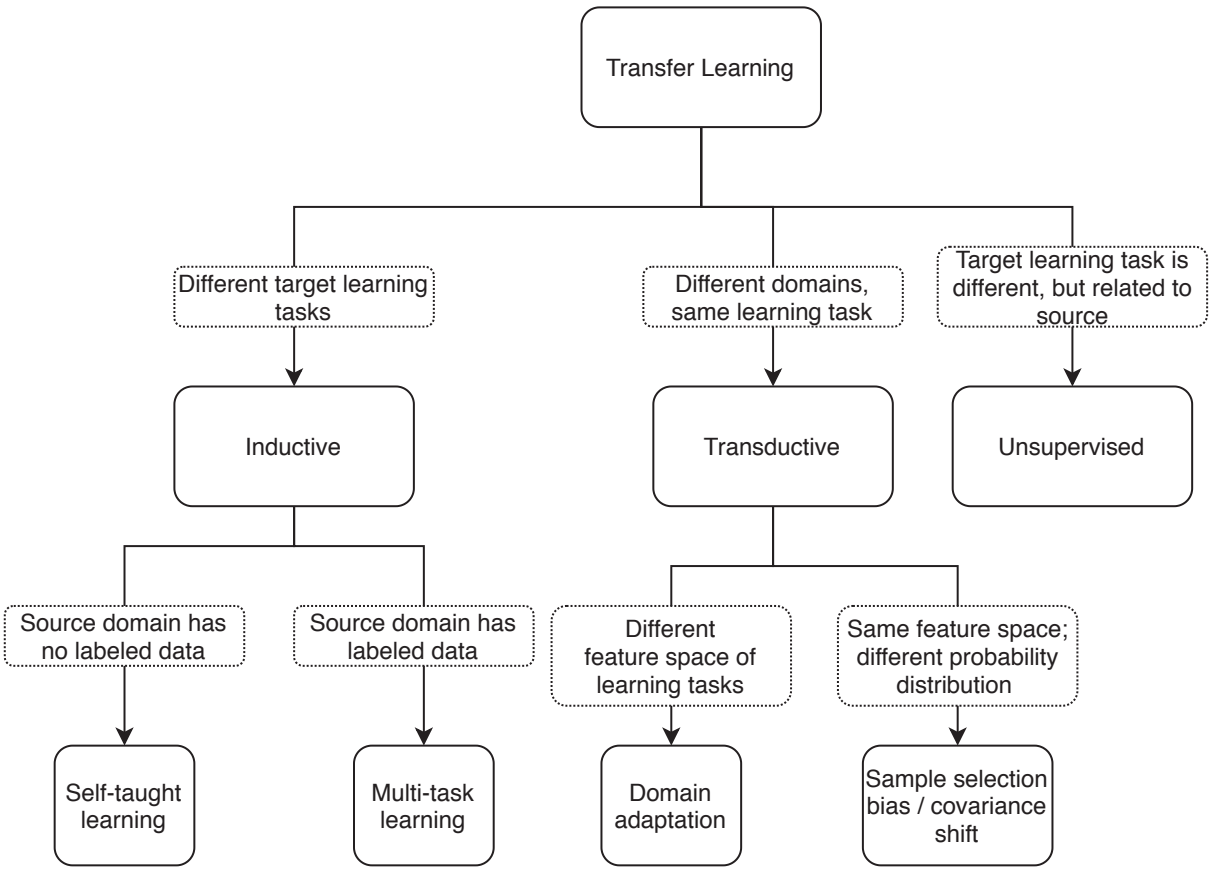


Figura 2.14: Transfer learning categories
Source: Adapted from [42]

2.4 METRICS

In this section, performance criteria for evaluation of 2D object detection and monocular depth estimation are presented. According to [43], predictions in classification problems can be classified as of four types, where:

- True positive (TP) is when the instance is positive and was correctly classified as a positive instance;
- True negative (TN) is when the instance is negative and was correctly classified as a negative instance;
- False positive (FP) is when the instance is positive but was incorrectly classified as a negative instance;

- False negative (FN) is when the instance is negative but was incorrectly classified as a positive instance.

With these assignments, building a confusion matrix is possible, be it for single-class, as shown in Figure 2.15, or multiclass problems as in Figure 2.16, where one can rapidly assess algorithm performance. Both binary and multiclass classification share the core metrics, with the main difference regarding the method for computing the four prediction types. On binary classification it is straightforward; for multiclass, however, the counting depends on which class is being analyzed. On the example of Figure 2.16, if the class b is chosen for analysis, then the TP count is only related to the row and column of b, while TN is the TP count of all other classes. Likewise, only FN and FP regarding the rows and columns of class b are considered.

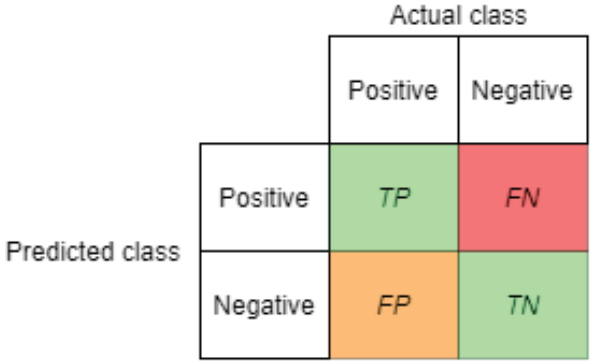


Figura 2.15: Single-class confusion matrix structure
Source: The author (2020)

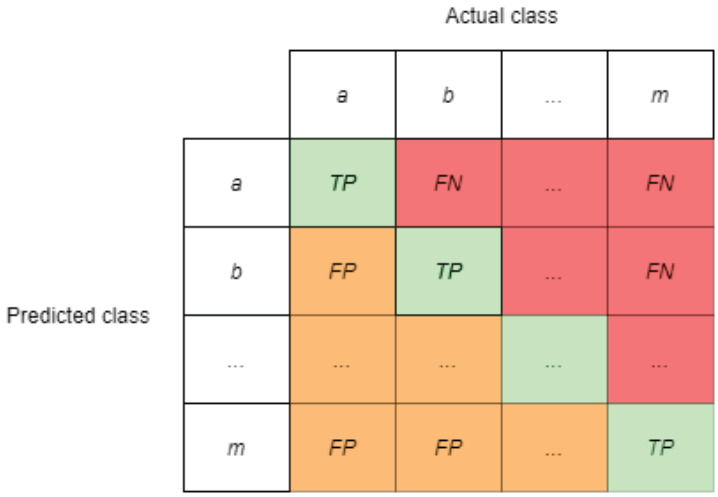


Figura 2.16: Multiclass confusion matrix structure
Source: The author (2020)

Still, according to [43], more metrics can be derived from the four classification types, such as accuracy, error rate, sensitivity, specificity, precision, recall, and F-measure.

Accuracy measures how many of the predicted instances were correct over the total, as shown in Eq. 2.5, while the error rate represents its opposite as shown in Eq. 2.6.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.5}$$

$$Error\ rate = \frac{FP+FN}{TP+TN+FP+FN} = 1 - Accuracy \tag{2.6}$$

Precision and recall, the latter also termed sensitivity, are two metrics commonly used in classification problems. Precision measures how many of the predicted positive instances were correct when compared with the total of predictions, while recall measures how many of the total positive instances were predicted when compared to the total of positive instances. They are computed by Eqs. 2.7 and 2.8.

$$Precision = \frac{TP}{TP+FP} \tag{2.7}$$

$$Recall = \frac{TP}{TP+FN} \tag{2.8}$$

Regarding object detection, a metric for measuring how well does the predicted object matches the ground-truth is the Intersection over Union (IoU). The equation for it is presented on Eq. 2.9, being the division amidst the area of intersection between each prediction area and the union of both. For further clarity, those areas are presented in Fig. 2.17. When both the ground-truth and prediction present close coordinates, then this value approaches to 1; when far from each other, then it approaches 0. Hence, the closer to 1, then the better is the performance of the algorithm.

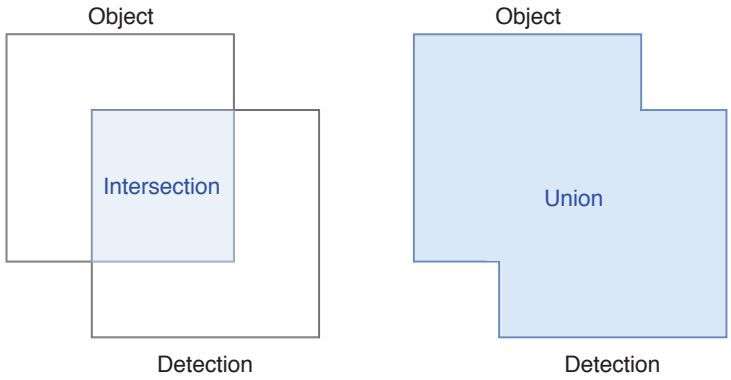


Figura 2.17: Areas of intersection and union example
Source: The author (2019)

$$IoU = \frac{Area\ of\ intersection}{Area\ of\ union} \tag{2.9}$$

Besides IoU, mean Average Precision (mAP) and Average Precision (AP) are metrics for evaluating how good are the suggested detections by the algorithm. The main difference between mAP and AP is that the first is considers all classes of a dataset, while the second is specific for a class. To compute the mAP, the steps are:

1. Count all ground-truth objects of the dataset;
2. Perform inference on each image;
3. Sort all dataset inferences from highest to lowest confidence;
4. Define IoU threshold;

5. Compute accumulated TP, FP, and FN for each prediction;
6. Compute accumulated precision and recall for each prediction;
7. Plot precision x recall curve;
8. (Optional) Smoothen precision by eliminating valleys;
9. Calculate *mAP* by computing integral of precision x recall curve.

Steps 1 through 3 are self-explanatory; as for the 5th and 6th step, the interpretation of TP, TN, FP, and FN for object detection are as follows:

- TP - the predicted detection is of the same class of the ground-truth and has an IoU higher than a set threshold;
- TN - the number of predicted detections that were not supposed to be detected (e.g. background) and were not detected. Since there are many possibilities for this situation, it is not applied;
- FP - either situation: the predicted detection has an IoU higher than the threshold but the class is different from the ground-truth; the predicted detection has the same class as the ground-truth, however, has an IoU lower than the threshold;
- FN - ground-truth instances that were not predicted.

With this, a table with the information from steps 1-6 can be assembled, with an example shown in Table 2.1. In this example, on the whole dataset, there are 5 labeled objects in the ground-truth data and the detector found 10 instances, which were correctly classified in rows 1, 2, 6, 7, and 10.

Tabela 2.1: Precision x recall table
Source: The author (2020)

idx	Confidence	TP	FP	FN	Precision	Recall
1	0.9995	1	0	4	1.00	0.20
2	0.9950	2	0	3	1.00	0.40
3	0.8000	2	1	3	0.67	0.40
4	0.7500	2	2	3	0.50	0.40
5	0.6000	2	3	3	0.40	0.40
6	0.5800	3	3	2	0.50	0.60
7	0.5500	4	3	1	0.57	0.80
8	0.3500	4	4	1	0.50	0.80
9	0.2000	4	5	1	0.44	0.80
10	0.1000	5	5	0	0.50	1.00

After having the accumulated precision and recall values for each inference, according to step 7, one can plot its chart, as shown in Figure 2.18, or on step 8 its smoothened version, shown in Figure 2.19.

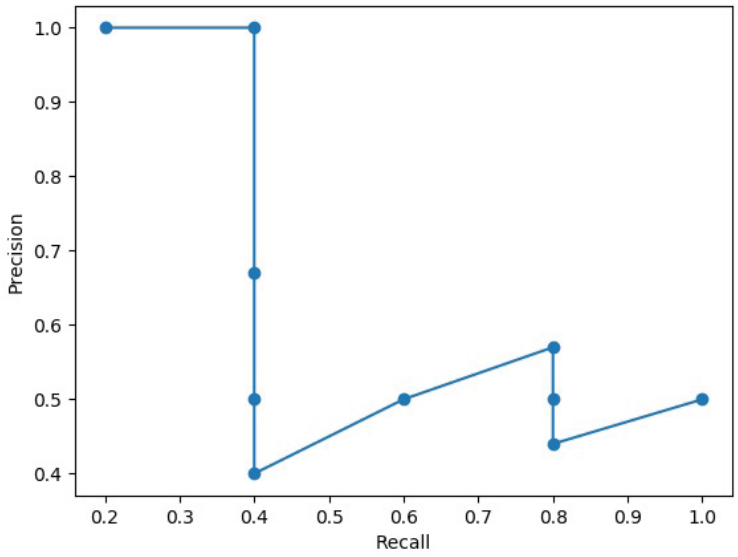


Figura 2.18: Precision x recall chart example
Source: The author (2020)

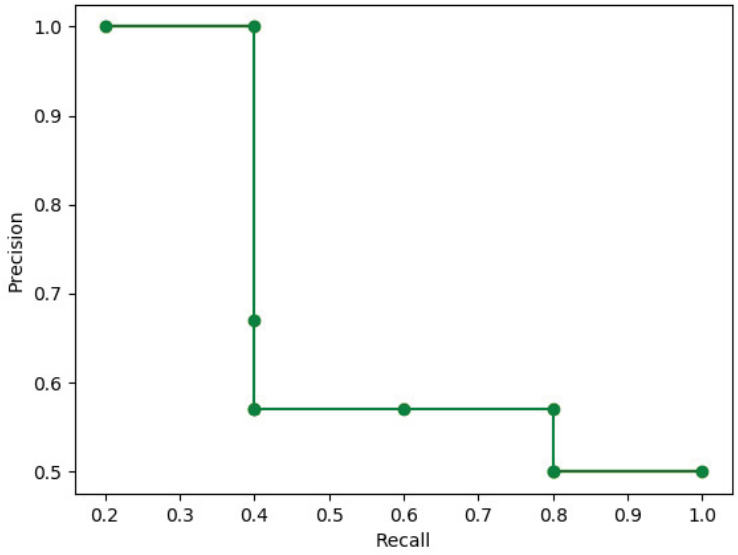


Figura 2.19: Precision x recall smoothed chart example
Source: The author (2020)

Finally, the mAP or AP is computed by calculating the area beneath the precision x recall curve. This way, the more mAP approaches 1, then the better is the performance of the algorithm. When there are multiple images and thus more predictions, then the curve becomes softer, with an example presented in Figure 2.20, where the gray area beneath the curve represents the mAP.

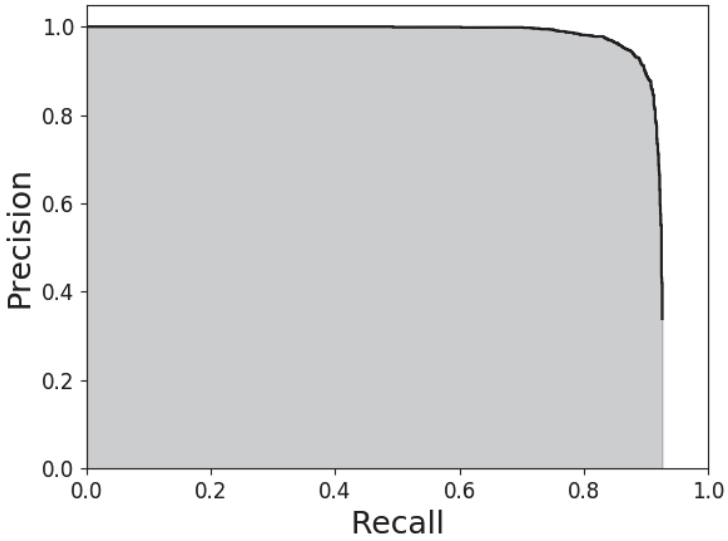


Figura 2.20: Precision x recall chart example
Source: The author (2019)

Likewise, mean Average Recall (mAR) is a metric derived from Average Recall (AR), but for computing the maximum recall given a fixed number of detections for each image and averaged over different classes and IoU thresholds.

For object detection tasks, COCO dataset [44] uses as evaluation metric mAP by fixing the threshold for IoU from 50% to 95% at a 5% step, mAR given 1, 10 or 100 detections per image and mAP and mAR across scales, where scales are referred to the size of the objects to be detected. The KITTI dataset [6] on the other hand uses as evaluation the AP with an IoU threshold of 70% for the vehicle category and 50% for the pedestrian category.

Regarding depth estimation, absolute average error (*Abs rel*), squared average error (*Sq rel*), root-mean-square error (*RMSE*) and *RMSE* with log at base 10 (*RMSE log*) are common metrics used for evaluation, where the smaller their values, the better the performance of the algorithm [45, 46]. The average error is the mean difference in depth for each pixel value on the image between the ground-truth and the predicted, presented in Eq. 2.10, where:

- **N** is the amount of depth pixels;
- **i** is the selected pixel;
- $\mathbf{d_i}$ is the depth value for a specified pixel on the predicted data;
- $\mathbf{d_i^*}$ is the depth value for a specified pixel on the ground-truth data.

$$Abs\ rel = \frac{1}{N} * \sum_{i \in N} \frac{|d_i - d_i^*|}{d_i^*} \tag{2.10}$$

The squared average error, shown on Eq. 2.11, follows the same basis, but with a norm applied to it. On Eqs. 2.12 and 2.13 RMSE is presented in both its linear and log form.

$$Sq\ rel = \frac{1}{N} * \sum_{i \in N} \frac{||d_i - d_i^*||^2}{d_i^*} \tag{2.11}$$

$$RMSE = \sqrt{\frac{1}{N} * \sum_{i \in N} ||d_i - d_i^*||^2} \tag{2.12}$$

$$RMSE\ log = \frac{1}{N} * \sum_{i \in N} |\log_{10}(d_i) - \log_{10}(d_i^*)| \tag{2.13}$$

Lastly, an accuracy metric (δ) is also devised for depth estimation, which measures the mean amount of pixels which present error value smaller than certain $treshold$, as shown in Equation 2.14. In this case, the higher its value, the better the performance.

$$\delta = \frac{1}{N} * \sum_{i \in N} max(\frac{d_i^*}{d_i}, \frac{d_i}{d_i^*}) < threshold \tag{2.14}$$

where $threshold = [1.25, 1.25^2, 1.25^3]$ are common values used for evaluation in NYU Depth Dataset V2 [47] and Eigen's KITTI split [48]. On this dissertation work, δ_1, δ_2 and δ_3 refers to δ at each threshold value, with 1.25 for δ_1, 1.25^2 for δ_2 and 1.25^3 for δ_3.

To conclude the theoretical foundation section, it's important to note that the concepts elucidated here, mainly on machine learning and neural networks, are just a fraction of the whole context since the current work is focused on the main techniques for 2D object detection and monocular depth estimation.

3 LITERATURE REVIEW

In this section, a review of the literature on topics surrounding automotive datasets and computer vision algorithms for object detection and monocular depth estimation is presented. The papers mentioned in this section were collected from both google scholar and google's database with the keywords "synthetic dataset, autonomous driving dataset, object detection, monocular depth estimation, domain adaptation, domain randomization"and selected according to their relevance to this dissertation's theme.

3.1 DATASETS

On datasets, two subsections are made: first an overview of synthetic and later on real-world based ones. For the synthetic part, both an outline of simulators and already assembled data are presented, whereas, on real-world ones, data collected from vehicles driving on different regions of various countries are presented.

3.1.1 Synthetic

On the context of synthetic datasets, advancements have been made to further approach virtual models to approximate more realistic ones. Examples of developed synthetic dataset simulators include SYNTHIA [49], Airsim [50], GTA PreSIL [51], NVIDIA Drive Constellation simulator [52], SynScapes [53] and CARLA [54]. A comparison on cientific impact between them is presented in Figure 3.1, with citation numbers updated in 28 May 2020.

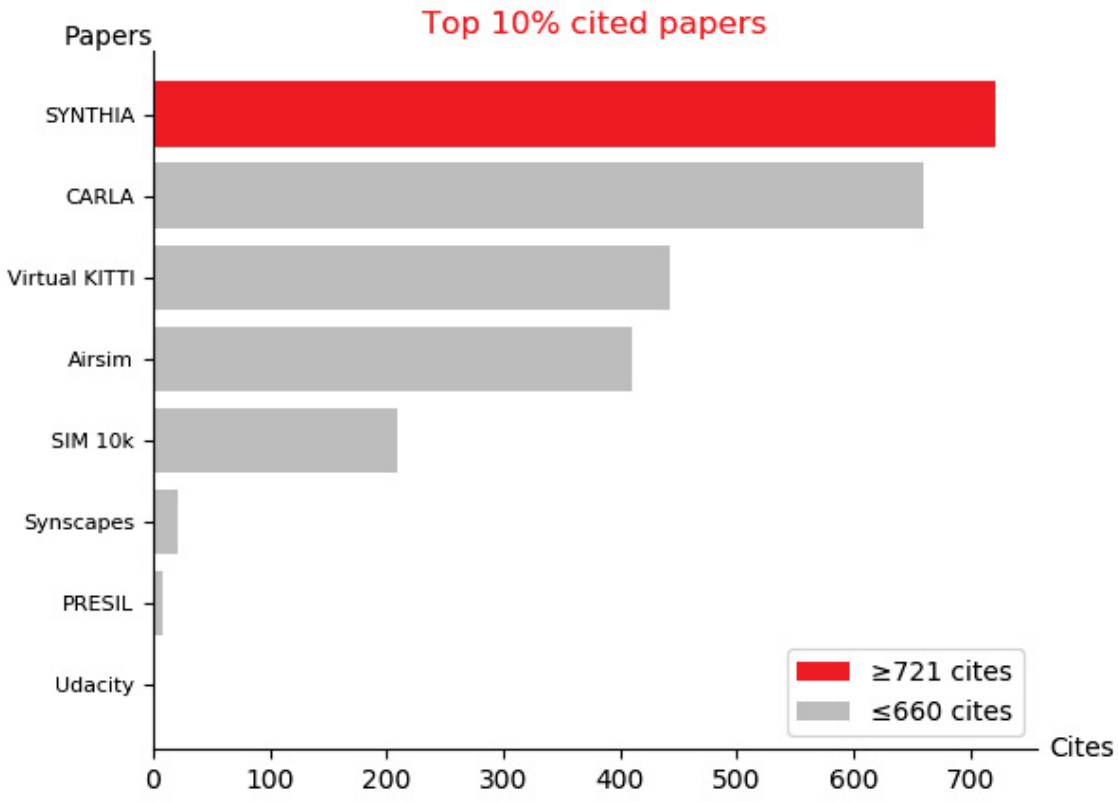
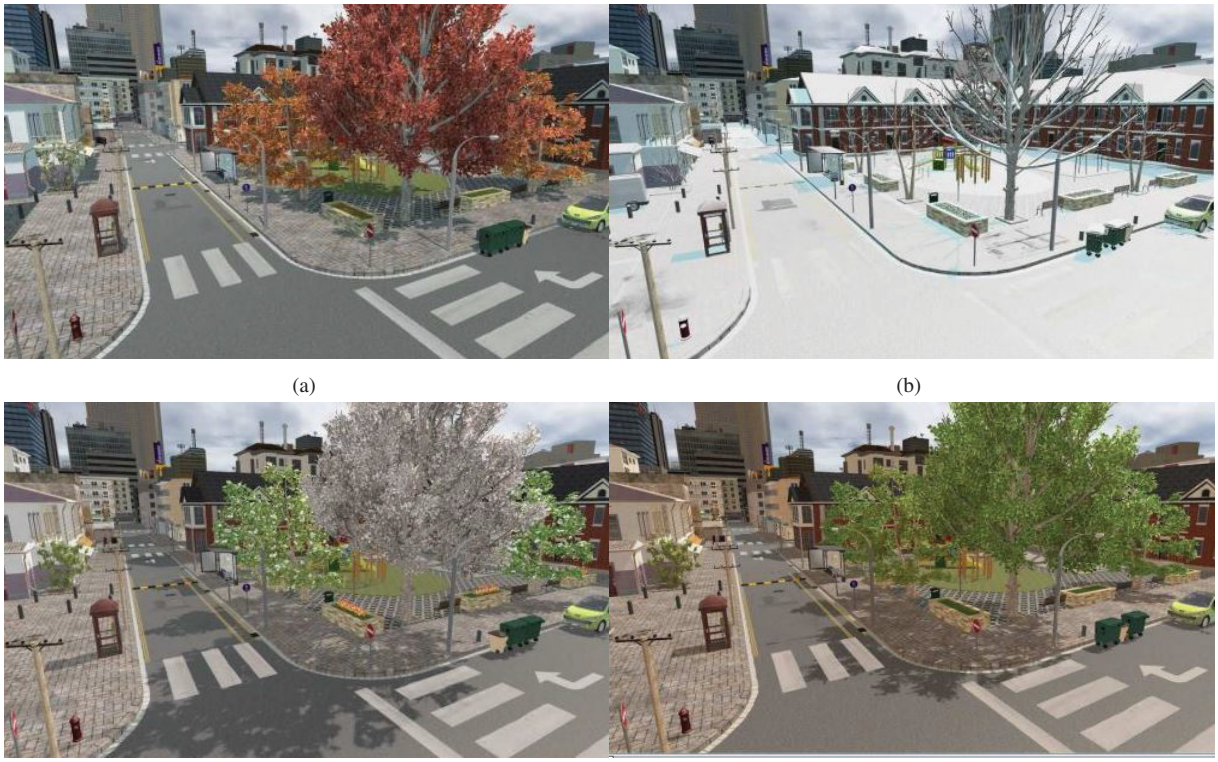


Figura 3.1: Synthetic datasets and simulators
Source: The author (2020)

In [55], a review on simulators and synthetic datasets for Unmanned Aerial Vehicles (UAVs) is presented, comparing the different types of data possible to be extracted, as well as each simulator's visual ambient and physics resemblance to the real-world. In their work, the authors compare Gazebo, Udacity, Sim4CV, AirSim, and CARLA simulators, and also build an extension on top of the later one. Besides this, SYNTHIA, Sintel, GTA V, and Virtual KITTI synthetic datasets are also compared.

SYNTHIA [49] is an automotive synthetic dataset created with Unity Engine, with the dataset being released in 2016 with 7 video sequences at 5 FPS in varied weather and illumination conditions. At its release, it contained RGB and semantic segmentation data with data from 2 multiview cameras each composed of 4 monocular pointed at each cardinal direction. SYNTHIA-Rand and SYNTHIA-Seqs datasets are presented, with 13.400 and 200.000 frames at 960x720, respectively. Annotated classes include sky, building, road, sidewalk, fence, vegetation, lane-marking, pole, car, traffic signs, pedestrians, cyclists, and miscellaneous. On its 2017 update depth data was added, and in 2019 annotations for instance segmentation, and 2D and 3D bounding box (BB) were implemented. The authors evaluated the performance of segmentation networks trained on it and tested on KITTI, which improved by around 8 percentage accuracy points for certain classes. Samples of its RGB data are presented in Figure 3.2.



(a) (b)

(c) (d)

Figura 3.2: Synthia dataset sample on all four weather seasons. (a) Autumn; (b) Winter; (c) Spring; (d) Summer
Source: [49]

Sim 10k, 50k, and 200k [56] are datasets released in 2016 and created with the video game Grand Theft Auto V with 10.000, 50.000, and 200.000 frames, respectively, with depth and annotated bounding boxes for cars. The authors compared the performance of a Faster-RCNN trained with their custom data and also mixed with Cityscapes, to then evaluate on KITTI. Their results show that a model trained only on their Sim 50k and 200k trained models outperform the same model trained only on Cityscapes for about 6, 7, and 4 percentage points on difficulty levels easy, medium, and hard. Samples of RGB data are presented in Figure 3.3.

(a) (b)

(c) (d)

Figura 3.3: Sim 10k dataset sample. (a) Late night; (b) Night; (c) Day; (d) Midday
Source: [56]

AirSim [50] is an open-source simulator developed by Microsoft and built on top of Unreal Engine, aimed mainly for data collection of aerial vehicles (e.g. drones) with a heavy emphasis on physics calculations for the EGO vehicle's positioning. Apart from that, it includes RGB, depth information, Inertial Measurement Unit (IMU) composed of gyroscopes and accelerometers, barometers, and magnetometers sensors, however, it also lacks native 2D and 3D bounding boxes annotations.

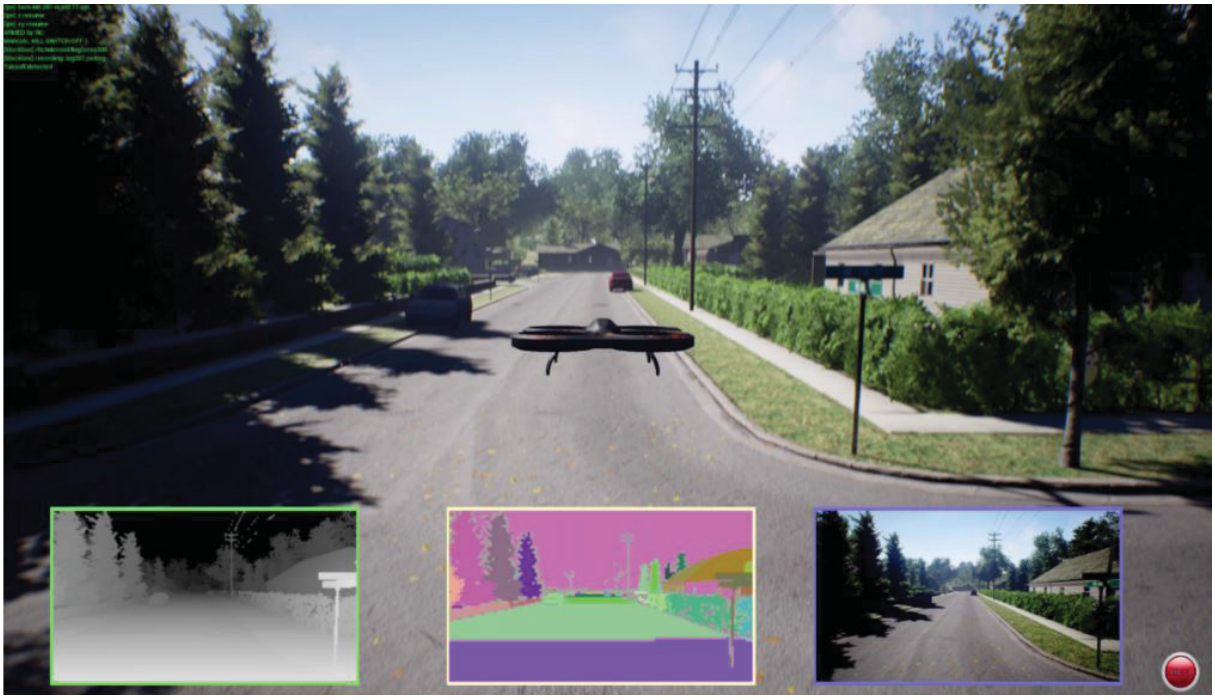
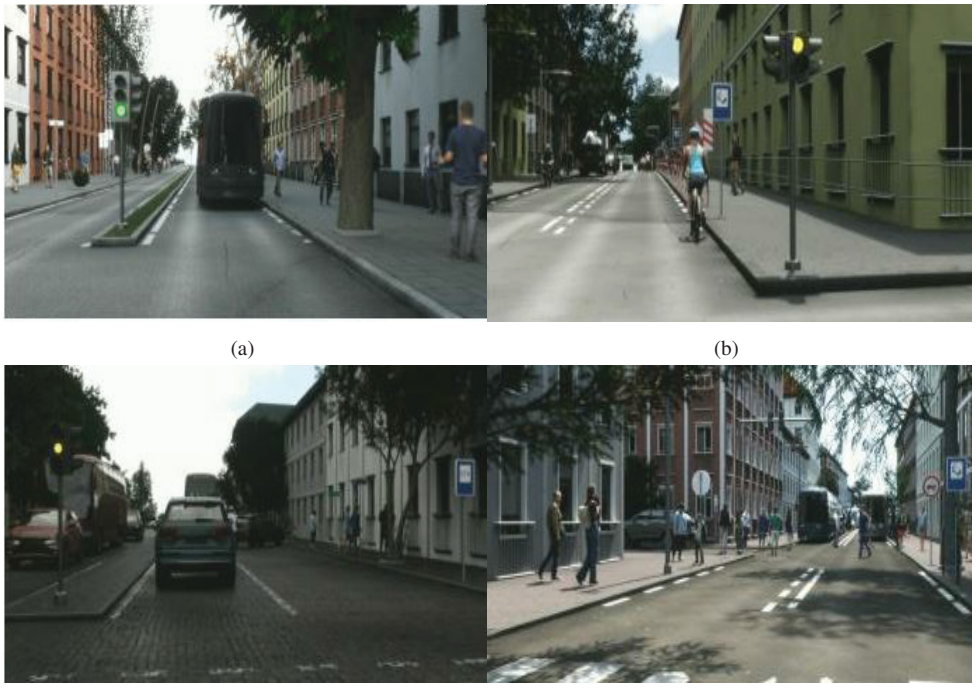


Figura 3.4: Airsim interface
Source: [50]

SynScapes [53] is a dataset built by 7D Labs and released in 2018 with the purpose of reproducing RGB images as photorealistic as possible, with 25.000 frames at 1440x720 and 2048x1024 resolution. With this in mind, besides prioritizing the illumination effects on the scene, the authors also worked on replicating disturbances common in RGB sensors, for instance, blur effects, sensor bit resolution, exposure, and the camera response curve. Available annotations are depth, semantic and instance segmentation, 2D, and 3D bounding boxes for the same classes present in Cityscapes [57]. To validate their work, the authors evaluated the 2D object detection task using a Faster R-CNN with ResNet101 backbone by training on their own synthetic and later validating on KITTI and vice-versa. When training their model with mixed data, an increase of around 6 percentage points was noted. For accessing the data, however, one needs to contact the authors via e-mail. A sample of the dataset is presented in Figure 3.5.



(a) (b)

(c) (d)

Figura 3.5: Synscapes samples. (a) Street way; (b) Cyclist; (c) Street way variant (d) Dense pedestrians population
Source: [53]

PreSIL [51] is a synthetic dataset released in 2019 with around 50.000 frames at 1920x1080 resolution, built on top of Grand Theft Auto V, and is one of the simulators which comes closer to CARLA's approach. The main idea is to take advantage of its already built commercial engine and the world and then further implement RGB, depth and LiDAR sensor data, semantic segmentation, and 2D and 3D bounding boxes annotations. One of the main points claimed by the authors is the more precise LIDAR calculations implemented since the native ray casting function has limitations regarding its range and dynamics. For the task of 3D object detection, the authors claim up to 5 AP percentage points increase on KITTI 3D when using 40.000 synthetic frames for pretraining. A sample of the dataset is shown in Figure 3.6.

Figura 3.6: PreSIL sample
Source: [51]

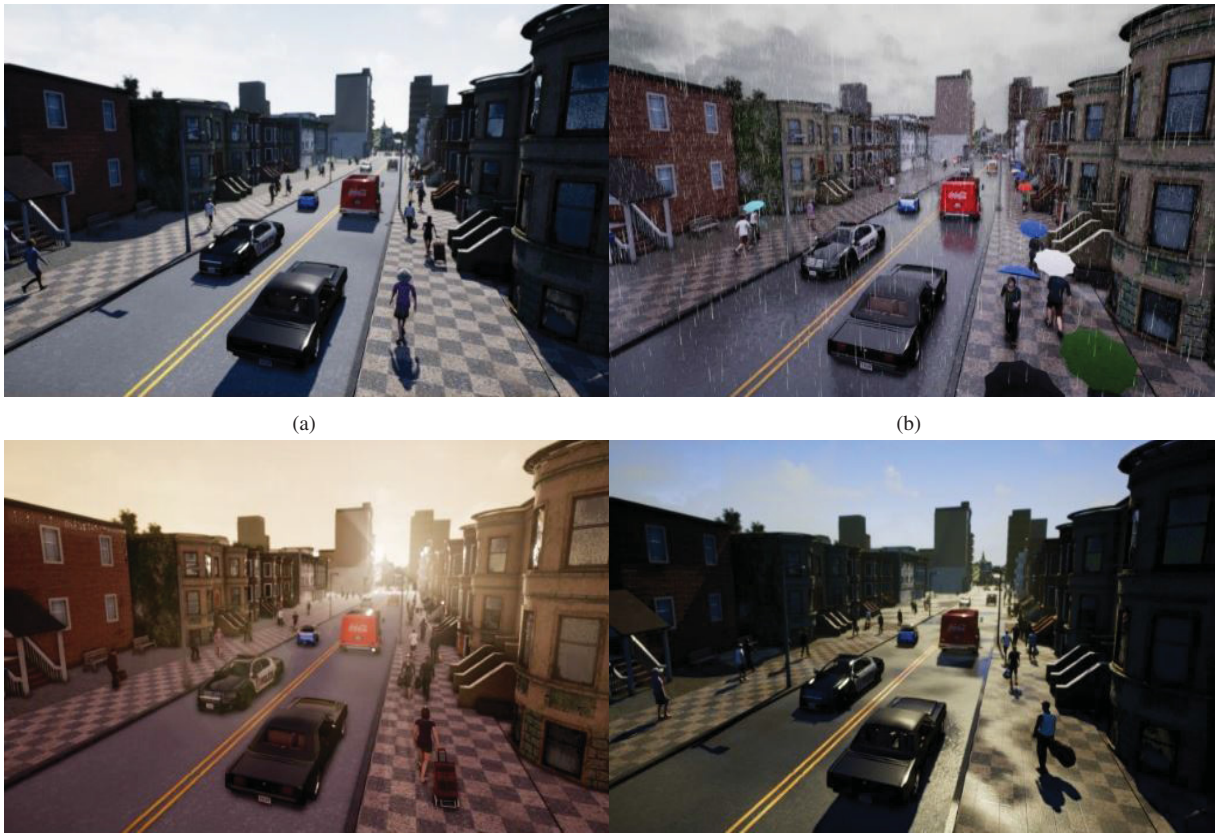
NVIDIA DRIVE™ Constellation AV simulator [52] also seems a promising option, but besides requiring specific NVIDIA hardware and permissions to use, at the time of writing no related paper was found. Thus an evaluation on it is not yet possible.

In an attempt to replicate real-world data, Gaidon et al. [58, 59] recreated the KITTI dataset in Unity Engine, named Virtual KITTI, with its first version being released in 2016 and latest in January 2020. To recreate such data synthetically, the authors mention the usage of "seed data"derived from the source dataset, which serves as a basis for initializing the virtual one. Next, off-the-shelf graphics assets are placed on the scene with positions according to the annotations from the source dataset; on roads and background objects, however, this process is done manually. Lastly, with the base synthetic world generated, automatic generation of ground-truth and weather variations becomes possible, with the advantage of the synthetic data, when compared to other types, indeed presenting a similar domain with a real-world dataset. One downside from this dataset, as mentioned as future works on their publication, is the lack of pedestrians in the data due to the challenges surrounding its animations. Samples of KITTI, Virtual KITTI, and Virtual KITTI 2 dataset are presented in Figure 3.7.

(a) (b) (c)

Figura 3.7: (a) KITTI, (b) Virtual KITTI, and (c) Virtual KITTI 2
Source: [59]

CARLA [54] is an open-source autonomous driving simulator released in 2017 and is being frequently updated, with the latest changes dating up to March 2020. It is built on top of the Unreal Engine and provides model assets designed by the authors. At the time of writing, besides RGB, 2D, and 3D bounding boxes, depth, and semantic segmentation sensors, it also contains LiDAR and RADAR devices, thus presenting itself as the most complete sensor suite. On their publication, however, the authors evaluated only the task of reinforcement learning for maneuver decisions. Therefore, the focus is going to be given on CARLA, since among its peers it presents the ambient that seems most similar to the real-world, as well as RGB camera feed data and depth information. Another point in favor of this simulator is the developers' disposition on further developing it for compatibility with other frameworks such as OpenDRIVE, SUMO, and ROS. Besides that, other works have been also evaluating their algorithms with CARLA, mainly with reinforcement learning [60, 61, 62]. A sample of the dataset is shown in Figure 3.8. A general overview of each synthetic dataset and simulator is shown in Table 3.1.



(a) (b)

(c) (d)

Figura 3.8: CARLA sample. (a) Cloudy; (b) Rainy; (c) Afternoon; (d) Midday
Source: [54]

Tabela 3.1: Comparison on synthetic datasets and simulators
Source: The author (2020)

Dataset	Frames	Resolution	Categories	Annotation*	Release Last update	Note	Reference
SYNTHIA	13407	1280x760	14	2D BB 3D BB instance segm. semantic segm. depth	2016 2019	-	[49]
SIM	10000 50000 200000	n/a	2	2D BB semantic segm. depth	2017 2017	data n/a	[56]
Airsim	-	-	-	semantic segm. depth gyroscope accelerometer barometer magnetometer	2017 2020	is a simulator	[50]
SynScapes	25000	1440x720 2048x1024	20	2D BB 3D BB instance segm. depth	2018 2018	-	[53]
PreSIL	50000	1920x1080	16	2D BB 3D BB instance segm. depth	2019 2020	-	[51]
Virtual KITTI 2	17000	1242x375	1	2D BB 3D BB instance segm. depth	2016 2020	-	[58, 59]
CARLA	-	-	13	3D BB semantic segm. depth	2017 2020	is a simulator	[54]

*On this dissertation work, not all annotation types shown on the table are required but are listed for completeness

3.1.2 Real-world

Many groups have assembled real-world automotive datasets, with examples being KITTI [6], Cityscapes [57], ApolloScape [63], BDD100K [64], and more recently nuScenes [65] and Waymo Open Dataset [66]. The main differences on each subsequent dataset lie on the types of sensors, recorded location, year, and presence of annotated data. A comparison on cientific impact between them is presented in Figure 3.9.

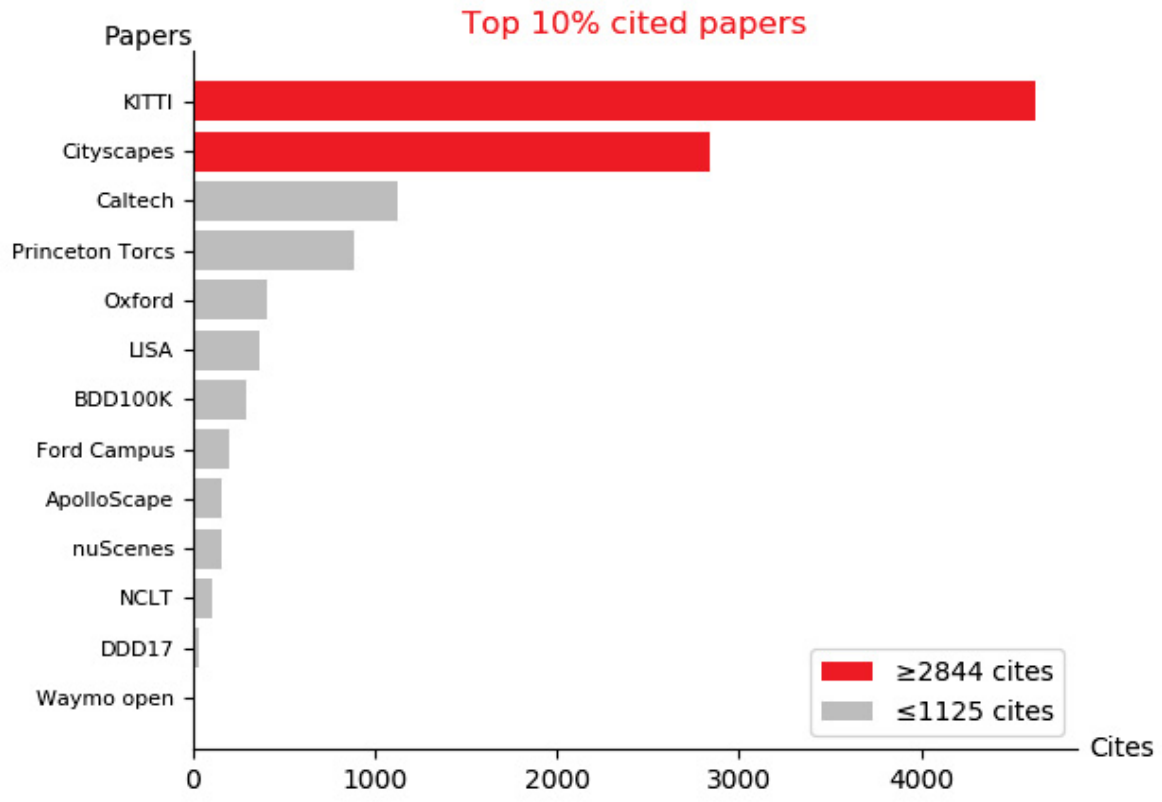


Figura 3.9: real-world automotive datasets
Source: The author (2020)

KITTI dataset [6] was recorded in 2011 by the Karlsruhe Institute of Technology and Toyota Technological Institute, and features a car equipped with 2 RGB and 2 grayscale cameras, a laser scanner (Velodyne LIDAR), Global Positioning Systems (GPS), and gyroscope sensors which drove in varied regions of Karlsruhe in Germany. In total, it presents around 15k annotated frames at 1242x375 resolution divided into 5 types of scenarios: city, residential, road, campus, and person. Regarding annotations, it contains 2D and 3D bounding boxes for cars, vans, trucks, pedestrians, person "sitting", cyclist, tram and misc, as well as semantic and instance segmentation. A sample of the dataset is shown in Figure 3.10.

(a) (b)

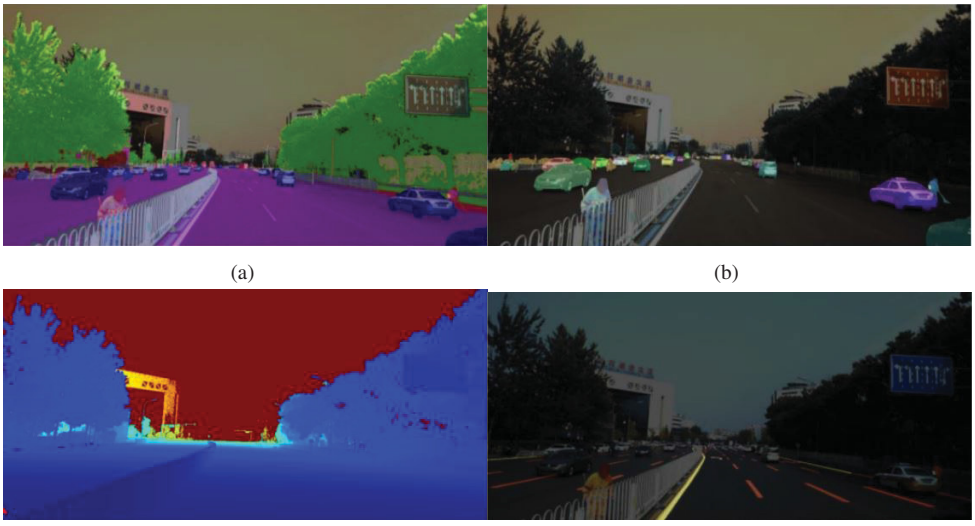
(c) (d)

Figura 3.10: KITTI sample. (a) Street way 1 (b) Street way 2 (c) Comercial area (d) Highway
Source: [6]

Following that, the Cityscapes dataset [57] was released in 2016, with the main focus on pixel-wise and instance-level annotated images captured in 50 different cities, mostly in Germany. The data was recorded with stereo RGB $\frac{1}{3}$ CMOS 2MP sensors with rolling shutter, vehicle odometry sensors, outside temperature, and GPS. In total, it presents 25.000 annotated frames at 2048x1024 with depth data computed according to the stereo cameras. There are 30 annotated classes, which contain different types of vehicles, persons, ambient, and traffic objects. A sample of the dataset is presented in Figure 3.11.

Figura 3.11: Cityscapes sample
Source: [67]

Afterward, the ApolloScape dataset [63] was made open in 2018 by Baidu Research, with a focus on providing data on a higher scale than the previous approaches. The acquisition setup is composed of two laser scanners with a range of 1.2m to 420m with 360°FOV, a VMX-CS6 camera system, a gyroscope, and GNSS. with over 140.000 RGB images at 3384x2710 captured in 10 cities in China [63] with semantic annotations and calibrated 3-dimensional (3D) point clouds. Annotated classes total 24, which include different types of vehicles, persons, and general objects, classified into movable objects, surfaces, infrastructure, or nature. A sample of the dataset is presented in Figure 3.12.



(a) (b)

(c) (d)

Figura 3.12: ApolloScapes sample (a) Semantic segmentation; (b) Instance segmentation; (c) Depth; (d) Lane markings
Source: [63]

Similar to Apolloscape, the BDD100K dataset [64] also focuses on acquiring large-scale data, however, their approach is centered in crowdsourcing the data. There are over 100.000 RGB video sequences at 1280x720 alongside GPS data collected in diverse regions in the United States, with the highest concentration being in New York, Berkeley San Francisco, and Bay area. Annotated data include bounding boxes, type of scene, and instance segmentation, with 10 classes: bus, light, sign, person, bike, truck, motor, car, train, and rider. It is to be noted that this dataset contains no depth data. Samples of this dataset are presented in Figure 3.13.

(a) (b)

Figura 3.13: BDD100k sample on two dashboard cameras. (a) Dashboard cmaera 1; (b) Dashboard camera 2
Source: [64]

More recently, nuScenes [65] and Waymo Open [66] were released in 2019 and are, to the best of the author's knowledge, the biggest multimodal annotated automotive datasets publicly available. NuScenes contains data recorded in areas of Boston and Singapore with a sensor suite of 6 RGB cameras at 1600x900, 5 radars, 1 LIDAR up to 70m range, and 360°FOV, GPS, and a gyroscope. The dataset presents calibrated annotated data of 3D bounding boxes between RGB and point cloud data. There are 23 annotated classes, which include different subclasses of pedestrians (child, police officer, adult, ...), vehicles, and traffic objects. Overall, the dataset contains 1.4 million labeled images and one sample is presented in Figure 3.14.

(a) (b)

(c) (d)

Figura 3.14: NuScenes sample. (a) Early day; (b) Night; (c) Midday; (d) Night alternate
Source: [65]

Waymo Open [66] introduced a similar sensor suite but without radar data, thus putting more emphasis on LIDAR information. The sensor suite is composed of 5 LIDAR sensors from

20 to 75m, 3 RGB cameras at 1920x1280, and 2 at 1920x1040 with 25.2°FOV. Notably, their sensor suite is not equipped with GPS or IMU, but rather their mapping is done entirely with a prior laser scan of the to be driven areas [68]. The dataset contains 1150 scenes with 230.000 annotated LiDAR data, 12 million 3D bounding boxes, and 9.9 million 2D bounding boxes, being recorded in regions of Phoenix and San Francisco in the United States. In total, it presents close to 1 million RGB frames, with a few samples of the frontal camera presented in Figure 3.15.

A general overview of each real-world dataset is shown in Table 3.2.

(a) (b)

(c) (d)

Figura 3.15: Waymo Open sample. (a) Midday; (b) Morning; (c) Rainy; (d) Clear weather
Source: [66]

Tabela 3.2: Comparison on real-world datasets
Source: The author (2020)

Dataset	Frames	RGB resolution LIDAR range (m) LIDAR FOV (HxV)	Categories	Annotation*	Release Last update	Reference
KITTI	15000	1392x512 up to 120 360°x26.9°	8	2D BB 3D BB GPS Gyroscope	2012 2015	[6]
Cityscapes	25000	2048x1024	30	semantic segm. instance segm.	2016 2019	[57]
ApolloScape	140000	3384x2710 1.2 to 420 360° "full circle"	24	3D BB semantic segm. GPS Gyroscope	2019 2019	[63]
BDD100K	100000	1280x720	10	2D BB instance segm. GPS	2018 2018	[64]
NuScenes	1400000	1600x900 up to 70 360°x40°	23	2D BB 3D BB GPS Gyroscope	2019 2020	[65]
Waymo Open	990342	1920x1280 and 1920x1040 up to 75 360x20° and 360x120°	3	2D BB 3D BB	2019 2020	[66]

*On this dissertation work, not all annotation types listed on the table are required but are listed for completeness

Since the current work will emphasize on 2D object detection and monocular depth estimation, then an analysis on Waymo Open, which was recently released, will be performed since the challenges surrounding Waymo Open are at the moment of writing still not deeply explored, besides possibly providing more accurate depth data than nuScenes. As of 15 April 2020, only 8 other works have referenced this dataset, with none presenting the same approach as described in this document. As for the synthetic part, CARLA was chosen due to its constant improvements, extension potential, and also Python Application Program Interface (API) availability.

3.2 COMPUTER VISION

In this section, approaches on object detection and depth estimation based on RGB images are presented. Object detection deals with the task of identifying objects on images, assigning classes, and positions for them. On another hand, monocular depth estimation deals with the task of assigning distances for each pixel on an image by using a single camera viewpoint. For both tasks, research on academic impact was evaluated, which indicates not which are the best state-of-the-art performing algorithms, but rather help guide a common workflow basis for other algorithms which can perform better on established datasets. Instead, the decision of evaluating certain(s) algorithm(s) was made considering three main points: metrics performance on a common dataset, availability of source code from the paper's authors, and presence frequency in similar works. Finally, it's important to note that the citations amount from the papers researched were collected between 18 and 20 of June 2019, so the reported numbers might vary considerably.

3.2.1 Object detection

On the object detection part of this work, the focus will be given to 2D bounding box estimation, and as such, the related work will be specific for this context. Even though other types of object detection are relevant, e.g. 3D, semantic, and instance segmentation which represent finer object contours, their ground-truth information is not always present in automotive large-scale datasets, whereas 2D bounding boxes are usually present. Therefore, without ground-truth information, quantitatively evaluating model performance and supervised training is not possible.

A survey on bounding box detection is presented on [69], which listed algorithms coupled with deep learning techniques that contributed the most to research on this area. Based on this study and also further delving into google scholar database and on repositories with open-source code, a listing with algorithms that perform bounding box detection was created and is presented in Figure 3.16, with the top ten percentile highlighted in red. The full list is openly available on the author's GitHub repository[1].

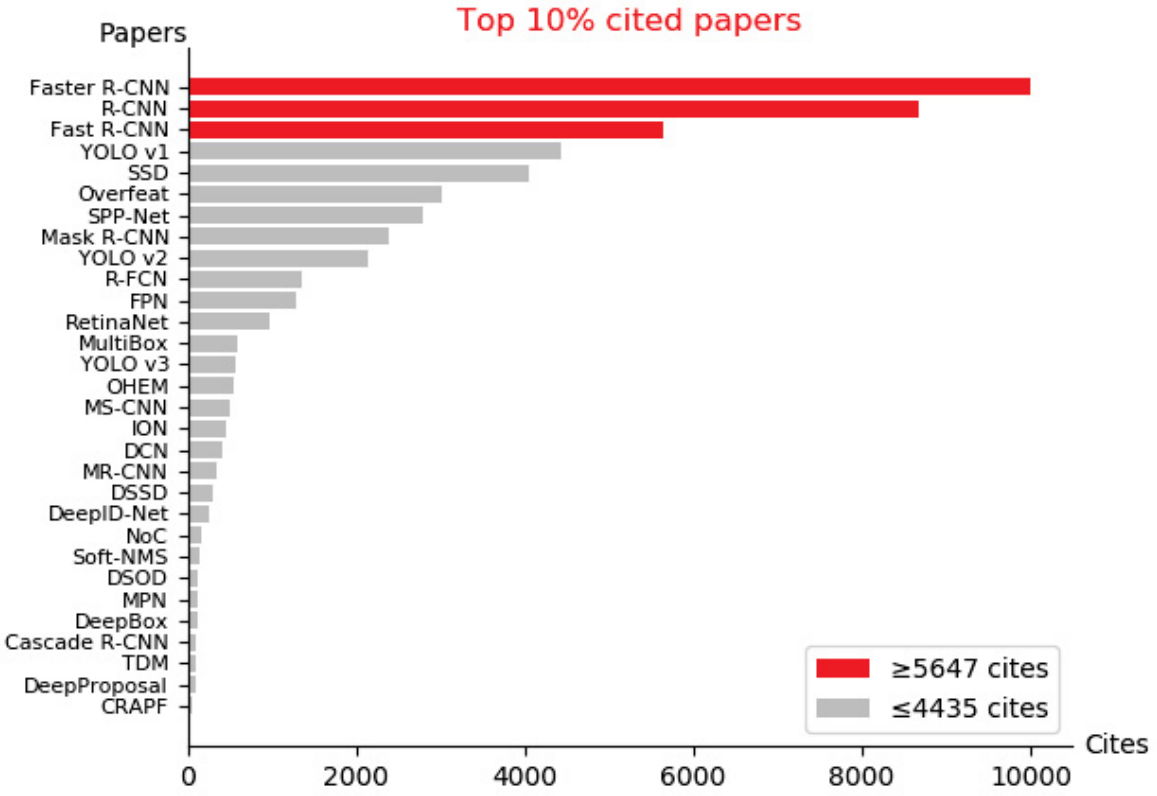


Figura 3.16: Relevance analysis on bounding box algorithms
Source: The author (2019)

Regions with CNN features (R-CNN) is a method proposed by Girschik et. al. [70] which combines the region proposal task with Convolutional Neural Networks (CNN). The main idea consists of four steps, as illustrated in Figure 3.17:

1. Compute a region proposal algorithm to gather around 2000 regions. In R-CNN, the selective search method is applied;

[1] github.com/AlanNaoto/papers_and_code_ranking

2. Warp affine each region to a fixed size;
3. Compute features by inputting the image into a CNN;
4. Classify each region using a set of Support Vector Machines (SVM).

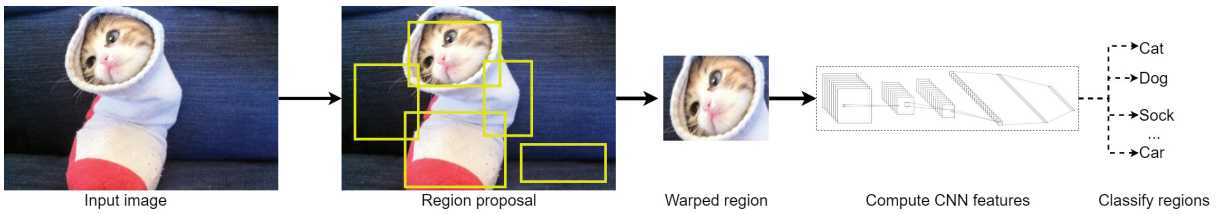


Figura 3.17: R-CNN workflow
Source: Adapted from [70]

The CNN reported by [70] has its first layer with 227x227 size followed by five convolutional and two fully connected layers. Lastly, it's output layer presents a 4096 feature vector. R-CNN achieved, at the time, the highest mAP with 53.3% on PASCAL VOC 2012, a visual object detection challenge containing twenty object classes, such as people, animals, vehicles, and indoor objects [71]. For training on this dataset, the authors first pretrained their model on ImageNet 2012 and then later fine-tuned it with the SGD optimizer at a learning rate of 0.001 and batch size 32.

Girschik [72] proposed Fast R-CNN, which is an extension of his previous algorithm, R-CNN. The main changes on Fast R-CNN are on making the training a single-stage pipeline where computational resources are shared for passing region proposals through the CNN and on changing the output layers to contain besides the softmax as a classifier, also a bounding box regressor for fine-tuning its coordinates. These concepts are shown in Figure 3.18. This algorithm achieved an mAP of 66% on PASCAL VOC 2012.

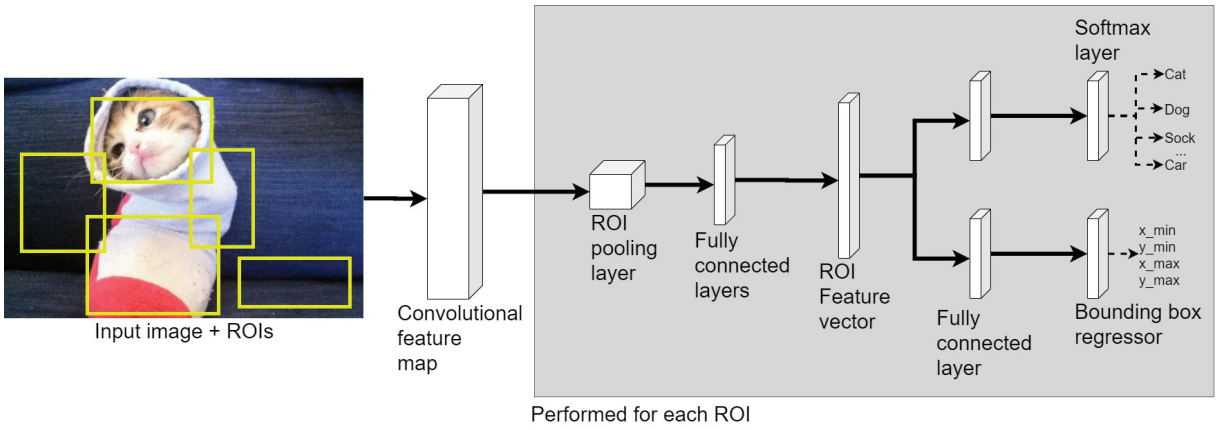


Figura 3.18: Fast R-CNN workflow
Source: Adapted from [72]

Afterward, Ren et. al. [11] proposed an extension to Fast R-CNN, the Faster R-CNN. Unlike its predecessors, this algorithm introduces the concept of Region Proposal Networks (RPN) that enables the algorithm to be feed with full images and learn the region proposals through it. In short, its the combination of two modules, one being a deep CNN that proposes regions and the Fast R-CNN detector that uses these regions, as shown in Figure 3.19. This algorithm achieved an mAP of 75.9% on PASCAL VOC 2012.

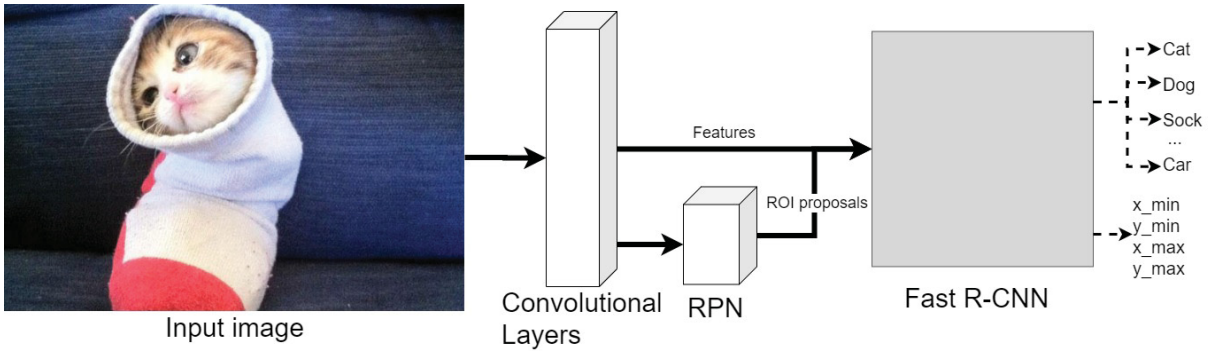


Figura 3.19: Faster R-CNN workflow
Source: Adapted from [11]

The previous approaches rely on proposing regions and then classifying each; another popular method is by performing only a single pass of the image to find regions and classes, thus treating the whole object detection task as a regression problem. Two popular algorithms are You Only Look Once (YOLO) [73] and Single Shot Detector (SSD) [10]. In both algorithms, the input image is divided into fixed-size cells, which then serve as input for a posterior CNN to extract features, infer which objects are present in it, and refine the bounding box location. YOLO achieved 63.4% mAP on PASCAL VOC 2012 and SSD 74.3%.

in Tables 3.3 and 3.4, an overview on metrics performance for 2D bounding box algorithms on COCO and KITTI are presented. An overview on COCO is useful since it serves to indicate the detector's performance on generic/common objects data, while analysis on KITTI can point to its potential performance for other automotive data. It is to be noted that the presented tables were built according to data reported in each algorithm's respective paper, source code repositories, and reviews. Therefore, for the same algorithm, more than one metric value can be reported. Surprisingly, from the algorithms listed on both tables, the Faster R-CNN is the one that presents the highest variation when compared to the others, with its lowest reported metric being more than double the highest reported. There are various reasons for such divergences, such as algorithm implementation, different backbones, and usage and source of pretrained weights.

Tabela 3.3: mAP from 0.50 IoU to 0.95 IoU at 0.05 step on COCO test set
Source: The author (2020)

Algorithm	**mAP@[.5, .95]**	**Reference(s)**
R-CNN	-	-
Fast R-CNN	20.1, 19.7	[74], [11]
Faster R-CNN	30, 40.2, 24.2, 19.3, 36.8	[75], [76], [74], [11], [77]
Mask R-CNN	36, 41.0	[75], [76]
Mobilenet	22	[75]
YOLO	-	-
YOLOv2	21.6, 21.6	[74], [77]
YOLOv3	33.0	[77]
RetinaNet	37.9, 39.1, 40.8	[76], [69], [77]
SNIPER	47.6, 47.9	[69], [78]
Autofocus	47.9, 47.9	[69], [78]
EfficientDet	**52.2**	[79]

Tabela 3.4: mAP for detections' difficulty easy, medium and hard on KITTI test set
Source: The author (2020)

Algorithm	**mAP (easy, medium, hard)**			**Reference(s)**
	easy	medium	hard	
R-CNN	-	-	-	-
Fast R-CNN	77.93	65.01	60.42	[80]$_{onlypedestrians}$
Faster R-CNN	**87.33, 80.44**	**86.67, 70.75**	**76.78, 62.89**	[81]$_{onlycars}$, [82]
	95.14, 77.12	**83.73, 61.15**	**71.22, 55.12**	[83]$_{onlycars}$ [84]$_{onlypedestrians}$
Mask R-CNN	-	-	-	-
Mobilenet	**86.25**	**86.07**	**76.18**	[81]
YOLO	47.69	35.74	29.65	[81]
YOLOv2	76.79	61.31	50.25	[81]
YOLOv3	-	-	-	-
RetinaNet	-	-	-	-
SNIPER	-	-	-	-
Autofocus	-	-	-	-
EfficientDet	-	-	-	-

*Some references only provide metrics for a specific class. In those cases, *onlypedestrians* refers to a pedestrian detector and *onlycars* a vehicle detector.

Many works on the automotive field have tackled Faster R-CNN as the base model for verifying object detection performance [66, 81, 85, 83, 84], which serve as an argument for its potential.

With the intent on developing an algorithm specialized for vehicle detection, Nguyen et al. [81] made several modifications to Faster R-CNN by using MobileNet as its architecture; changing the output of the region proposal network's non-maximum suppression layer for a soft non-maximum suppression to deal with heavy vehicle occlusion; and lastly the RoI pooling layer for a specified size. An ablation on the study of each change reported in around 3 percentual points for average precision on the KITTI's test dataset while speeding up the processing time from 2 to 0.15 seconds per frame. Similarly to Nguyen et al. [81], Leung et al. [85] also studied an optimized Faster R-CNN for vehicles, but with a focus on darker image data, i.e., with less lighting. On their work, however, the authors did not modify the Faster R-CNN structure, but instead trained their Faster R-CNN with a ResNet101 and later with a VGG16 backbone on their dataset. An 84.97% AP with the ResNet101 model was achieved on their dataset's validation split, which is a point in favor of the detector's capability.

Fan et al. [83], on the other hand, investigated the details on training a Faster R-CNN for vehicles on the KITTI dataset, with regards to hints for adequate hyperparameters tuning. Specifically, the authors analyze the input image rescaling method and the number of proposals. According to their results, higher training and testing image resolutions lead to better performance, with tests done on resolutions from 800x800 to 1800x1800 at a step of 200x200. A higher number of proposals, however, does not seem to have much of an effect on performance. Iterative training was also tested, i.e., the first RPN proposals are fed back to the beginning of the Faster R-CNN, but with only the convolutional layers changing weights. This method helped only slightly, with an improvement of around 3 percentage points on AP.

Leaving the context of vehicles, Zhang et al. [84] analyzed the Faster R-CNN's performance for detecting pedestrians and proposed bootstrapping for negative samples and a cascaded boosted forest to classify proposals from the RPN while sharing parameters. The

authors report their performance metric in a log-average Miss Rate on False Positive per Image following Caltech dataset's standards with a 14.9% value.

With the insights learned In this section, Faster R-CNN was chosen as the basis algorithm for evaluation in this work, due mainly to the following: scientific impact relevance; broad presence in solutions for the automotive context; decent performance in generic and automotive datasets, and availability of open-source code by the algorithm's authors.

3.2.2 Monocular depth estimation

Estimating depth from pixels on an image is a complicated task and an ill-posed problem since the same image can represent various possible depth values [86], i.e., is ambiguous. On this task, supervised, unsupervised and self-supervised approaches are possible, with the first potentially leading to better results. However, the problem with supervised methods is that a ground-truth is required. For 2D bounding boxes, acquiring annotated data is straightforward: a human labels on each image the contour of the objects of interest. For depth data, however, that is not possible, and instead, we have to rely on depth sensors, which are usually a combination of expensive, sparse (low resolution), and complex calibration. With this in mind, research interest has been shifting to unsupervised and self-supervised approaches. In this section, an overview of scientific relevant papers regarding monocular depth estimation is first analyzed; in sequence, the focus is given on unsupervised and self-supervised algorithms, and finally, a comparison of performance for the latter algorithms on an automotive dataset is presented.

A survey on monocular depth estimation techniques is shown on [87]. Based on this study and also further delving into google scholar database and on repositories with open-source code, a listing with algorithms was constructed and is presented as a chart in Figure 3.20, with the top ten percentile highlighted in red. From ascending order, the top 10% are discussed in the following.

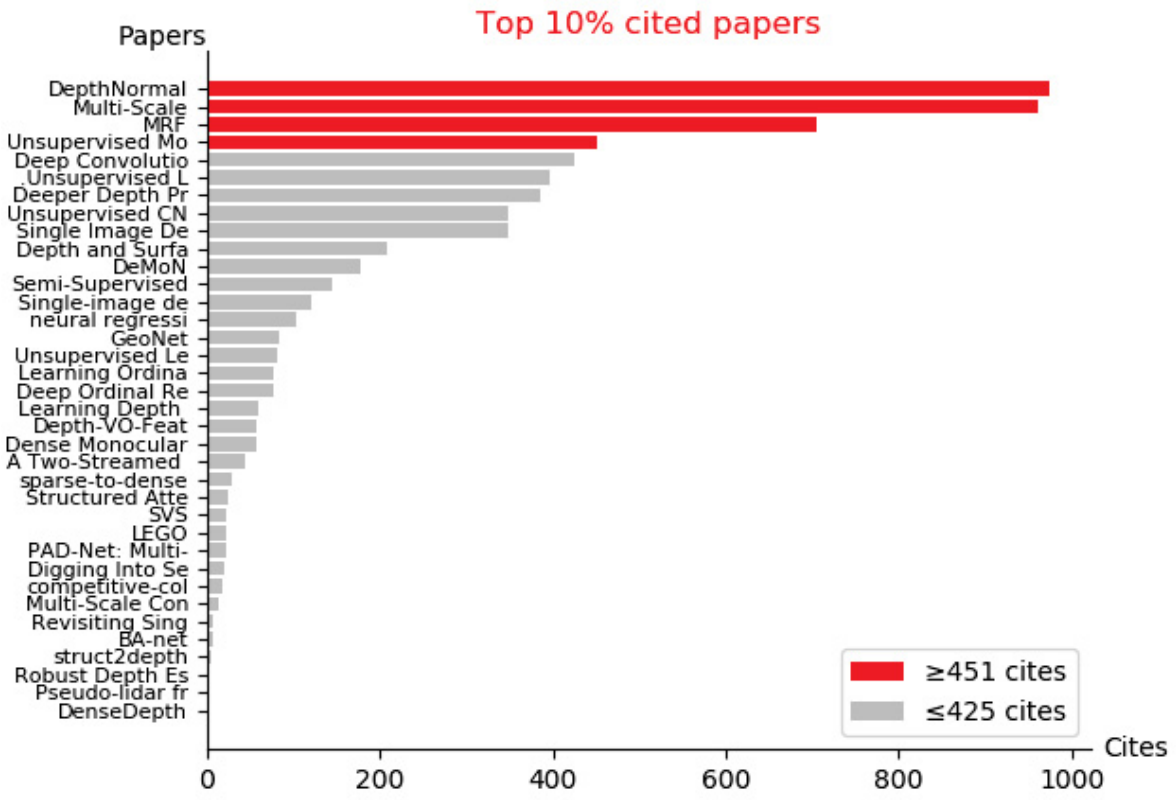


Figura 3.20: Relevance analysis on depth estimation algorithms
Source: The author (2019)

Godard et al. [17] proposed an unsupervised monocular depth estimation approach that takes advantage of epipolar geometry constraints between pairs of rectified stereo images during model training. The intuition on the authors' work is to force their model to learn how to reconstruct the image from one camera based on the view from the other camera. With that, the depth can be recovered on evaluation by computing the disparity given the input frame and the estimated image from the other view. The network model is based upon DispNet [88] and is composed of an encoder and decoder. On KITTI 2015 error metrics, the model presents 0.108, 0.657, 3.729, and 0.194 for Abs rel, Sq Rel, RMSE and RMSE log; for accuracy, 0.873, 0.954 and 0.979 for δ_1, δ_2, and δ_3. Inference samples on KITTI are presented in Figure 3.21, where noLR indicates the absence of the left-right consistency loss on the model.

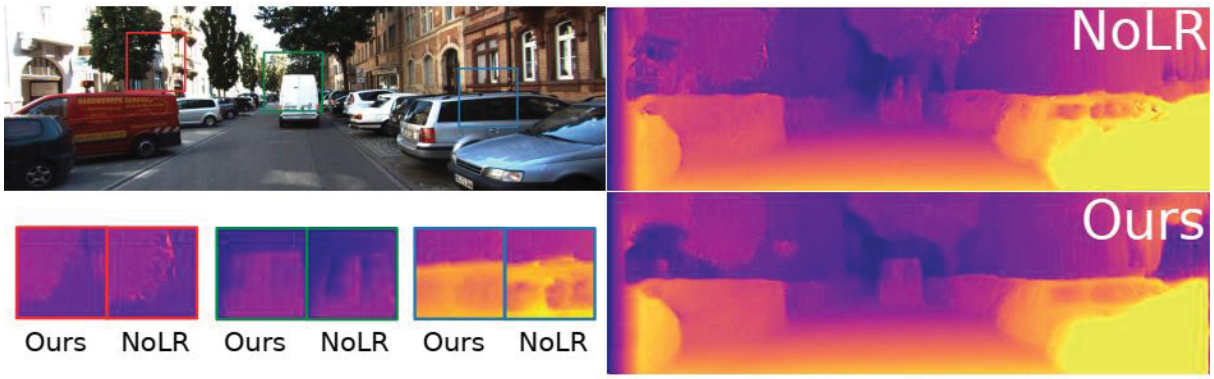


Figura 3.21: Inference samples of Godard et al. on KITTI
Source: [17]

On [89], a supervised approach for depth estimation from a single monocular image is studied. The idea is that since humans are capable of inferring depth from an image through context on it, then an algorithm should be capable of the same. A Markov Random Field is used, which assists in correlating global image features with depth points. As for feature types, the authors break down the image in small patches and then use absolute and relative depth, with the first being the depth to a patch and the latter the depth between patches. For the features, texture variations, gradients, and haze are analyzed. The authors train and test their algorithm on data they collected with a SICK 1-D laser ranger finder coupled on a motor, resulting in 425 images at 1704x2272 with 86x107 depth resolution. The model's errors are presented in log RMS per type of ambient, ranging from 0.132 to 0.295. The dataset, along with some inference samples from two of their models are presented in Figure 3.22.

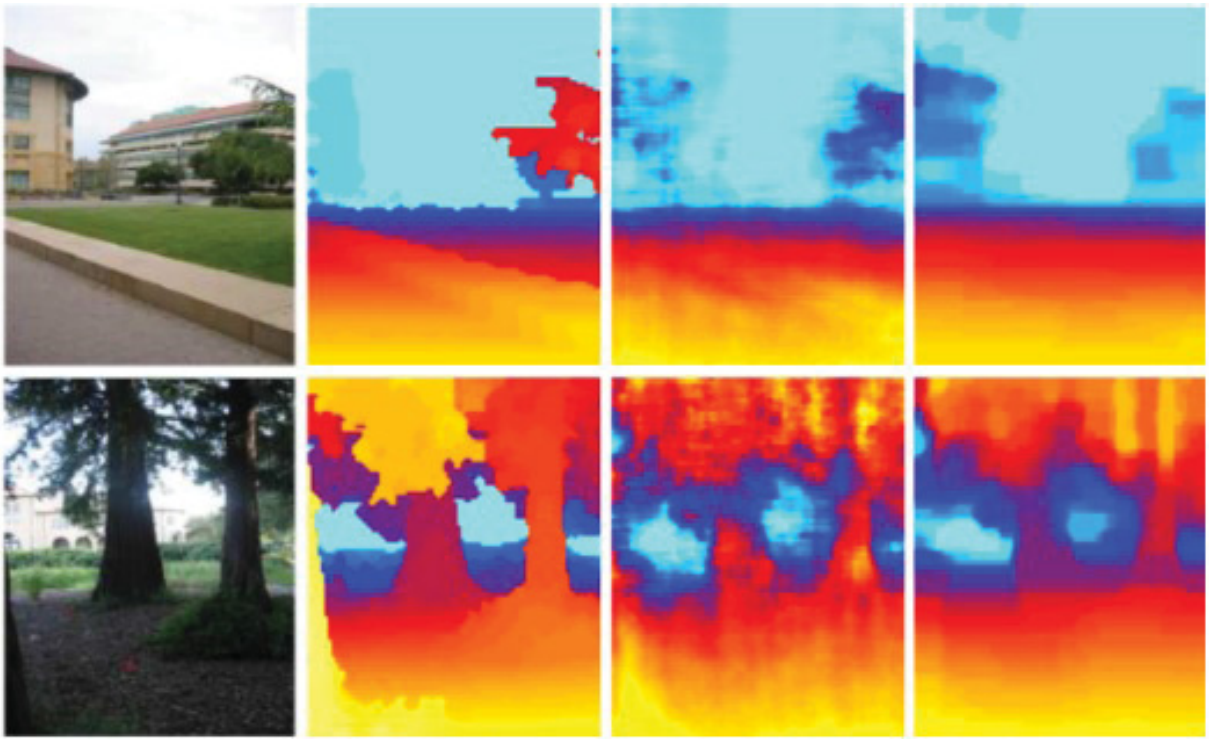


Figura 3.22: By column from left to right: image, ground-truth, gaussian model, laplacian model
Source: [89]

Eigen et. al. [48] proposed a supervised method for estimating depth from single images using a Multi-Scale deep network. The authors' main idea is to apply two deep neural networks, with one predicting depth for the entire image and the second one refining local predictions made by the first network, as shown in Figure 3.23. On the figure, both CNN 1 and CNN 2 are composed of 4 and 3 convolutional layers in series, with the first also presenting two fully connected layers at the end. It is interesting to note that depending on the source dataset, the authors change all dimensions of the network's layers. In addition to that, to account for the scaling issue, a scale-invariant error metric is formulated, where spatial relations take priority instead of the depth itself. Their model was trained and tested on NYU Depth v2 and KITTI, achieving 0.285 on log RMSE and 0.270, respectively. Some inference results of their model on KITTI is presented in Figure 3.24.

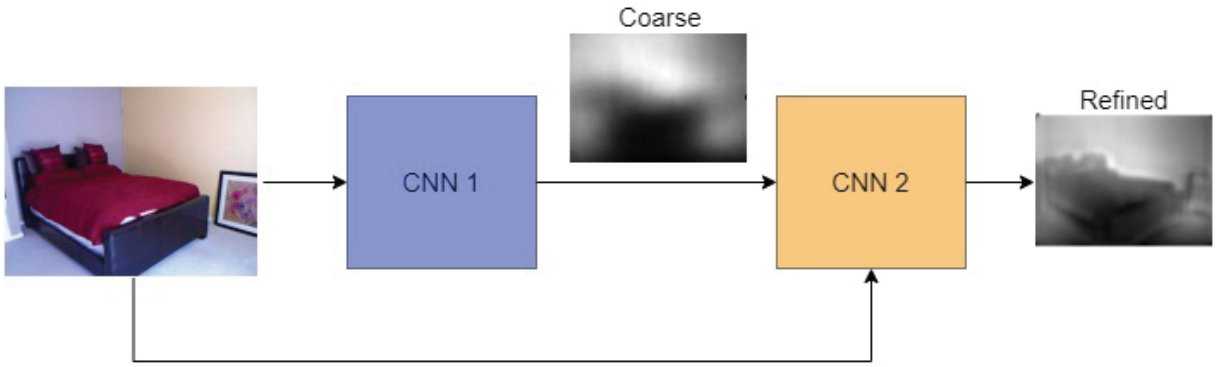


Figura 3.23: Multi-Scale high level
Source: Adapted from [48]

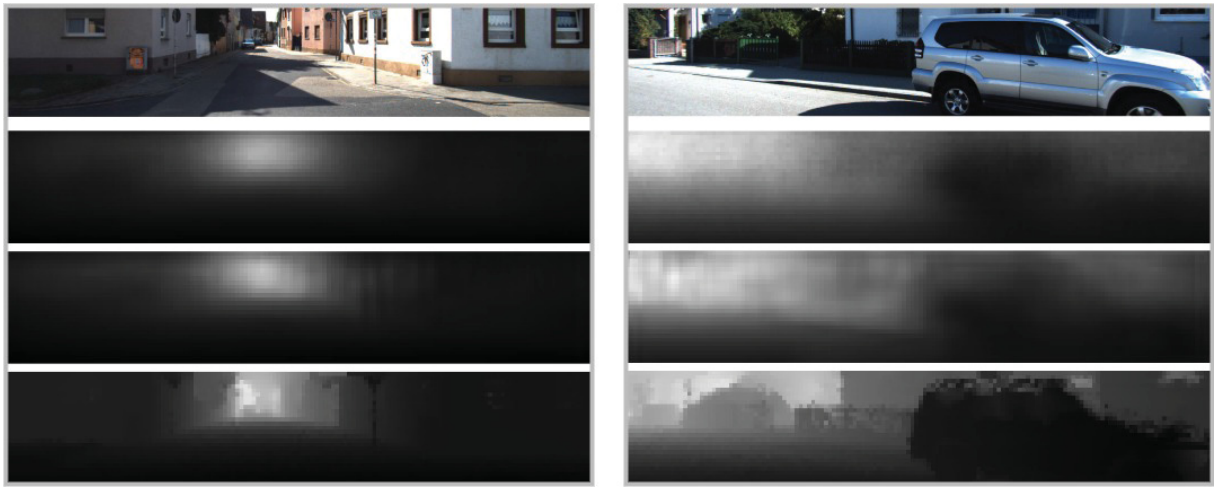


Figura 3.24: Multi-Scale inference on KITTI. From top to bottom: input image, coarse prediction, refined output, and ground-truth
Source: [48]

In a sequence to their work, Eigen et. al. [90] introduced the supervised common Multi-Scale Convolutional Architecture, which is capable of predicting not only depth but also surface normals and semantic segmentation. On their improved network, three scales are used instead of two, with the main idea being that with more scales, finer predictions can be made and better performance and resolution in depth estimation could be achieved. On the first scale, AlexNet and VGG are implemented, while on scales 2 and 3 several convolutional layers are

placed in series. The network architecture is shown in Figure 3.25. The authors train and test their model on NYUDepth v2, with the depth prediction log RMSE being 0.214.

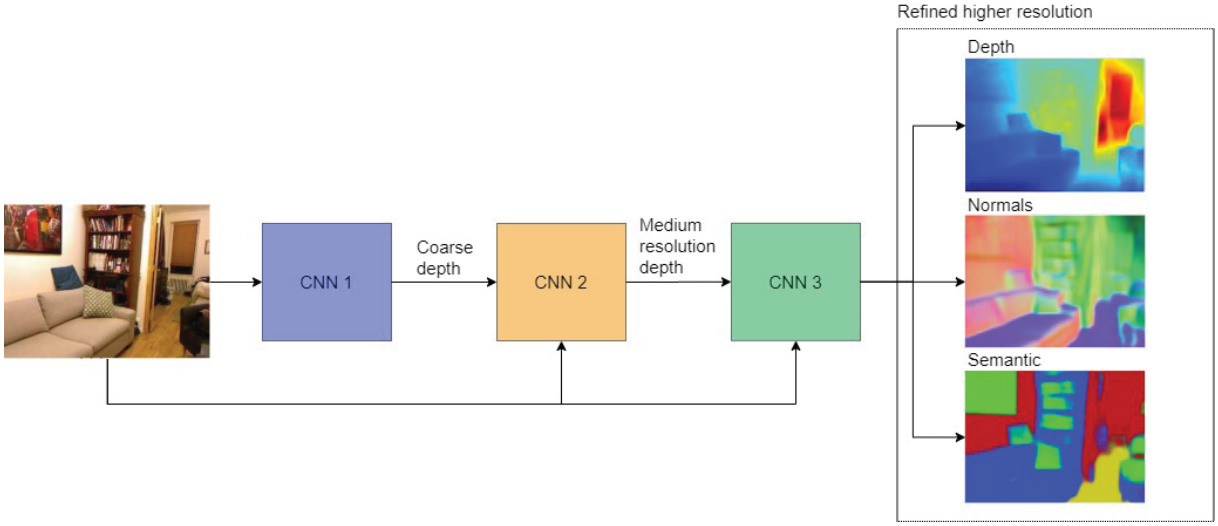


Figura 3.25: Multi-Scale Convolutional Architecture's network on a high level
Source: Adapted from [90]

Just as the best performing object detection algorithms rely on CNNs, for monocular depth estimation the same trend is observed. To further specify the focus of this work, attention is going to be given on self-supervised and unsupervised approaches for this task. In Table 3.5, an overview on metrics performance for said algorithms on Eigen's 2014 split of KITTI [48] is presented. For context, Eigen's split of KITTI dataset specifies which data should be assigned for training and testing; RGB frames resolution, and calibration between LIDAR with RGB data. In 2017 the Karlsruhe Institute of Technology [91] released an alternate official calibrated dense depth map for KITTI and metrics. However, most papers only report metrics on the previous split, with the current situation being a transition between evaluation in either.

Tabela 3.5: Self-supervised and unsupervised algorithms performance on Eigen's split of KITTI
Source: The author (2020)

Algorithm	**Abs rel**	**Sq rel**	**RMSE**	**RMSE log**	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$	**Ref**
Zhou	0.183	1.595	6.709	0.27	0.734	0.902	0.959	[92]
Yang	0.182	1.481	6.501	0.267	0.725	0.906	0.963	[93]
Mahjourian	0.163	1.24	6.22	0.25	0.762	0.916	0.968	[94]
LEGO	0.162	1.352	6.276	0.252	-	-	-	[95]
DDVO	0.151	1.257	5.583	0.228	0.81	0.936	0.974	[96]
DF-Net	0.15	1.124	5.507	0.223	0.806	0.933	0.973	[97]
Geonet	0.149	1.06	5.567	0.226	0.796	0.935	0.975	[98]
EPC++	0.141	1.029	5.35	0.216	0.816	0.941	0.976	[99]
Struct2depth	0.141	1.026	5.291	0.215	0.816	0.945	0.979	[100]
Ranjan	0.14	1.07	5.326	0.217	0.826	0.941	0.975	[101]
Gordon	0.128	0.959	5.23	-	-	-	-	[102]
monoResMatch	0.116	0.986	5.098	0.214	0.847	0.939	0.972	[103]
monodepth2	**0.115**	**0.882**	**4.701**	**0.19**	**0.879**	**0.961**	**0.982**	[86]
VNL	0.072	**-**	3.258	0.117	0.938	0.99	0.998	[104]
BTS	**0.059**	**0.245**	**2.756**	**0.096**	**0.956**	**0.993**	**0.998**	[105]

Even though BTS is the one who currently presents the best performing metric, at the time of this dissertation's writing, its corresponding paper has not yet been peer-reviewed; as such, the works tailing it up to monopdeth2 are briefly analyzed.

In [86], Godard et al. proposed the monocular depth estimation algorithm monodepth2 coupled with tree main methods which improve training on video sequences: an appearance matching loss to account for occluded pixels, auto masking to ignore pixels where there is no relative camera motion, and a multi-scale appearance matching loss to reduce depth artifacts. For training the network, the main idea is to predict the appearance of a target image based on the viewpoint of another image while constraining the network with an intermediary depth value. In other words, the relative pose between the two views is sought. The authors model their work for monocular, stereo, and mixed training, but specifically for the monocular case, the two views are the adjacent previous and posterior frames t' to the target frame t. For that, the photometric reprojection error (L_p) is minimized according to Equations 3.1, 3.2, and 3.3 as losses for their network, so that in the end, both the camera pose and depth are being solved.

$$L_p = \sum_{t'} pe(I_t, I_{t'\to t}) \tag{3.1}$$

where:

- $I_{t'}$ is the source image's pose,
- I_t is the target target's pose,
- $I_{t'\to t}$ is the relative pose between the source and target image's pose,
- pe is the photometric reconstruction error, calculated with the L1-Norm (Manhattan distance).

$$I_{t'\to t} = I_{t'} < proj(D_t, T_{t\to t'}, K) > \tag{3.2}$$

where:

- D_t is the dense depth map,
- $proj$ are the 2D coordinates of projected depths D_t in $I_{t'}$,
- K is the intrinsic matrix of the camera,
- $<>$ is the bilinear sampling operator.

$$pe(I_a, I_b) = \frac{\alpha}{2} * (1 - SSIM(I_a, I_b)) + (1 - \alpha) * ||I_a - I_b|| \tag{3.3}$$

where:

- α is a constant equal to 0.85,
- $SSIM$ is the "Structural SIMilarity"index, an equation defined in [106], which basically compares images based on luminance, contrast, and structure.

The algorithm is composed of a depth and pose network, with the first being a fully convolutional U-Net [14] with a ResNet18 and the latter a CNN with encoder-decoder structure from the authors' previous work [17] which is based upon DispNet [88]. As a quick note, DispNet is a CNN designed for optical flow, disparity, and scene flow estimation, which alternates between contracting and expanding convolutional layers.

Following the performance table, Yin et al. [104] proposed the use of an encoder-decoder network to predict depth maps. The main idea is to first reconstruct a 3D point cloud based on the 2D image with the encoder-decoder to then estimate virtual normals from a virtual plane formed by three randomly sampled non-colinear points. These concepts are presented by the authors as loss functions derived from surface normals calculation to enforce geometric constraints on the 3D plane. As such, the training is done in two stages: first on reconstructing the point clouds and later supervised by the constructed point clouds to form the virtual normal, which is used for predicting depth. The method for first reconstructing the point cloud directly from images, however, is not explained clearly on their publication or source code wiki.

Finally, Lee et al.'s work [105] holds the best performing metrics in Table 3.5 however has not yet been peer-reviewed. The author's proposal is of an encoder-decoder architecture, where encoders extract features and decoders predict depth from them. Their main contribution is on the encoder stage to use local planar guidance layers, which are 1x1 convolutions and a sigmoid block with equations for local depth cues and scale-invariant error for calculating the losses.

Considering the study on this subsection regarding academic impact, methodology, performance, and source code availability, monodepth2's approach was chosen to be further tested on this dissertation.

3.2.3 Transfer learning

Regarding transfer learning, papers that apply domain adaptation and randomization are presented. Even though due to time constraints only the general case of transductive transfer learning is applied to this dissertation work, a brief overview of these papers is pertinent to understand this work's results implications. Domain adaptation, according to Csurka [107], refers to the case of transfer learning for a classifier where labeled data from a source domain is leveraged for a target domain with unseen or unlabeled data. In short, modifications are done to the parametrization of the classifier itself to accommodate the data domain shift. On the other hand, Tobin [108] proposes the concept of domain randomization where the idea is to induce environment variability on the source domain to ease posterior model generalization on the target domain.

In [109] an approach for making an object detector more generalizable across data domains from different distributions is studied, with a focus on both image-level shift, which includes illumination, and instance-level shift, which deals with the object's appearance. The Faster R-CNN architecture was adapted with the concept of H-divergence theory [110] which measures divergence between data samples of different distributions. One of its core ideas is to take into account the losses of a smaller labeled subset of the to-be adapted domain. In their work, the model itself took shape by applying adversarial training. Tests were performed between Sim 10k and Cityscapes, and between KITTI and Cityscapes, with the first case presenting an increase of 8.8 percentage AP points on "car"on Cityscapes and 8.6 percentage points for the second case.

Sun and Saenko [111] studied the task of domain adaptation for 2D object detection where the model is fully trained only on 3D synthetic models available on the web, and later evaluated on the Office [112], a real-world dataset which consists of images containing single

objects. The authors develop their object detection method, which is based on an adapted Linear Discriminant Analysis as a feature extractor and adapt their algorithm by including the features' covariance from both data domains into their model.

Hsu et al. [113] approach the domain adaptation task for object detection by dealing with three domains: a source, synthetic, and a target, where the middle one is created on the run and serves to gradually adapt features from the source to the target. On their work, this is done by combining adversarial learning to the Faster R-CNN with a CycleGAN and evaluated between KITTI, Cityscapes, Foggy Cityscapes, and BDD100k. Overall, a significant metric improvement is observed compared to training from scratch, with around 10 AP percentage points increase.

For depth estimation and optical flow, Mayer et al. [114] presented an extensive study on which aspects of a synthetic dataset can best help generalize deep CNN models, with a series of tests in synthetic and real-world datasets. In short, the synthetic dataset features which improve performance the most are:

- Scene priors similarity (e.g. similar location placement of roads and sidewalks);
- Diversity of brightness, contrast, colors, color noise, position shift, rotation, scaling, texture, lighting complexity;
- Replication of camera modeling and flaws (sensor, lens distortion, artifacts);
- Rather unexpected, but scene realism is not one of the cores which influence the model the most.

And from the viewpoint of the algorithm:

- At which point the target dataset is introduced to the model. At early stages of learning, lower sophisticated datasets are desired, while for later stages the opposite is true;
- Scheduling the learning rate, i.e., decreasing it after a certain amount of iterations.

Zhao et al. [115] presented a supervised approach for monocular depth estimation with the domain adaptation framework Geometry-Aware Symmetric Domain Adaptation Network, which exploits ground-truth from synthetic data and epipolar geometry from real-world data in an adversarial manner. On their work, both synthetic-to-real and real-to-synthetic translations are performed, and as such two depth estimators are modeled, with the final depth prediction being the average between both. The model was evaluated on KITTI and used virtual KITTI for the synthetic part, and compared to other approaches that employed no synthetic data, an improvement of around 10 percentage points on the error metrics is observed compared to the previous best listed on the paper. On similar guidelines, Atapour-Abarghouei and Breckon [116] also present a monocular depth estimation model with adversarial training on synthetic data with depth ground-truth. A custom synthetic data based on GTA V and KITTI are evaluated, with their approach improving also around 10 percentage points on the error metrics compared to the previous best listed on the paper.

On [117], work has been done on further incrementing the training dataset, which regards domain randomization. In this work, the authors add varied synthetic objects on top of Virtual KITTI images to force the model to generalize for different objects, object positions, texture, visibility of ground plane, and illumination points. It is noted that the addition of synthetic objects indeed aided R-FCN and SSD detectors by around 8 AP percentage points, but did not increase for Faster R-CNN, the best performing model.

The work that most closely relates to this dissertation is that of Brekke et al. [118], which presents an open-source simulation toolkit for CARLA and experiments with an AVOD-FPN architecture [112] for 3D object detection of cars and pedestrians. Images and LIDAR data were used from their CARLA-based dataset and evaluated on KITTI. In their work, transductive transfer learning is performed from their synthetic dataset to KITTI and compared with training from scratch on KITTI. When fine-tuning from their synthetic dataset, the pedestrian category is, on specific cases, on average 10 percentual points of 3D AP higher. For most of their tests on cars, however, a decrease of around 5 percentual points of 3D AP is noticed.

4 MATERIALS AND METHODS

In this section, the methodology and materials used to produce the masters' work are presented which regards hardware, software, datasets, and algorithms.

4.1 METHODOLOGY

A high-level overview of the dissertation's methodology is presented in Figure 4.1. Overall, the main idea consists of three steps: creating a synthetic dataset, training and testing algorithms on this environment, and then evaluating adaptation of those algorithms on real-world data. The expectation is that, by somehow adding synthetic data in the learning pipeline, the proposed algorithm's performance will improve on the real-world data. For that, the main guidelines to verify this assumption are the metrics covered in Section 2.4 of this dissertation's work, along with a detailed analysis of the aspects of each data source itself.

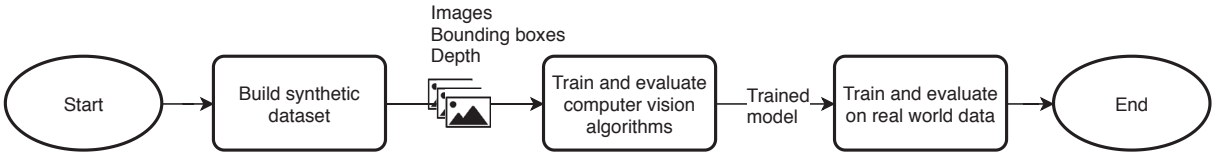


Figura 4.1: High-level view methodology flowchart
Source: The author (2020)

4.2 MATERIALS

For pre-processing the data and models training and evaluation, the hardware shown in Table 4.1 were used on Windows 10 and Ubuntu 18.04 operating systems:

Tabela 4.1: Hardware specification
Source: The author (2020)

	Video graphics card	Processor and CPU
Notebook	Nvidia GeForce GTX1050	Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
Server 1	Nvidia Tesla P100	Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz
Server 2	Nvidia GeForce RTX2080	Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz
Server 3	Nvidia GeForce RTX2060	AMD FX(TM)-8350 CPU @ 2.80GHz

As for software, CARLA 0.9.6 simulator [54] and Unreal Engine 4.20 were used to create synthetic data with Python 3.6 as the programming language for automating the processes. For applying the proposed computer vision algorithms, Python with Pytorch was used as the base framework, with the object detection model Faster R-CNN from Facebook AI research's Detectron2 [119] and monocular depth estimation model monodepth2 from Niantic Labs [120]. The training hyperparameters for each are presented in Table 4.2. As a note, the learning rate on Faster R-CNN is a scheduled one, i.e., it starts at $5 * 10^{-6}$ and increases by $5 * 10^{-6}$ every 20 samples up until it reaches $2.5 * 10^{-4}$; for monodepth2 the learning rate is constant.

Tabela 4.2: Algorithms hyperparameters and requirements
Source: The author (2020)

	Faster R-CNN	monodepth2
Input resolution	1920x1280	1024x320
Optimizer	SGD	Adam
Learning rate	$2.5 * 10^{-4}$	$1 * 10^{-4}$
Batch size	2	1
Backbone	ResNet50 + FPN	U-Net + ResNet18
GPU RAM (train | inference)	2817 | 1355 MB	2683 | 1005 MB

4.2.1 real-world and synthetic data

In this subsection, an overview of the activities performed to curate both the real-world and synthetic datasets are presented. On the real-world data emphasis is given on the steps necessary to adequate the data for the computer vision algorithms; as for the synthetic, focus is given on the methodology to create the data.

4.2.1.1 real-world data

Waymo Open released its first version of the dataset v1.0.0 in August 2019, however at the time of writing a newer version, v1.2.0 had already been released in March 2020, with more images and annotations. For simplicity, and due to time and data storage constraints, experiments on this dissertation work are based on v1.0.0.

As described on Waymo Open's paper [66], the data-capture vehicles are equipped with five cameras, labeled FRONT, FRONT_LEFT, FRONT_RIGHT, SIDE_LEFT, and SIDE_RIGHT, as shown in Figure 4.2. All cameras share the same horizontal Field of View (FOV) of $\pm 25.2°$, with cameras FRONT, FRONT_LEFT and FRONT_RIGHT sharing the same resolution of 1920x1280, and cameras SIDE_LEFT and SIDE_RIGHT with 1920x1040 resolution. To simplify the analysis, only data from the FRONT camera is considered.

A short description of the raw data structure of Waymo Open is presented. There are 40 TAR files of 25 GB each totaling 950 GB, which are divided natively into 32 for training and 8 for validation. Compressed in each TAR file are around 25 segment TFRECORD files, a binary format proprietary from Google which can store all necessary dataset information such as images, annotations, and metadata. Inside each TFRECORD segment file, a varied number of video sequences are stored with around 200 frames each. Hence, the conversion process follows the pseudocode shown in Listing 4.1.

```
1 For each TAR file, extract temporary TFRECORD files;
2       For each TFRECORD file, extract video sequences;
3             For each video sequence, save one frame data at a time;
4             Erase temporary TFRECORD files;
```

Listing 4.1: Extracting raw data from Waymo open

To parse the raw data from each TFRECORD file, sample modules are provided in Waymo's public GitHub repository `https://github.com/waymo-research/waymo-open-dataset`. Based on them, scripts in Python were created to account for the algorithm in Listing 4.1 while performing the following:

1. Exclusion of frames/segments recorded at night and which contain no bounding box and metadata annotations for the cameras, which are unavailable on v.1.0.0;

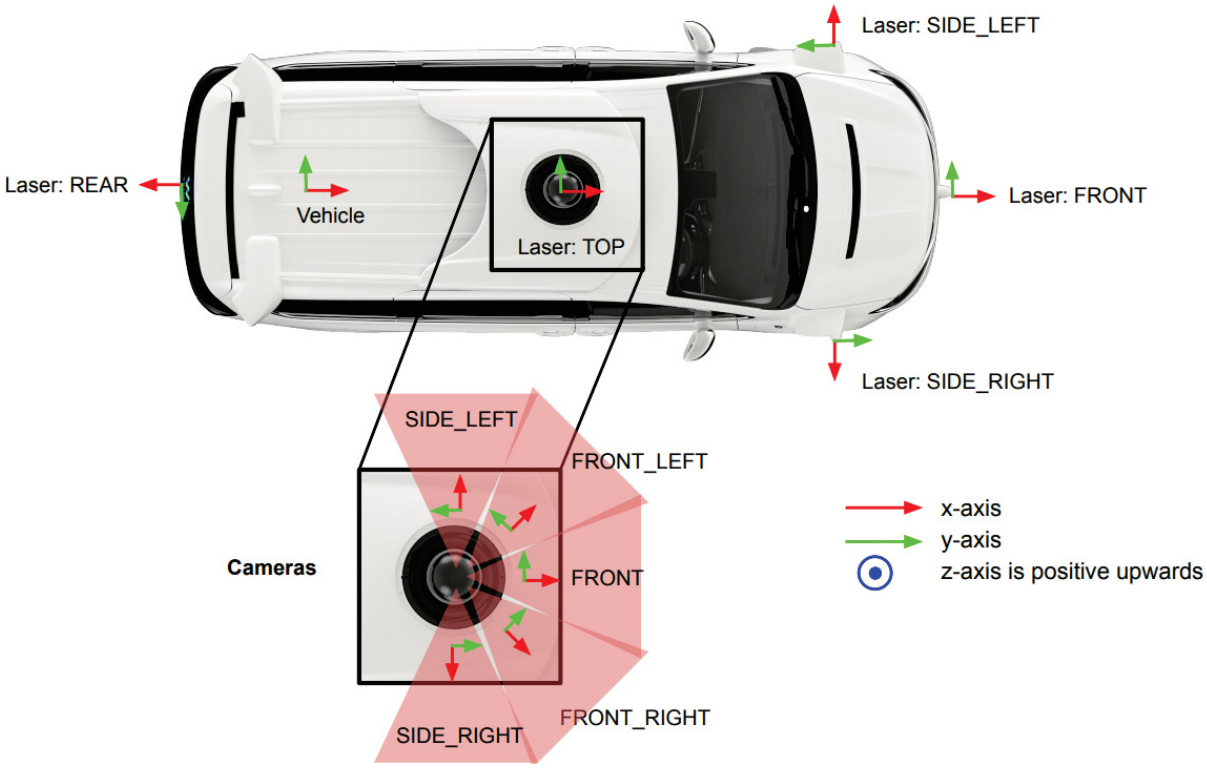


Figura 4.2: Waymo Open - sensor setup
Source: [66]

2. Collection of RGB frames only from the FRONT camera;
3. Extraction of calibrated LASER data to the FRONT camera's reference (sample shown in Figure 4.3);
4. Restriction of 2D Bounding box categories to vehicles and pedestrians;
5. Creation of a separate database file with metadata for each frame.

The idea of creating a database file is to later facilitate access to context data which might be relevant to training and evaluating computer vision algorithms. This way, one does not need to parse the TAR files again, which takes a long time, but rather only perform a few SQL (Structured Query Language) statements. The database file is structured with a single table named "waymo_metadata", which contains the columns unix_microseconds, time_of_day, location, weather, start_of_sequence, and end_of_sequence, as shown in Figure 4.4.

- The first column, unix_microseconds, regards the Unix timestamp contained in the raw file, and is assigned as the frame name;
- time_of_day refers to the time the frame was recorded, which can be night, day, and dawn/dusk;
- location refers to the city the segment was recorded, being location_other, location_phx, and location_sf;
- weather regards the annotator's judgment on it, which can be either rain or sunny;

Figura 4.3: Waymo Open - LIDAR calibrated on RGB
Source: [66]

- Lastly, start_of_sequence and end_of_sequence were assigned while extracting the data and represent whether that frame is the first or last of a given sequence, with 1 for true and 0 for false.

In particular, start_of_sequence and end_of_sequence are essential for video-based monocular depth estimation algorithms where correct referencing of anterior and posterior frames of a given frame is needed. Such metadata is necessary because the number of segments and sequences vary for each TAR file, thus not being possible of directly referencing frames with sequential indexes.

Snapshots of Waymo Open v1.0.0 are shown in Figures 4.5 to 4.7, with varied time_of_day, weather, and location conditions. On a note, on v1.0.0 some frames have incorrect weather labeling, e.g., the picture shows rainy weather but on the TFRECORD it is labeled as sunny.

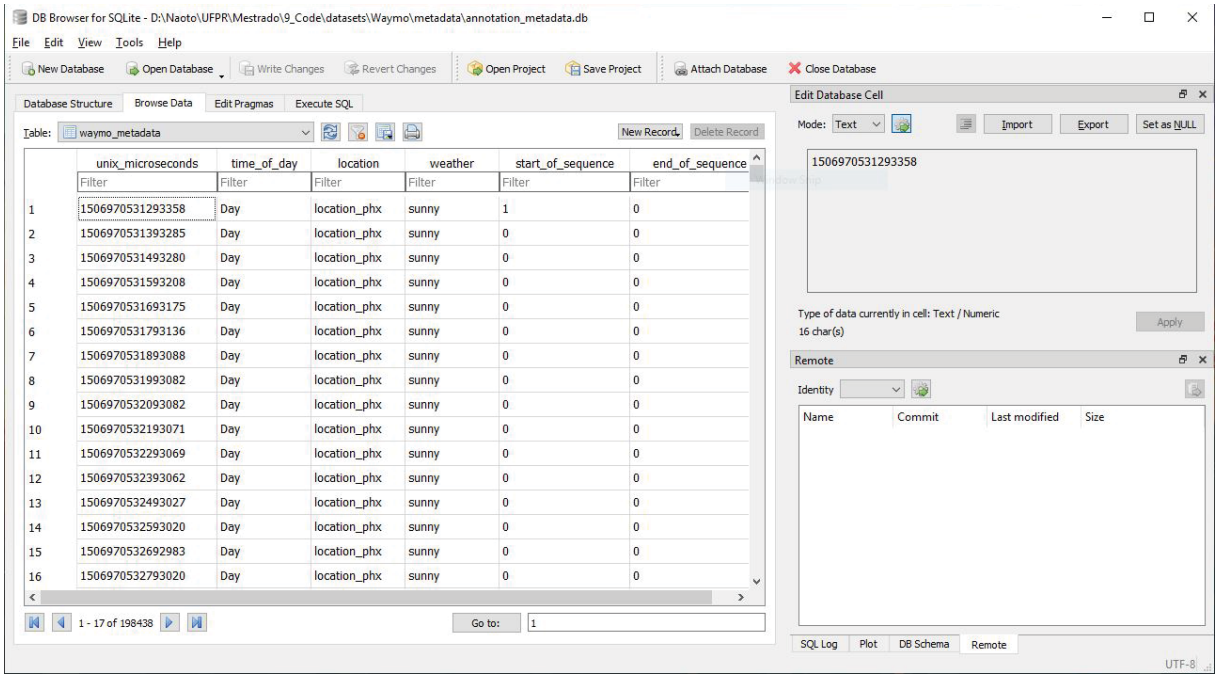


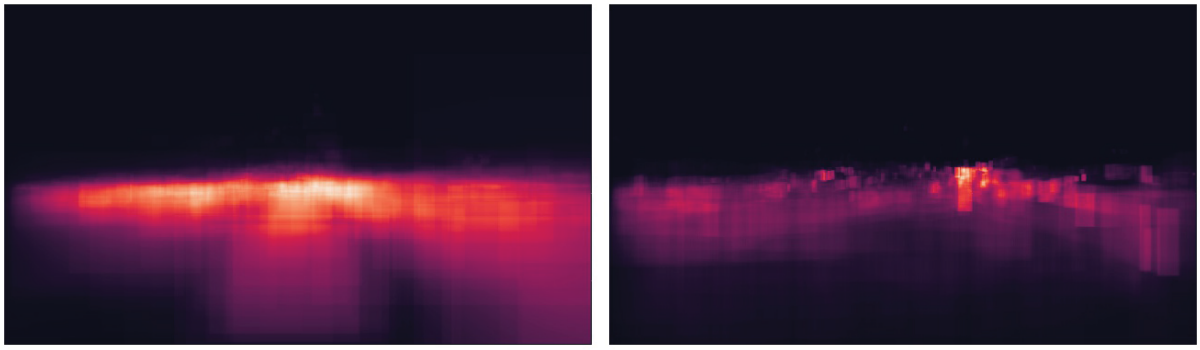
Figura 4.4: Waymo Open - Metadata database file
Source: The author (2020)

Figura 4.5: Waymo Open - Sample 1
Source: [66]

Figura 4.6: Waymo Open - Sample 2
Source: [66]

Figura 4.7: Waymo Open - Sample 3
Source: [66]

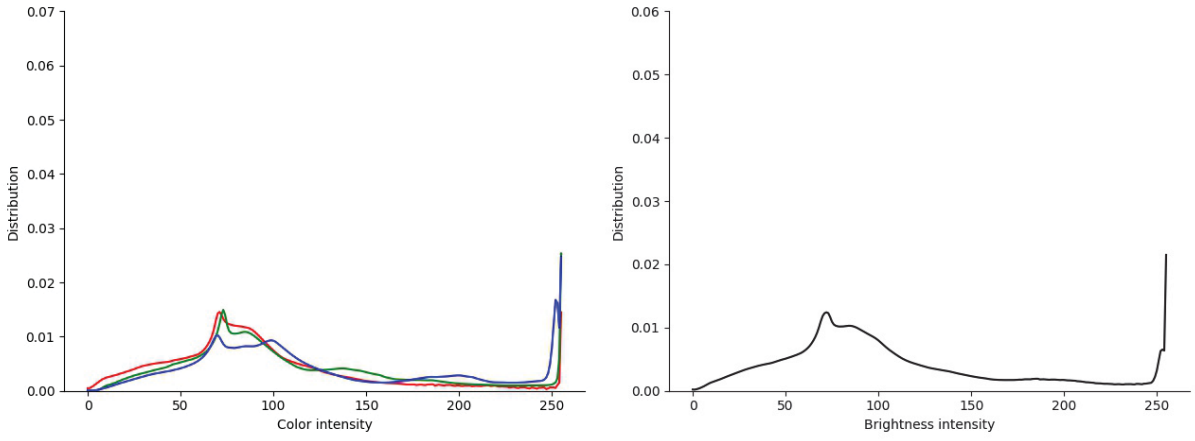
An analysis of Waymo v1.0.0 is presented next. Overall, it contains **312771** and **33506** bounding boxes for vehicles and pedestrians, with $\mathbf{18.33 \pm 12.41}$ and $\mathbf{1.96 \pm 3.38}$ vehicles and pedestrians annotations per frame. As a short comparison, on v1.2.0 there are 198068 frontal camera frames with 3575747 and 918149 vehicles and pedestrians bounding boxes. By observing the 2D bounding boxes heatmaps from Waymo v1.0.0, shown in Figure 4.8, it is noted that both vehicles and pedestrians are well spread across the horizon on the entire lower half of the scene, except for some specific regions in the horizon.



(a) Vehicles (b) Pedestrians

Figura 4.8: Bounding boxes location heatmap for vehicles (a) and pedestrians (b) on Waymo v1.0.0 frontal camera
Source: The author (2020)

in Figure 4.9 the color histograms for Waymo v1.0.0 are presented. It presents an overall high-intensity color profile, which indicates that there are overall brighter scenes, verified by analyzing the probability density function (PDF) of the histogram. By considering that bright scenes are those with gray-scale intensity higher than 128 and darker lower than it, then the PDF for dark and bright scenes in Waymo v1.0.0 is 0.7288 and 0.2712. Besides that, Waymo v1.0.0 also presents two peaks at around 75 and 255 color intensity, pinpointing a possible bias to these color profiles.



(a) RGB (b) Grayscale

Figura 4.9: RGB (a) and grayscale (b) histograms for Waymo v1.0.0
Source: The author (2020)

In conclusion, raw data from Waymo v1.0.0 was parsed and converted to formats suitable for the algorithms used in this work, with output data being 17062 RGB frames from the

FRONT camera, calibrated LASER depth for it, 2D bounding box annotations for vehicles and pedestrians, and a metadata database file. The distribution of bounding box annotations seems sparse and dense, which is a point in favor of this data's complexity.

4.2.1.2 Synthetic data

The idea of creating a custom CARLA dataset revolved around considering a sensor setup similar to that of the real-world. In this work's case, the frontal dashboard camera setup of Waymo Open was replicated, with a depth sensor sharing the same position as the camera, as shown in Figure 4.10.

Figura 4.10: RGB frame (left) and dense depth (right). For easier visualization, depth is shown in log scale
Source: The author (2020)

CARLA by default provides five maps, named Town 01 to 05, with varied worlds that were used as the basis for creating the synthetic data. A brief description and snapshot of each are presented in Figures 4.11 - 4.15.

Figura 4.11: CARLA Town 01 - Residential area with plenty of green areas
Source: The author (2020)

Figura 4.12: CARLA Town 02 - Residential area of a medium-sized city
Source: The author (2020)

Figura 4.13: CARLA Town 03 - Residential area with skyscrappers and tunnels
Source: The author (2020)

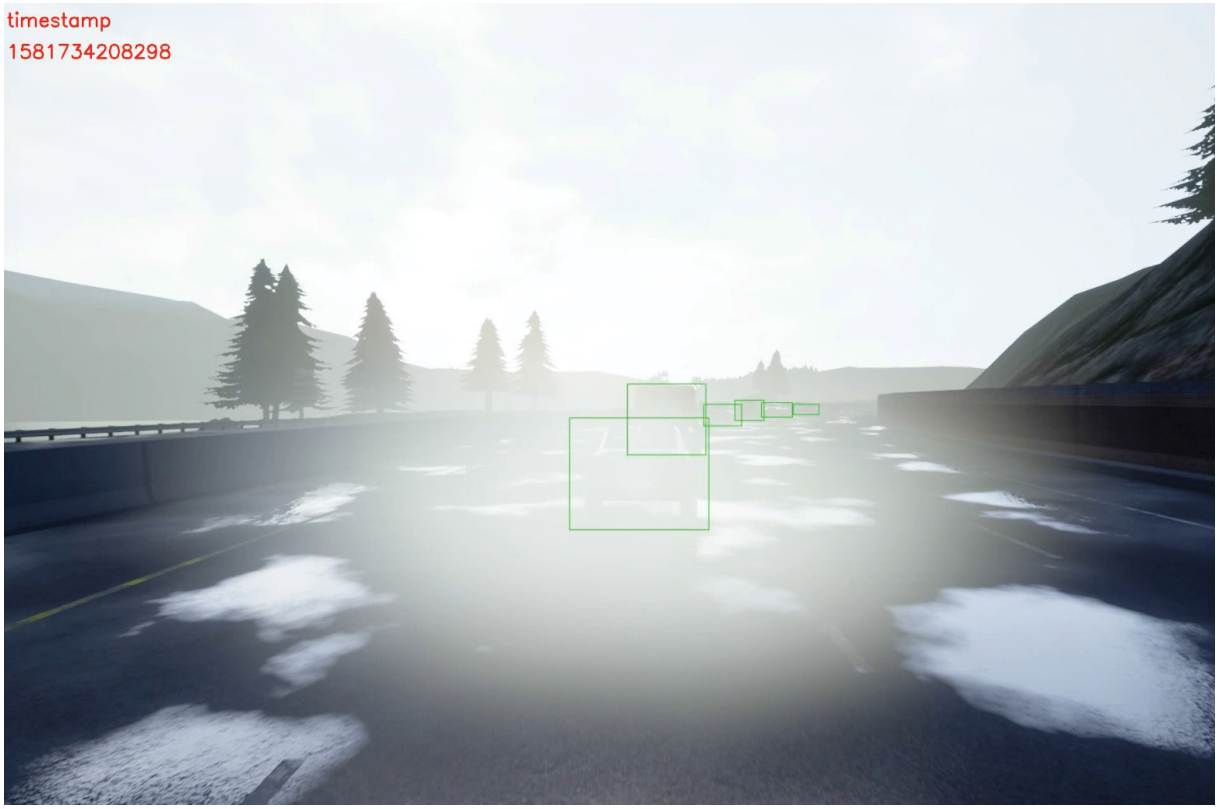


Figura 4.14: CARLA Town 04 - Sparse area with highways and a small residential area
Source: The author (2020)

Figura 4.15: CARLA Town 05 - Urban area with highways and skyscrappers
Source: The author (2020)

Similar to Waymo Open, two categories of annotated objects are considered in the dataset: vehicles and pedestrians. For vehicles, cars of random models and colors are included as well as vans and trucks. For pedestrians, children and adult models are present. Examples of vehicles and pedestrians are shown in Figure 4.16 and 4.17, with green and red boxes locating each respectively.

Figura 4.16: Vehicles and pedestrians - sample 1
Source: The author (2020)

Figura 4.17: Vehicles and pedestrians - sample 2
Source: The author (2020)

Five kinds of weather are applied, of which four were customized for this dissertation work - early morning, sunny midday, afternoon, and almost night. The fifth weather is a default one provided for each map by the CARLA's developers, with a sample of each shown in Town03 in Figure 4.18. On a side note, by default vehicles and lamp assets available on CARLA 0.9.6 don't contain dynamic lighting, which hinders the creation of data at night, which is inherently more challenging for computer vision tasks. Therefore, custom dynamic lighting assets were developed in Unreal Engine and later exported to a then custom CARLA version, as shown in all previous pictures.

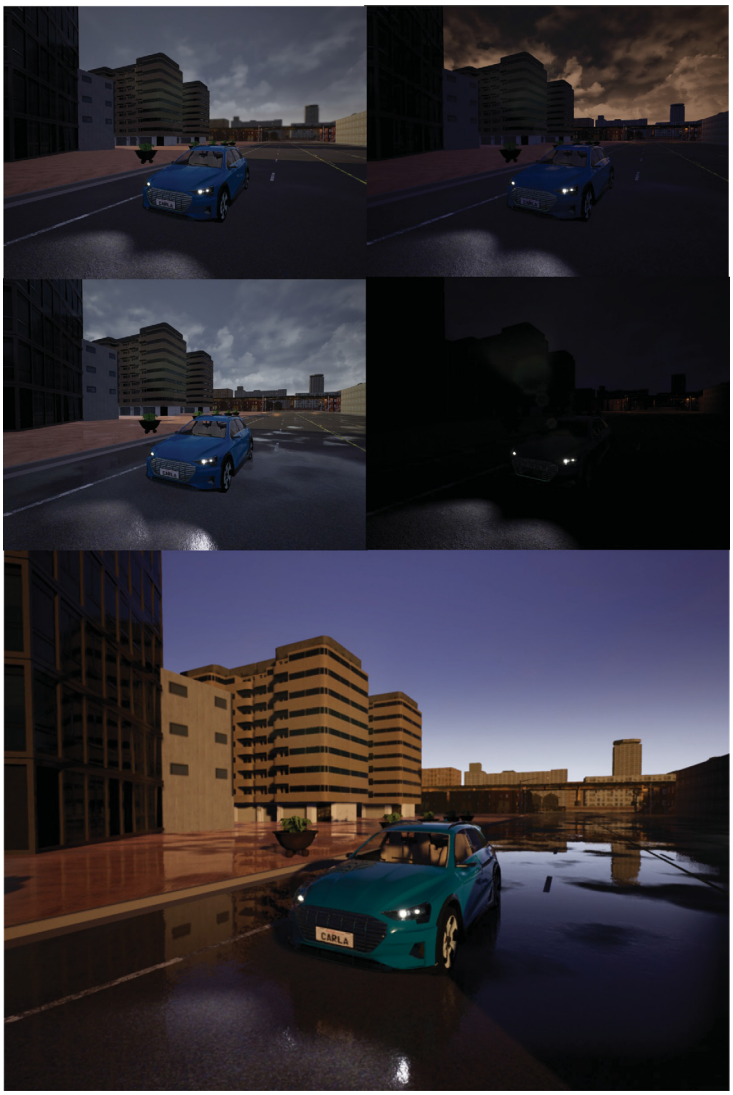


Figura 4.18: Custom weathers developed for CARLA
Source: The author (2020)

Special care also had to be taken when populating the town with vehicles and pedestrians. In the case of vehicles, spawning too many leads to the occurrence of traffic jams, invalidating the whole capture sequence with still repeated frames, and for pedestrians collision errors become more frequent, making the data less realistic. On the other hand, spawning few units leads to a sparse dataset with a low count of annotated objects per frame, i.e., more negative samples than positives, which is undesired for training most object detectors. In the end, iterative testing had to be done for each of the five maps to find a good balance between traffic flow and dense presence of objects. The amount of objects for each town is shown in Table 4.3.

Tabela 4.3: Objects count per town
Source: The author (2020)

Map	Vehicles	Pedestrians
Town 1	77	153
Town 2	41	68
Town 3	114	103
Town 4	191	69
Town 5	120	119

For data collection, the Python APIs provided by the CARLA developers contain modules for data collection which include 3D bounding boxes, depth maps, semantic segmentation, and EGO vehicle speed. The bounding box module, however, does not directly provide valid annotations for the desired camera but instead does so in the context of world coordinates. Thus, several hard-coded rules are established to account for occlusion between objects and for the cases where the boxes have a significant area shown on the frame. In the end, types of data captured are the following:

- RGB frame;
- Dense depth frame;
- 2D bounding box annotations for vehicles and pedestrians;
- Timestamp in UNIX time.

By default, the simulation engine forces it to process frames at a given FPS value, which if left at, can lead to various issues on data capture such as data loss and/or corruption. This happens due to the inherent time needed for processing and saving data, which sometimes is enough to desynchronize with the callbacks for the data capture method and the world state step. Therefore, the vehicle simulation is set to a synchronized mode, so that the simulation step advances only after a call to the API is given, guaranteeing sufficient time to capture and process all data from a snapshot before proceeding to the next frame, thus minimizing data loss risk.

The data capture procedure is done in several steps, shown in Figure 4.19. First, the map is loaded and the actors (vehicles and pedestrians) are spawned with an autonomous controller for each, which dictates their movement pattern. Next, one of the customized weathers is set, and finally, a random EGO vehicle captures data for a certain amount of frames. In the end, 13 EGO vehicles each capture 60 frames of data for 5 kinds of weather for 5 towns, which amounts to 19500 frames and 325 different scenes. On a side note, one random vehicle at a time captures 60 frames before switching to the next instead of all vehicles in parallel so that computational resources are eased since the CARLA simulation by itself is intensive for common commercial computers.

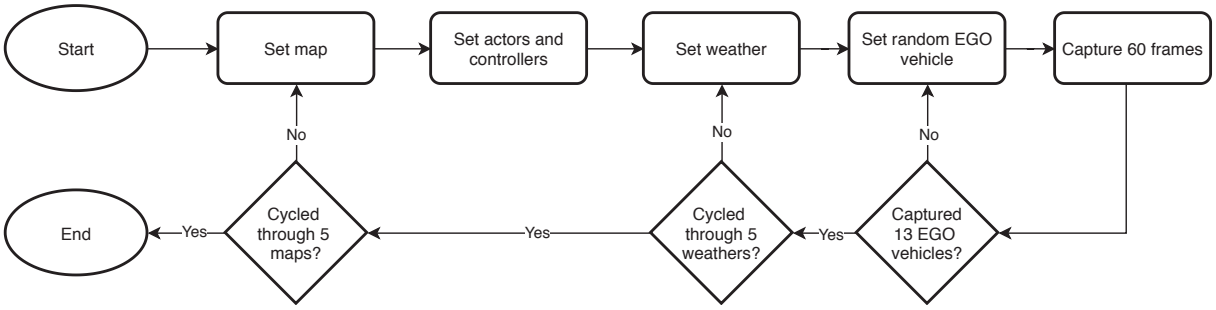


Figura 4.19: Data capture flowchart for CARLA
Source: The author (2020)

Finally, a single Hierarchical Data Format (HDF5) file is created on runtime which unifies all data, easing organization as opposed to a single file per frame. The HDF5 file was structured with one group for each type of data and indexed according to the UNIX timestamp which it was recorded with. In Figures 4.20 and 4.21 examples of the data structure are shown for timestamps and bounding boxes, respectively.

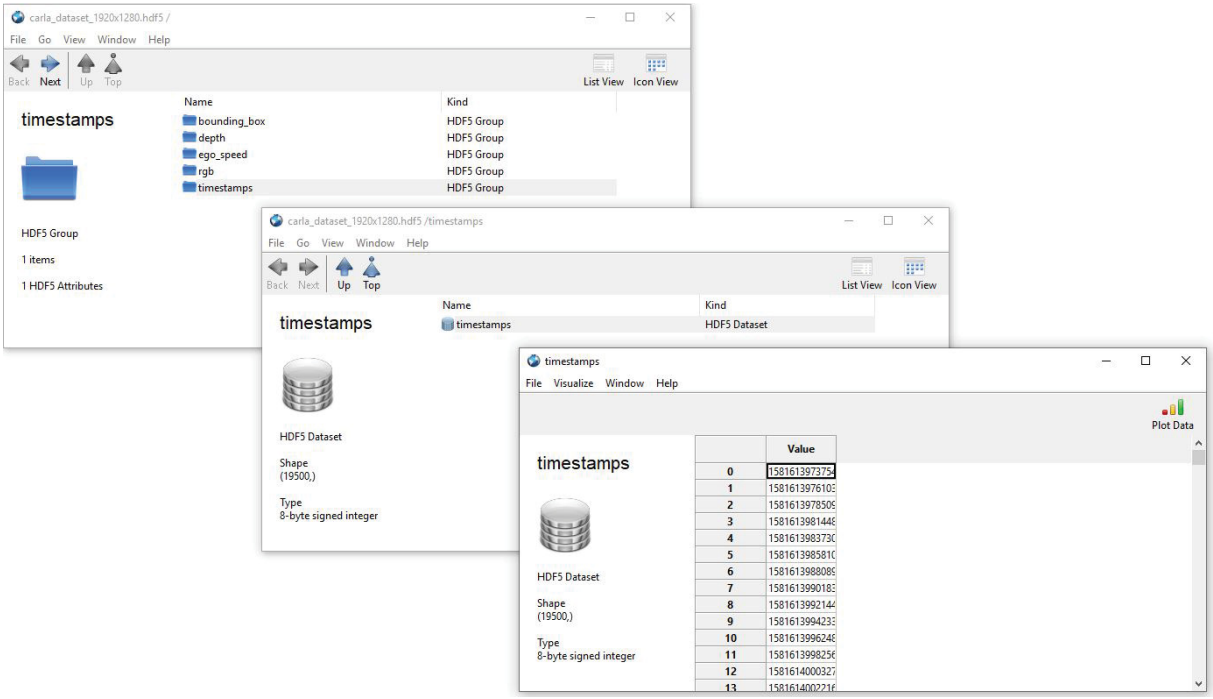


Figura 4.20: CARLA HDF5 timestamps structure
Source: The author (2020)

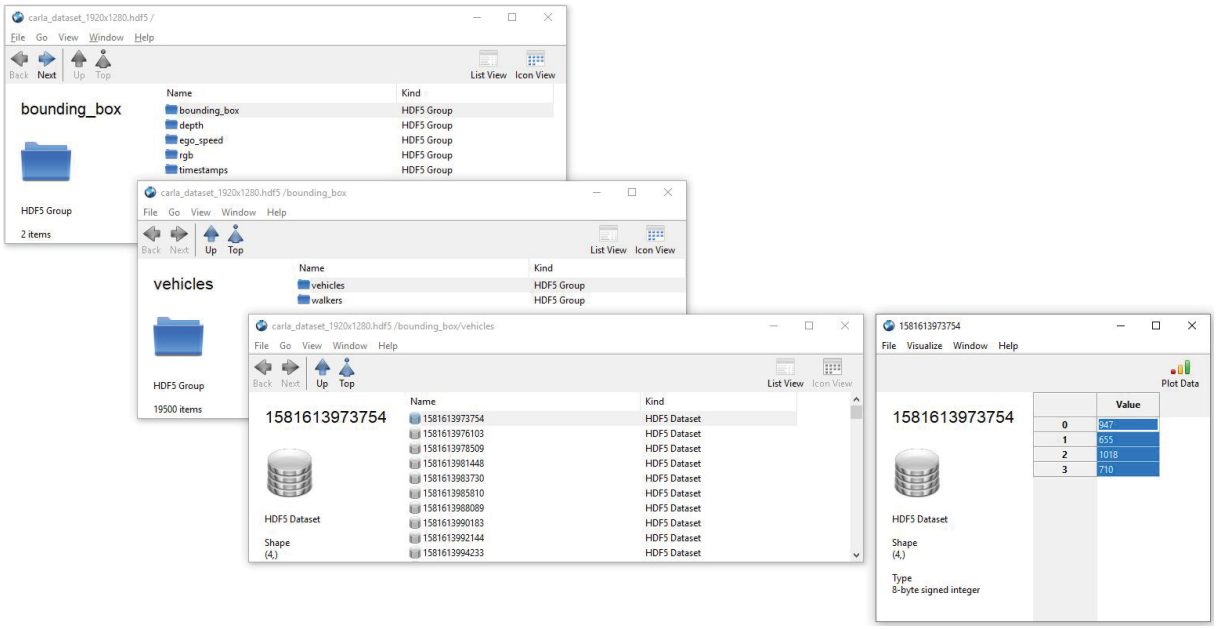
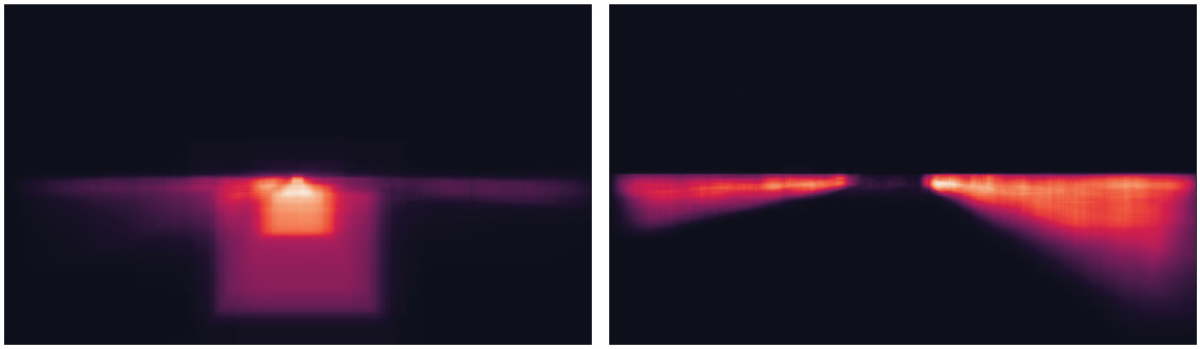


Figura 4.21: CARLA HDF5 bounding box annotations structure
Source: The author (2020)

More details regarding the synthetic dataset are presented. In Table 4.4 the distribution of annotations for each town is presented.The number of bounding boxes presents a high standard deviation, with values close to the mean. This indicates that there is a good distribution between frames with low and high object density. Overall, the average number of annotations per frame for vehicles is 3.01 ± 2.51 and for pedestrians is 1.38 ± 1.70, with a total of 58846 and 26973 bounding boxes for each class. In Figure 4.22 the heatmap for the presence of annotated vehicles and pedestrians is shown. Notably, the pedestrians' density is concentrated on the left and right sides of the image, while the vehicles are on the center. This suggests that a detection algorithm trained on this dataset might be slightly biased on proposing regions in these areas.

Tabela 4.4: Synthetic dataset annotations distribution
Source: The author (2020)

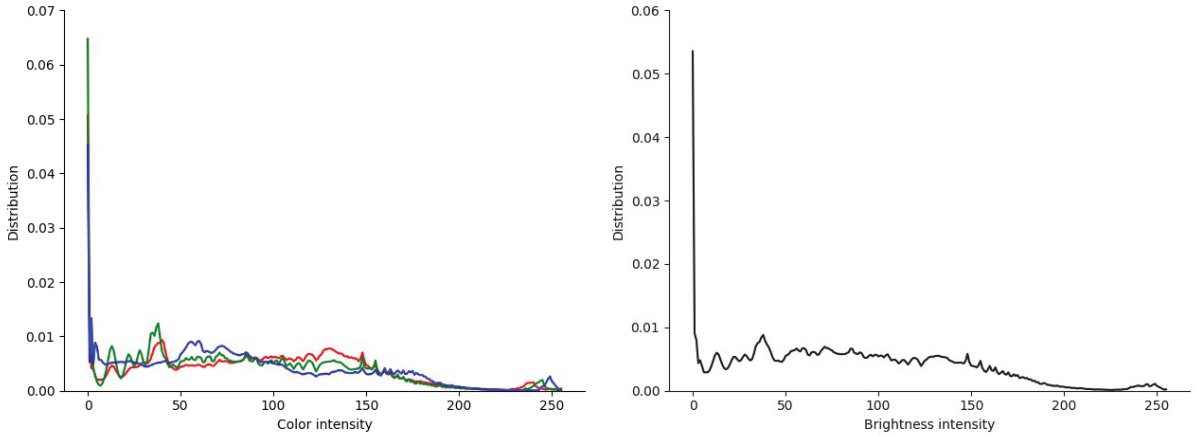
	Vehicles		Pedestrians	
Map	Bboxes	$\mu \pm \sigma$ per frame	Bboxes	$\mu \pm \sigma$ per frame
Town 01	8343	2.14 ± 1.62	7403	1.90 ± 1.90
Town 02	9045	2.32 ± 1.71	8098	2.08 ± 1.54
Town 03	13911	3.57 ± 2.48	3141	0.81 ± 1.32
Town 04	16952	4.35 ± 3.56	2837	0.73 ± 1.69
Town 05	10595	2.72 ± 1.96	5494	1.41 ± 1.60
All	**58846**	$\mathbf{3.01 \pm 2.51}$	**26973**	$\mathbf{1.38 \pm 1.71}$



(a) Vehicles (b) Pedestrians

Figura 4.22: Bounding boxes location heatmap for vehicles (a) and pedestrians (b)
Source: The author (2020)

When analyzing color histograms, the more evenly distributed it is, the better, since it indicates that the images will present higher color variety. In Figure 4.23 the synthetic dataset color density curves are presented and exhibit a darker profile, with the intensity in between having a low variation, verified by the PDF of the curve, with PDF for dark and bright scenes being 0.7525 and 0.2475, respectively.



(a) RGB (b) Grayscale

Figura 4.23: RGB (a) and grayscale (b) histograms for the synthetic dataset
Source: The author (2020)

It's important to note, once again, that the dataset was built on top of the latest CARLA version available at the time, 0.9.6, and as such when compared to the latest version released, some of the included novel features of this dissertation (nocturne lighting, 2D bounding boxes) might have been outdated on the simulator's newer versions. At the time of writing, for example, the latest version is 0.9.9 where the night environment is already added.

To conclude the synthetic dataset section: illumination assets were manually added to the simulation engine, which helped on creating more challenging scenarios; scripts to create the scenarios and capture data were developed, and finally a dataset with 19500 frames and dense depth was built by following a sensor suite similar to Waymo Open. Lastly, the script created to set up the simulation and collect data is made open-source on GitHub[1], as well as the created dataset at CARISSMA[2].

[1] `https://github.com/AlanNaoto/carla-dataset-runner`
[2] `https://www.thi.de/go/thi-synthetic-automotive-dataset`

Finally, a recap on the main points between the real-world and synthetic datasets assembled - both present a similar number of frames, sensor setup, and annotations. In favor of Waymo v1.0.0, there are more bounding boxes and they are sparser than the synthetic data, which could provide better learning of region proposal detectors. In terms of color distribution, both present opposite profiles - while Waymo v1.0.0 presents brighter scenes, the synthetic tends to darker ones. About color diversity, the developed synthetic dataset presents more color-diverse scenes than Waymo v1.0.0 due to the histogram's more flat-leveled curve profile. Finally, for the sake of less verbosity, Waymo v1.0.0 will be referred to as Waymo from this point of the dissertation.

5 RESULTS

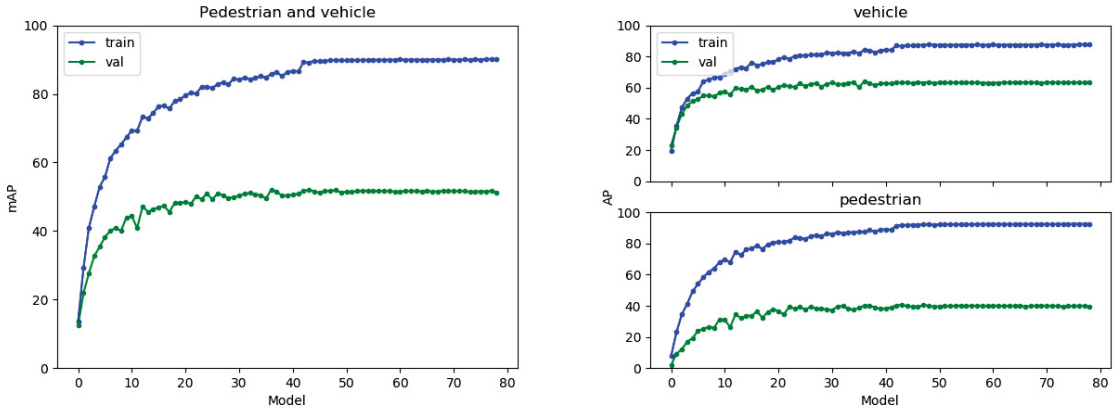
In this section, results from training and evaluating the selected algorithms on the synthetic-based CARLA dataset and WAYMO Open are presented, with analysis ranging from both individual and mixed domains perspective. First, the training progress are shown. Afterward, at the end of each subsection, a compilation of the metrics are presented in Table format.

5.1 OBJECT DETECTION

As explained on section 4.2, the Faster R-CNN with a ResNet50 has been evaluated for the following tests.

5.1.1 Synthetic-based CARLA

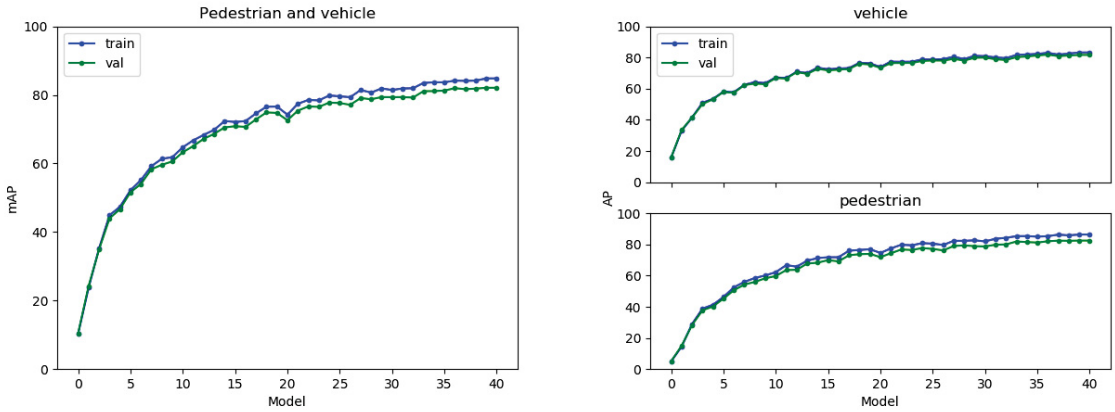
As a first test, the holdout method is applied by separating different video sequences into each dataset split, with data from Towns 01 to 04 for training and Town 05 for validation. In Figure 5.1 the AP and mAP progress during training are shown.



(a) (b)

Figura 5.1: Object detection - Town holdout on synthetic data
Source: The author (2020)

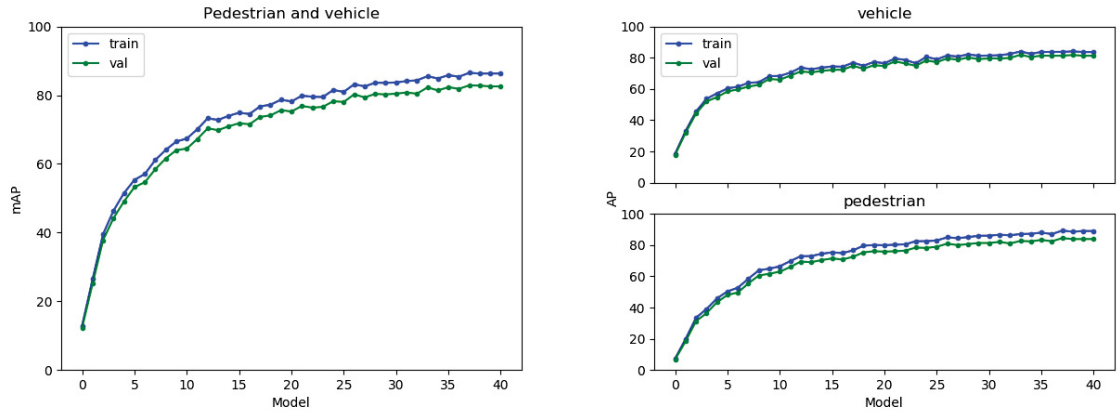
One assumption that could be derived by analyzing the previous results is that the input data used for training is too different from the validation/testing splits, incurring in poor performance. Therefore, to ensure that at least the environment had been seen by the algorithm, the second approach is on shuffling the whole dataset and then splitting it. In Figure 5.2 the AP and mAP progress during training are shown.



(a) (b)

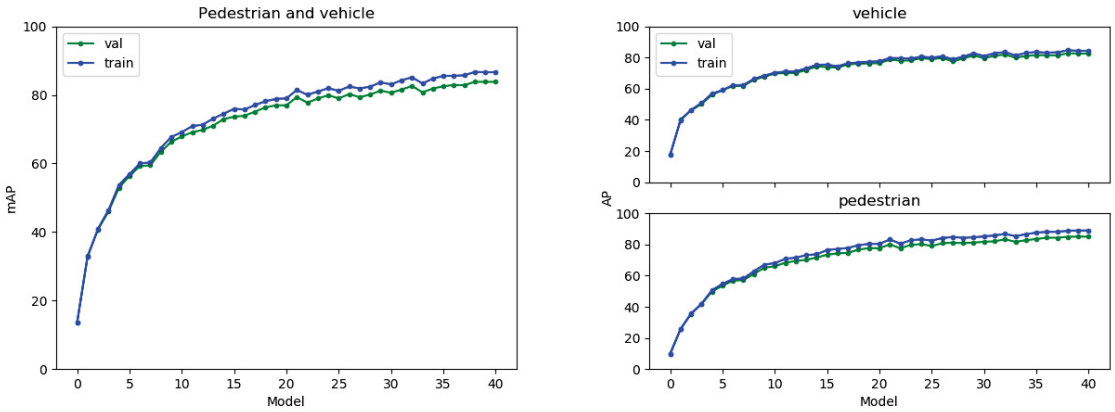
Figura 5.2: Object detection - Shuffled holdout on synthetic data
Source: The author (2020)

By comparing the curve profiles between the data splits, it is notable that since the profiles are too similar one to another, then data leakage between the splits might be happening. Before proceeding, to check if the previous results occurred due to chance or not, the K-FOLD cross-validation method with K=3 was applied on shuffled random frames. In Figures 5.3 to 5.5 the learning progresses are shown.



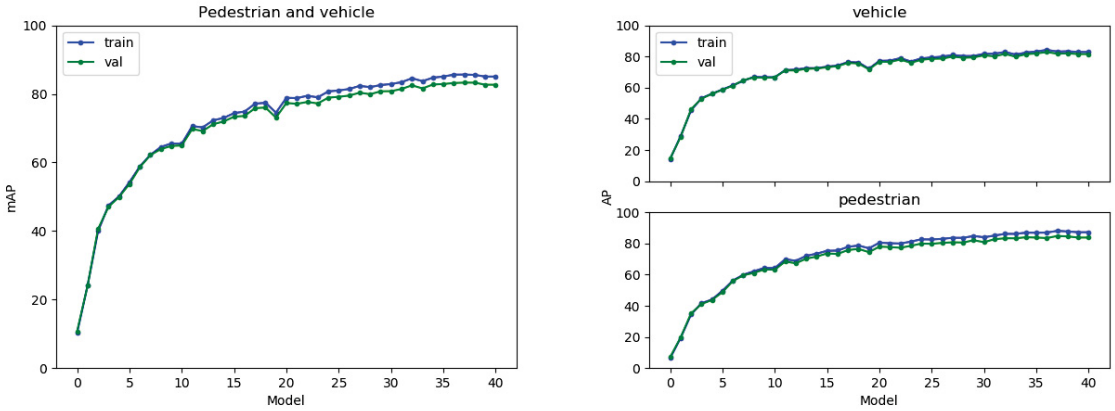
(a) (b)

Figura 5.3: Object detection - Kfold 1 on synthetic data
Source: The author (2020)



(a) (b)

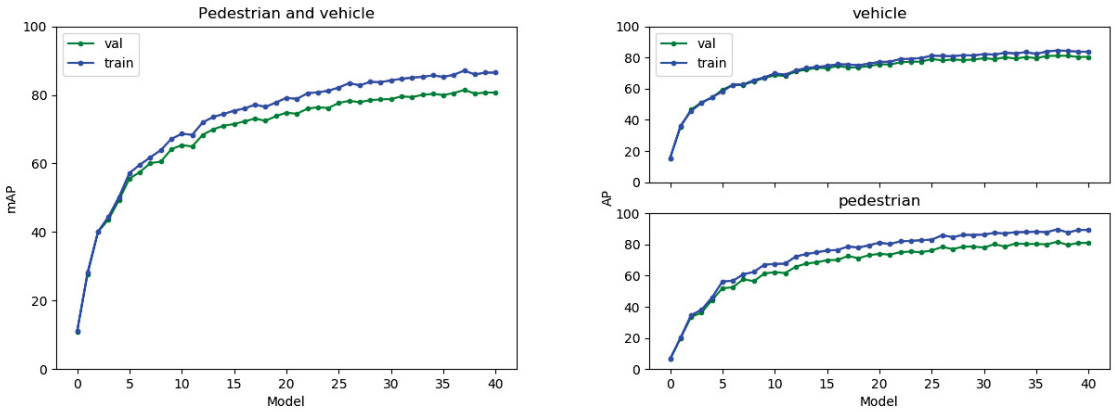
Figura 5.4: Object detection - Kfold 2 on synthetic data
Source: The author (2020)



(a) (b)

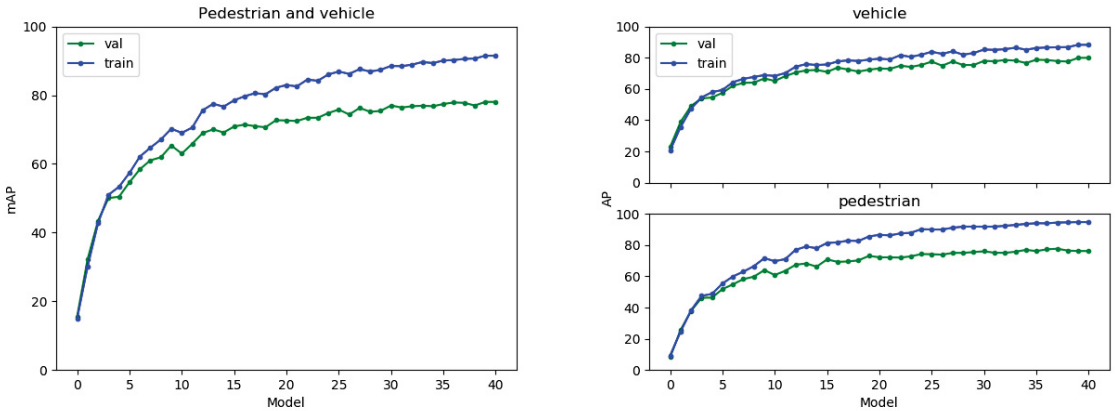
Figura 5.5: Object detection - Kfold 3 on synthetic data
Source: The author (2020)

Since the same issue persists, then the randomness factor can be discarded. Instead, the next approach to solving this problem focus on undersampling the dataset. Since the data is composed of video sequences, then it is inherent that sequential frames will be highly correlated to one another; therefore, three experiments were performed - sampling only every second, fifth and tenth frame for the new dataset, reducing it to 9750, 3900 and 1950 frames. For easier referencing, each subsampling method is respectively named "skip 2", "skip 5"and "skip 10"on this dissertation work. The training progresses are presented in Figures 5.6 to 5.8.



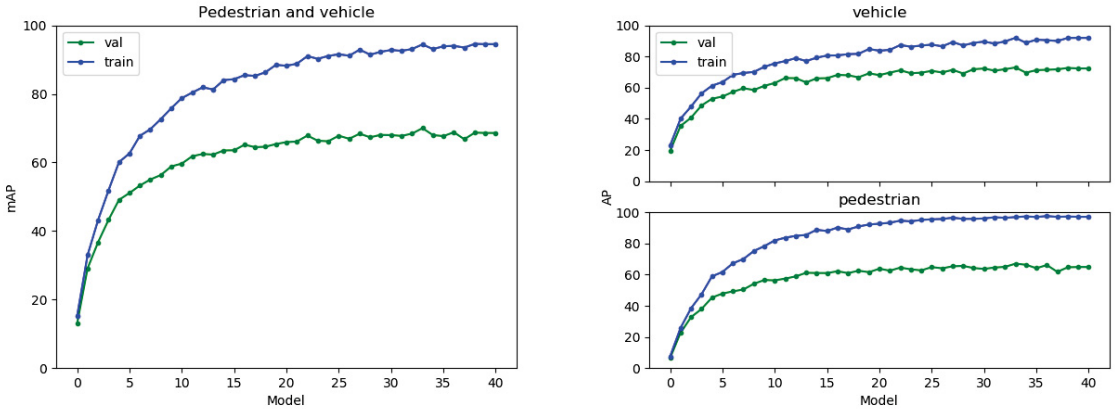
(a) (b)

Figura 5.6: Object detection - Skip 2 on synthetic data
Source: The author (2020)



(a) (b)

Figura 5.7: Object detection - Skip 5 on synthetic data
Source: The author (2020)



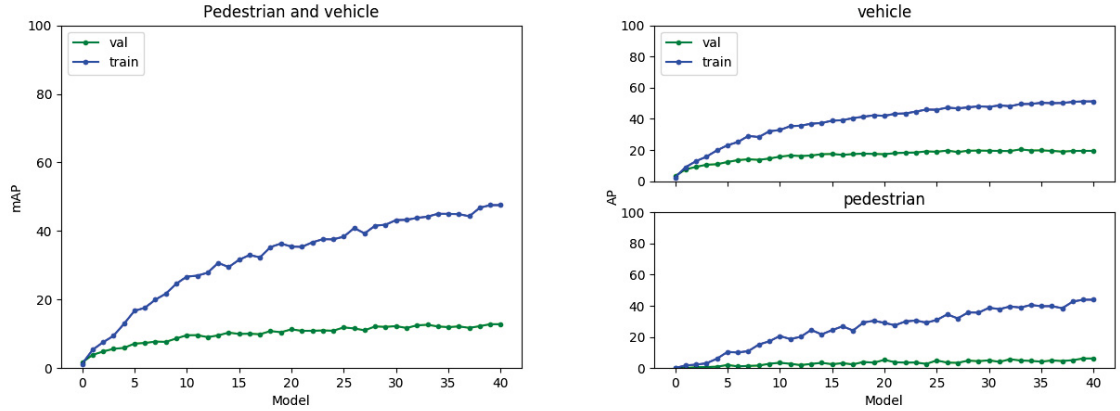
(a) (b)

Figura 5.8: Object detection - Skip 10 on synthetic data
Source: The author (2020)

By observing the curve profiles, the effect of having more spaced frames in the dataset is noticed with the curves for train and validation drifting more with larger spacing between frames. Notably, it is with the dataset when ten frames are skipped that this effect seems to diminish more, which leads to less data leakage at the cost of performance metric on the same dataset. The expectation on doing this is that while the performance on this split is worse than training with the original dataset, better results could be achieved on unseen data from other domains, i.e., the network could generalize better.

5.1.2 Waymo

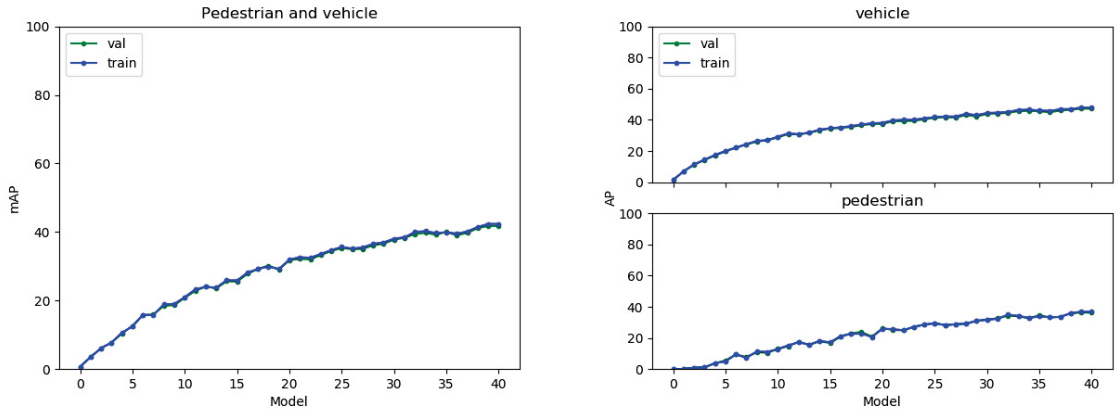
Similar steps were taken into consideration for training models on the Waymo dataset. First, training was performed on the native splits available within Waymo Open where training and validation splits consist of completely different video sequences. The training progress is shown in Figure 5.9. As expected from prior experience on the synthetic data, performance, in this case, is poor as well.



(a) (b)

Figura 5.9: Object detection - Native split of real-world data
Source: The author (2020)

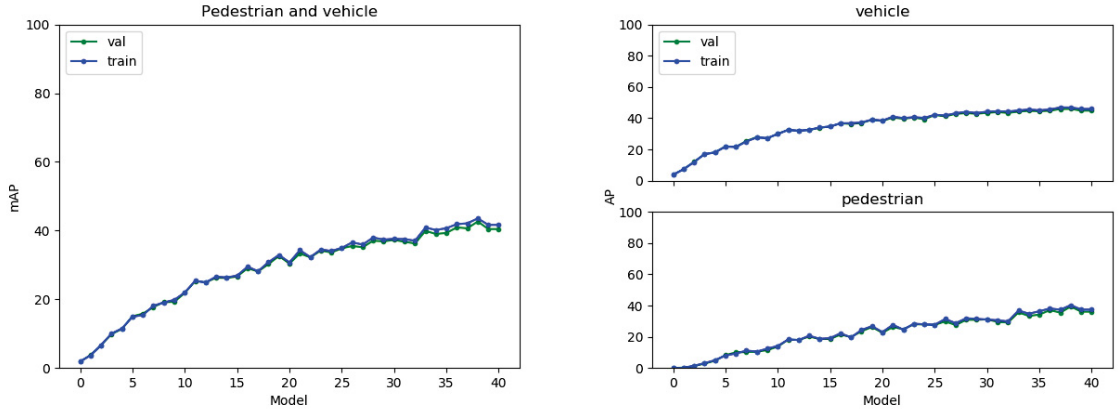
Next, the whole dataset was shuffled to verify the same assumption on the synthetic part. The training progress is shown in Figure 5.10. As with its synthetic counterpart, the profiles are too similar; which can indicate the effect of data leakage.



(a) (b)

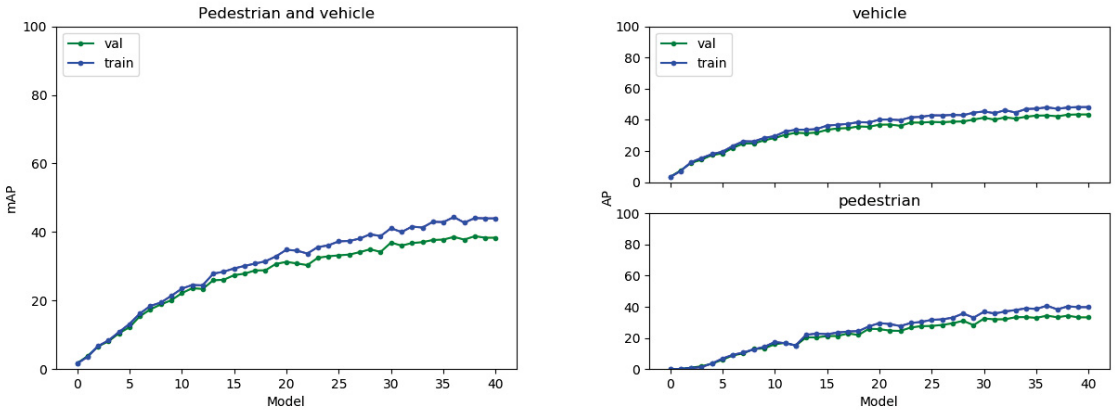
Figura 5.10: Object detection - Shuffled real-world data
Source: The author (2020)

Within the shuffled dataset, three variants were explored on it: skip 2, skip 5 and skip 10. The training progresses for the three variants are shown in Figures 5.11 to 5.13. Observing the charts, skip 10 indicates less of the data leakage effect.



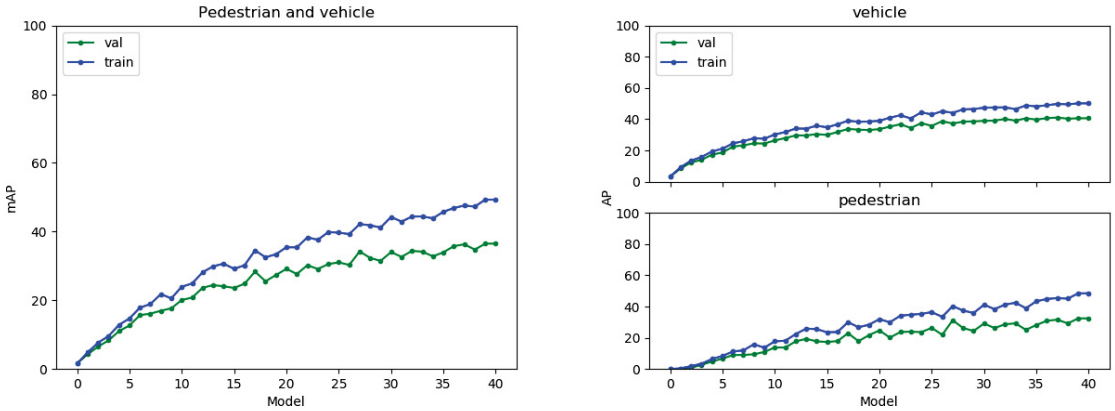
(a) (b)

Figura 5.11: Object detection - Skip 2 on the real-world data
Source: The author (2020)



(a) (b)

Figura 5.12: Object detection - Skip 5 frame on the real-world data
Source: The author (2020)



(a) (b)

Figura 5.13: Object detection - Skip 10 on the real-world data
Source: The author (2020)

One point that was not yet addressed is the class unbalance issue. In regular binary classification tasks, this is solved by performing stratified cross-validation methods, where the number of categories is assigned as roughly the same for each split. On object detection, however, this is not as simple since for each data entry multiple objects of different categories might be present. Therefore, two methods are proposed: undersampling and oversampling.

The idea for oversampling is to simply add repeated frames where the occurrences of the underrepresented category are higher than the other until a similar distribution for both categories is achieved. For undersampling the opposite is performed: frames that contain more of the overrepresented category than the other are removed until a similar distribution is achieved. The workflow for both processes is shown in Figure 5.14. First, a list containing the difference of categories instances is saved for each frame, which enables later lookup. Afterward, either the oversampling or undersampling method is performed. For oversampling, the frames on the list where the category is underrepresented are repeatedly added to the dataset, until the desired category distribution is achieved. In this case, for both methods, the desired distribution is defined as a threshold where the underrepresented category must present at least the same amount of annotations as the other. For undersampling, on the other hand, frames are removed

when there are more instances of the overrepresented category than of the underrepresented until the threshold is met. Contrary to the other method, however, depending on the distribution of the annotations this method can end prematurely without achieving the desired distribution. The described methods achieve satisfactory category balancing, however, some issues arise:

- When oversampling, if the presence of the underrepresented category is too low, then the data might be biased to these samples;
- When undersampling, if the presence of the underrepresented category is too low, then a too small dataset might be obtained;
- Annotation density is not differentiated, and therefore the final dataset can contain mostly sparse annotations.

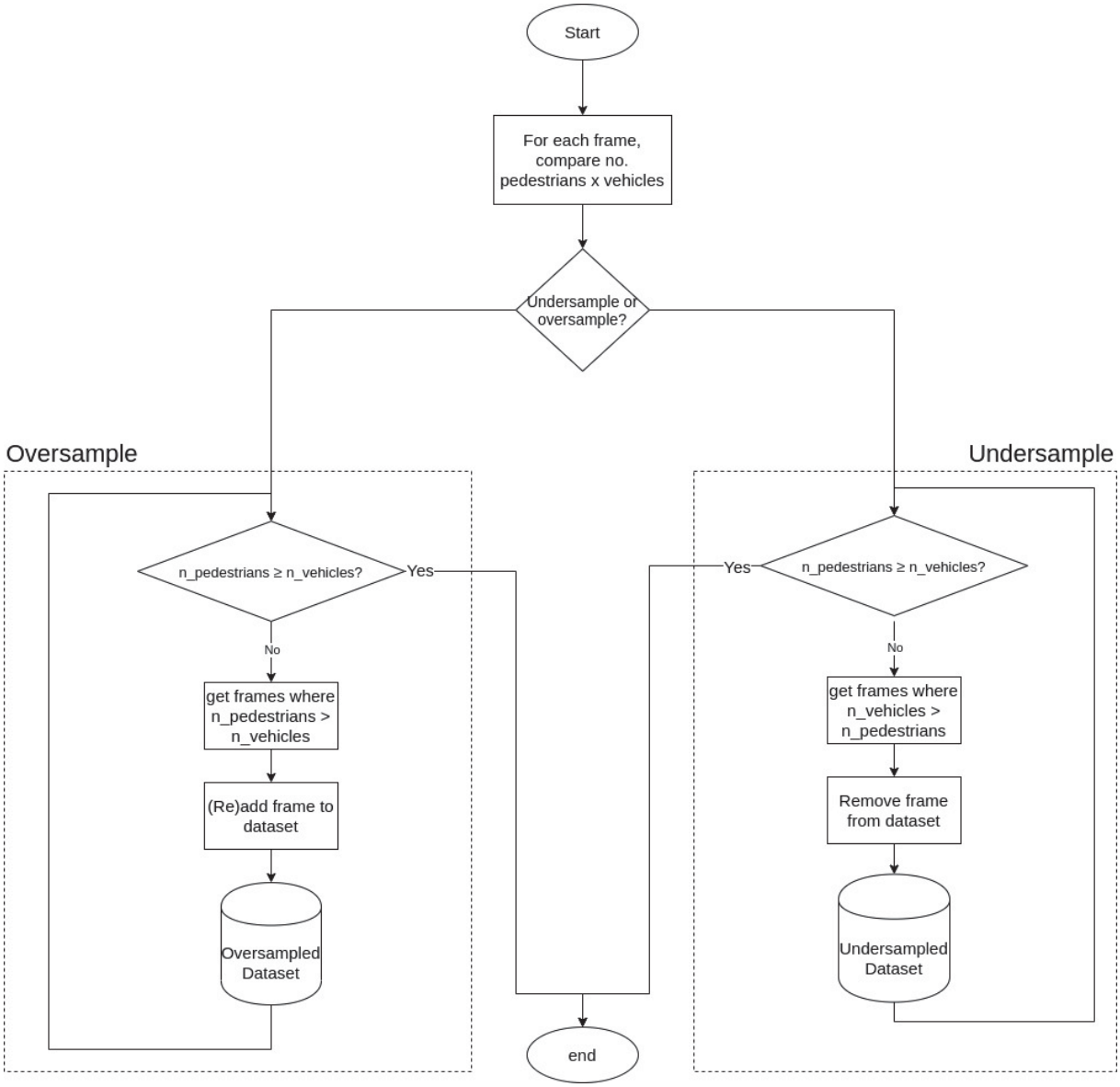
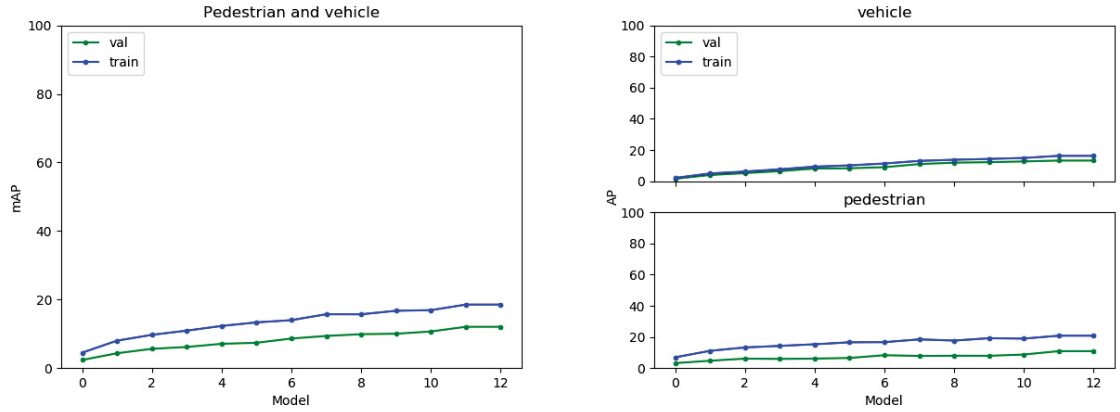


Figura 5.14: Categories balancing workflow
Source: The author (2020)

With that said, both methods were applied on Waymo skip 10 dataset. As expected, when performing the undersampling method, the training data is reduced from 1365 to 141

images, which is too little data to train deep learning models. When oversampling, the frames are increased from 1365 to 6783, which might seem ideal at a first glance; however, the problem is that around 100 frames were sampled repeatedly on the dataset which can lead to further generalization issues. Hence, the oversampled data was used for training and the original validation and testing splits for evaluation. The training progress is shown in Figure 5.15, and as expected, the model did not manage to generalize with this data.



(a) (b)

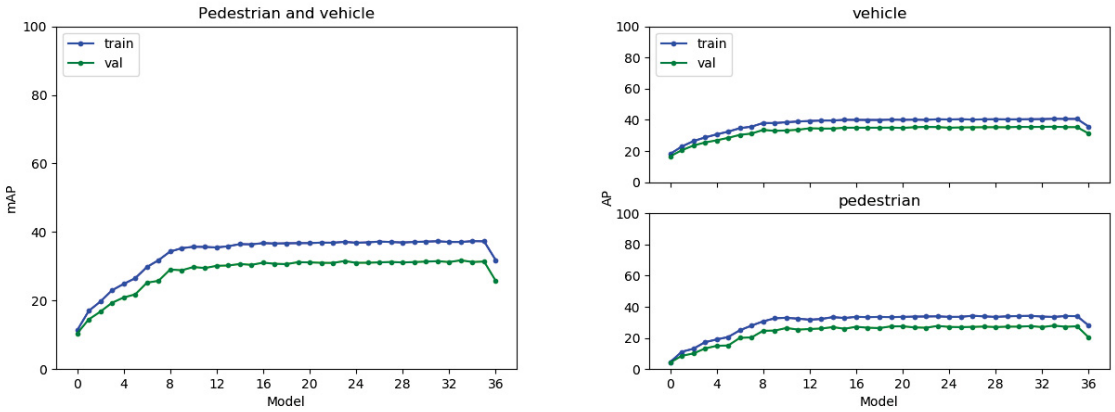
Figura 5.15: Object detection - Oversampling Waymo skip 10 dataset
Source: The author (2020)

A recap on some of the main conclusions of this subsection: the same issue of data leakage that happened on the synthetic part also occurred here, with a potential solution being subsampling the dataset for a higher temporal difference between frames. For that, skip 10 seems to yield the most consistent results, with the downside of reducing the dataset by a factor of ten, resulting in 1707 frames. Efforts on balancing categories distribution were also performed but did not yield satisfactory results, with the undersampling method reducing the data sample to an impracticable size for deep learning and the oversampling not managing to generalize well.

5.1.3 Transfer learning

After having a baseline on training results for both synthetic and real-world data, transfer learning approaches can be analyzed. The ones described In this section are based on transductive transfer learning, where the source and target tasks are the same (detecting vehicles and pedestrians) but the domains are related (synthetic data against real-world data or real-world data of distinct origins).

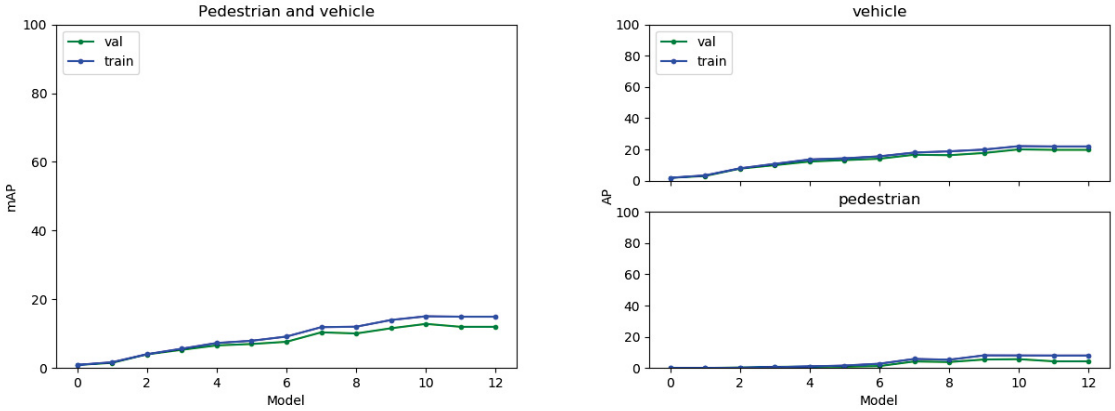
The first approach is on reusing network weights from the Faster R-CNN trained on CARLA skip 10 as a starting point, to then train and evaluate on data from Waymo skip 10. Even though the algorithm reaches a plateau in fewer iterations when compared with training from scratch, no improvement is noted on its performance - instead, a significant decrease is noted with the mAP decreasing from 36.36 to 32.02, which raises questions on the compatibility between datasets. The training progress is shown in Figure 5.16.



(a) (b)

Figura 5.16: Object detection - transfer learning from CARLA skip 10 to Waymo skip 10
Source: The author (2020)

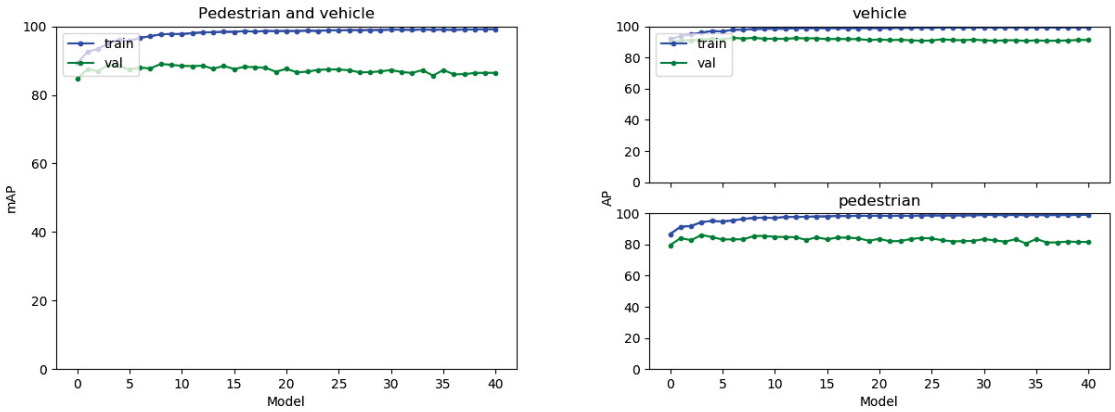
Another approach is by mixing synthetic with real data and training in a single stage instead of sequentially. This method was performed for both datasets with skip 10, and by comparing this approach against training sequentially, the mixed method did not improve algorithm performance, reducing the mAP from 32.02 to 13.44. The training progress is shown in Figure 5.17.



(a) (b)

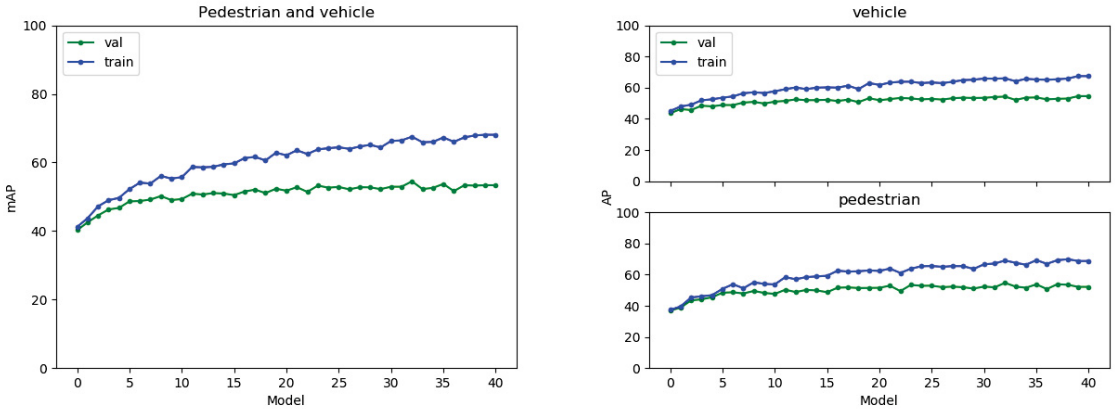
Figura 5.17: Object detection - training on mixed CARLA skip 10 and Waymo skip 10, and evaluating against Waymo skip 10 data
Source: The author (2020)

Besides evaluating only from synthetic to real-world, it is common to train algorithms with starting weights based on data from real-world large-scale datasets such as COCO. By performing this method, as expected in both cases their performances are greatly improved - in the synthetic part the mAP increases from 70.00 to 84.64, and on the real-world data from 36.36 to 57.30. The training progresses are shown in Figures 5.18 and 5.19 for CARLA and Waymo, respectively.



(a) (b)

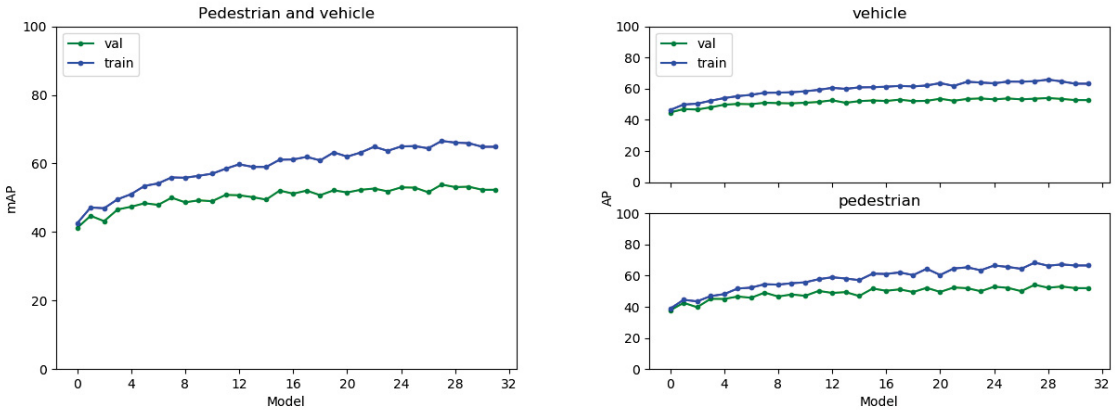
Figura 5.18: Object detection - training on CARLA skip 10 with pretrained weights from COCO
Source: The author (2020)



(a) (b)

Figura 5.19: Object detection - training on Waymo skip 10 with pretrained weights from COCO
Source: The author (2020)

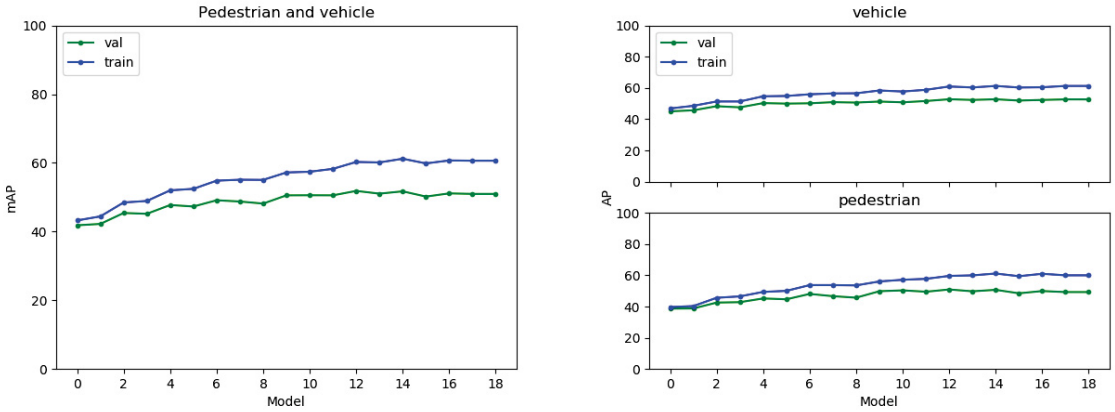
Another test is performed by training two times: first fine-tuning on CARLA from COCO and then fine-tuning on Waymo, both with skip 10. By comparing this method with using only COCO as a starting point, performance deterioration is noticed when adding synthetic data, with the mAP decreasing from 57.30 to 55.50. The training progress is shown in Figure 5.20.



(a) (b)

Figura 5.20: Object detection - training on Waymo skip 10 with pretrained weights from CARLA and COCO
Source: The author (2020)

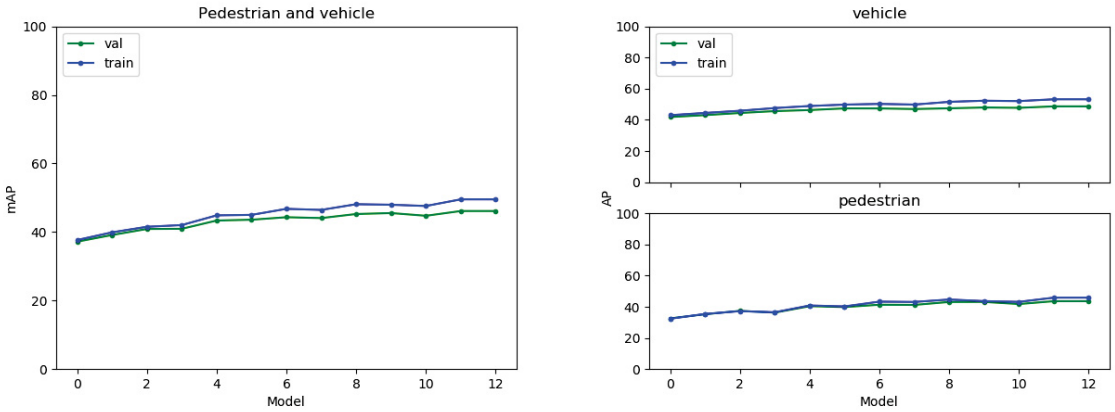
One thought for the reason of subpar performance when transferring synthetic data could be attributed to the illumination modeling of the synthetic dataset. Therefore, these frames were removed from the dataset and the model was first fine-tuned from COCO on this nightless version of CARLA skip 10 and later fine-tuned on Waymo skip 10. This hypothesis was proven to be false, with the performance degrading mAP from 55.50 to 53.63. On a positive note, this result helps foment the idea that further diversifying the synthetic data could result in improvements. The training progress is shown in Figure 5.21.



(a) (b)

Figura 5.21: Object detection - first fine-tuning on CARLA skip 10 without night frames with pretrained weights from COCO and then fine-tuning on Waymo skip 10
Source: The author (2020)

Finally, a combination of the previous methods is performed by fine-tuning from COCO on a mixed CARLA skip 10 with WAYMO skip 10 datasets. Compared to training on the mixed data from scratch, this method improves mAP from 13.44 to 47.89 however still does not beat using COCO as a starting point for directly training into Waymo skip 10. The training progress is shown in Figure 5.22.



(a) (b)

Figura 5.22: Object detection - training on mixed CARLA skip 10 and Waymo skip 10, with pretrained weights from COCO
Source: The author (2020)

Changing focus from dataset manipulation to algorithm parametrization, several single modifications were performed to verify if any could perform better than the established baseline. First, the batch size was increased from 2 to 6; the idea is that with more images being passed at once before computing the losses, more stable learning progress could be achieved. Next, the amount of ROI heads is increased from 128 to 512, which is an extension of the batch size with a higher focus on sampling labels from each image of the batch. The learning rate was also decreased from $2.5 * 10^{-4}$ to $1 * 10^{-5}$ to verify if the network was skipping over local minima. Lastly, a bigger ResNet backbone is tested by changing ResNet 50 to ResNet 101 to check the impact of applying a deeper and in theory more generalizable network. Surprisingly, none of the modifications presented an improvement over the original baseline, which suggests that for advancing algorithm performance, the focus should be on the analysis and treatment of the data itself. These results are shown in Table 5.1. One issue when fine-tuning algorithm hyperparameters is that the process itself is not entirely predictable, and thus automated approaches rely on grid search, random search, or evolutionary optimizers. This, however, comes with an expensive computational price since in any of these methods multiple iterations are necessary to reach a decent combination. Couple this with deep learning, where a single model devotes up to one week of intermittent computing, then it quickly becomes unfeasible for most hardware. This is the reason only a few educated hyperparameter guesses were performed.

Tabela 5.1: Algorithms fine-tuned from COCO and evaluated on Waymo skip 10
Source: The author (2020)

Modification	mAP	$AP_{vehicle}$	$AP_{pedestrian}$
baseline	**57.3**	**55.25**	**59.34**
batch=6	52.46	52.61	52.32
ROI heads=512	51.34	52.17	50.50
learning rate=$1 * 10^{-5}$	40.58	43.24	37.91
ResNet 101	51.80	51.37	52.23

To summarize the results of the object detection section, a general overview of the performance of the trained object detectors for both the synthetic and real-world data are shown in Table 5.2. Some key findings of the section are:

1. Training on temporal data requires special care to avoid data leakage, with subsampling being an alternative;

2. Additional scene complexity on the synthetic part is indeed beneficial for the real-world part, even when that specific weather condition is not present on the real-world data;

3. The complexity of the problem is not entirely tied to model size or hyperparametrization;

4. When evaluating metric performance on Waymo skip 10, directly fine-tuning from COCO yields the best results;

5. The created CARLA synthetic dataset did not improve algorithm performance for Waymo in any case.

For the first point, the effect of data leakage is assumed due to the high similarity between the training and validation curve profiles, whose cause is assigned due to the low temporality and thus high similarity between frames. To work around this issue, the data is iteratively subsampled until the profiles are qualitatively more dissimilar. However, a collateral side-effect is noticed: significant decreases in performance are noted as the dataset is reduced. For the synthetic part, mAP decreases from 82.43 to 70.00, vehicles AP from 81.91 to 72.97, and pedestrians AP from 82.95 to 67.03. For the real-world part, mAP declines from 41.37 to 36.36, vehicles AP from 48.07 to 41.43, and pedestrians AP from 34.68 to 31.29. Notably, on the synthetic part, the pedestrians decrease more than vehicles with 15.92 while on the real-world the impairment is more balanced. The reasons for that were not deeply explored in this work but could be the consequences of aggravating the class balance and frame numbers of the synthetic data.

For the second point, the scene complexity effect is noticed by comparing the metrics between the test cases "COCO $\rightarrow$ CARLA skip 10 no nights $\rightarrow$ Waymo skip 10"and "COCO $\rightarrow$ CARLA skip 10 $\rightarrow$ Waymo skip 10". When the synthetic nightly frames are removed from the data pipeline, then the final model degrades performance on Waymo skip 10, which corroborates this assumption. The mAP decreases from 55.50 to 53.63 and the AP from 54.37 to 53.13 for vehicles and 56.62 to 54.14 for pedestrians.

On the fourth point, surprisingly the method of simply fine-tuning directly from pre-trained weights on COCO was the one which yielded the best results, with 57.30 mAP, 55.25, and 59.34 AP for vehicles and pedestrians. Regarding the last point, the synthetic data could still be further improved by diversifying objects in the scenes as well as their positions. Possible issues could be low texture quality, animation fidelity, and lack of synthetic augmentation, which regards random displacements of objects of interest, noise addition, and differences in synthetic to real-world camera sensor modeling. Something to be noted is that on Waymo's dataset, many vehicles are stationary on the sides of the road, while the same is not true for CARLA where all vehicles are in motion on the road. Other approaches that could improve knowledge transfer from the synthetic to the real-world part are based on the domain adaptation concept. On domain adaptation, the main concern is on closing the gap between two data domains by measuring feature distribution from the target and source datasets and implanting it into the model. This way, less discrepancy is observed when fine-tuning on the target dataset, which aids in improving the learning progress and performance. Due to time restraints, however, this method was not performed on this dissertation work but could be a focal point for future works.

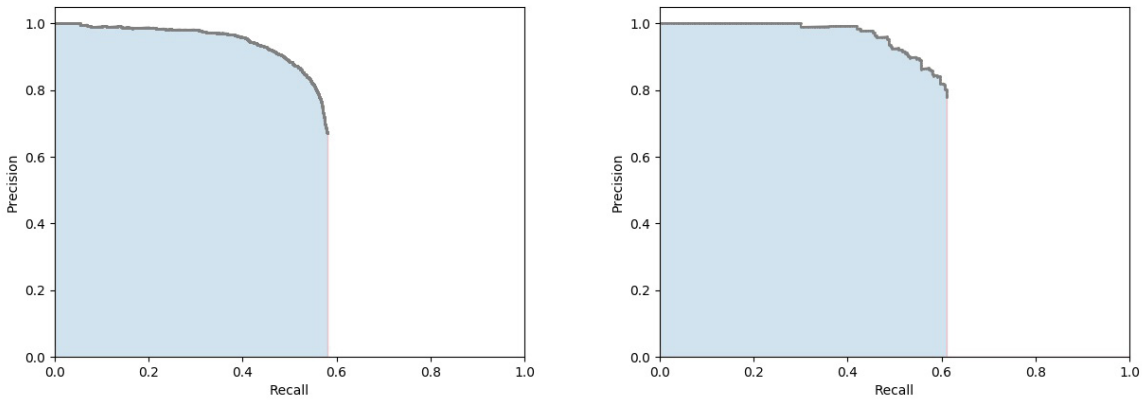
To close this section, a closer look at the validation and testing results from the method in bold from Table 5.2 are presented. In Figure 5.23 the AP charts for vehicles and pedestrians are shown. Notably, many instances were not detected, which decreases significantly its recall

and therefore mAP. One of the reasons for that is the indistinction between far and close objects annotation when computing the metrics, with the former being of harder detection. In Figures 5.24 to 5.27 inference samples of the model are shown, with the ground-truth boxes painted as red and the predictions as green.

Tabela 5.2: Overall detection performance according to different data sources and targets
Source: The author (2020)

Source dataset	**Target dataset**	*mAP*	$AP_{vehicle}$	$AP_{pedestrian}$
CARLA holdout by town	-	52.10	64.15	40.05
CARLA shuffled holdout	-	82.43	81.91	82.95
CARLA shuffled k-fold, k=3	-	83.32	81.92	84.73
CARLA skip 2	-	81.43	81.10	81.76
CARLA skip 5	-	78.05	79.91	76.19
CARLA skip 10	-	70.00	72.97	67.03
Waymo native split	-	12.81	19.43	6.192
Waymo shuffled holdout	-	41.37	48.07	34.68
Waymo skip 2	-	42.01	46.64	37.37
Waymo skip 5	-	38.32	42.14	34.50
Waymo skip 10	-	36.36	41.43	31.29
Waymo skip 10, balanced with oversample	Waymo skip 10	9.94	12.91	6.97
CARLA skip 10 →Waymo skip 10	Waymo skip 10	32.02	35.47	28.56
CARLA skip 10 mixed with Waymo skip 10	Waymo skip 10	13.44	20.57	6.31
COCO →CARLA skip 10	CARLA skip 10	84.64	83.86	85.42
COCO →Waymo skip 10	**Waymo skip 10**	**57.30**	**55.25**	**59.34**
COCO →CARLA skip 10 →Waymo skip 10	Waymo skip 10	55.50	54.37	56.62
COCO →CARLA skip 10 no nights →Waymo skip 10	Waymo skip 10	53.63	53.13	54.14
COCO →Waymo skip 10 →CARLA skip 10	Waymo skip 10	36.12	32.34	39.90
COCO →CARLA skip 10 mixed with Waymo skip 10	Waymo skip 10	47.89	49.10	46.67

The symbol "→"implies that trained weights on the dataset to the left of the arrow were used as starting point for the dataset to the right of the arrow. The symbol -"indicates that the target dataset is the same as the source.



(a) Vehicle *AP* (b) Pedestrian *AP*

Figura 5.23: Vehicle (left) and pedestrian (right) AP on the test split from the model fine-tuned on Waymo skip 10 from COCO
Source: The author (2020)

Figura 5.24: Inference sample 1 from the model fine-tuned on Waymo skip 10 from COCO
Source: The author (2020)

Figura 5.25: Inference sample 2 from the model fine-tuned on Waymo skip 10 from COCO
Source: The author (2020)

Figura 5.26: Inference sample 3 from the model fine-tuned on Waymo skip 10 from COCO
Source: The author (2020)

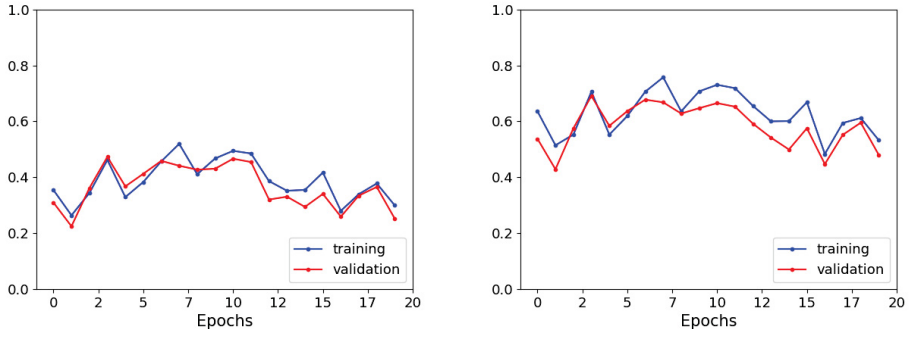
Figura 5.27: Inference sample 4 from the model fine-tuned on Waymo skip 10 from COCO
Source: The author (2020)

5.2 MONOCULAR DEPTH ESTIMATION

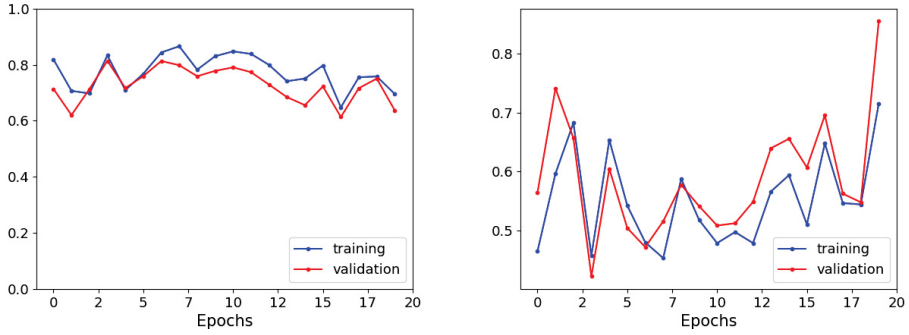
The monodepth2 architecture, shown in section 4.2, has been evaluated for the following tests. Contrary to the workflow of skipping consecutive frames from the object detection part, on monodepth2 a shorter period between frames is desired since the method relies on frame sequentiality. Regarding hyperparameters, image resolution and batch size had to be reduced to 1024x320 and 1 so that GPU RAM resources could fit since at training time three frames are stored in memory for each step which makes it resource-intensive. It should be noted that since the method is self-supervised, then the error metrics shown In this section are only a rough estimation on the learning progress and do not represent the actual loss of the network.

5.2.1 Synthetic-based CARLA

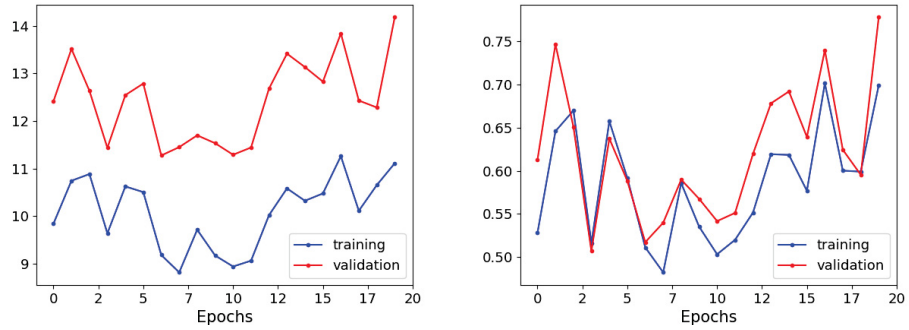
As a first test, the holdout method is applied considering completely different sequences of videos for training and validation stages. The training progress is shown in Figure 5.28, with each point representing an epoch.



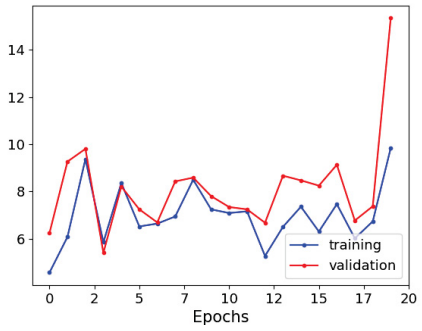
(a) δ_1 (b) δ_2



(c) δ_3 (d) *Abs rel*



(e) *RMSE* (f) *log RMSE*



(g) *Sq rel*

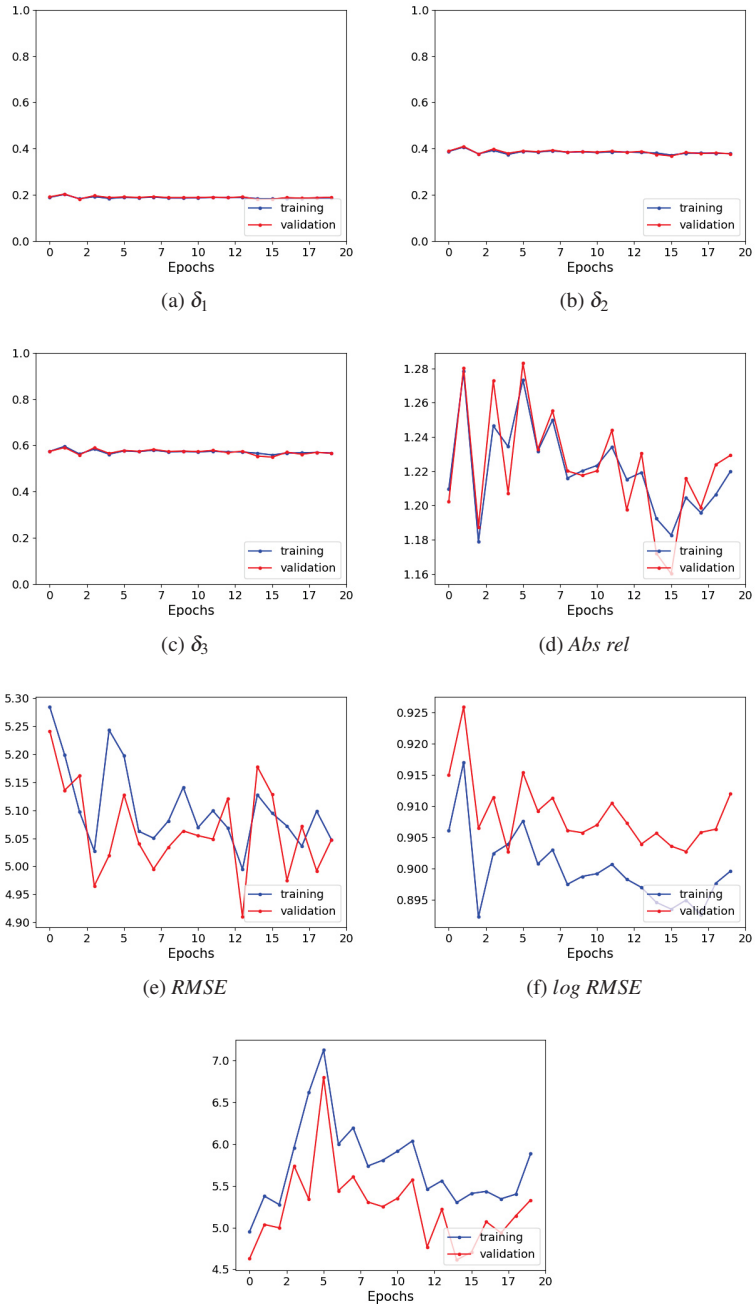
Figura 5.28: Monocular depth estimation - training on CARLA from scratch
Source: The author (2020)

After observing the results, one hypothesis for which could improve algorithm performance is the removal of night frames in the dataset, since the pixel differences on a frame-by-frame basis might be too abrupt for its loss calculation, thus prejudicing training. Therefore, the

model was trained without frames at night and evaluated on the whole dataset to confirm this hypothesis. During training, however, the model had problems with convergence, contradicting this theory.

5.2.2 Waymo Open

As in its synthetic counterpart, the holdout method is applied to different sequences of videos for training and validation. The training progress is shown in Figure 5.29.



(a) δ_1 (b) δ_2

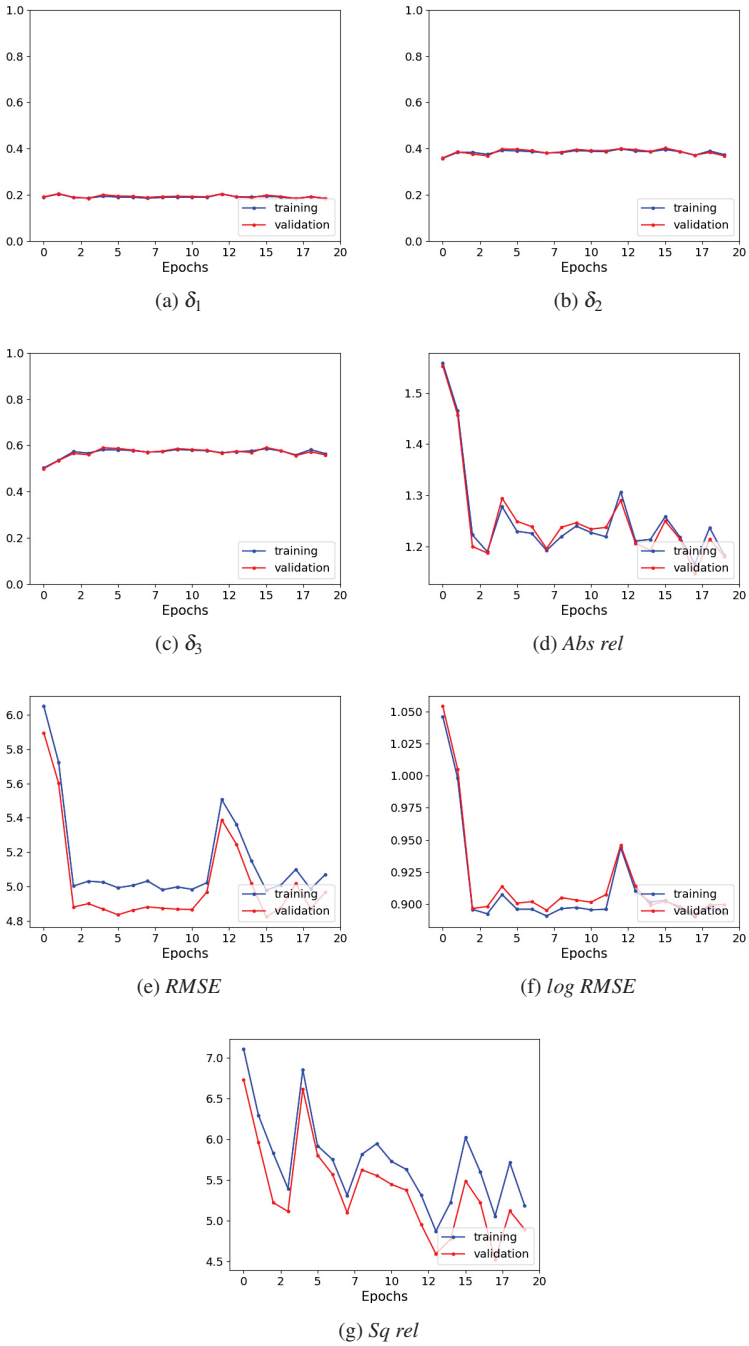
(c) δ_3 (d) *Abs rel*

(e) *RMSE* (f) *log RMSE*

(g) *Sq rel*

Figura 5.29: Monocular depth estimation - training on Waymo from scratch
Source: The author (2020)

Again, the nocturne frames hypothesis is tested for real-world data. Compared to training directly from scratch, significant deterioration is noticed on the model's performance by around 7%, which suggests that diverse illumination conditions, even if complex, aid on the model's generalization. The training progress is shown in Figure 5.30.



(a) δ_1 (b) δ_2

(c) δ_3 (d) *Abs rel*

(e) *RMSE* (f) *log RMSE*

(g) *Sq rel*

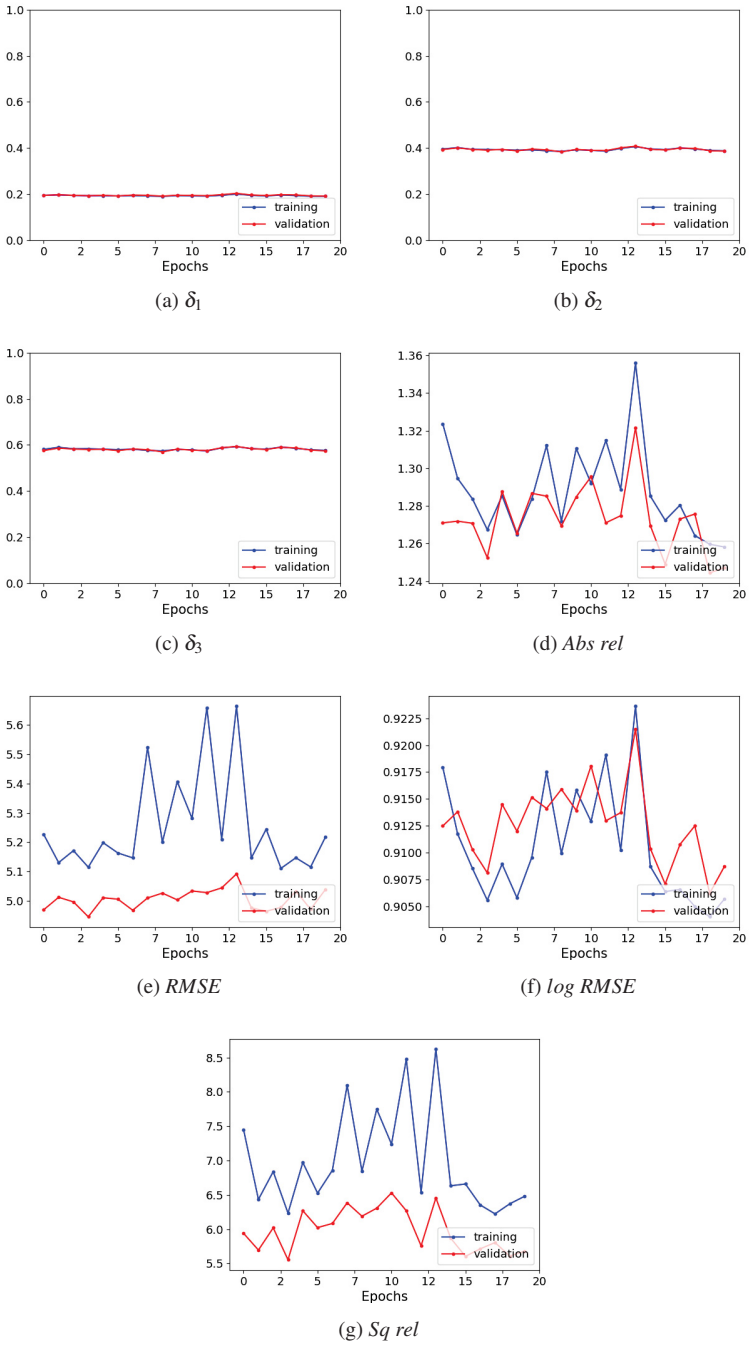
Figura 5.30: Monocular depth estimation - training on waymo from scratch without nocturne frames
Source: The author (2020)

5.2.3 Transfer learning

As with object detection, similar procedures were taken for the transfer learning subsection. First, weights from the synthetic data were used as a starting point for the real-world data,

followed by mixing them both and finally by repeating the previous steps while incorporating pretrained weights from a real-world established dataset as a starting point.

The training progress for fine-tuning on Waymo directly from CARLA is shown in Figure 5.31. Compared to training from scratch, the model can converge and generalize slightly better, with the metrics improving by around 10%.



(a) δ_1 (b) δ_2

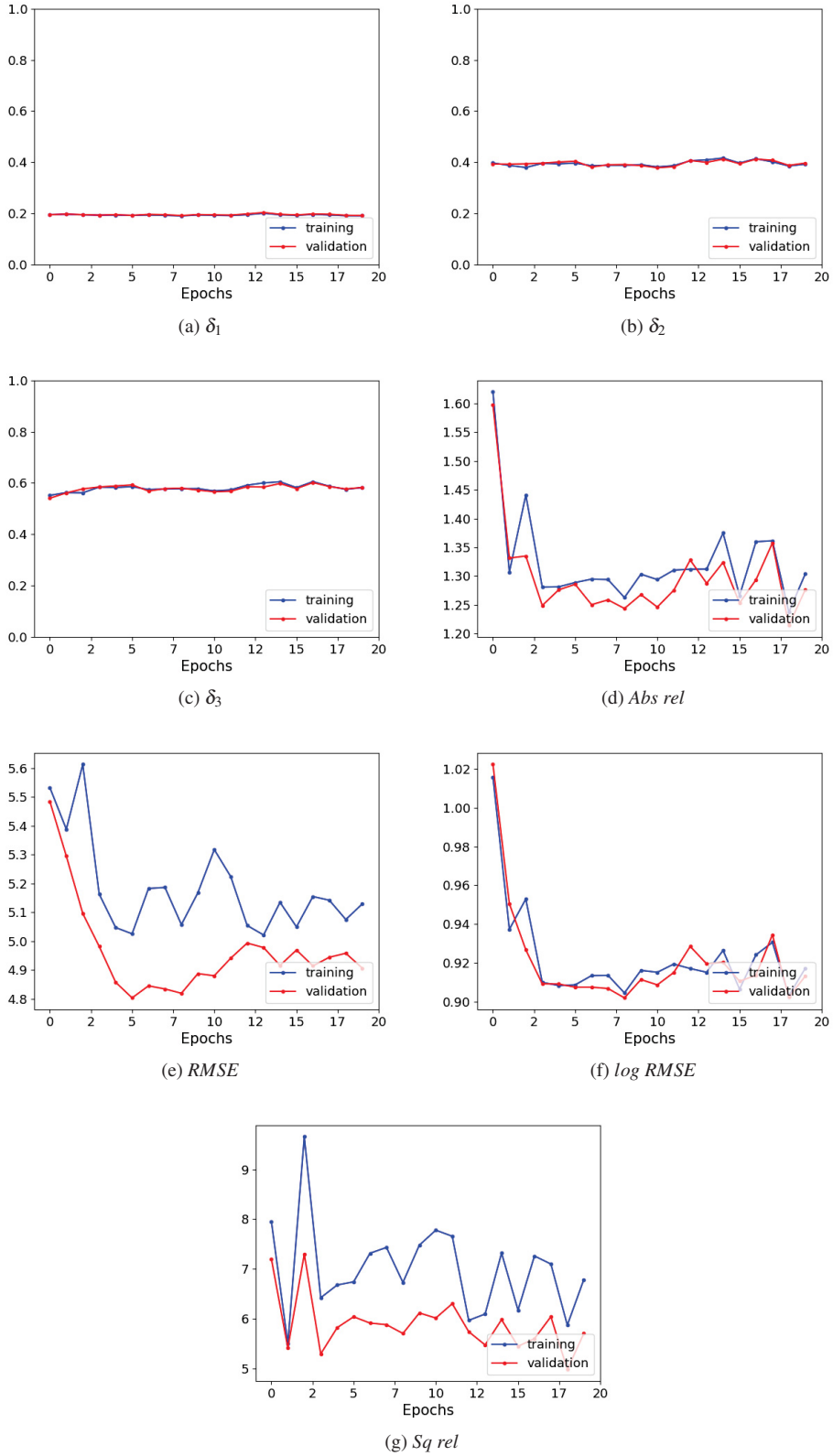
(c) δ_3 (d) *Abs rel*

(e) *RMSE* (f) *log RMSE*

(g) *Sq rel*

Figura 5.31: Monocular depth estimation - training on Waymo from CARLA
Source: The author (2020)

Next, the model is trained directly from a mixed perspective, containing both synthetic and real data. The training progress is shown in Figure 5.32. Compared to training sequentially from CARLA to Waymo, improvement is noted only on δ_1.



(a) δ_1 (b) δ_2

(c) δ_3 (d) *Abs rel*

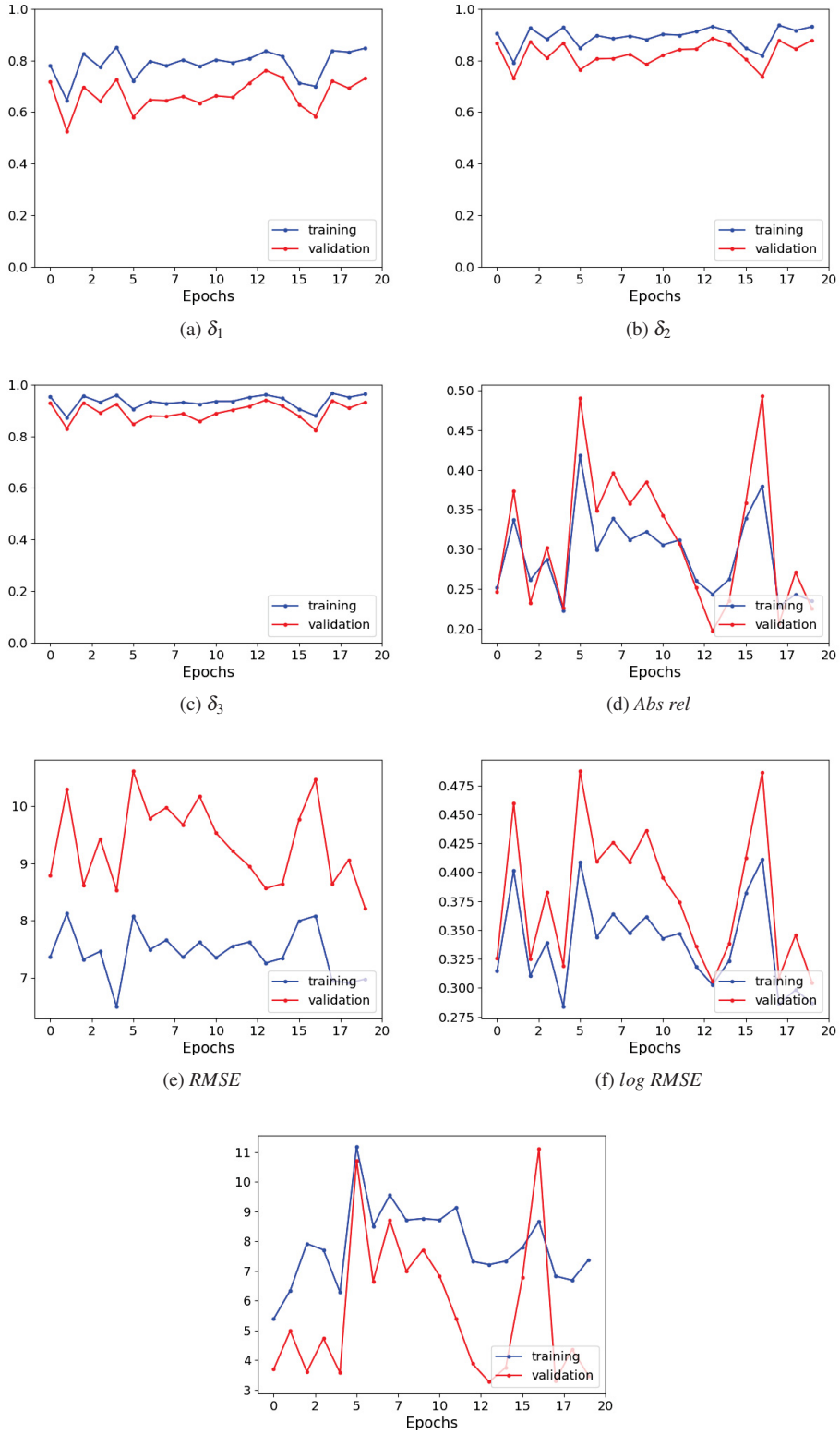
(e) *RMSE* (f) *log RMSE*

(g) *Sq rel*

Figura 5.32: Monocular depth estimation - training on mixed data from scratch
Source: The author (2020)

Next, the model is fine-tuned on CARLA from pretrained weights on KITTI. When compared with the approach trained from scratch, significant improvements are observed, with

the Abs rel error decreasing by half and with double δ accuracy. The training progress is shown in Figure 5.33. Qualitatively analyzing both charts, no smoother learning profile is observed.



(a) δ_1 (b) δ_2

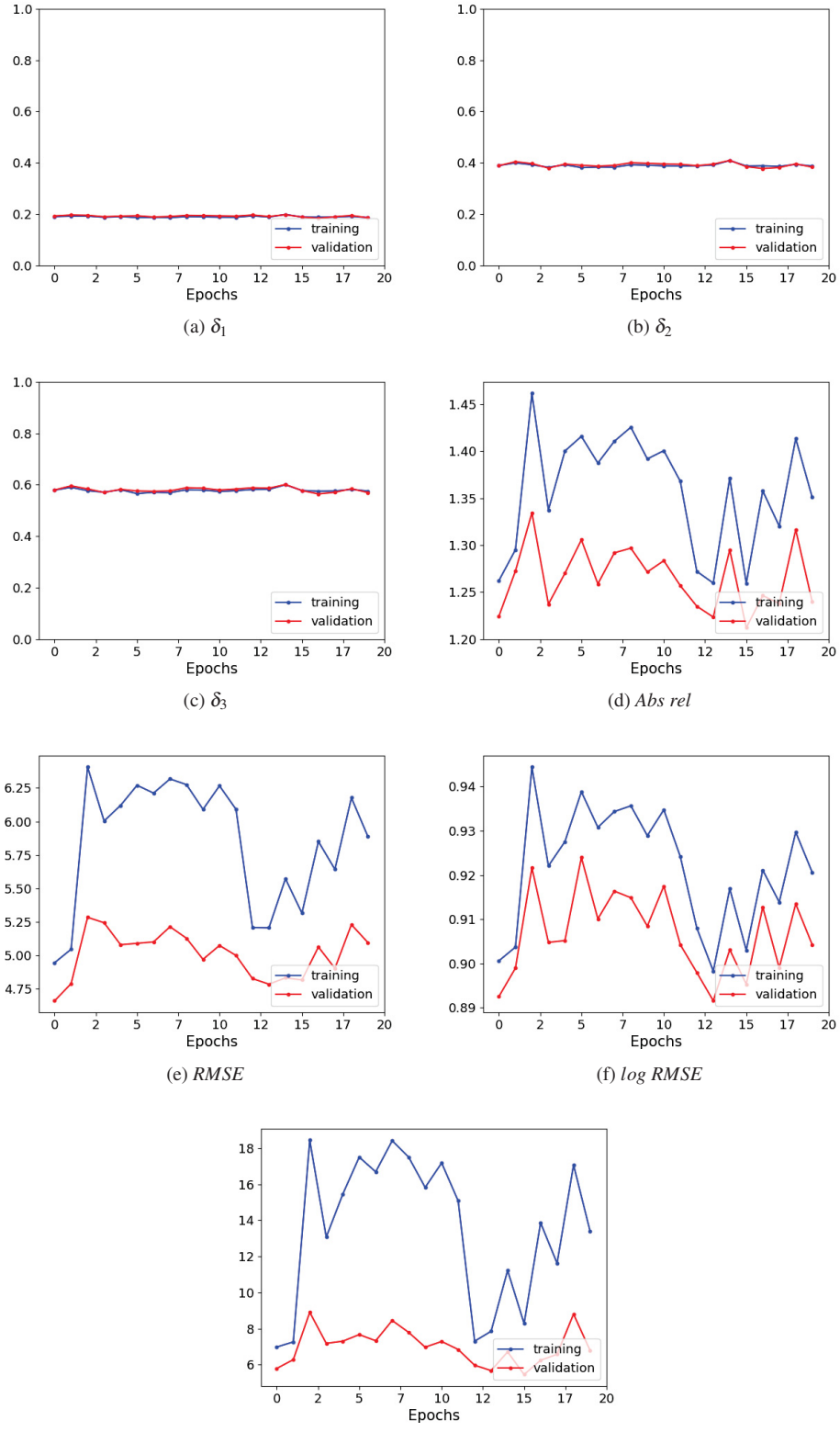
(c) δ_3 (d) *Abs rel*

(e) *RMSE* (f) *log RMSE*

(g) *Sq rel*

Figura 5.33: Monocular depth estimation - fine-tuning on CARLA from KITTI
Source: The author (2020)

Following it, the model is fine-tuned on Waymo from pretrained weights on KITTI. The training progress is shown in Figure 5.34. Compared to training from scratch, the model is surprisingly prejudiced, with the metrics deteriorating by around 10%.



(a) δ_1 (b) δ_2

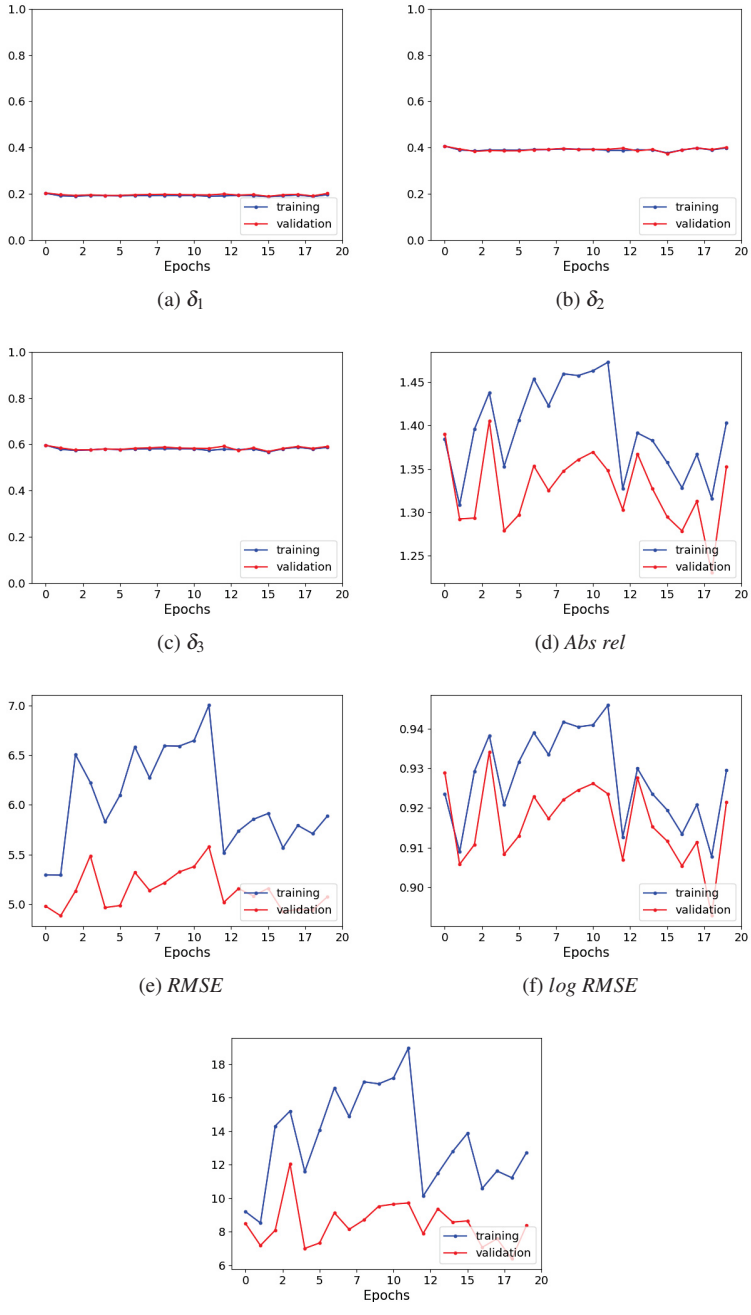
(c) δ_3 (d) *Abs rel*

(e) *RMSE* (f) *log RMSE*

(g) *Sq rel*

Figura 5.34: Monocular depth estimation - fine-tuning on Waymo from KITTI
Source: The author (2020)

Finally, the model is fine-tuned on the mixed CARLA and Waymo dataset from pre-trained weights on KITTI. The training progress is shown in Figure 5.35. Compared to training from scratch on the mixed data, a significant deterioration is observed for *Sq rel*, increasing from 7.199 to 12.046. On the other metrics, around 5% of improvement is noted.



(a) δ_1 (b) δ_2

(c) δ_3 (d) *Abs rel*

(e) *RMSE* (f) *log RMSE*

(g) *Sq rel*

Figura 5.35: Monocular depth estimation - fine-tuning on mixed data from KITTI
Source: The author (2020)

To summarize the results of the monocular depth estimation section, a general overview of the performance of the models for both synthetic and real-world data are shown in Table 5.3. Some key findings are:

1. Scene complexity is beneficial for model generalization;

2. No single model outperformed all others;

3. Addition of synthetic data did incur better performance.

On the first point, this is evaluated by excluding nightly frames from the dataset and then evaluating on the, with its effects observed on the models "CARLA no nights"and "Waymo no nights". In both cases, performance is generally worsened, with up to 19% and 16% more errors for the first and second models. Accuracy, Likewise, suffers the same effect with the degradation of up to 93% and 15% on the first and second models. This suggests that for the tests performed, the impact of less scene diversity has a higher impact on the synthetic than on real-world data.

Regarding the second point, this is noted by observing the 3 models listed in bold on the table. The model "CARLA $\rightarrow$ Waymo"presented best results on *Sq rel*, *RMSE*, δ_2, and δ_3 with 6.455, 5.092, 0.408, and 0.593; the model "KITTI $\rightarrow$ CARLA $\rightarrow$ Waymo"on *Abs rel*, *RMSE log*, and δ_3 with 1.277, 0.903, and 0.593; lastly, the model "CARLA mixed with Waymo"outperformed the others on δ_1 with 0.211. With such diverging results, the purpose of a variety of evaluation metrics is clearer. On the accuracy metrics δ, for example, both the first and second cited models present close values, diverging by less than 4% in the worst case. On the other hand, for the other error metrics, the divergence is more pronounced, with up to 16% difference.

The statement of the last point is surprising since the opposite effect of that observed on the object detection section is observed. On monocular depth estimation, simply adding more data resulted in improvements on model performance, whereas on the other task either domain adaptation or domain randomization techniques should be applied to promote performance. The monocular depth estimation improvement is verified mainly by two comparisons: between the models "Waymo"and "CARLA $\rightarrow$ Waymo", and between "KITTI $\rightarrow$ Waymo"and "CARLA $\rightarrow$ Waymo". In both cases, improvements are noted with an overall improvement of 2.13 ± 3.04 % on the first case and 5.75 ± 8.97 % on the second. Besides the higher performance boost on fine-tuning from CARLA, another surprising result is that the model trained from scratch and the model fine-tuned from KITTI present opposite behavior on the metrics. While the former performs better in the error metrics, the latter presents higher accuracy. Another hypothesis was formed on the method of how the data is presented to the model, being either sequentially or in a mixed matter. Compared to training from scratch both methods seem adequate, however when comparing the sequential with the mixed, then the sequential method still outperforms in most cases, with 6.00 ± 9.22 % better metrics.

Lastly, inference results of the model "CARLA $\rightarrow$ Waymo"are shown in Figures 5.36 to 5.39. By qualitatively analyzing the inference results on the images, it is explicit the effect of illumination in Figure 5.37 on model performance, with most depth contours being blended into the environment. Besides that, on areas where rain droplets are struck to the camera, as shown in Figures 5.38 and 5.39, the model also performs poorly, albeit not as much than with illumination effects.

As additional results, inference results which combine the depth from monodepth2 and detections from Faster R-CNN are shown in Figures 5.40 to 5.42, where the green boxes indicate the detections and the number above the distance of it in meters to the camera. To measure the distance, the average of the depth points in the bounding box region is calculated.

Tabela 5.3: Overall metrics performance for monodepth2 on CARLA and WAYMO
Source: The author (2020)

Source	**Target**	*Abs rel*	*Sq rel*	*RMSE*	*RMSE log*	δ_1	δ_2	δ_3
CARLA	-	0.855	15.351	14.179	0.778	0.252	0.480	0.637
Waymo	-	1.283	6.800	5.128	0.915	0.192	0.390	0.578
CARLA no nights	CARLA	0.896	12.022	17.547	0.959	0.130	0.266	0.436
Waymo no nights	Waymo	1.553	6.728	5.895	1.054	0.192	0.360	0.499
CARLA →Waymo	Waymo	1.322	**6.455**	**5.092**	0.921	0.204	**0.408**	**0.593**
KITTI →CARLA	CARLA	0.493	11.127	10.459	0.486	0.583	0.737	0.824
KITTI →Waymo	Waymo	1.334	8.902	5.283	0.922	0.196	0.397	0.584
KITTI →CARLA →Waymo	Waymo	**1.277**	7.694	5.153	**0.903**	0.197	0.404	**0.593**
CARLA mixed with Waymo	Waymo	1.597	7.199	5.484	1.022	**0.211**	0.393	0.541
KITTI →CARLA mixed with Waymo	Waymo	1.405	12.046	5.486	0.934	0.196	0.388	0.576

The symbol "→"implies that trained weights on the dataset to the left of the arrow were used as starting point for the dataset to the right of the arrow. The symbol -"indicates that the target dataset is the same as the source.

Figura 5.36: Monocular depth estimation - sample 1, standard view
Source: The author (2020)

Figura 5.37: Monocular depth estimation - sample 2, sunlight on camera
Source: The author (2020)

Figura 5.38: Monocular depth estimation - sample 3, rain droplets on camera
Source: The author (2020)

Figura 5.39: Monocular depth estimation - sample 4; rain droplets on camera
Source: The author (2020)

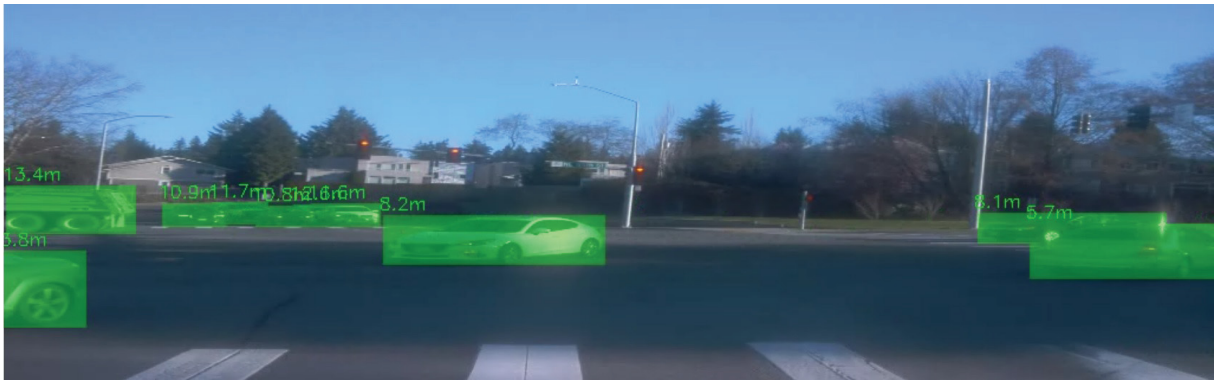


Figura 5.40: Monocular depth estimation and object detection - sample 1
Source: The author (2020)

Figura 5.41: Monocular depth estimation and object detection - sample 2
Source: The author (2020)

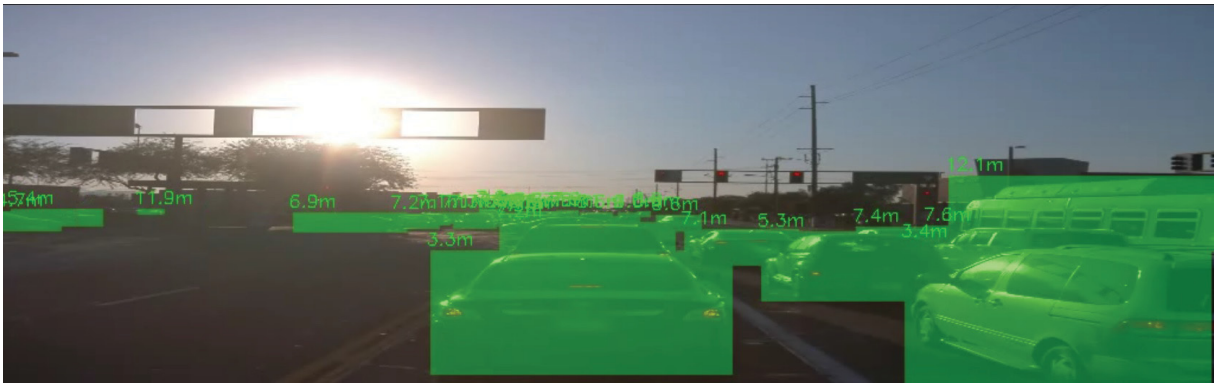


Figura 5.42: Monocular depth estimation and object detection - sample 3
Source: The author (2020)

6 CONCLUSIONS AND FUTURE WORKS

In this section, the main findings of this dissertation work along with suggestions for future works are presented.

6.1 CONCLUSIONS

As a short recap, on this dissertation work, a review on the landscape for topics concerning autonomous vehicles' perception with computer vision was performed, which in turn paved the way for selecting the most appropriate algorithms and datasets. On this vein, a data collection tool for CARLA was developed and made open-source for the synthetic part and Waymo Open dataset for the real-world part, with Faster R-CNN and monodepth2 being evaluated with varied transfer learning methods. The general and specific objectives of this dissertation work were fulfilled, with both synthetic and real-world data being assembled and object detection and monocular depth estimation models evaluated on them.

In addition to that, a review on datasets, SOA of object detection, and monocular depth estimation models were brought together, enabling an overview of the applied computer vision autonomous driving landscape. About synthetic datasets, there is still significant ground to cover, with a common issue on open-source simulators being poor or lacking pedestrians' animations and textures. On real-world datasets, a prevalent issue, perhaps not on the data itself, but on the algorithms which use them, is that there is a significant context difference between them. This includes varied camera models and positions, and recorded location, all of which affect generalization on SOA general-purpose object detectors.

For the real-world dataset, the Waymo Open dataset version 1.0.0 was collected and parsed for its frontal view, with RGB frames, 2D bounding box annotations, and LIDAR-based depth data. On the synthetic part, an open-source custom CARLA-based automotive dataset and data gathering tool were created based on version 0.9.6 of the CARLA simulator, with sensor suite and annotations equivalent to the frontal view of Waymo Open dataset version 1.0.0.

For object detection, the Faster R-CNN with a ResNet 50 and FPN as backbone was evaluated. The key findings on it include the importance of treating temporal data on atemporal models, the benefit of promoting scenes variety on the synthetic data, the complexity of the problem not being only tied to model parameterization, and the most adequate model being the one fine-tuned directly from another real-world dataset. On the first point, where temporal data is managed with an atemporal model, the persistent issue was on data leakage since several frames presented too similar data, thus biasing the model to be highly specific. To diminish this, subsampling the whole dataset at one-tenth of its frames present the most realistic results. On the second point, the absence of nightly frames on the synthetic training data incurred in worse performance on the real-world when compared to using the whole synthetic data, which confirms this assumption. On the third point, several hyperparameters modifications were performed on the model, however with none of them improving generalization. Even when switching the backbone to a deeper one, ResNet 101, no improvement was noticed which suggests that either more data or a different model approach is necessary for progress in this field. On the final point, the best performing model on Waymo skip 10 achieved on the validation set 57.30 mAP, 55.25, and 59.34 AP for vehicles and pedestrians. Although the synthetic data did not improve model performance, it aids in cementing the idea that its use for real-world applications is not trivial

with simple transfer learning techniques, but rather requires further research towards domain shift with techniques such as domain adaptation and randomization.

For monocular depth estimation, the monodepth2 model with a U-Net and ResNet 18 as backbone was evaluated. There are three key findings: the importance of scenes complexity, no single best model, and the benefit of directly adding synthetic data to the model pipeline. On the first point, this hypothesis is evaluated analogously to the object detection: by removing nightly frames from the datasets, evaluation on the whole data was prejudiced, which sustains this affirmation. On the second point, three models, all of which involve the synthetic data, presented the best results in separate metrics. This indicates that the most adequate model would depend on the context it is applied, with the ones with less error metric being more appropriate when precision is the main concern or on more accuracy when it is more relevant. Lastly, unlike on the object part, significant improvements are noticed when synthetic data is injected into the model, with the best combinations depending on the metric analyzed. For *Sq rel*, *RMSE*, δ_2, and δ_3, the model with fine-tuning directly from CARLA presented the best results, with 6.455, 50.92, 0.408 and 0.593, respectively. For *Abs rel*, *RMSE log*, and δ_3, the model which fine-tunes on CARLA from KITTI and then on Waymo presents the best results, with 1.277, 9.03, 0.593, respectively. Lastly, for δ_1 the model trained on mixed CARLA and Waymo data outperformed the others with 0.211.

It is observed that the requirements for a good dataset differ depending on which task is foreseen. While on object detection quality and variety of the annotated objects and environment pose importance, for monocular depth estimation more temporally consistent and regular scenes are desired. Still, on monocular depth estimation, direct addition of synthetic data with varied asset types and colors indeed benefit model generalization, an insight also shared by other works of the field.

6.2 FUTURE WORKS

To further explore this subject matter, the following topics are suggestions for future works:

- Domain adaptation with generative adversarial networks;
- Domain randomization for further data variety;
- Extension on the objects suite of CARLA with focus on pedestrians;
- Procedural generation of synthetic worlds;
- Evaluation of other network models, such as capsule networks and graph networks.

REFERÊNCIAS

[1] F Kuhnert, C Stürmer, and A Koste. Five trends transforming the automotive industry. *PricewaterhouseCoopers: Frankfurt, Germany*, 2018.

[2] Alice Matthews. Active and passive automotive safety systems. `https://automotive.electronicspecifier.com/safety/active-and-passive-automotive-safety-systems`, 2017. Accessed in 04/06/2019.

[3] Amir Mukhtar, Likun Xia, and Tong Boon Tang. Vehicle detection techniques for collision avoidance systems: A review. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2318–2338, 2015.

[4] Aldona Jarašūniene and Gražvydas Jakubauskas. Improvement of road safety using passive and active intelligent vehicle safety systems. *Transport*, 22(4):284–289, 2007.

[5] Lex Fridman, Daniel E. Brown, Michael Glazer, William Angell, Spencer Dodd, Benedikt Jenik, Jack Terwilliger, Aleksandr Patsekin, Julia Kindelsberger, Li Ding, Sean Seaman, Alea Mehler, Andrew Sipperley, Anthony Pettinato, Bobbie Seppelt, Linda Angell, Bruce Mehler, and Bryan Reimer. MIT autonomous vehicle technology study: Large-scale deep learning based analysis of driver behavior and interaction with automation. *CoRR*, abs/1711.06976, 2019.

[6] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[7] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168, Baden-baden, Germany, June 2011.

[8] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. Zurich, Switzerland.

[9] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[10] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37, Amsterdam, Netherlands, 2016. Springer.

[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, Montreal, Canada, 2015.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, Las Vegas, USA, June 2016.

[13] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, USA, June 2015.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, Boston, USA, 2015.

[15] Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, Veneza, Italy, 2017.

[17] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–279, Honolulu, USA, 2017.

[18] Yevhen Kuznietsov, Jorg Stuckler, and Bastian Leibe. Semi-supervised deep learning for monocular depth map prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, USA, July 2017.

[19] European Commission. 2019 road safety statistics: what is behind the figures? `https://ec.europa.eu/commission/presscorner/detail/en/qanda_20_1004`. (Accessed on 08/16/2020).

[20] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[21] Anil K Jain. *Fundamentals of digital image processing*. Englewood Cliffs, NJ: Prentice Hall,, 1989.

[22] Khawaja Tehseen Ahmed, Aun Irtaza, and Muhammad Amjad Iqbal. Fusion of local and global features for effective image extraction. *Applied Intelligence*, 47(2):526–543, 2017.

[23] Dong-Chen He and Li Wang. Texture unit, texture spectrum, and texture analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 28(4):509–512, 1990.

[24] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

[25] David G Lowe et al. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision*, volume 99, pages 1150–1157, Corfu, Greece, 1999.

[26] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision*, pages 404–417, Graz, Austria, 2006.

[27] Robert K McConnell. Method of and apparatus for pattern recognition, January 28 1986. US Patent 4,567,610.

[28] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893, San Diego, USA, 2005.

[29] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[30] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[31] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[32] Michael Nielsen. Neural networks and deep learning. `http://neuralnetworksanddeeplearning.com/chap1.html`, December 2019. Accessed on 12/03/2020.

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

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

[35] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[36] John McGonagle, George Shaikouski, Christopher Williams, Andrew HSU, and Jimin Khim. Backpropagation | brilliant math & science wiki. `https://brilliant.org/wiki/backpropagation/`. Accessed on 24/03/2020.

[37] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[38] Eric W Weisstein. Convolution. *https://mathworld. wolfram. com/*, 2003.

[39] dshahid380. Convolutional neural network - towards data science. `https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529`, February 2019. Accessed on 04/06/2020.

[40] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`. Accessed on 30/03/2020.

[41] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29(9):2352–2449, 2017.

[42] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[43] Mohammad Hossin and MN Sulaiman. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5(2):1, 2015.

[44] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Coco - common objects in context. `http://cocodataset.org/#detection-eval`, 2015. Accessed in 15/06/2019.

[45] Tobias Koch, Lukas Liebel, Friedrich Fraundorfer, and Marco Korner. Evaluation of cnn-based single-image depth estimation methods. In *Proceedings of the European Conference on Computer Vision (ECCV)*, Munich, Germany, 2018.

[46] Huan Fu, Mingming Gong, Chaohui Wang, and Dacheng Tao. A compromise principle in deep monocular depth estimation. *arXiv preprint arXiv:1708.08267*, 2017.

[47] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760, Firenze, Italy, 2012.

[48] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems*, pages 2366–2374, Montreal, Canada, 2014.

[49] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, USA, June 2016.

[50] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Conference on Field and Service Robotics*, pages 621–635, Zürich, Switzerland, 2017.

[51] Braden Hurl, Krzysztof Czarnecki, and Steven Waslander. Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 2522–2529, Paris, France, 2019.

[52] NVIDIA. Nvidia drive constellation | nvidia developer. `https://developer.nvidia.com/drive/drive-constellation`. Accessed on 26/01/2020.

[53] Magnus Wrenninge and Jonas Unger. Synscapes: A photorealistic synthetic dataset for street scene parsing. *arXiv preprint arXiv:1810.08705*, 2018.

[54] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, Mountain View, USA, 2017.

[55] Bianca-Cerasela-Zelia Blaga and Sergiu Nedevschi. Semantic segmentation learning for autonomous uavs using simulators and real data. In *IEEE 15th International Conference on Intelligent Computer Communication and Processing*, Cluj-Napoca, Romania, 09 2019.

[56] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? *arXiv preprint arXiv:1610.01983*, 2016.

[57] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3213–3223, Las Vegas, USA, 2016.

[58] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, Las Vegas, USA, 2016.

[59] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual kitti 2. `https://arxiv.org/abs/2001.10773`, 2020. Accessed on 11/06/2020.

[60] Peide Cai, Yuxiang Sun, Hengli Wang, and Ming Liu. Vtgnet: A vision-based trajectory generation network for autonomous vehicles in urban environments. *arXiv preprint arXiv:2004.12591*, 2020.

[61] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation*, pages 1–9, Brisbane, Australia, 2018.

[62] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun. Driving policy transfer via modularity and abstraction. *arXiv preprint arXiv:1804.09364*, 2018.

[63] Xinyu Huang, Peng Wang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application. *arXiv preprint arXiv:1803.06184*, 2018.

[64] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

[65] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[66] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. *arXiv*, pages arXiv–1912, 2019.

[67] Cityscapes. Dataset overview – cityscapes dataset. `https://www.cityscapes-dataset.com/dataset-overview/#class-definitions`. Accessed on 02/04/2020.

[68] Waymo. Building maps for a self-driving car - waymo - medium. `https://medium.com/waymo/building-maps-for-a-self-driving-car-723b4d9cd3f4`. Accessed on 02/04/2020.

[69] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *arXiv preprint arXiv:1809.02165*, 2018.

[70] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, Columbus, USA, 2014.

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

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

[73] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[74] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, PP:1–21, 01 2019.

[75] Google Brain. models/detection_model_zoo.md at master · tensorflow/models. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`. Accessed on 14/04/2020.

[76] Facebook Research. detectron2/model_zoo.md at master · facebookresearch/detectron2. `https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md`. Accessed on 14/04/2020.

[77] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[78] Mahyar Najibi, Bharat Singh, and Larry S. Davis. Autofocus: Efficient multi-scale inference. In *The IEEE International Conference on Computer Vision (ICCV)*, Seoul, South Korea, 2019.

[79] Google Brain. automl/efficientdet at master · google/automl. `https://github.com/google/automl/tree/master/efficientdet`. Accessed on 14/04/2020.

[80] Jianan Li, Xiaodan Liang, ShengMei Shen, Tingfa Xu, Jiashi Feng, and Shuicheng Yan. Scale-aware fast r-cnn for pedestrian detection. *IEEE Transactions on Multimedia*, 20(4):985–996, 2017.

[81] Hoanh Nguyen. Improving faster r-cnn framework for fast vehicle detection. *Mathematical Problems in Engineering*, 2019:1–11, 11 2019.

[82] KITTI. The kitti vision benchmark suite. `http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=2d`. Accessed on 14/04/2020.

[83] Quanfu Fan, Lisa Brown, and John Smith. A closer look at faster r-cnn for vehicle detection. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 124–129, Gothenburg, Sweden, 2016.

[84] Liliang Zhang, Liang Lin, Xiaodan Liang, and Kaiming He. Is faster r-cnn doing well for pedestrian detection? In *European Conference on Computer Vision*, pages 443–457, Amsterdam, Holland, 2016.

[85] Ho Kwan Leung, Xiu-Zhi Chen, Chao-Wei Yu, Hong-Yi Liang, Jian-Yi Wu, and Yen-Lin Chen. A deep-learning-based vehicle detection approach for insufficient and nighttime illumination conditions. *Applied Sciences*, 9(22):4769, 2019.

[86] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3828–3838, Seoul, South Korea, 2019.

[87] Amlaan Bhoi. Monocular depth estimation: A survey. *CoRR*, abs/1901.09402, 2019.

[88] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4040–4048, Las Vegas, USA, 2016.

[89] Ashutosh Saxena, Sung H Chung, and Andrew Y Ng. Learning depth from single monocular images. In *Advances in Neural Information Processing Systems*, pages 1161–1168, 2006. Vancouver - Canada.

[90] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, Washington, D.C., USA, 2015.

[91] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns. In *International Conference on 3D Vision (3DV)*, pages 11–20, Qingdao, China, 2017.

[92] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1851–1858, Honolulu, USA, 2017.

[93] Zhenheng Yang, Peng Wang, Wei Xu, Liang Zhao, and Ramakant Nevatia. Unsupervised learning of geometry with edge-aware depth-normal consistency. *arXiv preprint arXiv:1711.03665*, 2017.

[94] Reza Mahjourian, Martin Wicke, and Anelia Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5667–5675, Salt Lake City, USA, 2018.

[95] Zhenheng Yang, Peng Wang, Yang Wang, Wei Xu, and Ram Nevatia. Lego: Learning edge with geometry all at once by watching videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 225–234, Salt Lake City, USA, 2018.

[96] Chaoyang Wang, José Miguel Buenaposada, Rui Zhu, and Simon Lucey. Learning depth from monocular videos using direct methods. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2022–2030, Salt Lake City, USA, 2018.

[97] Yuliang Zou, Zelun Luo, and Jia-Bin Huang. Df-net: Unsupervised joint learning of depth and flow using cross-task consistency. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 36–53, Munich, Germany, 2018.

[98] Zhichao Yin and Jianping Shi. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1983–1992, Salt Lake City, USA, 2018.

[99] Chenxu Luo, Zhenheng Yang, Peng Wang, Yang Wang, Wei Xu, Ram Nevatia, and Alan Yuille. Every pixel counts++: Joint learning of geometry and motion with 3d holistic understanding. *arXiv preprint arXiv:1810.06125*, 2018.

[100] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8001–8008, Honolulu, USA, 2019.

[101] Anurag Ranjan, Varun Jampani, Lukas Balles, Kihwan Kim, Deqing Sun, Jonas Wulff, and Michael J Black. Competitive collaboration: Joint unsupervised learning of depth, camera motion, optical flow and motion segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12240–12249, Long Beach, USA, 2019.

[102] Ariel Gordon, Hanhan Li, Rico Jonschkowski, and Anelia Angelova. Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8977–8986, Seoul, South Korea, 2019.

[103] Fabio Tosi, Filippo Aleotti, Matteo Poggi, and Stefano Mattoccia. Learning monocular depth estimation infusing traditional stereo knowledge. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9799–9809, Long Beach, USA, 2019.

[104] Wei Yin, Yifan Liu, Chunhua Shen, and Youliang Yan. Enforcing geometric constraints of virtual normal for depth prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5684–5693, Seoul, South Korea, 2019.

[105] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. From big to small: Multi-scale local planar guidance for monocular depth estimation. *arXiv preprint arXiv:1907.10326*, 2019.

[106] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[107] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. *arXiv preprint arXiv:1702.05374*, 2017.

[108] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 23–30, Vancouver, Canada, 2017.

[109] Yuhua Chen, Wen Li, Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Domain adaptive faster r-cnn for object detection in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3339–3348, Salt Lake City, USA, 2018.

[110] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

[111] Baochen Sun and Kate Saenko. From virtual to reality: Fast adaptation of virtual object detectors to real domains. In *British Machine Vision Conference*, volume 1, page 3, Nottingham, England, 2014.

[112] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L Waslander. Joint 3d proposal generation and object detection from view aggregation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, Madrid, Spain, 2018.

[113] Han-Kai Hsu, Chun-Han Yao, Yi-Hsuan Tsai, Wei-Chih Hung, Hung-Yu Tseng, Maneesh Singh, and Ming-Hsuan Yang. Progressive domain adaptation for object detection. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 749–757, Snowmass Village, USA, 2020.

[114] Nikolaus Mayer, Eddy Ilg, Philipp Fischer, Caner Hazirbas, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. What makes good synthetic training data for learning disparity and optical flow estimation? *International Journal of Computer Vision*, 126(9):942–960, 2018.

[115] Shanshan Zhao, Huan Fu, Mingming Gong, and Dacheng Tao. Geometry-aware symmetric domain adaptation for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9788–9798, Long Beach, USA, 2019.

[116] Amir Atapour-Abarghouei and Toby P Breckon. Real-time monocular depth estimation using synthetic data with domain adaptation via image style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2800–2810, 2018.

[117] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 969–977, Salt Lake City, USA, 2018.

[118] Åsmund Brekke, Fredrik Vatsendvik, and Frank Lindseth. Multimodal 3d object detection from simulated pretraining. In *Symposium of the Norwegian AI Society*, pages 102–113, Trondheim, Norway, 2019.

[119] FAIR. facebookresearch/detectron2: Detectron2 is fair's next-generation platform for object detection and segmentation. https://github.com/facebookresearch/detectron2. Accessed on 24/04/2020.

[120] Niantic Labs. nianticlabs/monodepth2: Monocular depth estimation from a single image. https://github.com/nianticlabs/monodepth2. Accessed on 24/04/2020.