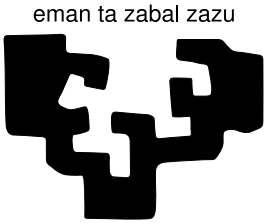


Universidad del País Vasco
Euskal Herriko Unibertsitatea

COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE DEPARTMENT

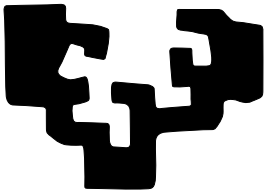
DATA-CENTRIC DESIGN AND TRAINING OF DEEP NEURAL NETWORKS WITH MULTIPLE DATA MODALITIES FOR VISION-BASED PERCEPTION SYSTEMS

by:
Nerea Aranjuelo Ansa

Supervised by:
Dr. Ignacio Arganda-Carreras
&
Dr. Luis Unzueta

Donostia – San Sebastian, Thursday 27th April, 2023

eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE DEPARTMENT

DATA-CENTRIC DESIGN AND TRAINING OF DEEP NEURAL NETWORKS WITH MULTIPLE DATA MODALITIES FOR VISION-BASED PERCEPTION SYSTEMS

by:
Nerea Aranjuelo Ansa

Supervised by:
Dr. Ignacio Arganda-Carreras
&
Dr. Luis Unzueta

Donostia – San Sebastian, Thursday 27th April, 2023

To my Family

"Life's battles don't always go
To the stronger or faster man.
But soon or late the man who wins,
Is the man who thinks he can."
R. Kipling

Abstract

The advances in computer vision and machine learning have revolutionized the ability to build systems that process and interpret digital data, enabling them to mimic human perception and paving the way for a wide range of applications. In recent years, both disciplines have made significant progress, fueled by advances in deep learning techniques. Deep learning is a discipline that uses deep neural networks (DNNs) to teach machines to recognize patterns and make predictions based on data. Deep learning-based perception systems are increasingly prevalent in diverse fields, where humans and machines collaborate to combine their strengths. These fields include automotive, industry, or medicine, where enhancing safety, supporting diagnosis, and automating repetitive tasks are some of the aimed goals.

However, data are one of the key factors behind the success of deep learning algorithms. Data dependency strongly limits the creation and success of a new DNN. The availability of quality data for solving a specific problem is essential but hard to obtain, even impracticable, in most developments. Data-centric artificial intelligence emphasizes the importance of using high-quality data that effectively conveys what a model must learn. Motivated by the challenges and necessity of data, this thesis formulates and validates five hypotheses on the acquisition and impact of data in DNN design and training.

Specifically, we investigate and propose different methodologies to obtain suitable data for training DNNs in problems with limited access to large-scale data sources. We explore two potential solutions for obtaining data, which rely on synthetic data generation. Firstly, we investigate the process of generating synthetic training data using 3D graphics-based models and

the impact of different design choices on the accuracy of obtained DNNs. Beyond that, we propose a methodology to automate the data generation process and generate varied annotated data by replicating a 3D custom environment given an input configuration file. Secondly, we propose a generative adversarial network (GAN) that generates annotated images using both limited annotated data and unannotated in-the-wild data. Typically, limited annotated datasets have accurate annotations but lack realism and variability, which can be compensated for by the in-the-wild data. We analyze the suitability of the data generated with our GAN-based method for DNN training.

This thesis also presents a data-oriented DNN design, as data can present very different properties depending on their source. We differentiate sources based on the sensor modality used to obtain the data (e.g., camera, LiDAR) or the data generation domain (e.g., real, synthetic). On the one hand, we redesign an image-oriented object detection DNN architecture to process point clouds from the LiDAR sensor and optionally incorporate information from RGB images. On the other hand, we adapt a DNN to learn from both real and synthetic images while minimizing the domain gap of learned features from data.

We have validated our formulated hypotheses in various unresolved computer vision problems that are critical for numerous real-world vision-based systems. Our findings demonstrate that synthetic data generated using 3D models and environments are suitable for DNN training. However, we also highlight that the design choices during the generation process, such as lighting and camera distortion, significantly affect the accuracy of the resulting DNN. Additionally, we show that a simulation 3D environment can assist in designing better sensor setups for a target task.

Furthermore, we demonstrate that GANs offer an alternative means of generating training data by exploiting labeled and existing unlabeled data to generate new samples that are suitable for DNN training without a simulation environment.

Finally, we show that adapting DNN design and training to data modality and source can increase model accuracy. More specifically, we demonstrate that modifying a predefined architecture designed for images to accommodate the peculiarities of point clouds results in state-of-the-art performance in 3D object detection. The DNN can be designed to handle data from a single modality or leverage data from different sources. Furthermore, when training with real and synthetic data, considering their domain gap and designing a DNN architecture accordingly improves model accuracy.

Resumen

Los avances en visión artificial y aprendizaje automático han revolucionado la capacidad de construir sistemas que procesen e interpreten datos digitales, permitiéndoles imitar la percepción humana y abriendo el camino a un amplio rango de aplicaciones. En los últimos años, ambas disciplinas han logrado avances significativos, impulsadas por los progresos en las técnicas de aprendizaje profundo (*deep learning*). El aprendizaje profundo es una disciplina que utiliza redes neuronales profundas (DNNs, por sus siglas en inglés) para enseñar a las máquinas a reconocer patrones y hacer predicciones basadas en datos. Los sistemas de percepción basados en el aprendizaje profundo son cada vez más frecuentes en diversos campos, donde humanos y máquinas colaboran para combinar sus fortalezas. Estos campos incluyen la automoción, la industria o la medicina, donde mejorar la seguridad, apoyar el diagnóstico y automatizar tareas repetitivas son algunos de los objetivos perseguidos.

Sin embargo, los datos son uno de los factores clave detrás del éxito de los algoritmos de aprendizaje profundo. La dependencia de datos limita fuertemente la creación y el éxito de nuevas DNN. La disponibilidad de datos de calidad para resolver un problema específico es esencial pero difícil de obtener, incluso impracticable, en la mayoría de los desarrollos. La inteligencia artificial centrada en datos enfatiza la importancia de usar datos de alta calidad que transmitan de manera efectiva lo que un modelo debe aprender. Motivada por los desafíos y la necesidad de los datos, esta tesis formula y valida cinco hipótesis sobre la adquisición y el impacto de los datos en el diseño y entrenamiento de las DNNs.

Específicamente, investigamos y proponemos diferentes metodologías para obtener datos adecuados para entrenar DNNs en problemas con acceso limitado a fuentes de datos de gran escala. Exploramos dos posibles soluciones

para la obtención de datos de entrenamiento, basadas en la generación de datos sintéticos. En primer lugar, investigamos la generación de datos sintéticos utilizando gráficos 3D y el impacto de diferentes opciones de diseño en la precisión de los DNN obtenidos. Además, proponemos una metodología para automatizar el proceso de generación de datos y producir datos anotados variados, mediante la replicación de un entorno 3D personalizado a partir de un archivo de configuración de entrada. En segundo lugar, proponemos una red neuronal generativa (GAN) que genera imágenes anotadas utilizando conjuntos de datos anotados limitados y datos sin anotaciones capturados en entornos no controlados. Por lo general, el primer conjunto de datos suele tener anotaciones precisas pero carecen de realismo y variabilidad, lo que compensamos con los datos de entornos no controlados. Analizamos la idoneidad de los datos generados con nuestro método para el entrenamiento de DNNs.

Esta tesis también presenta un diseño de DNNs orientado a datos, ya que los datos pueden presentar propiedades muy diferentes dependiendo de su fuente. Diferenciamos las fuentes según la modalidad de sensor utilizada para obtener los datos (p. ej., cámara, LiDAR) o el dominio de generación de datos (p. ej., real, sintético). Por un lado, rediseñamos una arquitectura DNN orientada a imágenes para detección de objetos en nubes de puntos del sensor LiDAR y, opcionalmente, incorporar información de imágenes RGB. Por otro lado, adaptamos una DNN para aprender de imágenes reales y sintéticas mientras minimizamos la brecha de dominio que ambos dominios presentan.

Hemos validado nuestras hipótesis formuladas en varios problemas de visión artificial no resueltos, que son críticos para numerosos sistemas basados en visión del mundo real. Nuestros hallazgos demuestran que los datos sintéticos generados utilizando modelos y entornos 3D son adecuados para el entrenamiento de DNNs. Sin embargo, también destacamos que las elecciones de diseño durante el proceso de generación, como la iluminación y la distorsión de la cámara, afectan significativamente la precisión del DNN

final. Además, mostramos que un entorno de simulación 3D puede ayudar a diseñar mejores configuraciones de sensores para una tarea objetivo. Adicionalmente, demostramos que las GAN ofrecen un medio alternativo para generar datos de entrenamiento mediante la explotación de datos existentes, etiquetados y no etiquetados, para generar nuevas muestras que sean adecuadas para el entrenamiento de DNNs, sin necesidad de un entorno de simulación.

Finalmente, mostramos que adaptar el diseño y entrenamiento de DNNs a la modalidad y fuente de datos puede aumentar la precisión del modelo. Más específicamente, demostramos que la modificación de una arquitectura predefinida diseñada para imágenes para adaptarse a las peculiaridades de las nubes de puntos da como resultado un rendimiento de vanguardia en la detección de objetos 3D. La DNN se puede diseñar para procesar datos de una sola modalidad o aprovechar datos de diferentes fuentes. También demostramos, que al entrenar con datos reales y sintéticos, considerar su brecha de dominio, diseñando una arquitectura de DNN acorde, mejora la precisión del modelo.

Acknowledgements

Putting into words the immense gratitude I feel towards those who supported and accompanied me on this journey is undoubtedly the most difficult part of this document.

First of all, I would like to express my sincere gratitude to my thesis supervisors, Ignacio Arganda-Carreras and Luis Unzueta, for their trust, guidance, and support throughout my research. Their insightful feedback and willingness to provide constructive criticism have been very valuable to this work, and I appreciate their dedication, especially in busy times. I remember the first days preparing the paper for the AMDO as if it were yesterday.

I would like to thank Julián Florez, Edurne Loyarte, and Jorge Posada for the opportunity to conduct this research at Vicomtech. This place has allowed me to grow personally and professionally.

I would also like to thank Oihana Otaegui, whom I greatly admire, for her trust and support since my early days. Special thanks to Marcos Nieto, for all the lessons over these years and for being a friend, a boss, and an inspiring leader, all together. I would like to express my gratitude to Unai Elordi, Guus Engels, José Luis Apellaniz, and Sara García for contributing to this research work too. I would also like to thank the rest of my colleagues (too many names to cite them all!), to whom I owe a lot for every day's support and the great environment they create. I am proud to be part of the group. Special mention to my colleagues and friends Juan Diego Ortega and Andoni Cortes, who, from the first day I arrived at Vicomtech, have been there whenever I have needed and whose company I greatly appreciate. I also feel grateful to Orti Senderos, whose light has filtered through this work's pages, just as it always does. Thanks to all the colleagues with whom I share or have shared caffeine in the past, you are great and so are the moments you

have given me. These moments are always essential for the most difficult times.

A special thanks to Donglai Wei, for hosting me at Boston College, for all the knowledge shared, and for the always inspiring discussions. Thanks also to Siyu Huang, for all the lessons and new ideas in the world of generative models. I feel grateful for our collaboration and everything I have learned from you. Thanks too, to everyone who made my Boston days memorable.

I want to mention my dear friends from university and school, who have always been aware of how I was doing and have always had words of encouragement when needed. Special thanks to Amaia Alustiza and Edurne Mayoz, who, apart from being incredible people and friends, have been through thick and thin to celebrate, encourage, help, or simply chat about this thesis or anything in life. I wish you all the best for your own life journeys.

I would like to express my heartfelt gratitude to my family, who have been a constant source of encouragement and support over these years (and always). Infinite thanks to my parents, to whom I will always be indebted for everything they have given me, starting with their endless and unconditional love. Thank you for the values that you have transmitted to me, many of them were key for my research to go forward. Among many other things, thank you for always believing in me and teaching me that perseverance, effort, and sacrifice are necessary to pursue goals and that humility and good humor should not be lacking. Last but not least, thank you Erik for always being there, for helping me maintain the work-life balance, for always believing in me, for the long talks, and for encouraging me to jump on all the adventure trains that come through life. I'm looking forward to going up with you to the next one.

Thank you with all my heart to all of you, for being part of this wonderful journey.

Gracias

Nerea Aranjuelo Ansa

May 2023

Contents

List of Figures

List of Tables

Part I

Introduction

The best way to predict the future
is to invent it.

Alan Kay

CHAPTER

1

Scope of the Research

1.1 Motivation

Computer vision is the field of computer science that focuses on developing digital systems capable of processing, analyzing, and interpreting visual data, such as images and videos, in the same way that humans do. Computer vision-based systems "see" or sense external stimuli through a sensor (e.g., camera) to get some target information so that it can be used in other processes. The idea behind computer vision is to teach machines to understand and interpret visual data in a manner that is similar to human perception. Computer vision involves using different techniques, such as machine learning and image processing, to enable machines to perform tasks that require human visual perception, such as object recognition, motion detection, and scene reconstruction.

Machine learning is a sub-field of Artificial Intelligence (AI). Machine learning is a broad discipline that studies algorithms and statistical models to perform a predefined task. Compared to computer vision, machine learning is not limited to visual data. The idea behind machine learning is to enable computers to learn from data, recognize patterns, and make predictions based on the data rather than using explicitly programmed rules. The core mechanism to do this is to train a model. This training stage typically involves using a dataset so that the model (algorithm) identifies patterns and relations from data and refines its predictions from that experience. For that purpose, several

techniques exist, such as decision trees, support vector machines, clustering algorithms, and deep learning.

Figure 1.1 shows that both computer vision and machine learning involve different techniques but also interact with each other. Both fields share some common techniques and are often combined in perception systems.

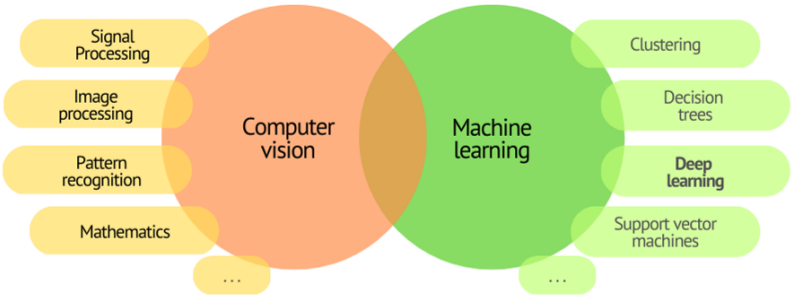


Figure 1.1: Computer vision and machine learning fields and some of their techniques.

Machine learning and computer vision often use deep learning to learn features from data. Deep learning is a field of machine learning that uses deep neural networks (DNNs) to learn relations and patterns from data. Deep learning has been outstanding for the last decade, as it has quickly become a method of choice for most machine learning problems. State-of-the-art results achieved by traditional algorithms have been rapidly overcome [9, 10]. Consequently, deep learning today is part of most machine learning systems. Thanks to deep learning, the advances in computer vision and machine learning tasks have expanded the possibilities to improve and create new advanced perception systems [11, 12].

Using computer vision to perceive an environment involves different stages, which include acquiring, processing, analyzing, and understanding the corresponding digital data, as shown in Figure 1.2. Vision-based perception systems require sensors, computers, and, most frequently, machine learning and image processing algorithms for these processes [13, 14]. The sensors mimic the eye function, and the algorithms mimic the brain function in interpreting and classifying image content.

For example, a vision-based intrusion detection system may monitor a region of interest to detect if any agent enters a restricted zone. The application receives the data captured by a sensor or a group of sensors (e.g., a surveillance camera). The system uses the developed computer vision algorithms to process the data in the chosen hardware,

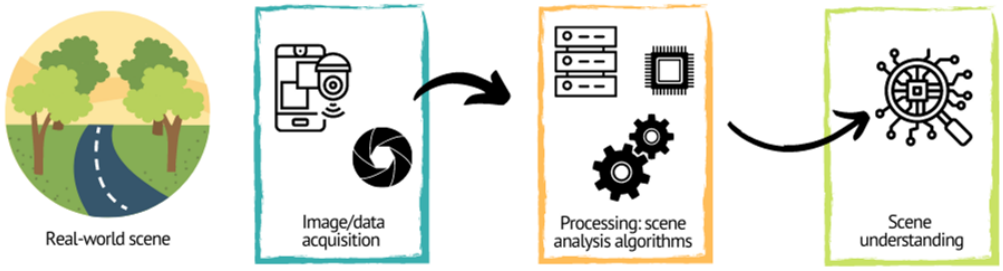


Figure 1.2: Overview of the different processes in computer vision-based systems.

for example, a server. It provides an alert if the corresponding algorithm detects an intrusion. These systems may be limited to perceiving the agents in a specific environment (e.g., people detection) or may also interpret the scene or complete more complex tasks, such as predicting the behavior of different actors. These systems have multiple fields of applications, such as robotics, automotive, or surveillance [15, 16, 17]. Figure 1.3 shows examples of different computer vision tasks, such as traffic sign recognition, driving scene segmentation, or pose estimation.

Figure 1.3: Different computer vision task examples. From left to right and top to bottom: traffic sign recognition, face and key points detection, scene segmentation, instance-based segmentation, pose detection, and people detection and tracking.

Perception-based systems have improved many tasks we perform daily without even realizing it. We find computer vision-based systems in our cars, in the Advanced Driving Assistance Systems (ADAS) [18], that, for example, allow us to drive safer because they "see" the road lines and ensure we drive correctly between them. Computer vision is

also in our mobiles, in applications such as authentication, which uses the mobile camera to rapidly verify if the person trying to unlock the mobile is the owner. Computer vision is also present in many factories, where camera and computer vision-based applications automatically perform quality assessments to detect defective products. These are only a few examples of the multiple computer vision applications we interact with within different fields. Some potential advantages that these kinds of systems could provide us are the possibility to complete repetitive tasks faster and more efficiently, reduce costs, improving security, and assist humans in complex tasks, among others. We are still about to see the potential and the possibilities the latest advances will bring us.

However, the creation and success of a new deep learning-based perception system involve a significant challenge: data dependence. The success of deep learning models in solving specific tasks is closely related to the availability of appropriate and quality data for the corresponding problem [19]. Obtaining and processing these data considering their properties is essential but not trivial for getting an accurate and robust model. Correct model behavior is indispensable for the success of the system. Given these challenges, in this thesis, we explore and propose different potential solutions to alleviate the challenges, problems, and bottlenecks that data can pose when developing vision-based perception systems. Figure 1.4 shows the overview of the opportunities and problems that motivate the presented research.

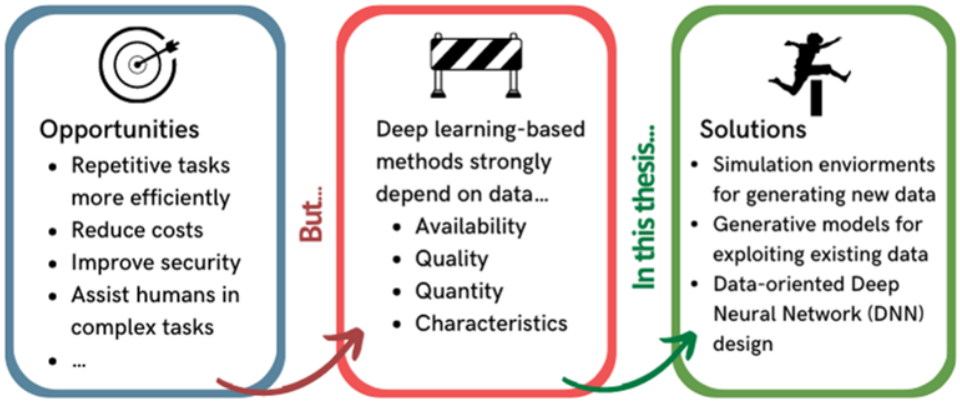


Figure 1.4: Deep learning for vision-based perception systems opens up new opportunities, but their success is hindered by data dependence. This thesis presents different potential solutions to alleviate these challenges.

1.2 Deep Learning

Deep learning allows computational models composed of multiple processing layers to learn representations of data with various levels of abstraction [9]. These computational models are the DNNs. The term *deep* emphasizes the number of processing layers, or depth, of the modern DNNs compared to classical artificial neural networks (ANNs).

The goal of a DNN is to find an approximated function $f*$ that maps an input x to a category y. The network architecture defines a mapping $y = f(x,\theta)$, and during the training, it learns the value of the parameters θ that results in the best function approximation. For example, if we work with images and want to classify each sample depending on the object shown in the image, we would need to train a DNN to find the function $f*$ that maps the input set of pixels of each image to an object category y. Deep learning breaks this complex mapping into a sequence of simple nested mappings done in the model's different layers, as shown in the example in Figure 1.5.

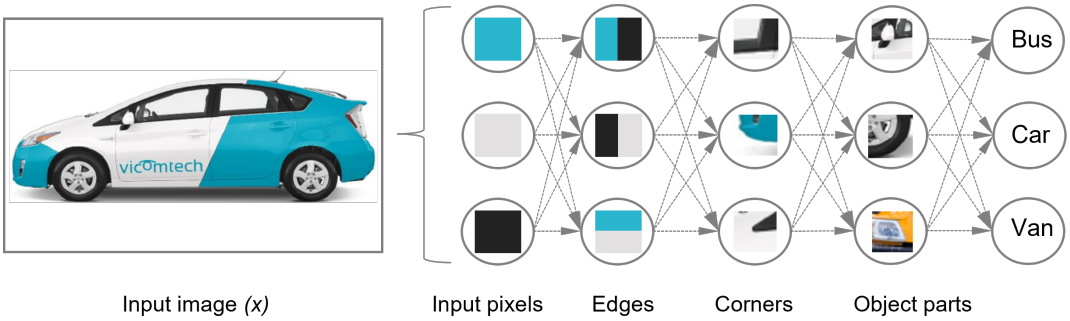


Figure 1.5: Intuition of the hierarchical representations of the data that a DNN learns to complete a task such as image classification.

Figure 1.5 represents a DNN that classifies the category an image belongs to, in this case, bus, car, or van. The input to the network is formed by the image pixels fed to the DNN in the input layer. Then, some hidden layers extract increasingly abstract and complex features from the image. The images in each layer show the kind of features represented by each hidden layer. The first layer extracts simple features, such as the edges of the objects in the image. The second hidden layer looks for corners and some contours based on the data from the previous layer. Given the information on the corners and contours of the image, the third hidden layer detects specific object parts, which are formed by contours and corners. Depending on the object parts detected

in the input data, the image is finally classified into a predefined category. Deep learning allows the DNN to learn complex concepts out of simpler concepts. It is the model during the training phase that must determine which concepts are useful and should be extracted in each hidden layer to explain the relationships in the data and solve a specific task (e.g., image classification).

Deep learning has changed the way researchers work. As shown in Figure 1.6, in traditional machine learning methods, the researcher is in charge of designing the features that need to be extracted from data to solve a specific task (hand-crafted features), while with deep learning, the models are capable of learning the features that should be extracted from the given data (machine-learned features).

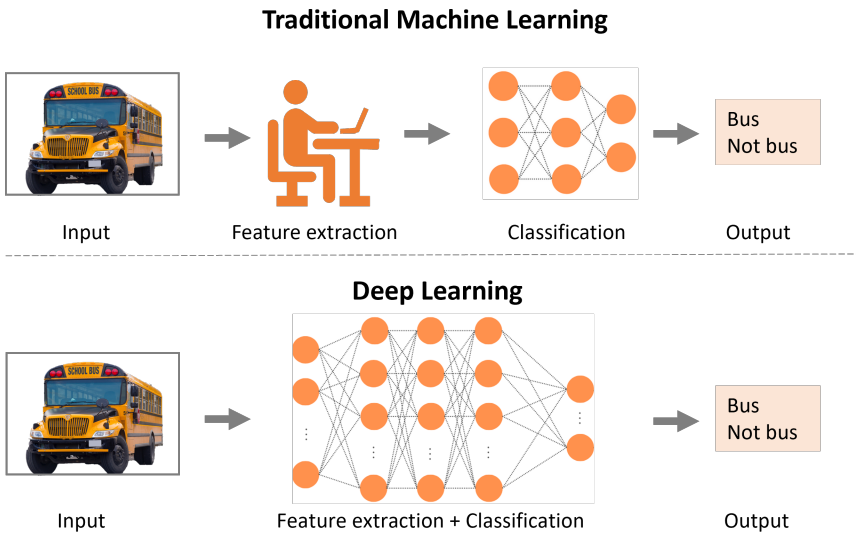


Figure 1.6: The change of paradigm from traditional machine learning to deep learning. With deep learning features are no longer hand-crafted, they are automatically learned from data.

1.3 Lifecycle of Model Development

Developing the core deep learning model for deploying a vision-based perception system involves a lifecycle with different stages. Figure 1.7 presents the main stages. The first step is defining the problem to be solved with a clear scope and understanding. Then, the sensor setup in charge of data capturing must be designed. Data acquisition

is necessary for training the corresponding DNN. The last step is deploying the model in the real environment, but this is usually not the cycle's end. It is common to keep improving the machine learning model with the new data captured in the real scenario or go back to a previous step to refine the model to ensure the system works as accurately as desired.

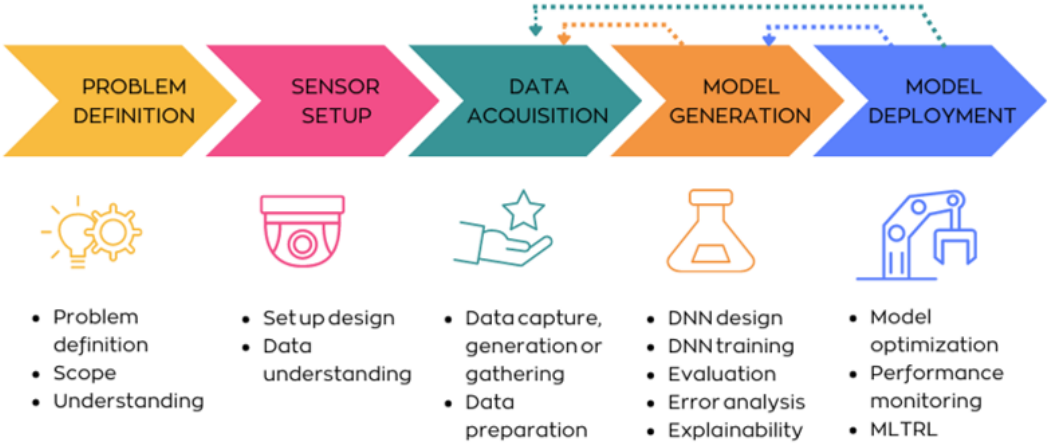


Figure 1.7: Different stages in the development of a DNN-based perception system.

Machine Learning Operations (MLOps) might help in a model's lifecycle. MLOps are a set of tools and practices for automating and simplifying the management and deployment of machine learning models [20]. MLOps aims to improve the speed, quality, and reliability of deploying machine learning models to production. For example, MLOps include tools for data and model version controlling, testing, and monitoring.

The different stages in the lifecycle of model development are described below.

1.3.1 Problem Definition and Scope

The first step is understanding and defining the problem that the system should solve. This stage is vital to the project's success, as all the other tasks will depend on the defined goal. The use cases the system should solve, the operative conditions (under what circumstances do we want it to work?), the available resources, or the scope of the system (e.g., target KPIs) should be clearly defined. Problem understanding is also important for the success of the following stages. Collecting the correct data or designing a proper setup is difficult if the problem is poorly understood.

1.3.2 Data Understanding and Sensor Setup Design

The second step is to understand what kind of data we need to solve the corresponding task and what kind of sensors and setup provide that. Data can come from different sensing devices. Cameras are one of the most common sensors for perceiving the world by imitating the human eye and developing vision-based systems [21, 22]. However, there are different possibilities among cameras depending on the problem we must solve. RGB cameras are often used in perception systems, but thermal or depth cameras may be more appropriate depending on the task to be solved. Other sensors, such as LiDAR, could be more suitable for problems requiring an accurate 3D representation of the surroundings [23]. Different sensor modalities and specifications must be considered for designing a proper setup for solving the target problem. It is also common to have a setup that includes various sensor types. The setups that combine different data sources (e.g., sensor types) or data types are known as multimodal [24]. It is essential to understand the type of data each sensor provides, its advantages and drawbacks, and the properties of the data they provide. For example, Figure 1.8 shows a driving scene captured by an RGB camera (top) and a LiDAR sensor (bottom). The image and the point cloud obtained from the signal captured by both sensors have different properties that should be considered to make the best of their information. This is tightly related to the goal of the system but also to the way the machine learning model processes the data.

In addition to the sensor types, the system design also requires defining the number of sensors, specifications, positions, and orientations. The decisions made in this stage condition the machine learning models and algorithms, which will be developed in the next stages, but can also limit the scope of the system or include redundant sensors by mistake. The most effective way to design an appropriate and optimal setup and maximize the potential of a data modality is an open question.

1.3.3 Data Acquisition

Once the sensor setup is designed and to train a deep learning model, data are required. One of the main factors that have driven the DNNs revolution is the data. Large quantities of available data have been crucial for the DNNs' training and success. However, not only the amount is important but also the quality. Techniques, such as fine-tuning [25],

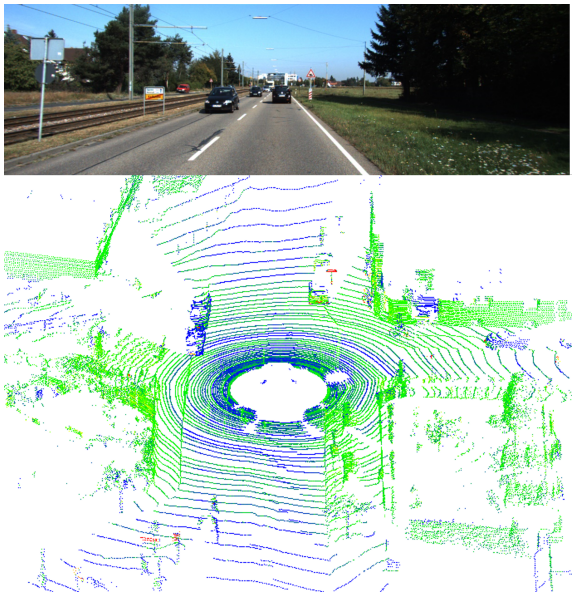


Figure 1.8: Example of a driving scene captured by an RGB camera (top) and a LiDAR (bottom) from the KITTI benchmark [1].

allow training DNNs with smaller datasets (e.g., thousands or hundreds of samples) by taking advantage of the knowledge learned in previous trainings with big generalist datasets, which may contain millions of samples. Quality is crucial for the success of model training. Data quality includes different aspects, such as having representative samples for the target domain, balanced data, or well-annotated data in the case of supervised models. Having enough samples with enough quality is usually a significant challenge. It is also important to consider that the data we collect or generate may lead to undesired bias if we ignore the data distribution and may even lead to ethical problems. Public large-scale datasets are often used for DNNs training [26], but obtaining training data is largely application specific. Another way to get the data is by generating them, capturing them, or generating them artificially, for example, using a virtual environment (Figure 1.9).

Once we have enough data, there is an additional preparation step before the training. This step includes cleaning the data (e.g., noisy or irrelevant data), converting them to a proper format, and preprocessing them if required. In order to train supervised models, annotations are also required. Annotating the data is not a trivial task. It requires a clear annotation protocol, appropriate annotation tools for the data modality

(e.g., videos, LiDAR scans), and a big manual effort. In order to have good-quality data is important to ensure that the annotations are defined consistently, data cover important cases, and that there are enough samples.

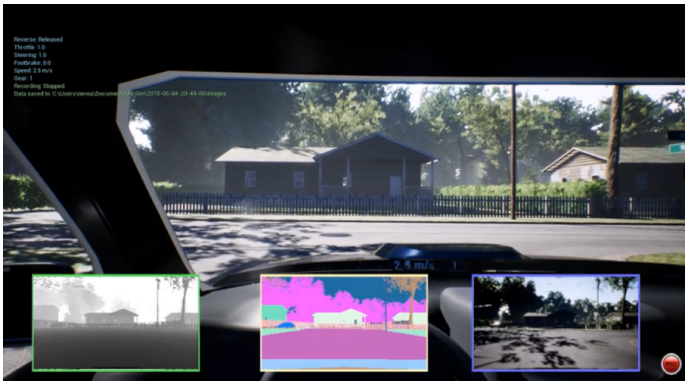


Figure 1.9: Example of synthetic data and annotations generated with AirSim simulation environment [2].

1.3.4 **Model Design and Training**

DNN design typically does not begin from scratch. It is common to use DNN architectures that have shown good accuracy for a specific task as the baseline for training a model for a new target problem. In the last years, some DNNs have become popular because of their good performance in generalist benchmarks such as the ImageNet one [26] (e.g., AlexNet [27], ResNet [28], EfficientNet [29]). Consequently, these models, along with their pretrained weights, are often used as the baseline for training a new model. The architecture is often not adapted even if the new data have different properties, which may not be the most optimal approach for different data modalities. These common practices lead us to the following open questions: Can data from different modalities be handled in the same manner? Is it enough to select and train a predefined DNN architecture with the new task data? If we generate synthetic data for the DNN training, is it appropriate to train with them just as with real data?

Once the DNN is designed or selected, model training requires a proper hyperparameter setup and training strategy for model convergence. Model training and parameter tuning is an iterative process that often requires model redesign and training dataset

modifications. The training dataset, the training setup, the DNN architecture, or a combination of these factors may cause low accuracy or poor generalization obtained by a model. Error analysis is crucial for guiding the changes to real model improvements. Error analysis can help identify the data type the model does poorly on. Model explainability techniques may be used to help to guide the required improvements. For example, model explainability can consist of giving insights into the part of the image a model relies on for a specific answer [30].

1.3.5 **Model Deployment**

Deploying the trained model in the target system may require additional steps, especially when the available hardware has limited resources. Different techniques exist to optimize the model inference, such as model compression via pruning or quantization [31]. Knowledge distillation [32] might be another choice for getting a smaller DNN to transfer the knowledge from a bigger one when the resources are scarce. The target deploying environment may also require special inference pipelines. For example, suppose the developed vision-based perception system uses an edge device to capture sensitive data and has not enough processing capability. In that case, a possible inference strategy may be partitioning the DNN and distributing its computation between the device and the cloud. These are some of the aspects that need to be considered when deploying the model.

Once we deploy the system, its development lifecycle may not be over if we include some monitoring functionalities to ensure its good performance, avoid deterioration of performance over time (e.g., data drifting), or keep improving the vision-based model with the new data captured while running in the real scenario (e.g., continual lifelong learning [33]).

The rapid adoption of machine learning-based systems in the last years has promoted the debate on how AI should be developed and regularized to address the opportunities, challenges, and risks it entails. For example, the European Union AI Act [34] is a European law on AI, defined to manage AI risk and prioritize human rights in the development and deployment of AI.

Once we deploy the model for its correct usage and adoption, it is important to know its possibilities and limitations. Model cards [35] can be included as documentation

that describes the context in which models are intended to be used and promote transparency in AI. Model cards include detailed performance evaluations and potentially relevant information about the corresponding model. Machine Learning Technology Readiness Levels (MLTRL) [3] can also be used to represent the maturity of a model, data pipelines, software module, or composition thereof. MLTRLs assess the maturity and readiness of the technology for commercialization. It is a way to evaluate the progress of a machine learning system from its initial conception to its implementation. MLTRLs define ten different maturity levels shown in Figure 1.10. For example, level 4 corresponds to a proof of concept (PoC) development. This stage is typically the first touch-point to demonstrate the utility of specific technology in a real scenario, taking care to report assumptions and limitations of the solution. The MLTRL definition helps better understand the maturity, scope, and limitations of developed algorithms or systems.

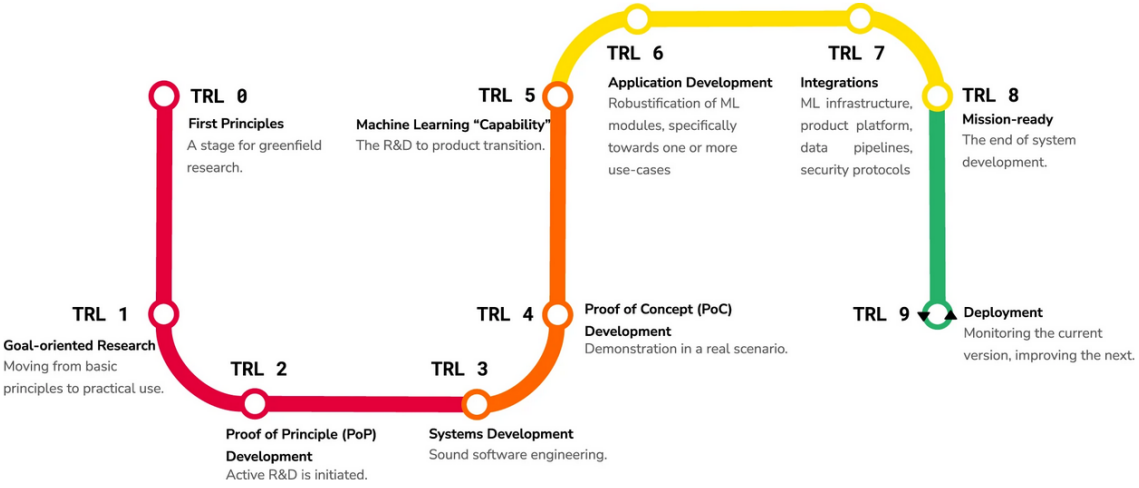


Figure 1.10: Machine Learning Technology Readiness Levels (MLTRL) defines a range of readiness levels, starting from research (red), continuing through prototyping (orange), leading to productization (yellow), and finally ending with deployment (green) [3].

1.4 The Role and the Challenges of Data

Data are critical and necessary to build AI systems [19]. To train DNNs-based models, suitable training data are key to help DNNs learn appropriate pattern recognition features, which means the success or failure of the model. Obtaining an appropriate

training dataset is a challenge. In addition, note that data are present in all the lifecycle of DNN-based system development in different ways and are the core of many uncertainties about the best way to proceed (Section 1.3). Figure 1.11 shows some of these uncertainties. However, it is common for researchers to concentrate on creating better models rather than better datasets. This section presents some key challenges and use cases that have not been solved yet and are the focus of the research presented in this thesis project.

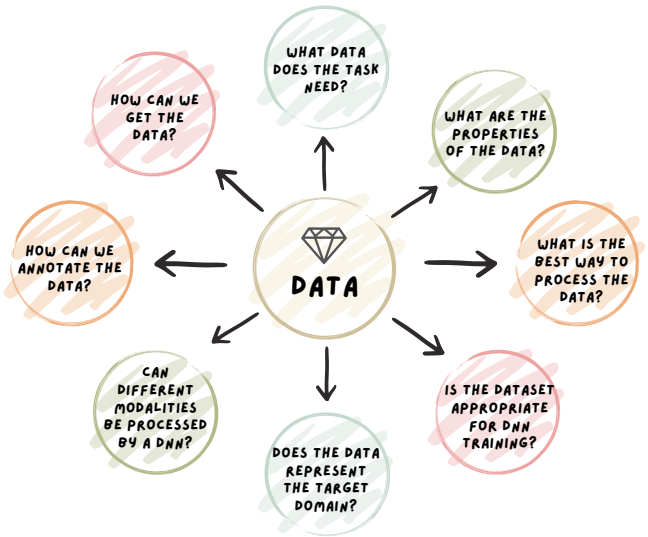


Figure 1.11: Different questions that arise around the data when developing a DNN-based perception system.

Data-centric AI focuses on data instead of code. The goal is to work with quality data to ensure that the data clearly conveys what the model must learn [36]. Data quality and excellence [37] are decisive for avoiding reduced accuracy when a model is deployed in the real world and avoiding negative issues such as bias [38]. Compounding events causing negative, downstream effects from data issues are also known as data cascades [39]. Very often, the accuracy problems when deploying a model do not come from the model but from the training data. Initiatives such as data cards [40] try

to promote transparent and responsible AI.

Training dataset importance cannot be decoupled from the data-obtaining process itself. The way to gather the data implies specific problems or challenges that might arise during training or final model deployment. There are three common ways to obtain the training data: collect data from the Internet (e.g., web scraping, large-scale public datasets [41]), capture data, or generate data synthetically [42].

In the vast majority of cases, the data cannot be directly extracted from the Internet since the new application tries to solve a particular problem in a specific scenario with a specific sensor setup. Thus, custom datasets need to be created. Data capturing implies the execution of several actions such as target data specification, sensor setup installation, technical preparation, recording protocols, possibly even actors simulating precise guide notes, and finally, annotating the data. These are some of the reasons that promote the alternative way of generating the data artificially. Artificial or synthetic data generation is a possible choice to obtain training data for DNNs, but it also can augment the accuracy drop when the model is deployed in the final target environment. The reason is the domain gap or difference between the data distribution of the source domain (virtual world) and the target domain (real world). This does not happen only when training with synthetic data. The domain gap often happens because of discrepancies between the training and target domain, even if both belong to the real world. However, the gap when working with synthetic data is bigger, and it is of special relevance due to the possibilities and current popularity of synthetic data for DNN training. Despite the benefits of using synthetic data and being an increasingly used technique, we identify the following uncertainties:

- **Synthetic data generation design using 3D models**. The use of synthetic data may help in generating sufficient and balanced training data. However, models trained with such data often present a domain gap when applied to real-world scenarios. Many studies focus on techniques such as domain adaptation to minimize this gap, but little attention is paid to the data generation itself. Do the design choices made during this process impact the DNNs' accuracy? Is this impact negligible? Are there any guidelines that should be followed?

- **Simulated 3D environments for deep learning.** We can generate synthetic data following different strategies, such as compositing real data, relying on generative

models, or using simulated environments. The latter has an additional advantage when building multi-sensor systems, as it can also help design the sensor system to cover the use cases in the targeted real scenario. This is particularly relevant in scenarios not easily accessible for system designers and/or when the sensor positions and characteristics are not predefined. In addition, the agents in the environment can interact with each other or with the environment itself. However, there are no general simulated environments ready to solve the sensor setup and data generation tasks for all kinds of scenarios and use cases. Typically, 'ad hoc' environments are built. Would it be possible to have a solution with sufficient generality for generating suitable training data for DNNs in different contexts? Would data from a simulated environment help in DNN training?

- **Synthetic data generation for deep learning using generative models.** It is also possible to use generative models to generate training data. The state-of-the-art generative adversarial networks (GANs) can generate very realistic images with no need for a 3D simulation environment. In addition, data realism might alleviate the possible domain gap between synthetic and real data domains. In many computer vision tasks, obtaining annotated data is a tedious and challenging task. Would it be possible to use GANs to generate appropriate training data for DNNs? Would it be better to use a 3D simulated environment?

Model-centric AI focuses on obtaining high accuracy working on the DNN architecture improvement rather than the data itself. Benchmark datasets [1, 26, 41] are the point of reference to measure how good or bad a model is. Consequently, much of the research continues to focus on improving the algorithmic aspect of the models to surpass state-of-the-art models. These benchmarks have accelerated machine learning development, even if questioning or analyzing the data has generally been left in the background.

Most of the time, the success of an AI-based system cannot be limited to working on the data or the model. Even if we use a promising state-of-the-art DNN for solving a task, if training data properties are not considered, we might lose a lot of room for improvement. Data properties include, for example, the domain of the data (e.g., real or virtual world) or the modality of the data with its particularities (e.g., point clouds, images). Regarding data modality, when a new task gains attention in machine learning research,

it is common to try to adapt solutions that have already worked to the new problem. If data are different, special care should be taken to adapt the techniques correctly. We find an example of this situation in the automotive field, where it is common to have multi-sensor setups and where new approaches emerge as machine learning and sensors progress. When designing and training a DNN with different data modalities or domains, we identify the following uncertainties:

- **Data-oriented deep neural network design: data modality.** Driven by the importance of 3D scene understanding in the automotive field and the adoption of the latest LiDAR sensors, diverse works have emerged with different deep learning-based proposals for 3D object detection. Advances have been made rapidly and show promising results, but it is still an open question what the best way to process point cloud data with DNNs is. Many works try to adapt mature networks commonly used for camera images to this task [43, 44, 45], but they usually do not consider the special properties of LiDAR point clouds. An object in a LiDAR point cloud at a distance of 5 meters from the sensor and the same object at 40 meters do not have the same distribution and quantity of points. In addition, point clouds are unstructured compared to images. Is it possible to adapt image-oriented DNNs to the new data modality? In that case, how should it be done? Do current proposals consider the data's particularities? Can these DNNs handle both data modalities successfully?

- **Data-oriented deep neural network design and training: data domain.** Adequate and abundant data are crucial for DNNs to learn suitable pattern recognition features and develop sufficient generalization abilities. Although synthetic data might mitigate the burden of producing the corresponding data, there is typically a domain gap between synthetic and real-world samples. Can the DNN architecture help to minimize this domain gap? Could considering the data source domain during model design and training result in improved model performance?

1.5 Goals and Contributions

As described in previous sections, data are key in the process of developing a DNN-based perception system. However, obtaining an appropriate training dataset for DNNs

is hard. Researchers increasingly adopt synthetic data generation to alleviate the corresponding difficulties. The way the synthetic data are generated might impact the final accuracy of the system, but little attention is paid to the choices made in the data generation design. Simulated environments can help in synthetic data generation, but there are no generalist solutions prepared for the custom problems faced by most real applications. Generative models are another possibility to generate data with no need for a 3D simulated environment. Even after obtaining the training data, special care should be taken to process them correctly by the DNN, depending on their properties. The properties of the data depend on their source (e.g., captured or simulated data) but also on the sensor modality they come from (e.g., LiDAR, camera).

In order to bring some light to the uncertainties and challenges that data involves in DNN training, this thesis aims to validate the following hypotheses:

1. **Hypothesis 1: Artificially generated samples using 3D models and environments are valid for training DNNs.** To validate if 3D graphics-based synthetic data might help to train DNNs, we propose a methodology to generate synthetic data to solve a computer vision task that lacks available data: people detection in large spaces from omnidirectional cameras. We analyze the impact of different design choices and the target setup properties in the final DNN accuracy. We compare the results when using real and synthetic data. This work is presented in Chapter 4.

2. **Hypothesis 2: A simulated 3D environment can help define the required data and sensor setup for vision-based perception systems and generate appropriate data for DNN training with a high level of automation.** To validate this hypothesis, we define a methodology to build a 3D simulated environment for configuring and training multi-sensor systems with sufficient generality to be usable in different surveillance contexts. We show a practical implementation of the method in the context of digitized on-demand aircraft cabin readiness verification. We generate synthetic data to train a DNN to identify when luggage are incorrectly placed. This work is presented in Chapter 5.

3. **Hypothesis 3: Artificially generated samples using generative models are valid to train DNNs with no need for a simulation environment.** We design and train

a conditioned GAN for gaze estimation task to verify this hypothesis. This task requires challenging data, and publicly available annotated data are very limited. Chapter 6 presents our research on using GANs to exploit labeled and unlabeled existing data and their impact when used in DNN training.

4. **Hypothesis 4: Adapting the DNN architecture design to the data modalities improves the results obtained by predefined and pretrained models.** We adapt an image-oriented DNN architecture to 3D object detection from point cloud data to confirm this hypothesis. We train the proposed model with the new data modality and compare its accuracy to state-of-the-art methods when using only point cloud data and also when including additional data from an RGB camera. This work is presented in Chapter 7.

5. **Hypothesis 5: Adapting the DNN architecture design to the data domains improves the results obtained by predefined and pretrained models.** To validate this hypothesis, we propose a DNN design and training pipeline that handles real and synthetic images in the context of aircraft cabin readiness verification with a camera-based system. We analyze the impact of training a classification DNN with domain adaptation and images' redundancy reduction on the final model accuracy. This work is presented in Chapter 8.

The current document is organized as follows. Chapter 1 presents the motivation, introduction, and goals of this thesis. Chapter 2 introduces the reader to the fundamentals of deep learning, and Chapter 3 is an in-detail study of relevant related work for this thesis research. The remaining chapters present the research results. In Chapter 4, we explore synthetic data generation design and propose a methodology to generate appropriate training data for DNNs using 3D models. Chapter 5 presents a methodology to build a simulated 3D environment to design adequate sensor setups and automatize training data generation for DNNs. In Chapter 6, we investigate generative models for generating DNN training data. This chapter shows the design and training of a GAN for leveraging labeled and unlabeled existing data. Chapter 7 presents DNN architecture adaptation to data modality (RGB images and LiDAR point clouds) and Chapter 8 to data

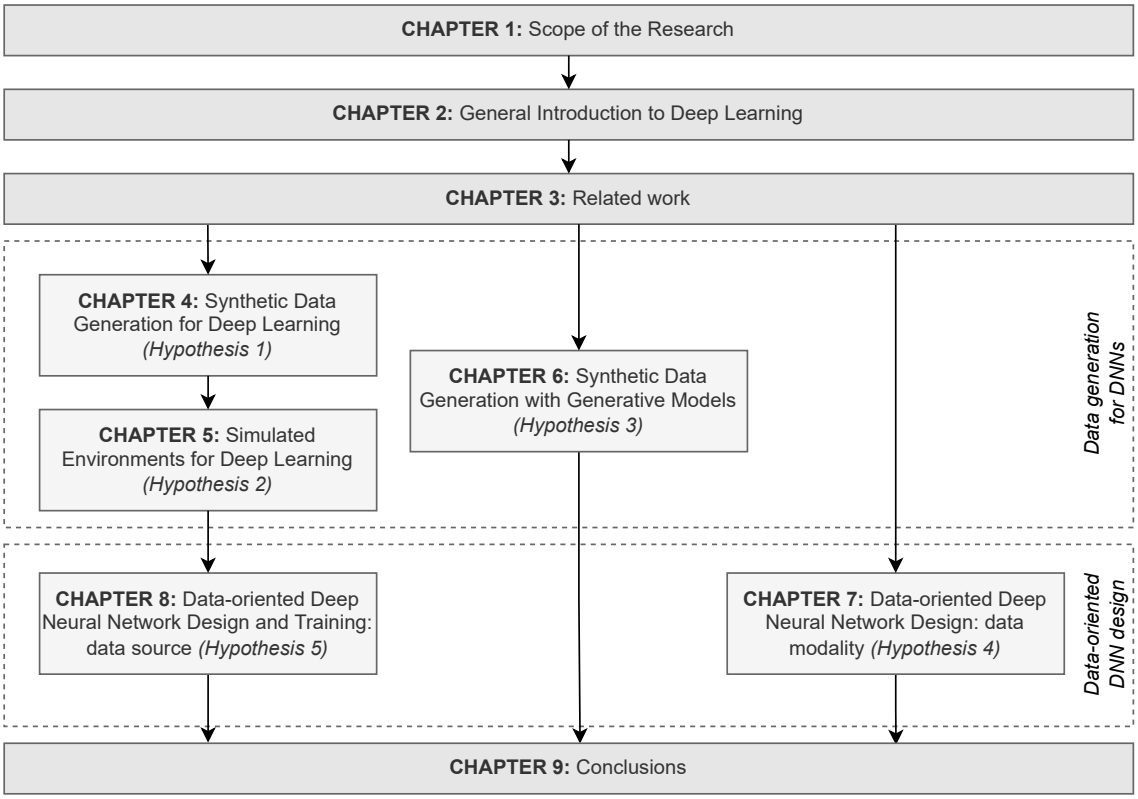


Figure 1.12: Organization of chapters and connections between them and the formulated research hypotheses in this thesis.

domain (real and synthetic domains). Finally, Chapter 9 summarizes the presented research's conclusions and future work. Figure 1.12 shows an overview of the organization of the chapters and their connection with the formulated research hypotheses.

We have published the research results obtained in this thesis in several publications presented at international conferences and journals. We refer the reader to the Appendix (Section A) to see an in-detail list of the papers published during this research.

Part II

State of the Art

What is now proved was once only imagined.

William Blake

CHAPTER

2

General Introduction to Deep Learning

2.1 Artificial Neural Networks

2.1.1 The Origin of Deep Learning

Deep learning is a machine learning technique that uses artificial neural networks (ANNs) to extract features and learn patterns from data to complete a specific task. The origins of deep learning can be traced back to the mid-1950s. In the early days of AI, researchers wanted to make computers learn and build an artificial electronic brain. In 1958 psychologist Rosenblatt proposed the perceptron, a simplified mathematical model of a neuron inspired in our brain neurons [46]. Figure 2.1 shows the analogy between the artificial neuron (the perceptron) and the biological neuron.

A biological neuron receives electrical signals from other neurons through dendrites. The neuron's nucleus produces an output signal based on the signals provided by dendrites. The neuron fires an output signal when the total strength of the input signals exceeds a certain threshold. This output signal is carried away by the axon. In the artificial neuron model (the perceptron), the neuron takes a set of inputs, multiplicates each of them by a weight value, and sums these weighted values. The sum is thresholded

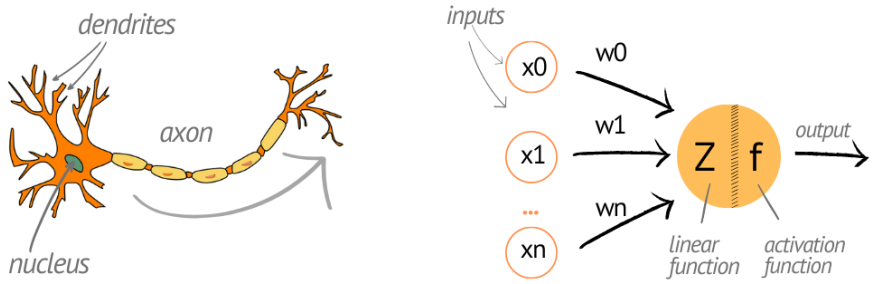


Figure 2.1: Analogy between the biological neuron (left) and the perceptron (right).

to output 1 or 0, depending on the value. A bias value is added to be able to offset the summed value. Equation 2.1 defines the perceptron mathematically.

$$y = f(\sum_i w_i \cdot x_i + b) \tag{2.1}$$

where x_i are the input data, w_i are the weights, b is a bias value, and f is the activation function that, in this case, outputs 1 if the input is bigger than 0 and 0 on the contrary (step function). The specific values of the weights and the bias are defined after an optimization process, where data are fed to the mathematical model, and the optimal parameters are estimated for minimizing a predefined loss function. The way that the perceptron learns the weights and the bias is one of the first supervised learning methods for neural networks. During this phase, some input values are fed to the model, as well as some expected outputs. If the output of the model for a set of weights and bias values does not result in the expected output, we adjust the values to minimize the difference (error) iteratively. This stage is referred to as the training of the model.

The perceptron's parameters (weights and bias) are learned in a supervised way because we use inputs and the expected outputs for each input. For each sample, we know the category it should belong to, and we use this information to find the optimal parameters.

The perceptron can be used as a binary linear classifier. Suppose we have a set of data with positive and negative examples which are linearly separable (Figure 2.2 (left)).

In that case, we could use a perceptron to find a separation hyperplane in the data. Once we find the hyperplane, given any new sample, we can classify it into the corresponding category. However, in the real world, most problems are not linearly separable (Figure 2.2 (right)). In order to solve non-linear problems, multi-layer perceptrons (MLP) and non-linearity functions are introduced.

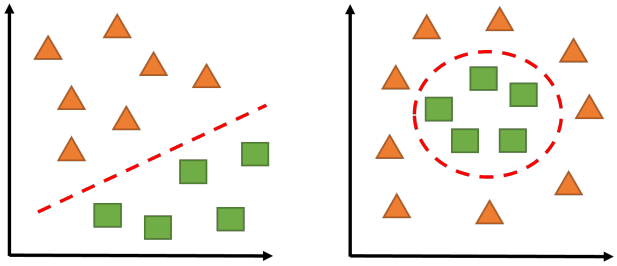


Figure 2.2: Linearly separable data (left) and non-linearly separable data (right).

2.1.2 The Multi-layer Perceptron

The inputs to a perceptron might be some input data or the output of another perceptron. Putting together different perceptrons, we build an MLP. The MLP is a group of perceptrons stacked in several layers. Figure 2.3 shows an MLP with three layers. The outputs of each perceptron in the first layer (input layer) are fed to all neurons in the second layer (hidden layer). The hidden layer outputs are the input to the final layer (output layer). Each signal from one perceptron to another is multiplied by different weights w. In the MLP, the connections between nodes do not form a cycle, the information moves in only one direction (forward) from the input nodes to the output nodes. When a network meets this condition, it is called a feed-forward network.

The MLPs can have a different number of hidden layers, and each layer can have a different number of perceptrons or neurons. The activation functions of each neuron can be linear or not. Adding non-linearity functions to the model enables learning non-linear relationships in data, which are often found in real-world data. There are multiple possible non-linear activation functions. Some common functions are the following ones.

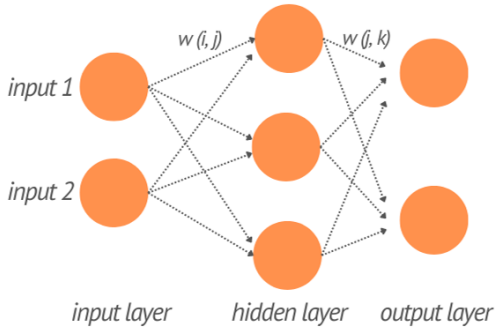


Figure 2.3: Multi-layer perceptron architecture.

Rectified Linear Unit (ReLU): The function ReLU provides the same value as the input when it is positive and a value of zero for negative input values. The neuron is not activated for negative values (Equation 2.2).

$$f(x) = max(0, x) \tag{2.2}$$

The ReLU function cannot learn via gradient-based methods on examples for which the activation is zero. The weights and biases for some neurons are not updated during the optimization process. Therefore, different alternatives of the ReLU exist based on using a non-zero slope for negative values.

Leaky Rectified Linear Unit (Leaky ReLU): Leaky ReLU is defined to address the gradients problem for negative values in ReLU [47]. It returns small values instead of zero for negative input values (Equation 2.3). In Leaky ReLU, α is fixed to a small value like 0.01, but a variant of the Leaky ReLU, called parametric ReLU (PReLU), treats α as a learnable parameter [48].

$$f(x) = max(\alpha x, x) \tag{2.3}$$

Sigmoid / logistic activation function: Before introducing the ReLU function, most neural networks used the sigmoid or hyperbolic tangent activation function. The sigmoid function takes a real value as input and outputs a value from 0 to 1 (Equation 2.4). The output saturates across most of the function domain, and it is sensitive to the input when it is near 0.

$$f(x) = \frac{1}{(1+e^{x})} \tag{2.4}$$

This function can transform a value into a probability from 0 to 1. The function is differentiable, but the gradients of the function are only significant in the value around zero, so bigger and smaller values will have very small gradients. The neural network stops learning as the gradient approaches zero and suffers from the vanishing gradient problem. For this reason, using this function as an activation function is not very common in modern feed-forward networks.

Hyperbolic tangent activation function (Tanh): Like the sigmoid, the Tanh function converts the input values to the range from -1 to 1 (Equation 2.5). However, the Tanh activation is zero-centered and might lead to faster convergence.

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.5}$$

Softmax: The softmax function is usually used as the function in the output layer but might also be used as the activation function in some hidden layers. The softmax function converts a vector of n real numbers into a probability distribution of n possible values. It is mathematically defined in Equation 2.6.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad for\ i = 1, 2, \ldots, n \tag{2.6}$$

Many other activation functions are possible but are used less frequently.

If an MLP or neural network does not include activation functions or all of them are linear, it will perform a linear transformation on the inputs using the weights and biases no matter the number of hidden layers. The reason is that the composition of two linear functions is linear. In the classic perceptron, the decision function is the step function that gives a binary output. Still, other activation functions can be used to output real values, for example, between 0 and 1 or between -1 and 1.

The three-layer MLP is a shallow neural network. The length of the network is called the depth and is the reason behind the term *deep learning*. When the MLP has four or more layers, it is a DNN. ANNs are composed of multiple processing layers, the input layer, multiple hidden layers, and the output layer, as shown in Figure 2.4. In each layer, there can be a different number of neurons. ANNs can be used as good function

approximators. The universal approximation theorem [49] states that regardless of what function we are trying to learn, there exists an MLP large enough able to represent that function. However, this does not mean success in learning that function, as the optimization might fail for different reasons.

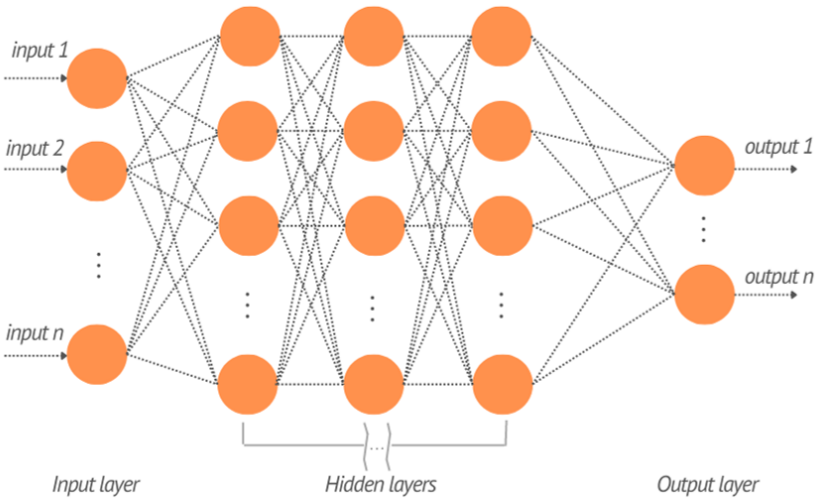


Figure 2.4: Artificial neural network architecture.

2.1.3 Deep Learning Era

As described in the previous section, DNNs and deep learning are not new. The research of ANNs has gone through different names, which reflect the influence of different researchers, perspectives, and advances. There have been three main waves of development of deep learning [50]: cybernetics (the 1940s–1960s), connectionism (the 1980s–1990s), and the current renaissance under the name deep learning, which started in 2006. The interest in ANNs in previous waves was lost or diminished because of different limitations, such as the constrained performance of the initial models, mathematical difficulties, or the lack of computational resources.

The current wave of neural networks began in 2006 with the work that showed how a kind of many-layered feed-forward network, called a deep belief network, could be effectively trained using a strategy called greedy layer-wise pretraining [51]. The revolution of deep learning exploded in 2012 when a convolutional neural network (a type of DNN) proposed by [27] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

[26] by a large margin over previous methods. The image classification state-of-the-art top-5 error decreased from 26.1% to 15.3%. In the following years, new neural networks continue beating the record. By 2015, the best model had better-than-human accuracy [48]. DNNs started outperforming competing machine learning systems based on other technologies in different competitions and impacting diverse fields in addition to computer vision, such as speech recognition [52] or robotics [17].

The main reasons behind the resurge and success of the DNNs are the large-scale data availability, the computational resources, and the improvements in the technique. The increasing digitization of society has made it easier to centralize data and generate appropriate datasets for machine learning applications. The age of Big Data has enabled it to generate datasets of millions of labeled samples [26, 41]. Another important fact is that today we have the computational resources to work with much larger models. Regarding the number of neurons of ANNs, it has been small until quite recently but has increased rapidly in the last few years. The increase in model size over time has been possible thanks to faster CPUs, GPUs, and better software infrastructure. Faster computers with larger memory and the availability of larger datasets drive the models' growth.

Currently, the wave of popularity of deep learning continues. Many technology companies such as Google, Microsoft, NVIDIA, or IBM use deep learning in their applications. Advances in deep learning also depend on the software libraries that support its deployment and research, such as Tensorflow [53] or Pytorch [54].

2.2 How a Deep Neural Network Learns

Deep learning is learning relations and patterns from data with DNNs. This learning process is called training. Training a DNN involves using an optimizer to find a set of network parameters that maximizes or minimizes a predefined objective. This process consists of having a training dataset, a designed DNN model, defining a cost function, and implementing an optimization process to estimate the model's optimal learnable parameters (weights).

We must know how close or far we are from that objective to maximize or minimize an objective. We measure how close or far we are with an objective or loss function. The loss measures the error incurred from incorrect predictions of the model compared to

Table 2.1: Common last-layer activation and loss functions for training DNNs for different tasks.

Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	binary cross-entropy
Multiclass, single-label classification	softmax	categorical cross-entropy
Multiclass, multilabel classification	sigmoid	binary cross-entropy
Regression to arbitrary values	none	mse
Regression to values between 0 and 1	sigmoid	mse

what we expect. For example, the loss function can be the mean squared error (mse) between the model's predictions when classifying some images and the output expected for those inputs (the ground truth, the real categories of those images).

The choice of the loss function is tightly coupled with the selection of the output unit of the model. To compute the difference between the predictions and the ground truth in the final layer of the DNN, the output signal must be transformed into an appropriate format (e.g., from arbitrary numbers to class probabilities). This transformation is done using an activation function, which depends on the target task and is also related to the loss function. Table 2.1 shows the most common last-layer activation and loss functions for different problems.

Sigmoid and softmax functions (Section 2.1) transform values to 0 and 1 considering two possible classes or n, respectively, so they are used to provide probabilistic outputs. Regarding the loss functions, binary cross-entropy compares the predicted probabilities to the expected output (0 or 1) and computes a penalization score based on the distance from the expected value. The binary cross-entropy loss L for a sample x is defined in Equation 2.7.

$$L(x) = -(y\log(p) + (1-y)\log(1-p)) \tag{2.7}$$

The ground truth label y is 0 or 1, while the predicted probability p for the specific category ranges from 0 to 1. The categorical cross-entropy is the extension of binary cross-entropy to multiple classes. Apart from the loss functions in Table 2.1, other possible loss functions exist depending on the problem type.

The loss function measures how good or bad the model performs, which is used in the optimization process to update the weights of the DNN towards a better set of values for solving the target task. One of the most common algorithms for solving the

optimization is stochastic gradient descent, where model weights are updated each iteration using the backpropagation of the model's error in the corresponding iteration. The backpropagation process [55] computes the gradient of an objective function with respect to the weights of the model using the chain rule for derivatives. The gradient or derivative of the objective for the input is computed by propagating the gradients through all layers, starting from the output of the network to the input layer. The loss for a batch of samples x^i is computed during the training for a set of weights θ in the called forward pass, then the gradients are computed in the backward pass, as shown in Figure 2.5. The derivatives with respect to the weights are used to update the weights.

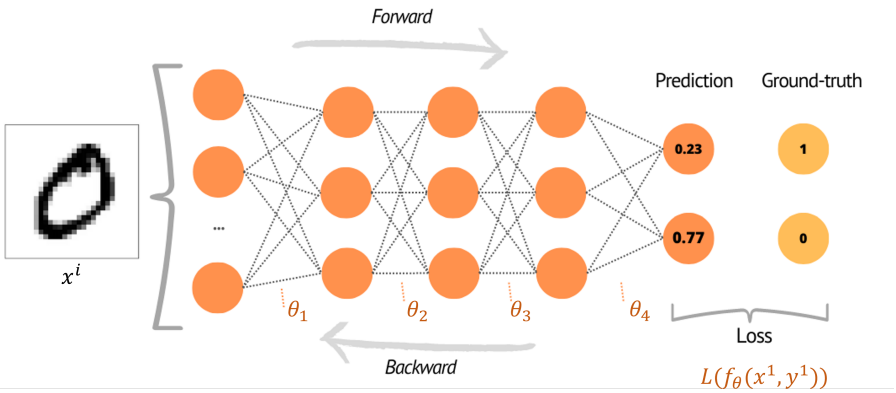


Figure 2.5: Training process of a DNN. In the forward pass the predictions of the model for a sample x^i and a set of weights θ are used to compute the loss. In the backward pass, the gradients of the loss with respect to the weights of the model are computed using the chain rule.

The weights for the next iteration w_{t+1} are updated based on the current values w_t and the computed gradients with respect to the weights to be updated (Equation 2.8).

$$w_{t+1} = w_t - \lambda \frac{\partial L}{\partial w_t} \tag{2.8}$$

Note that the optimization process involves defining additional hyperparameters, such as λ, which is the learning rate and defines how fast weights will be updated. There are other hyperparameters, such as the number of iterations to train the model or the batch size (number of samples processed in each forward pass before computing gradients). There also exist other gradient-based possible optimizers which present different properties (e.g., Adaptive Gradient Descent [56]).

Ideally, after we train the DNN, it must perform well on new previously unseen inputs, not just those included in the training dataset. The ability to perform well on those new inputs is called generalization. There are two possible reasons why the model converges but does not generalize. Overfitting happens when the gap between the training and test errors is too large because the model adjusts too much to the seen data. The model memorizes the training data, but it cannot do correct estimations on the new data. Underfitting happens when the model cannot get enough low error values, not even on the training set. The model can neither model the training data nor generalize to new data. Figure 2.6 shows how a function looks like when it suffers from underfitting (left), overfitting (middle), or has a good fitting to data (right).

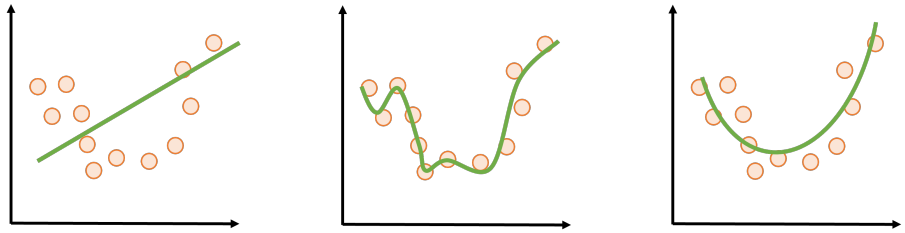


Figure 2.6: Model with underfitting to data (left), overfitting (middle), and a right fitting to data (right).

Different techniques can be used to handle these problems, for example, choosing a more complex model to reduce underfitting or using regularization during the training [57] to reduce overfitting. Reducing overfitting to the training data and ensuring the model can generalize is one of the key challenges of deploying a good machine-learning model. However, often the key is to have an appropriate dataset with enough samples, variability, and problem representation.

2.3 Types of Deep Neural Networks

There are different kinds of DNNs depending on their structure and the data flow. If the information flows forward from the input data to the output without feedback connections or loops, it is classified as a feed-forward network. Otherwise, they are called recurrent neural networks (RNNs). RNNs use sequential data as input or output and are widely used in fields such as natural language processing (NLP). Among feed-forward

networks, one of the most common types is the Convolutional Neural Network (CNN) [58], used across different applications and domains. CNNs are especially established in image and video processing tasks but are also well suited to other spatial or temporal signals, such as sounds. Other architectures, such as Transformers [59], are also gaining popularity in recent years. Even if the architectures of the possible DNNs are different, they share some common layer types.

Linear layers. The linear layers of the DNNs contain almost all the learnable parameters of the network. A basic linear operation in a DNN looks like Equation 2.9.

$$x_{out} = W \cdot x_{in} + b \tag{2.9}$$

where W and b are the learnable weights and biases, respectively.

Activation layers. Non-linear activation layers add nonlinearity to the DNNs. Commonly parameters of these layers are not learned, but they can be. Some common activation functions are sigmoid, ReLU, or leaky ReLU (Section 2.1.2).

Normalization layers. Normalization layers modify the neurons' values based on the collective behavior of a group of neurons. For example, batch normalization standardizes feature maps with respect to the mean and variance over a batch of samples [60]. Batch normalization can provide faster convergence of the DNNs.

Output layers. As explained in Section 2.2, output layers map a high-dimensional vector of values to the desired output representation (e.g., class probabilities).

Among DNN types, CNNs show state-of-the-art accuracy in many tasks for different fields [61, 62] and are the most used networks for visual perception tasks. Due to their relevance for perception systems, we introduce them next.

2.4 Convolutional Neural Networks

The data has a special structure in different domains, and specific architecture designs can help exploit that structure. CNNs are a neural network type designed for the structure in visual signals (e.g., images) or data with grid-like topology. Time-series data can also be treated as a 1D grid taking samples at specific time intervals. If there is a possibility to scan across the input signal, like a sliding window, then CNNs might be used. CNNs are neural networks that have at least one convolutional layer.

In the neural networks seen so far, each input is connected to each unit in the following layer. These types of layers are also known as fully connected layers. When using these layers, as the model size increases, the number of weights to learn increases quickly. In addition, the same features are learned for every position in an image. Convolutional layers are motivated by both issues and have the following properties.

Sparse connectivity. CNNs have sparse interactions or connectivity, which means that each neuron is not connected to all the units in the following layer. Considering that all the weights applied to a neuron's values are structured in a kernel, we obtain sparse connectivity using a kernel smaller than the input dimension. When the connectivity is not sparse (Figure 2.7, left), each unit interacts with all the units in the next layer. When the connections are sparse (Figure 2.7, right), only some outputs are affected by the input x, as units are locally connected. How many inputs affect a specific neuron depends on the kernel size and defines the receptive field of that neuron for the corresponding layer.

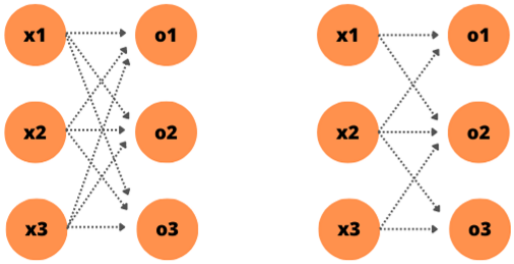
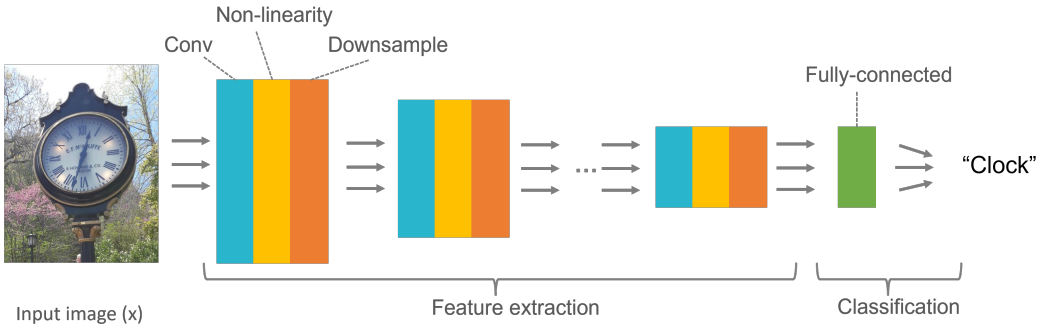


Figure 2.7: Difference in not sparse (left) and sparse (right) connectivity between neurons of consecutive layers.

Parameter sharing. In convolutional layers, each member of a learned kernel is used in every position of the input rather than learning a separate set of weights for every position. We can think of it as a kernel that works as a sliding window over the input data. This makes the model much more efficient regarding memory requirements and efficiency. In addition, the parameter sharing causes the operation to be equivariant to translation (not to scale or rotation). A function is equivariant if when the input changes, the output changes in the same way.

The typical structure of a basic CNN is composed of an initial stage with stacked convolution, non-linearity, and pooling layers, repeated n times, followed by some final fully-connected layers, as shown in Figure 2.8.



$$f(x) = f_L\left(f_{L-1}\left(\ldots f_2(f_1(x))\right)\right)$$

Figure 2.8: Basic CNN architecture scheme composed of convolution, non-linearity, pooling (downsample), and fully-connected layers for image classification.

Convolutional layer. The operation in a convolutional layer can be defined as a weighted sum of a kernel and a patch of the input data of the same size, as shown in Figure 2.9. Note that the input and output data are represented as matrices, which is closer to how the data are handled in practice, but they could also be visualized as in Figure 2.7 (each value would be a neuron unit).

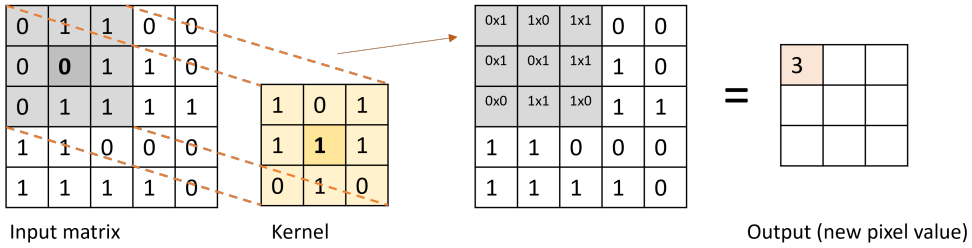


Figure 2.9: Example of the convolutional layer. The kernel is applied to the first patch of the input data to generate an output value. The operation will be repeated, sliding the kernel over the input data to get an output feature map.

Generally, convolutional layers apply a kernel group to a multi-channel input signal to produce a multi-channel output signal. This is mathematically defined in Equation 2.10.

$$x_{out_k} = \sum_{c=1}^{C} w_{k,c} \cdot x_{in_c} + b_k \tag{2.10}$$

where x_{out_k} is the output of channel k, which is called output feature map. x_{in_c} is

the channel c of the input data. The kernel bank comprises the multi-channel kernels (also called filters) $[w_1, \ldots, w_k]$. Each filter applies a convolution operation per channel c and sums the responses over the channels. The channels' number C when we have an RGB image is 3, but a different number if the input data are another type of data or feature maps from a previous layer.

The output dimensions depend on the kernels' size, the stride, and the padding. The padding adds extra pixels of filler around the boundary of the input image, increasing the effective size of the image and helping to preserve the input data's dimensionality or control the output dimensions. Regarding the stride, it defines the number of rows and columns traversed per slide when sliding a kernel over the input data. Using a stride bigger than one increases the computational efficiency and downsamples the data.

Note that the values of the kernels are the weights that should be learned during the model training phase. Depending on their values, different features will be extracted from the data. Figure 2.10 shows that, for example, depending on the values of the kernels, different edges, such as horizontal or vertical ones, can be extracted. The output images after applying the filters are the feature maps. During the training, the filter values are optimized for estimating the most helpful features for solving a predefined task.

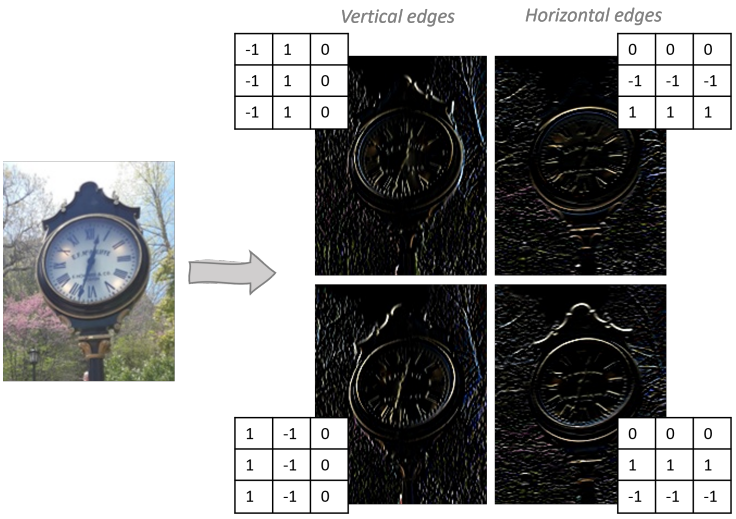


Figure 2.10: Example of different features extracted from an image when kernels with different values are applied. It can be seen that depending on the values, specific edges, such as horizontal or vertical, stand out.

Pooling layer. The pooling layer downsamples the data by dividing the data into patches of a predefined shape (kernel size) and summarizing the information in each patch using some statistics, such as the patch's mean value (mean pooling) or the maximum value (max pooling). Figure 2.11 shows this operation.

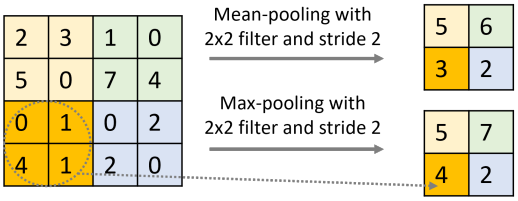


Figure 2.11: Example of mean-pooling and max-pooling operations applied to an input feature map.

The pooling layer reduces the input feature map's resolution and helps make the representation invariant to small translations of the input. If we translate the input data by a small amount, the values of most of the pooled outputs do not change.

Global pooling layers reduce to the extreme the input data by pooling over the entire dimensions of the feature maps. The global pooling reduces an input feature map with width W, height H, and C channels to a vector of length C.

Fully connected layer. In the fully connected layers, all the nodes from a layer are connected to all the units in the following layer, as seen previously for the MLP. These layers are generally at the end of the CNN to obtain the final outputs (e.g., image classification).

CNN architectures have evolved from the baseline CNN presented here to improve different aspects, such as higher accuracy or more efficiency [27, 28, 29]. Thanks to these advances, big progress has been made in many tasks in computer vision, such as face recognition [63] or pedestrian detection [64]. The progress in the last years, thanks to CNNs and other network architectures, provides an opportunity of developing a wide range of new applications. The DNN is the key element in many vision-based perception systems, but its development is not limited to the DNN architecture design. It entails different stages, where each consecutive stage affects the rest of the flow. Each stage involves different challenges and open questions crucial to the system's success.

If you only read the books that everyone else is reading, you can only think what everyone else is thinking.

Haruki Murakami

CHAPTER

3

Related Work

3.1 Overview

As explained in the previous chapter, deep learning models have made big progress in many areas, fueled by the advancement of DNN architectures, powerful computation, and access to big data. DNNs have been successfully applied to computer vision for tasks such as object detection, classification, and tracking, thanks to the development of CNNs [65, 66]. Training and testing a DNN involves a lifecycle, during which data are a key element of the process (Chapter 1). Data can be obtained using different sensor modalities and can include samples from the real world or synthetically generated samples. Both possibilities involve challenges and advantages, currently open research topics.

The sensor setups used for perception systems rely on different sensor modalities to capture the required signals from the environment. Each sensor and the data obtained from those signals have different properties that affect both the design of the sensor setup and the DNN (Section 1.3.2).

This chapter presents an in-detail study of the different sensor devices to capture information about an environment in vision-based perception systems. Additionally, we present the various sources for obtaining data for deep learning: data capture, large-scale datasets, and synthetic data generation.

3.2 Multimodal Data for Perception Systems

A machine learning-based system's final goal and requirements set the information that must be captured from the environment and the possible sensors that provide that. Most of the time, there is more than one possible sensor type for solving a task, but each has different properties, advantages, and drawbacks. The automotive field is a representative case to see how multimodal setups are used to solve different or the same tasks with various sensor types, making use of their advantages and properties to get a robust representation of the environment (Section 1.3.2).

3.2.1 Automotive Use Case

In the last decade, self-driving vehicle research has gained large attention. Advances in algorithms, along with improvements in sensor technology, are promoting the race to develop driverless vehicles. Efforts are not only focused on achieving total autonomy. Advanced Driver Assistance Systems (ADAS) have become part of most recent vehicles, such as automatic parking assistance or traffic sign recognition [18]. They promote a human-machine interaction environment that combines the complementary strengths of humans and machines to achieve higher performance than each. Big progress in machine learning has played a key role in the uprising of both paradigms, most thanks to high-level feature extraction based on deep learning, which has also attracted strong investments and efforts in the sector. However, the promise of safer, more efficient, and more comfortable vehicles has been in the mind of researchers for decades. By the 1960s, AI pioneers started imagining driverless cars, and many institutions worldwide initiated projects to develop Intelligent Transport Systems (ITS). The idea of using different sensor modalities for driving, as well as ANNs, was already in the mind of some scientists. In 1989, Pomerleau [67] proposed the first self-driving system based on a 3-layer ANN. The system was fed with images from a camera and a laser range and was able to estimate the direction the vehicle should follow. Autonomous vehicle development has continued over the years in an evolutionary rather than revolutionary manner. ADAS have become part of most recent vehicles, such as automatic parking assistance or traffic sign recognition. Backed and promoted by big advancements in

machine learning and technology development, the space for possible achievements has expanded.

Deep learning algorithms have quickly become a method of choice for most machine learning problems. Deep learning has been widely applied to the field of driving in recent years, frequently to process data coming from a single sensor modality, most of the times cameras [68], and in some cases to process data from various modalities, for example, RGB and thermal cameras [69]. Even if some works have been recently developed for the last case, there is still a lot of uncertainty on how to process and combine data from heterogeneous sensors in the best way.

3.2.2 Sensor Modalities for Driving

Intelligent vehicles understand their surrounding based on the information fed to their computer system by processing signals from their external sensors. Various technologies are needed to have enough data to detect, predict and react to the surrounding environment factors, such as other road users. The range of possible sensors is wide, as shown in Figure 3.1, and each one presents specific strengths and limitations. Nevertheless, the combination of sensor modalities provides loads of both complementary and redundant data, which favors their use in safety-critical environments. The most common sensor types are depicted in Figure 3.1 and described in the sequel.

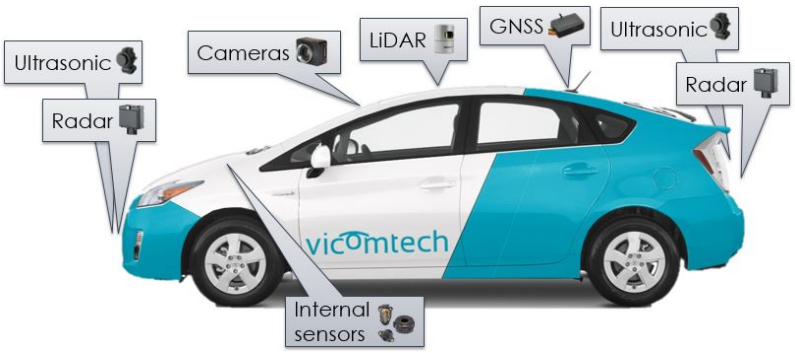


Figure 3.1: Frequent sensors self-driving research vehicles are equipped with.

Light Detection And Ranging (LiDAR): This laser-based system measures distances to surrounding objects sending out high-speed pulses of laser light and calculating the reflection time of the beams. The collision location between an object and the laser

beam is represented as a point in a 3D point cloud. There are two main trends in models: systems that use a rotating laser, or solid-state LiDARs, which have no moving parts.

They cover high distances accurately, usually more than 100 meters up to 200 meters range. It is not affected by different lighting conditions. However, it is not able to perceive color or textures. It can provide noisy measurements when suspended particles, such as rain, fog, or snow, are present in the air. LiDARs allow an accurate 3D analysis and are mainly used for mapping, obstacle avoidance, free space detection on the road, and localization [70, 71].

Vision Cameras: Multiple cameras are often installed in vehicles to have a detailed sight of the environment, covering the front or back view or even 360° around the vehicle.

They cover a medium distance. Cameras preserve detailed semantic information about the surrounding, making it possible to interpret objects like traffic signs. They are sensitive to lighting conditions, do not work well at nighttime or in dazzling sunlight, and often acquire poor-quality images under unfavorable weather conditions. Cameras need to be calibrated to address 3D measurements. They are mainly used for object detection, road guidance, and park assistance [68, 72]. Internal cameras are also common for driver monitoring.

Thermal Cameras: Thermal cameras detect the heat from pedestrians, animals, and objects and, consequently, the differences in temperatures emitted by living and inanimate objects. Although this sensor has been widely applied to different fields for years, its use is not so widespread for self-driving research.

They cover a medium distance. These cameras are advantageous to help edge cases, where some sensors might have difficulties, for example, differentiating between images of humans and real ones, as well as poor lighting scenes when vision cameras have problems. They are mainly used for object detection [69].

Ultrasonic Sensors: Ultrasonic sensors send out sound waves at a high frequency imperceptible by the human ear and measure the time it takes for the signal to return. This way, the distance to an object can be calculated.

They cover short distances. Due to the sensing fundamentals, air temperature, humidity, and wind can affect the sensor's accuracy, as they affect the speed of sound in the air. They are employed for short-distance applications at low speeds, such as park assistance or close obstacle and blind spot detection [73].

Radio Detection And Ranging (Radar): This sensor emits radio waves which are reflected when they hit an obstacle, revealing the distance to the object and how fast it is approaching. Radar can be categorized based on different operating distance ranges, starting from 0.2m to more than 200m, in Short Range Radar (SRR), Medium Range Radar (MRR), and Long Range Radar (LRR).

They are affected much less than other sensors by weather conditions, such as rain, fog, dust, and snow. Nonetheless, they can be confused by small very reflective metal objects. They do not provide any information about detected object types. They are usually used for very close obstacle avoidance [74].

Global Navigation Satellite System (GNSS): It is a global localization system that triangulates multi-constellation satellite signals to calculate the 3D position of the receiver (its latitude, longitude, and altitude). Currently, the considered GNSS providers are GPS, GLONASS, and Galileo.

The absolute position provided by this technology is affected by several error sources, such as lonosphere, multipath effect, or urban canyons. Therefore it is not enough to achieve lane-level accuracy. The position is usually enhanced by either using Differential GPS or fusing its information with inertial sensors like Inertial Measurement Units (IMUs) and accelerometers. It is used for localizing the ego vehicle and path planning [75].

In addition to external sensors, internal vehicle parameters also provide a very relevant information source for driving. These signals are available through the vehicle's Controller Area Network (CAN) bus and include parameters such as wheel speed, acceleration, steering, and powertrain values.

In the last years, vehicle communication with the cloud has been included in the intelligent driving scenario due to the possibility of sharing real-time map data and anticipating different situations. Information exchange with other vehicles or infrastructures is also considered in cooperative systems [76] through the use of vehicle-to-vehicle (V2V) and infrastructure-to-vehicle (I2V) communication.

Signal capturing and processing from the aforementioned sensor modalities is a complex task, which requires addressing various aspects such as sensor capturing synchronization. Some libraries and tools exist that help in this procedure, mainly in capturing, recording, and managing sensor timestamps or integrating developed approaches. Among the most frequent ones, we find RTMaps [77], ADTF [78], and ROS [79].

At the same time, some public datasets provide already captured multisensory data that is ready to be used by researchers who want to start deploying their approaches.

3.3 Large-scale Datasets

Big amounts of data have been crucial for DNNs' success. Large datasets have played an important role in the advances and development of DNNs. Annotated data are indispensable not only for developing and training deep learning models but also for generating a quantitative evaluation of them. However, collecting large amounts of annotated data with quality is tedious and complex work, and it is often beyond the reach of researchers. In an effort to alleviate these needs, some large datasets have been made public.

ImageNet [26] is one of the most popular datasets for deep learning, with more than 14 million hand-annotated images. The ILSVRC uses a subset of the ImageNet dataset and is the most popular benchmark for image classification algorithms. MS COCO [80] is another large-scale generalist dataset focused on object detection, segmentation, and captioning of objects in a natural context. OpenImages [41] contains 9 million annotated images with a high variety of classes (more than 6000).

Apart from generalist datasets, there are also large-scale datasets focused on specific tasks. For example, for people detection, there are different available annotated datasets (e.g., [81, 82, 83]). Most of these datasets provide images from a vehicle perspective [82] or tilted surveillance cameras [83]. However, when the application setup differs remarkably, these data might not be useful. For example, frequent surveillance setups include omnidirectional cameras, which provide a top-view perspective of people. Data from omnidirectional cameras are scarce, and datasets for training people detection CNNs from this perspective are sparsely available. Recently, some studies have opened their datasets of fish eye images for people detection [84], but they only cover setups with cameras installed at a very low height. If we would like to train a DNN for this task, for example, to monitor people flow in big infrastructures such as airports, we would face a data lack problem.

Other large-scale datasets are focused on specific fields, such as automotive. In this case, the value of the datasets is not only related to the data quantity or quality and the

Table 3.1: Sensor modalities in large-scale automotive datasets.

Dataset	Relevant information	Vision Camera	Thermal Camera	LiDAR	GNSS/ IMU	Radar	Internal parameters
KITTI [85]	6 hours of recordings, multi task annotations	✓		✓	✓		
CityScapes [86]	segmentation benchmarks, coarse and accurate labels	✓					
TorontoCity [87]	multitask annotations. various perspectives	✓		✓			
Paris-Lille [88]	point cloud segmentation and classification			✓	✓		
RobotCar [89]	recorded in Oxford through a year	✓		✓	✓		
Comma.ai [90]	11 videos, mostly on highway	✓			✓		✓
BDDV [91]	400 hours of HD video, multitask annotations	✓			✓		✓
Mapillary [92]	segmentation annotations, 66 classes	✓					
KAIST [93]	multispectral, bounding box annotations	✓	✓	✓	✓		
NuScenes [94]	3D bounding boxes (360° coverage)	✓		✓	✓	✓	

tasks and scenarios they include but also the sensor modalities they consider. It is common to use multimodal models for driving applications, which presents an additional difficulty for data gathering. Many of the open datasets are focused on solving a specific problem and do not include data from all desired sensor modalities. Table 3.1 summarizes the most recent and relevant real-world datasets for the driving context within the sensor modalities they include and the most relevant information. None of the datasets contains data from the ultrasonic sensor.

However, generalist datasets do not cover all possible computer vision tasks for all possible use cases. Smart video surveillance systems, for example, may need to cover specific use cases or camera perspectives that are not available in any open dataset. Training techniques that rely on pretrained DNN models, such as transfer learning, fine-tuning, or BiT [95], can be used to benefit from models already trained with generalist datasets and avoid collecting and annotating big amounts of data. Moreover, works like [96, 97, 98] suggest that pretraining on domain-specific datasets improves accuracy compared to generalist datasets.

Other techniques, such as few-shot learning [99], try to train DNNs using a limited number of samples per target category. However, this does not avoid the need for data specific to the target task, which could be recorded by replicating the sensor setup

and the target scenarios. In these situations, usually, it is common to capture redundant samples, for example, because the captured scene remains the same for some time. Data with the same content add less additional information than data with new content, but beyond this, they can be problematic during the DNN training. The redundant data can lead to model overfitting during the training process. Data need to be representative.

3.4 Synthetic Data for DNN Training

Capturing data typically involves collecting and manually annotating a large amount of data for supervised learning. DNNs have achieved excellent results using large-scale supervised learning approaches in which a large amount of labeled data sets are usually required for training [28, 100]. Typically, the more data we have available, the better is their performance. In this context, collecting such an amount of data can be challenging specifically for two principal reasons.

First, we face compliance with privacy-related regulations in some countries, such as the 'General Data Protection Regulation' (GDPR) of the European Union. The data collection process should put in place appropriate technical and organizational measures to implement the data protection principles, such as data protection or data anonymization.

Second, the time and cost that we need to devote to the data collection process, considering that not only the quantity is important but also the variety and balance to cover all possibilities during training appropriately. In most cases, the data cannot be directly extracted from the Internet since we are trying to solve a particular problem in a specific scenario with a specific camera setup. Thus, custom data sets need to be created. This implies the execution of several actions such as data set specification, camera setup installation, technical preparation, recording protocols, and even actors simulating precise guide notes. All these actions, together with the well-known problem of labeling data [101] due to the lack of effective tools and/or the annotation complexity, make the data set generation process more challenging.

In addition, some applications may involve capturing data in dangerous situations to develop intelligent systems, for example, in the automotive industry.

Even if all these difficulties are overcome, the need for labeled data involves an expensive and time-consuming annotation process, which can make the dataset-gathering process unaffordable. For all these reasons, synthetic data generation has gained increasing popularity for training DNNs [102, 103, 104].

3.4.1 Synthetic Public Datasets

Currently, there exist different public synthetic datasets available to train DNNs. For example, the Virtual KITTI dataset [102] includes different environmental conditions, camera position, and instance-level segmentation ground truth for analyzing multi-object tracking in driving scenes. The SUNCG [105] and the Matterport datasets [106] are 3D model repositories for indoor scenes in perspective views focusing on depth, physical-based rendering, and volumetric ground truth. The Synthia dataset [107], based on a virtual city, includes automatically extracted pixel-level and instance-level semantic annotations in both videos and independent snapshots. The THEODORE dataset [6] is a large-scale top-view indoor dataset containing 100,000 fish eye images with 16 classes, including people. Like real datasets, it is limited to small 3D virtual environments (living rooms) and with a low-camera setup. The distortion of the images and the number of people differs notably from large spaces because of the camera position and the small region captured. As with the datasets containing real data, the mentioned datasets are limited to specific use cases and tasks. Developing a model for a new custom application is prone to require generating new data.

3.4.2 Synthetic Data Generation Techniques

The techniques used to generate the previously mentioned datasets are widespread in the literature. As stated in [42], synthetic data can be generated following different strategies, such as compositing real data, relying on generative models, or using simulated environments. Some techniques are oriented to the specific network that will make use of them. These methods are task-aware since they generate hard examples that help improve the target network performance. One of the main techniques in this class is image composition. Some cut-and-paste approaches are those in [108] to synthesize positive examples for object detection tasks. The advantage of these approaches comes

from generating novel and diverse juxtapositions of foregrounds and backgrounds that substantially increase the available training data. The approach in [108] was the first to demonstrate boosts in performance through the cut-and-paste procedure.

In some other techniques, synthetic data generation is decoupled from training the target model. They range from photo-realistic image rendering [109] and learning-based image synthesis [110] to methods for data augmentation that automate the process of generating new example images from an existing training set [111]. Traditional approaches to data augmentation have exploited image transformations that preserve class labels [112] while recent studies [111] use a more general set of image transformations, even including compositing images.

Simulated 3D environments. The recent survey on synthetic data for deep learning [42] shows that in the last years there has been a shift from static synthetic datasets to interactive simulation environments, grouped in the following categories: (1) outdoor urban environments for learning to drive, (2) indoor environments and (3) robotic and aerial navigation simulators. Current state-of-the-art environments have been built upon *Grand Theft Auto V (GTA V)* [4], *Unity3D* [6, 113], *Unreal Engine* [5, 114], *CityEngine* [115] or *Blender* [116], among others. Some of these simulators are shown in Figure 3.2.

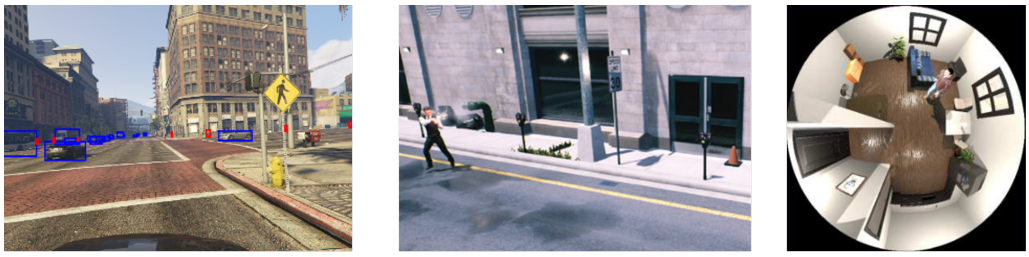


Figure 3.2: Different simulation 3D environments. From left to right: PreSIL [4], Vivid[5], THEODORE [6].

GTA V is an action-adventure video game with realistic graphics of a large detailed city and surrounding areas from which diverse data can be extracted, involving virtual people, animals, cars, trucks, motorbikes, planes, etc. As stated in [113], the main drawback of video-game-based environments is the limited freedom for customization and control over the scenes to be captured, which makes obtaining a large diversity and

good balance of classes difficult, due to the rather complicated procedure required to obtain ground-truth instance-level annotations.

On the contrary, [113] claims that their environment, built upon the game engine *Unity3D*, can be set up by one person in one day. This is very little effort, considering that it allows access to a virtually unlimited number of annotated images with the object classes of state-of-the-art real urban scene datasets. It captures synthetic images and instance-level semantic segmentation maps at the same time and in real time, with no human intervention. While generating the data, it renders the original textures and shaders of included 3D objects and other automatically created unique ones for their corresponding instances. *Unity3D* is also the basis of the indoor environment proposed by [6] for generating synthetic data for object detection from an omnidirectional camera placed on the ceiling of a room. The 3D assets are generated using the skinned multi-person linear model proposed in [117]. This work points out that *Unity3D* only provides a camera model for perspective and orthographic projection. They overcome this limitation by combining four perspective cameras following the procedure proposed in [118].

[5] proposed a universal dataset and simulator of outdoor scenes such as pedestrian detection, patrolling drones, forest fires, shooting, and more. It is powered by the *Unreal Engine* and leverages the *AirSim* [114] plugin for hardware simulation. The TCP/IP protocol is used to communicate with external deep learning libraries. As it is focused on training dynamic systems relying on deep reinforcement learning [119], it prioritizes the real-time interactivity of the virtual agents over photorealistic rendering. This differs from static systems for surveillance applications, like those that motivate this work, in which, at least for the data generation process, achieving a better rendering quality is more important than the real-time interactivity during training.

CityEngine is a program that allows the generation of 3D city-scale maps procedurally from a set of grammar rules. It is used in [115] as part of their method to generate an arbitrarily large, semantic segmentation dataset reflecting real-world features, including varying depth, occlusion, rain, cloud, and puddle levels while minimizing required effort.

Blender is an open-source 3D graphics software with Python APIs that facilitate loading 3D models and automating scene rendering. It was used in [116] to generate a dataset to train a DNN-based detector for recognizing objects inside a refrigerator. It

used *Cycles Render Engine* available with *Blender*. This engine is a physically-based path tracer that allows getting photorealistic results, which is beneficial for avoiding a big domain gap between the feature distribution of synthetic and real data domains.

In self-driving and ADAS, simulation has an additional application besides getting data to train models. Developed approaches must be evaluated during a vast number of kilometers and varying conditions to ensure and demonstrate they are safe, which could be impractical in its entirety. In addition, dangerous or uncommon driving situations must be evaluated. As a solution, various simulation environments have been released, such as AirSim [120] or Carla [121]. They facilitate the user collecting data from different sensors or integrating developed approaches to be tested.

There are no generalist simulation environments ready to solve any kind of scenario or application, so it is common to develop custom environments for specific problems in which obtaining data is challenging. For example, realistic eye image synthesis has been extensively studied because of its importance in various applications and the difficulty of getting annotated data. Graphics-based methods are used to produce high-resolution images based on a 3D model of the human eye region, which is used in a rendering framework [122]. However, the use of artificial eye texture can make the generated images appear unrealistic and widen the discrepancies between generated and target data. [123] combines real images and 3D eye region modeling, which is fitted and rendered for a specific gaze direction.

Generative adversarial networks. In recent years, generative models have made big progress in synthetic data generation. Recent advances in diffusion models are gaining attention, and we are still about to see their possibilities [124]. Other models, such as generative adversarial networks (GANs) [125], have seen significant advances in recent years (e.g., improved architectures and evaluation metrics).

GANs are a type of deep generative model used for data generation and are commonly composed of two main components: a generator and a discriminator. The generator creates synthetic samples, and the discriminator figures out if they are real or fake. The generator and the discriminator are trained against each other in an adversarial process until the generator is able to generate synthetic samples that are indistinguishable from real data. The state-of-the-art GANs, such as BigGAN [126], and StyleGAN3 [127], can generate very realistic images with no need for a 3D simulation environment.

GANs might be used to generate new unseen data or refine or edit already generated data for a specific task. The following works are related to the use of GANs in the context of data generation for gaze estimation. Some works propose initially generating data with 3D simulation environments and then mitigating the domain discrepancies between real and generated data by improving synthetic data realism using GANs. In these works, GANs are used to refine synthetic samples to look more realistic before using them for DNN training. SimGAN [128] uses an adversarial network to refine synthetic data so that they look more similar to target unlabeled real data. They introduce a penalty for the pixel-level deviation between simulated images and their refined versions to preserve the gaze direction, but this limits the diversity of the resulting images. GazeGAN [129] treats the problem as unpaired image-to-image translation. In this approach, an image from one domain is converted to the other and back to the original, controlled using a cycle consistency loss [130]. These approaches involve generating synthetic data from 3D models, which can be time-consuming due to the need for 3D asset design and rendering. Furthermore, these approaches require an additional refinement step that is restricted to a cropped region around one of the eyes.

Given the existing but limited and hard-to-obtain gaze data, GANs might be convenient to augment them. Many works have attempted to use GANs to achieve controllable image generation and editing .This can be tackled by learning a model to manipulate the latent space of a pre-trained GAN model [131, 132] or training a GAN with additional supervision so that a more disentangled latent space is learned [7, 133]. These works are often applied to facial data and focus on editing face attributes such as hair or skin. Most of the generated images tend to look forward as it is the most common gaze direction in typical GAN training datasets [134, 135]. GANs also have limitations, based on what the GAN has seen during the training [136]. Consequently, the training data domain influences the capabilities of the trained model to generate varied images.

Specific GAN studies focus on gaze correction or redirection. In [137], a cycle consistency and perceptual loss are introduced to redirect the gaze of input eye crops. InterpGaze [138] redirects the eye gaze given two gaze images to generate intermediate results using an encoder, a controller, and a decoder. Some works additionally learn to blend the eye crop into the face, such as DeepWarp [139] and GazeGAN [140] but are constrained to gaze correction or limited eye movements.

3.5 Deep Neural Network Design based on the Data Source and Modality

For training a DNN, data might come from different sensors and might be captured in a real scenario or artificially generated. The source of the data implies having specific data properties. Often, it is possible to select a generalist DNN architecture (e.g., EfficientNet [29]) and train it without further consideration of the data. In some cases, adapting the DNN to the data specifications is necessary and might improve the model's accuracy. The following sections describe different proposals for adapting the DNNs to multimodal setups or synthetic data.

3.5.1 Deep Neural Networks for Multimodal Data

Multimodal deep learning is about learning features over multiple modalities. In this section, multisensory deep learning, popularly known as multimodal deep learning, is presented for the automotive field, which, as mentioned in previous sections, is a representative use case of setups composed of different sensor types for perception. There are two major paradigms for automated driving: mediated perception approaches and behavior reflex approaches.

Behavior reflex approaches build a direct mapping from sensory input to a driving reaction, such as turning left or braking. Here belong the so-called end-to-end driving approaches.

In 2016, [141] trained a CNN to map images directly from a single frontal-facing camera to steering commands. 72 hours of driving were collected to train the model. In [142], Xu et al. propose an FCN-LSTM architecture trained with large-scale crowd-sourced data [91]. The input images are processed by a dilated FCN [143], and the segmented images are concatenated with previous sensor information, such as speed and angular velocity, to feed an LSTM and predict driver actions.

Point cloud data is also present in some multimodal approaches. The method proposed in [144] fuses the depth and vision from LiDAR and camera to predict steering commands. Features from RGB images and depth range images are extracted independently through a series of convolutional operations, then combined to be transferred to fully connected layers and finally predict the commands. The network was trained

with some corrupted samples from one of the sensors to achieve robustness in case of sensor failure. [145] also considers the handling of partial failure in the sensor set and proposes a solution by introducing Sensor Dropout, which randomly drops sensor modules in the training stage. They consider a Deep Reinforcement Learning (DRL) setup for their tests.

Mediated perception approaches [146] rely on the decomposition of autonomous driving into multiple sub-components that are combined to obtain a comprehensive understanding of the vehicle surroundings. This perception information is often used to feed on-board world models such as Local Dynamic Maps (LDM) [76]. Various tasks are included in which different sensor types are present, combined or alone. In this diversity, vision-based systems are maturing within their limitations of the data type [147, 148], while models that incorporate other data modalities are still emerging with no standardized methodologies to follow.

For road segmentation, [71] trained a model that fuses camera and LiDAR data through a hybrid Conditional Random Field (CRF). In the field of path prediction, [75] proposes an LSTM architecture that estimates the future position of obstacles given a sequence of their past trajectory data obtained from sensors like LiDAR and GPS. In [69], multispectral pedestrian detection is proposed using a CNN that fuses color and thermal images. They also explore results obtained by early, halfway, and late fusion of images in the architecture.

Among the different tasks, object detection plays a key role in mediated perception approaches. It has advanced notably in the domain of 2D in the last years, but self-driving vehicles also need 3D information. For this task, advantage of various sensor modalities and their strengths can be taken.

3.5.1.1 3D Object Detection

When 3D object detection using DNNs and LiDAR started gaining attention, most focus was still on 2D object detection for image data. Consequently, many methods converted point clouds to image-like data so that state-of-the-art detectors could be used.

2D object detection DNN architectures are often divided into two main categories. The first is a single-stage approach like YOLO [149], where anchor boxes are directly related to output boxes. Two-stage methods like Faster R-CNN [150] first try to identify

regions of interest (ROIs) that are likely to hold an object and then, in the second stage, fit the bounding more tightly around the object and decide to which class the object belongs to. Generally, the single-stage models are faster, while the two-stage methods are more accurate. Both categories are the base of many 3D object detection works [8, 151].

Point cloud to image conversion is required for applying one of the aforementioned models to point clouds. In the literature, we find two main ways to convert from point cloud to image. The first way is using a front view image in a polar coordinate representation, which is the native representation of the LiDAR sensor. 3D points from the LiDAR data are projected on a depth map to create a front view image [152]. The images produced are similar to the images from a camera. These images have a dense pixel representation. Consequently, there may be occlusions between the objects in the image, and the objects' size is related to the distance to the sensor. Different works use the front view as the only or as a complementary representation [151].

The Bird's Eye View (BEV) representation is the second way to represent the point cloud in an image-like representation [153]. The BEV image represents the point cloud from a top-view perspective. The point cloud is projected to a BEV representation (Figure 3.3), and then it can be used as input to a mature 2D CNN. Features that are encoded in the BEV map vary but often include the height of points, reflectance intensity, or the density of points [45, 151, 153]. When storing this information as 3 channels, a network architecture for RGB images is directly applicable. Some works store points' height information in more than 3 channels to retain more information [151, 153], but the architectures they use do not vary much from networks designed for image processing. With the BEV-based approach, there might be a significant compression in the height dimension. This could be a challenge for classes that are difficult to see from a top-view perspective (e.g., pedestrians) because most of their information is in the height dimension. However, thanks to the top-view perspective, objects are clearly separated, and there are no occlusions between them. Moreover, all objects of a specific class are the same size, independent of the distance to the sensor, as the object dimensions are proportional to the real ones.

The BEV representation has been the most used pre-processing step for 3D object detection from the LiDAR data and is used by many methods [8, 153]. However, most of them have not paid much attention to the different properties of LiDAR data compared to image data. LiDAR point clouds are sparse and irregular and contain

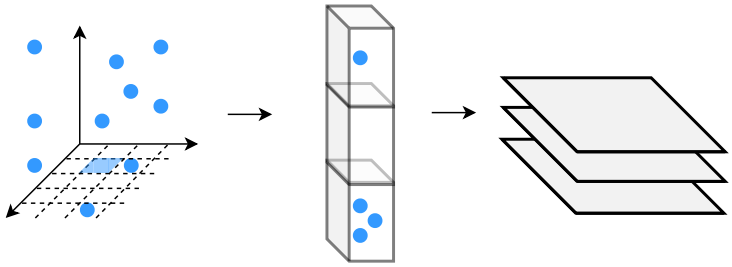


Figure 3.3: Point cloud to BEV conversion. The point cloud is discretized in a grid of columns from a top-view perspective and for every cell, the selected features are encoded in an image-like data format.

distance-dependent features. Not adapting the network designs properly to the processed data nature may be one of the reasons for these methods to be lately surpassed by works processing directly the raw point cloud.

DNNs are not limited to frontal or BEV representations and some methods combine the benefits of both. Some of these networks do not only rely on LiDAR, but they also add front camera data [154].

Multimodal DNNs propose fusing both images and point clouds to achieve better results. In MV3D [151], Chen et al. propose an object detection model that fuses data from images and LiDAR point clouds, which they project on a bird's eye view and a frontal view. They extend the image-based Region Proposal Network (RPN) of Faster R-CNN [150] to 3D so that 3D proposals are created based on the bird's eye view. The features of these candidates in all views are then combined through a deep fusion to produce the final result. AVOD [155] also feeds an RPN with the extracted features, but in this case, not only from the bird's eye view but also from the image, so that candidates are generated and transferred to a second detection network, which estimates the bounding box orientation, refines dimensions and classifies them. Wang et al. [156] also focus on an image and point cloud-based feature fusion before the region proposal stage through a new non-homogeneous pooling layer.

Point-based architectures have been introduced in recent years, which avoid converting point clouds to images to generate a representation based on engineered features. Some recent architectures for 3D object detection propose learning on the point cloud data directly without the conversion to image-like data. One of the first methods that applied this idea is VoxelNet [70]. They use a voxel feature encoding (VFE) layer to learn

features from all the raw points in a particular voxel. Then, the output is connected to an RPN. PointNet [157] goes a step further and can directly consume unstructured point clouds by processing points without any voxelization or pre-processing. Although the original paper is not applied to the driving field, it is the foundation for many other methods [155, 158, 159, 160]. For example, PointPillars [159] extracts point features with PointNet and then transforms them to a BEV representation to apply an FPN-inspired backbone [161]. PointRCNN [160] extracts point-wise features for 3D proposals generation, which are later on refined by a second stage. Due to the improved accuracy of these methods in benchmarks such as KITTI, little attention is paid now to the development of better BEV-based methods. Most state-of-the-art networks are based on a VoxelNet or PointNet-like approach. Handling the point cloud data directly involves some extra challenges, such as the volume of data to be processed in real-time or the unstructured form of the data. BEV-based methods are left behind without a clear answer to the following questions: is it better to process point clouds directly? Is it possible to achieve similar results with BEV-based methods?

3.5.2 **Deep Neural Networks for Synthetic Data**

The data discrepancies that data from different domains present is a challenge for training DNNs. Training a model with data from a source domain and then using it with data from another domain often leads to poor accuracy. This problem is especially noticeable when training a model with synthetic data and testing it with real data, known as the domain gap. The domain gap happens when a model is trained on a source distribution and then used in the context of a different (but related) target distribution.

Domain adaptation tries to minimize the domain gap to avoid an accuracy drop in the final model performance. Domain adaptation can be defined as a type of transfer learning where there are two domains with different data distributions. This problem has been researched in computer vision using various strategies to reduce or deal with the difference between the two domains.

The most basic method is to train on the synthetic domain first and then fine-tune the model on the real domain [103]. An alternative method is to jointly train with both domains' images using mini-batches from source and target domains [79, 162]. These

methods require having an adequate number of annotated real samples for optimal accuracy.

Other works apply the domain randomization technique to reduce the domain gap's effects when training [163]. During the data simulation process, they use random parameters (e.g., random light, objects' pose, or texture) to generate non-realistic images, which force the model to learn the essential features of the objects of interest. Some other works apply image augmentations to the generated samples instead of focusing on the 3D scene generation and rendering step. [164] generate new augmented synthetic samples using a semi-supervised errors-guide method to improve the DNN accuracy on cross-domain datasets.

Another way to solve the domain differences is the image-to-image translation method. Several works have emerged about GANs, which try to convert an image from one domain to another different one [130]. These methods are making remarkable advances, but they tend to have some difficulties accurately preserving the image's structure.

Rather than forcing the synthetic samples to look more like the real ones, another research line focuses on extracting domain-invariant features [165] by modifying the DNN architecture. These methods investigate how DNNs can better be trained with different domains' data. These approaches commonly consist of two main steps: (i) learn features that minimize the target task loss and (ii) make the features from both domains as indistinguishable as possible to make models trained in one domain work correctly in a target domain. For learning cross-domain features, Domain Adversarial Neural Networks (DANNs) include an adversarial domain classifier, whose loss is maximized using a gradient reversal layer [166]. The idea is that the DANN is not able to correctly classify the domain of the input images because it extracts domain-agnostic features.

These studies focus on the technical aspect of domain adaptation and lack an analysis of the effort of adapting the data or the training versus creating more adequate data in terms of overall performance.

3.6 Beyond the State of the Art

In this chapter, we have gone through the different sensing devices that can be used to perceive an environment (e.g., LiDAR, camera) and the different ways or sources to

obtain the corresponding data to train a DNN.

The sensor types used in a vision-based perception system have specific advantages and drawbacks, such as the robustness to different weather conditions or providing rich information about object textures around the capturing sensor. The data type a sensor provides also influences the DNN architecture design. The most common data modality is images, but different works have emerged to process other modalities, such as point clouds, or to combine modalities. However, the best way to process different kinds of data or adapt more mature models (e.g., image-oriented DNNs) to other modalities is an open question. Apart from the data modality, the data source significantly impacts the final DNN accuracy.

There are various large-scale public datasets, corresponding to single-sensor or multimodal setups, that alleviate the process of gathering data, but they frequently do not cover the task at hand or are limited. Synthetic data is a promising alternative, but their generation is not trivial. Some open simulated 3D environments might be used for data generation. Still, as no generalist simulation environment exists, it is common to build an 'ad hoc' environments when working on a new model. Using state-of-the-art generative models to generate synthetic data does not require a 3D environment and can generate realistic images. However, current works for generating data for a task such as gaze estimation are very limited to be used for training data. The way synthetic data are generated might also significantly impact the final model's accuracy due to the domain gap that is generated when source and target domains are different. Some works train the model directly, combining real and synthetic samples, while others explore domain adaptation techniques. Little attention is paid to the generation process itself.

Given the limitations and open questions in the current state of the art, the following chapters present our research results in the context of the hypotheses formulated in Chapter 1.

Part III

Research Results

Creativity is seeing what others see and thinking what no one else ever thought.

Albert Einstein

CHAPTER

4

Synthetic Data Generation for Deep Learning

In order to validate Hypothesis 1 (Chapter 1), in this chapter, we explore the 3D graphics-based synthetic data generation process applied to solve a real-world problem that lacks available training data: people detection from omnidirectional cameras in large spaces. This task has been widely investigated for typical surveillance cameras with little or no distortion. However, omnidirectional cameras have a significant advantage in monitoring large areas, such as train stations or airports. An omnidirectional camera has a 360º-field-of-view in the horizontal plane or covers a hemisphere or almost the entire sphere [167]. Thanks to their field of view, a large area can be visualized by a single sensor mounted at a particular height. We have not found any available dataset, which takes advantage of this property and that could be used to train a people detection model from omnidirectional cameras for large spaces. This chapter addresses the process of generating a suitable synthetic dataset to meet this need and train a state-of-the-art person detection DNN. We create a specific top-view dataset of people in different environments, mainly large infrastructures such as airports or stations. In addition, we evaluate the impact of different design choices, such as using different lighting effects or generating photo-realistic scenes, on the final model accuracy.

We have published the related research at Computers & Electrical Engineering Journal [162].

4.1 Methodology

To understand which attributes of an image are most effective for training DNNs, we follow the next steps, which are shown in Figure 4.1:

1. Study the different design possibilities that can be adopted when generating a synthetic dataset for people detection from omnidirectional cameras (See Section 4.1.1): analyze the distortion of the capturing sensor; apply a correction algorithm if needed; select scene environment design technique; select lighting and materials approach; include avatars and assets; model camera positioning; combine the different design possibilities defined in Step 1 for generating datasets with different features and degrees of complexity (Section 4.1.2).

2. Train an object detection CNN for people detection from omnidirectional cameras with each generated dataset (See Section 4.1.3).

Finally, in order to evaluate the domain gap of a training dataset, the trained models' accuracy is evaluated using a real-world dataset (Section 4.2).

In this way, estimating which variations applied to the image produce higher detection performance is possible. The generated scenes are mostly large spaces for common use, such as stations and airports, where people detection systems are usually deployed. They include variations in the distribution of materials and furnishings, but patterns of social behavior are always repeated.

We use 3D modeling software that can develop a mathematical representation of objects' three-dimensional surfaces to create scene images. Some of the most widely used 3D modeling frameworks are Blender, Maya, 3ds Max, Cinema 4D, Unity, and Unreal Engine. To produce the final images with the created scenes in the 3D modeling software, 3D rendering software is necessary. This software computes final pixels based on light placement and material types. In our scenes, we use 3ds Max software with V-Ray as the rendering engine for generating synthetic images and the 3ds Max Populate plugin for creating the avatars. In addition, we collect a dataset of real-world images so that it

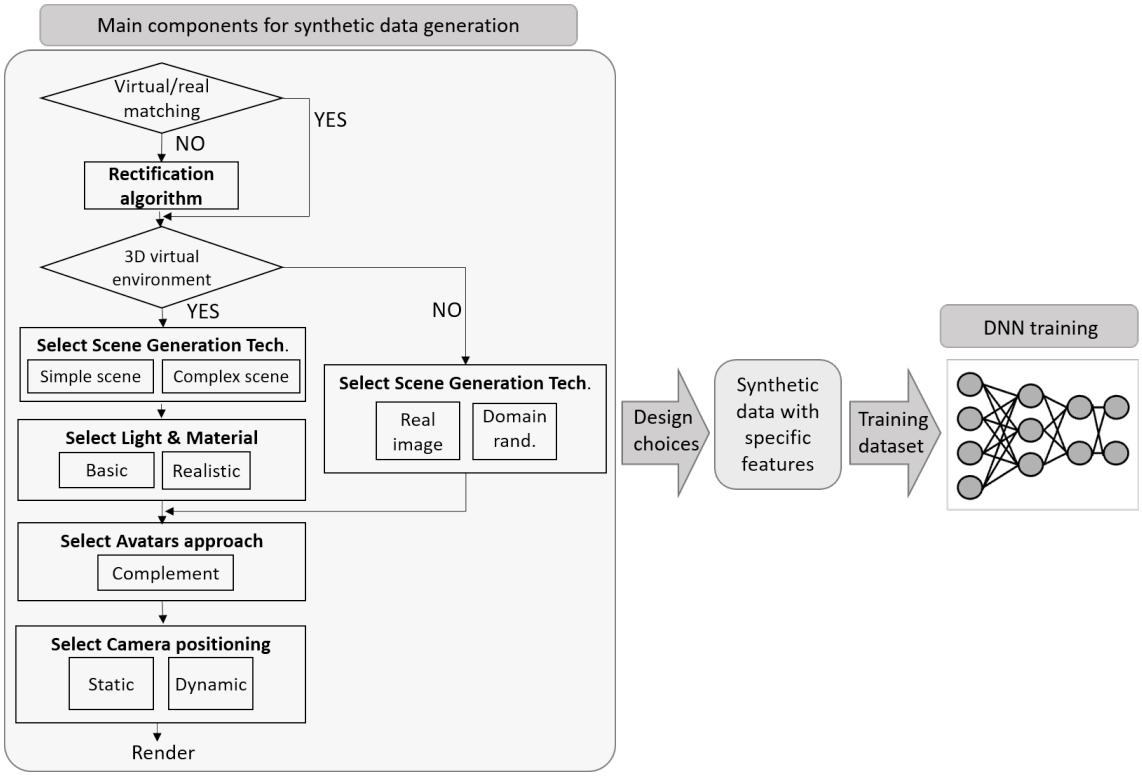


Figure 4.1: General scheme of the proposed methodology for analyzing and identifying the key features in synthetic data generation for DNN training.

is possible to compare the results achieved when training with synthetic samples with those achieved with real samples.

In the first stage, we generate basic scenes, including animated people. Subsequently, we improve lighting accuracy, enrich the scene decoration, add more assets, and include more complex relationships between the avatars.

The data used to train a DNN in a supervised way must be labeled. In our case, we need the bounding boxes around the people in the images. To get these annotations, we render masks that segment and instantiate the people in each image. For each frame, two images are generated: the final rendered frame and another image where only the people are rendered, and each person is assigned a single color using the V-Ray post-processing technique Render ID. A unique ID is assigned to each object in the scene. These people segmentation instances are used to compute the minimum bounding box enclosing each person.

4.1.1 Key Features in Synthetic Data Generation

In this section, we present the key features and parameters in generating synthetic data. We consider these parameters for the generation of the databases, described in Section 4.1.2.

4.1.1.1 Camera Lens Distortion: Rectification Algorithm

The cameras used to capture data introduce some artifacts into the image that are unique to the sensor. Simulating a camera without considering this can contribute to a larger domain shift between the synthetic and real data. In omnidirectional cameras, the lens distortion varies the captured data notably.

The closest camera model in most 3D computer graphics software is similar to a fish eye camera, which can be modeled in a simpler way than an omnidirectional one [167]. In addition, these virtual models are often simplified. This may not coincide with the content captured by the camera in the real world.

To check the possible matching, we prepare a scene with two calibration patterns of known dimensions and replicate the same patterns in 3ds Max with the same camera parameters. Figure 4.2(a) shows the image captured by the real camera. Figure 4.2(b) is the result of overlapping the simulated patterns and some person avatars on top of it. It can be seen that the patterns do not match, as the rendered assets show a different distortion.

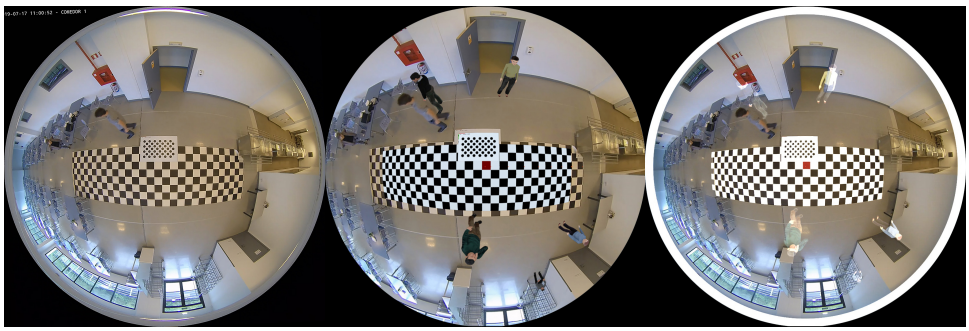


Figure 4.2: a) Image of a real scene; b) The same scene virtually rendered on top of the real image (patterns do not match); c) Virtual and real patterns match after the rectification step.

We apply an image rectification algorithm (Algorithm 1) to solve this. The input for

Algorithm 1: Distortion rectification algorithm

Input: Synthetic image $\mathbf{I}_s$, real image $\mathbf{I}_r$, markers' positions in real image $\mathbf{m}_r$, markers' positions in synthetic image $\mathbf{m}_s$
Output: Rectified synthetic image $\mathbf{I}_R$

$C_s \leftarrow$ Centre of $\mathbf{I}_s$ (width of $\mathbf{I}_s/2$, height of $\mathbf{I}_s/2$)
$\mathbf{r}_s = C_s$ - $\mathbf{m}_s$
$C_r \leftarrow$ Centre of $\mathbf{I}_r$ (width of $\mathbf{I}_r/2$, height of $\mathbf{I}_r/2$)
$\mathbf{r}_r = C_r$ - $\mathbf{m}_r$
$a, b, c \leftarrow$ estimate coefficients for $r_r = ar_s^2 + br_s + c$ (least squares method)
for *all pixels' position P in* $\boldsymbol{I}_s$ **do**
  $d_s = \sqrt{(P_x - C_x)^2 + (P_y - C_y)^2}$
  $d_r = ad_s^2 + bd_s + c$
  $P' = C + d_r(P - C)/d_s$
  $\mathbf{I}_R(P') = \mathbf{I}_s(P)$
end
Return $\mathbf{I}_R$

the algorithm are the real and synthetic images ($\mathbf{I}_r$ and $\mathbf{I}_s$ respectively), as well as the position in pixels of 10 random markers we place on the calibration patterns ($\mathbf{m}_r$ and $\mathbf{m}_s$). These markers represent the same spatial points, but their position in the images do not coincide because of the different virtual and real distortions. As the cameras produce radial distortion, we compute the distance of the markers' to the center of their corresponding image. These distances ($\mathbf{r}_r$ and $\mathbf{r}_s$) can be related by a second-degree polynomial equation. Then, the rectified position can be computed for each pixel's position in the synthetic image (P'). We use it to remap each pixel's value to the position it should be in in the rectified output image ($\mathbf{I}_r(P')$) based on the real image distortion. This algorithm is applied to the virtual scene and overlapped again in the real one, as is shown in Figure 4.2(c). It can be seen that the patterns now match.

This image rectification is applied to some of the generated synthetic datasets to test its influence on the detector's accuracy (Section 4.1.2).

4.1.1.2 **Scene Generation**

We perform various approach tests to create increasingly complex scenarios. We consider three scene environment configurations: based on real background images, simple

virtual scenarios and advanced virtual scenarios.

The fastest way to generate a synthetic dataset is to render the target objects (people) over real background images. An easy way to do this is to eliminate people from the background of a recording (real image) and add several avatars to it digitally. Another option is to use random background images. Based on the domain randomization technique [168], it is possible to generate non-realistic images to force the model to learn the essential features of the target object.

One step further is the creation of simple virtual 3D scenarios. The avatars are integrated into indoor synthetic scenarios with different decoration configurations and minimal furniture. The simplified configuration of the scene reduces the rendering time of each frame.

Finally, from these simple scenes, other advanced environments closer to reality can be generated. Following this, we create some scenarios where decoration focuses on different uses: commercial, transport, leisure, or private sector. Even if these images are more expensive in rendering time, a large variety of samples is achieved. Figure 4.3 shows some examples of these scenarios.

Figure 4.3: Advanced generation of interior scenarios generated for rendering.

4.1.1.3 **Lighting and Materials**

One of the most wanted features when generating synthetic data is realism. The key parameters to achieve this are scene lighting and materials, which can present different degrees of complexity. As shown in Figure 4.4 we develop two configurations, one more precise than the other. Both configurations include exterior lighting and artificial interior lighting.

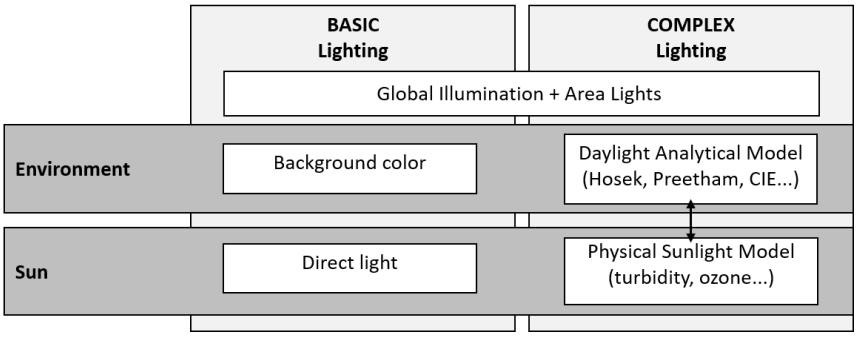


Figure 4.4: Different approaches of illumination

First, we create basic lighted scenarios where illumination is functional but only partially realistic. These scenes are illuminated with a direct light source, such as sunlight, and area lights along the ceiling of the virtual infrastructures to simulate typical artificial lighting used in large public spaces (e.g., LED panels). Also, global illumination methods are introduced to add more realism based on light-bouncing techniques. In this case, the materials used are very basic, without reflections or transparencies.

In the second configuration, we generate environments with more accurate exterior lighting in addition to the interior area lights. We introduce illumination changes in the same scenario due to weather conditions and the time of day. To achieve this, a daylight analytical model [169, 170] for sunlight and skylight prediction is added to the scene. This model reproduces the real-life sun and sky environment of the Earth, and consequently, different variants are introduced in the image, reflecting climatic and light alterations. The direction of the light source and the sky change dynamically depending on the position of the sun, as in real life. The sunlight intensity and color depend on the angle with the horizon. To achieve this, we translate the sun into the scenes, creating an animation curve from sunrise to sunset. Thus, the different colors caused by wavelengths add more naturalness and variety to the renders. Also, turbidity and ozone effects are introduced for fine-tuning the scene, which affects the color of the sky.

Regarding materials, in this accurate lighting model, we introduce a wide variety of elements with different characteristics in their composition, such as metals, porcelains, glass, or wood, to achieve higher realism and variation.

It is necessary to find a balance between realistic lighting and materials and the rendering time. Consequently, we reduce and simplify the quality of some materials to

avoid render times that are too long. Likewise, the number of polygons of the objects in the scene is simplified.

4.1.1.4 **Avatars and Accessories**

We use Autodesk's Populate plugin to add a wide variety of people to the scene. Characters can walk along paths or stay inactive in idle areas. It is possible to choose between men and women, although their age, complexion, and height are fixed for each gender. We generate two main types of animations for the avatars. We create free-moving corridors, where avatars walk at different speeds, and idle areas, where groups of people stand. In addition, we create some ad-hock animations for simulating queues of people or crowds.

Finally, the variety and complexity of the scenes can be increased by adding several accessories. We generate common objects such as handbags, backpacks, suitcases, umbrellas, magazines, and hats.

4.1.1.5 **Camera Position**

We test two configurations of the camera; statically positioned and moving during the sequence. When the camera is static, we place it where most scene action occurs. The camera's height in the different scenarios is varied between 5 and 7 meters since it is a common height for surveillance cameras in large spaces.

Regarding the dynamic camera positioning, following the experiments in [6], a batch of images is generated with random camera positions in the scene. The objective is to cover the different areas of the scene while varying its position and height in a range between 5 and 7 meters. Some results are shown in Figure 4.5. It can be seen that sometimes the images show uninteresting areas for detection, like walls or empty areas, but with much more enrichment in terms of pixel color distribution, sunlight, or scene assets visualization.

4.1.2 **Training Datasets**

We conduct experiments with 7 datasets. All of them are composed of the same number of images (5,600 images that are augmented to 28,000 samples). However, the choices for data generation design are different. Complex rendered images take

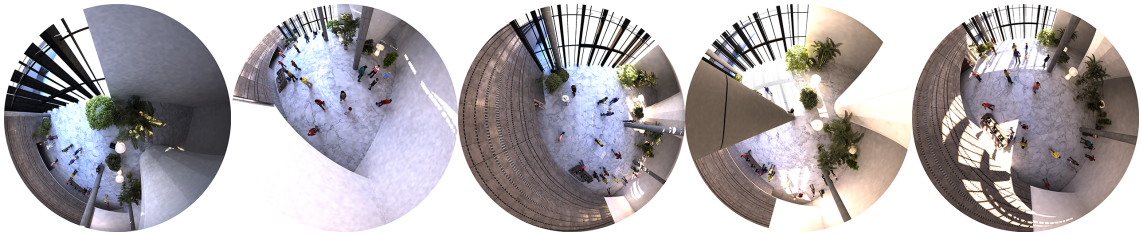


Figure 4.5: The same virtual scene captured from different positions during the sequence.

an average of 10 minutes. We use a render farm supported by 4 computers to generate the synthetic images. We use a Core i7 with 4 Core, 8 Threads, and 16GB RAM. There is no need to use a powerful GPU. The generated sets are available at `https://datasets.vicomtech.org/v4-osd/OSD_download.zip`.

Domain Randomization Dataset (DRD) The first dataset is based on the simplest scenario generation method based on the usage of random background images, explained in Section 4.1.1.2. We generate non-realistic images rendering a group of people walking along paths and standing on top of random backgrounds from [171], which present different textures and patterns.

Simplified Synthetic Dataset (SSD) This dataset contains more realistic images than the DRD. People follow the same movement patterns as in the DRD, but we simulate 3D environment scenes (4.1.1.2). People are rendered in different basic 3D scenarios with some common elements, such as chairs or columns.

Simplified Rectified Synthetic Dataset (SRSD) The images included in the DRD and the SSD are rendered using the fish eye camera model of 3ds Max. As explained in Section 4.1.1.1, these camera models' distortion differs from that of real-world cameras. The difference can negatively impact the detection accuracy in a real scenario. This dataset contains the same images as the SSD, but they are rectified in a postprocess step to mitigate the sensor differences and mimic better real-world data.

Advanced Synthetic Dataset (ASD) The generation of this dataset focuses on having more realistic scenes. This idea is applied to the avatars, the scenarios, and the lighting. People in real scenarios present a wide variety of appearances, not only because of their physiognomic differences but also because of the accessories they can carry (e.g., hats, bags). The avatars in this dataset include these kinds of accessories (Section 4.1.1.4). Regarding the light, we include realistic lighting (Section 4.1.1.3). Finally, more objects are added to the scenarios to simulate typical objects and architectural elements that can

be found in these kinds of places (e.g., specific furniture, stairs, booths) (Section 4.1.1.2). Image distortion is rectified as a postprocess step (Section 4.1.1.1).

Dynamic Advanced Synthetic Dataset (DASD) This dataset contains the same scenes as the ASD, but with a different camera position. Unlike the rest of the datasets, the position of the camera in the DASD is randomly animated and changes through the frame interval (Section 4.1.1.5).

Real Data Dataset (RDD) In order to compare the accuracy achieved when training with real data with the results obtained with synthetic data, we generate the RDD. This dataset contains 5,600 real-world images we have captured with an omnidirectional camera.

Real and Synthetic Data Dataset (RSDD) This dataset combines synthetic and real images. In order to have the same number of images as the other datasets and a balanced quantity of real and synthetic samples, 50% of the images are randomly selected from the RDD and the other 50% from the ASD. Image distortion is rectified as a postprocess step (Section 4.1.1.1).

The features of the described datasets are summarized in Table 4.1, and some example images of the corresponding datasets are shown in Figure 4.6.

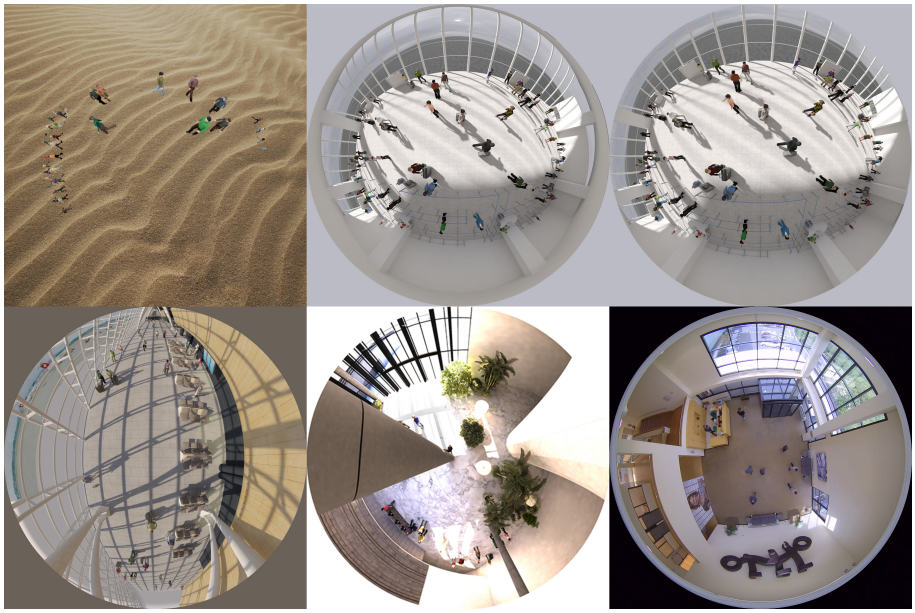


Figure 4.6: Datasets samples (from left to right and from top to down): a) DRD, b) SSD, c) SRSD, d) ASD, e) DASD, f) RDD.

Dataset	Real data	Distortion rectification	Camera position	Virtual scene	Virtual lighting	Avatars	Accessories
DRD	Background	-	Static	-	-	✓	-
SSD	-	-	Static	Simple	Simple	✓	-
SRSD	-	✓	Static	Simple	Simple	✓	-
ASD	-	✓	Static	Complex	Complex	✓	✓
DASD	-	✓	Dynamic	Complex	Complex	✓	✓
RDD	B&P*	-	Static	-	-	-	-
RSDD	B&P*	✓	Static	Complex	Complex	✓	✓

Table 4.1: Features of the generated datasets (*B&P: Background and People)

4.1.3 People Detection Model and Training

We train the single-stage YOLO-v3 [149] detector with different variants of the datasets. The same CNN configuration is replicated to test the influence of different choices in the dataset generation. Although different DNNs can be used, we use this model because it is a representative state-of-the-art DNN architecture when the target application needs a balance of accuracy and speed [65]. This is the case for intelligent systems that must provide real-time detection results. The YOLO-v3 model provides a faster inference speed than two-stage object detection networks.

The YOLO-v3 model splits the input image into a grid of cells. Each cell is responsible for detecting any object whose center falls within it. Compared to previous versions, it includes skip connections and upsampling operations and makes object detections at three scales. Therefore, it is better at detecting small objects. The DNN uses 9 anchor boxes as box priors. Predicted bounding boxes are defined by their four coordinates, the confidence score, and the class probability. The model uses the Non-Maximum Suppression (NMS) method to select the best bounding box when multiple are estimated for the same target.

We use an input resolution of 512x512 pixels for the network. Only one class is considered for the detection task (*person*). Regarding the anchors, we use the K-Means clustering algorithm to estimate the most appropriate anchors in the ground truth bounding boxes (3 anchors for each scale).

We apply different data augmentation techniques to the training images. The augmentations contribute to a more varied dataset and help to enhance the DNN robustness and generalization capability [172]. For each sample in the dataset, we generate 5 modified versions. We randomly combine geometrical transformations, image

rotation, or flipping, with color space augmentations, more specifically: Gaussian noise addition, brightness modification, Contrast Limited Adaptive Histogram Equalization (CLAHE) or motion blurring addition. Figure 4.7 shows an example of an augmented image.

Figure 4.7: Image augmentations applied to a rendered image sample. Color space augmentations are combined with geometrical transformations to generate new augmented samples.

All the models are trained on an NVIDIA Tesla P100 using the Darknet framework [149]. We train the model using stochastic gradient descent for 40,000 iterations with a learning rate of 0,001, a weight decay of 0,0005, and a momentum of 0,9. We set the batch size to 64. We initialize all the network weights with pre-trained weights on the MS COCO dataset [80]. We replicate the same training using each generated dataset (Section 4.1.2). The performance of these models is then evaluated on the evaluation data.

4.2 Experiments and Evaluation

To conduct the experiments, we train the people detection model with the same CNN architecture and training configuration but with different datasets. In this way, we can analyze the influence of the training data on detection accuracy. We evaluate these models on real-world images. Due to the lack of omnidirectional imaging real datasets, we make some recordings to generate a real-world evaluation dataset. To generate the recordings, we place a camera at 5.5 meters high in a corridor through which groups of people walk along, as shown in Figure 4.8. The evaluation set contains 625 images with a resolution of 2048x2048 pixels. These images have been manually annotated to provide the ground truth labels used for the evaluation. The annotation and evaluation region has been constrained to the area inside a radius of 8 meters from the center

of the image. This guarantees good annotation quality in the annotated area and is enough to evaluate the detector's accuracy. The scenario used for the recording differs from the one used in the RDD. Using the same scenario could provide some distorted results regarding the model's generalization ability.

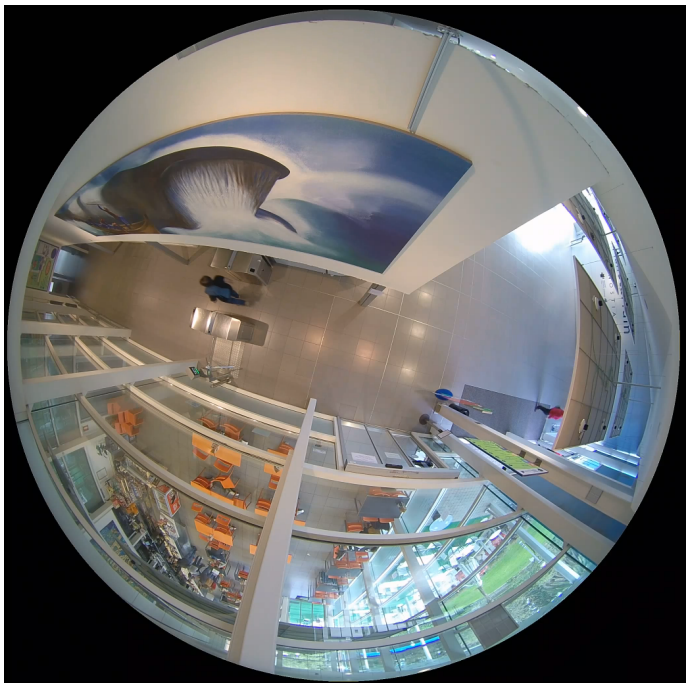


Figure 4.8: A sample from the captured real-world evaluation dataset.

In order to evaluate each model's accuracy, we compute the Intersection over Union (IoU) between the estimated and the ground truth bounding boxes. We consider a correct detection when the IoU is higher than 0.5. We use this to compute each model's precision-recall curve and the Average Precision (AP). We follow the PASCAL VOC evaluation criteria presented in [173].

4.3 Results and Discussion

We show the precision-recall curves for the models trained with the different datasets in Figure 4.9 and the resulting APs in Table 4.2.

As can be seen, the model based on the domain randomization technique (DRD) gets the worst result. Adding random backgrounds to the simulated people does not

Dataset	Main feature	Synthetic data	Real data	AP
DRD	Random backgrounds	✓	–	20
SSD	Sinple synthetic scenes	✓	–	37
SRSD	Distortion rectification	✓	–	43
ASD	Improved scenario, accessories	✓	–	57
DASD	Dynamic camera	✓	–	35
RDD	Real-world	–	✓	70
RSDD	Real and synthetic world	✓	✓	82

Table 4.2: Results of the YOLO-v3 models trained with the generated datasets (28,000 images in each dataset), which follow different strategies. AP values on the evaluation samples.

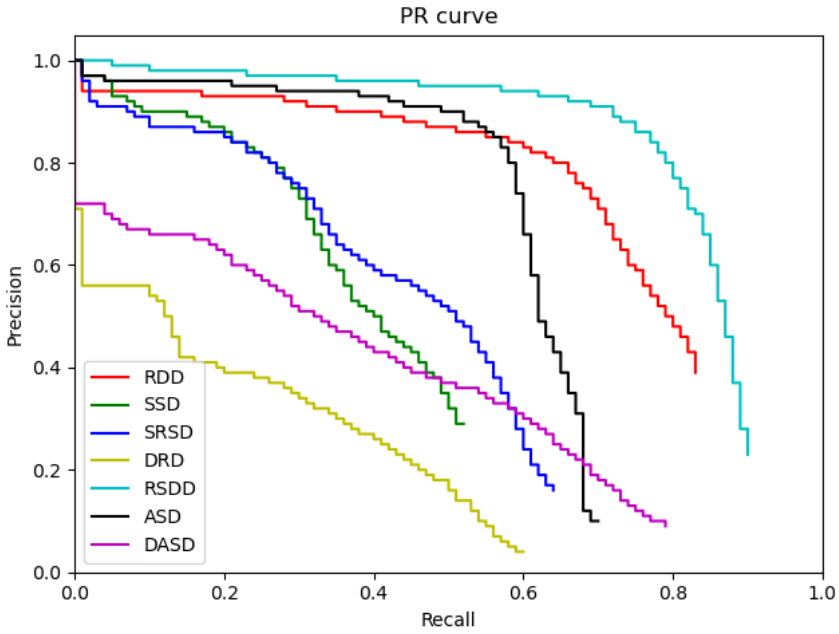


Figure 4.9: Precision-recall curves for the YOLO-v3 models trained with the generated datasets. Precision and recall values on the evaluation samples.

help the network learn the essential object features. The AP achieved in the real-world samples shows this strategy did not help get a detector with enough generalization capability. Replacing the random backgrounds with 3D scenarios (SSD) improves the AP from 13 to 37. The DRD includes randomness to generate non-realistic results but does not consider lighting or physical rules. This is different in the SSD. Simulating and rendering people in 3D scenarios adds some realism regarding the scene's lights, shadows,

and possible 3D assets, which seems to be an important feature.

The SRSD and the SSD only differ in the distortion rectification step. Adding this postprocessing step boosts the AP from 37 to 43. Ensuring that the images' distortion is closer to reality improves the final accuracy.

The ASD goes further in the samples' realism and improves the AP from 43 to 57. The accessories, the lighting, and the 3D scene assets help in learning more robust features with a smaller domain gap.

The DASD adds more variation to the ASD. The camera is moved through the scene, from the center to the corners. Therefore, samples in this dataset are very different from each other. However, the result is much worse (AP from 57 to 35). Even if the variation between images increases, the number of people in the samples decreases. When the camera is close to the corners of the scene, the perspective is different, and new objects in the scene are visible (e.g., plants, columns), but also fewer people are present in those images. We believe this imbalance of positive samples in the dataset is the reason behind the drop in AP.

The model trained with real data obtains an AP of 70, higher than the ASD (57). But when combining half of this dataset with half of the ASD, the AP increases still to 82. Training with half of the real images, but combining them with the synthetic samples, results in a higher AP than training with all real data. This may be due to the domain influence in the feature extraction learned by the DNN. Combining the real and best synthetic samples encourages the model to learn valid features for both domains. The more domain-invariant features the model extracts, the better it will perform in a new scenario. This is not always achieved because the results can worsen when the differences between both domains are too big. In those situations, more advanced domain adaptation techniques are needed. To validate that the results are not a coincidence, we repeat the DNN training with two new versions of the RSDD, which contain different random samples from the ASD and the RDD. The corresponding models obtain an AP of 81 and 82. Therefore, the results validate that the model benefits from synthetic and real domain training data.

Figure 4.10 shows some qualitative results of the model trained with the RSDD in the evaluated region. The scene has some challenging glitters and elements that can be mistaken for people. It can be observed that the model provides robust detections.

A false positive is shown in the right image around a pole. This could be improved by adding more scene elements to the synthetic scenarios.

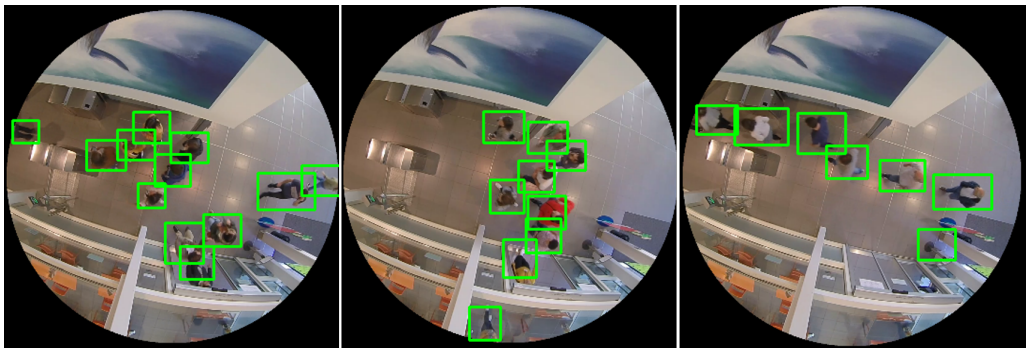


Figure 4.10: Inference samples of the model trained with the RSDD applied to evaluate real-world images. Inference is limited to the evaluated radio (8m).

Coming together is a beginning. Keeping together is progress. Working together is success.

Henry Ford

CHAPTER

5

Simulated Environments for Deep Learning

In this chapter, we propose a methodology to build a simulated 3D environment that aims to help define the required data and sensor setup for vision-based perception systems and generate appropriate data for DNN training (Chapter 1, Hypothesis 2). This work is motivated by the lack of general simulated environments ready to solve the camera setup and data generation tasks for all kinds of scenarios and use cases (Chapter 3). Typically, 'ad hoc' environments are built, or 3D scenarios are manually designed and set up each time a scene needs to be simulated. On the contrary, a generalist solution should allow:

- Including the required scenario-related graphical assets in an easy manner.
- Configuring context- and use-case-based scenes with user-friendly parameters.
- Capturing images from virtual camera viewpoints quickly, considering that the rendering time could be a significant bottleneck in this process. These images should include camera-related effects, such as the geometric distortion introduced by the lenses.

- Generating a wide and balanced range of plausible situations of interest, randomly applying suitable noise to the labeled data for the appropriate training of DNNs.

This chapter presents a methodology to build synthetic simulated environments for configuring and training multi-camera systems with sufficient generality to be usable in different surveillance contexts with little effort. We focus on static systems, i.e., those in which the cameras visualize the scene from specific positions. This means that applications involving dynamic systems (e.g., robots that interact with the environment while patrolling areas of interest), are beyond this scope. We show a practical implementation example of this methodology in the context of digitalized on-demand aircraft cabin readiness verification with a camera-based smart sensing system. We also compare it to alternative state-of-the-art approaches, including the qualitative and quantitative analysis of the data generation process and the required modifications to adapt it to another surveillance context. To ensure the suitability of the generated data by our methodology, we train a classification DNN and evaluate its accuracy when trained with real and synthetic images.

We have published the related research on the Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications [174](nominated for best student paper award).

5.1 Methodology

Figure 5.1 shows the architecture of the proposed methodology for generating synthetic data for training deep learning models. The input data of our approach are the 3D assets of the scene and the scene's description file. The outputs are the synthetic images and the annotations for training DNNs. The principal modules in our approach are: (i) the scene manager which is responsible to load the scene configuration; (ii) the engine which sets up the 3D scene according to the provided configuration and generates the training images by rendering different camera viewpoints and; (iii) the label generator which generates an output file containing the annotations corresponding to the generated images.

Similar to the data-collecting process carried out in a real environment, the first step is gathering all the necessary 3D graphical assets to reproduce the scene of interest.

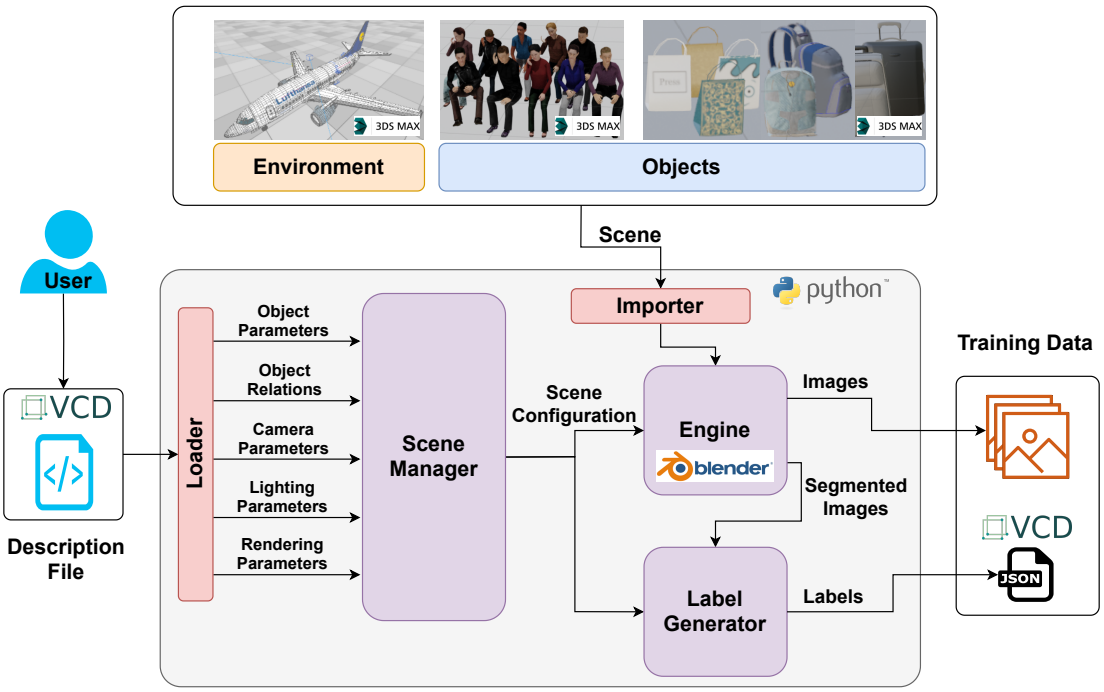


Figure 5.1: Software architecture of the proposed methodology for creating synthetic training data for vision-based systems.

Specifically, two different groups of graphical assets are required. The first type contains those graphical assets representing the environment (i.e., the scenario in which we should accomplish the recordings). We assume that such assets belonging to the environment are static since they represent the background. The second group contains those graphical assets representing the dynamic objects of the scene. The combination of the locations of these assets, their poses, sizes, and appearances, together with the variations of the lighting sources and the camera properties, will provide a wide range of variety for generating our custom training data.

For use cases in which some graphical assets are unavailable, the user should create them using 3D modeling software applications. We use *3DS Max*, but it is possible to use other applications such as *Autodesk Maya*, *Lightwave-3D*, *Vectary*, *Blender*, etc. Depending on the complexity, additional plugins such as *Populate* (for *3DS Max*) can alleviate the effort of designing the objects. Working with a very detailed 3D model can become a challenge due to the vast amount of polygons and materials the render engine has to process. This is directly related to the rendering time of the scene, and it could be

a troublesome bottleneck in the data flow when a user tries to generate thousands of samples. However, using very detailed 3D assets or more lightweight ones should be considered based on the scope of the use case.

Once all the scene assets are designed, our method allows us to configure different camera setups, place objects at multiple locations and poses, have multiple illumination sources, or configure the rendering parameters by minimum user interaction. Furthermore, the user can easily define multiple combinations of parameters for generating different training data samples in a single iteration.

Specifically, the user has to define a configuration file in which the parameterization of the scene is specified in a user-friendly way. Then, a loader interprets the content related to the different entities according to the nature of the parameters (objects, cameras, illumination, rendering). Such information is received by the scene manager, which interprets and handles these data to build the scene configuration for the user-defined sequence. This configuration is used to replicate the target scenes in the 3D synthetic environment, which implies loading the environment and dynamic assets with the corresponding configurations, as well as setting all the cameras, lighting, and rendering parameters to get the desired results. Then, the engine renders the images from the defined camera perspectives. We use *Blender* for this purpose, although the same methodology could be applied using similar alternative programs.

The label generator is in charge of generating the corresponding annotations based on generated segmented images. The user can choose different annotation types depending on the target computer vision task (e.g., object detection or semantic segmentation).

5.1.1 Scene Management

The synthetic scene is replicated based on the information provided by the user. The user is in charge of the configuration through the description file. For this purpose, we adopt the Video Content Description (VCD) structured JSON-schema file format [175]. VCD is an open-source metadata structure able to describe complex scenes, including annotations and all the needed scene information, in a very flexible way. In addition, it allows a very easy and fast user interaction for the files generation process. The VCD format is compatible with the OpenLABEL standard [176]. The configuration file contains

information related to the cameras and lighting setup, the 3D assets in the scene and the relation between them, and the rendering parameters. Modifying something in the scene, such as the position of an object or the camera specifications, is simply done by changing the corresponding field in the configuration file.

5.1.2 Camera Setup and Lighting Sources

The camera setup controls how the scenario and the objects are represented in the 2D images. In order to obtain realistic training data, the virtual cameras should simulate the same properties of the sensor and the lens that the expected cameras, which will be installed in the real environment. One of the major benefits of using *Blender* as an engine for creating and rendering the scene, in contrast with others such as *Unity3D*, is the multiple choices of camera models. In particular, we can generate the image projection of the virtual camera by using an orthographic model, a perspective model, or a panoramic model. These three models allow emulation of any combination of the image sensor and lens type mounted in a real camera. Table 5.1 shows the parameters that need to be added to the configuration file to add as many cameras as desired with their corresponding parameters.

Table 5.1: Camera Parameters

Camera model	Camera model (different distortion types).
Resolution	Output image resolution (pixels).
Position	Sensor position (m) in X, Y and Z axis.
Orientation	Sensor orientation (degrees) in X, Y, and Z axis.
Size	Sensor size (mm).
FOV	FOV of the sensor (degrees).
Focal length	Focal length of the sensor (mm).
Custom params	Extra params defined by the user.

Another important factor affecting the projection of visual information from 3D to 2D is the lighting of the scene. Depending on the target scenario, the user may want to add light coming from single points which emit light in all directions or from spots with a single direction (e.g., indoor lamps) or outdoor lighting simulating the sun. Figure 5.2

shows some examples of the mentioned lighting types. Table 5.2 shows the parameters that should be defined in the configuration file to add as many different light sources as desired to the 3D environment.

Table 5.2: Lighting Parameters

Light model	Type (point, spot, sun).
Position	Light position (m) in X, Y and Z axis.
Orientation	Light orientation (degrees) in X, Y and Z axis.

The scene configuration is automatically replicated in the 3D environment from the description file. During this step, the scene configuration is exported as a *Blender* project for cases where the user wants to explore the camera setup interactively. This way, it can be used as an interactive tool that helps to design a proper setup for a target application. Parameters such as the intrinsic and extrinsic camera parameters, their positions, or even the number of needed cameras can be modified by the user while he/she visualizes the images that would be captured. Simulating a specific setup with no need for physically deploying it can greatly help avoid wrong decisions that lead to a not optimal or wrong setup.

This way, it may help define the appropriate number of cameras, locations, poses, and viewpoints. Thus, the proposed methodology includes the following two bidirectional features:

- VCD2Scene: The user defines the VCD description file, and the scene is replicated in the 3D environment, including the 3D assets, camera configurations, and lighting sources.
- Scene2VCD: The user loads a 3D scene, and after making the desired modifications, he/she exports the new setup to a VCD description file.

5.1.3 3D Assets in the Scene

Regarding the 3D assets' configuration in the scene, it is also defined in the VCD file. The user can add as many objects as desired to the scene in specific configurations. These objects should belong to the available 3D asset types (e.g., humans, cars). Then

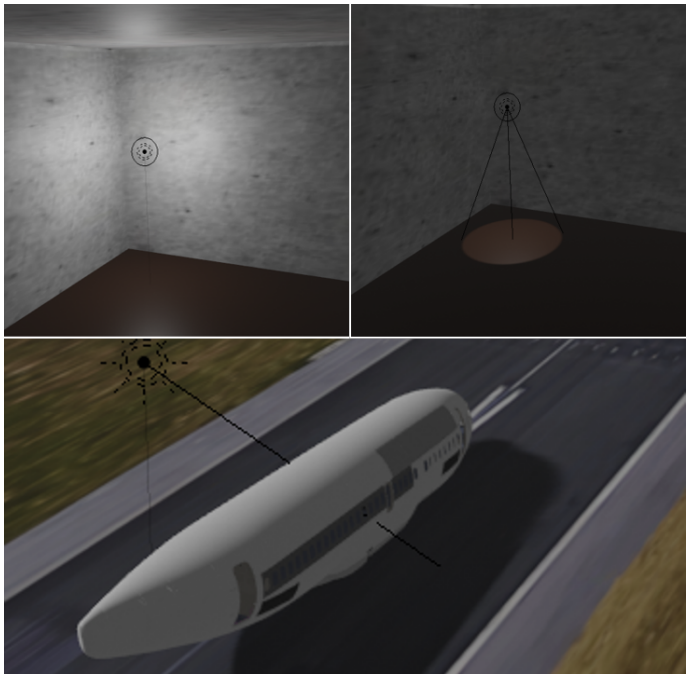


Figure 5.2: Light sources for a 3D environment: lamp emitting light in all directions (top left) and a single direction (top right), and lighting simulating the sun (bottom).

the 3D environment simulator interprets this information through the local relation between the assets. The user should have previously defined the relations present in the scene and interactively select the positions they would belong to. For example, if the target application is about detecting abandoned objects in an airport, some local relations that would be necessary could be "on" or "below". These relations would let the user relate different assets for example, by describing that certain objects (e.g., a bag, a suitcase) are *on* a desk or *below* the waiting seats. At the same time, the environment simulator would relate these positions with the user-selected positions. In addition, the user defines in the VCD file the time interval when each specific asset is present in the scene in the described configuration. This method provides high flexibility for the user to generate a wide variety of configurations in a very fast and user-friendly way.

In order to add a higher degree of variety to the generated data, when each asset is placed in a specific position some random noise is added to slightly perturb its position and orientation. In addition, the user can choose to apply random colors to the 3D assets to include more diversity in the data or, on the contrary, maintain the original textures.

5.1.4 Rendering Parameters

The rendering step turns the 3D scene into the output 2D image. The configuration of the rendering affects both the image quality and rendering time. The optimum configuration is closely related to the target task requirements and the available hardware. Therefore, our approach lets the user change the most influential parameters in the configuration file. The parameters that can be modified are shown in Table 5.3. The user can choose between the available rendering engines (in the case of *Blender* there are three available engines), computing caustics or not, the rendering tile size, the device to be used (CPU or GPU), and the maximum allowed light bounces. When the light hits a surface, it bounces off the surface and hits another one, and then the process is repeated. This is very expensive in terms of rendering time. Decreasing the maximum bounces implies limiting the number of times a ray can bounce before it is killed and, therefore, decreasing the time spent computing rays. Depending on the scene and the task, the user can adjust this parameter to get the desired output.

Table 5.3: Rendering Parameters

Device	Device used for rendering (CPU/GPU).
Engine	Engine type used for rendering.
Tile	Area of the image considered during the rendering.
Bounces	Maximum light bounces to be applied.
Caustics	Caustics computation.

5.1.5 Labeled Data Generation

Training machine learning models in a supervised way implies not only collecting the needed data but also the corresponding annotations. Annotating the data is an expensive process for which synthetic data generation can be very helpful thanks to automatic label generation. Our approach provides flexible annotations with different levels of detail depending on the target computer vision task (object detection, object segmentation, or visual relationship detection).

Getting the 3D assets' world coordinates and projecting them to the image plane would be a very efficient way to obtain the annotations automatically. However, it can

be observed in Figure 5.3 (corresponding to a scene in the context explained in the following section) that this could lead to wrong annotations because of occlusions. Different elements of the environment can occlude some objects, so if these occlusions are not considered, we would obtain annotations for objects that are not visible in the rendered images. An additional challenge is the camera distortion. Getting accurate annotations implies considering and replicating the distortion algorithm used by the chosen 3D graphics software's camera model to obtain the final objects' positions.

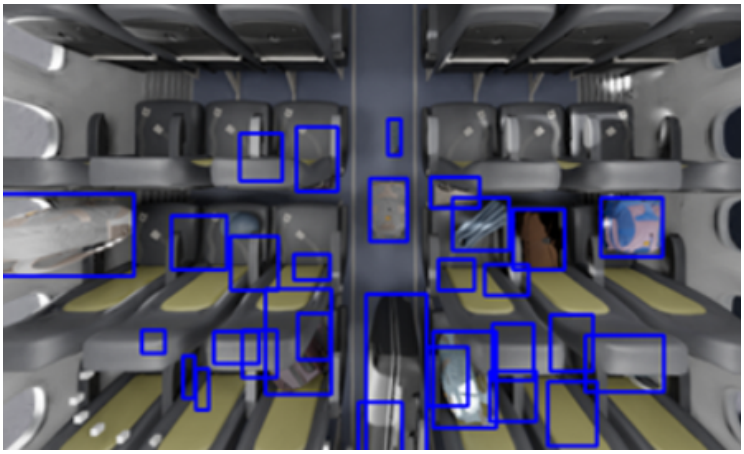


Figure 5.3: Object annotations based on 3D coordinates can result in some annotated objects occluded by the seats.

We opt for rendering accurate instance-level masks. Each asset in the simulated scene is given a unique ID apart from the object class ID it belongs to. These instance IDs are used to compute an alpha mask per object. These masks are combined in a single segmentation mask. When the data generation starts, the synthetic images are rendered simultaneously as the instance-level semantic segmentation maps.

The segmentation masks are used to generate the annotations, which are stored in the output VCD file. This file contains both the input configuration data and the annotations. By default, the masks are used to get the minimum bounding boxes containing each object's pixels, which are represented in the VCD file by each box's corner coordinates. These annotations can be used to train object detectors. However, our approach can be configured to save the objects' segmentation mask too. These data are already in the instance-level masks and are stored as object contours, which are described as polygons in the output file. In addition, the output VCD file contains the description

of the relations between the assets in the scene. These data are complemented with the objects' bounding boxes. These annotations can be used to train visual relationship detectors. The annotations need to be parsed to the required format depending on the chosen deep learning framework (e.g., *TensorFlow*, *PyTorch*).

The annotations in the output VCD file are stored per object and framewise. Consequently, each annotation is stored along with its corresponding image path.

5.2 Practical Case and Experiments

In order to validate the proposed methodology, we apply it to a real problem in the context of digitalized on-demand aircraft cabin readiness verification with a camera-based smart sensing system. Verifying Taxi, Take-off, and Landing (TTL) requirements in aircraft cabins is manual. The cabin crew members must check that all the luggage is correctly placed in each TTL phase. During these phases, the luggage should not be situated in such a way that an emergency evacuation of the aircraft would be delayed or hindered. Figure 5.4 shows the allowed and not allowed positions for the cabin luggage during TTL. The verification done by the crew members could be automated with the development of a vision-based system. This system would be beneficial in terms of operational efficiency and safety. Developing a system capable of detecting the luggage positions in the cabin entails two main challenges: the design of the camera setup and the generation of suitable data for training the corresponding machine learning models. As stated previously, a 3D environment simulator can be very helpful for these tasks.

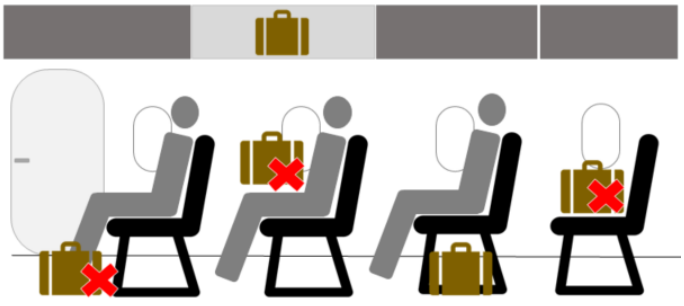


Figure 5.4: Authorised positions for luggage during TTL.

5.2.1 3D Assets for the Use Case

To address this problem, the first step of our methodology is the generation of the assets. We first generate all the involved 3D assets for the scene of interest. In this case, we model 22 different object types to simulate typical cabin luggage (e.g., backpacks, magazines, laptops), a cabin model representing a Boeing 737 aircraft (with 19 seats), and a group of human models with different poses and appearances for the seated passengers. The generated 3D assets are shown in Figure 5.5.

Figure 5.5: Generated 3D assets for the use case: cabin luggage, passengers, and aircraft cabin.

5.2.2 Cabin Camera Setup Design

For the camera setup task, our feature VCD2Scene allows loading the aircraft model within some of the modeled 3D assets using an initial configuration file and visualizing the 3D scene directly from the virtual camera viewpoints. The initial setup idea could be placing some cameras on top of the seats to control the luggage in these areas (e.g., backpacks partially below the seats) and other cameras above the corridor to verify a

clear exit. However, how many cameras should be installed? Where should they be to guarantee the system can see every seat without occlusions? What lens parameters should these cameras have?

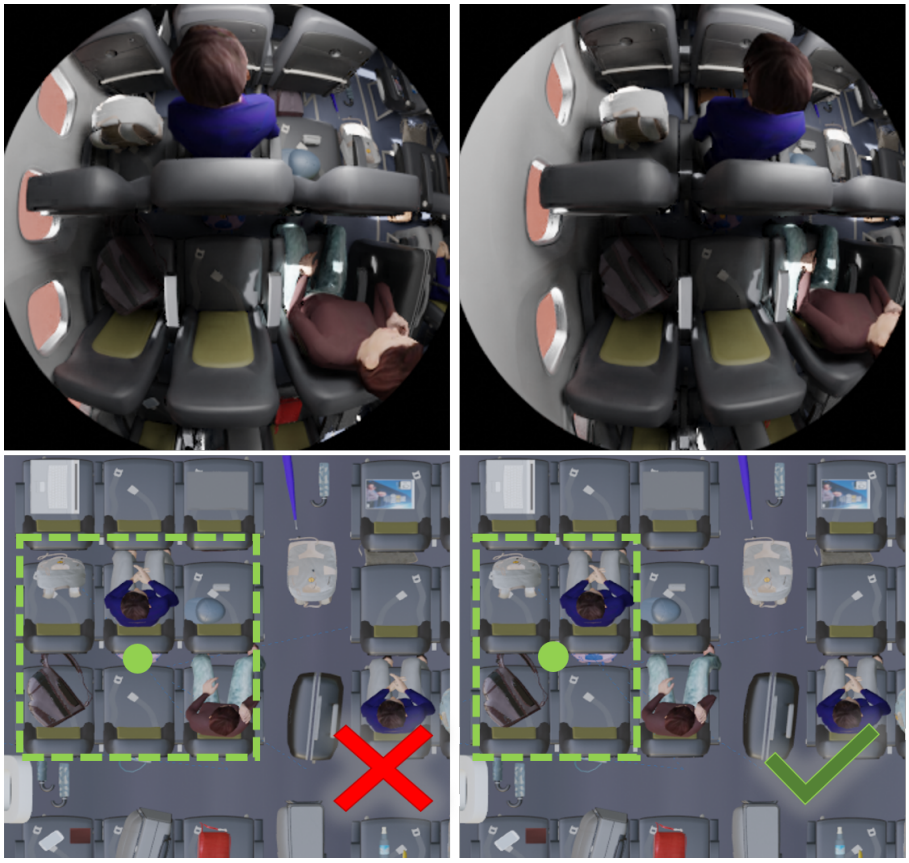


Figure 5.6: Captured scene from different camera positions. The configuration shown in the left images is discarded because of the occlusions in the front middle seat.

To answer these questions, we interactively change the virtual cameras' extrinsic and intrinsic parameters to check the results we would obtain with each configuration. We move the camera positions and orientations to guarantee that the minimum number of cameras captures all the regions of interest.

Figure 5.6 shows an example of accepted and discarded camera positions for capturing the luggage in the seat areas. At first, we could think the cameras should be on top of the middle seats to capture the status of 6 individual seats (left images). As it can be observed, the passenger seated in the front middle generates an occlusion that does not allow visualizing if he/she has some luggage incorrectly placed. To avoid this blind

spot, we test moving the camera toward the windows to capture 4 individual seats (right images). From this viewpoint, we can see no occlusion problems, so we opt for this configuration. Then, we test setting some cameras in the corridors to verify that area and the seats next to the corridor. Figure 5.7 shows the tested configurations. Even though the corridor space is well visualized in both shown camera positions, the camera should be aligned with the seats (right image) to guarantee the minimum possible occlusions in the feet of the passengers for as many seats as possible. These tests resulted in a setup design of 20 perspective cameras on top of the seats and 19 on top of the corridor. To guarantee a good visualization of the target areas, we set the parameters of all the cameras to a FOV of 118 degrees, a focal length of 2.13mm, and a sensor size of 4mm.

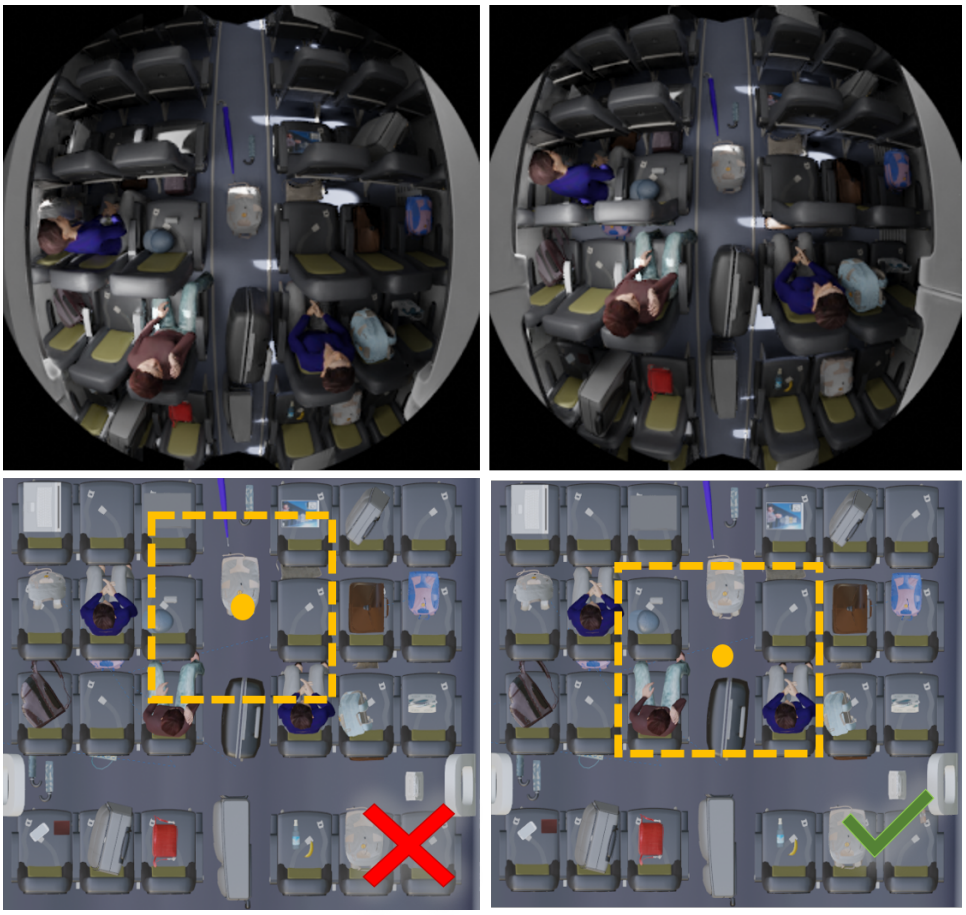


Figure 5.7: Captured scene from different camera positions. The configuration shown in the left images is discarded because of more occlusions.

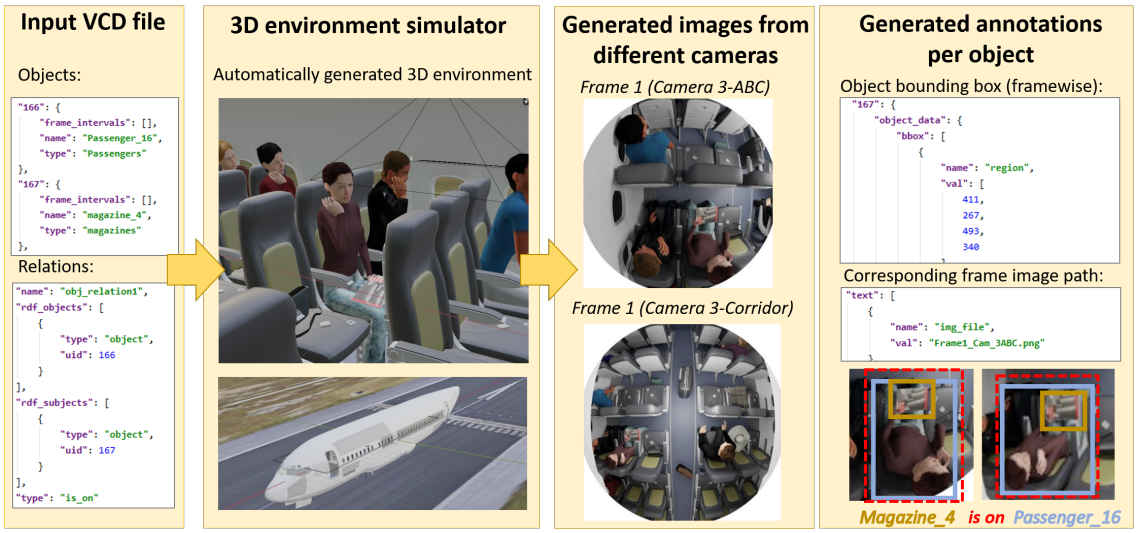


Figure 5.8: Data generation pipeline example. The input VCD file describes the scene that is replicated by the synthetic 3D environment generator, including present objects and their relations. The tool outputs rendered images and corresponding annotations in VCD format, corresponding to data from different camera perspectives.

5.2.3 Configuration Files Generation

Once we have the final cameras' configuration, we export it to an updated VCD configuration file with the Scene2VCD functionality. The file should also contain the 3D assets' configuration so that our environment generator replicates the defined sequences. The situations' variety and the number of object instances of each class can be easily controlled but depends on our configuration file. Generating a balanced dataset is important to guarantee that the trained model does not have a bias for the most common objects in the dataset. Consequently, we define the following requirements for generating the VCD files:

1. An object sample from each object category should be placed at all the possible configurations and places at least once.

2. All the object classes should be present in the generated sequences' frames the same number of times.

3. Samples should show a wide variety of object appearances.

Following these criteria, we generate three VCD files. The first file describes a sequence where each frame contains an object type placed in a specific configuration at all the cabin places (e.g., backpacks on all the seats). The goal of this sequence is to generate enough samples of each object type's appearance in simple configurations (with no interaction with other object types). The second file randomly combines all the objects in all the possible configurations the same number of times. We define a sequence of 500 frames. Each of these frames contains different object configurations. The last file follows the same strategy of random combinations, but it is focused on appearance variations. In addition to the default random variations aggregated to the 3D assets' position, it enables color randomization for objects (Section 5.1.3). Consequently, this file extends the already-defined object combinations with new ones that contain objects with a wide variety of random appearances. Providing a 3D asset with a random color can make some objects look less realistic but increases the variety of samples and can benefit the robustness of the trained models. The strategy used for generating the VCD files is not limited to the current use case, it can be applied to other tasks and scenarios. The defined sequences are summarized in Table 5.4.

Table 5.4: Summary of the generated data

Sequence	**VCD 1**	**VCD 2**	**VCD 3**	**Total**
Description	Single object class/frame	Random, balanced	Random appearances	-
Number of frames	45	500	500	1,045
Number of rendered images	1,755	19,500	19,500	40,755
Number of 3D object instances	5,966	33,000	33,000	71,966

5.2.4 **3D Scenes Generation**

We use the VCD files to automatically replicate the 3D scenes described in them with no manual intervention. The 3D environment is configured with all the data regarding the cameras, lighting, rendering, and 3D assets to replicate the defined scenes in the cabin. The environment is dynamically configured when the data generation starts.

5.2.5 Generated Data

Each frame in the sequences is captured from all the defined cameras, so for each frame, 39 images are rendered from different camera viewpoints. An output VCD file is generated for each sequence containing the data already in the input file (e.g., camera configurations, relations between objects) and the addition of the output data. The output data include the paths to the generated images and the corresponding bounding box annotations for each object or passenger. Figure 5.8 shows the data generation process from the input VCD file to the synthetic output data. The left image shows an example of the objects and their relations as defined in the VCD file. The 3D environment simulator processes this information to configure the 3D scene. It can be seen that the example data ('magazine-4 is on Passenger-14') is replicated in the synthetic 3D world). This scene is captured from the defined cameras to produce the corresponding rendered images and annotations. In the output synthetic images, it can be observed that the same 3D assets can be observed from different cameras at the same time. The right images of Figure 5.8 show an example of the output VCD file's additional information (objects' bounding box coordinates in each rendered frame). This information can be used not only for training object detectors but also for training visual relationship detectors.

5.2.6 Analysis of the Proposed Approach

This section provides a qualitative and quantitative analysis of the proposed methodology's characteristics. Table 5.5 shows a qualitative comparison of some state-of-the-art approaches to ours.

The environment and 3D assets involved in all the data generation methods are manually modeled with the help of a 3D modeling software or gathered from public repositories. [115] also uses data priors such as OpenStreetMap data to model different city environments but still needs manual work to complete and adjust the scene data.

Once the 3D environment is prepared, if the user wants to change certain scene configurations with our method, such as the light properties, the objects in the scene, or the relation between these objects, he/she can define the modified scene in the configuration file using a user-friendly parameterization. Then the new 3D scenario will be automatically replicated. Our approach is the only one that proposes to dynamically

Feature	[5]	[6]	[113]	[116]	[115]	Our approach
Environment modeling	Manual	Manual	Manual	Manual	Manual (data priors)	Manual
Scene modifications	Manual	Manual	Manual	Manual	Manual	Automatic
Scene capture	Single-sensor	Single-sensor	Single-sensor	Multisensor	Single-sensor	Multisensor
3D assets relations	Users' interactions	None	None	None	None	Spatial configuration
Annotations	Obj. detection, sem. segmentation, reinforcement learning	Obj. detection, sem. segmentation	Sem. segmentation	Obj. detection	Sem. segmentation, depth estimation	Obj. detection, sem. segmentation, visual relationship detection
Generality	Default scenarios	Limited	Driving scenarios	Limited	Driving scenarios	Surveillance scenarios

Table 5.5: Comparison between state-of-the-art synthetic dataset generation methodologies and our approach.

configure the environment when the data generation process starts. Scene modifications can be done in the different approaches, but none of them provides a high-level mechanism like ours to vary the environment. Related to the possible 3D assets relations, [5] allows adding some limited user interactions. Our approach allows for adding spatial location relations between the 3D assets.

Regarding image capture, all the works propose a single source to capture the scene, except for [116] and our approach, which allows capturing the same time interval from cameras with different viewpoints.

The data generated by the presented works are oriented to the training of DNNs. Typical output includes annotations for object detection or semantic segmentation tasks, to which [5] adds reinforcement learning. Our approach also generates labels suitable for training visual relationship detectors.

One of the main advantages of our approach over the others is its generality and the possibility to adapt it with little effort to new scenarios and tasks in the surveillance field. This feature is very limited to the predefined scenarios in state-of-the-art works.

Regarding the rendering time, it depends on different factors such as the rendering engine, the scene's complexity, the number of 3D assets included, the light configuration, or the polygon number of the modeled objects. For the current aircraft use case, we configure the rendering to be as fast as possible using the parameters in the configuration file, maintaining a good output quality. We use the real-time viewport shading rendering for the camera setup design (Section 5.2.2) so that we can see the modifications' effect interactively. We use the *Cycles Rendering Engine* for the data generation, which is slower but provides more photorealistic results. Table 5.6 shows the

Max Bounces			**Caustics**		**Tile Size**				
12	**6**	**1**	**Yes**	**No**	**16**	**32**	**64**	**128**	**256**
11.9	11.7	9.5	9.5	9.4	16.3	10.8	9.3	9.4	9.5

Table 5.6: Rendering time in seconds based on different configuration parameters using an Nvidia Tesla T4 GPU.

rendering time (s) computed in different tests to choose the optimum parameters to render a 640x480 pixel image of the cabin environment, including 50 3D assets (e.g., suitcases, passengers) apart from the background ones, 16 point lights and outdoors sunlight. We use an Nvidia Tesla T4 GPU. It can be seen that by reducing the maximum light bounces from 12 to 1 we need 2.4 seconds less without noticeable changes in the output image so we adopt only 1 light bounce. Disabling caustics effects neither change the scene appearance and we save an additional 0.1 seconds. These times are based on using a tile size of 128, but the optimum tile size for our hardware setup is 64. These parameter changes allow us to move from a rendering time of 11.9 seconds to 9.3 seconds. More than 2 seconds per sample in a process where we need to generate thousands of images is a remarkable time-saving. Regarding the rendering time of related state-of-the-art approaches, some works based on game engines claim to render in real-time [6, 115]. As stated in [115], a simplified lighting model allows real-time rendering of massive amounts of geometry with limited realism. The *Cycles Rendering Engine*, as a physically-based path tracer, allows for generating good quality results in a reasonable time. Domain adaptation techniques are important to solve the domain gap that DNNs trained with synthetic data can present. Generating data with a certain degree of realism minimizes the problem to be solved by those techniques. However, *Blender* also has a real-time rendering engine (*Eevee*) that can be used in tasks where the rendering time is considered to be a bigger priority than the data quality.

We apply our methodology to the aircraft use case, but adapting it to another surveillance scenario requires little effort from the user. Once the user collects all the 3D assets of the new environment, the flexibility of the method allows for building a new scene. The user interactively designs a suitable camera setup, along with the selection of target places. Then he/she can start gathering variate training data for the considered task based on the configuration files.

5.2.6.1 **Deep Neural Network Training**

In order to see how a DNN would perform with the generated synthetic data and real data, we train the model EfficientNet-B0 [29] to classify whether an image contains a correct or incorrect situation (e.g., cabin luggage correctly or incorrectly placed). We define an ROI for each seat, which will then be classified as correct or incorrect. We replicate the cabin mock-up for also capturing real images from the camera over the seats. Some examples of both real and synthetic ROI images are shown in Figure 5.9.

Figure 5.9: Real (top) and synthetic (bottom) ROI samples for training a classification DNN with correct and incorrect situations.

We do two tests to see the contribution of our synthetic images to the DNN accuracy. First, we train the DNN only with real data and test it on a separate subset of real images. Then, we repeat the training but incorporate the same number of samples from our generated synthetic dataset and test it again on the real test images. We capture and annotate 1,600 real images.

We initialize the DNN with pretrained weights on the ImageNet dataset [177] and fine-tune it with our datasets. We train each DNN for 50 epochs with a batch size of 40 and the RMSprop optimizer [178]. The model trained only with real images achieves 88.52% accuracy when classifying the real test samples, while the model, which also includes synthetic samples, achieves 94.26% accuracy on the same images. Consequently, the positive contribution of the generated synthetic data in the model accuracy confirms the suitability of the generated images for DNNs training.

Many of life's failures are people who did not realize how close they were to success when they gave up.

Thomas A. Edison

CHAPTER

6

Synthetic Data Generation with Generative Models

In this chapter, we explore the use of synthetic data generation for deep learning using generative models (Chapter 1, Hypothesis 3). We work on the generation of synthetic data in the context of gaze estimation. Gaze estimation is an essential task in computer vision for many applications. For example, the valuable information provided through gaze estimation can be used to create more natural ways to interact with computers, design more immersive experiences in virtual worlds, and improve driving safety by detecting driver fatigue or distraction.

Despite the abundance of facial images through the Internet and open large-scale datasets [134], obtaining gaze-annotated images for training DNNs remains a significant challenge. The process of capturing gaze-annotated data is labor-intensive. This process typically requires specialized equipment and long capturing sessions, where a volunteer is asked to progressively direct their gaze to various points [179].

Given the high value of each captured sample, it is crucial to explore alternative methods for obtaining such data. Synthetic data generation and augmentation through the use of generative models might be a promising alternative.

In this chapter, we propose a Gaze-aware Compositional GAN learned from limited annotations and study the suitability of generated data for DNN training and other potential applications.

The research presented in this chapter is submitted and under revision to be presented at the 32nd International Joint Conference on Artificial Intelligence (IJCAI).

6.1 Methodology

In a typical GAN, a generator model is trained to map a noise vector z (usually from a standard normal distribution) to an image x. Our proposed method learns to generate an eyes' region image x and a corresponding segmentation mask y given a vector z and a target gaze direction θ, as defined below:

$$G : (z, \theta) \longrightarrow (x, y) \tag{6.1}$$

where $y \in \{0,1\}^{HxWxK}$, being H and W the image dimensions, and K the number of segmented face components.

6.1.1 Gaze-aware Compositional GAN

We have the following three observations: (1) Some facial components are related to the gaze (e.g., iris), while others are not (e.g., nose). (2) Our facial image generator needs to be compositional, as we need the gaze-related facial components to remain the same during the gaze-invariant data augmentation for the gaze estimation task and to be changed alone during the gaze redirection editing task. (3) Our discriminator needs to penalize the sample if it doesn't match the conditioned gaze direction in addition to being not realistic. Thus, we designed the following model components.

6.1.2 Gaze-aware Facial Image Generation

Realistic eye image synthesis has been extensively studied because of its importance in various applications. There are two main approaches to synthesizing eye images: 3D modeling of the eye and image generation using generative models.

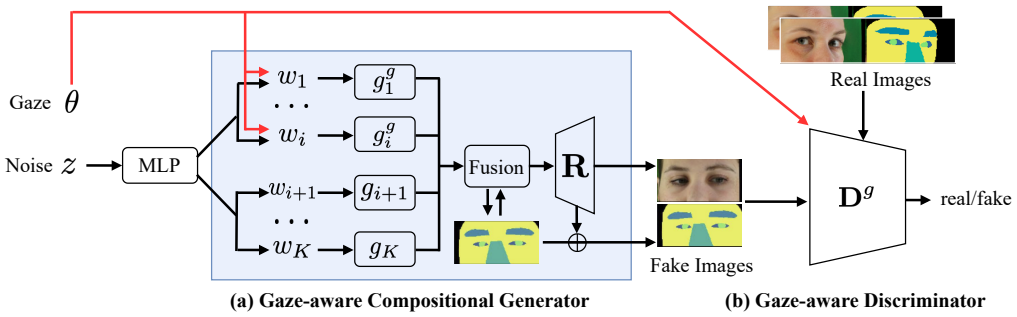


Figure 6.1: The proposed Gaze-aware Compositional GAN model (GC-GAN). Building upon the SemanticStyleGAN [7], we first group facial components into gaze-related $\{w_1,\ldots,w_i\}$ and gaze-unrelated $\{w_{i+1},\ldots,w_K\}$. Then we extend (a) the Local Generators $(\{g_j^g\}_1^i)$ in the generator and the discriminator D^g to condition on the input gaze θ.

6.1.3 Framework: Gaze-aware SemanticStyleGAN

To ensure the composability of our GAN, we adopt the state-of-the-art SemanticStyleGAN model [7]. As illustrated in Figure 6.1, our Gaze-aware Compositional GAN model (GC-GAN) is divided into three main components: noise to latent vectors mapping, Gaze-aware Compositional Generator for image generation, and Gaze-aware Discriminator for discriminating images during training. The input vector z is mapped to an intermediate latent code $w \sim W$ using an 8-layer MLP to better model the non-linearity of the data distribution, similar to [135]. To obtain a disentangled latent space for different components of the face, the latent code w is divided into K local latent codes and an additional base latent code $\mathbf{w}^{base}$, common for all the face components. The gaze-aware compositional generator processes these latent codes. This module has K generators, each responsible for generating feature maps for a specific face component given a local latent code w_k. All the output feature maps are fused and fed to the final generator, which generates the synthetic images and their corresponding segmentation masks. During the training, the generated images and masks are fed to the discriminator and real samples from the training dataset. Gaze vectors are also input to the discriminator when available.

The main modules of the model are detailed as follows.

6.1.4 Gaze-aware Compositional Generator

As the Render Net is gaze agnostic, we only need to condition the gaze direction for the Local Generators of gaze-related facial components, $\{g_j^g\}_1^i$. We condition the Local Generators with the gaze direction feature for gaze-related features. We refer to these generators as gaze-aware local generators (GLG). Figure 6.2 shows the architecture of a Gaze-aware Local Generator (GLG) for a specific facial component k. The GLG is formed by modulated 1x1 convolution layers with latent code-conditioned weights and input Fourier features f_p for position encoding. The input latent codes for each Local Generator are the base latent code $\mathbf{w}^{base}$ and the face component-specific w_k. The latent code w_k is divided into shape and texture latent vectors, w_s^k and w_t^k. The target gaze direction θ is also input to the GLG to control the gaze of generated samples. The input gaze θ is defined as yaw and pitch angles (φ_y, φ_p) and is fed to a fully connected layer for mapping it to an adequate 64-dimensional space before fusion. The output of this layer is concatenated to the component-specific latent vectors, and the extended latent vectors, w_s^{k+} and w_t^{k+}, are fed to a series of modulated 1x1 convolutions and final fully connected layers. The output of the GLG is a 1-channel pseudo-depth d_k and 512-channel feature map f_k. The hidden layers have 64 channels.

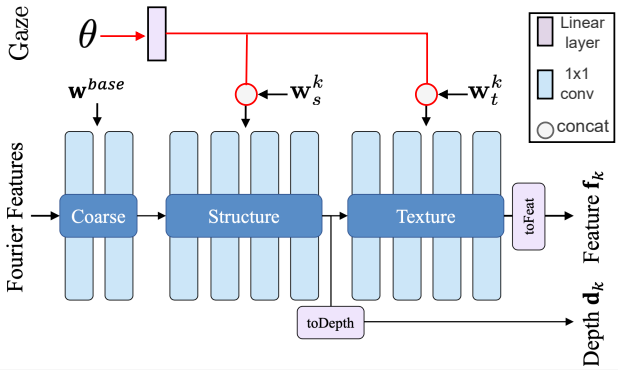


Figure 6.2: Architecture of our Gaze-aware Local Generator (GLG). The red lines are additions compared to Shi *et al.* [7].

The local generators unrelated to gaze follow the same architecture except for the gaze branch, which we discard.

6.1.5 Gaze-aware Discriminator

Our generated samples should be discriminated based on the joint distribution of the image, the mask, and the gaze. As shown in Figure 6.3, for the image and mask, the discriminator has two convolution branches composed of residual blocks whose outputs are summed up. The mini-batch standard deviation of the resulting feature maps is concatenated, and 3x3 kernel convolution is applied before adding the data from the gaze. The gaze is included as a third input to the discriminator. A linear layer processes the gaze to map it to a 64-dimensional space before being concatenated to the feature maps from the dual branch. The concatenated maps are processed by the final linear layer, which classifies the input data as real or fake.

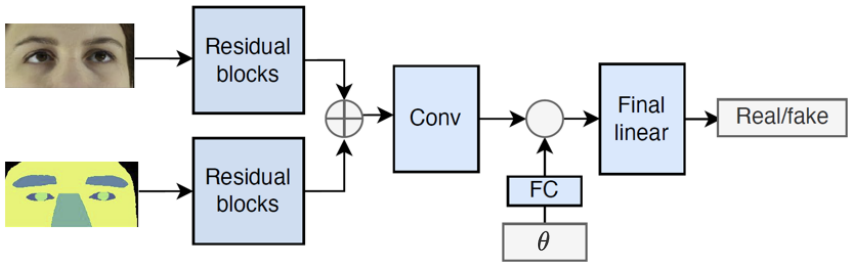


Figure 6.3: Architecture of the discriminator in the first training stage. Outputs from residual blocks are summed, while features from gaze direction are concatenated.

6.1.6 Two-stage Training

We first train the GC-GAN model on the labeled data and then transfer the facial appearance from the unlabeled data with the domain adaptation approach (Figure 6.4).

6.1.7 Stage 1: Training on Labeled Data

During the training of the first stage, the following loss function is minimized.

$$L_{s1} = \lambda_l L_l + \lambda_r L_r + \lambda_p L_p + \lambda_m L_m + \lambda_s L_s \tag{6.2}$$

where L_l is non-saturating logistic loss [180], L_r is R1 regularization loss [181], L_p is path length regularization loss [135], L_m and L_s are mask and segmentation regularization loss [7]. Each loss is weighted with the corresponding λ hyperparameter. However,

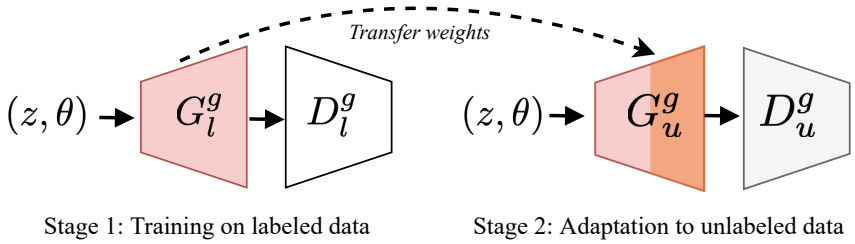


Figure 6.4: Overview of the two-stage training: (1) training on limited annotated data, (2) adaptation to unlabeled data.

due to the limited amount of labeled data, it is hard for such a trained model to generalize well to new images acquired in the wild that are challenging to have accurate gaze annotations.

6.1.8 Stage 2: Adaptation to Unlabeled Data

To adapt GANs to the target domain, it is a common practice to freeze the lower-level layers of the generator module [182]. The challenge here is to find out which module to freeze. Moreover, in our case, the target domain lacks the label information that is used in the source domain, which requires the modification of the discriminator model.

Freezing Gaze-aware Compositional Generator. To transfer the model to the appearance distribution from a new image domain, we rely on the hypothesis that samples with the same segmentation mask share the same gaze direction. A specific latent vector w and input gaze direction θ, fed to the gaze-aware generators module, results in a fused coarse feature map and mask, which are then refined in the Render Net. Our goal is that the same latent vector generates images in both dataset domains that share the same gaze direction and coarse features (e.g., pose), but domain-specific appearances. For that purpose, we initialize the model with the pretrained weights from the first stage, freeze the GLG and the first block of the rendering generator, and fine-tune the rest of the model. Random input gaze directions are fed to the generator.

Modifying Gaze-aware Discriminator. There are no available gaze annotations to use as input along with real images and masks, so we constrain the gaze requirement by using the segmentation mask. The discriminator has the same architecture as in Section 6.1.1 without the input gaze branch. The discriminator receives real pairs of RGB

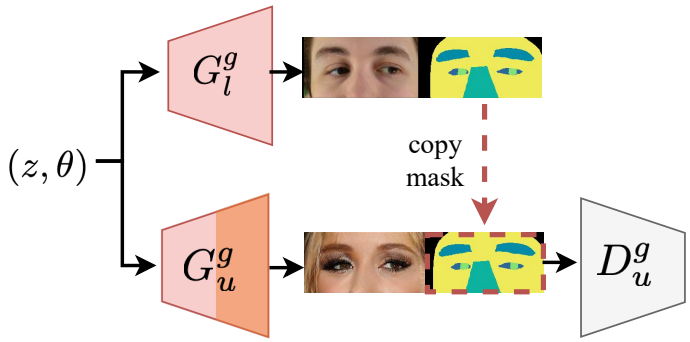


Figure 6.5: Stage 2: Adaptation to unlabeled data. The generated image is combined with the mask generated by the pretrained stage-1 model given the same latent and gaze vectors.

images and the corresponding segmentation masks, and the same with synthetic pairs. In the synthetic pairs, we use the synthetic image generated by the generator given a latent vector w and a target gaze direction θ, but an expected segmentation mask for that latent vector instead of the generated one. The expected mask is generated by the previously pretrained model, as shown in Figure 6.5. The model trained in the first stage has already learned how to model the relation between the latent code and the gaze in the generated images. Consequently, using it to generate the masks for the discriminator's input pairs forces the learning GAN to generate images that fit the given masks and consequently the input gaze direction.

We also add a mask loss L_g, which is minimized together with the same losses as the first stage. This loss helps to preserve the same generated mask for the same latent vector and consequently the same gaze direction. The new mask loss measures the mean squared error between the generated mask with the updated weights and the mask generated using the pretrained weights given the same latent and gaze vector. The final loss of the second stage is defined as follows:

$$L_{s2} = L_{s1} + \lambda_g L_g \tag{6.3}$$

where L_{s1} are the losses defined in Eqn. 6.2 and L_g is the mask loss for gaze preservation.

6.1.9 Applications

6.1.10 Gaze-aware Facial Image Synthesis

The composition-based architecture promotes a disentangled and interpretable latent space for image editing. After model training, we can generate a synthetic sample given an intermediate latent vector w, and a target gaze θ. We can redirect the gaze of the generated sample by modifying the input gaze direction θ. Regarding the semantic facial components, we can modify the specific k_{th} component by varying the corresponding local latent vector $\mathbf{w}_k^s$ or $\mathbf{w}_k^t$. We use cubic spline interpolation to smoothly move from a local latent vector to a different one.

6.1.11 Data Augmentation for Gaze Estimation

Figure 6.6 shows examples of the different ways for generating data we propose: data generation in different domains using the same latent and gaze vector, redirecting the gaze, and modifying specific facial components.

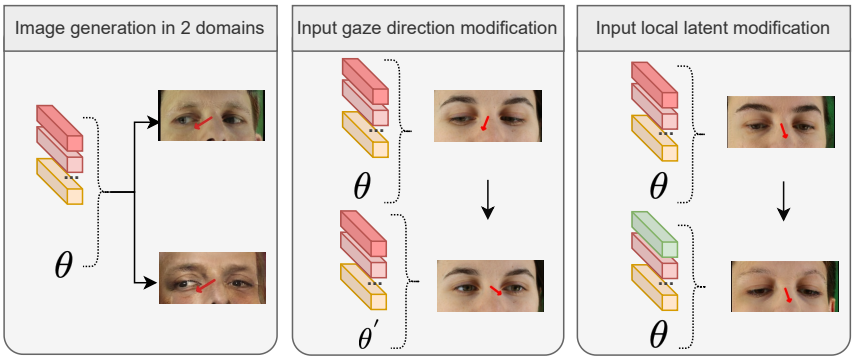


Figure 6.6: Design choices for data augmentation with the trained GC-GAN model.

6.1.12 Gaze-aware Facial Image Editing

To edit the gaze direction of the input facial image, we need to map the image to the learned latent space. This can be done by inverting each image. Image inversion can be approached as an optimization process that finds the latent vector that results in

the closest possible image to the target one when input to the generative model. In this optimization, we use the losses defined in Equation 6.4.

$$L_{inv} = \lambda_p L_p + \lambda_d L_d + \lambda_m L_m, \tag{6.4}$$

where the perceptual loss L_p measures the high-level similarity between the generated image and the target image computing the L2 distance between intermediate image features extracted with an ImageNet-pretrained VGG19 network [183]. The L_d computes the pixel-wise MSE loss between both images. Finally, the L_m computes the difference between the evaluated latent vector and a mean latent vector, so that the result does not deviate too much from it. The mean latent vector is computed by averaging multiple latent vectors from random z vectors. All losses are weighted with the corresponding λ hyperparameter.

6.1.13 Implementation Details

We first train the composition-based GAN with the ETH-XGaze subset and then transfer the model in the second stage to the CelebAMask-HQ dataset. We define the iris and sclera as the conditioned facial components. Random input gaze angles are fed to the model during training, sampled from the real gaze range of the labeled dataset. We set a dimensionality of 512 for z and w, and the image size is 256x256. Similar to StyleGAN2, we use style mixing regularization [135] and leaky ReLU activations. We empirically set the first stage's λ hyperparameters to $\lambda_l = 1$, $\lambda_r = 10$, $\lambda_p = 2$, $\lambda_m = 100$, and $\lambda_s = 500$. In the second stage, the additional λ_g is set to 100. We use the ADAM optimizer [184] with the hyperparameters $\beta_1 = 0$, $\beta_2 = 0.99$, and 8 as the minibatch. We implement our framework using PyTorch 1.7. All the experiments are done using an Nvidia GPU GeForce RTX 3090.

6.2 Experiments

We present a thorough examination of our proposed framework by implementing both qualitative and quantitative experiments. We aim to demonstrate the efficacy of the framework in generating both within-domain and cross-domain data augmentations, as

well as gaze-controlled images. For further implementation specifics, we direct readers to the appendix.

6.2.1 Datasets

We use the ETH-XGaze dataset [179] as the primary source of gaze-labeled data. It contains facial images of 110 subjects captured from 18 cameras in a controlled environment. We randomly select a subset of 8 subjects for training (14,464 images) and 2 for testing (3,920 images) for the four most frontal cameras. While the ETH-XGaze dataset features a wide and balanced gaze range, the variety and naturalness of individuals captured in controlled environments may be limited. To address this limitation, we use the CelebAMask-HQ dataset [134] as the unlabeled dataset, which contains 30,000 in-the-wild images of celebrities.

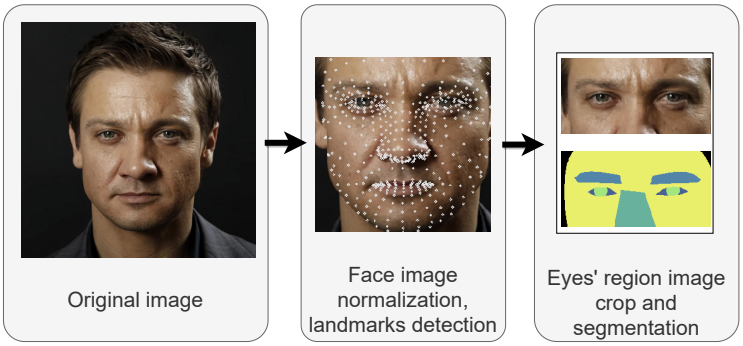


Figure 6.7: Preprocessing of a sample from the CelebAMask-HQ dataset: face and face landmarks are detected for image normalization and eyes' region image crop and segmentation.

Both datasets were preprocessed to generate 256 × 256 sized eyes' region images. It is common to use image crops containing one or both eyes for gaze estimation DNNs [128, 185, 186, 187]. We use the eyes' region crop for our experiments. Individual eyes can be cropped after synthetic image generation if needed. For models using the entire face, our approach could be extended to the face. We preprocess the images to get a normalized 256 × 256 eyes' region image crop and the corresponding segmentation mask. We use the face detector in [**?**] to detect the faces in images, and we detect the face landmarks with [**?**]. The image normalization includes rotating the images to eliminate any roll angle due to the head pose, centering the images based on the face center, and

Table 6.1: Error (degrees) in the test set when trained with different kinds of augmentations.

Augmentation	**Error↓**
No Aug	4.54
Color Aug	4.47
Color+Geometry Aug	4.52
Ours (in-domain)	4.22
Ours (in- and cross-domain)	**3.86**

scaling them to have 1.7 times the eyes' region width considering the furthest eye corners horizontally. We define the face center as the average point considering the average of the eyes' coordinates and the average of the mouth's coordinates. The eyes' region is the facial image's upper half. We use the face landmarks to generate the corresponding segmentation mask, which includes the following categories: background, face, iris, sclera, eyebrows, and nose. Figure 6.7 shows an example of image preprocessing for a sample in the CelebAMask-HQ dataset.

6.2.2 **Data Augmentation for Gaze Estimation**

To assess the effectiveness of our proposed method for data augmentation, we train a DNN for gaze estimation with and without the use of augmented data, respectively. We train a baseline gaze estimation DNN using an off-the-shelf ResNet-50 network [28] as in [179]. The DNN takes 224×224 eyes' region images and estimates the gaze direction as yaw and pitch angles.

We build four configurations for the augmentations: (1) a baseline configuration with augmentations with geometric and color modifications (e.g., CLAHE, shift and scale perturbations), (2) a variation of the baseline configuration that only employs color modifications (e.g., brightness variations), (3) GAN-based within-domain augmentations, and (4) GAN-based within and cross-domain augmentations.

For GAN-generated augmented samples, we first embed the ETH-XGaze subset samples into the closest latent vectors. New samples are then generated through iterative modification of unconditioned face components and redirecting the gaze vector of some of the samples (Section 6.1.9). We generate 8,000 synthetic images for the within-domain augmentations and further augment the training set by an additional 5%

with the cross-domain augmentations. We use the same inverted images' latent vectors to generate the synthetic images in the CelebAMask-HQ dataset domain.

Table 6.1 shows the error obtained by the DNN in the test set when trained with the different augmented sample configurations, as well as when trained without any augmentations (4.54 degrees). While all augmentations lead to an error drop, the error reduction is significantly higher when utilizing GAN-based augmentations. Additionally, we observe that even though the cross-domain augmentations belong to a different domain, they still help to improve the model's generalization capability.

Figure 6.8 illustrates qualitative results of the different augmentations, as evaluated in Table 6.1. The first row shows examples of images that have undergone geometry and color modifications, as used in the baseline augmentations configuration. The second row displays within-domain augmentations for the same subject, including the original image and modification of the nose, eyebrows, and gaze direction. The images in the last row depict pairs of synthetic images in the ETH-XGaze and CelebAMask-HQ data domains, generated using the same latent vector for each pair. It can be observed that despite the domain-specific appearance of the images in each pair, they maintain the same gaze direction.

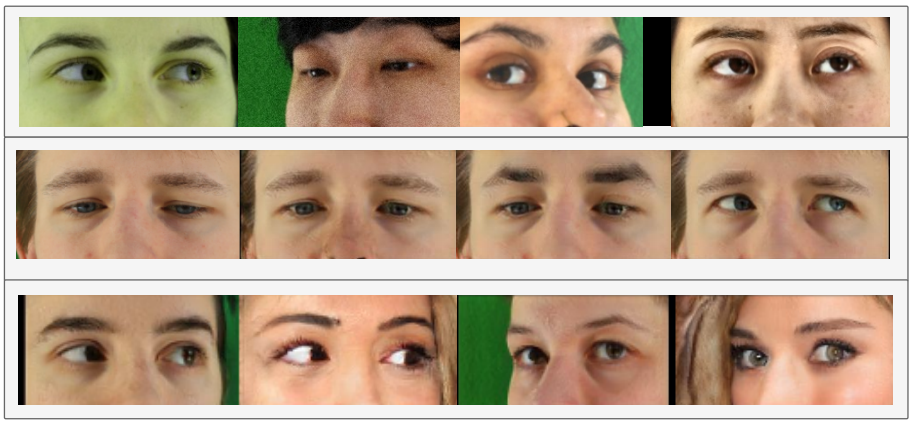


Figure 6.8: Different augmentations evaluated for DNN training. Top to bottom rows: color and geometric augmentations, within-domain augmentations, and cross-domain augmentations.

6.2.3 Gaze-aware Facial Image Generation

6.2.4 Facial Image Synthesis

The quality of image synthesis in generative models is evaluated using metrics such as the Fréchet Inception Distance (FID) [188] and the Inception Score (IS) [189]. Our models obtain a mean FID of 15.3 and a mean IS of 1.92. As far as we know, there are no generative models with input latent vector and gaze to compare with. Consequently, we compare these metrics to the state-of-the-art methods for in-the-wild images' gaze editing. Table 6.2 shows that while both DeepWarp and GazeGAN achieve a higher IS, they also present a much higher FID. This suggests that our method is able to generate more realistic synthetic images. In addition, existing methods for gaze redirection are typically limited to frontal gaze or between two images, lacking fine-grained control.

Table 6.2: Image quality comparison with state-of-the-art methods for in-the-wild images' gaze redirection.

Method	**FID ↓**	**IS↓**
DeepWarp	106.53	2.89
GazeGAN	30.21	3.10
Ours	**15.3**	**1.92**

In Figure 6.9, we present some qualitative results of synthetic images generated with our method in the CelebAMask-HQ and ETH-XGaze data domains (top and bottom rows, respectively). We use randomly generated input latent and gaze vectors. We observe that the synthetic images look realistic and preserve the diversity of both training datasets.

Figure 6.9: Synthetic images generated with random latent vectors for the ETH-XGaze and CelebAMask-HQ data domains.

6.2.5 Facial Image Editing

In Figure 6.10, we see an example of two synthetic images generated with the models trained in the first and second stage, given the same latent vector w and gaze direction θ.

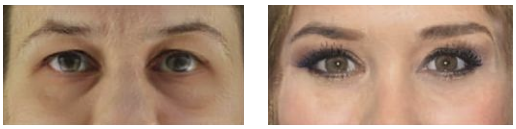


Figure 6.10: Synthetic images generated by models of stage 1 and stage 2 given the same latent vector w and gaze direction θ.

Next, we generate new augmented samples by editing a local latent vector w_k or the input gaze direction θ. Figure 6.11 shows some examples generated by the model trained in the first stage.

Figure 6.11: Synthetic image augmentations by input latent or gaze vectors' modification (ETH-XGaze data domain).

We generate the images in the first row by modifying the latent vector of the eyebrows. We observe that augmented samples vary on the eyebrows but preserve the other face components unaltered, including the gaze direction. We generate the second-row images' by sampling new local latent vectors for the nose generator and leaving the rest of the inputs the same as in Figure 6.10. The third row shows examples of face shape

modifications. The images in the last row are generated by modifying the input gaze direction θ. Figure 6.12 shows examples of the same kinds of augmentations generated with the model trained in the second stage (CelebAMask-HQ data domain).

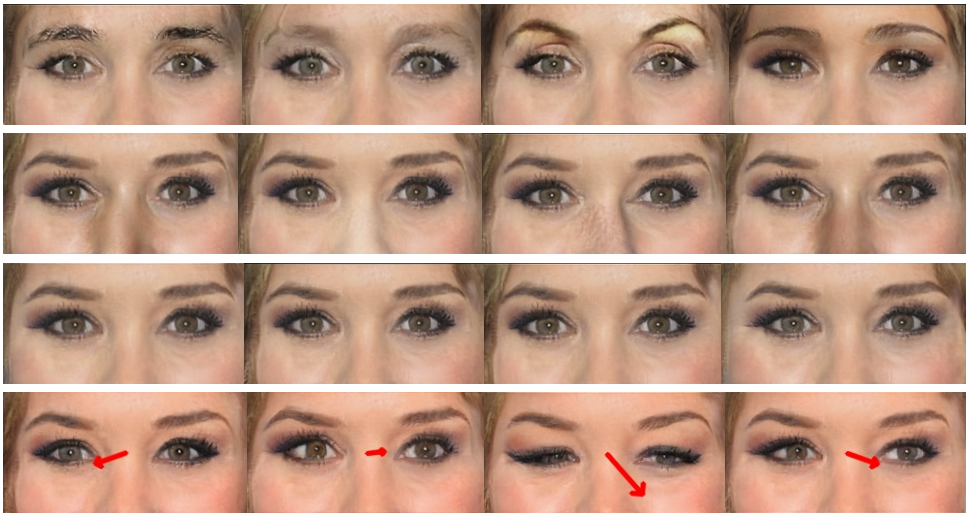


Figure 6.12: Synthetic image augmentations by input latent or gaze vectors' modification (CelebAMask-HQ data domain).

As shown in Figure 6.11 and Figure 6.12, the proposed GC-GAN model allows for fine-grained control image editing. In addition to image editing, it is also crucial when generating training data for DNNs. Training datasets for DNNs should have enough samples and be varied and balanced. Having control over the data generation is, therefore, essential.

6.2.6 Ablation Studies

This section presents different ablation studies regarding data generation for gaze estimation DNN training and generative model training.

6.2.7 Data Augmentation: in-domain synthesis

We evaluate the influence of a different number of generated GAN-based augmentations in the performance of the gaze estimation DNN. For that purpose, we repeat the experiment in Section 6.2.2, varying the number of synthetic samples in the training set.

Figure 6.13 shows the obtained mean error in the estimated gaze vector (in degrees) for the different training sets when adding within-domain augmentations.

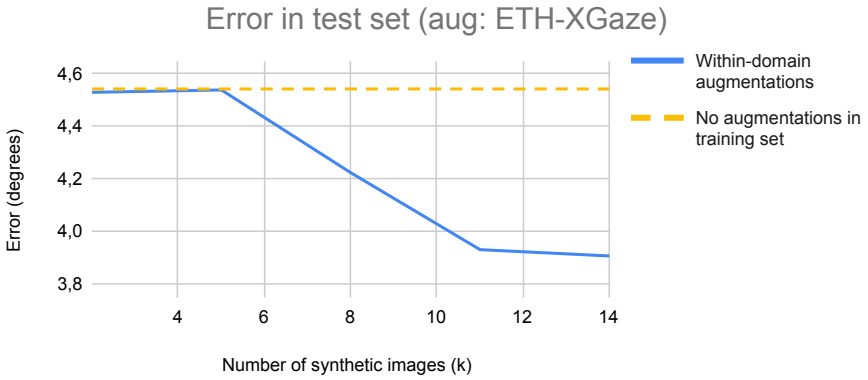


Figure 6.13: The mean error in the test set when using a gaze estimation DNN trained with a different number of synthetic samples from the ETH-XGaze dataset distribution.

When training with no synthetic samples, the DNN obtains an error of 4.53 degrees in the test set. When we add some synthetic samples to the training set, the test error is quite similar (4.52 degrees), but as we increase the number of synthetic samples (more than 5,000 images), the error decreases from 4.52 to 3.90 degrees. The error drop shows that the augmentations are helpful and the gaze is accurately preserved.

6.2.8 Data Augmentation: cross-domain synthesis

We increase each training set a 5% by selecting the images with the highest confidence. We compute this confidence score based on the MSE between the generated masks in both dataset domains. We repeat the DNN training with different sets of augmented data. Figure 6.14 shows the obtained results in the test set when adding within-dataset and cross-dataset augmentations to the training set. It can be seen that adding cross-dataset augmentations helps in reducing the error. Adding images from a new domain increases the accuracy of the DNN, but adding an excessive amount of images does not seem to necessarily lead to further improvement (the minimum test error is 3.86 degrees, and it is achieved with 9,000 synthetic images).

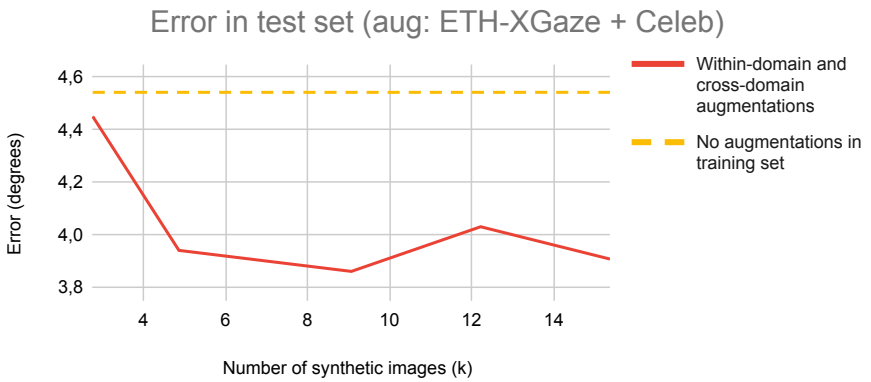


Figure 6.14: The mean error in the test set when using a gaze estimation DNN trained with a different number of synthetic samples from the ETH-XGaze and CelebAMask-HQ dataset distribution.

6.2.9 Stage 2: Modules to Freeze in the Gaze-aware Compositional Generator

There are several combinations of modules in the generator to be frozen for the second stage. We train the model after freezing different parts. Then, we qualitatively analyze the gaze preservation in each one by generating multiple pairs of images in both domains given a latent vector.

In Table 6.3, we observe that freezing only the shape-oriented layers in the GLG is not enough, the weights of the whole GLG need to be preserved. The Render Net R also contains some information related to the gaze direction, as it is required to freeze the 2 initial residual blocks to transfer the gaze. It is also possible freezing an additional block, but the image quality is much worse. Regarding the segmentation mask constraint (Section 6.1.6), it is a necessary but not sufficient condition.

6.2.10 Gaze Conditioning

The proposed GC-GAN design groups gaze-related and gaze-unrelated facial components' generators. This differentiation is possible thanks to the composition-based architecture. The motivation behind this design is the possibility of controlling different facial components independently and having a disentangled latent space for them and the gaze direction. We rely on the hypothesis that if some facial components, such as

Table 6.3: Gaze preservation between domains when different components are frozen, Gaze-aware Local Generator (*GLG*) or Render Net (*R*), and whether mask constraint is applied.

ID	*GLG*	*R*	$Mask_c$	Gaze preservation
1	shape only	✗	✓	No
2	✓	✗	✓	No
3	✓	1	✓	No
4	✓	2	✓	Yes
5	✓	3	✓	Yes
6	✓	2	✗	No

the nose, are independent of the gaze direction, their generator should learn without any gaze conditioning. To demonstrate whether grouping the gaze-related and unrelated generators helps, we retrain the GC-GAN with the ETH-XGaze dataset but modify the Local Generators so that all are gaze-conditioned and have the same architecture as a GLG.

Table 6.4: Pixel-wise mean absolute error between images generated given the same latent vector but different gaze directions. Errors per sample are averaged from images corresponding to 32 gaze directions.

Sample ID	All generators conditioned	Gaze-related generators conditioned
1	1.32	0.83
2	1.24	0.86
3	1.45	1.15
4	1.24	0.99
5	1.37	0.81
Mean	**1.33**	**0.93**

We generate random images with the initially proposed and totally conditioned models. Then we vary the input gaze direction of each sample to generate new images with 32 different gazes and analyze how the images change. Ideally, only the eyes should change due to the gaze, and the rest of the image should be maintained unaltered. Therefore, the differences between images should be minor. We measure the image differences when varying the gaze by computing the mean absolute error pixel-wise between the initially generated image and the image with the redirected gaze. We calcu-

late the mean error of the 32 gaze variations per sample and compare the error obtained by both models. Table 6.4 summarizes the results.

As can be seen, the error is lower when we group the generators into gaze-related and unrelated. This difference is also noticeable in the generated images. Figure 6.15 shows some examples of a sample when varying the gaze. Images in the top row are generated using the model with all Local Generators conditioned. It is noticeable that the gaze is entangled with the pose. When the gaze changes, the pose in some images changes too. Even the face texture changes in the last image. On the contrary, the images generated with grouped generators (bottom row) show a more disentangled behavior. The eyes' appearance changes when the gaze varies, but the rest of the image remains unaltered.

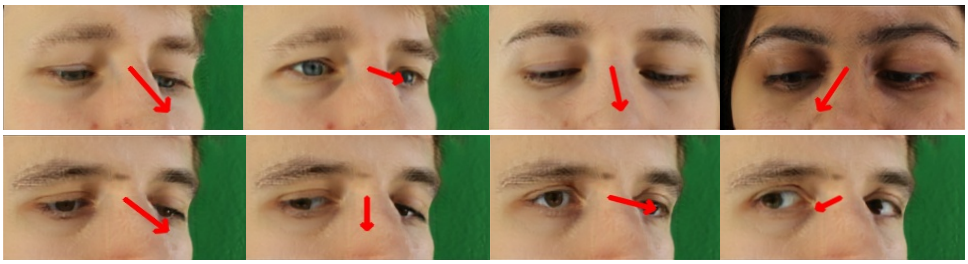


Figure 6.15: Synthetic images generated by varying the input gaze when all Local Generators are gaze-conditioned (top row) and when generators are grouped as in GC-GAN (bottom row).

I am always doing what I cannot do
yet, in order to learn how to do it.

Vincent Van Gogh

CHAPTER

7

Data-oriented Deep Neural Network Design: data modality

This chapter presents our research on adapting the DNN architecture design to the data modality (Chapter 1, Hypothesis 4). We work with point clouds from the LiDAR sensor for the automotive field. Over the last years, object detection has attracted much research attention in computer vision. 2D object detection has increasingly improved through advances in deep learning. However, many applications, such as advanced driving, need 3D object detection. 3D object detection is a less mature problem compared to 2D detection, but it is fundamental for perception systems in the automotive field. In the last years, the rapid progress of DNNs and the emergence of the LiDAR sensor for the automotive have promoted research in 3D object detection from LiDAR data.

Data captured by the LiDAR sensor are used to construct 3D point clouds. Different strategies have been explored to adequate these point clouds for object detection algorithms, including the conversion to image-like representations and processing point clouds directly. Handling point cloud data directly involves extra challenges like data volume and unstructured form. Image-based methods are left behind without a clear answer to the following questions: is it better to process point clouds directly? Is it possible to appropriately adapt 2D object detection architectures to the point cloud data to achieve similar results?

The research presented in this chapter has been published at the 13th International Joint Conference on Computational Intelligence (IJCCI) [190]. The publication was selected as the best student paper.

7.1 Methodology

We propose a DNN that takes as input the BEV representation of a point cloud and outputs the 2D oriented bounding boxes corresponding to the detected objects. The heights of the 3D boxes are estimated based on the highest point inside each box. Figure 7.1 shows the entire pipeline.

Figure 7.1: Proposed 3D object detection pipeline from LiDAR point cloud data.

7.1.1 Point Cloud to BEV Representation

Our approach first converts the point clouds to BEV images. Different works use varying configurations, resolutions, and amount of height channels for this step. We opt for keeping the height of the highest points in every voxel as in [191].

We consider a resolution of $0.1m$ for discretizing the point cloud in a grid of columns from a top-view perspective. We keep the height information in 3 channels to have an RGB-like data structure. Each column is divided into three voxels. Each voxel is of size $0.1m \times 0.1m \times 1m$, which results in a $700 \times 700 \times 3$ image (we consider a maximum distance of $70m$ in the longitudinal direction and $35m$ in both sides in the lateral direction). For every pixel, there will be three voxels. From all the points in each voxel, we only keep the highest point's height information and discard the rest. This process compresses the data but leaves enough information to detect objects.

7.1.2 Baseline Architecture

We base our network on Faster R-CNN [150] with a ResNet50 [28] backbone with some modifications related to the BEV representations used as the input data. The BEV images used in our research contain small objects that are challenging to detect. In addition,

we want to detect oriented bounding boxes rather than axis-aligned detections as done in [150]. Next, we describe the proposed changes.

Our first modification is using a Feature Pyramid Network-like architecture [161] to detect objects at the higher-resolution feature maps and not just at the final output maps of the ResNet.

Secondly, we adjust the stride of the third ResNet blocks from 2 to 1. This prevents the feature maps from being downsampled too quickly, which would cause a loss of detailed spatial information.

Thirdly, we remove the last ResNet block since we empirically found that it did not improve the results and allows for faster processing. The features extracted in this block probably do not contribute much because of the too-large receptive field of the pixels at this depth. The receptive field after the third ResNet block for each output value is already 195 pixels of the original BEV image, corresponding with almost a 20m x 20m area when using a 0.1m resolution. Although the context can help to detect the objects and the effective receptive field is smaller than the theoretical one [192], it seems to be already a large enough region.

We also use custom anchor boxes. The size of a car is consistent everywhere in the image because of the relation between their real size and their representation in the BEV image. This allows the designing of anchor boxes that best-fit cars, pedestrians, and cyclists.

In the second stage of the network, we use ROI-Align [193] to get 14×14 candidate ROIs, which are max pooled with a 2×2 kernel and fed to three fully connected layers with 1024 parameters each. After each layer, a ReLU activation function is used.

Lastly, the output detection bounding boxes are processed by the NMS method [150] so that unwanted bounding boxes are removed based on their classification score and the overlap between the boxes. If the DNN mistakes the classification scores of overlapping correct and incorrect boxes, this may lead to poor results. To make these classification scores more robust based on the LiDAR data peculiarities, we compute the percentage of non-empty pixels of each candidate box. The correctly oriented boxes mostly have a higher density of points, so we add this value to the classification score to help the NMS choose the correct detections.

7.1.2.1 **Bounding Box Regression**

The original Faster R-CNN network regresses axis-aligned bounding boxes (with no orientation). The bounding box regression is done in two stages. The first bounding box regression step is in the RPN. In addition, the network does a second bounding box regression at the end of the second stage. Both steps regress axis-aligned bounding boxes, which are represented by the center of the bounding box (x, y) and the dimensions of the bounding box (h, w). For accurate 3D detection, we need to detect oriented bounding boxes, so an additional parameter (theta) has to be added to regress the angle of the bounding box. We maintain the axis-aligned regions' regression in the RPN but regress five parameters in the second stage, including the box's orientation. The oriented bounding box parameters can be regressed individually with a loss function like L1, L2, or smooth L1. The underlying assumption is that regressing these parameters individually will lead to a global optimum for the bounding box estimation. For axis-aligned boxes, this works, but for oriented bounding boxes does not. [194] show the conflicting interest between the angle regression and the other regressions. A way to solve this problem is to directly minimize the loss for what you are evaluating, which is the IoU between ground truth and estimated bounding boxes. We use the IoU loss for the second stage [195].

7.1.2.2 **Loss Function**

Regarding the loss function, both network stages have a regression loss and a classification loss. Consequently, the total loss is a combination of four losses. The first stage is the RPN and uses the same loss objectives as the original Faster R-CNN. Four regression parameters are optimized with a smooth L1 loss, being (x, y) the center and (h, w) the dimensions of the bounding box. Once the difference between the ground truth parameter and the estimated one is computed (d), the smooth L1 loss is used according to the following Equation 1:

$$L1(d) = \begin{cases} 0.5d^2\sigma & \text{if } |d| < \frac{0.5}{\sigma} \\ |d| - \frac{0.5}{\sigma} & \text{otherwise} \end{cases} \tag{7.1}$$

where σ is a tuning hyperparameter. We use $\sigma = 3$ for the RPN network and $\sigma = 1$ for the head network, as in the Faster R-CNN implementation [150]. The RPN classification loss is binary cross-entropy with foreground and background boxes. We use an IoU

regression loss with five regression parameters instead of four [195] in the second stage. For the classification, we use a multi-class cross-entropy loss.

7.2 Network Extensions

One of the biggest challenges for the BEV-based object detection methods is distinguishing the smallest objects (e.g., pedestrians). When converting point clouds to BEV images, compression takes place where some data are lost. This is not a problem if there is enough data left to accurately perform detection of the target objects. When looking at the BEV images, this clearly seems to be the case for the cars. Many points are discarded, but they still show a clear structure, making detecting them possible. This is unclear for small classes such as pedestrians and the cyclists. The top view perspective only covers a small area of these objects. This might be alleviated with additional data that helps to distinguish these objects from others. For this reason, we test two network extensions. The first uses data already provided in the BEV image, and the second incorporates external data.

Each point in a LiDAR point cloud is represented by its 3D coordinates and its reflectance values. These values are related to the object class it represents and can provide meaningful information. Even if some values are stored in the BEV image, the specific pixel values are often lost along the network's operations [196]. The detectors consequently rely on the shape patterns in the BEV image more than on the extra information provided by the specific values. To test if these data could complement the patterns learned by the network, we add a skip connection from the input BEV map to the head network. This connection adds the raw values to the data received in the head network (feature maps of the candidate ROIs from the RPN) so that we guarantee that the values do not get discarded in the backbone and they can be used in the second stage of the network. In addition, we add two fully connected layers to the input values before concatenating them to the input data of the head network. These layers allow the network to learn a better fusion mechanism from the two feature spaces. Figure 7.2 shows the fusion scheme.

Based on the same fusion scheme, we could add information from a different BEV image instead of the heights-based one we use as input to the network. For instance, we

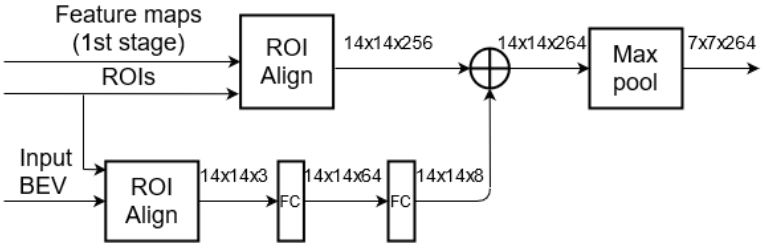


Figure 7.2: Fusion scheme to add information to the head network from a BEV image.

can add data related to another sensor, such as the camera. The camera captures information about the textures and appearances of the objects. The information provided by the camera pixel values about the object appearances is complimentary to the data from the LiDAR and might help gain robustness. To test this, we project the camera image pixel values from the camera to the point cloud. An example of the resulting point cloud is shown in Figure 7.3. Next, we compute the BEV representation, which contains pixel color information in each cell instead of height. Figure 7.4 shows an example of this kind of BEV. The new BEV image is added to the head network with the same fusion mechanism as the heights-based BEV image (Figure 7.2).

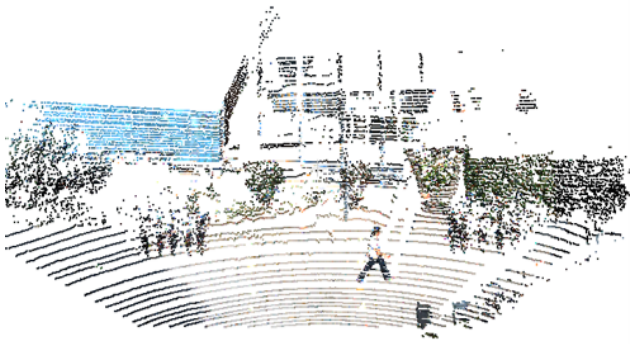


Figure 7.3: A LiDAR scan of the KITTI benchmark to which its corresponding camera image RGB values are projected.

7.3 Network Optimization

Using BEV images for 3D object detection reduces the amount of data to be handled. This translates into less processing time and makes the approach suitable for real-time

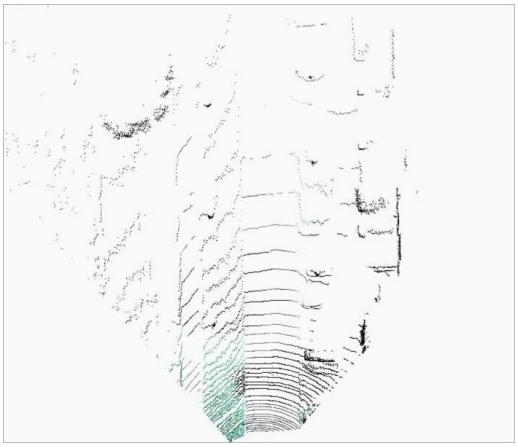


Figure 7.4: A BEV image which contains the projected image pixels color information instead of the heights of the points.

applications or collaborative environments. The Faster R-CNN network is more accurate than most single-stage object detection architectures but also slower. Consequently, we optimize the model after training to make the inference faster without reducing the architecture complexity.

The number of proposals generated in the first network stage varies in every image. However, the input batch of the second stage remains fixed. If the number of generated proposals is smaller than the batch number, the missing proposals must be simulated with empty ones. To avoid this processing, we divide the network into two parts corresponding to each stage and treat them as two independent networks running asynchronously. Thus, the first network processes batches of N images and generates a different number of proposals for each image, which are processed together in one batch by the second network. This way, the batch of the second network is optimized for the total number of proposals.

Using a dynamic batch for the second network may lead to time overheads, as the network needs to be re-adapted for the new size. To avoid this, the proposals are grouped in batches of a fixed size M and fed asynchronously to the network. Once the proposals are classified, they are returned again to their original batches according to the image they belong to. We optimize the models with TensorRT [197].

7.4 Experiments

We test our approach on the KITTI dataset, which has 7,158 training samples. We use a 50/25/25 split for training/validation/test sets. We convert the point clouds to $700 \times 700 \times 3$ images with a $0.1m$ resolution. Only the annotated area is considered, corresponding to the camera's field of view. We evaluate the network on the three classes of the benchmark, but we train two separate networks, one for cars and another for pedestrians and cyclists. For each network, we train and evaluate three variants: the baseline model, the baseline with the fusion of the input BEV in the head network, and the same model but with the fusion of the RGB data-based BEV. The car class has by far the most objects, which is the reason to train it apart. Otherwise, the network would be very biased in favor of this class. The KITTI results are evaluated by calculating average precision (AP) for every class's three difficulty categories (easy, moderate, hard). We consider an x, y, z range of [(0, 70), (-35, 35), (0, 3)] meters, respectively, for the car class and [(-35, 35), (-35, 35), (0, 3)] meters for pedestrians and cyclists. As the benchmark proposes, we use an IoU threshold of 0.7 for cars and 0.5 for pedestrians and cyclists.

Regarding the training, our loss function is optimized with stochastic gradient descent with a momentum of 0.9. An initial learning rate of 0.003 is used with decay steps at 35 and 42 epochs, with a factor of 0.1. The network is trained for 70 epochs with a batch size of 1 and a mini-batch size of 256. The mini-batch corresponds to the number of region proposals that are fed to the second stage. Weight decay is set at 0.0001. The entire baseline model has 19M learnable parameters. The backbone is initialized with pre-trained weights from ImageNet, whereas the head network is trained from scratch. All tests are done on a single Nvidia Tesla V100 GPU. All other hyperparameters are the same as those used for the Faster R-CNN.

7.5 Results and Discussion

Table 7.1 shows the AP obtained by our method compared to other state-of-the-art approaches for car, pedestrian, and cyclist classes. In the same way as in [8], we validate our results on our split validation dataset, whereas the others are validated on the official KITTI test set. Our models outperform most works with only LiDAR data for the three classes and difficulties. Our proposed CNN architecture achieves a higher AP than

other BEV-based pipelines, such as [153] and [8]. These models mainly differ in the proposed backbone architecture and the regression loss. In addition, our work also surpasses the approaches which process raw point clouds, such as [158] and [159]. Only the method in [155] presents a slightly better accuracy for cars in the hard category. This shows that using a BEV-based approach is not a limitation for obtaining a high detection accuracy.

Method	Modality	BEV	Car			Pedestrian			Cyclist		
			Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
PIXOR [153]	LiDAR	Yes	81.70	77.05	72.95	N/A	N/A	N/A	N/A	N/A	N/A
Complex-YOLO [8]	LiDAR	Yes	85.89	77.40	77.33	46.08	45.90	44.20	72.37	63.36	60.27
PointPillars [159]	LiDAR	No	88.35	86.10	79.83	58.66	50.23	47.19	79.14	62.25	56.0
SECOND [158]	LiDAR	No	88.07	79.37	77.95	55.10	46.27	44.76	73.67	56.04	48.78
Joint 3D [155]	LiDAR	No	90.23	87.53	**86.45**	N/A	N/A	N/A	N/A	N/A	N/A
F-PointNets [154]	RGB+LiDAR	No	88.16	84.02	76.44	**72.38**	**66.39**	**59.57**	81.82	60.03	56.32
Baseline	LiDAR	Yes	**97.3**	**89.6**	85.0	54.1	52.5	51.0	73.2	60.7	58.7
Baseline + input BEV	LiDAR	Yes	93.7	88.7	84.4	52.9	51.7	49.1	80.4	66.6	64.2
Baseline + cam.-based BEV	RGB+LiDAR	Yes	94.6	87.1	83.3	56.1	54.0	50.7	**82.6**	**73.1**	**70.5**

Table 7.1: Our proposed approach compared to other state-of-the-art methods on our KITTI validation set based on the AP. Note that our work and [8] are validated on our split validation dataset, whereas all others are validated on the official KITTI test set.

Regarding the network extension tests, no improvement is obtained for the car class. This may be because the cars already had enough points and information to be accurately detected in the BEV image. This is not the case of smaller objects such as pedestrians and cyclists. Adding the skip connection with the input values to the head network boosts the accuracy of the cyclist class. Regarding the addition of the camera-based BEV image, it increments the accuracy of the pedestrian and cyclist detections. When comparing this network to another multimodal approach like [154], we observe that the accuracy obtained by Frustum PointNets is higher for the pedestrian class but not for the cars and the cyclists. This may happen because of the feature space where the detections are estimated. Frustum PointNets detect the object proposals on the RGB image, and then, computing a 3D viewing frustum, they predict the final 3D box on the point cloud. Therefore, the accuracy is closely related to the 2D detector for the RGB image. This differs from our approach, which projects the RGB values to the point cloud instead of detecting the objects in the RGB image. For small classes in the BEV representation, the camera detections seem more reliable, even if the projected information boosts the BEV-based detection accuracy.

Figure 7.5 shows some qualitative results of the baseline model on the BEV images. The left image corresponds to the model trained on cars, whereas the right image shows

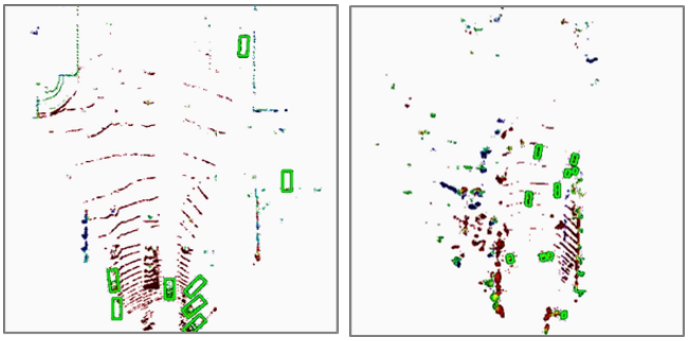


Figure 7.5: Qualitative results on our KITTI validation set. Detections are shown on the BEV representation for the car class (left) and the pedestrian class (right).

the detections from the pedestrian and cyclist model (only pedestrians are present in the image). The mean inference time per scan (entire pipeline) is 30 ms. Figure 7.6 shows the same detections projected to the camera images to visualize the results better.

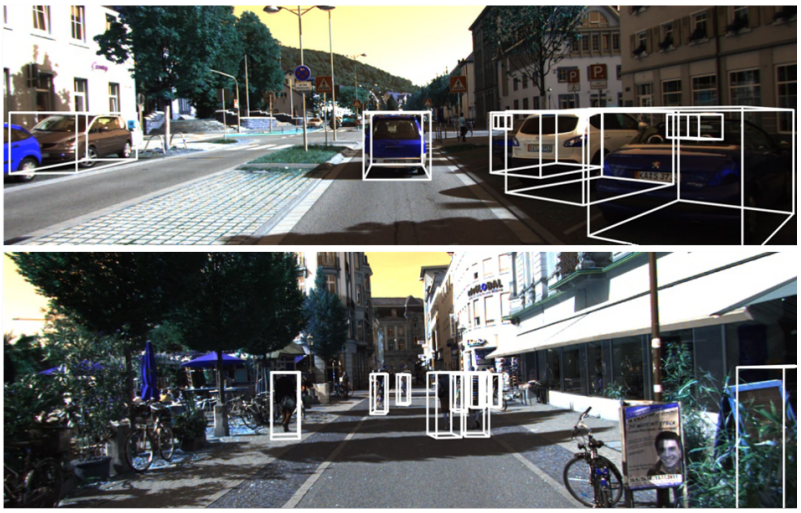


Figure 7.6: Qualitative results on our KITTI validation set. Detections are shown on the BEV representation (left) and projected to the camera image (right).

The optimization applied to the trained DNNs allows scaling the detection task to process different BEV images at the same time. The batch-oriented optimization may be interesting for processing the data from the different LiDARs of a vehicle or for an automotive ground truth annotation application that requires handling batches of scans. Compared to a 3D CNN, the computational cost of a BEV-based CNN is lower. The computational complexity of a 3D CNN grows cubically with the voxel resolution [158]. In

addition, the BEV representation reduces the target raw point cloud (up to millions of points) to a 3-channel image covering the area of interest and filtering many points that are not significant. The biggest challenge for BEV-based methods is the detection of the objects covering a small area in the BEV image.

We keep moving forward, opening new doors, and doing new things, because we're curious and curiosity keeps leading us down new paths.

Walt Disney

CHAPTER

8

Data-oriented Deep Neural Network Design and Training: data source

As seen in Chapter 1, abundant and well-distributed data are essential to make DNNs learn appropriate pattern recognition features and have enough generalization ability. Chapter 4, Chapter 5, and Chapter 6 show different use cases where the use of synthetic data reduces the effort of generating varied and annotated data. However, synthetic data usually present a domain gap with real-world samples, which limits the final model's accuracy or generalization ability. This gap can be reduced with domain adaptation techniques. This chapter proposes a methodology to adapt a DNN design and training to the training data source (Hypothesis 5, Chapter 1). We start from the work presented in Chapter 5 in the context of digitalized on-demand aircraft cabin readiness verification with a camera-based system. We incorporate synthetic images generated in the simulated environment together with real captured samples to train a DNN for passenger seat analysis.

The research presented in this chapter has been published in the SN Computer Science journal [198].

8.1 Methodology

In order to evaluate the impact of adapting a DNN design and training to the training data source, we train a DNN with real and synthetic data. The objective of the DNN is to analyze the visual appearance of an aircraft seat to determine if it contains luggage correctly or incorrectly placed for the TTL requirements (Chapter 5). Figure 8.1 shows an example of the image ROIs that are classified by the DNN, from the perspective of a camera installed above the seats.

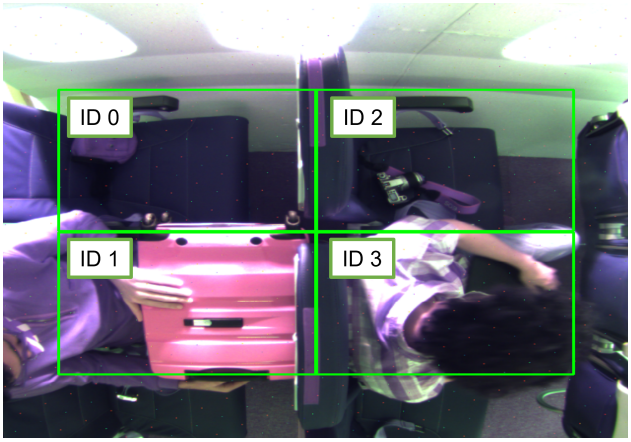


Figure 8.1: Image from a camera installed on top of the aircraft seats and ROIs classified by the DNN.

8.1.1 Real Data Acquisition

We consider the same camera-based setup designed in Chapter 5 (cameras on top of the seats and the corridor). Once the virtual camera setup is done, it can also be replicated for real data acquisition.

When capturing real data for video surveillance systems, it is common to get some very similar or redundant images, especially in scenarios where events happen from time to time and stay the same the rest of the time. The advantage of capturing data continuously is that skipping consecutive frames may reduce data redundancy. However, this process can exclude interesting samples as it only relies on time consistency but does not consider the content of the images. We propose an alternative process

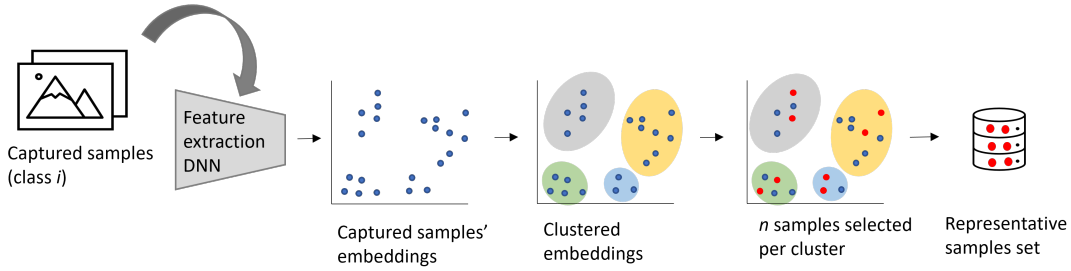


Figure 8.2: Clustering-based pipeline to select samples of interest from all the captured data of each target class.

for selecting varied samples in Figure 8.2. The goal of the pipeline is to choose specific samples of interest to train the target DNN.

For that purpose, we do the following steps:

- Use a pretrained generalist DNN's backbone as a feature extractor to encode all the images in image embeddings (m-dimensional vectors).
- Cluster the embeddings using a clustering algorithm, such as K-Means.
- Select a specific number of n samples randomly from each cluster to generate the new subset of images.

The optimum number of clusters to be used in a clustering algorithm such as K-Means can be computed using the Elbow method. This method consists of plotting the sum of squared distances from each point to its assigned center as a function of the number of clusters, and selecting the elbow of the curve as the optimum number of clusters. The process is applied to all the images of a specific target category i so that a balanced dataset is generated. The number of samples per cluster depends on the target size of the dataset.

Once the samples are selected, they are combined with the rest of the synthetic data to complete the final dataset.

8.1.2 Synthetic Data Generation

For synthetic data, we follow the methodology presented in Chapter 5 to generate varied images based on the simulated 3D environment. Figure 8.3 shows some examples of the generated images from different cameras' perspectives.

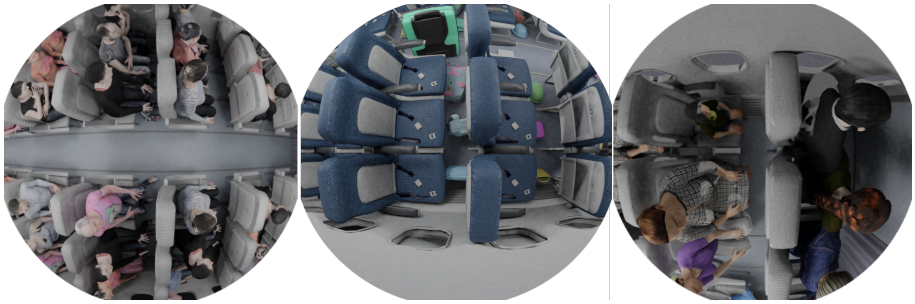


Figure 8.3: Examples of synthetically generated images with different appearances and objects' configurations.

8.1.3 Deep Neural Network Design

To analyze how a DNN performs with the generated synthetic and real data, we train the model EfficientNet-B0 [29] to classify images as correct or incorrect (e.g., cabin luggage correctly or incorrectly placed).

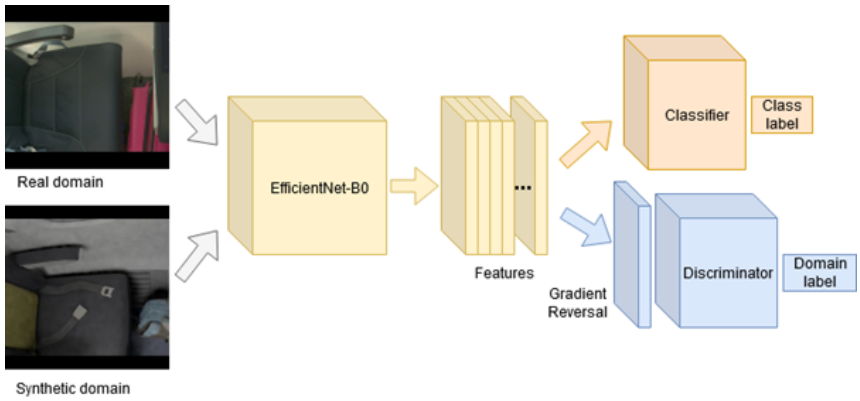


Figure 8.4: DNN architecture for learning domain-invariant image features.

To minimize the domain gap between the synthetic and real domains during the training, we include the domain adaptation technique of [166]. The idea is to learn to extract domain-invariant features from the images using a Domain Adversarial Neural Network (DANN), of which an overview schema is shown in Figure 8.4. The DANN architecture contains three main parts: the feature extractor, the main classifier, and the discriminator.

The feature extractor is the DNN responsible for extracting the features that represent the images. We use EfficientNet-B0 as the feature extraction backbone, as in

Chapter 5. The main classifier of the architecture classifies the images in the correct or incorrect category and contains a fully connected layer and a softmax activation. The discriminator is an adversarial domain classifier whose loss is maximized using a gradient reversal layer [166]. It tries to classify the domain an input sample belongs to using two fully connected layers' output and a softmax activation. The fully connected layers have 1,280 and 2 outputs, respectively. The discriminator's role is to encourage the feature extractor to find domain-invariant latent representations. The gradient reversal layer is key in obtaining this, as it acts like an identity function during the forward propagation but during the backpropagation, it multiplies its input by -λ. The output of this layer leads to the opposite direction of gradient descent with respect to the classification loss of the domain classifier. Consequently, during learning, this loss is maximized.

8.2 Experiments

8.2.1 Dataset Generation

We replicate a cabin mock-up with the chosen camera configuration to capture some real images. We simplify the setup to a single camera above the seats.

We prepare a recording protocol for a group of 20 participants. Each participant is asked to place a piece of cabin luggage correctly or incorrectly to generate different scenes. We also ask them to act naturally and stay calm so artificial situations with a lot of movement are avoided. Otherwise, blurry images which belong to unnatural behaviors would be captured. As a disadvantage, staying calm reinforces the possible redundancy between the captured frames. Figure 8.5 shows an example of this redundancy in the image crop of one of the seats. A participant stays calm with a suitcase between his legs for some seconds. We can observe that consecutive frames look very similar.

We capture 24,000 images, half of them belonging to situations where the luggage is incorrectly placed. To avoid including many redundant samples in the training dataset, we apply the clustering-based image selection pipeline (Section 8.1.1). We use the classification model EfficientNet-B0, already pretrained on the ImageNet dataset, without the last classification layers as a feature extractor. We choose EfficientNet-B0 for the

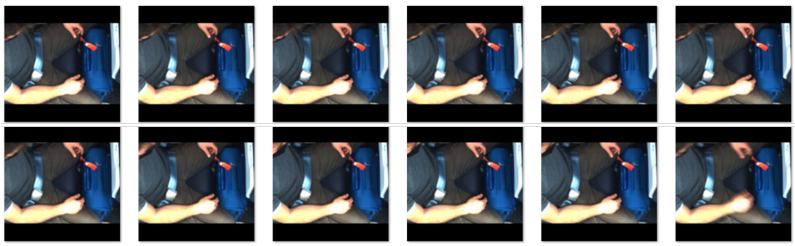


Figure 8.5: Similar captured images of a participant staying calm with a suitcase between his legs.

backbone as it obtains high-accuracy results with better efficiency than other alternative state-of-the-art DNNs. In this context, efficiency is also a key factor, as the system should be energetically sustainable to go on board. We resize input images to resolutions of 300x300, as the minimum resolution that allows visualizing smaller objects with sufficient size for the classifications. This way, each input image with a size of 300x300 pixels is embedded as a 1280-length feature vector. Then, we use the Elbow curve for both target categories (correct and incorrect situations) to choose the number of clusters for applying the K-Means algorithm. This results in 300 clusters. From each cluster, we randomly select ten samples. We combine these images with synthetic images.

For the synthetic dataset generation, see Chapter 5, Section 5.2.

8.2.2 **Real Samples Selection**

To analyze whether the images' redundancy problem affects the DNN training and, in that case, how much our clustering-based approach helps mitigate it, we do the following tests. We train the EfficientNet-B0 image classifier with different datasets but with the same training configuration to analyze the impact of varying data sampling strategies. We define an ROI for each seat, which will be classified as correct or incorrect by the DNN, depending on the correctness of the cabin luggage. We generate different datasets following each of the subsequent strategies:

- Use all the captured data.
- Use a random subset of the recorded data.
- Use a subset of the recorded data, skipping some images when selecting them based on the order they were captured.

- Use a subset of the recorded data based on the proposed clustering algorithm that helps us select non-redundant images.

Then, we add the real samples to a subset of 4,000 synthetic samples for each option to complete the training dataset. We use these datasets in the following training experiments.

8.2.3 Deep Neural Network Training

In order to see how a DNN performs when trained with different data sampling strategies for real data, and with domain adaptation to combine both real and synthetic samples, we do the following experiments.

Sampling strategies. To see the impact of the different sampling strategies, we train the DANN with the different generated datasets (Section 8.2.2). We initialize the DANN backbone with pretrained weights on the ImageNet dataset and fine-tune it with our datasets. The classifier and the discriminator are trained from scratch. We train each model for 50 epochs with a batch size of 40, and the RMSprop optimizer [178]. We split the data into training, validation, and test sets. We separate the captures of a recording session to use them in the validation and test sets so that neither the participants nor the lighting conditions used in training are repeated in the validation or test sets, which contain 2,000 and 500 images, respectively. Table 8.1 shows the results.

Sampling Method	**Test Accuracy**
All images	86.9%
Random subset of images	86.7%
Skipping consecutive frames	92.2%
Selecting images based on the clustered embeddings	95.3%

Table 8.1: Comparison between achieved DANN accuracy in the test set for different sampling strategies. Synthetic images are added to the real ones in all the trainings.

As can be seen, using all the captured images does not lead to the best results. Randomly selecting a subset of the images (30% of the images) does not help either, resulting in the lowest accuracy (86.7%). The image redundancy is confirmed as a problem because when we skip 10 consecutive frames, we obtain an accuracy boost of almost 6% compared to using all the samples. Selecting the training samples with the proposed

approach improves an additional 3.1% of the final accuracy. Consequently, selecting the images based on their content helps to improve the training dataset.

Synthetic and real data combination. We compare the proposed DANN with other state-of-the-art methods that combine synthetic and real domain data using different strategies. No works are applied to the same task, so we adopt the methods followed by other authors for different tasks and apply them to our DNN training. We summarize the results of the experiments in Table 8.2. The authors in [103, 163] train their DNN on the synthetic domain and then fine-tune the model on the real domain in a second step. We replicate this process with our data. We train the classification DNN with no domain adversarial branch for this experiment. We use the EfficientNet-B0 backbone as in our proposed method and maintain the same training parameters configuration to make both models comparable. The model achieves an accuracy of 93.8%, which is 1.5% lower than our proposed methodology. We additionally compare our method with [79, 162], which jointly trains with both domains' images by using mini-batches from the source and target domain. We follow the same data strategy and train the classification DNN with no domain adversarial branch. We use the EfficientNet-B0 backbone and maintain the same training parameter configuration. The model achieves an accuracy of 92.2% in the test set, which is 3.1% lower than our proposed method.

Synthetic and real data combination strategy	**Test Accuracy**
Pretrain with synthetic data, fine-tune with real data [103, 163]	93.8%
Train with real and synthetic mini-batches [79, 162]	92.2%
Train with a domain adversarial strategy (ours)	95.3%

Table 8.2: Comparison between achieved DNN accuracy in the test set for different data combination strategies.

The goal of the adversarial branch of the DANN model is that the extracted features do not discriminate between the source domains. That would mean that the domain gap is minimal. In order to qualitatively analyse this domain gap, we extract the features of all the training images using the trained DANN for the seat with ID 0 and visualize them. We also repeat the same DANN training but without domain adaptation (no discriminator) to see the difference. The features extracted have a dimension of 1,280 (features before the heading networks), so we apply a Principal Component Analysis

(PCA) to reduce their dimensionality to 3. PCA applies an orthogonal linear transformation that transforms the features into a new coordinate system organized by their variance content in the form of principal components. Applying the PCA to our features, we reduce them to a 3-dimensional space where we can visualize them, as shown in Figure 8.6.

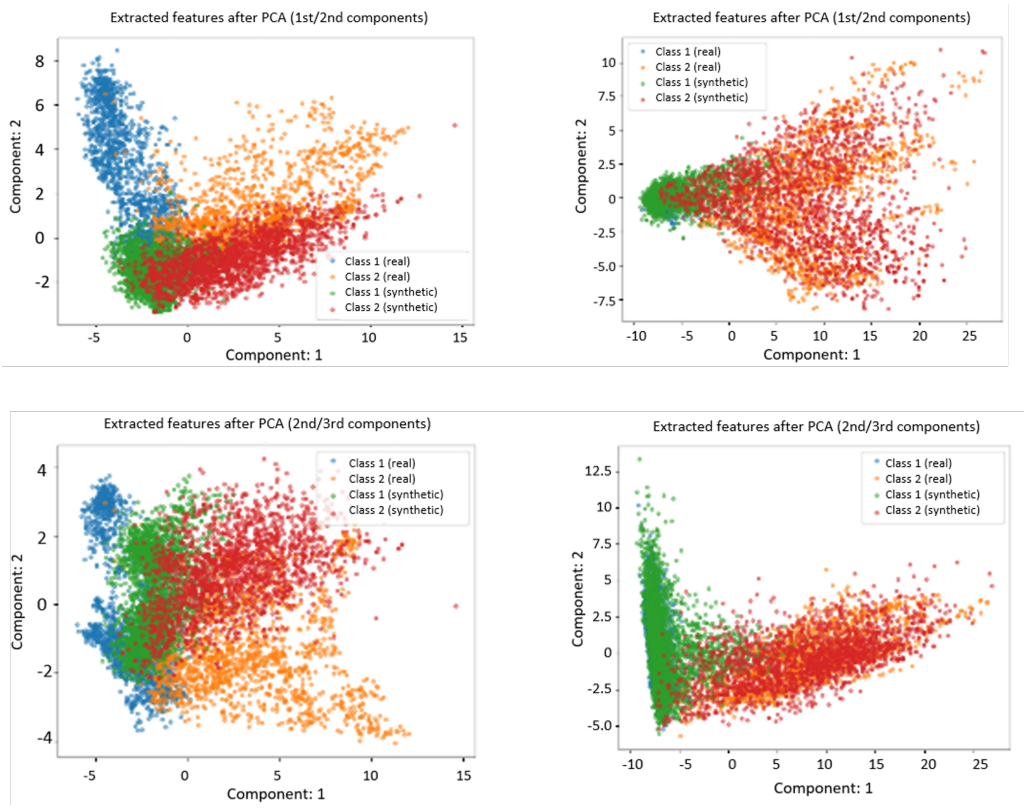


Figure 8.6: Extracted features from training images after PCA when the DNN is trained without and with domain adaptation (left and right, respectively).

Graphs on the left belong to the model trained with no domain adaptation, whereas graphs on the right are computed with the DANN. The top images show the 1st and 2nd components, and the bottom images show the 1st and 3rd components. Each processed sample is represented as a colored point. The color depends on the class of the image (correct luggage or incorrect) and the domain it belongs to. Real samples are drawn in blue (correct class) and orange (incorrect class). Synthetic samples are shown in green (correct class) and red (incorrect class). Ideally, the samples that belong to the cor-

rect class from both domains should be mixed together, and the same for the samples belonging to the incorrect class. If this is met, the features are not domain dependent.

When the model is trained without domain adaptation, we can see that the data distribution is sparse and has no clear pattern. The real and synthetic samples' distributions are quite differentiable, and the quality of the distributions seems quite poor. Real correct samples and incorrect synthetic samples appear quite mixed too in the different projections. The effect of domain adaptation on the distribution of the data is remarkable. We can see in the right projection images that the distributions of the correct real and synthetic samples and the distributions of the incorrect real and synthetic samples are much closer, even aligned, and it is quite difficult to differentiate between them.

Last, Figure 8.7 shows some correctly and incorrectly classified test images using our proposed DANN for the synthetic and real data domains. In the first column, we show two samples of situations with cabin luggage incorrectly placed, which are correctly classified as so. There is no cabin luggage in the second and third column images, but the last ones are incorrectly classified. We think this is probably because of these images' dark or blurry characteristics. In the last column, we show two more samples of incorrect classifications. In both images, cabin luggage is barely visible on the egress. This is the most challenging situation for the classification, as the passenger or the seat might remarkably occlude the objects. We think temporal analysis of the consecutive frames' classification might help in these situations.

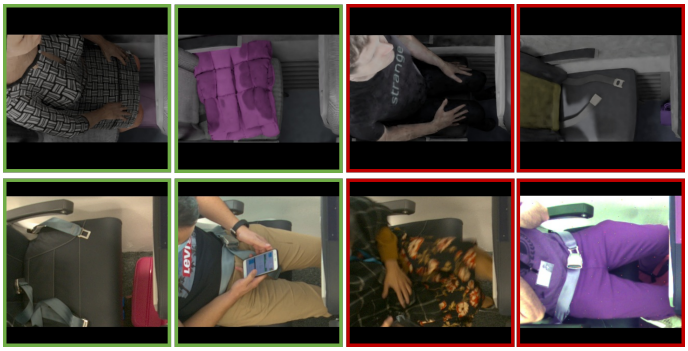


Figure 8.7: Some examples of correctly (green) and incorrectly (red) classified images using the proposed DANN model. Images in the top row are synthetic, and images in the bottom row are real.

Part IV

Conclusions

The journey not the arrival matters.

T.S. Eliot

CHAPTER

9

Conclusions

This thesis studies key factors in data-centric training and design of DNNs for perception systems. Data play a crucial role in deep learning-based models and, consequently, also how data are obtained and handled in a DNN is of utmost importance. More specifically, our research has validated the following hypotheses:

***Hypothesis 1**: Artificially generated samples using 3D models and environments are valid for training DNNs.*

In the work presented in Chapter 4, we analyze, reproduce, and combine different synthetic data generation strategies to create an accurate dataset for training an object detection CNN. In particular, we focus on generating a dataset to train a people detection model from omnidirectional cameras in large spaces, such as airports or train stations. This task is a real-world computer vision challenge that lacks available training data.

We have proposed a methodology to design data generation and evaluate the impact of adopting different strategies when generating the data. The results demonstrate the importance of properly modeling the sensor effects, such as distortion. More specifically, we apply a distortion rectification algorithm that matches the distortion obtained by a real and a virtual camera. More accurate results are obtained when the simulated

distortion matches the real one. Our tests also show that not any domain randomization method is valid for generating a good synthetic dataset. Additionally, the adoption of various accessories and accurate lighting configurations brings the rendered images closer to reality, and this also results in more accurate DNN performance. Thus, considering the results of this study can minimize the domain gap between real and synthetic data and consequently improve DNN models' generalization capabilities. We have published all generated and evaluated datasets for the benefit of the research community.

Moreover, we demonstrate that combining real and synthetic images produces superior results compared to using only real data. Thus, synthetic data generation is not only a way to avoid the laborious task of collecting and annotating real data, but also an effective means to enhance DNN performance. We generate a suitable dataset for DNN training and develop a people detection model using omnidirectional cameras, which achieves an AP of 82% in a real-world evaluation scenario. The findings of this study can be applied to other 3D computer graphics software and computer vision tasks.

***Hypothesis 2**: A simulated 3D environment can help define the required data and sensor setup for vision-based perception systems and generate appropriate data for DNN training with a high level of automation.*

Chapter 5 presents a methodology for creating simulated 3D environments to configure and train multi-camera systems for various surveillance contexts with minimal effort. This approach assists in designing an appropriate camera system that covers the target use cases and avoids expensive system setup errors. After configuring the camera, our method allows for generating a diverse set of situations that are relevant for training DNNs with suitable synthetic data. For successful DNN training, the generated situations should be balanced, and this is ensured by controlling the content of the data with input configuration files and a user-friendly fast scene parameterization. To ensure greater data variability, the scenes are designed to have certain randomness in object spatial locations and appearances.

We present a practical implementation of our methodology in the form of a camera-based smart sensing system for digitized on-demand aircraft cabin readiness verification. Using our proposed environment, we design a 39-camera-based system and

generate a training dataset of 40,755 images, containing 71,966 object instances in the cabin environment. We follow a data balancing strategy to guarantee the suitability of the generated data based on the configuration files. Compared with other state-of-the-art approaches, our methodology offers generality, flexibility, and a multi-camera setting. We evaluate the effectiveness of our approach by comparing it to other methods and training a classification DNN using a subset of the generated dataset jointly with real images. Our results demonstrate that incorporating our synthetic samples into the training dataset boosts the model's accuracy when tested on real images. Consequently, our methodology generates images that are suitable for training DNNs.

***Hypothesis 3**: Artificially generated samples using generative models are valid to train DNNs with no need for a simulation environment.*

In Chapter 6, we introduce a new approach to create annotated synthetic data by leveraging the advantages of various labeled and unlabeled data sources through a generative adversarial network. We use this technique to produce synthetic data for the gaze estimation task. First, we train a gaze-aware compositional GAN to create realistic synthetic images with specific gaze directions in a labeled data domain. Then, we transfer this model to an unlabeled data domain to exploit the variance these data provide. Our experiments demonstrate that this method can enhance existing annotated data for gaze estimation DNN training by generating within-domain and cross-domain augmentations that improve the DNN accuracy. We also demonstrate that our work can be used for additional applications such as facial image editing and gaze redirection.

***Hypothesis 4**: Adapting the DNN architecture design to the data modalities improves the results obtained by predefined and pretrained models.*

In Chapter 7, we propose a two-stage neural network architecture for 3D object detection from LiDAR point cloud data, based on bird's eye view (BEV) representations that avoid processing the entire raw point clouds. Our approach extends an image-detection architecture to point clouds and introduces two network enhancements that improve the detection accuracy for the most challenging object classes by incorporating BEV data from the same sensor or an additional modality to the head network. We

evaluate our method on the KITTI BEV benchmark and show that it achieves state-of-the-art results, outperforming recent approaches based on both BEV images and raw point clouds. Our findings suggest that BEV-based detection research should not be replaced with more complex point cloud-based detectors, as BEV representations provide a lightweight scene representation that is suitable for cooperative scenarios or for fast inference. The key to achieving these results is adapting the DNN architecture to the point cloud BEV representation.

***Hypothesis 5**: Adapting the DNN architecture design to the data domains improves the results obtained by predefined and pretrained models.*

In Chapter 8, we introduce a methodology to effectively combine real and synthetic data for DNN training. This approach is demonstrated in the context of aircraft cabin readiness verification using a camera-based smart sensing system. Combining real data with synthetic data can help reduce the domain gap between the two types of data. However, captured real data may contain high redundancy that could negatively impact DNN accuracy. To mitigate this, our method uses feature clustering to avoid including redundant samples in the training dataset. As for synthetic data, we generate them using the methodology outlined in Chapter 5, and incorporate an adversarial domain branch to facilitate the learning of domain-agnostic features during training. By incorporating synthetic samples in the training dataset, we demonstrate that our method can improve the accuracy of DNN models when tested on real images. Finally, we highlight the importance of adapting the neural network architecture and training approach to the specific data sources used to achieve the best possible results.

9.1 Future Work

Based on the results and findings of the research presented in this thesis project, we outline some potential avenues for future research.

Data are essential to training machine learning-based perception models, but obtaining enough quality data is still one of the major challenges in most perception systems' development. As shown in the current thesis, synthetic data are a promising alternative. Data obtained with GANs are realistic, can be transferred to a specific

domain, and do not require setting up a simulation environment. Consequently, generative models might be a better solution in some situations than simulated data using 3D environments. The suitability of the approach depends on the target data. Suppose it is, for example, a complex scenario, such as the aircraft in Chapter 5, or a multimodal setup, such as a driverless vehicle. In that case, it might be more effective to use a 3D simulation environment. In a simulation environment, agents can interact with each other, and it is possible to have more control over the spatial configuration of the scenario. If the target data are oriented to an instance (e.g., person, object) in a relatively controlled setting (e.g., gaze estimation, driver monitoring, pose estimation), then using GANs for data generation might be more effective. In future work, simulated 3D environments could benefit from generative models. Emerging DNN techniques to generate 3D models may speed up the manual process of gathering or designing the 3D assets required for any virtual environment. Generative models could also be used to augment the data generated in a simulated environment, alleviating the time that heavy rendering processes might involve, minimizing the domain gap, or transferring the variance learned by a generative model from real data to the rendered images globally or locally to specific instances in the image. Generative models are not limited to the GANs, recent proposals on diffusion models seem promising, and more research is required to see how far they can take us.

Another exciting future research is extending to new computer vision tasks the work done on leveraging unlabeled data and limited annotations to learn GANs. This could allow exploiting already existing data in large-scale datasets. These datasets provide plenty of varied data but usually focus on specific tasks and, consequently, contain annotations only for those tasks. Combining strengths from different datasets might help develop more robust models with better generalization capability. It would also be interesting to explore the the use of GANs with other data modalities, such as point clouds. Research in this field is still scarce, but DNN training would benefit from it.

Regarding model design, future research includes working on model explainability. Being able to interpret or understand why a model makes a specific prediction will help us to design better datasets and, consequently, models. In addition, it would prevent negative artifacts such as bias. Model explainability can also be beneficial to analyze the role of different data sources in the output of multimodal DNNs and redesigning the architectures accordingly.

Part V

Appendix

APPENDIX

A

Publications

List of publications during the presented research period:

A.1 Learning Gaze-aware Compositional GAN from Limited Annotations [*under review*]

Title: Learning Gaze-aware Compositional GAN from Limited Annotations

Authors: **Nerea Aranjuelo**, Siyu Huang, Ignacio Arganda-Carreras, Luis Unzueta, Oihana Otaegui, Hanspeter Pfister, Donglai Wei

Proceedings: The 23rd International Joint Conference on Artificial Intelligence

Year: 2023

Abstract: *Gaze-annotated facial data is crucial for training deep neural networks (DNNs) for gaze estimation. However, obtaining these data is labor-intensive and requires specialized equipment due to the challenge of accurately annotating the gaze direction of a subject. In this work, we present a generative framework to create annotated gaze data by leveraging the benefits of labeled and unlabeled data sources. We propose a Gaze-aware Compositional GAN that learns to generate annotated facial images from a limited labeled dataset. Then we transfer this model to an unlabeled data domain to take*

advantage of the diversity it provides. Experiments demonstrate our approach's effectiveness in generating within-domain image augmentations in the ETH-XGaze dataset and cross-domain augmentations in the CelebAMask-HQ dataset domain for gaze estimation DNN training. We also show additional applications of our work, which include facial image editing and gaze redirection.

A.2 Leveraging Synthetic Data for DNN-Based Visual Analysis of Passenger Seats

Title: Leveraging Synthetic Data for DNN-Based Visual Analysis of Passenger Seats

Authors: **Nerea Aranjuelo**, Jose Luis Apellaniz, Luis Unzueta, Jorge Garcia, Sara Garcia, Unai Elordi, Oihana Otaegui

Journal: SN Computer Science

Year: 2022

DOI: `https://doi.org/10.1007/s42979-022-01453-x`

Abstract: *Deep neural network (DNN)-based vision systems could improve passenger transportation safety by automating processes such as verifying the correct positioning of luggage, seat occupancy, etc. Abundant and well-distributed data are essential to make DNNs learn appropriate pattern recognition features and have enough generalization ability. The use of synthetic data can reduce the effort of generating varied and annotated data. However, synthetic data usually present a domain gap with real-world samples, that can be reduced with domain adaptation techniques. This paper proposes a methodology to build simulated environments to generate balanced and varied synthetic data and avoid including redundant samples to train classification DNNs for passenger seat analysis. We show a practical implementation for detecting whether luggage is correctly placed or not in an aircraft cabin. Experimental results show the contribution of the synthetic samples and the importance of correctly discarding redundant data.*

A.3 Designing Automated Deployment Strategies of Face Recognition Solutions in Heterogeneous IoT Platforms

Title: Designing Automated Deployment Strategies of Face Recognition Solutions in Heterogeneous IoT Platforms

Authors: Unai Elordi, Chiara Lunerti, Luis Unzueta, Jon Goenetxea, **Nerea Aranjuelo**, Alvaro Bertelsen, Ignacio Arganda-Carreras

Journal: Multidisciplinary Digital Publishing Institute

Year: 2021

DOI: `https://doi.org/10.3390/info12120532`

Abstract: *In this paper, we tackle the problem of deploying face recognition (FR) solutions in heterogeneous Internet of Things (IoT) platforms. The main challenges are the optimal deployment of deep neural networks (DNNs) in the high variety of IoT devices (e.g., robots, tablets, smartphones, etc.), the secure management of biometric data while respecting the users' privacy, and the design of appropriate user interaction with facial verification mechanisms for all kinds of users. We analyze different approaches to solving all these challenges and propose a knowledge-driven methodology for the automated deployment of DNN-based FR solutions in IoT devices, with the secure management of biometric data, and real-time feedback for improved interaction. We provide some practical examples and experimental results with state-of-the-art DNNs for FR in Intel's and NVIDIA's hardware platforms as IoT devices.*

A.4 Key Strategies for Synthetic Data Generation for Training Intelligent Systems based on People Detection from Omnidirectional Cameras

Title: Key Strategies for Synthetic Data Generation for Training Intelligent Systems based on People Detection from Omnidirectional Cameras

Authors: Nerea Aranjuelo, Sara García, Estíbaliz Loyo, Luis Unzueta, Oihana Otaegui

Journal: Computers & Electrical Engineering

Year: 2021

DOI: https://doi.org/10.1016/j.compeleceng.2021.107105

Abstract: *To train Deep Neural Networks (DNNs)-based methods, suitable training data are key to help DNNs learn appropriate pattern recognition features. The use of synthetic data may help in generating sufficient and balanced data. However, models trained with such data often present a domain gap when applied to real-world scenarios. Many studies focus on techniques such as domain adaptation to minimize this gap, but little attention is paid to the data generation itself. Our work shows that this gap can be minimized by enhancing the generated data features. More specifically, we generate different synthetic training datasets with particular features and use them to train a DNN for people detection in large spaces using omnidirectional cameras. Experimental results with real-world data show that proper synthetic data minimize the domain gap. We also show that expanding a training dataset to include synthetic samples in addition to real samples, can improve the model's capabilities.*

A.5 Optimal Deployment of Face Recognition Solutions in a Heterogeneous IoT Platform for Secure Elderly Care Applications

Title: Optimal Deployment of Face Recognition Solutions in a Heterogeneous IoT Platform for Secure Elderly Care Applications

Authors: Unai Elordi, Alvaro Bertelsen, Luis Unzueta, **Nerea Aranjuelo**, Jon Goenetxea, Ignacio Arganda-Carreras

Proceedings: Procedia Computer Science *[Best Student Paper Award]*

Year: 2021

DOI: https://doi.org/10.1016/j.procs.2021.09.093

Abstract: *Face recognition provides a desirable solution for authentication and surveillance in Internet of Things platforms for elderly care. However, its inclusion is challenging because of the possibly reduced interaction capabilities of users, the high variety of interaction devices, and the need of managing biometric data securely. Our approach relies on lightweight deep neural networks for secure recognition and to guide users during interaction. An automated procedure selects the appropriate inference engine, model*

configurations, and batch size, based on edge device characteristics. Biometric data is homomorphically encrypted to preserve privacy. An evaluation with respect to state-of-the-art alternatives shows its potential.

A.6 Building Synthetic Simulated Environments for Configuring and Training Multi-camera Systems for Surveillance Applications

Title: Building Synthetic Simulated Environments for Configuring and Training Multi-camera Systems for Surveillance Applications

Authors: **Nerea Aranjuelo**, Jorge García, Luis Unzueta, Sara García, Unai Elordi, Oihana Otaegui

Proceedings: Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications

Year: 2021

Abstract: *Synthetic simulated environments are gaining popularity in the Deep Learning Era, as they can alleviate the effort and cost of two critical tasks to build multi-camera systems for surveillance applications: setting up the camera system to cover the use cases and generating the labeled dataset to train the required Deep Neural Networks (DNNs). However, there are no simulated environments ready to solve them for all kind of scenarios and use cases. Typically, 'ad hoc' environments are built, which cannot be easily applied to other contexts. In this work we present a methodology to build synthetic simulated environments with sufficient generality to be usable in different contexts, with little effort. Our methodology tackles the challenges of the appropriate parameterization of scene configurations, the strategies to generate randomly a wide and balanced range of situations of interest for training DNNs with synthetic data, and the quick image capturing from virtual cameras considering the rendering bottlenecks. We show a practical implementation example for the detection of incorrectly placed luggage in aircraft cabins, including the qualitative and quantitative analysis of the data generation process and its influence in a DNN training, and the required modifications to adapt it to other surveil-*

lance contexts.

A.7 **Multi-Stage Dynamic Batching and On-Demand I-Vector Clustering for Cost-effective Video Surveillance.**

Title: Multi-Stage Dynamic Batching and On-Demand I-Vector Clustering for Cost-effective Video Surveillance.

Authors: David Montero, Luis Unzueta, Jon Goenetxea, **Nerea Aranjuelo**, Estíbaliz Loyo, Oihana Otaegui, Marcos Nieto

Proceedings: Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications

Year: 2021

Abstract: *In this paper, we present a cost-effective Video-Surveillance System (VSS) for face recognition and online clustering of unknown individuals at large scale. We aim to obtain Performance Indicators (PIs) for people flow monitoring in large infrastructures, without storing any biometric information. For this purpose, we focus on how to take advantage of a central GPU-enabled computing server, connected to a set of video-surveillance cameras, to automatically register new identities and update their descriptive data as they are re-identified. The proposed method comprises two main procedures executed in parallel. A Multi-Stage Dynamic Batching (MSDB) procedure efficiently extracts facial identity vectors (i-vectors) from captured images. At the same time, an On-Demand I-Vector Clustering (ODIVC) procedure clusters the i-vectors into identities. This clustering algorithm is designed to progressively adapt to the increasing data scale, with a lower decrease in its effectiveness compared to other alternatives. Experimental results show that ODIVC achieves state-of-the-art results in well-known large scale datasets and that our VSS can detect, recognize and cluster in real time faces coming from up to 40 cameras with a central off-the-shelf GPU-enabled computing server.*

A.8 Robust 3D Object Detection from LiDAR Point Cloud Data with Spatial Information Aggregation

Title: Robust 3D Object Detection from LiDAR Point Cloud Data with Spatial Information Aggregation

Authors: **Nerea Aranjuelo**, Guus Engels, Luis Unzueta, Ignacio Arganda-Carreras, Marcos Nieto, Oihana Otaegui

Proceedings: 15th International Conference on Soft Computing Models in Industrial and Environmental Applications

Year: 2020

Abstract: *Current 3D object detectors from Bird's Eye View (BEV) LiDAR point cloud data rely on Convolutional Neural Networks (CNNs), which have originally been designed for camera images. Therefore, they look for the same target features, regardless of the position of the objects with respect to the sensor. Discarding this spatial information makes 3D object detection unreliable and not robust, because objects in LiDAR point clouds contain distance dependent features. The position of a group of points can be decisive to know if they represent an object or not. To solve this, we propose a network extension called FeatExt operation that enables the model to be aware of both the target objects features and their spatial location. FeatExt operation expands a group of feature maps extracted from a BEV representation to include the distance to a specific position of interest in the scene, in this case the distance with respect to the LiDAR. When adding the proposed operation to a baseline network in an intermediate fusion fashion, it shows up to an 8.9 average precision boost in the KITTI BEV benchmark. Our proposal can be easily added to improve existing object detection networks.*

A.9 Building a Camera-based Smart Sensing System for Digitalized On-demand Aircraft Cabin Readiness Verification.

Title: Building a Camera-based Smart Sensing System for Digitalized On-demand Aircraft Cabin Readiness Verification.

DATA-CENTRIC DESIGN AND TRAINING OF DEEP NEURAL NETWORKS WITH MULTIPLE DATA MODALITIES FOR VISION-BASED PERCEPTION SYSTEMS

Authors: Luis Unzueta, Sandra Garcia, Jorge García, Valentin Corbin, **Nerea Aranjuelo**, Unai Elordi, Oihana Otaegui, Maxime Danielli

Proceedings: ROBOVIS

Year: 2020

Abstract: *Currently, aircraft cabin operations such as the verification of taxi, take-off, and landing (TTL) cabin readiness are done manually. This results in an increased workload for the crew, operational inefficiencies, and a nonnegligible risk of human errors in handling safety-related procedures. For TTL, specific cabin readiness requirements apply to the passenger, to the position of seat components and cabin luggage. The usage of cameras and vision-based object-recognition algorithms may offer a promising solution for specific functionalities such as cabin luggage detection. However, building a suitable camera-based smart sensing system for this purpose brings many challenges as it needs to be low weight, with competitive cost and robust recognition capabilities on individual seat level, complying with stringent constraints related to airworthiness certification. This position paper analyzes and discusses the main technological factors that system designers should consider for building such an intelligent system. These include the sensor setup, system training, the selection of appropriate camera sensors and lenses, AI-processors, and software tools for optimal image acquisition and image content analysis with Deep Neural Network (DNN)-based recognition methods. Preliminary tests with pre-trained generalist DNN-based object detection models are also analyzed to assist with the training and deployment of the recognition methods.*

A.10 3D Object Detection from LiDAR Data using Distance Dependent Feature Extraction

Title: 3D Object Detection from LiDAR Data using Distance Dependent Feature Extraction

Authors: Guus Engels, **Nerea Aranjuelo**, Ignacio Arganda-Carreras, Marcos Nieto, Oihana Otaegui

Proceedings: 6th International Conference on Vehicle Technology and Intelligent Transport Systems *[Best Industrial Paper Award]*

Year: 2020

Abstract: *This paper presents a new approach to 3D object detection that leverages the properties of the data obtained by a LiDAR sensor. State-of-the-art detectors use neural network architectures based on assumptions valid for camera images. However, point clouds obtained from LiDAR are fundamentally different. Most detectors use shared filter kernels to extract features which do not take into account the range dependent nature of the point cloud features. To show this, different detectors are trained on two splits of the KITTI dataset: close range (objects up to 25 meters from LiDAR) and long-range. Top view images are generated from point clouds as input for the networks. Combined results outperform the baseline network trained on the full dataset with a single backbone. Additional research compares the effect of using different input features when converting the point cloud to image. The results indicate that the network focuses on the shape and structure of the objects, rather than exact values of the input. This work proposes an improvement for 3D object detectors by taking into account the properties of LiDAR point clouds over distance. Results show that training separate networks for close-range and long-range objects boosts performance for all KITTI benchmark difficulties.*

A.11 BEV Object Tracking for LIDAR-based Ground Truth Generation

Title: BEV Object Tracking for LIDAR-based Ground Truth Generation

Authors: David Montero, **Nerea Aranjuelo**, Orti Senderos, Marcos Nieto

Proceedings: 27th European Signal Processing Conference

Year: 2019

Abstract: *Building ADAS (Advanced Driver Assistance Systems) or AD (Autonomous Driving) vehicles implies the acquisition of large volumes of data and a costly annotation process to create labeled metadata. Labels are then used for either ground truth composition (for test and validation of algorithms) or to set-up training datasets for machine learning processes. In this paper we present a 3D object tracking mechanism that operates on detections from point cloud sequences. It works in two steps: first an online phase*

which runs a Branch and Bound algorithm (BBA) to solve the association between detections and tracks, and a second filtering step which adds the required temporal smoothness. Results on KITTI dataset show the produced tracks are accurate and robust against noisy and missing detections, as produced by state-of-the-art deep learning detectors.

A.12 Web-based Video-assisted Point Cloud Annotation for ADAS Validation

Title: Web-based Video-assisted Point Cloud Annotation for ADAS Validation
Authors: Andoni Mujika, Ana Dominguez Fanlo, Iñigo Tamayo, Orti Senderos, Javier Barandiaran, **Nerea Aranjuelo**, Marcos Nieto, Oihana Otaegui
Proceedings: The 24th International Conference on 3D Web Technology
Year: 2019
Abstract: *This paper introduces a web application for point cloud annotation that is used in the advanced driver assistance systems field. Apart from the point cloud viewer, the web tool has an object viewer and a timeline to define the attributes of the annotations and a video viewer to validate the point cloud annotations with the corresponding video images. The paper also describes several strategies we followed to obtain annotations correctly and quickly: (i) memory management and rendering of large-scale point clouds, (ii) coherent combination of video images and annotations, (iii) content synchronization in all parts of the application and (iv) automatic annotation before and during the annotation task of the user.*

A.13 Fractal Characterization of Retinal Microvascular Network Morphology during Diabetic Retinopathy Progression

Title: Fractal Characterization of Retinal Microvascular Network Morphology during Diabetic Retinopathy Progression
Authors: Natasa Popovic, Mirko Lipovac, Miroslav Radunovic, Jurgi Ugarte, Erik Isusquiza, Andoni Beristain, Ramón Moreno, **Nerea Aranjuelo**, Tomo Popovic

Journal: Microcirculation

Year: 2019

DOI: https://doi.org/10.1111/micc.12531

Abstract: *Objective: The study aimed to characterize morphological changes of the retinal mi- crovascular network during the progression of diabetic retinopathy. Methods: Publicly available retinal images captured by a digital fundus camera from DIARETDB1 and STARE databases were used. The retinal microvessels were seg- mented using the automatic method, and vascular network morphology was ana- lyzed by fractal parametrization such as box-counting dimension, lacunarity, and multifractals. Results: The results of the analysis were affected by the ability of the segmentation method to include smaller vessels with more branching generations. In cases where the segmentation was more detailed and included a higher number of vessel branch- ing generations, increased severity of diabetic retinopathy was associated with in- creased complexity of microvascular network as measured by box-counting and multifractal dimensions, and decreased gappiness of retinal microvascular network as measured by lacunarity parameter. This association was not observed if the seg- mentation method included only 3-4 vessel branching generations. Conclusions: Severe stages of diabetic retinopathy could be detected noninvasively by using high resolution fundus photography and automatic microvascular segmenta- tion to the high number of branching generations, followed by fractal analysis para- metrization. This approach could improve risk stratification for the development of microvascular complications, cardiovascular disease, and dementia in diabetes.*

A.14 Multimedia Analysis in Police–Citizen Communication: Supporting Daily Policing Tasks

Title: Multimedia Analysis in Police–Citizen Communication: Supporting Daily Policing Tasks

Authors: Peter Leškovský, Santiago Prieto, Aratz Puerto, Jorge García, Luis Unzueta, **Nerea Aranjuelo**, Haritz Arzelus, Aitor Álvarez

Journal: Social Media Strategy in Policing: From Cultural Intelligence to Community

Policing

Publisher: Springer International Publishing

Year: 2019

DOI: https://doi.org/10.1007/978-3-030-22002-0_13

Abstract: *This chapter describes an approach for improved multimedia analysis as part of an ICT-based tool for community policing. It includes technology for automatic processing of audio, image and video contents sent as evidence by the citizens to the police. In addition to technical details of their development, results of their performance within initial pilots simulating nearly real crime situations are presented and discussed.*

A.15 Fully Automatic Detection and Segmentation of Abdominal Aortic Thrombus in Post-operative CTA Images using Deep Convolutional Neural Networks

Title: Fully Automatic Detection and Segmentation of Abdominal Aortic Thrombus in Post-operative CTA Images using Deep Convolutional Neural Networks

Authors: Karen López-Linares, **Nerea Aranjuelo**, Luis Kabongo, Gregory Maclair, Nerea Lete, Mario Ceresa, Ainhoa García-Familiar, Iván Macía, Miguel A González Ballester

Journal: Medical image analysis

Year: 2018

DOI: https://doi.org/10.1016/j.media.2018.03.010

Abstract: *Computerized Tomography Angiography (CTA) based follow-up of Abdominal Aortic Aneurysms (AAA) treated with Endovascular Aneurysm Repair (EVAR) is essential to evaluate the progress of the patient and detect complications. In this context, accurate quantification of post-operative thrombus volume is required. However, a proper evaluation is hindered by the lack of automatic, robust and reproducible thrombus segmentation algorithms. We propose a new fully automatic approach based on Deep Convolutional Neural Networks (DCNN) for robust and reproducible thrombus region of interest detection and subsequent fine thrombus segmentation. The DetecNet detection network is adapted to perform region of interest extraction from a complete CTA and a new segmentation network architecture, based on Fully Convolutional Networks and a*

Holistically-Nested Edge Detection Network, is presented. These networks are trained, validated and tested in 13 post-operative CTA volumes of different patients using a 4-fold cross-validation approach to provide more robustness to the results. Our pipeline achieves a Dice score of more than 82% for post-operative thrombus segmentation and provides a mean relative volume difference between ground truth and automatic segmentation that lays within the experienced human observer variance without the need of human intervention in most common cases.

A.16 Multimodal Deep Learning for Advanced Driving Systems

Title: Multimodal Deep Learning for Advanced Driving Systems
Authors: **Nerea Aranjuelo**, Luis Unzueta, Ignacio Arganda-Carreras, Oihana Otaegui
Proceedings: Articulated Motion and Deformable Objects, AMDO 2018
Publisher: Springer International Publishing
Year: 2018
DOI: `https://doi.org/10.1007/978-3-319-94544-6_10`
Abstract: *Multimodal deep learning is about learning features over multiple modalities. Impressive progress has been made in deep learning solutions that rely on a single sensor modality for advanced driving. However, these approaches are limited to cover certain functionalities. The potential of multimodal sensor fusion has been very little exploited, although research vehicles are commonly provided with various sensor types. How to combine their data to achieve a complex scene analysis and improve therefore robustness in driving is still an open question. While different surveys have been done for intelligent vehicles or deep learning, to date no survey on multimodal deep learning for advanced driving exists. This paper attempts to narrow this gap by providing the first review that analyzes existing literature and two indispensable elements: sensors and datasets. We also provide our insights on future challenges and work to be done.*

APPENDIX

B

Glossary

.

Acronyms

ADAS Adavanced Driver Assistance Sytems

AI Artificial Intelligence

ANN Artificial Neural Network

AP Average Precision

ASD Advanced Synthetic Dataset

BEV Bird's Eye View

CAN Controller Area Network

CLAHE Contrast Limited Adaptive Histogram Equalization

CNN Convolutional Neural Network

CRF Conditional Random Field

DANN Domain Adversarial Neural Network

DASD Dynamic Advanced Synthetic Dataset

DL Deep learning

DNN Deep Neural Network

DRD Domain Randomization Dataset

DRL Deep Reinforcement Learning

FID Fréchet Inception Distance

GAN Generative Adversarial Network

GC-GAN Gaze-aware Compositional GAN

GDPR General Data Protection Regulation

GLG Gaze-aware Local Generator

GNSS Global Navigation Satellite System

GPU Graphics processing unit

I2V Infrastructure-to-vehicle

IMU Inertial Measurement Units

IoU Intersection over union

IS Inception Score

ITS Intelligent Transport Systems

LiDAR Light Detection And Ranging

MLops Machine learning operations

MLP Multi-layer perceptron

MLTRL Machine Learning Technnology Readiness Level

NMS Non-Maximum Supression

PoC Proof of concept

Radar Radio Detection And Ranging

RDD Real Data Dataset

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

ROI Region of interest

RPN Region Proposal Network

RSDD Real and Synthetic Data Dataset

SRSD Simplified Rectified Synthetic Dataset

SSD Simplified Synthetic Dataset

TTL Taxi, Take-off, and Landing

V2V Vehicle-to-vehicle

VCD Video Content Description

VFE Voxel feature encoding

Part VI

Bibliography

Bibliography

[1] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012. xvii, 11, 17

[2] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018. xvii, 12

[3] Alexander Lavin, Ciarán M Gilligan-Lee, Alessya Visnjic, Siddha Ganju, Dava Newman, Sujoy Ganguly, Danny Lange, Atílím Güneş Baydin, Amit Sharma, Adam Gibson, et al. Technology readiness levels for machine learning systems. *Nature Communications*, 13(1):6039, 2022. xvii, 14

[4] Braden Hurl, Krzysztof Czarnecki, and Steven L. Waslander. Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception. In *IEEE IV*, pages 2522–2529, 2019. xviii, 50

[5] Kuan-Ting Lai, Chia-Chih Lin, Chun-Yao Kang, Mei-Enn Liao, and Ming-Syan Chen. VIVID: Virtual environment for visual deep learning. In *Proc. ACM MM*, pages 1356–1359, 2018. xviii, 50, 51, 95

[6] T. Scheck, R. Seidel, and G. Hirtz. Learning from theodore: A synthetic omnidirectional top-view indoor dataset for deep transfer learning. In *Winter Conference on Applications of Computer Vision (WACV)*, 2020. xviii, 49, 50, 51, 70, 95, 96

[7] Yichun Shi, Xiao Yang, Yangyue Wan, and Xiaohui Shen. SemanticStyleGAN: Learning compositional generative priors for controllable image synthesis and editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11254–11264, 2022. xx, 53, 101, 102, 103

[8] Martin Simony, Stefan Milzy, Karl Amendey, and Horst-Michael Gross. Complex-yolo: An euler-region-proposal for real-time 3D object detection on point clouds. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018. xxiv, 56, 126, 127

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 4, 7

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. 4

[11] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022. 4

[12] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In *Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1 1*, pages 128–144. Springer, 2020. 4

[13] Mohamed Elgendy. *Deep learning for vision systems*. Simon and Schuster, 2020. 4

[14] Jiachen Yang, Chenguang Wang, Bin Jiang, Houbing Song, and Qinggang Meng. Visual perception enabled industry intelligence: state of the art, challenges and prospects. *IEEE Transactions on Industrial Informatics*, 17(3):2204–2219, 2020. 4

[15] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020. 5

[16] G Sreenu and Saleem Durai. Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data*, 6(1):1–27, 2019. 5

[17] Yen-Chen Lin. *Visual Transfer Learning for Robotic Manipulation*. PhD thesis, Massachusetts Institute of Technology, 2021. 5, 31

[18] Adam Ziebinski, Rafal Cupek, Damian Grzechca, and Lukas Chruszczyk. Review of advanced driver assistance systems (adas). In *AIP Conference Proceedings*, volume 1906, page 120002. AIP Publishing LLC, 2017. 5, 42

[19] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE intelligent systems*, 24(2):8–12, 2009. 6, 14

[20] Mark Treveil, Nicolas Omont, Clément Stenac, Kenji Lefevre, Du Phan, Joachim Zentici, Adrien Lavoillotte, Makoto Miyazaki, and Lynn Heidmann. *Introducing MLOps*. O'Reilly Media, 2020. 9

[21] Ke Li, Gang Wan, Gong Cheng, Liqiu Meng, and Junwei Han. Object detection in optical remote sensing images: A survey and a new benchmark. *ISPRS journal of photogrammetry and remote sensing*, 159:296–307, 2020. 10

[22] Tianfei Zhou, Fatih Porikli, David J Crandall, Luc Van Gool, and Wenguan Wang. A survey on deep learning technique for video segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 10

[23] Georgios Zamanakos, Lazaros Tsochatzidis, Angelos Amanatiadis, and Ioannis Pratikakis. A comprehensive survey of lidar-based 3D object detection methods with deep learning for autonomous driving. *Computers & Graphics*, 99:153–181, 2021. 10

[24] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th ICML-11*, pages 689–696, 2011. 10

[25] Stuti Jindal and Sanjay Singh. Image sentiment analysis using deep convolutional neural networks with domain specific fine tuning. In *2015 International Conference on Information Processing (ICIP)*, pages 447–451. IEEE, 2015. 10

[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 11, 12, 17, 31, 46

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. inadvances in neural information processing systems 25. *Go to reference in article*, 2012. 12, 30, 39

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 12, 39, 48, 109, 120

[29] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 12, 39, 54, 97, 134

[30] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017. 13

[31] Giosué Cataldo Marinó, Alessandro Petrini, Dario Malchiodi, and Marco Frasca. Deep neural networks compression: A comparative survey and choice recommendations. *Neurocomputing*, 520:152–170, 2023. 13

[32] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021. 13

[33] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019. 13

[34] The Artificial Intelligence act (AI Act). `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021PC0206`. Accessed: 2023-01-28. 13

[35] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 220–229, 2019. 13

[36] Mark Mazumder, Colby Banbury, Xiaozhe Yao, Bojan Karlaš, William Gaviria Rojas, Sudnya Diamos, Greg Diamos, Lynn He, Douwe Kiela, David Jurado, et al. Dataperf: Benchmarks for data-centric ai development. *CoRR*, abs/2207.10062, 2022. 15

[37] Lora Aroyo, Matthew Lease, Praveen Paritosh, and Mike Schaekermann. Data excellence for ai: why should you care? *Interactions*, 29(2):66–69, 2022. 15

[38] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6):1–35, 2021. 15

[39] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. "everyone wants to do the model work, not the data work": Data cascades in high-stakes ai. In *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2021. 15

[40] Mahima Pushkarna, Andrew Zaldivar, and Oddur Kjartansson. Data cards: Purposeful and transparent dataset documentation for responsible ai. In *2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 1776–1826, 2022. 15

[41] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, et al. The open images dataset v4. *International Journal of Computer Vision*, 128(7):1956–1981, 2020. 16, 17, 31, 46

[42] Sergey I Nikolenko. *Synthetic data for deep learning*, volume 174. Springer, 2021. 16, 49, 50

[43] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle detection from 3D lidar using fully convolutional network. In David Hsu, Nancy M. Amato, Spring Berman, and Sam Ade Jacobs, editors, *Robotics: Science and Systems XII, University of Michigan, Ann Arbor, Michigan, USA, June 18 - June 22, 2016*, 2016. 18

[44] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3D lidar point cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893. IEEE, 2018. 18

[45] Jorge Beltrán, Carlos Guindel, Francisco Miguel Moreno, Daniel Cruzado, Fernando Garcia, and Arturo De La Escalera. Birdnet: a 3D object detection framework from lidar information. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3517–3523. IEEE, 2018. 18, 56

[46] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. 25

[47] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013. 28

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 28, 31

[49] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 30

[50] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning (adaptive computation and machine learning series). *Cambridge Massachusetts*, pages 321–359, 2017. 30

[51] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006. 30

[52] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Twelfth annual conference of the international speech communication association*, 2011. 31

[53] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016. 31

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 31

[55] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 33

[56] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011. 33

[57] Ishan Jindal, Matthew S. Nokleby, and Xue-wen Chen. Learning deep networks from noisy labels with dropout regularization. In *ICDM*, pages 967–972. IEEE Computer Society, 2016. 34

[58] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989. 35

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 35

[60] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 35

[61] Xin Li, Shenqi Lai, and Xueming Qian. Dbcface: Towards pure convolutional neural network face detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(4):1792–1804, 2021. 35

[62] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017. 35

[63] Guodong Guo and Na Zhang. A survey on deep learning based face recognition. *Computer vision and image understanding*, 189:102805, 2019. 39

[64] Antonio Brunetti, Domenico Buongiorno, Gianpaolo Francesco Trotta, and Vitoantonio Bevilacqua. Computer vision and deep learning techniques for pedestrian detection and tracking: A survey. *Neurocomputing*, 300:17–33, 2018. 39

[65] Nhat-Duy Nguyen, Tien Do, Thanh Duc Ngo, and Duy-Dinh Le. An evaluation of deep learning methods for small object detection. *Journal of Electrical and Computer Engineering*, 2020, 2020. 41, 73

[66] Yang Zhang, Philip David, and Boqing Gong. Curriculum domain adaptation for semantic segmentation of urban scenes. In *Proceedings of the IEEE international conference on computer vision*, pages 2020–2030, 2017. 41

[67] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989. 42

[68] Gabriel L Oliveira, Wolfram Burgard, and Thomas Brox. Efficient deep models for monocular road segmentation. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4885–4891. IEEE, 2016. 43, 44

[69] Jingjing Liu, Shaoting Zhang, Shu Wang, and Dimitris N Metaxas. Multispectral deep neural networks for pedestrian detection. In *27th British Machine Vision Conference, BMVC 2016*, 2016. 43, 44, 55

[70] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3D object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4490–4499, 2018. 44, 57

[71] Liang Xiao, Ruili Wang, Bin Dai, Yuqiang Fang, Daxue Liu, and Tao Wu. Hybrid conditional random field based camera-LiDAR fusion for road detection. *Information Sciences*, 2017. 44, 55

[72] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Košecká. 3d bounding box estimation using deep learning and geometry. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5632–5640. IEEE, 2017. 44

[73] Alessio Carullo and Marco Parvis. An ultrasonic sensor for distance measurement in automotive applications. *IEEE Sensors journal*, 1(2):143–147, 2001. 44

[74] Jakob Lombacher, Markus Hahn, Jürgen Dickmann, and Christian Wöhler. Potential of radar for static object classification using deep learning methods. In *2016 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*, pages 1–4. IEEE, 2016. 45

[75] Jaskaran Virdi. *Using Deep Learning to Predict Obstacle Trajectories for Collision Avoidance in Autonomous Vehicles*. PhD thesis, UC San Diego, 2017. 45, 55

[76] Hideki Shimada, Akihiro Yamaguchi, Hiroaki Takada, and Kenya Sato. Implementation and evaluation of local dynamic map in safety driving systems. *JTTs*, 5(02):102, 2015. 45, 55

[77] Intempora. RTMaps. `https://intempora.com/products/rtmaps/`. (Accessed on 12/18/2022). 45

[78] Elektrobit. EB Assist ADTF. `https://www.elektrobit.com/products/eb-assist/adtf/`. (Accessed on 12/18/2022). 45

[79] ROS. `http://www.ros.org/`. (Accessed on 12/18/2022). 45, 58, 138

[80] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 46, 74

[81] Liang Zheng, Hengheng Zhang, Shaoyan Sun, Manmohan Chandraker, Yi Yang, and Qi Tian. Person re-identification in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1367–1376, 2017. 46

[82] Markus Braun, Sebastian Krebs, Fabian Flohr, and Dariu M Gavrila. Eurocity persons: A novel benchmark for person detection in traffic scenes. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1844–1861, 2019. 46

[83] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, et al. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR 2011*, pages 3153–3160. IEEE, 2011. 46

[84] Zhihao Duan, Ozan Tezcan, Hayato Nakamura, Prakash Ishwar, and Janusz Konrad. Rapid: rotation-aware people detection in overhead fisheye images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 636–637, 2020. 46

[85] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012. 47

[86] 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 *Proc. of the IEEE conference on CVPR*, pages 3213–3223, 2016. 47

[87] Shenlong Wang, Min Bai, Gellért Máttyus, Hang Chu, Wenjie Luo, Bin Yang, Justin Liang, Joel Cheverie, Sanja Fidler, and Raquel Urtasun. Torontocity: Seeing the world with a million eyes. In *IEEE International Conference on Computer Vision,*

ICCV 2017, Venice, Italy, October 22-29, 2017, pages 3028–3036. IEEE Computer Society, 2017. 47

[88] Xavier Roynard, Jean-Emmanuel Deschaud, and François Goulette. Paris-lille-3d: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *The International Journal of Robotics Research*, 37(6):545–557, 2018. 47

[89] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The Oxford RobotCar dataset. *International Journal of Robotics Research*, 36(1):3–15, 2017. 47

[90] Eder Santana and George Hotz. Learning a driving simulator. *arXiv preprint arXiv:1608.01230*, 2016. 47

[91] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182, 2017. 47, 54

[92] Gerhard Neuhold, Tobias Ollmann, S Rota Bulò, and Peter Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *Proc. ICCV*, pages 22–29, 2017. 47

[93] Yukyung Choi, Namil Kim, Soonmin Hwang, Kibaek Park, Jae Shin Yoon, Kyounghwan An, and In So Kweon. KAIST Multi-Spectral Day/Night Data Set for Autonomous and Assisted Driving. *IEEE Transactions on Intelligent Transportation Systems*, 2018. 47

[94] 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. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 47

[95] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation

learning. In *European conference on computer vision*, pages 491–507. Springer, 2020. 47

[96] Maithra Raghu, Chiyuan Zhang, Jon M. Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning for medical imaging. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*, pages 3342–3352, 2019. 47

[97] Shuteng Niu, Meryl Liu, Yongxin Liu, Jian Wang, and Houbing Song. Distant domain transfer learning for medical imaging. *IEEE J. Biomed. Health Informatics*, 25(10):3784–3793, 2021. 47

[98] Colorado J. Reed, Xiangyu Yue, Ani Nrusimha, Sayna Ebrahimi, Vivek Vijaykumar, Richard Mao, Bo Li, Shanghang Zhang, Devin Guillory, Sean Metzger, Kurt Keutzer, and Trevor Darrell. Self-supervised pretraining improves self-supervised pretraining. In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV*, pages 1050–1060, 2022. 47

[99] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *Proc. IEEE/CVF CVPR*, pages 403–412, 2019. 47

[100] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 48

[101] Andoni Mujika, Ana Dominguez Fanlo, I. Tamayo, Orti Senderos, Javier Barandiaran, Nerea Aranjuelo, Marcos Nieto, and Oihana Otaegui. Web-based video-assisted point cloud annotation for ADAS validation. In *Proc. International Conference on 3D Web Technology*, pages 1–9, 2019. 48

[102] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *IEEE conference on computer vision and pattern recognition*, page 4340–4349, 2016. 49

[103] Alireza Shafaei, James J. Little, and Mark Schmidt. Play and learn: Using video games to train computer vision models. In *Proc. BMVC*, 2016. 49, 58, 138

[104] Tuan Anh Le, Atilim Giineş Baydin, Robert Zinkov, and Frank Wood. Using synthetic data to train neural networks is model-based reasoning. In *2017 International Joint Conference on Neural Networks*, pages 3514–3521. IEEE, 2017. 49

[105] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic scene completion from a single depth image. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 49

[106] W. Li, S. Saeedi, J. McCormac, R. Clark, D. Tzoumanikas, Q. Ye, Y. Huang, R. Tang, and S. Leutenegger. Interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset. In *British Machine Vision Conference (BMVC)*, 2018. 49

[107] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio Lopez. The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. 2016. 49

[108] D. Dwibedi, I. Misra, and M. Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 49, 50

[109] S. Tripathi, S. Chandra, A. Agrawal, A. Tyagi, J. M. Rehg, and V. Chari. Learning to generate synthetic data via compositing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 50

[110] P. Krahenbuhl. Free supervision from video games. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 2955–2964, 2018. 50

[111] T. Tran, T. Pham, G. Carneiro, L. Palmer, and I. Reid. Learning from simulated and unsupervised images through adversarial training. In *Advances in Neural Information Processing Systems*, page 1–10, 2017. 50

[112] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid. Learning from synthetic humans. In *IEEE Conference on Computer Vision and Pattern Recognition*, page 109–117, 2017. 50

[113] Fatemeh Sadat Saleh, Mohammad Sadegh Aliakbarian, Mathieu Salzmann, Lars Petersson, and Jose M. Alvarez. Effective use of synthetic data for urban scene semantic segmentation. In *Proc. ECCV*, volume 11206 of *LNCS*, pages 86–103, 2018. 50, 51, 95

[114] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. *Field and Service Robotics*, pages 621–635, 2017. 50, 51

[115] Samin Khan, Buu Phan, Rick Salay, and Krzysztof Czarnecki. ProcSy: Procedural synthetic dataset generation towards influence factor studies of semantic segmentation networks. In *Proc. CVPR Workshops*, pages 88–96, 2019. 50, 51, 94, 95, 96

[116] Param S. Rajpura, Hristo Bojinov, and Ravi S. Hegde. Object detection using deep CNNs trained on synthetic images. *arXiv preprint arXiv:1706.06782*, 2017. 50, 51, 95

[117] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, October 2015. 51

[118] Paul Bourke and Dalai Felinto. Blender and immersive gaming in a hemispherical dome. In *International Conference on Computer Games, Multimedia and Allied Technology*, volume 1, pages 280–284, 2010. 51

[119] Pablo Hernandez-Leal, Bilal Kartal, and Matthew Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33:750–797, 10 2019. 51

[120] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Marco Hutter

and Roland Siegwart, editors, *Field and Service Robotics, Results of the 11th International Conference, FSR 2017, Zurich, Switzerland, 12-15 September 2017*, volume 5 of *Springer Proceedings in Advanced Robotics*, pages 621–635. Springer, 2017. 52

[121] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 2017. 52

[122] Erroll Wood, Tadas Baltrušaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. Learning an appearance-based gaze estimator from one million synthesised images. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, pages 131–138, 2016. 52

[123] Erroll Wood, Tadas Baltrušaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. Gazedirector: Fully articulated eye gaze redirection in video. In *Computer Graphics Forum*, volume 37, pages 217–225. Wiley Online Library, 2018. 52

[124] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021. 52

[125] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 52

[126] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 52

[127] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *Advances in Neural Information Processing Systems*, 34:852–863, 2021. 52

[128] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2107–2116, 2017. 53, 108

[129] Matan Sela, Pingmei Xu, Junfeng He, Vidhya Navalpakkam, and Dmitry Lagun. Gazegan-unpaired adversarial image generation for gaze estimation. *arXiv preprint arXiv:1711.09767*, 2017. 53

[130] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017. 53, 59

[131] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9243–9252, 2020. 53

[132] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls. *Advances in Neural Information Processing Systems*, 33:9841–9850, 2020. 53

[133] Yu Deng, Jiaolong Yang, Dong Chen, Fang Wen, and Xin Tong. Disentangled and controllable face image generation via 3D imitative-contrastive learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5154–5163, 2020. 53

[134] Cheng-Han Lee, Ziwei Liu, Lingyun Wu, and Ping Luo. Maskgan: Towards diverse and interactive facial image manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5549–5558, 2020. 53, 99, 108

[135] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 53, 101, 103, 107

[136] Ali Jahanian, Lucy Chai, and Phillip Isola. On the "steerability" of generative adversarial networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. 53

[137] Zhe He, Adrian Spurr, Xucong Zhang, and Otmar Hilliges. Photo-realistic monocular gaze redirection using generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6932–6941, 2019. 53

[138] Weihao Xia, Yujiu Yang, Jing-Hao Xue, and Wensen Feng. Controllable continuous gaze redirection. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 1782–1790, 2020. 53

[139] Yaroslav Ganin, Daniil Kononenko, Diana Sungatullina, and Victor Lempitsky. Deepwarp: Photorealistic image resynthesis for gaze manipulation. In *European conference on computer vision*, pages 311–326. Springer, 2016. 53

[140] Jichao Zhang, Meng Sun, Jingjing Chen, Hao Tang, Yan Yan, Xueying Qin, and Nicu Sebe. Gazecorrection: Self-guided eye manipulation in the wild using self-supervised generative adversarial networks. *arXiv preprint arXiv:1906.00805*, 2019. 53

[141] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. 54

[142] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 3530–3538. IEEE Computer Society, 2017. 54

[143] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. 54

[144] Naman Patel, Anna Choromanska, Prashanth Krishnamurthy, and Farshad Khorrami. Sensor Modality Fusion with CNNs for UGV Autonomous Driving in Indoor Environments. In *International Conference on Intelligent Robots and Systems (IROS). IEEE*, 2017. 54

[145] Guan-Horng Liu, Avinash Siravuru, Sai Prabhakar, Manuela Veloso, and George Kantor. Learning end-to-end multimodal sensor policies for autonomous navigation. In *Conference on Robot Learning*, pages 249–261. PMLR, 2017. 55

[146] Shimon Ullman. Against direct perception. *BBS*, 3(3):373–381, 1980. 55

[147] Yanming Guo, Yu Liu, Theodoros Georgiou, and Michael S Lew. A review of semantic segmentation using deep neural networks. *IJMIR*, pages 1–7, 2017. 55

[148] Junwei Han, Dingwen Zhang, Gong Cheng, Nian Liu, and Dong Xu. Advanced Deep-Learning Techniques for Salient and Category-Specific Object Detection: A Survey. *IEEE Signal Processing Magazine*, 35(1):84–100, 2018. 55

[149] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. 55, 73, 74

[150] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 55, 57, 120, 121, 122

[151] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D object detection network for autonomous driving. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017. 56, 57

[152] Jie Zhou, Xin Tan, Zhiwen Shao, and Lizhuang Ma. FVnet: 3D front-view proposal generation for real-time object detection from point clouds. In *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–8. IEEE, 2019. 56

[153] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3D object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018. 56, 127

[154] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3D object detection from rgb-d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 918–927, 2018. 57, 127

[155] Dingfu Zhou, Jin Fang, Xibin Song, Liu Liu, Junbo Yin, Yuchao Dai, Hongdong Li, and Ruigang Yang. Joint 3D instance segmentation and object detection for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1839–1849, 2020. 57, 58, 127

[156] Zining Wang, Wei Zhan, and Masayoshi Tomizuka. Fusing bird's eye view lidar point cloud and front view camera image for 3d object detection. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–6, 2018. 57

[157] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 58

[158] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 58, 127, 128

[159] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019. 58, 127

[160] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D object proposal generation and detection from point cloud. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 770–779. Computer Vision Foundation / IEEE, 2019. 58

[161] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 58, 121

[162] Nerea Aranjuelo, Sara García, Estíbaliz Loyo, Luis Unzueta, and Oihana Otaegui. Key strategies for synthetic data generation for training intelligent systems based on people detection from omnidirectional cameras. *Computers & Electrical Engineering*, 92:107105, 2021. 58, 64, 138

[163] 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, 2018. 59, 138

[164] Andoni Cortés, Clemente Rodríguez, Gorka Vélez, Javier Barandiarán, and Marcos Nieto. Analysis of classifier training on synthetic data for cross-domain datasets. *IEEE Trans. on Intelligent Transportation Systems*, 2020. 59

[165] Michele Tonutti, Emanuele Ruffaldi, Alessandro Cattaneo, and Carlo Alberto Avizzano. Robust and subject-independent driving manoeuvre anticipation through domain-adversarial recurrent neural networks. *Robotics and Autonomous Systems*, 115:162–173, 2019. 59

[166] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016. 59, 134, 135

[167] D. Scaramuzza. Omnidirectional camera. *Springer US*, pages 1–1, 2014. 63, 66

[168] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. 68

[169] L. Hosek and A. Wilkie. An analytic model for full spectral sky-dome radiance. *ACM TOG 31(4)*, 2012. 69

[170] P. Shirley A.J. Preetham and B. Smits. A practical analytic model for daylight. *ACM Press/Addison-Wesley Publishing Co.*, 1999. 69

[171] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613, 2014. 71

[172] Barret Zoph, Ekin D Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V Le. Learning data augmentation strategies for object detection. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII 16*, pages 566–583. Springer, 2020. 73

[173] 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. 75

[174] Nerea Aranjuelo Ansa, Jorge García Castaño, Luis Unzueta Irurtia, Sara García Torres, Unai Elordi Hidalgo, and Oihana Otaegui Madurga. Building synthetic simulated environments for configuring and training multi-camera systems for surveillance applications. 2021. 80

[175] Vicomtech. VCD - video content description. `https://vcd.vicomtech.org/`, 2020. 82

[176] ASAM. OpenLABEL. `https://www.asam.net/project-detail/scenario-storage-and-labelling/`, 2020. 82

[177] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 97

[178] Yoshua Bengio and MONTREAL CA. Rmsprop and equilibrated adaptive learning rates for nonconvex optimization. *corr abs/1502.04390*, 2015. 97, 137

[179] Xucong Zhang, Seonwook Park, Thabo Beeler, Derek Bradley, Siyu Tang, and Otmar Hilliges. Eth-xgaze: A large scale dataset for gaze estimation under extreme head pose and gaze variation. In *European Conference on Computer Vision*, pages 365–381. Springer, 2020. 99, 108, 109

[180] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018. 103

[181] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR, 2018. 103

[182] Sagie Benaim and Lior Wolf. One-shot unsupervised cross domain translation. *advances in neural information processing systems*, 31, 2018. 104

[183] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 107

[184] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 107

[185] David Masko. Calibration in eye tracking using transfer learning, 2017. 108

[186] Sonia Porta, Benoit Bossavit, Rafael Cabeza, Andoni Larumbe-Bergera, Gonzalo Garde, and Arantxa Villanueva. U2eyes: A binocular dataset for eye tracking and gaze estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. 108

[187] Neelabh Sinha, Michal Balazia, and François Bremond. Flame: Facial landmark heatmap activated multimodal gaze estimation. In *2021 17th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–8. IEEE, 2021. 108

[188] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 111

[189] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016. 111

[190] Nerea Aranjuelo, Guus Engels, David Montero, Marcos Nieto, Ignacio Arganda-Carreras, Luis Unzueta, and Oihana Otaegui. Accurate 3D object detection from point cloud data using bird's eye view representations. In *IJCCI*, pages 246–253, 2021. 120

[191] Nerea Aranjuelo, Guus Engels, Luis Unzueta, Ignacio Arganda-Carreras, Marcos Nieto, and Oihana Otaegui. Robust 3D object detection from lidar point cloud data with spatial information aggregation. In *International Workshop on Soft Computing Models in Industrial and Environmental Applications*, pages 813–823. Springer, 2020. 120

[192] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4905–4913, 2016. 121

[193] 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, 2017. 121

[194] Wen Qian, Xue Yang, Silong Peng, Junchi Yan, and Yue Guo. Learning modulated loss for rotated object detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 2458–2466, 2021. 122

[195] Dingfu Zhou, Jin Fang, Xibin Song, Chenye Guan, Junbo Yin, Yuchao Dai, and Ruigang Yang. Iou loss for 2d/3d object detection. In *2019 International Conference on 3D Vision (3DV)*, pages 85–94. IEEE, 2019. 122, 123

[196] Guus Engels, Nerea Aranjuelo, Ignacio Arganda-Carreras, Marcos Nieto, and Oihana Otaegui. 3d object detection from lidar data using distance dependent feature extraction. In Karsten Berns, Markus Helfert, and Oleg Gusikhin, editors,

Proceedings of the 6th International Conference on Vehicle Technology and Intelligent Transport Systems, VEHITS 2020, Prague, Czech Republic, May 2-4, 2020, pages 289–300. SCITEPRESS, 2020. 123

[197] NVIDIA. Nvidia tensorrt, 2021. 125

[198] Nerea Aranjuelo, Jose Luis Apellaniz, Luis Unzueta, Jorge Garcia, Sara Garcia, Unai Elordi, and Oihana Otaegui. Leveraging synthetic data for dnn-based visual analysis of passenger seats. *SN Computer Science*, 4(1):40, 2022. 131