NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

COLLISION AVOIDANCE ON UNMANNED AERIAL VEHICLES USING DEEP NEURAL NETWORKS

DÁRIO FILIPE ROMANA PEDRO

Master in Electrical and Computer Engineering

DOCTORATE IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon

March 2022

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

COLLISION AVOIDANCE ON UNMANNED AERIAL VEHICLES USING DEEP NEURAL NETWORKS

DÁRIO FILIPE ROMANA PEDRO

Master in Electrical and Computer Engineering

Adviser: André Teixeira Bento Damas Mora
Assistant Professor, NOVA University Lisbon

Co-advisers: José Manuel Fonseca
Associate Professor with Aggregation, NOVA University Lisbon

Examination Committee:

Chair: João Carlos da Palma Goes,
Full Professor, NOVA University Lisbon

Rapporteurs: Luís Miguel Parreira e Correia,
Full Professor, Faculdade de Ciências da Universidade de Lisboa
Henrique José Monteiro Oliveira,
Coordinator Professor, Instituto Politécnico de Beja

Adviser: André Teixeira Bento Damas Mora
Assistant Professor, NOVA University Lisbon

Members: Alexandre Miguel Martins Pinto,
Guest Lecturer, Imperial College London - Faculty of Natural Sciences
João Carlos da Palma Goes,
Full Professor, NOVA University Lisbon
Luís Augusto Bica Gomes de Oliveira,
Associate Professor with Aggregation, NOVA University Lisbon

DOCTORATE IN ELECTRICAL AND COMPUTER ENGINEERING

NOVA University Lisbon
March 2022

Collision Avoidance on Unmanned Aerial Vehicles using Deep Neural Networks

To all that supported me.

Acknowledgements

"Talent wins games, but teamwork and intelligence win championships." - Michael Jordan

Individually we can attain outstanding achievements, but the most significant successes are always achieved in union. Greatness can easily be achieved with teamwork and the support of those around us. For this reason, I would like to save a special place in this work to thank the people who were present and supported me throughout this learning process.

First of all, I want to thank my academic advisor and co-advisor, Professor Dr. André Mora and Professor Dr. José Fonseca, who sparked my interest in Computer Vision and Machine Learning. I also have to thank them for all the support, intellect, patience, and time provided over the last 4 years. Additionally, I sincerely thank Dr. Luis Campos. He convinced me to change my career from consultancy to Research and Development and pushed me to start my PhD. Even with his always overbooked agenda, he managed to find time to help me and guide me through out my career, continuously providing new challenging ideas without ever thinking of the financial risks associated with trying. This has created the fantastic research team that I work with nowadays, for which I'm really grateful.

Next, I would like to thank FCT-UNL and DEE, which were decisive for my intellectual and professional growth. Also, I would like to thank PDMFC and Beyond Vision for all the conditions they provided me and for being a second home where I could explore all the ideas I could think of.

I want to thank my parents and sister for giving me with all the conditions to carry out and complete this stage of my life. You have always given more than was asked of you and are an example to me. I know the effort and dedication that went into this, and for that, I am eternally grateful to you. I also want to thank my closest family members, who taught me so much and made it possible to be who I am today. To my childhood friends, who are practically family, I have to thank the good moments spent and shared experiences, which I consider very important to finish this stage of my life.

I also want to thank all my colleagues and friends who accompanied me throughout this journey, specially to João Carvalho for all his help and partnership, for the hours of socializing and intense work, and for all the co-developed work and articles. Furthermore, I want to thank Álvaro Ramos and Pedro Lousã for teaching me a lot from their area of expertise and with whom I could debate a lot of ideas.

Additionally, I would like to thank Didier Lopes and Hugo Pedro for their patience and carefulness in reviewing this document. Their inputs significantly improved the document quality.

Finally, I am grateful for the financial support given by the ECSEL Joint Undertaking (JU) under grant agreement No 783221. This work has also received funding from the ECSEL JU under grant agreement No 783119. In addition, the JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Belgium, Czech Republic, Finland, Germany, Greece, Italy, Latvia, Norway, Poland, Portugal, Spain, Sweden. Finally, this work was also partially funded by Fundação para a Ciência e a Tecnologia under Projects UIDB/00066/2020, foRESTER PCIF/SSI/0102/2017, and IF/00325/2015.

Abstract

Unmanned Aerial Vehicles (UAVs), although hardly a new technology, have recently gained a prominent role in many industries, being widely used not only among enthusiastic consumers but also in high demanding professional situations, and will have a massive societal impact over the coming years. However, the operation of UAVs is full of serious safety risks, such as collisions with dynamic obstacles (birds, other UAVs, or randomly thrown objects). These collision scenarios are complex to analyze in real-time, sometimes being computationally impossible to solve with existing State of the Art (SoA) algorithms, making the use of UAVs an operational hazard and therefore significantly reducing their commercial applicability in urban environments. In this work, a conceptual framework for both stand-alone and swarm (networked) UAVs is introduced, focusing on the architectural requirements of the collision avoidance subsystem to achieve acceptable levels of safety and reliability. First, the SoA principles for collision avoidance against stationary objects are reviewed. Afterward, a novel image processing approach that uses deep learning and optical flow is presented. This approach is capable of detecting and generating escape trajectories against potential collisions with dynamic objects. Finally, novel models and algorithms combinations were tested, providing a new approach for the collision avoidance of UAVs using Deep Neural Networks. The feasibility of the proposed approach was demonstrated through experimental tests using a UAV, created from scratch using the framework developed.

Keywords: Artificial Intelligence, Collision Avoidance, Collision Dataset, Deep Learning, Drones, Image Processing, Machine Learning, Neural Network, Optical Flow, UAV

Resumo

Os veículos aéreos não tripulados (VANTs), embora dificilmente considerados uma nova tecnologia, ganharam recentemente um papel de destaque em muitas indústrias, sendo amplamente utilizados não apenas por amadores, mas também em situações profissionais de alta exigência, sendo expectável um impacto social massivo nos próximos anos. No entanto, a operação de VANTs está repleta de sérios riscos de segurança, como colisões com obstáculos dinâmicos (pássaros, outros VANTs ou objetos arremessados). Estes cenários de colisão são complexos para analisar em tempo real, às vezes sendo computacionalmente impossível de resolver com os algoritmos existentes, tornando o uso de VANTs um risco operacional e, portanto, reduzindo significativamente a sua aplicabilidade comercial em ambientes citadinos. Neste trabalho, uma arquitectura conceptual para VANTs autônomos e em rede é apresentada, com foco nos requisitos arquitetônicos do subsistema de prevenção de colisão para atingir níveis aceitáveis de segurança e confiabilidade. Os estudos presentes na literatura para prevenção de colisão contra objectos estacionários são revistos e uma nova abordagem é descrita. Esta tecnica usa técnicas de aprendizagem profunda e processamento de imagem, para realizar a prevenção de colisões em tempo real com objetos móveis. Por fim, novos modelos e combinações de algoritmos são propostos, fornecendo uma nova abordagem para evitar colisões de VANTs usando Redes Neurais Profundas. A viabilidade da abordagem foi demonstrada através de testes experimentais utilizando um VANT, desenvolvido a partir da arquitectura apresentada.

Palavras-chave: VANT, Drones, Segurança, Inteligência Artificial, Máquina Inteligente, Rede Neuronal, Conjunto de Dados de Colisão, Prevenção de Colisão.

Contents

List of Figures

List of Tables

Listings

Glossary

Artificial	Made or produced by human beings rather than occurring naturally, especially as a copy of something natural.
Artificial Intelligence	The simulation of human intelligence processes by machines, especially computer systems.
Big Data	Very large amounts of data, usually collected by companies or institutions. This type of data can be useful for companies, but only if they know how to glean information from them.
Clustering	Cluster analysis, or clustering, is an unsupervised machine learning task. It involves automatically discovering natural grouping in data. Unlike supervised learning (like predictive modeling), clustering algorithms only interpret the input data and find natural groups or clusters in feature space.
Collision	Instance of one moving object or person striking violently against another.
Collision Avoidance	Capability to detect and prevent a collision.
Computer	Electronic device which is capable of receiving information (data) in a particular form and of performing a sequence of operations in accordance with a predetermined but variable set of procedural instructions (program) to produce a result in the form of information or signals.
Computer Vision	Interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do.

Deep Learning	Type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher level features from data.
Detection	The action or process of identifying the presence of something concealed.
Distributed computing	System whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another from any system.The components interact with one another in order to achieve a common goal.
Drone	Refers to any unpiloted aircraft. Sometimes referred to as “Unmanned Aerial Vehicles" (UAVs), these crafts can carry out an impressive range of tasks, ranging from military operations to package delivery. Drones can be as large as an aircraft or as small as the palm of your hand.
Drone Swarms	Approach to the coordination of multiple drones as a system which consist of large numbers of less complex drones.
Framework	Abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software. It provides a standard way to build and deploy applications and is a universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate the development of software applications, products and solutions.
Intelligence	The ability to acquire and apply knowledge and skills.
Intelligent Systems	Technologically advanced machines that perceive and respond to the world around them.
Internet of Things	Term that encompasses everything involved in connecting everyday devices to the Internet, in order to collect data from them, exchange data between devices or control them from a distance.

Machine Learning	Method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.
Neural Network	Series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature.
Operating System	An operating system is system software that manages computer hardware, software resources, and provides common services for computer programs.
Path	Course of action, trajectory, or way of achieving a specified result.
Resilience	The capacity to recover quickly from difficulties; toughness.
Resolution	The solution to a problem.
Robotics	Robotics is the field based on science and engineering, which focuses on designing, creating, and building robots and the computer programs that control them.
ROS	ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

Smart Industry 4.0	Term which is surfacing to describe the current trend of automation and sharing of data in the manufacturing sectors.
Technology	Sum of any techniques, skills, methods, and processes used in the production of goods or services or in the accomplishment of objectives, such as scientific investigation. Technology can be the knowledge of techniques, processes, and the like, or it can be embedded in machines to allow for operation without detailed knowledge of their workings.
UAS	Means an unmanned aircraft and the equipment to control it remotely.
UAV	Any aerial aircraft operating or designed to operate autonomously or to be piloted remotely without a pilot on board.
Unmanned	Not carrying, staffed, or performed by people; not manned.

Acronyms

AA	Agglomerative Average
AAA	Artificial Algae Algorithm
ADS	Autonomous Driving System
AE	Auto Encoder
AFarCloud	Aggregate Farming in the Cloud
AI	Artificial Intelligence
AI4RealAg	Artificial Intelligence for Real Agriculture
ANN	Artificial Neural Network
ANSI	American National Standards Institute
API	Application Programming Interface
ASV	Autonomous Surface Vehicle
AUV	Autonomous Underwater Vehicle
AW	Agglomerative Ward
B3DO	Berkeley 3-D Object
BLOS	Beyond Line of Sight
CAL	Collision Aware Planner
CAP	Credit Assignment Path
CBAM	Convolutional Block Attention Module
CB-CNN	Channel Boosted Convolutional Neural Network
CDR	Conflict Detection and Resolution
CM	Command Multiplexer
CMPE-SE	Competitive Inner-Imaging Squeeze and Excitation
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPS	Cyber Physical Systems
CPU	Central Processing Unit
CSNDSP	Communication Systems, Networks and Digital Signal Processing
CSO	Cat Swarm Optimization
CSOA	Chicken Swarm Optimization Algorithm

CUDA	Compute Unified Device Architecture
CVPR	Conference on Computer Vision and Pattern Recognition
DAIS	Distributed Artificial Intelligent Systems
DAVIS	Densely-Annotated Video Segmentation
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DCA	Dynamic Collision Avoidance
DDT	Dynamic Driving Task
DL	Deep Learning
DNN	Deep Neural Network
DoCEIS	Doctoral Conference on Computing Electrical and Industrial Systems
EESP	Escape Elliptical Search Point
EKF	Extended Kalman Filter
ELU	Exponential Linear Unit
ESA	Elephant Search Algorithm
FE	Feature Extraction
FFAU	Framework for Fully Autonomous UAVs
FN	False Negative
FNN	Feedforward Neural Network
FP	False Positive
FPS	Frames Per Second
FSA	Fish Swarm Algorithm
GBCA	Genetic Bee Colony Algorithm
GNSS	Global Navigation Satellite System
GPU	Graphics Processing Unit
Grad-CAM	Gradient-weighted Class Activation Mapping
GRU	Gated Recurrent Unit
GUI	Graphical User Interface
GWO	Grey Wolf Optimization
HAL	Hardware Abstraction Layer
HN	Highway Network
I2C	Inter-Integrated Circuit

IDG	Image Data Generator
IEEE	Institute of Electrical and Electronics Engineers
IEoT	Intelligent Edge of Things
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
IMU	Inertial Measurement Unit
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute
KNN	K-Nearest Neighbor
LiDAR	Light Detection and Ranging
LoA	Level of Autonomy
LoG	Laplacian of Gaussian
LSTM	Long Short-Term Memory
M5G	Mobilizadores 5G
MCMC	Markov Chain Monte Carlo
MDPI	Multidisciplinary Digital Publishing Institute
MDS	Multidimensional Scaling
MFO	Moth Flame Optimization
MGU	Minimal Gated Unit
MINC	Materials in Context
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
MNV2	MobileNetV2
NAT	Network Address Translation
NIN	Network in Network
NN	Neural Network
NNP	Neural Network Pipeline
NYU	New York University
OAF	Optical Agglomerated Flow
OBC	On-Board Computer
OD	Obstacle Detection
ODD	Operational Design Domain
OEDR	Object and Event Detection and Response
OF	Optical Flow
OFC	Optical Flow Clustering
OME	Object Motion Estimator

OMPL	Open Motion Planning Library
OS	Operating System
OTE	Object Trajectory Estimation
PCA	Principal Component Analysis
PCC	Pedro-Carvalho Clustering
PID	Proportional Integral Derivative
PReLU	Parametric ReLU
RAN	Residual Attention Network
ReLU	Rectified Linear Unit
REST	Representational state transfer
RGB	Red, Green and Blue
RL	Reinforcement Learning
RMSE	Root Mean Square Error
RNN	Recurrent Neural Networks
ROS	Robot Operating System
ROV	Remotely Operated Underwater Vehicle
RPV	Remotely Piloted Vehicle
RTL	Return To Launch
RTP	Real-time Transport Protocol
RTSP	Real-time Streaming Protocol
RTT	Round Trip Time
SAE	Society of Automotive Engineers
SBC	Single Board Computer
SCA	Static Collision Avoidance
SDN	Software Defined Networking
SE	Squeeze and Excitation
SECREDAS	Security for Cross Domain Reliable Dependable Automated Systems
SfM	Structure from Motion
SGD	Stochastic Gradient Descent
SL	Supervised Learning
SoA	State-of-Art
SPI	Serial Peripheral Interface
SRDF	Semantic Robotic Description Format
STUN	Session Traversal of UDP Through NAT
SVM	Support Vector Machine
SYNTHIA	SYNTHetic Collection of Imagery and Annotations

t-SNE	T-distributed Stochastic Neighbor Embedding
TL	Transfer Learning
TN	True Negative
TP	True Positive
TRL	Technology Readiness Level
TURN	Traversal Using Relays around NAT
UAS	Unmanned Aircraft System
UASSC	Unmanned Aircraft Systems Standardization Collaborative
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UGV	Unmanned Ground Vehicle
UI	User Interface *
UL	Unsupervised Learning
URDF	Unified Robot Description File
URLLC	Ultra Reliable Low Latency Communications
USV	Unmanned Surface Vehicle
VGG	Visual Geometry Group
VOC	Visual Object Classes
WebRTC	Web Real-Time Communication
WOA	Whale Optimization Algorithm
WSS	WebSocket Secure
ZCA	Zero-phase Component Analysis

Chapter 1

Introduction

1.1 Context and Motivation

It is the 'Era' of Unmanned Aircraft Systems (UASs), an all-encompassing term which includes the aircraft or the Unmanned Aerial Vehicle (UAV), the ground-based controller (the person operating the machine), and the communication system connecting the two, commonly known as *Drones*. Today, UAVs are revolutionizing the world and businesses in a way that hardly anyone could have ever imagined. An UAV is an aircraft without a human pilot aboard. The term UAVs includes both autonomous aircrafts and Remotely Piloted Vehicles (RPVs).

This rapid evolution of UAVs increases the need of safer and more reliable solutions. For reliable solutions using UAVs above cities skylines, it is necessary that the solution is completely safe and works regardless of the world conditions and unexpected events, being necessary to create a collision-free architecture that is agnostic of the environmental constraints, able to rationalize and give answers to new events in realtime. The lack of such architecture has lead to multiple disasters in the past (BBC, 2016; BBC, 2017; Canada, 2017; Caron, 2017; Goglia, 2017; Rawlinson, 2016; Tellman and News, 2018) that will tend to augment with the increasing number of UAVs in operation.

Most of the commercial UAVs are equipped with Red, Green and Blue (RGB) cameras, which create the opportunity to develop new algorithms that use the data from these cameras to create solutions that are capable of avoiding collisions. This algorithm can either predict the incoming collision and/or generate escape trajectories. Furthermore, it is vital to have a framework capable of integrating different modules that enable new levels of autonomy. For example, an algorithm that handles the collision avoidance with static objects should be easily replaced, not affecting the performance of other algorithms that are running in parallel.

1.2 Research Question and Hypothesis

For autonomous vehicles to be completely reliable in a collaborative network, with both static and dynamic obstacle avoidance, one question must be answered:

Research Question	What could be an adequate conceptual framework for representing the behavior of individually or networked unmanned aerial vehicles, ensuring reliability and safety in the flight, regardless of the environmental conditions and unexpected events?

In the seek of a good answer for the proposed research question, different hypotheses with potential solution approaches were advocated utilizing the knowledge from fellow researchers, being summarized in chapter 2. The considered hypotheses are:

Hypothesis 1	The proposed framework can be built utilizing different SoA blocks, that are present in distinct fields that use concepts of autonomous vehicles, such as aviation or autonomous cars, but are not benchmarked nor integrated properly. Furthermore, new blocks that do not exist can be developed and integrated into the developed framework.

Hypothesis 2	The proposed framework can be built utilizing different nodes, that are based on SoA architectures, which handle atomic tasks alone but realize complex tasks in collaboration . Furthermore, new complex blocks that don't exist can be developed utilizing AI technology and then integrated into the developed framework.

The main goal of this research work is then to design and develop the proposed framework and validate it with a set of UAVs and make further improvements with tests. Not only that, but the building blocks will be deeply researched and improved, which will also involve general improvements in machine learning architectures and models.

1.3 Goals

The purpose of this thesis is to enhance UAVs autonomy, focusing on safety and resilience. For this, a new UAV architecture, both on software and hardware level, is presented. This UAV can recognize and avoid multiple types of collisions. With this in mind, four goals are planned to be achieved:

- Evaluate State-of-Art (SoA) of UAVs, their planning and collision avoidance algorithms, and study the latest advances on Artificial Intelligence (AI), in order to tailor a solution for the proposed problem;
- Propose a custom drone architecture capable of handling SoA planners with collision awareness and with the ability to handle events in realtime;

- Develop a collision detector with spatial awareness, which will provide an escape direction;
- Create and publish collision datasets that can be reutilized and expanded by the opensource community to develop new and more reliable algorithms for collision avoidance.

The algorithms and architectures proposed in this work should be generic enough so that it is possible to adapt to other types of autonomous vehicles, such as autonomous cars or autonomous surface vehicles. In addition, studying the problem from the drone's perspective provides a 3D vision of the problem, which should be practical to adapt to 2D.

1.4 Contributions

Throughout the work, it was intended that the system could cope with natural environmental adversities and complete its task with precise and safe results. Having this scope in mind, a review of the literature was conducted, highlighting the latest developments in the field that act as the base for the proposed solution. Then, a set of contributions were developed, such as:

1. **A Framework for Fully Autonomous UAVs:** a framework that allows the integration and modification of different algorithms, facilitating complex UAVs missions on Beyond Line of Sight (BLOS) scenarios.
2. **Open-sourced Datasets:** data which is open for anyone and everyone for access, modification, reuse, and sharing. Two datasets were developed and made available to the community:
    - ColANet - A dataset with a different type of collisions produced from multiple videos from the internet. This was cataloged and labeled accordingly.
    - BallANet - A dataset focusing on the collisions between thrown ball and a UAVs, which were also cataloged and labeled accordingly.
3. **Multiple dynamic collision avoidance algorithms**: that can be adapted to different circumstances, depending on the available dataset, and the UAV camera specifications. The collision avoidance of thrown objects (or similar) is a topic that is not deeply explored, and therefore these were, in most cases, novel algorithms/approaches (which use Neural Networks and Optical Flow).

During the development of this work, the author published **7 journal papers** regarding some of the algorithms and ideas explored directly and indirectly by the thesis. The publications can be found in journals such as Applied Sciences, Institute of Electrical and Electronics Engineers (IEEE) Access, Multidisciplinary Digital Publishing Institute

(MDPI) Drones, and Remote Sensing. Moreover, it were published **7 conference papers** in several symposiums, including Doctoral Conference on Computing Electrical and Industrial Systems (DoCEIS), the Intelligent Systems Conference, the International Symposium on Communication Systems, SAFECOMP, and the International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP). A summary of all the articles published can be found in table 1.1.

Table 1.1: Articles published during the thesis development.

	Conference	Journal
1st Authour	(2020) Pedro, D., Mora, A., Carvalho, J., Azevedo, F., & Fonseca, J. **ColANet: A UAV Collision Avoidance Dataset.** In: DoCEIS 2020 (2021) Pedro D., Rato R.T., Matos-Carvalho J.P., Fonseca J.M., Mora A. **Flow Empirical Mode Decomposition.** In: IntelliSys 2021. (2021) Pedro D., Lousã P., Ramos Á., Matos-Carvalho J.P., Azevedo F., Campos L. **HEIFU - Hexa Exterior Intelligent Flying Unit.** In: SAFECOMP 2021 Workshops.	(2019) Pedro, D., Tomic, S., Bernardo, L., Beko, M., Oliveira, R., Dinis, R., Pinto, P., & Amaral, P. **Algorithms for Estimating the Location of Remote Nodes Using Smartphones.** In: IEEE Access. (2020) Pedro, D., Matos-Carvalho, J. P., Azevedo, F., Sacoto-Martins, R., Bernardo, L., Campos, L., Fonseca, J. M., & Mora, A. **FFAU—Framework for Fully Autonomous UAVs.** In: Remote Sensing. (2021) Pedro, D., Matos-Carvalho, J. P., Fonseca, J. M., & Mora, A. **Collision avoidance on unmanned aerial vehicles using neural network pipelines and flow clustering techniques** In: Remote Sensing.
Co-author	(2019) Matos-Carvalho J.P., Pedro D., Campos L.M., Fonseca J.M., Mora A. **Terrain Classification Using W-K Filter and 3D Navigation with Static Collision Avoidance.** In: IntelliSys 2019 (2020) Campos, L. M., Ribeiro, L., Karydis, I., Karagiannis, S., Pedro, D., Martins, J., Marques, C., Armada, A. G., Leal, R. P., Lopez-Morales, M. J., Velez, F. J., Sebastiao, P., & Ramos, A. R. **Reference Scenarios and Key Performance Indicators for 5G Ultra-dense Networks.** In: 12th CSNDSP (2020) Pino, M., Matos-Carvalho, J. P., Pedro, D., Campos, L. M., & Costa Seco, J. **UAV Cloud Platform for Precision Farming.** In: 12th CSNDSP (2022) Matos-Carvalho, J. P. , Vong, A., Pedro, D., Tomic, S., Beko, M., Azevedo, F., Mora, A. **Open-source mapping method applied to thermal imagery.** In: Computing Conference	(2019) Matos-Carvalho, J.P.; Moutinho, F.; Salvado, A.B.; Carrasqueira, T.; Campos-Rebelo, R.; Pedro, D.; Campos, L.M.; Fonseca, J.M.; Mora, A. **Static and Dynamic Algorithms for Terrain Classification in UAV Aerial Imagery.** In: Remote Sensing (2021) Nakama, J., Parada, R., Matos-Carvalho, J. P., Azevedo, F., Pedro, D., & Campos, L. **Autonomous environment generator for uav-based simulation.** In: Applied Sciences (Switzerland) (2021) Vong, A., Matos-Carvalho, J. P., Toffanin, P., Pedro, D., Azevedo, F., Moutinho, F., Garcia, N. C., & Mora, A. **How to build a 2d and 3d aerial multispectral map?—all steps deeply explained.** In: Remote Sensing. (2021) Moreira, M., Azevedo, F., Ferreira, A., Pedro, D., Matos-Carvalho, Ramos, A., Loureira, R., Campos, L. **Precision Landing for Low-Maintenance Remote Operations with UAVs** In: MDPI Drones

Furthermore, to disseminate the work developed under the scope of this thesis, some of the algorithms developed were applied in **6 research projects** financed by P2020 and H2020, such as: Mobilizadores 5G (M5G), Aggregate Farming in the Cloud (AFarCloud), Security for Cross Domain Reliable Dependable Automated Systems (SECREDAS), Artificial Intelligence for Real Agriculture (AI4RealAg), Distributed Artificial Intelligent Systems (DAIS), and Intelligent Edge of Things (IEoT) . All these projects share the desire to push UAVs to global usage, but also the consternation of the possible risks that this may provoke.

1.5 Dissertation Structure

This dissertation is organized in seven chapters:

- **Chapter 1: Introduction** presents the work and proposes the implementation approach. The motivations are outlined and the architecture is explained.
- **Chapter 2: Related Work** summarizes the technologies behind the various collision avoidance techniques. It is presented the main characteristic of Autonomous Vehicles, their planning algorithms, techniques for performing image processing with neural networks and how to use these networks over a drone video feed.
- **Chapter 3: Framework for Fully Autonomous UAVs** puts forward a framework architecture that contains all the necessary UAVs modules that allow a collision safe flight. Each of the modules is described and their interconnections and responsibilities are highlighted.
- **Chapter 4: UAV Collision Avoidance Dataset** presents two novel UAV collision datasets. These datasets provide a foundation for training new Machine Learning (ML) algorithms that are required to handle the collision avoidance problem with high efficiency and robustness. It is also shown that using these datasets is easy to build new Neural Network (NN) models and test them.
- **Chapter 5: Dynamic Collision Avoidance** focuses the main contributions of this work. A detailed explanation of the algorithms develop is presented, which make use of the datasets of chapter 4.
- **Chapter 6: Applications and Results** explains how the proposed solutions can be deployed in simulation or in real scenarios, showcasing some of the results obtained. This chapter also serves as a guidance for future improvements on the algorithms to increase their Technology Readiness Level (TRL).
- **Chapter 7: Conclusion and Future Work** summarizes the work developed. In this chapter, some comments, criticisms and plans are presented for the future work to be developed.

Chapter 2

Related Work

This chapter consolidates the core concepts that are tackled along with the dissertation, such as Autonomous Vehicles, Collision Avoidance, and AI with a focus on Deep Neural Networks (DNNs).

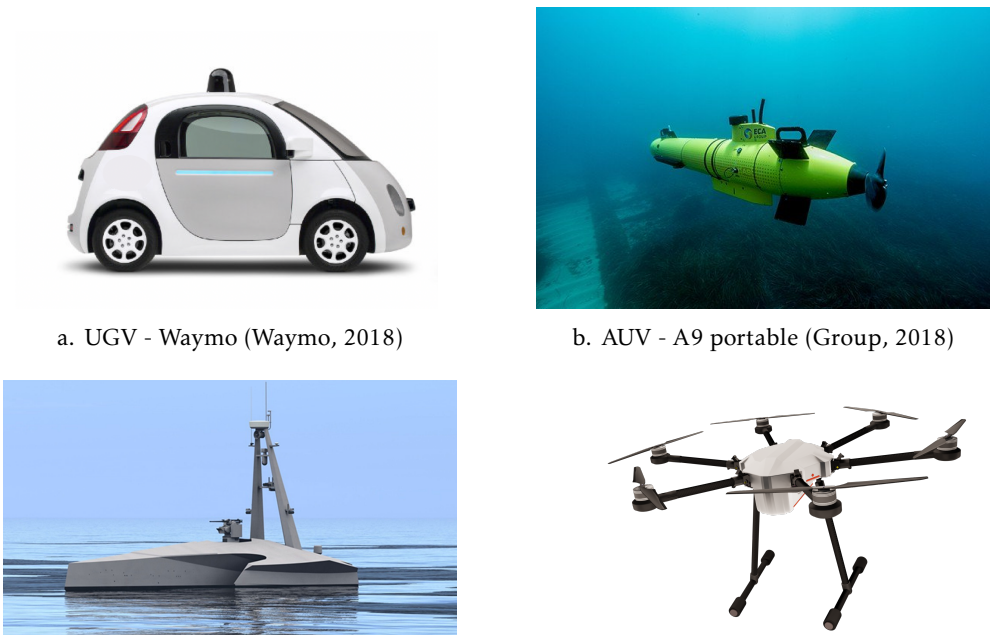
2.1 Autonomous Vehicles

An autonomous vehicle, sometimes known as a robotic or driverless vehicle, represents any vehicle without using a human driver. Its main objective is to integrate a set of sensor technologies, control systems, and actuators to sense the environment, determinate what to do and perform tasks safely and reliably (Davis et al., 2014; Litman, 2014).

The autonomous vehicles can be divided into four categories, as is depicted in Figure 2.1, and described below :

- Unmanned Ground Vehicles (UGVs) - This set represents all vehicles that are capable of sensing their environment and move on the ground safely with little or no human input. The most known type of UGV is the autonomous car, as is presented as an example in Figure 2.1 (a).
- Autonomous Underwater Vehicles (AUVs) - Part of a large group of submarine systems that includes vehicles, controlled and fed from the surface by an operator (pilot) via a wire (also called "umbilical cord"), or via a remote control. On Figure 2.1 (b) is represented an A9 portable AUV (Group, 2018).
- Autonomous Surface Vehicles (ASVs) - Represent boats that operate at the water surface level without a crew. ASVs are valuable in oceanography, as they are more capable than moored or drifting weather buoys and far cheaper than similar ships and research vessels. One ASV can be visualized in Figure 2.1 (c).

- Unmanned Aerial Vehicle (UAVs) - Commonly known as drones, they are aircrafts without a human pilot onboard. UAVs are a component of an UAS which include a UAV, a ground-station controller, and a system of communications between the two. The flight of UAVs may operate with various degrees of autonomy: either under remote control by a human operator or autonomously by onboard computers. On Figure 2.1 (d) is presented an Hexacopter UAV.



a. UGV - Waymo (Waymo, 2018)

b. AUV - A9 portable (Group, 2018)

c. ASV - SAAB Halcyon (Saab, 2018)

d. UAV - HEIFU (Pedro et al., 2021)

Figure 2.1: Different types of autonomous vechicles such as UGV, AUV, ASV and UAV.

Although they are all considered autonomous vehicles in general terms, they can have different levels of autonomy. These levels of autonomy vary depending on the vehicle control system and auxiliary systems. Systems that currently need improvement include the main vehicle navigation system, the location system, the electronic map, the map matching, the global path planning, the environment perception, the laser perception, the radar perception, the visual perception, the vehicle control, the perception of vehicle speed, and direction and the vehicle control method (Zhao et al., 2018).

2.1.1 Level of Autonomy

The purpose of the Level of Autonomy (LoA) specifies the degree of autonomy wanted, needed, or required during operations. The term LoA refers to the degree that the system and the operator can intervene when an operation is taking place (the extent to which a human or machine has control in different stages of the operation) and is being developed for many years (Sheridan, 1993). Another aspect in LoA is the context of machine-machine collaboration, which assumes that machines decide their LoA based

on the problem. The latter case is also applicable when there are humans in the loop. However, a significant problem is divided and solved between intelligent systems, which also decide their roles through defining their autonomy levels. This setting has practical applications. On automated ground vehicles, the autonomy level as defined in Society of Automotive Engineers (SAE) J3016 (Society of Automotive, 2014) can be used to define the autonomy level. However, the classification is not based on the overall task (e.g., a strategic decision regarding path planning or task execution), being focused on the direct control task and Autonomous Driving System (ADS).

The levels include not only the ground vehicle and the controlling/involved person but also the Operational Design Domain (ODD) since the autonomy can be restricted based on the environment (e.g., an agricultural vehicle that needs a human operator for driving on public roads). Based on this, six levels are defined, as described in table 2.1.

Table 2.1: Automation Levels.

Level	Name	Narrative Definition	DDT		DDT fallback	ODD
			Sustained lateral and longitudinal vehicle motion control	OEDR		
Driver performs part or all of the DDT						
0	**No Driving Automation**	The performance by the driver of the entire DDT, even when enchanced by active safety systems.	Driver	Driver	Driver	n/a
1	**Driver**	The sustained and ODD-specific execution by a driving automation system of either the lateral or the longitudinal vehicle motion control subtask of the DDT (but not both simultaneously) with the expectation that the driver performs the remainder of the DDT.	Driver & System	Driver	Driver	Limited
2	**Partial Driving**	The sustained and ODD-specific execution by a driving automation system of both the lateral and longitudinal vehicle motion control subtasks of the DDT with the expectation that the driver completes the OEDR subtask and supervises the driving automation system.	System	Driver	Driver	Limited
ADS ('System') performs the entire DDT (while engaged)						
3	**Conditional Driving Automation**	The sustained and ODD-specific performance by an ADS of the entire DDT with the expectation that the DDT fallback-ready user is receptive to ADS-issued requests to intervene, as well as to DDT performance-relevant system failures in other vehicles systems, and will respond appropriately.	System	System	Fallback ready user	Limited
4	**High Driving Automation**	The sustained and ODD-specific performance by an ADS of the entire DDT and DDT fallback without any expectation that a user will respond to a request to intervene.	System	System	System	Limited
5	**Full Driving Automation**	The sustained and unconditional (i.e., not ODD-specific) performance by an ADS of the entire DDT and DDT fallback without any expection that a user will respond to a request to intervene.	System	System	System	Unlimited

Similar nomenclatures were proposed depending on the domain and application. This thesis focus on the UAVs due to their axis freedom and movement. Typically, an UAV can move towards any 3D trajectory, being harder or impossible for some other autonomous vehicles. All algorithms presented in this thesis are intended to be as generic as possible, so they can be easily adapted to other types of vehicles, but for the sake of specification and performance optimization, all the work is around UAVs. On a UAV the LoA is still being explored, as there is a big lack of standardization and regulation, which is mainly being pushed by the American National Standards Institute (ANSI) on the Unmanned Aircraft Systems Standardization Collaborative (UASSC). It is emphasized that the topic of LoA is a top priority, and it is required to clarify that there are significant differences between "fully autonomous" and "fully automated" systems. In this work, the algorithms will always target fully autonomous behaviors, with safety architecture in mind.

2.1.2 Types of UAVs

This section presents an analysis of different UAVs models and their mode of operation (Valavanis and Vachtsevanos, 2015). It introduces some important concepts which will help the reader to better understand the content and use cases of this dissertation.

UAVs are growing in number and variety of potential applications. Some possible applications are pollution and forest fire monitoring, delivery of retail products, border patrol, aerial mapping and surveillance, traffic monitoring, precision agriculture, or search and rescue operations (P. Valavanis, 2007).

Usually, users tend to divide UAVs into four categories (UAV, 2019) that are summarized on table 2.2. It is possible to advance that the Multi-Rotor solution will be a proper choice for the main use cases that are going to be addressed in this thesis, given that this solution can easily maneuver in any axis and is considered to be low cost.

The vehicle shown in Figure 2.1 (d) is the solution used in this thesis, which is a hexacopter, thus possessing all of the maneuverability enunciated previously. Moreover, it carries a gimbal with a depth camera, which will help with the Obstacle Detection (OD) tasks.

2.1.3 UAV typical software architecture

Most UAVs use a processor that collets information from the sensors and stabilizes the aircraft. This is usually called the UAV flight controller (Chao et al., 2010). There are multiple types of flight controllers, but the most common share a similar architecture to the diagram in Figure 2.2. The indicated architecture is divided into five main blocks:

- **Hardware components:** there are multiple types of processors that can be used to ran a flight controller. An open-source option is the Pixhawk, but there are several others, such as the HEIFU board, the Bebop2 board, or the Navio2 board.

Table 2.2: UAVs' Categories' Comparison.

	Advantages	**Drawbacks**
Fixed-Wing	Cover wide areas, due to the longer flight of time; The main function of the motors is to move the vehicle forward (more efficient).	Low maneuverability; Considerably expensive; Take off/land is and may require mechanisms such as catapult, runway or hand launching.
Single-Rotor Helicopter	Possibility of vertically take-off and land; Can carry a heavy payload; present high autonomy (time of flight).	If the main propeller fails, it completely loses control. Less stable
Multi-Rotor	Cheapest solution ; high maneuverability and control over position.	Do not allow high speeds ; do not present good autonomy given the high effort required from the motors to hover. This makes this solution unsuitable to large scale coverage.
Fixed-Wing Hybrid	Combines the previous approaches: contains both the hover and forward flight modes.	Rather recent solution; higher price due to the extra technology requirements.

- **Operating System (OS):** depending on the selected hardware, a compatible OS must be adopted.
- **Flight controller code:** it contains all the logic required to read data from the UAV sensors and send commands to the actuators on the Hardware Abstraction Layer (HAL). Furthermore, it has libraries to fuse different types of data (e.g., using Extended Kalman Filter (EKF)) and controls the UAV on the desired mode (e.g., Stabilize, Loiter, Alt-hold, Return To Launch (RTL), Land).
- **Communication layer:** on most cases, the flight controller allows all the information that it is acquiring and processing to be periodically listened to by another system. In most of open-source flight controllers, this communication is done by the Mavlink protocol (Koubaa et al., 2019).
- **External integrations:** multiple types of external devices can make use of the information published via Mavlink. For example, a ground station that receives this data via a radio link, or an On-Board Computer (OBC) that can compute order type of algorithms and provide another layer of computation (e.g., Jetson Nano running AI algorithms).

Furthermore, additional external sensors can be integrated on the flight controller, which usually communicates via Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C) or Serial.

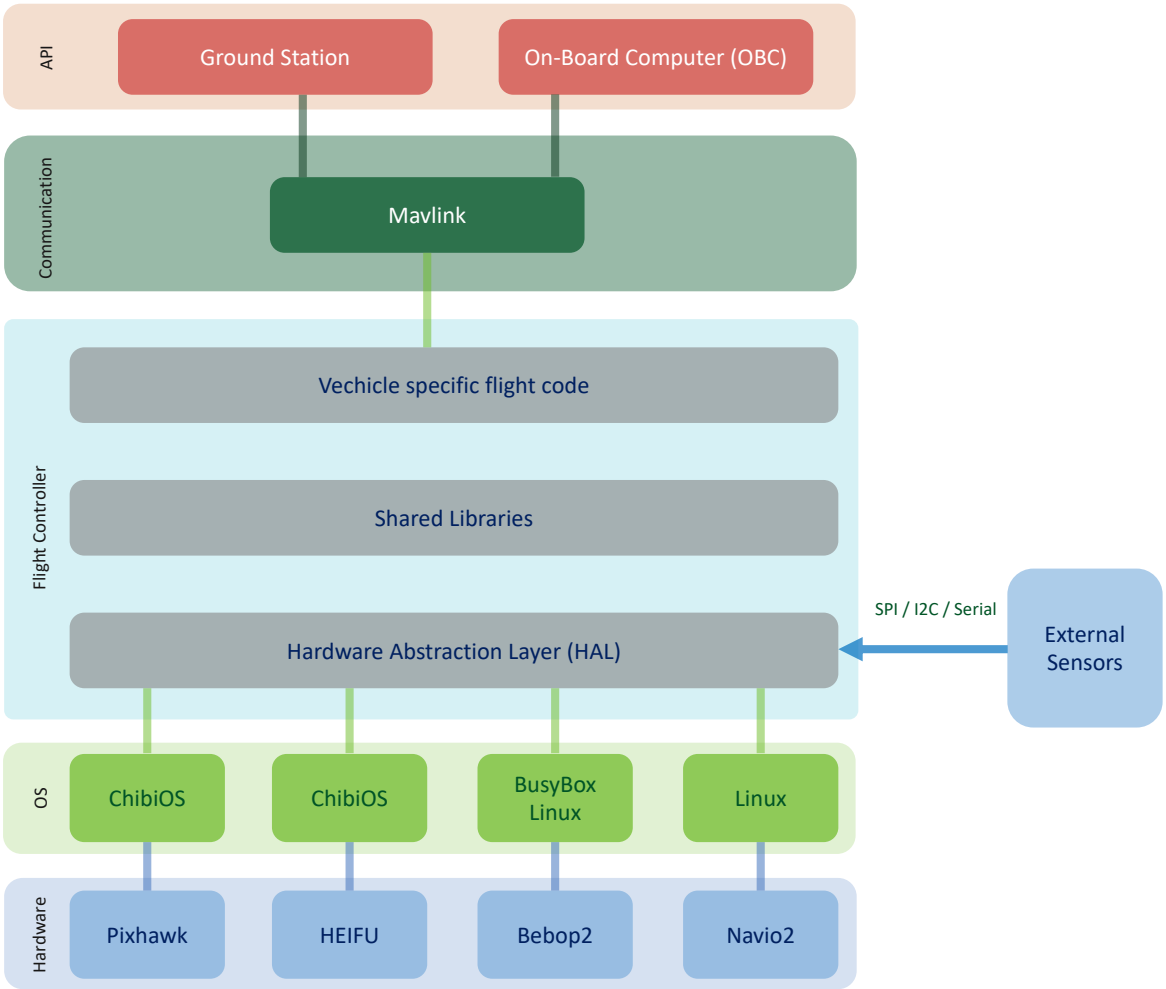


Figure 2.2: UAV flight controller architecture.

Examples of the most known open-source flight controllers are: ArduCopter (Ardupilot, 2013), MultiWii (MultiWii, 2020), OpenPilot (Hotz, 2020), PX4 autopilot (PX4 Open Source Autopilot, 2019), and Paparazzi (Brisset and Hattenberger, 2008).

2.2 Collision Avoidance

Path planning from location A to location B, while simultaneously avoiding obstacles and reacting to environment changes, are simple tasks for humans but not so straightforward for autonomous vehicles. These tasks present challenges that each mobile robot needs to overcome to become fully autonomous. A robot uses sensors to perceive the environment (up to some degree of uncertainty) and to build or update its environment map. To determine appropriate motion actions that lead to the desired goal, it can use different decision and planning algorithms. For an adequate path planning, the robot's kinematic and dynamic constraints should be considered.

Path planning is used to solve problems in different fields, from simple spatial route

planning to the selection of an appropriate action sequence that is required to reach a certain goal. Since the environment is not always known in advance, this type of planning is often limited to the environments designed in advance and environments that can be described accurately enough before the planning process. Path planning can be used in fully known or partially known environments, as well as in entirely unknown environments where sensed information defines the desired robot motion.

Path planning in known environments is an active research area (Jiang and Ma, 2020; Zhang et al., 2018) and presents a foundation for more complex cases where the environment is not known *a priori*. This section presents an overview of the most common path planning approaches applicable to UAVs.

In this thesis, the collision avoidance problems are divided into two categories:

- Static collision s_c - This category includes collisions between the considered vehicle and any obstacle that moves considerably slower than it. In this work, it is considered that using the world as a referential, an object will produce a static collision if it is moving bellow 5% of the vehicle maximum speed v_{max}.
- Dynamic collision d_c - This set represents collisions between the vehicle and any obstacle that moves at a speed that is hard for the path planner to plan a safe path avoiding the collision. It is considered that using the world as a referential, an object will produce a dynamic collision if it is moving at speed o_{max} faster than 5% of the v_{max}.

These two categories can be formulated as:

$$\begin{cases} s_c \in o_{max} \leqslant 0.05 \cdot v_{max} \\ d_c \in o_{max} > 0.05 \cdot v_{max} \end{cases} \tag{2.1}$$

It is important to note that the heuristic of 5% maximum speed is not of great importance because, in the end, the Static Collision Avoidance (SCA) algorithm should be able to handle obstacles that are moving at a faster speed (up to 25% of its maximum speed) without being eluded into erroneous paths by fast-moving objects. Similarly, the collision avoidance algorithm should be able to handle all sorts of collisions even if the path planner that considers static objects do not generate a trajectory that avoids an obstacle.

2.2.1 Static Collision Avoidance

As stated by Marr in (Marr, 1980), most of the structures in the visual world are rigid or at least nearly so. This statement is the starting point for most collision avoidance algorithms. The core concept is that the planner should try to create a plan that maximizes the distance to obstacles while heading to the way-point.

A vast number of methods have been proposed to automate air traffic Conflict Detection and Resolution (CDR) (Kuchar and Yang, 2000). Most of the methods can be separated by dimensions of state information (vertical, horizontal, or three-dimensional, 3-D),

methods of dynamic state propagation (nominal, worst case, or probabilistic), conflict resolution method (planned, optimized, force field, or manual), maneuvering dimensions (speed change, lateral, vertical, or combined maneuverer), and management of multiple aircraft conflicts (pairwise or global).

This section focus specifically on multi-rotor UAVs solutions using the real-time obstacle avoidance algorithms with the most common sensorial configurations, namely monocular cameras (Kovacs, 2016), Light Detection and Ranging (LiDAR) (Hrabar, 2012; Merz and Kendoul, 2011), stereo cameras (Hrabar, 2012) or their combination (Hrabar, 2012).

LiDARs are active sensors. This means that they are insensitive to the environment light, providing better accuracy than stereo cameras and good performance for far obstacles. The low processing power required makes them efficient for real-time applications. However, the data collected is produced sequentially, and the maximum range is limited. Azevedo et al. (Azevedo et al., 2017) presented a solution using a LiDAR-based real-time collision avoidance algorithm, denoted by Escape Elliptical Search Point (EESP), with the ability to be integrated into autonomous and manned modes of operation. Some other examples of LiDAR-based detection systems are (Merz and Kendoul, 2011) and (Ramasamy et al., 2016).

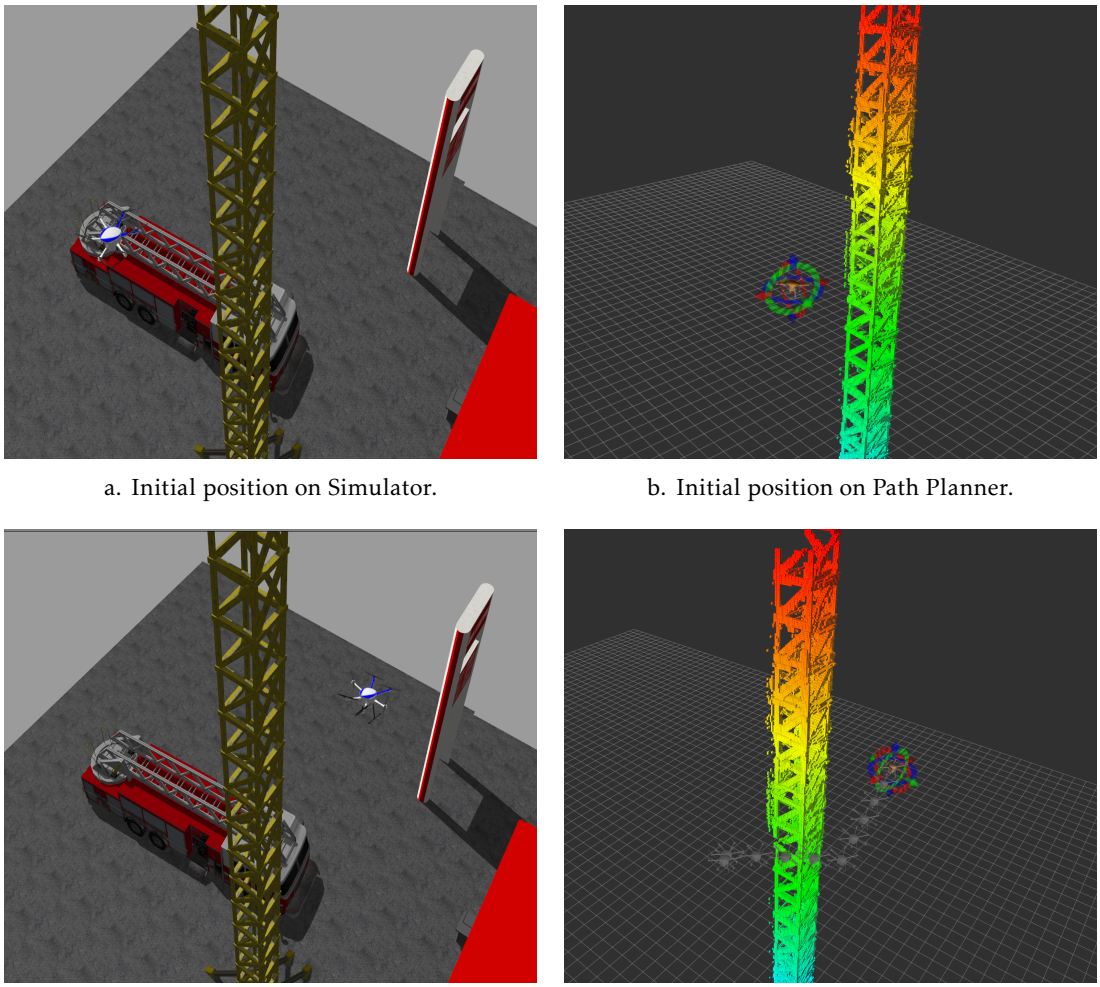
Stereo cameras are another source of depth information. Global Shutter cameras can provide a snapshot of the environment for a given instant and generate dense 3D depth information, with RGB color map to each point in space, which enables the possibility of detecting objects from a long distance (depending on the baseline between the cameras). The main disadvantage of this solution is the dependency on the visual environment conditions and the necessary processing computing power. Besides that, its range accuracy decreases with range squared (Hrabar, 2008).

In addition to these two range sensors, there are different solutions (Kovacs, 2016; Li and Ling, 2015; Magree et al., 2014; Yang et al., 2017) that use monocular cameras and radars to generate 3D maps for collision avoidance with static structures.

The obstacles can be represented on a map by a simple point cloud with the measurements given by a depth sensor. However, this is computationally costly and can compromise the real-time requirements. For reducing this cost, data can be clustered, resulting in a sparse representation. Another disadvantage of this method is that it is hard to distinguish between clear and unmapped spaces. Using a point cloud as input, the memory space required for storing the map information can be reduced using techniques like the representation using Octrees, Octomaps, or Voxel Grids (Hornung et al., 2013). Other ways of representing occupancy maps are analyzed and summarized in (Burgard et al., 2019).

For a better understanding of all the presented concepts, a Hexacopter drone was simulated using Gazebo and Robot Operating System (ROS) (Joseph, 2015), which is represented in figure 2.3. In this example, the drone is processing the environment and generating an Octomap. Afterwards, a destination way-point was added, and the path

planned for the drone was represented as a drone model with higher transparency.



a. Initial position on Simulator.

b. Initial position on Path Planner.

c. End position on Simulator.

d. End position on the Path Planner.

Figure 2.3: Hexacopter navigation on a simulated environments using Static Collision Avoidance awareness.

Another obstacle avoidance maneuver is presented in (Hrabar, 2011). In this case, the vehicle is considered a sphere, which simplifies the collision calculations and constructs a safety volume around it. Whenever an obstacle enters the safety volume, it constructs an ellipsoid area around the obstacle and searches for a point that allows a clear path from the current position to the escape point and also ensures no collisions through a defined distance from the escape point, on the direction to the way-point. If no clear path is found, it extends the ellipse radius (a certain number of times) and performs another search. If no clear path is found with the maximum ellipse radius, the aircraft will alert the pilot and remain in the same position until the pilot takes control.

This method has the advantage of allowing an uninterrupted flight for avoiding the obstacle while having considerable low processing without a clear necessity to recalculate the trajectory considering arbitrary avoidance points.

Having this line of thought, Sabatini et al. (Sabatini et al., 2014) implemented an obstacle avoidance ellipsoid-shaped safety zone around obstacles. The planning algorithm for obstacle avoidance takes into account the aircraft dynamics. For example, when the aircraft is moving with high velocity and/or acceleration, the time to find an alternative path and the distance to the obstacle are the major inputs of the cost function, as they are the main parameters to be considered in critical situations.

2.2.2 Dynamic Collision Avoidance

To prevent a collision with a dynamic obstacle (such as an animal) or an incoming object (such as a thrown ball), a UAV needs to detect it as fast as possible and execute a safe maneuver to avoid it. The higher the relative speed between the UAV and the object, the critical the role of perception latency becomes. This leads us to conclude that perception latency is the time necessary to perceive the environment and process the captured data to generate control commands (Andrew, 2001; Gallup et al., 2008; Mueggler et al., 2015).

Compared to SCA algorithms, the Dynamic Collision Avoidance (DCA) algorithms haven't yet been explored since the task is much harder. There are some works, such as the one from Poiesi and Cavallaro, where multiple image processing algorithms that estimate the time of collision of incoming object are explored (Poiesi and Cavallaro, 2017). The detection is accurate, but the algorithm takes more than 10 seconds to process each frame, making the solution not applicable in real-time scenarios with SoA hardware. Also, Falanga et al. (Falanga et al., 2019) delved into the event cameras to generate a computing efficient sensing pipeline that was capable of avoiding a ball thrown towards a quad-copter at speeds up to 9 m/s similar to the work done in (Mueggler et al., 2015).

2.3 Artificial Intelligence

> *Some people call this artificial intelligence, but the reality is this technology will enhance us. So instead of artificial intelligence, I think we'll augment our intelligence. — Ginni Rometty*

Technology is rapidly evolving, and engineers are seeking harder and harder tasks to solve. In computer science, the core of the SoA is resolving complex tasks using Artificial Intelligence, so it's important to have a clear idea of these concepts and the correct roadmap for evolution (Bengio et al., 2013; Schmidhuber, 2015). Detailed analysis on SoA Artificial Intelligence is presented in appendix A, and in this chapter, it was chosen to stress out the most relevant topics.

2.3.1 Data Sets

In the last decade, there has been an increasing number of publications of datasets that are enabling the development of new ML models and solutions. In this section it is

presented a review of the existing datasets, highlighting their relevance and impact in the field. These datasets were selected using both a criterion of usefulness for the community (novel data or utility scenarios) and relevance (amount of citations reports, usage on benchmarks zoos, scientific quality extrapolated by top-tier conferences and journals). As can been found in some other works (Garcia-Garcia et al., 2018), our analyses will be split according to their data representation, 2D or RGB datasets, 2.5D or RGB-Depth datasets, and 3D or video (volumetric) datasets.

2.3.1.1 2D image Datasets

In the last years, most of the ML algorithms were developed using 2D datasets that tried to understand the correlation between pixels and classify images. In this section, it is described most of the 2D datasets, whether they are RGB or greyscale.

- Adobe's Portrait Segmentation (Shen et al., 2016b): Dataset of 1800 portrait images gathered from Flickr, which were cropped to 800x600, and the background and face were annotated pixel by pixel.
- CamVid (Brostow et al., 2009; Brostow et al., 2008) : Scene background understanding dataset with around 700 images of 960x720, manually annotated with 32 classes.
- Cityscapes (Cordts et al., 2015): Dataset of urban street scenes understanding, which was recorded from 50 cities during multiple days and different weather conditions. It has 30 different classes grouped into eight categories.
- Densely-Annotated Video Segmentation (DAVIS) (Perazzi et al., 2016): Dataset of 50 high-definition sequences of up to 4219 frames with pixel-wise annotations with four different categories: human, animal, vehicle, and object.
- ImageNet (Jia Deng et al., 2009; Russakovsky et al., 2015): Image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently, there exists an average of over five hundred images per node.
- Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) (Geiger et al., 2013): Popular dataset used in mobile robotics and autonomous vehicles. It has a large variety of sensors, being possible to use high-resolution RGB, greyscales camera arrays, or 3D laser scans.
- LabelMe (Russell et al., 2008): MIT dataset project that contains digital images with annotations, which are dynamic, free to use, and open to public contribution. It's used in computer vision research having more than 200 000 images, with more than 1 000 000 labeled objects.

- Materials in Context (MINC) (Bell et al., 2015) : Dataset used for material classification with 7061 images of (on average) 800x500 that have 23 different labels. Most of the that of this dataset is from OpenSurfaces dataset (Bell et al., 2013).

- Microsoft Common Objects in Context (COCO) (Lin et al., 2014b): One of the most used datasets for image processing. It consists of labeled 100 000 images of 80 classes, some with a different types of annotations that are useful for different challenges such as image recognition, captioning, pose estimation, and segmentation.

- Modified National Institute of Standards and Technology (MNIST) (LeCun et al., 1998): The MNIST database of handwritten digits has a training set of 60 000 examples and a test set of 10 000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It's one of the most used datasets in the world for introduction to those initiating in the ML area. It was used by Yann LeCun et al. (LeCun et al., 1998), which started the Convolutional Neural Network (CNN) revolution.

- Pascal Visual Object Classes (VOC) (Everingham et al., 2014): Some researchers consider this the most popular dataset for semantic segmentation, so most of the SoA methods in the literature are submitted and tested using it. It consists of 21 categorized classes for 3000 images.

- SYNTHetic Collection of Imagery and Annotations (SYNTHIA) (Ros et al., 2016): Dataset of photo-realistic images of a virtual city, completely annotated, created scene understanding on driving context in urban scenarios. It contains 11 classes and over 14 000 images from rendered video.

- Youtube-Objects (Ronneberger et al., 2015): Database of Youtube videos annotated with objects with 10 Pascal VOC classes. It does not have pixel-wise annotations, but there are annotated frames.

2.3.1.2 RGB-Depth Datasets

In the past years, cameras with multiple sensors that capture both RGB and depth are becoming more popular due to their decrease in price and increase of applications. Below are shortly described a few RGB-Depth datasets:

- New York University (NYU) Depth v2 (Silberman et al., 2012): Collection of 1449 indoor images captured with a Microsoft Kinect. It contains annotations per pixel of 40 indoor object classes.

- RGB-D Object Dataset (Lai et al., 2011): Dataset of video sequences of 300 household objects arranged using 50 WordNet categories. This dataset was also recorded with a Microsoft Kinect having 640x480 RGB-D at 30Hz.

- SUN3D (Xiao et al., 2013): Dataset of indoor objects with more than 400 sequences captured over 250 spaces. It contains a lot of diversity because it was captured multiples times at different moments of the day.

- SUNRGBD (Song et al., 2015) – Dataset captured with multiple RGB-D sensors, containing more than 10 000 images. It merges data from multiple other datasets such as NYU depth v2 (Silberman et al., 2012), Berkeley 3-D Object (B3DO) (Janoch et al., 2011) and SUN3D (Xiao et al., 2013). It is highly annotated with polygons, bounding boxes, and layout info and categories, being excellent for scene understanding tasks.

2.3.1.3 Video Datasets

Three-dimensional databases are more unusual. They are either Point-clouds or videos, which are costly to store and difficult to segment and annotate.

- A Benchmark for 3D Mesh Segmentation (Chen et al., 2009): Dataset that was used for algorithms benchmark and is composed of 380 meshes of 19 categories. All meshes were manually segmented into functional parts.

- Large-Scale Point Cloud Classification Benchmark (Hackel et al., 2016): Manually annotated 30 3D point clouds of multiple natural and urban environments.

- Objectnet3D (Xiang et al., 2016) : Database for 3D object recognition with 100 categories, 90 127 images, 201 888 objects in these images and 44 147 3D shapes. The objects in the 2D images are aligned with the 3D shapes, and the alignment provides both 3D pose annotation and the closest 3D shape annotation for each 2D object.

- ScanNet (Dai et al., 2017): Video dataset that contains 2.5M views in 1513 scenes annotated with 3D camera poses, surface reconstructions, and semantic segmentation. The data was collected using a scalable RGB-D capture system that includes automated surface reconstruction and crowdsourced semantic annotation.

- Sydney Urban Objects Dataset (Quadros et al., 2012): Velodyne Point Cloud Dataset, which has a 360-degree annotations scan of all objects. The recordings were done on an urban road with multiple objects.

The open datasets in nowadays communities are great and highly improve the creation of new and better AI algorithms. Nevertheless, models are data-hungry, and regardless of the amount of data from a dataset, when applying the model to the real world, the data used is almost never enough. For this reason, there are multiple trends that try to amplify the amount of that available. One popular trend is the Transfer Learning (TL), which consists of training a model in one dataset, saving the model with all the information

learned on the training stage, and then re-train it on the final dataset. This allows the model to have some knowledge of other sources, reducing its tendency of overfitting the dataset. Another popular, yet more complicated approach, is Data Augmentation, which consists of tweaking the dataset to generate new trainable inputs to the model.

2.3.2 CNN Models

The main approach for spatial perception problems nowadays is through the use of CNN. Most of the architectures are inspired by models proposed in 1980 by Fukushima (Fukushima, 1980; Fukushima and Miyake, 1982) and then improved by LeCun (LeCun et al., 1989; LeCun et al., 2008). Multiple improvements were undertaken, which can be categorized as parameter optimization, regularization, and structural reformulation. However, it is observed that the major innovations that boosted performance came from restructuring the processing units and designing blocks. Most of the innovations in CNNs architectures have been made in relation to depth and spatial exploitation. Depending upon the type of architectural modification, CNN can be broadly categorized into seven different classes, namely: spatial exploitation, depth, multi-path, width, feature map exploitation, channel boosting, and attention-based CNNs (Khan et al., 2020).

CNNs tend to have many parameters and hyperparameters like weights, biases, number of neurons, number of layers, filter size, stride, learning rate, activation function, or drop out rate (Shin et al., 2016). For this reason, researchers exploit spatial filters to improve performance. Different kernel sizes were explored to evaluate their impact on network learning. Multiple kernel sizes encapsulate different levels of granularity, but frequently, small size filters extract fine-grained details, and large size extracts coarse-grained information. In this way, by adjusting the filter size, CNNs can learn simultaneously coarse and fine-grained details.

2.3.2.1 LeNet

LeCun proposed LeNet in 1995 (LeCun et al., 1995). By the time, it was a complete novelty, being considered the first CNN which displayed SoA performance on hand digit recognition tasks. The NN classifies handwritten digits without losing accuracy of rotations, small distortions, and variation of position and scale. It is a simple architecture of five convolutional layers with polling between each layer. These five layers are followed by another two layers of fully connected neurons.

In 1995, when this paper was proposed, there were no Graphics Processing Units (GPUs) implementations for CNNs, and the training on Central Processing Units (CPUs) was quite slow (Potluri et al., 2011). The main drawback of the oldest implementation that uses multilayer fully connected NN was that it considered each pixel as separate input and applied a transformation on it, which was a huge computational burden (Gardner and Dorling, 1998). In that regard, LeNet was a great evolution because it exploited the underlying basis of image, that the neighboring pixels are correlated to each other

and are distributed across the entire image. For this reason, convolution parameters are an effective way to extract similar features at multiple locations with fewer parameters. This updated the conventional view of training where each pixel was considered as a separate input feature and ignored the correlation among them. LeNet was the first CNN architecture, which not only reduced the number of parameters and computation but was able to automatically learn feature maps.

2.3.2.2 AlexNet

LeNet was limited to hand digit recognition tasks and didn't scale well to other classes of images. Noting these problems, AlexNet (Krizhevsky et al., 2017) was proposed and demonstrated groundbreaking results for image classification and recognition tasks. Krizhevesky et al. managed to enhance the learning capacity of a CNN by making it deeper and applying multiple new parameter optimization strategies (Krizhevsky et al., 2017). The network architectural design is represented in figure 2.4. By the year 2000, the hardware limitations complicated the learning process in DNNs. AlexNet was trained in two NVIDIA GTX 580 GPUs in parallel to minimize the difficulties with hardware.

The feature extraction stages were extended to seven layers making CNNs applicable for other categories of images. Even though depth improves generalization of different image resolutions, the main downside with this increase of depth is the overfitting. To solve this challenge, in a later version, Krizhevesky et al. used the idea of Hinton (Dahl et al., 2013; Srivastava et al., 2014), where the algorithm randomly skips some neural units during training, which enforces the model to learn different paths, learning more combinations of features and thus making it more robust. Additionally, Rectified Linear Unit (ReLU) was exploited as an activation function that improved the convergence rate by minimizing the problem of vanishing gradients.

In addition to this, ReLU was employed as a non-saturating activation function to improve the convergence rate by alleviating the problem of to some extent (Hochreiter, 1998; Nair and Hinton, 2010). Local response normalization and overlapping subsampling were also utilized to improve the generalization and to reduce overfitting. More adjustments were the use of different size filters (11x11 and 5x5) at the initial layers, compared to previously proposed networks. Due to the efficient learning approach of AlexNet, it was important in the CNNs SoA as it started a new era of research in the models' architectures.

2.3.2.3 ZefNet

The learning mechanism before ZefNet was based on hit-and-trial, without knowing the exact reason behind the updates. The lack of knowledge limited the performance of DNN on complex images. Zeiler and Fergus presented a multilayer Deconvolutional NN, named DeconvNet, which was also mentioned as ZefNet (Zeiler and Fergus, 2014). ZefNet was developed to visualize the network performance. The core idea was to visualize the

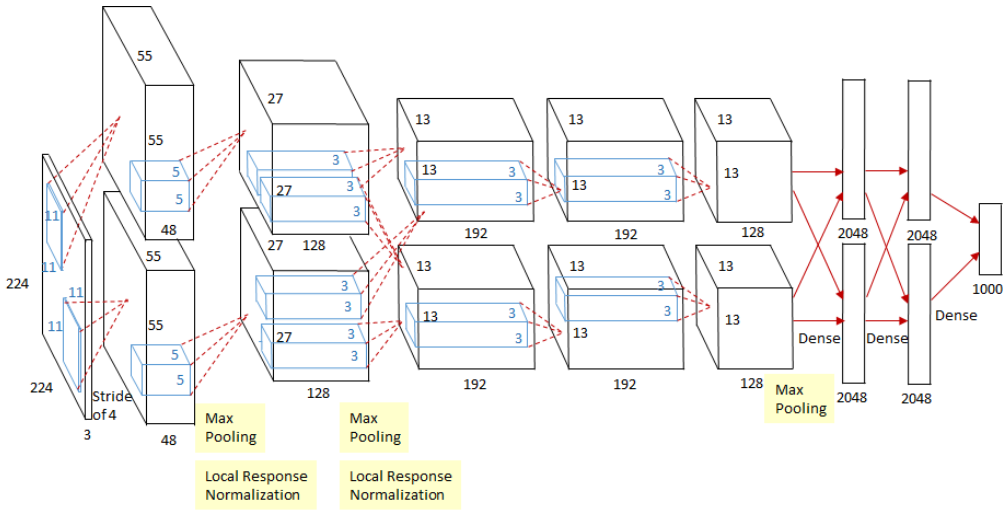


Figure 2.4: Layout of AlexNet Model (Fukushima and Miyake, 1982).

neuron's activation and try to understand its outputs. One of the previous studies of Erhan et al. also had the same idea and boosted the performance of Deep Belief Networks (DBNs) by analyzing the hidden layers' features (Erhan et al., 2009). In the same line of thought, Le et al. (2011) evaluated the performance of deep unsupervised Auto Encoder (AE) by visualizing the image classes generated by the output neurons (Le, 2013).

ZefNet execution is similar to the forward pass CNN but swaps the order of convolutional and pooling operation. This reverse mapping allows the visualization of image patterns from the projected outputs of the convolutional layers. This enables neuron-level interpretation and visualization of the hidden layers feature maps (Grun et al., 2016; Simonyan et al., 2014).

This network allows the monitoring of the learning scheme during training and thus facilitates in diagnosing a potential problem associated with the NN. The idea was experimentally validated on AlexNet using ZefNet, which demonstrated that only a few neurons were active while other neurons were inactive in the first and second layers of the network. Also, it showed that the features extracted by the second layer exhibited aliasing artifacts. Based on these findings, major adjustments to the CNN topology were performed, and some parameter optimization was done. Moreover, both filter size and stride were reduced to retain more features in the two initial convolutional layers. These tweaks in the CNN led to major performance improvements, which proved that feature maps visualization could be used for the identification of design redefinitions and for precise adjustment of parameters during training.

2.3.2.4 VGG

Simonyan and Zisserman proposed a simple yet effective design principle for CNN architectures. The architecture was named as Visual Geometry Group (VGG), which is a CNN with modular layers pattern (Simonyan and Zisserman, 2015). VGG is made of nineteen

layers deep, which is a clear depth increased when compared to AlexNet and ZefNet, allowing it to have a better representational capacity (Krizhevsky et al., 2017; Zeiler and Fergus, 2014), as it can be observed in figure 2.5. ZefNet, proved that small-size filters could improve the performance of CNNs. Leveraging these findings, VGG removed the $11x11$ and $5x5$ filters with a stack of $3x3$ filters layer and experimentally demonstrated that concurrent placement of $3x3$ filters could induce the effect of larger size filter. The $3x3$ filters stacks provide the additional benefit of lower computational work because it reduces the number of operations and parameters. This NN simplifies complexity by utilizing $1x1$ convolutional filters in between convolutional layers, which also adds linear combinations of the sets of the features maps. Max polling was also utilized after the $1x1$ filters, and padding was performed to retain the spatial resolution (Ranzato et al., 2007). VGG did not win the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) contest but got popularity due to its simplicity, homogeneous architecture, high depth, and good results in both image classification and localization problems. The major drawbacks of VGG are the computational cost. It uses small $3x3$ filters, but it has so many that it reaches over 140 million parameters, being the biggest CNN this dissertation analyzed, being detailed in section 2.3.2.21.

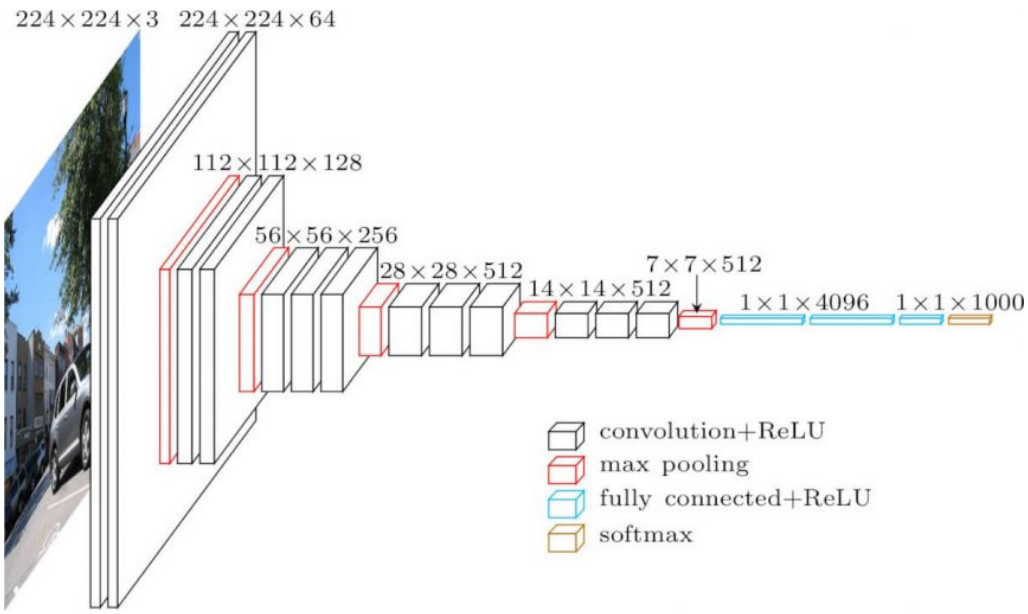


Figure 2.5: Layout of VGG Model (Zhang et al., 2016).

2.3.2.5 GoogLeNet

GoogLeNet was the winner of the ILSVRC competition in 2014 and is also known as Inception-V1. It was built to achieve the highest accuracy with low computational cost (Szegedy et al., 2015). A new concept of the inception block was introduced, which presented multi-scale convolutional transformations using split, transform, and merge concepts. The inception block is represented in figure 2.6. It encapsulates a set of $1x1$, $3x3$,

$5x5$ filters that allows to capture different spatial information (at fine and coarse levels). On GoogLeNet architecture, conventional convolutional layers are replaced by small in a similar fashion to the substitutions of micro NN proposed in Network in Network (NIN) architectures (Lin et al., 2014a). The parallelism concept of splitting, transform, and then merge addressed problems related to the learning of diverse types of variations present in the same class of different images. Furthermore, GoogLeNet focused on parameterization tunning efficiency. It regulates the computation costs by placing a downsample layer with a $1x1$ convolutional filter before any large size kernel filter. The sparse connections overcome the problem of redundant information and also reduce the cost by reducing the feature maps that were not relevant. Additionally, the connection's density was reduced by using a global average pooling at the last layer instead of the usual fully connected layer. These modifications decreased by eight times the number of parameters, resulting in a network of 5 million parameters.

Other regulatory factors applied were batch normalization and the use of Root Mean Square Propagation as an optimizer (Dauphin et al., 2015). GoogLeNet also introduced the concept of middle network learners, which speeds up the convergence rate. Nevertheless, the main disadvantage of GoogLeNet was its heterogeneous topology that requires to be customized from module to module. Another limitation of GoogLeNet is the representation of the downsample layer drastically reduces the feature space passed to the next layer and thus sometimes leads to loss of useful features.

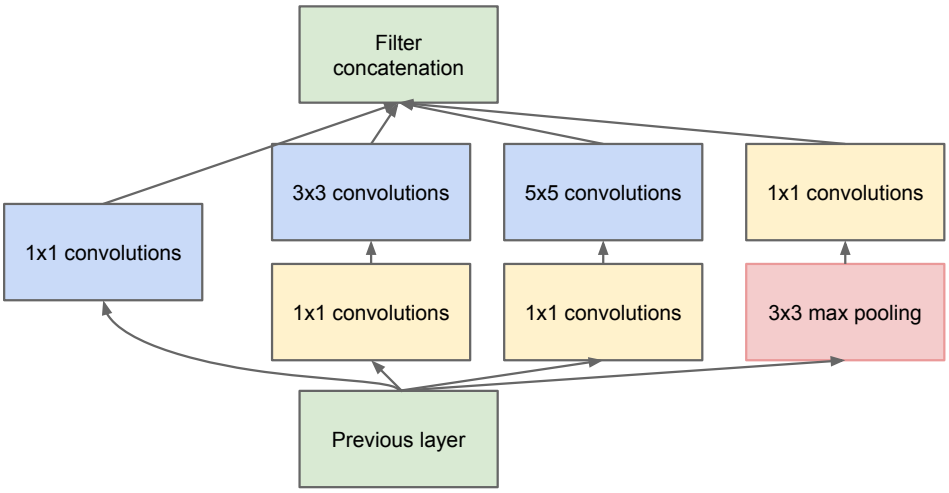


Figure 2.6: Layout of the inception block (Szegedy et al., 2015).

Subsequently, depth-based CNNs started being a trend. These networks are based on the assumption that with the increase in depth, the network approximates the target function with several nonlinear mappings and improved feature representations (Bengio et al., 2013). Furthermore, studies have shown that deep networks can represent certain classes of function more efficiently than shallow architectures (Montúfar et al., 2014).

A single hidden layer is sufficient to approximate any function, but this comes at the

cost of exponentially many neurons. Thus, often making it computationally unpractical (Csáji, 2001). For that reason, Bengio and Delalleau advocated that deeper networks have the same capabilities at a reduced cost (Delalleau and Bengio, 2011; Wang and Raj, 2017). They empirically showed that DNN are computationally more efficient for complex tasks (Bengio, 2013; Nguyen et al., 2018). Both Inception and VGG showed the best performance in ILSVRC 2014 competition, further strengthen the idea that the depth is an intrinsically correlated dimension of the learning capacity in CNNs (Simonyan and Zisserman, 2015; Szegedy et al., 2015; Szegedy et al., 2016).

2.3.2.6 Highway Networks

Following the line of intuition that the learning capacity can be improved by going deeper, Srivastava et al. proposed the Highway Network (HN) (Srivastava et al., 2015a). The main problem concerned with DNNs is the slow training and convergence speed (Huang et al., 2016). The HNs exploited depth for learning features representation by adding a novel inter-layer connectivity. HN with 50-layers converge faster than thin but deeper architectures on ImageNet dataset (Morar et al., 2012; Russakovsky et al., 2015). The researchers showed that the performance of a plain CNN decreases when ten or more hidden layers units are used (Glorot and Bengio, 2010). On the other hand, HN manage to converge faster than the plain ones, even with depths of 900 layers.

On HNs, as proved in equation 2.2, the uninterrupted flow of information across layers is enabled by imparting two gating units within a layer. The idea of a gating mechanism comes from the Long Short-Term Memory (LSTM) used in Recurrent Neural Networks (RNN) (Mikolov et al., 2010; Sundermeyer et al., 2012). The aggregation of information by joining the lth layer and previous $l-k$ layers information creates a regularizing effect, making gradient descend of deep networks flow. This allows the training of networks with depths up to 900 layers with Stochastic Gradient Descent (SGD) algorithm. Cross-layer connectivity for HN is defined in equation 2.2 and 2.3, where T_g refers to transformation gate (expresses the amount of the produced output), and C_g is a carry gate. The working hidden layers and the residual implementation are represented by $H_l(x_i, W_{H_l})$. Finally, $1 - T_g(x_i, W_{C_g})$ behaves as a switch in a layer, which decides the path for the flow of information.

$$y = H_l(x_i, W_{H_l})T_g(x_i, W_{T_g}) + x_i C_g(x_i, W_{C_g}) \tag{2.2}$$

$$C_g(x_i, W_{C_g}) = 1 - T_g(x_i, W_{C_g}) \tag{2.3}$$

2.3.2.7 ResNet

He et al. proposed ResNet was a continuation of the deep models (He et al., 2016). It was revolutionary by presenting the concept of residual learning for CNNs and devised an efficient methodology for the training of models. In a similar fashion of HN, it proposes a

152-layers deep CNN winner of the 2015 ILSVRC competition. The layouts of the residual block are presented in figure 2.7. It is 20 times deeper than AlexNet and eight times deeper than VGG and manages to have less computational complexity than previously proposed (Krizhevsky et al., 2017; Simonyan and Zisserman, 2015). ResNet has better accuracy on image classification tasks than 34 layers plain with any of the proposed setups (50, 101, or 152 layers). Not only that, but ResNet obtained 28% better results on the COCO benchmark dataset (Lin et al., 2014b). This result enforces the idea that for image recognition and localization tasks, depth is key.

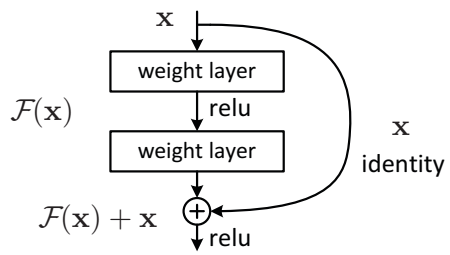


Figure 2.7: Layout of the residual block (He et al., 2016).

2.3.2.8 Inception-V3, Inception-V4 and Inception-ResNet

The models Inception-V3, Inception-V4 and Inception-ResNet, are upgraded versions of Inception-V1 and Inception-V2 (Szegedy et al., 2017; Szegedy et al., 2015; Szegedy et al., 2016).

Inception-V3 reduces the computational cost of CNNs without affecting the generalization. To do this, the $5x5$ and $7x7$ size filters were replaced with small and asymmetric filters of $1x7$ and $1x5$ and used 1x1 convolution as downsampling prior to the large filters (Szegedy et al., 2016). This transforms the convolution operation into something similar to cross-channel correlation. Lin et al. exploited the potential of 1x1 filters in NIN architecture (Lin et al., 2014a). The same concept was reutilized in Inception (Szegedy et al., 2016). On the V3 version, $1x1$ convolutional operation was used to map the input data into 3 or 4 smaller spaces than the original input space and then maps all correlations in these smaller 3D spaces via $3x3$ and $5x5$ convolutions. On the ResNet version, the residual learning and inception block were used (He et al., 2016; Szegedy et al., 2017). This way, filter concatenation is done by the residual connection. Moreover, it was experimentally demonstrated in the V4 version that with residual connections, it is possible to achieve the same generalization power as plain Inception-V3 but with increased depth and width. However, it was observed that Inception-ResNet converges faster than Inception-V4, which shows that training with residual connections accelerates training.

The idea of bypassing pathways used in this version of Inception is similar to HNs. It can be expressed by the equation:

$$g(x_i) = f(x_i) + x_i \tag{2.4}$$

On equation 2.4, $f(x_i)$ is the transformed signal and x_i is an original input. Original input x is added to $f(x)$ through figure 2.7 bypass pathway. The key idea is that $g(x_i) - x_i$ performs residual learning. ResNet introduced a new paradigm of connections within layers to enable cross-layer connectivity, but these gates are data-independent and parameter-free in comparison to HNs. In HNs, when a gated shortcut is closed, the layers represent non-residual functions. On the contrary, in ResNet, residual information always flows and identity shortcuts are never closed.

2.3.2.9 ResNext

As an improvement of ResNet, the Aggregated Residual Transform Network, also known as ResNext, was proposed (Xie et al., 2017). The concept of the split, transform and merge was exploited with the addition of cardinality (Szegedy et al., 2015). It can be seen as an added dimension, which refers to the set of transformations size (Han et al., 2018b; Sharma and Muttoo, 2019). The group of Inception NN not only improved the learning capability of CNNs but also proved that CNN can be resource effective. On the negative side, due to the use of diverse spatial embedding is in the transformation branch, each layer needs to be customized separately. ResNext is sometimes described as an aggregation of different types of NN, because it has characteristic features from Inception, VGG, and ResNet (He et al., 2016; Simonyan and Zisserman, 2015; Szegedy et al., 2015). It uses the deep homogeneous topology of the VGG, the simplified GoogLeNet architecture by fixing spatial resolution to $3x3$ filters within the transformations block, and it also uses residual learning. Building block for ResNext is depicted in figure 2.8. Xie et al. demonstrated that by increasing cardinality, the performance could be improved. ResNext complexity is regulated by applying $1x1$ filter slow embedding is before the $3x3$ convolutions. Also, training was optimized by using skip connections (Han et al., 2018b).

Vanishing gradient results in both test error increase and also in higher training error (Dauphin et al., 2017; Dong et al., 2016; Hochreiter, 1998; Pascanu et al., 2012). To address it, the concept of multipath and cross-layer connectivity were proposed (Huang et al., 2017; Kuen et al., 2017; Larsson et al., 2019; Srivastava et al., 2015a). The multiple paths or shortcut connections systematically connect layers by skipping some intermediate layers, allowing the information flow across the layers (Mao et al., 2016; Tong et al., 2017). These paths also partially solve the vanishing gradient problem by making the gradient accessible to initial layers.

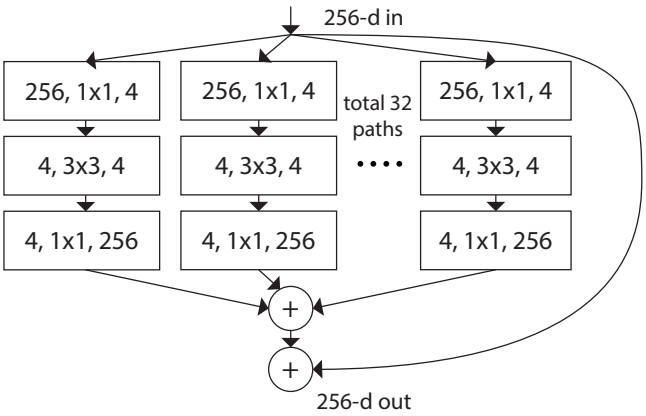


Figure 2.8: Layout of the ResNext residual block (Xie et al., 2017).

2.3.2.10 DenseNet

After HNs and ResNet, DenseNet was proposed, in a tentative to solve the vanishing gradient problem (He et al., 2016; Srivastava et al., 2015a). ResNet show SoA results, but it preserved all neurons, having some of them contributing with almost no information. DenseNet uses a new approach of cross-layer connectivity to handle this issue. It has each layer interconnected in, similar to a feed-forward NN, meaning that the feature maps of all previous layers are connected to subsequent layers. These are $\frac{l^2+l}{2}$ direct connections, compared to the l connections between a layer and its preceding layer in the previously presented models.

It reflects the notion of cross-layer depth-wise convolutions. These types of connections provide the DNN the ability to explicitly differentiate between information that is added to the network and information that is preserved. DenseNet becomes parametrically expensive when the number of feature maps is increased. The direct permeability of each layer to the gradients through the loss improves the flow throughout the DNN. This acts as a regularizing effect, which minimizes overfitting.

During the time were HN, Inceptions, and DenseNets were proposed, the focus was mainly on exploiting the depth and minimizing depth issues with strategies such as the multi-pass regulatory connections in the network regularization (He et al., 2016; Srivastava et al., 2015a). Kawaguchi et al. approached the problem from a different perspective, studying the width of NN (Kawaguchi et al., 2019). Multilayer neurons have the capability of handling complex functions by utilizing parallel processing units within a layer. This puts forward for consideration that width is an essential parameter in defining principles of learning along with depth. Moreover, researchers (Hanin and Sellke, 2017) have shown that DNN that use ReLU have to be wide enough in order to hold universal approximation property with the increase in depth. Furthermore, some classes of functions from a small dataset cannot be well approximated if the maximum width of the NN is smaller than the input dimension (Lu et al., 2017; Nguyen et al.,

2018). Nevertheless, increasing depth enables a variety of feature representations, which may disguise false learning. To address this problem, a group of research focuses their attention on wider but shallower models.

2.3.2.11 WideResNet

The deep residual networks have the drawback of the feature reuse where feature transformations and blocks contribute with an insignificant amount to learning (Srivastava et al., 2015b). WideResNet proposes a solution to this issue (Zagoruyko and Komodakis, 2016). The learning of deep residual networks is due to the residual units, whereas depth has a variety effect. WideResNet exploited residual blocks by making ResNet wide rather than deep (He et al., 2016). A new factor k is added, to regulate the width of the network. The results show that utilizing a wider NN, the performance is improved over the same deeper residual NN.

The core idea of wider residual network is based on the observation that most of architectures before residual networks, including the most successful Inception and VGG, were wider in comparison to ResNet. DNNs that use residual blocks have a better representational capacity, but at the cost of intensive training, inactivation of many feature maps and suffering from vanishing gradients and saturation issues. This problems were partially minimized by the use of dropout in residual blocks (He et al., 2016). In a analog way, Huang et al. utilized the concept of stochastic depth by using dropouts to solve vanishing gradients and slow training (Huang et al., 2016). A test based study demonstrated that WideResNet has twice the number of parameters when compared to ResNet, but can be trained faster than deeper networks (Zagoruyko and Komodakis, 2016).

2.3.2.12 Pyramidal Net

In the previously presented CNNs architectures such as AlexNet, VGG and Inception, deep stacking of multiple convolutional layers increases the depth of feature maps on subsequent layers. However, the spatial dimension shrinks because each convolutional layer is followed by a downsampling layer (He et al., 2016; Krizhevsky et al., 2017; Simonyan and Zisserman, 2015). As a consequence, Han et al. proposed that in DNNs, feature representation is compensated by a decrease in feature map size (Han et al., 2017). As a result, ResNet has demonstrated remarkable results for image classification. However, the deletion of a residual block, where feature map depth increases while spatial dimension decreases, deteriorates performance. As a result, the proposed stochastic ResNet improves performance by utilizing less information loss from the removal of the residual unit (Huang et al., 2016). Using this concept, Han et al. proposed Pyramidal Net (Han et al., 2017).

The main difference from ResNet is that Pyramidal Net gradually increases the width per residual unit, whereas ResNet is the complete opposite. By doing this, Pyramidal Net obtain features from all possible locations instead of maintaining the same spatial

dimension within each residual block until down-sampling. The gradual increase in the depth of features map in a top-down approach gives this model the 'Pyramidal' name. Depth of features maps is regulated by factor l, and is computed using the equation:

$$D_l = \begin{cases} 16 & , l = 1 \\ D_{l-1} + \frac{\lambda}{n} & , 2 \leq l \leqslant n+1 \end{cases} \tag{2.5}$$

In equation 2.5, D_l denotes the dimension of lth residual unit, n represents the total number of the residual units, λ is a step factor and $\frac{\lambda}{n}$ regulates the increase in depth. The depth regulating factor manages the trade-off for the increase of feature maps. Also, the residual connections were inserted in between the layers by using zero-padded identity mapping.

Zero-padded identity mapping needs fewer parameters as compared to the projection-based shortcut connection, so it produces a better generalization (Wang et al., 2019). Pyramidal Net has two different approaches for the widening of the network, including addition and multiplication-based widening. The first approach increases linearly, whereas the multiplicative one increases geometrically (Ioffe and Szegedy, 2015; Xu et al., 2015). Withal, the major problem of Pyramidal Net is that by increasing width, space, and processing time scale exponentially.

2.3.2.13 Xception

Xception is considered an extreme Inception architecture, which exploits the idea of depthwise separable $3x3$ convolutions introduced by AlexNet (Chollet, 2017; Krizhevsky et al., 2017). It modifies the original inception block by making it wider and removing the different spatial dimensions, utilizing $3x3$ filters followed by $1x1$ that regulate computational complexity. The Xception block is represented in figure 2.9. Xception makes the network computationally efficient by decoupling spatial and feature maps correlation. The convolved output is mapped to embeddings by $1x1$ convolutions, which are then spatially transformed k^{th} times. The cardinality width represented by k determines the number of transformations. It softens the computation of Inception by separately convolving each feature map across spatial axes, which is followed by $1x1$ convolutions, which performs cross-channel correlation.

In Xception, the $1x1$ convolution was used to regulate feature map depth. Compared to the previously presented models, where either the convolutional operation uses only one transformation segment or in the inception block where three transformation segments are used, the Xception number of transformation segment is equal to the number of feature maps. Nevertheless, this does not reduce the number of parameters but augments the learning performance per parameter, which improves performance.

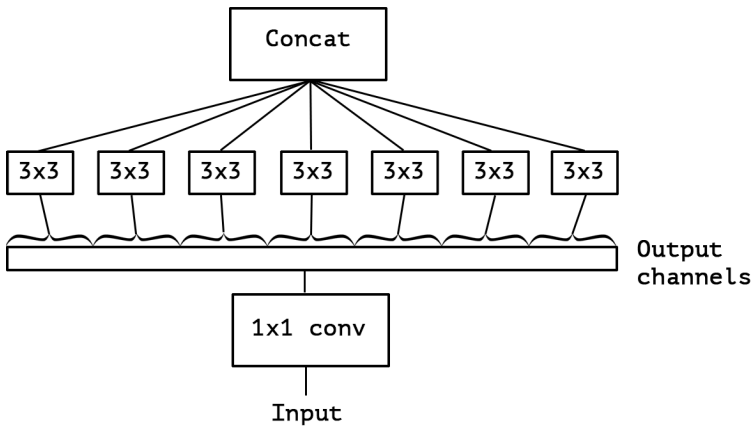


Figure 2.9: Layout of the Xception residual block (Chollet, 2017).

2.3.2.14 Squeeze and Excitation Network

Hu et al. proposed Squeeze and Excitation (SE) Network (Hu et al., 2018a), which reported a new record on the ImageNet dataset. It presents a new block for the selection of feature maps used on object discrimination. This new block, named SE block, gives weights to the most relevant feature maps and suppresses the unusable ones. The proposed block is designed generically, and therefore can be added in any CNN architecture before the convolution layer. It consists of two operations:

1. Squeeze.
2. Excitation.

Convolution kernels capture information locally, but they ignore the contextual relation of features outside of this receptive field. To obtain global information of feature maps, the squeeze block generates feature map-wise statistics by suppressing spatial information of the convolved input. Thus, the global average pooling has the potential to learn the extent of the target object effectively. For this reason, it is employed by the squeeze operation to generate feature map wise statistics using (Lin et al., 2014a; Zhou et al., 2016):

$$D_m = \frac{1}{m * n} \sum_{i=1}^{m} \sum_{j=1}^{n} x_c(i, j) \tag{2.6}$$

On equation 2.6, D_m is a feature map descriptor and $m * n$ is the spatial dimension of input. The output D_m is passed to the excitation operation, which models interdependencies by exploiting the gating mechanism. This excitation operation assigns weights to feature maps using two-layer feed-forward NN, which is mathematically expressed:

$$V_M = \sigma(w_2 \delta(w_1 D_m)) \tag{2.7}$$

On equation 2.7, V_M denotes the weight of each feature map, where δ and σ refer to the ReLU and sigmoid functions. In excitation operation, $w1$ and $w2$ are used as a limit factor to the model complexity, aiding generalization (LeCun et al., 2012; Xu et al., 2015). After the squeeze operation, ReLU activation function is utilized to add non-linearity in feature maps. The gating mechanism is then exploited using the sigmoid activation function, which models interdependencies among feature maps and assigns a weight based on feature map relevance (Zheng et al., 2017).

2.3.2.15 Competitive Squeeze and Excitation Networks

The Competitive Inner-Imaging Squeeze and Excitation (CMPE-SE) Network was proposed by Hu et al. in 2018 (Hu et al., 2018b). It uses the idea of SE block to improve the learning of deep residual networks (Hu et al., 2018a). The SE Network recalibrate feature maps upon their contribution to class discrimination. However, the main pitfall of SE NN is that in residual networks, it only considers the residual information for the weight of each channel (Hu et al., 2018a). Therefore it does not use the complete abilities of SE block and makes the residual information redundant. Researchers addressed this problem by generating feature map-wise statistics from both residual and identity mapping-based features. By doing this, the global representation of feature maps is generated using a global average pooling operation. In contrast, the relevance of feature maps is estimated by making competition between residual and identity mapping-based descriptors. This mechanism is named inner imaging (Hu et al., 2018b). The CMPE-SE block not only models the correlation between residual feature maps but also maps their correlation with identity feature maps and makes a competition between residual and identity feature maps. The CMPE-SE block can be expressed by:

$$y = F_{se}(u_r, x_{id})F_{res}(x_{id}, w_r) + x_{id} \tag{2.8}$$

On equation 2.8, F_{se} represents the squeeze operation applied on residual feature map u_r, x_{id} is the identity mapping of input and F_{res} expresses the SE block on residual feature maps. The output of the squeeze operation is multiplied with the SE block output F_{res}. The backpropagation algorithm optimizes the competition between identity and residual feature maps and the correlation between all feature maps in the residual block.

2.3.2.16 Channel Boosted CNN using Transfer Learning

The Channel Boosted Convolutional Neural Network (CB-CNN) was proposed by Khan et al. and is build on the idea of boosting the input channels for improving the representational capacity of the network (Khan et al., 2018). In figure 2.10 is represented the block diagram of CB-CNN. The Channel Boosting is performed by generating auxiliary channels with the aid of a deep generative model, exploiting it through the deep discriminative models. Furthermore, it utilizes the concept of TL at the generation and discrimination stages (Hamel and Eck, 2010; Vincent et al., 2008).

As it was demonstrated in section A.4, data representation plays a vital role in the generalization of a classifier, as different representations may present different modalities of information (Bengio et al., 2013). The generative learners characterize data creation distribution during the training phase. In CB-CNN, autoencoders are used as generative learners to handle explanatory factors of variation behind the data. Inductive TL is applied in a novel way to build a boosted input representation by augmenting the learned distribution of the input data with the original channel space. The CB-CNN encapsulates the channel-boosting phase into a generic block, which is inserted at the beginning of a DNN. During training, a pre-trained CNN is used to reduce computational cost. The main contribution of this architecture is that DNN learners are used instead of generative learning models, enhancing the representational capacity of deep CNN based discriminator.

Khan et al. only demonstrated the potential of the channel boosting by utilizing the boosting block at the beginning of the NN. However, they suggest that it can be extended to any layer in the architecture.

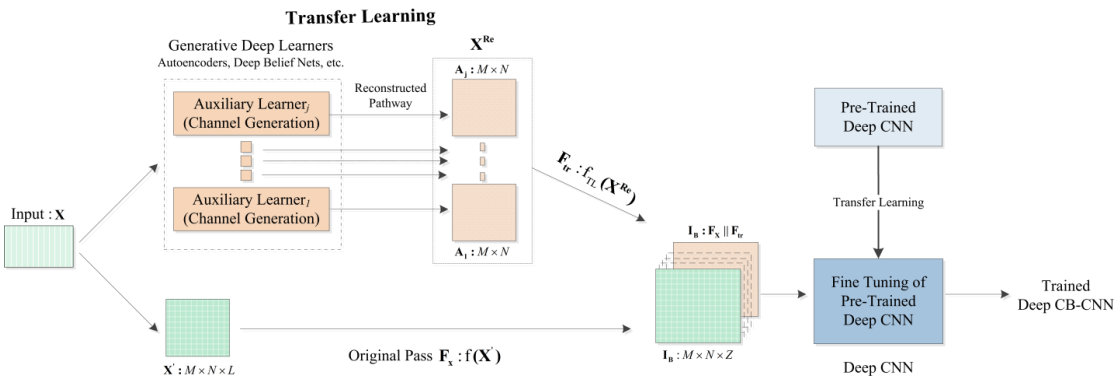


Figure 2.10: Layout of the CB-CNN residual block (Khan et al., 2018).

The different levels of abstraction play an important role in defining discrimination. Additionally, to learn different levels of abstraction, focusing on features relevant to the context also plays a significant role in image localization and recognition. In the human visual system, this phenomenon is referred to as attention. Humans view the world in a quick sequence of glimpses, always paying attention to a part of it. The attention mechanism focuses on a selective region and deduces information more accurately for that particular zone than the rest. A similar approach is done in RNN with LSTM (Mikolov et al., 2010; Sundermeyer et al., 2012). This NN exploits attention to generate sequences of data, and the new samples are weighted based on the previous iterations. This concept of attention was incorporated into CNNs by various researchers to improve representation and reduce computation.

2.3.2.17 Residual Attention Neural Network

The Residual Attention Network (RAN) was proposed to improve feature representation (Wang et al., 2017). It utilizes attention to learn object aware features. Its genesis is a feed-forward CNN, built with a stack of residual blocks with an attention module. The novel attention module is a branch using a bottom-up, top-down learning strategy. Combining these two different learning approaches in the attention module allowed fast feed-forward processing and top-down attention feedback in a single pass.

Bottom-up feed-forward produces low-resolution feature maps with semantic information. Top-down passage produces features in order to make an inference at a pixel level. Previously, the top-down, bottom-up learning strategy was used by Restricted Boltzmann Machines (Salakhutdinov and Larochelle, 2010). The top-down attention mechanism as a regularizing factor in Deep Boltzmann Machine (DBM) during the reconstruction phase in training. Top-down learning strategy globally optimizes network in such a way that gradually output the maps to input during the learning process (Hinton et al., 2006; Salakhutdinov and Larochelle, 2010). The attention module can be despited by:

$$A_{i,FM}(x_c) = S_{i,FM}(x_c) * T_{i,FM}(x_c) \tag{2.9}$$

On equation 2.9, the attention module generates object aware soft mask $S_{i,FM}(x_c)$ per layer (Goh et al., 2013). Soft mask, $S_{i,FM}(x_c)$ focus attention towards object using equation 2.9 by recalibrating trunk branch $T_{i,FM}(x_c)$ output and thus, behaves like a control gate.

Researchers also presented Transformation network (Jaderberg et al., 2015) which also exploits the idea of attention by incorporating it with convolution block. However, the attention module in the Transformation network is fixed and cannot adapt to changing circumstances. Furthermore, RAN was made efficient towards the recognition of cluttered, complex, and noisy images by stacking multiple attention modules. The hierarchical organization of RAN enables the ability to adaptively weight each feature map based on their relevance (Wang et al., 2017).

2.3.2.18 Convolutional Block Attention Module

The performance of attention mechanism and feature map exploitation is validated through RAN and SE Network respectively (Hu et al., 2018a; Wang et al., 2017). Utilizing the core concepts of both, Woo et al. proposed Convolutional Block Attention Module (CBAM) (Woo et al., 2018). It is similar to SE Network design, but SE Network only considers the contribution of feature maps for classification, ignoring the locality of the object in images. The spatial location of the object is important for accurate object detection. For this reason, CBAM infer attention maps sequentially by first applying feature map attention and then spatial attention to find the refined feature maps. Usually, $1x1$ convolutions precede pooling operations for spatial attention, Woo et al. demonstrated that the pooling of features along the spatial axis generates an efficient feature descriptor.

It concatenates average pooling operation with max pooling, which generates a strong spatial attention map. Similarly, feature map statistics were modeled using a combination of max pooling and global average pooling operation. Max polling provides hints about distinctive object features, whereas the use of global average pooling returns suboptimal inference of feature map attention. Also, the formulation of 3D attention maps via the serial learning process reduces parameters as well as computational cost. Also, CBAM can be integrated with any CNN architecture.

2.3.2.19 Concurrent Spatial and Channel Excitation Mechanism

In Concurrent Spatial and Channel Excitation Mechanism, the effect of spatial information in combination with feature map information is exploited for segmentation tasks (Hu et al., 2018a; Roy et al., 2018). It introduces three different modules:

1. Squeezing spatially and exciting feature map wise (cSE).
2. Squeezing feature map wise and exciting spatially (sSE).
3. Concurrent spatial and channel squeeze and excitation (scSE).

An autoencoder-based convolutional DNN is used for segmentation, and the proposed modules were inserted after the encoder and decoder layer. In the cSE module, the technique presented by SE block is exploited. In this module, a scaling factor is derived based on the combination of feature maps in object detection. In the sSE module, spatial locality gives more importance than feature map information. For this purpose, different combinations of feature maps are selected and exploited spatially to be used in segmentation. Finally, in the scSE module, attention to each channel is assigned by deriving scaling factors both from spatial and channel information and highlighting the object-specific feature maps (Roy et al., 2018).

2.3.2.20 MobileNetV2

MobileNetV2 builds upon the ideas from MobileNetV1 (Howard et al., 2017; Sandler et al., 2018a), using depthwise separable convolution as efficient building blocks. However, the second version introduces two new features to the architecture. The first is the linear bottlenecks between the layers, and the second is shortcut connections between the bottlenecks. This architecture is specifically tailored for mobile and resource constrained environments.

Its main contribution is the inverted residual with linear bottleneck layer module. It takes a low-dimensional compressed representation as an input, which is first expanded to high dimension and filtered with a lightweight depthwise convolution. This makes features to be subsequently projected back to a low-dimensional representation with a linear convolution. Instead of using a full convolutional operator, it applies a factorized version that splits convolution into two separate layers. One layer is called a depthwise

convolution, and it performs lightweight filtering by applying a single convolutional kernel per channel. The other layer is a 1×1 convolution, which is a pointwise convolution. This is responsible for building new features through computing linear combinations of the input channels.

2.3.2.21 Models Comparison

Looking backward at the CNNs history, the turning point started with the success of AlexNet (Krizhevsky et al., 2017) for ImageNet classification in 2012. Afterwards, significant efforts have been made in developing CNN models that would increase performance without compromising goal convergence, including VGG (Simonyan and Zisserman, 2015), GoogLeNet (Szegedy et al., 2015), and ResNet (He et al., 2016).

AlexNet CNN represented a milestone for computer vision and ML because it focuses on generalization rather than memorization by using methods, such as dropout, which mitigates overfitting. Not only that, but there were a lot of effort in GPU implementations for the convolution operation, which reduced the training duration.

After AlexNet, the VGG (Simonyan and Zisserman, 2015) was proposed and won the localization and classification tasks of the ILSVRC 2014 competition. The VGG has been widely used because of its simplicity. The core idea was to pass the complexity to the network and figure out how to solve the presented task.

Google quickly managed to catch up and launched GoogLeNet (Szegedy et al., 2015), which has two main advantages:

1. The utilization of filter kernels of different sizes at the same layer preserves more spatial information.

2. The reduction of the network's number of parameters makes it less sensitive to overfitting and allows it to be deeper. Moreover, the 22-layer GoogLeNet has more than 50 convolutional layers distributed inside the inception modules, but it has 12 times fewer parameters than AlexNet.

ResNet was then presented as an evolution, being one of the most successful CNNs with Conference on Computer Vision and Pattern Recognition (CVPR) 2016 Best Paper Award (He et al., 2016). The idea behind ResNet is that each layer should not learn the whole feature space transformation but only a residual correction to the previous layer, which allows training much deeper networks efficiently. Its extremely deep representations have excellent generalization performance and led it to win first place in ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation at the 2015 ILSVRC and COCO competitions.

In Figure 2.11 is presented one-crop accuracies of the most relevant entries submitted to the ImageNet challenge, from the AlexNet (Krizhevsky et al., 2017), on the far left, to one of the best performing, the Inception-V4 (Szegedy et al., 2017). The models based

on ResNet and Inception architectures obtained the other architectures by at least 7%, presenting around 25% more accuracy than AlexNet.

In figure 2.12 it is possible to compare the trade-offs between Top-1 accuracy, the number of Giga Operations (billion of Operations Per Second), and the number of parameters. The blobs' size is proportional to the number of network parameters. This plot gives an informative view of the accuracy values, representing the computational cost for performance. It is possible to conclude that VGG are the most expensive architecture in terms of computational requirements and number of parameters. The rest of the architectures form a steep straight line that seems to start to flatten with the latest incarnations of Inception and ResNet. This graph also suggests that models are reaching an inflection point on this ImageNet dataset, where the costs start to outweigh gains in accuracy.

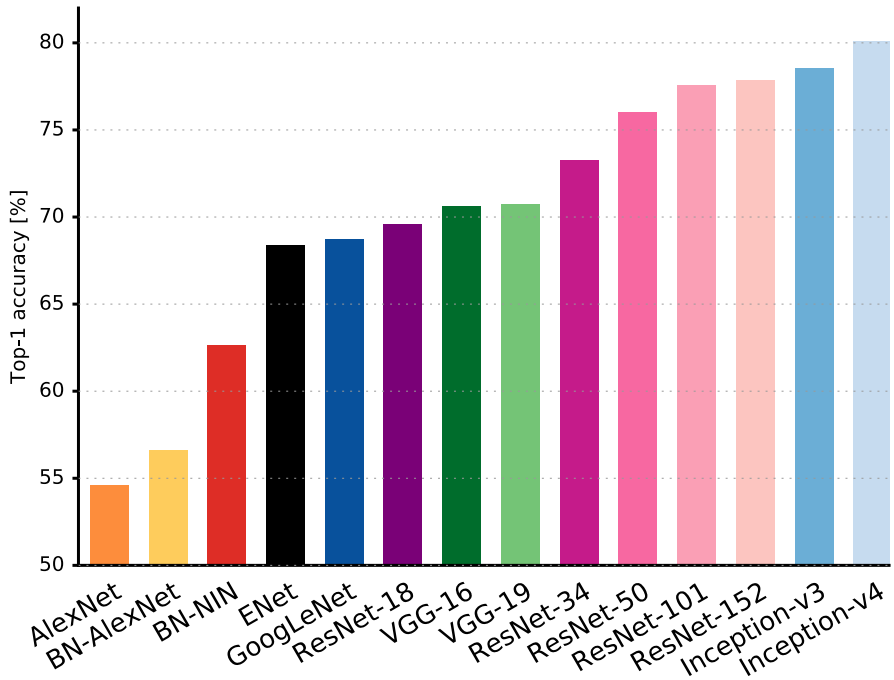


Figure 2.11: Result comparison of top-1 validation accuracies for top scoring single-model architectures (Canziani et al., 2016).

As represented in figure 2.12, VGG is the most expensive network when performance per accuracy is concerned. Because of its straightforward implementation and easy demonstration of results, this dissertation will utilize VGG for sample demonstration and then move to complex architectures combinations when hardware constraints are present. However, for practical applications, VGG should not be considered as other architectures achieve the same performance at a lower computational cost.

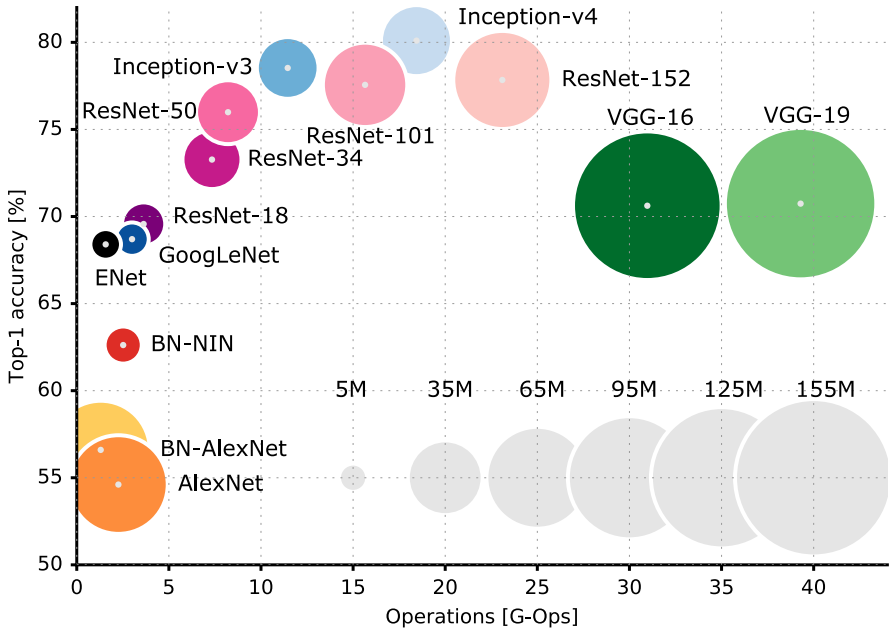


Figure 2.12: Result trade-off analyzes between top-1 validation accuracies, number operations and network size (parameters) (Canziani et al., 2016).

Some comparison summary between these models, based on (Han et al., 2018a) and some tests developed on Keras is presented on table 2.3.

2.3.2.22 Disadvantages of Deep Neural Networks

After all the hype is tested and some scientific analysis is produced with real scenarios on published algorithms, some downsides are revealed. In some cases, this analysis comes from outside the field of computer science (Kamilaris and Prenafeta-Boldú, 2018).

The lack of theory surrounding some methods (Marcus, 2018) is usually the primary concern. Deep Learning (DL) methods are often looked at as a black box, with most confirmations done empirically, rather than theoretically (Knight, 2017).

The most generalized models are still a long way from integrating abstract knowledge, such as details about what objects are, what they are used for, and they integrate with others. On the other hand, commercial AI systems, such as Watson, apply multiple algorithms combined with an ensemble of techniques to solve problems (Marcus, 2015).

In some use cases, DL classify unrecognizable images as belonging to a group of images (Nguyen et al., 2015) and misclassifies perturbations of correctly classified images. Goertzel proposed that these behaviors arise from internal representations and that these

Table 2.3: Comparison between various CNN architectures.

Model Name	Year	Major novalty	Parameters	Top 5 Error Rate	Depth	Reference
LeNet	1995	First Popular CNN architecture.	0.060 M	MNIST: 0.95	7	LeCun et al., 1995
AlexNet	2012	Deeper and wider than the LeNet; Relu; Dropout; Overlap Pooling; GPUs NVIDIA GTX 580.	60 M	ImageNet: 16.4	8	Krizhevsky et al., 2017
ZefNet	2014	Intermediate layers outputs visualization.	60 M	ImageNet: 11.7	8	Zeiler and Fergus, 2014
VGG	2014	Homogenous topology; Small kernel size; High number of parameters.	138 M	ImageNet: 7.3	19	Simonyan and Zisserman, 2015
GoogLeNet	2015	Split Transform Merge; Introduces block concept.	4 M	ImageNet: 6.7	22	Szegedy et al., 2015
Inception-V3	2015	Handles the problem of a representational bottleneck; Replace large size filters with small filters; Replaces the bigger filter with smaller filters.	23.6 M	ImageNet: 3.5	48	Szegedy et al., 2016
Highway Networks	2015	Multi-Path Concept.	2.3 M	CIFAR-10: 7.76	19	Srivastava et al., 2015a
Inception-V4	2016	Split; Transform; Merge; Asymmetric filters.	- -	ImageNet: 4.01	- -	Szegedy et al., 2016
Inception-ResNet	2016	Residual Links.	- -	ImageNet: 3.52	- -	Szegedy et al., 2016
ResNet	2016	Residual Learning; Skip connections.	6.8 M	ImageNet: 6.7	152	He et al., 2016
DelugeNet	2016	Cross layer information inflow.	20.2 M	CIFAR-10: 3.76	146	Kuen et al., 2017
FractalNet	2016	Different path lengths interacting with each other without any residual connection.	38.6 M	CIFAR-10: 7.27	20	Larsson et al., 2019
WideResNet	2016	Width is increased and depth is decreased.	36.5 M	CIFAR-10: 3.89	28	Zagoruyko and Komodakis, 2016
Xception	2017	Depth wise Convolution followed by pointwise convolution.	22.8 M	ImageNet: 5.5	36	Chollet, 2017
Residual Attention Neural Network	2017	Introduces Attention Mechanism.	8.6 M	ImageNet: 4.8	452	Wang et al., 2017
ResNexT	2017	Cardinality; Homogeneous topology; Grouped convolution.	68.1 M	ImageNet: 4.4	101	Xie et al., 2017
Squeeze & Excitation Networks	2017	Models Interdependencies between feature maps.	27.5 M	ImageNet: 2.3	152	Hu et al., 2018a
DenseNet	2017	Cross-layer information flow.	25.6 M	CIFAR-10+: 3.46	190	Huang et al., 2017
PolyNet	2017	Experimented structural diversity; Introduces Poly Inception Module; Generalizes residual unit using Polynomial compositions.	92 M	ImageNet: 4.25	- -	Zhang et al., 2017
PyramidalNet	2017	Increases width gradually per unit.	116.4 M	ImageNet: 4.7	200	Han et al., 2017
Convolutional Block Attention Module	2018	Exploit both spatial and feature maps.	48.96 M	ImageNet: 5.59	101	Woo et al., 2018
Concurrent Squeeze & Channel Excitation	2018	Squeezing spatially followed by exciting channel-wise; Squeezing channel-wise followed by exciting spatially; Performing spatial and channel squeeze & excitation in parallel;	- -	MALC: 0.12	- -	Roy et al., 2018
Competitive Squeeze & Excitation Network	2018	Residual/identity mappings both are responsible for rescaling the channel.	36.92 M	CIFAR-10: 3.58	28	Hu et al., 2018b
MobileNetV2	2018	Tradeoff of performance vs accuracy	2.2 M	Imagenet: 7.9	40	Sandler et al., 2018a

limit the integration into heterogeneous multi-component architectures (Goertzel, 2015). These issues are addressed by architectures that have internal form states homologous to image-grammar (Zhu and Mumford, 2006) decompositions of observed entities and events (Goertzel, 2015). These works are still in the initial phase. However, some researchers believe that from training data, it would be possible to restrict the system to commonsense reasoning, similar to what operates on concepts in terms of grammatical production rules, human language acquisition, and AI (Einser, 2013).

Finally, the last set of disadvantages is related to cybersecurity. As DL evolves from the research into real applications, experiences show that Artificial Neural Networks (ANNs) are vulnerable to hacks and deception(Gu and Rigazio, 2015; Huang et al., 2019).

By identifying patterns that these neural networks use at the core, attackers can inject inputs in a way that the networks find a match that is not recognizable by humans. Such a manipulation is termed an *adversarial attack* (Gu and Rigazio, 2015; Samangouei et al., 2018). Researchers showed in (Gent, 2017) that printouts of doctored images then photographed successfully tricked an image classification system.

The DNN can be further trained to detect attempts at deception, potentially leading attackers and defenders into a similar state to what is found in the malware defense industry.

2.3.3 Video Models

Motivated by the profound learning breakthroughs in the image domain (Canziani et al., 2016), various CNNs models (Jia et al., 2014; Nelli and Nelli, 2018) are made available for extracting image features. These features are transferred to the network's last fully-connected layers activations, which perform well on transfer learning tasks (Zhang et al., 2014; Zhou et al., 2014). However, such image-based deep features are not directly suitable for videos due to a lack of motion modeling. Also, when compared to image data domains, there are only a few works on applying CNNs to video classification (Karpathy et al., 2014).

The video models can be designed as the previous sections with an extra dimension of complexity, the temporal dimension (time). The computer vision community has been working on video analysis for decades and has tackled different problems such as action recognition (Laptev and Lindeberg, 2003), abnormal event detection (Boiman and Irani, 2007), and activity understanding (Kitani et al., 2012). Considerable progress has been made in these individual problems by employing different specific solutions. However, there is still a growing need for a generic and transferable video descriptor that helps solve video tasks homogeneously.

In order to achieve this, an effective video descriptor should have four properties (Tran et al., 2015):

1. Generability. To represent different types of videos well while being discriminative. For example, Internet videos can be of sports, chats, animation, landscapes, pets,

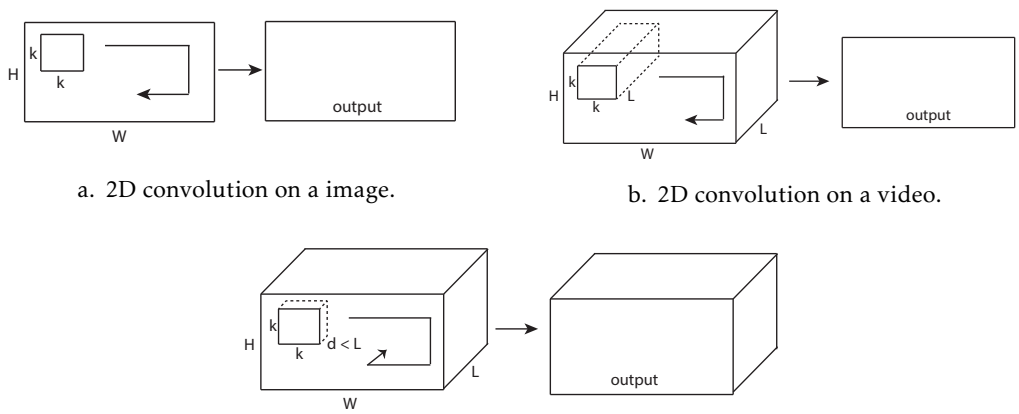
among others.

2. Compact. When working with millions of videos, a compact descriptor helps process, storing, and retrieving tasks much more scalable.

3. Efficiency. It is required to be efficient to compute thousands of videos every minute.

4. Simple. Instead of using complicated feature encoding methods and classifiers, a good descriptor should work well even with a simple model (e.g., Support Vector Machine (SVM)).

As illustrated in figure 2.13, there are three main approaches to video. The first one is to handle each frame as one image and classify it at the frame level. The second option is to consider a set of frames as input and treat them with standard 2D techniques. For example, ten frames of an RGB video would represent 30 channels of input to the network. Finally, as a third option, are presented the deep 3-dimensional convolutional networks (3D ConvNets) (Tran et al., 2015). The d represents the kernel temporal depth, k is spatial kernel size, L is the number of channels, whereas H and W are the height and width of the frame, respectively.



a. 2D convolution on a image.

b. 2D convolution on a video.

c. 3D convolution on a video.

Figure 2.13: Comparison between 2D and 3D convolution operations (Tran et al., 2015).

a) 2D convolution on an image results in an image. b) 2D convolution on a video volume (multiple frames as multiple channels) also results in an image. c) 3D convolution on a video volume results in another volume, preserving temporal information of the input signal.

A quick analysis might consider the 3D ConvNets as the best approach to most video problems. However, the computational processing and graphical virtual memory allocation are complex, and most of the SoA GPUs will struggle (usually require over 16 GBs of RAM))with handling a network, such as the one proposed by the 3D ConvNets authors (Tran et al., 2015). The proposed network (named C3D) comprises eight 3D convolutional

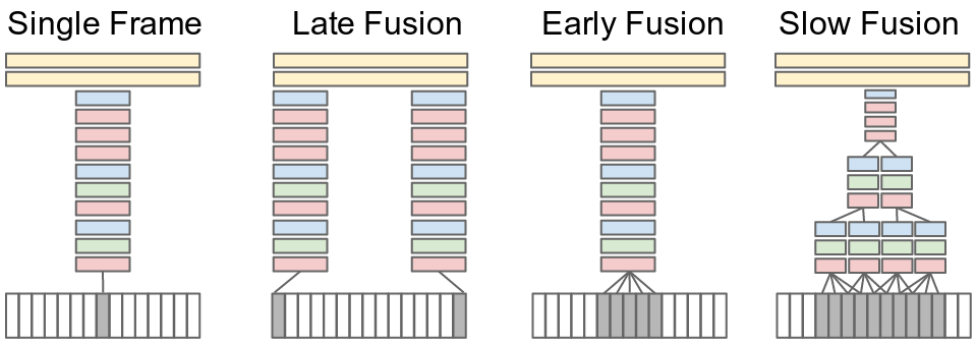


Figure 2.14: Video frame features fusing over temporal dimension through the network.

Red, green and blue boxes indicate convolutional, normalization and pooling layers respectively. (Karpathy et al., 2014).

layers, five max-pooling, and two fully connected layers, followed by a softmax output layer. The 3D convolution kernels are $3 \times 3 \times 3$ with stride 1 in both spatial and temporal dimensions. The number of filters is denoted in each box. All pooling kernels are $2 \times 2 \times 2$, except for the first, which is $1 \times 2 \times 2$. Lastly, each fully connected layer has 4096 output units.

A lighter and more used approach is to compute each frame, use a DL model as a feature extractor, and then implement another model that handles the time dimension. Since each video contains multiple frames in time, Karpathy et el. categorized the solutions to extend the connectivity of the network in the time dimension to learn spatio-temporal features into three categories (Karpathy et al., 2014). The three connectivity pattern categories are represented in figure 2.14.

The categories visualized in figure 2.14 can be described as:

- Single Frame - The standard image processing architecture baseline serves as a comparison. It is a stream of fully connected layers with n nodes. Pooling layers P pool only spatially in non-overlapping 2×2 with the addition of normalization layers. The final layer is connected to a softmax classifier with dense connections.

- Late Fusion - The Late Fusion architecture uses two separate single-frame networks with shared parameters a distance of 15 frames and then merges the two streams in the fully connected layers. Therefore, neither single frame tower alone can detect any motion, but the first fully connected layer can obtain global motion characteristics by comparing outputs of both towers.

- Early Fusion - It combines information across an entire time window immediately at a pixel level. This is implemented by modifying the filters on the first convolutional layer in the single-frame model by extending them to size $11 \times 11 \times 3 \times T$ pixels, where T is the temporal extent. The early and direct connectivity to pixel data allows the network to detect local motion direction and speed precisely.

- Slow Fusion - A balanced mix between early and late fusion that slowly fuses temporal information throughout the network such that higher layers get access to progressively more global information in both spatial and temporal dimensions. This can be implemented by extending the connectivity of all convolutional layers in time and carrying out temporal convolutions in addition to spatial convolutions to compute activations (Baccouche et al., 2011; Ji et al., 2013).

2.3.3.1 Recurrent Neural Networks

Another way to interpret video is by utilizing frames features data in RNN. On this type of network, connections between neurons form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior, and therefore use temporal video information. As shown in equation 2.10, RNNs are suitable for capturing sequential relationships (i.e temporal). A simple RNN has a recurrent hidden state that can be obtained from:

$$h_t = g(Wx_t + Uh_{t-1} + b) \tag{2.10}$$

where x_t is the m-dimensional input vector at time t, h_t the n-dimensional hidden state, g is the element-wise activation function, such as the logistic function, the hyperbolic tangent function or ReLU (Boulanger-Lewandowski et al., 2012; Caterini and Chang, 2018; Chung et al., 2014; Maas et al., 2011; Mnih et al., 2014), W, U and b are the weights and a bias parameters. Matrix W is an $n \times m$, U is an $n \times n$ matrix, and b is an $n \times 1$ matrix (or vector). An illustration of a RNN unit is despicted on figure 2.15.

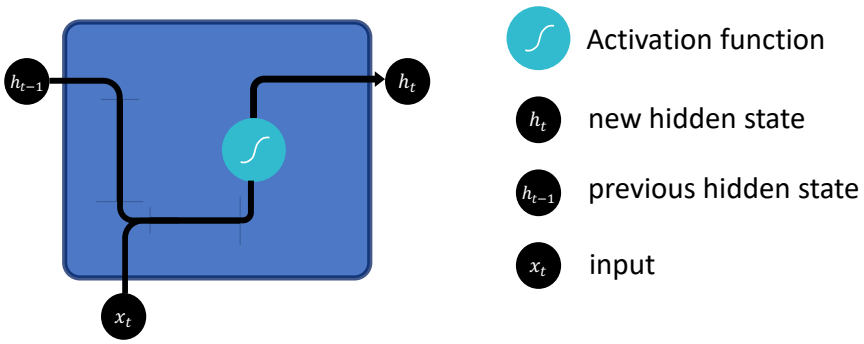


Figure 2.15: Recurrent neural network unit representation.

Some studies have shown that it is difficult to capture long-term dependencies using such simple RNNs because the gradients tend to either vanish or saturate with long sequences (Bengio et al., 1994). Two particular models, the LSTM (Gers et al., 2003; Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Chung et al., 2014) have

been proposed to solve the vanishing or exploding gradient problems. Furthermore, they have been successfully shown to perform well with long sequence applications (Boulanger-Lewandowski et al., 2012; Chung et al., 2014; Maas et al., 2011).

Long Short-Term Memory A common LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate. The cell remembers values over arbitrary time intervals, and the three gates regulate the flow of information in and out of the cell, as is presented in figure 2.16.

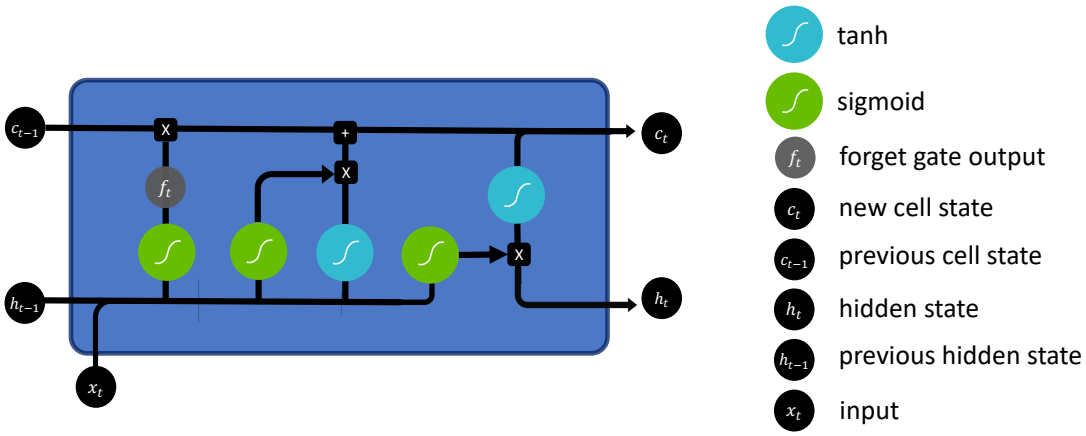


Figure 2.16: Long Short-Term Memory unit representation.

Observing figure 2.16, it is possible to retain that LSTM architectures use the computation of the simple RNN of equation 2.10 as an intermediate candidate for the internal memory cell state, $\widetilde{c}_t$, and add it in a element-wise weighted-sum to the previous value of the internal memory state c_{t-1}, producing the current value of the memory cell state c_t. This is expressed by the following discrete dynamic equations:

$$c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{c}_t \tag{2.11}$$

$$\widetilde{c}_t = g(W_c x_t + U_c h_{t-1} + b_c) \tag{2.12}$$

$$h_t = o_t \odot g(c_t) \tag{2.13}$$

In equations 2.12 and 2.13, the activation nonlinear activation function g is typically an hyperbolic tangent but more recently may be implemented as a ReLU. The weighted sum is implemented on equation 2.11 via element-wise (Hadamard) multiplication denoted by $\odot$ to gating signals. The gating signals i_t, f_t and o_t denote, respectively, the input, forget, and output gating signals at t time. These gating signals are, in fact, an analog of the basic equation 2.12, with their parameters, and replacing g by the logistic function. The logistic function limits the gating signals to a normalized value between

0 and 1. The specific mathematical form of the gating signals are thus expressed as the vector equations:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.14}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.15}$$

$$h_t = o_t \odot g(c_t) \tag{2.16}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.17}$$

On equations 2.14, 2.15, 2.16 and 2.17 σ is the logistic nonlinearity and the parameters for each gate consist of two matrices and a bias vector. Thus, the total number of parameters for the 3 gates and the memory cell structure are, respectively, W_i, U_i, b_i, W_f, U_f, b_f, W_o, U_o, b_o, W_c, U_c and b_c. It is immediately noted that the number of parameters in the LSTM model is increased 4 times from the simple RNN model. Assuming that the cell state is n-dimensional, and that the input signal is m-dimensional, the total parameters in the LSTM cell is equal to $4(n^2 + nm + n)$.

Gated Recurrent Unit The GRU reduces the gating signals when compared to the LSTM RNN model. As illustrated in figure 2.17, this unit is composed of two gates, which are known as the update gate z_t and the reset gate r_t.

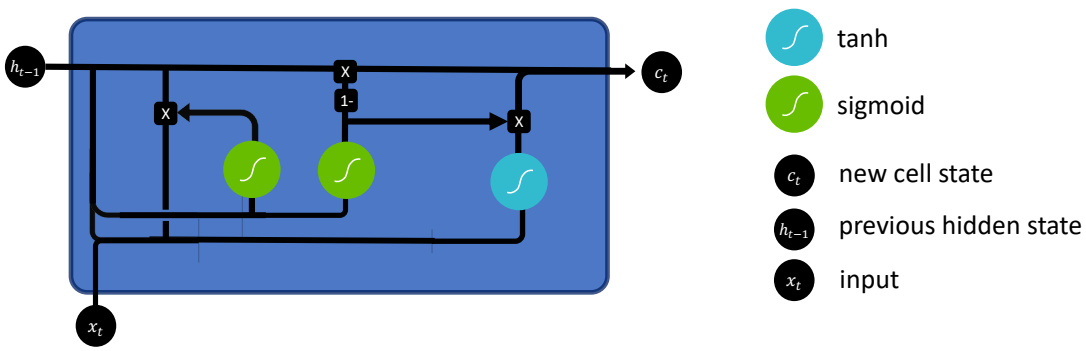


Figure 2.17: Gated Recurrent unit representation.

Mathematically, the GRU can be represented in the form:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \widetilde{h}_t \tag{2.18}$$

$$\widetilde{h}_t = g(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{2.19}$$

The two gates of GRU can be presented as:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.20}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.21}$$

It is possible to observe that the GRU equations are similar to LSTM equations, with the difference that have less external gating signal in the interpolation of equation 2.18. This saves one gating signal and the associated parameters (Bengio et al., 1994). In essence, the GRU has a 3-fold increase of parameters in comparison to the simple RNN. The total number of parameters in the GRU is equals to $3(n^2 + nm + n)$. It has been noted that GRU is comparable to, or even outperforms, the LSTM in most cases (Bengio et al., 1994). Moreover, there are other reduced gated RNNs, such as the Minimal Gated Unit (MGU), where only one gate equation is used, and it is reported that MGU performance is comparable to the GRU, and by inference, to the LSTM (Zhou et al., 2016),.

In sum, gated RNNs' success is primarily due to the gating network signaling that controls how the present input and previous memory are used to update the current activation and produce the current state. These gates have their own sets of weights which are adaptively updated in the learning phase. While these models empower successful learning in RNNs, they introduce an increase in parameterization through their gated networks. Consequently, there is an added computational expense when compared to the simple RNN model. It is noted that the LSTM employs three distinct gate networks while the GRU RNN reduces the gate networks to two.

In this chapter, base concepts that were explored in this work were examined, such as the levels of autonomy across autonomous vehicles, the SoA in collision avoidance, and AI. Additionally, some research directions explored by other authors were presented that will later be used for comparison or integration. Out of the scope of this review are detailed descriptions of some concepts and protocols in the areas of Robotics and the Cloud. If required, please consult the recommended bibliography, such as MAVLink protocol Willee, 2005, Robotic Operation System (ROS) *Powering the world's robots* 2007, Back-End La and Kim, 2010, Web Application La and Kim, 2010, docker Combe et al., 2016, and kubernetes Acuña, 2016; Cloud Native Computing Foundation, 2019. Details regarding ML algorithms are presented in appendix A.

The world of ML applied to video is just starting to emerge, and there are many unexplored paths. The author wishes that with this dissertation, new ones will be unveiled. The following chapters present a framework that accommodates different collision avoidance algorithms, which were tested in novel datasets developed during this work.

CHAPTER 3

Framework for Fully Autonomous UAVs

To deploy the dynamic collision avoidance algorithm on a UAV, it was required to design an architecture that could fully benefit from this algorithm. For this, the Framework for Fully Autonomous UAVs (FFAU) was elaborated. The core elements are implemented on top of ROS framework and make use of its abstractions and message passing systems to implement all necessary features (*Powering the world's robots* 2007). Moreover, ROS provides several modules that can be adjusted or reused to fit new purposes. In this section, we use terms intrinsic to ROS in order to explain each component represented in Figure 3.1.

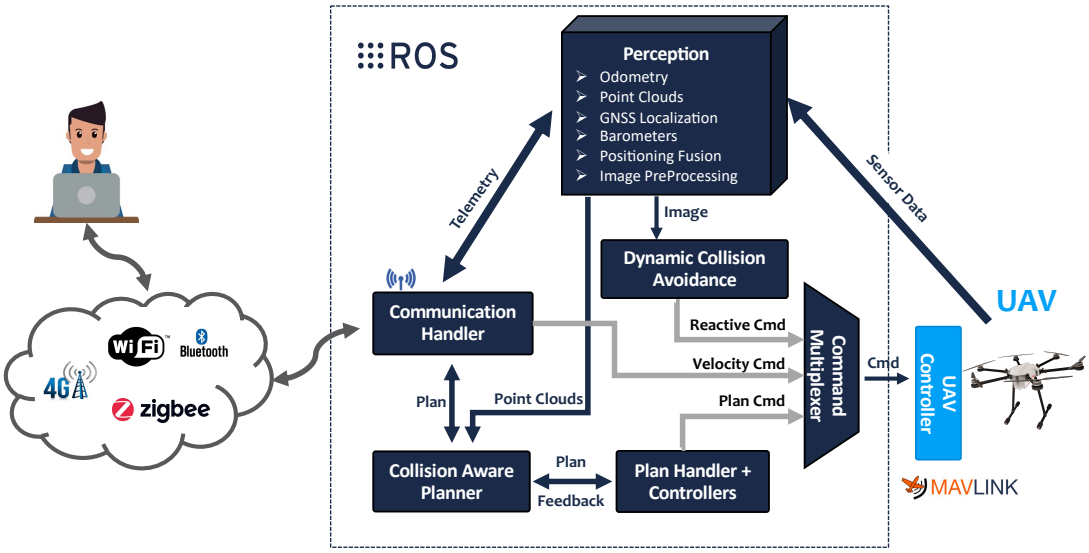


Figure 3.1: Architecture of the proposed framework for safer UAVs, on the collision avoidance task.

The operator can access via a desktop application or web application to the beXStream

platform, which allows remote control and observation of the UAV in real-time. This platform will be later detailed in this chapter.

3.1 Perception

Perception is the only component that triggers some procedure in all the remaining components, either directly or indirectly. As such, it should be considered the core node of the presented architecture. For the sake of simplicity, this component is represented as a single node on the architecture's diagram. However, in practice, the perception node is usually broken down into several standalone nodes (usually one per each sensor). Then, higher-level nodes gather the data from those nodes that are directly handling the sensors.

Nodes communicate with each other via messages published on the ROS network. A message has a specific format depending on the subject. Usually, ROS messages that refer to sensors start with the messages' followed by the name of the sensor.

The nodes that handle sensors directly (standalone nodes) are:

- **Camera Node**: Receives data from the camera and publishes it in a sensor Image message format.

  This looks like a relatively simple node, but it can be extremely complex. For example, ROS passes images in its message format, but many developers use images bundled with different image processing libraries such as OpenCV. For example, in this case, a CvBridge should be used as an interface between ROS and OpenCV. On the camera node, all necessary bridges must be implemented, dynamically initialized, and shutdown on run-time, taking into account the subscribers and publishers on the ROS network at a given time.

  It is also important to note that different cameras will require different extended nodes. For example, integrating a 360-degree camera requires different interfaces from a depth camera due to the characteristics intrinsic of each one.

- **Laser Node** (optional): Reads data from the laser sensor and publishes a sensor laser scan message on the ROS network.

- **Odometry Node**: This node can estimate the UAVs position relative to its starting point. This can be done by using the UAVs motion sensors data, performing estimations via visual odometry, or by using any fusion algorithm that mixes any such methods. The data from this node is published in a message format denominated navigation odometry messages.

- **IMU Node**: The Inertial Measurement Unit (IMU) is responsible for handling the IMU sensor (e.g., accelerometer, gyroscope, magnetometer) and periodically publishing sensor IMU messages to the ROS network.

- **GNSS Node**: Reads data from the Global Navigation Satellite System (GNSS) and periodically publishes navigation messages on the ROS network.

In each low-level node (the closest to the sensors), filtering algorithms can be applied before publishing the data on the ROS network. It is crucial to filter erroneous readings from sensors and avoid polluting the top-level nodes (and algorithms) with disposable data. Thus, on top of the first line of perception nodes, complex nodes can be built that use the filtered data. Some of these nodes are:

- **Positioning Node**: This node subscribes to all nodes that publish data related with positioning (odom, IMU and GNSS). Then, it merges the data from the different nodes (using, for example, a Kalman filter (Kalman, 1960)) and publishes the drone's current position with the best accuracy possible.

- **Structure from Motion Node**: Since not all UAVs will have a laser sensor, and most of the planners and collision avoidance algorithms require 3D data, it is essential to be able to generate a point cloud from the camera data alone. The Structure from Motion (SfM) node takes features from different frames and, hence, by calculating the feature trajectories over time, it can reconstruct their 3D positions and the camera's motion.

- **Depth image to Point Cloud Node**: Some cameras like the Intel RealSense already provide a 3D structure of the environment. This node is responsible for transforming the 3D data from the camera into a Point Cloud and then publishing that data on the ROS network.

- **Laser and imaging data fusion node**: If there are data of the type sensor laser scan messages being published by a laser and data of image messages from a monocular camera, it is possible to fuse the data of both and build a detailed point cloud. The laser gives us the range of the points in space, and the monocular camera gives us an RGB estimation of each point.

- **Imaging Object Motion Node** (with UAV behavior filtering): This node merges the positioning data with the motion analysis (optical flow) in order to better understand the movement of the bodies (objects) around him. This refined analysis can then be used by other nodes such as reactive nodes that try to precept the collision of a moving body into the UAV.

3.2 Collision Aware Planner

The Collision Aware Planner (CAL) module is responsible for establishing a safe path. This module receives one or multiple coordinates and generates a path between those coordinates, taking into account the data from the perception layer (obstacles point clouds).

For example, on an extended autonomous mission, the CAL creates a trajectory between the two waypoints of the global mission. Its output is the trajectories, an array of georeferenced trajectory waypoints.

Ground-based robots are limited to 2D navigation due to their dynamics. However, since a UAV can adjust its vertical position, 3D navigation can be implemented. Navigation in 3D gives the UAV more maneuverability to explore its environment, the ability to get a much more complete understanding of the environment, and navigate the environment through more complex paths. This is especially useful when it comes to obstacle avoidance but comes with a high complexity price.

There are multiple forms to approach the planning problem, as described in Galceran et al. survey on path planning for robotics (Galceran and Carreras, 2013). The majority of SoA algorithms assume that the world can be modeled as a simple planar surface. Hert et al. (Hert et al., 1996) applied some of the 2D knowledge to the 3D environment. Another approach is to interpolate the input goal points and set a group of escape callbacks that enter in play whenever some obstacle is detected (Azevedo et al., 2017).

In this work, two different approaches for the Static collision problem are explored. The first closely relates to the chosen robotic framework and uses MoveIt features to solve the path planning task. This solution has good debugging and visualization functionalities. Also, it was built to facilitate the integration of different planners on a single platform. It is a good solution from an academic perspective because it simplifies some algorithms' benchmark and results comparison. The drawback of this solution is that it was not developed to optimize computational performance. The second solution is more complex from the developers' point-of-view, but it outperforms the first by exploring GPU functionalities and optimizing and constrained solutions.

The first proposed solution integrates MoveIt (Chitta et al., 2012) since it has been deemed a SoA software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control, and navigation. This planner has an interface in RVIZ (Dave Hershberger, David Gossow, 2009) that can create plans, define trajectories and execute them. The Graphical User Interface (GUI) generates the Semantic Robotic Description Format (SRDF), the Unified Robot Description File (URDF), as well as other necessary configuration files.

Furthermore, this module receives one or more coordinates and generates a path between those coordinates, orchestrating the plan. The module stores the entire plan generated by the GUI and interpolates the goal points. If the distance between goal points is higher than a predefined threshold t_{PH}, the module is responsible for generating new intermediates points.

This module subscribes to all point clouds and depth images of obstacles present in the ROS network, receiving the data from the ROS Param Server. At the core of MoveIt, and the feature that CAL take the most advantage of is Open Motion Planning Library (OMPL), which is an open-source motion planning library that primarily implements motion planners. The planners in OMPL are abstract, meaning that OMPL has no concept

of a robot. Instead, MoveIt configures OMPL and provides the back-end for OMPL to work with problems in Robotics. It is important to notice that OMPL was initially developed for stationary robots, usually robotic arms, and to make it work with a UAV, it was required to implement a virtual joint to the base world.

Since OMPL has no concept of a robot, CAL requests three parameters from the ROS Param Server, namely:

- URDF: ROS move group looks for the robot description parameter on the ROS param server to get the URDF for the robot.
- SRDF: ROS obtains the robot description semantic parameter on the ROS param server to get the SRDF that is typically created (once) by a user using the MoveIt Setup Assistant.
- CAL configuration: ROS move group will look on the ROS param server for other configuration specific including joint limits, kinematics, motion planning, perception, and other information. Configuration files for these components are automatically generated by the MoveIt setup assistant and stored in the config directory of the corresponding MoveIt config package for the robot.

The second option is to use the pipeline of a point-cloud-based collision avoidance algorithm, illustrated in Figure 3.2.

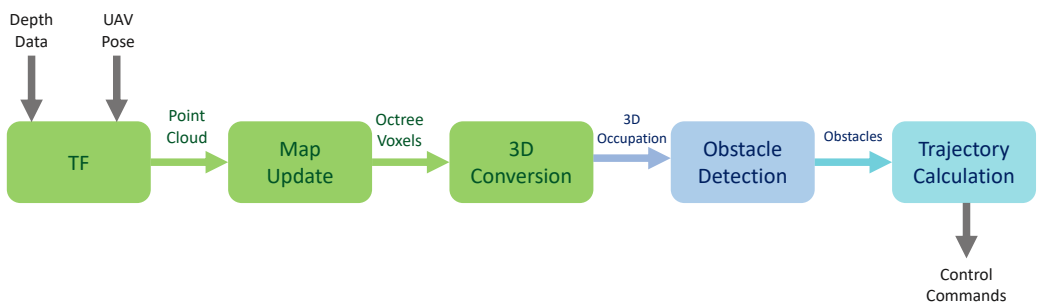


Figure 3.2: Collision avoidance execution flow.

The green blocks are related with the point cloud treatment, the darker blue to the obstacle detection and the light blue to path trajectory calculation.

To maintain data coherence across time, data collected from sensors must be mapped into a global reference frame before being placed into the map representation. In general, planners require input data to be mapped into 3D space. As a result, after an octree representation of the environment, both free and occupied voxels are mapped into 3D Euclidean space as the input for the obstacle detection phase. Finally, the collision-free trajectory is calculated from the obstacle list, and the commands are sent to the Plan Handler.

The static collision avoidance algorithm flow presented in Figure 3.2 is divided into three stages: the insertion of the point cloud and environment representation (in green); the search and detection of obstacles (in dark blue); and the final computation of the collision avoidance path (in light blue). All stages were implemented using ROS nodelets (Christian, 2016), linked to the same handler to decrease memory use (zero-copy between nodes) and processing time since the data does not need to be serialized and deserialized via the publish-subscribe method.

The leading optimization of this approach is the use of the GPU for 3D internal representation. Instead of Octomaps, this GPU approach uses GPU-Voxels (Hermann et al., 2014) in conjunction with the Voxel Map storage technique. The Voxel Map storage technique was chosen despite the high memory requirements because it allows for quicker updates while maintaining a high collision detection throughput. It also creates a distance map, which is updated on insertion and contains the distance to the nearest occupied voxel at a given location.

3.3 Plan Handler

The plan handler interpolates the trajectory points with movement constraints and command actions based on the CAL output. The plan handler node receives as input the joint state data from the UAV actuator's encoders and an input set point. It uses a generic control feedback loop mechanism, typically a Proportional Integral Derivative (PID) controller, to control the output. Since the number of joints of a UAV is usually straightforward, it is possible to create a standard action controller to translate the trajectory produced by this CAL into commands for the UAV controller.

3.4 Dynamic Collision Avoidance

The DCA node is a novelty that distinguishes the FFAU from SoA frameworks by improving safety. The core idea is that it implements the logic associated with unplanned actions that require immediate attention, for instance, if someone throws an object at the UAV while executing an autonomous mission. This node is responsible for forcing the UAV to change its trajectory and avoid the obstacle.

To accomplish this, it receives the live video stream information from the perception block, and when an oncoming collision is detected, it sends commands to the Command Multiplexer (CM) with high priority. On the proposed DCA algorithm, a set of DNN are combined to achieve this result.

The collision avoidance algorithms require the UAVs positioning to be coupled with image processing since the motion drift caused by inertia can easily lead the node to miscalculate a safe trajectory around an incoming object.

This work introduces a novel DCA node for moving object detection and a collision avoidance algorithm that uses data from a standard camera, fully described in chapter 5. The node is optional to the architecture and will only handle incoming collisions.

3.5 Command Multiplexer

Prioritizing robots' security and control topics is a mandatory precaution in nowadays' UAVs. It is required to automatically switch from an autonomous behavior to a manual command when pushing any remote controller button. This means that there is more than a single control point to move the UAV. Therefore, all those input sources must be multiplexed into a single convergence point that communicates with the hardware controller.

This node subscribes to a list of topics publishing commands and multiplexes them according to some priority criteria. The input that turns out to have the highest priority is given the control of the UAV, thus becoming the active controller. The active controller is selected according to its relevance in the control hierarchy, timeout, and input lock topics. In practice, the node will take multiple input topics from different issuers (control points) and output the messages of the issuer with the highest priority.

3.6 Communication Handler

In order to communicate with this framework, a communication module translates ROS publisher/subscriber into websockets. By doing this, it is possible to control a fleet of UAVs from any distance from the beXStream platform[1]. The module also handles the handover between WiFi, 4G, and 5G. This is done by creating a stream in all available communication channels and always using the one with the best connectivity. When the UAV is authenticated, it receives a dedicated channel to stream video and audio data using the Real-time Transport Protocol (RTP) (Paul, 1998).

3.7 Simulation

It is of utter importance to test the implementation of all intermediate steps and verify whether integrating those steps builds towards the expected result. This includes running exhaustive and extensive tests to verify if all variables and metrics previously defined fall within an acceptable interval.

Thus, to perform these tests, Gazebo simulator was integrated (Koenig and Howard, 2004) in the framework. Gazebo offers us a simulation capable of representing the real world, and so it becomes easy to do any test without damaging the hardware. The rest of the components are completely agnostic of the aircraft is being simulated, or the data is coming from a real aircraft.

[1]The developed beXStream platform can be accessed at `https://bexstream.beyond-vision.pt`

The command multiplexer is agnostic to the node he is sending commands. For example, it might be a mavros node subscribing to the topic, translating the messages into the UAV Controller, or a simulation node that allows to test and debug the developments.

3.8 beXStream - UAV Managment Cloud Platform

The beXStream platform is the mastermind of the solution[2]. It is a cloud web drone terminal that mimics a drone Radio Command with the advantage that the pilot can control a drone from any distance. The pilot only needs an internet connection between him and the drone. Furthermore, it integrates different frontend applications that enable users to monitor and execute actions with ease.

The beXStream platform can handle different types of assets, but it was developed having the use-cases of UAVs. It integrates external Application Programming Interfaces (APIs) and databases and is simple for third-party developers to integrate. Figure 3.3 depicts the platform's high-level block component architecture. In this diagram, the backend is in charge of controlling all other modules. It manages transactions between several components that are dockerized and instantiated as needed. Depending on the data type, several databases, such as influxdb (Kaplan, 2021), elasticsearch, and MySQL, are incorporated into the beXStream. Various frontend apps can use the APIs offered by the backend. Kubernetes technology was utilized to make the entire system scalable, allowing maximum usefulness from containers and building cloud-native apps that can operate anywhere, regardless of cloud-specific constraints.

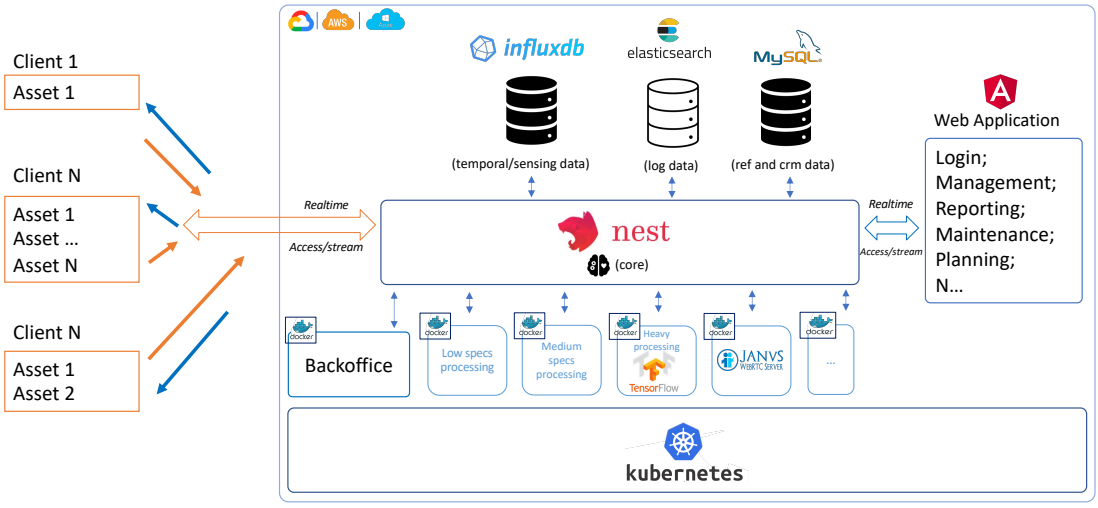


Figure 3.3: The beXStream platform architecture overview.

It is critical for safety reasons to be able to see the UAV's video feed in real-time. To solve this issue, a media-gateway based on Janus-Gateway (Amirante et al., 2014) was

[2]beXStream can be accessed `https://bexstream.beyond-vision.pt/`

implemented on the platform, in addition to the modules necessary to address network connections, which will be addressed in the next sub-modules.

3.8.1 Backend

The backend is the platform's core module, where all administration and security mechanisms are built. The backend manages the access of the many users who log in through multiple frontends, as well as the UAVs. This module manages permission levels and limits users' access to their organization, allowing them to only control/view UAVs that belong to them. That is, only people with the appropriate authority and organization may change the UAV's configurations, issue instructions, or see its stream. Multiple logins with the same account are banned for users and UAVs. The backend communicates with all of the system's modules. It generates the settings required for the video stream, which is subsequently transmitted to the Media-Gateway, depending on the camera's specifications of the UAV.

3.8.2 Media-Gateway

To handle the video streams, a media-gateway was used, in this case, Janus-Gateway (Amirante et al., 2014). This gateway can accept a stream from an UAV and broadcast it to many clients, allowing them to see the UAV's perspective. This module also provides the administration of several streams from various UAVs, as well as the change of views between them. The quality of the video stream is highly reliant on the connection quality, whether between the UAV and the platform or between the gateway and the client. The media-gateway can include a buffer for out-of-order receiving packets to decrease this vulnerability, enhancing the stream's dependability. To improve compatibility, the video is transcoded in the media-gateway, (Hanhart et al., 2018). However, most UAVs are not transcoded because of the high processing power required in the process, which is proportional to the video resolution and framerate.

To begin the transmission of the UAV stream, the UAV must be registered in the platform via a frontend. Each UAV must be assigned a unique name and password. When the registration is complete, the client receives an encrypted configuration file. This file must be included in the UAV's OBC. The file contains the access information for that specific UAV, needed to communicate with the system, encrypted using AES-128. This file must be in the home directory of the UAV. When an UAV is turned on, it decrypts the configuration file and sends the login information to the backend module, which replies with a token for this session. For example, if the UAV is equipped with a camera, then a session for the stream is created in the media-gateway. The backend will generate a configuration depending on the camera's characteristics and send the information to the media-gateway, which will give feedback once the session is created. After the session setup is completed, the media-gateway will inform the backend of what port it is expecting the stream on, informing the UAV that it can begin streaming towards

that port. A flow chart of the process between a UAV and a client can be visualized in Figure 3.4. This stream is transmitted using RTP or Real-time Streaming Protocol (RTSP). In addition, Session Traversal of UDP Through NAT (STUN) and Traversal Using Relays around NAT (TURN) may be used to resolve connections problems. The communication between the client and the media-gateway is established by WebSocket Secure (WSS). At the same time, the connection between the backend and the Media-Gateway is made by Representational state transfer (REST).

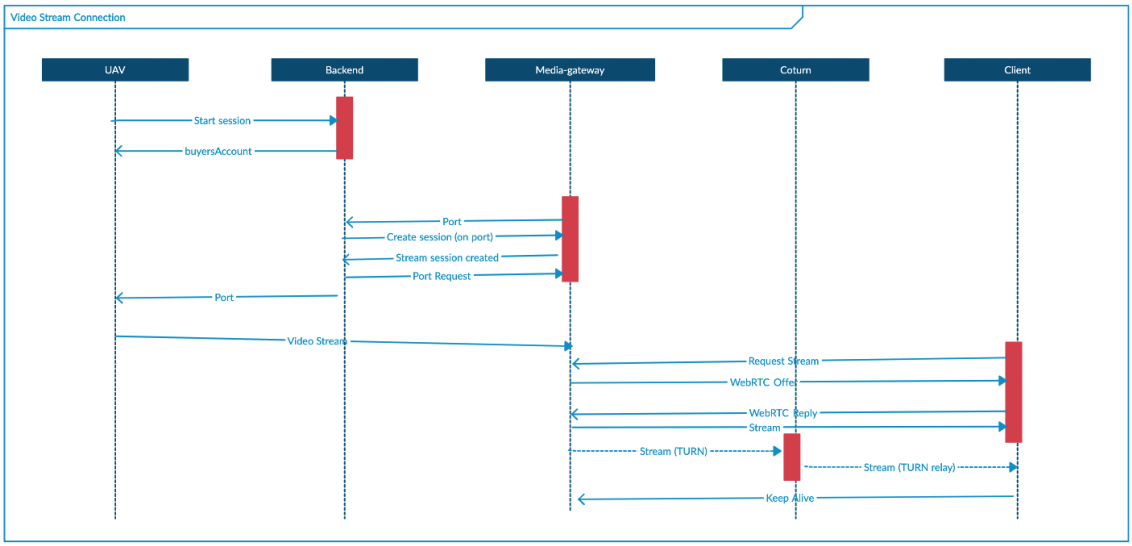


Figure 3.4: Message exchanges to establish a video stream connection between the UAV and the platform.

The dashed lines represent the possible flow the RTP/RTSP streams when using STUN or TURN.

Communication performance is an essential factor for the remote applications the authors are trying to address. The article (Pedro et al., 2020b) details the communication pipelines used, which allows telemetry data for the UAV to tolerate a delay and fit the Ultra Reliable Low Latency Communications (URLLC) requirements (Popovski et al., 2019). The tests show that in the majority of the European countries it is possible to achieve a Round Trip Time (RTT) of approximately 100 ms. Furthermore, if lower RTT is required, a new national instance of the presented backend and media gateway can be deployed, which shortens the physical traveling distances of the packages between the UAV to the backend and the frontend application. From the multiple deployment experiences, the average RTT is not as critical as the jitter, and the network congestion are, which becomes more probable with the increase of the physical distance and the different network hopping.

3.8.3 Frontend

The frontend was developed in Angular, exposing a website for the registration of UAVs, managing users/organizations, and visualizing streams. It allows visualizing details of the UAV and the landing platform, allowing the user to design different types of missions and push them to the system. The mission planning screen can be visualized in Figure 3.5. Using this page, a user can quickly draw the region of interest and remotely push the mission to the UAV. Furthermore, a video player is used with baseline characteristics to comply with all browsers on the website. The stream is received over Web Real-Time Communication (WebRTC), which uses a peer-to-peer connection. In most scenarios, the clients will be behind a Network Address Translation (NAT), which means the platform cannot address the client directly. In order to circumvent the NAT rules, a STUN server may be used, which serves to inform the corresponding peer of what their public IP is. By sharing the STUN responses, the peers can communicate even when behind NATs. However, if the client is behind a symmetric or carrier-grade NAT, then a TURN server must be used to establish the connection. In order to ensure connectivity, a TURN server is deployed in the system, based on Coturn server (Janczukowicz et al., 2015).

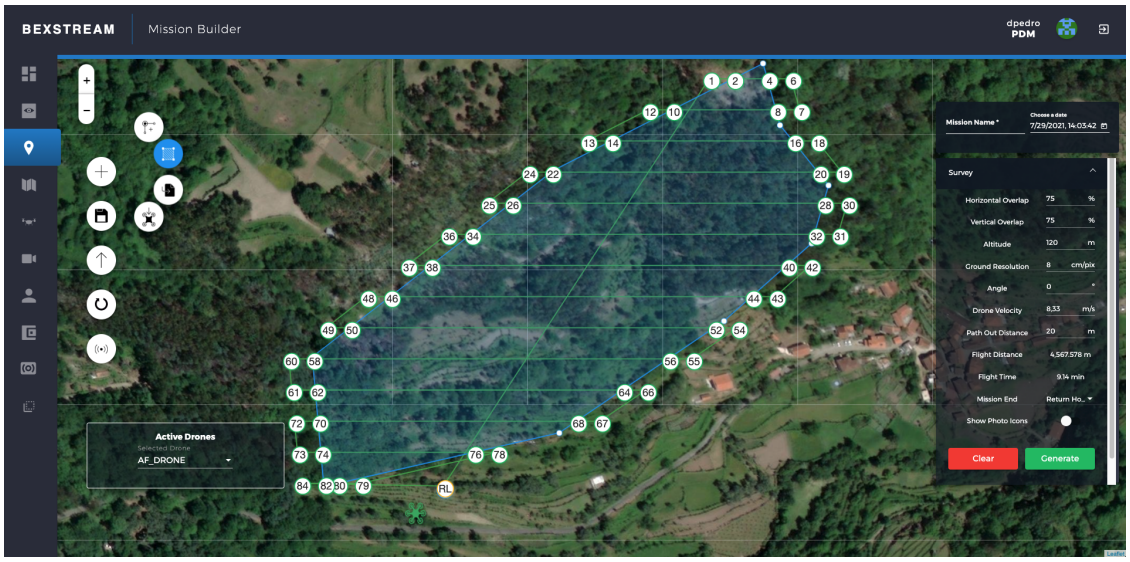


Figure 3.5: Frontend module of the beXStream platform to generate survey mission and transmiting it to a UAV.

Chapter 4

UAV Collision Avoidance Datasets

Images contain a high amount of information in a relatively concise way (Berg et al., 2012). However, their processing is complex and resource-consuming due to the infinite variations that might represent the same structure. Nevertheless, by adding a time reference and sequencing the images, it is possible to build videos. Due to the fast development of cameras, CPUs, and image/video processing algorithms (Akyildiz et al., 2007), this kind of data source is becoming widely used. For example, on YouTube (Stewart and Stewart, 2019) are uploaded approximately 72 hours of videos every minute, being expected that, by the end of 2022, online video will be responsible for four-fifths of global internet traffic.

UAVs have as main objective aiding or making possible the execution of difficult or impossible tasks for the human being (Amazon.com Inc., 2015; Hartmann and Giles, 2016). Nowadays, UAVs are a trendy tool for the industry market. Despite the advantages, UAVs' operation might be challenging when operating in complex or confined environments (Pedro et al., 2018; Ryan et al., 2004). The presence of obstacles is dangerous and requires the use of obstacle detection and collision avoidance algorithms. For the static obstacles, there are already algorithms that deal relatively well with them, being capable of generating safe paths to the desired positions (Matos-Carvalho et al., 2020). However, in real applications, it is common to find dynamic moving obstacles. The latter can be detected by using the video stream provided by the onboard cameras. Because video processing is computationally heavy (Waizenegger et al., 2011), collision avoidance algorithms need to be fast enough to retrieve solutions without colliding. Therefore, a neural network trained for this kind of situation is a plausible hypothesis that has been put to the test. The problem of the neural networks is that they are generally data-hungry, turning them extremely hard to train with small datasets, which leads to very likely memorization of the dataset (Zhao et al., 2017). In this chapter, two new open-source datasets are presented and made

available to the community, in order to accelerate and facilitate the development of new collision avoidance algorithms, increasing the safety and performance of UAVs.

In the last decade, there has been an increasing number of publications of datasets that are enabling the development of new ML models and solutions (Wu et al., 2019). Some of the existing datasets were presented in chapter 2, highlighting their relevance and impact in the field. These datasets were selected using both criteria of usefulness for the community (novel data or utility scenarios) and relevance (number of citations, usage on benchmarks zoos, scientific quality extrapolated by top-tier conferences and journals (Garcia-Garcia et al., 2018)).

4.1 ColANet Dataset

The ColANet can be summarized as a Collision Avoidance Video Dataset (Pedro et al., 2020a). This dataset is an open repository of UAV collisions and intends to be an initial step towards safer UAV operations without collisions. The ColANet dataset was developed alongside an annotation script, which simplifies the video annotation process, therefore abridging the complexity of expanding the dataset. The process starts by uploading the video with collision timings (presented in Figure 4.1), and the script will automatically generate the frame by frame annotations with an escape vector (escape direction for the UAV) for each image.

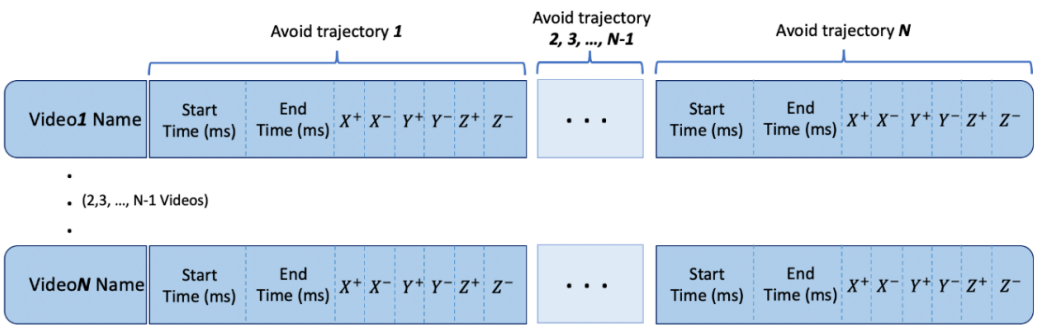


Figure 4.1: File structure used on dataset annotated frames generation one video per line.

As Figure 4.1 illustrates, each line consists of four elements:

- Video name;
- Start Time (in milliseconds),
- End Time (in milliseconds);
- Escape Vector ($X^+, X^-, Y^+, Y^-, Z^+, Z^-$).

The start time, end time, and escape vector can have multiple occurrences per video, representing multiple potential collision situations where the escape vectors might differ.

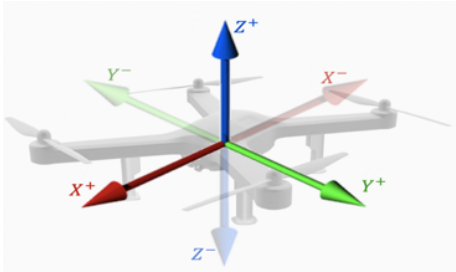


Figure 4.2: UAV with an escape vector.

With this information, the script iterates over all videos (one per line) and generates a labeled set of images extracted from the video frames. For this purpose, the algorithm opens the video, retrieves the frames per second information, and generates one image per frame and a text file containing the annotations. The directions of the escape vector are illustrated in figure 4.2, where it is possible to see a UAV with the directions vector overlaid.

The software written in python is open source and can be found alongside the dataset[1]. Note that this also gives the freedom to fine-tune the working dataset. For example, the user can run normalization and regularization when passing the data from a video to a temporal labeled set of images.

The ColANet dataset already contains 100 videos of UAVs with fast objects collisions that were recorded during flights of different models, with different environment conditions (sunny days, cloudy days, and during the night). Some examples are represented in figure 4.3. In most videos, UAVs are flying freely until the moment of collision. These videos are labeled with escape vectors and represent over 2000 collision frames and 6000 free-flying frames.

Figure 4.3: Visualization of 10 frames from 4 different collision videos on ColANet.

[1]The dataset can be downloaded at `https://colanet.qa.pdmfc.com/`

Figure 4.4: VGG16 model architecture.

Figure 4.5: VGG16 model without the classifier block.

Figure 4.6: Developed model based on the VGG16 model architecture.

To train an algorithm that classifies the current instant (frame) as collision or no collision, all it has to do is check if all the numbers on the line are 0, and he can label it as 'no collision'. However, suppose the researcher intends to estimate the escape route or free path trajectory directly. In that case, he can use the escape vector directly and, from the algorithm output, take action to avoid the collision.

4.1.1 Experimental Neural Network using ColANet

To test the ColANet dataset, a model based on VGG16 (Simonyan and Zisserman, 2015) was trained. The overlying structure of this module is depicted in figure 4.4. First, for the sake of simplification, the output of the NN was translated from an escape vector to two labels (collision or no collision). Next, the escape vector file was iterated, and the frames whose all six escape values equaled 0 were labeled as 'no collision'; and all the others received the label 'collision'.

As stated in figure 4.4, the output has the form of a 1x1000 probability vector, which is not ideal (only two classes). To overcome that, the VGG16 model was adapted, removing the classifier block (figure 4.5) and adding a new dense layer with 1024 neurons and a SoftMax activation layer (figure 4.6). The output was reduced to a 1x2 probability vector.

The newly added layers are composed of a Flatten layer (because the input is from a convolutional layer), a Dense layer with a ReLU activation function, a Dropout layer with a dropout of 50% to prevent overfitting, and finally, a Dense layer with a SoftMax activation function.

An epoch usually corresponds to the complete processing of the training set. However, the data-generator used from TensorFlow produces batches of training data for eternity. Therefore, it is required to define the number of steps we want to run for each epoch,

Table 4.1: Training classifier results for ColANet.

VGG – Pre train	**VGG – Trained classifier**		**VGG – Fully trained**	
Test	Train	Test	Train	Test
58,18%	93,89%	92,73%	96,53%	94,55%

multiplied by the batch size defined. In the illustrated model, 100 steps per epoch and a batch size of 20 are used. So a pseudo-epoch consists of 2000 random images from the training set and ran 20 times.

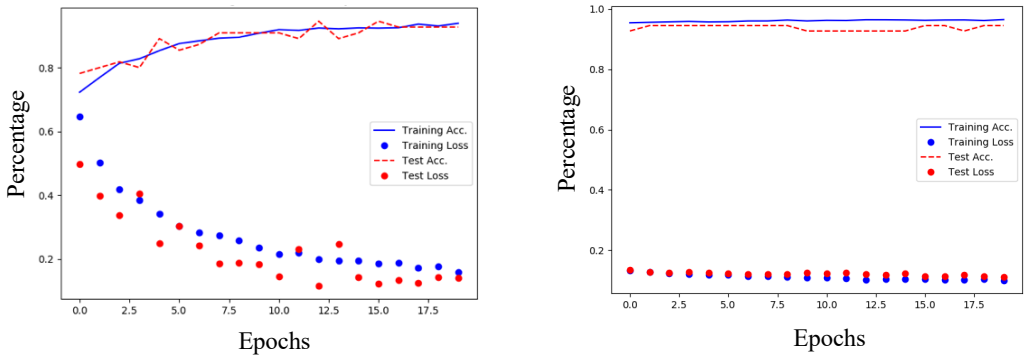
These were chosen empirically because they were enough to complete the training with this model and dataset, taking approximately 12 hours on an Intel i7 8700 with an Nvidia RTX 2070. The results also contain 20 data points (one for each pseudo-epoch) which can be plotted afterward. It is also worth noting that the input frames are all normalized to have values in the range from 0 to 1.

Using TL techniques (Pan and Yang, 2010), it is possible to check the compatibility and interoperability of the presented dataset. For that, it was used the network weights obtained in the ImageNet (Jia Deng et al., 2009; Russakovsky et al., 2015). For getting most of this pre-trained network without compromising the results of the classifier block (last layers of the model), the training was split into two phases. In the first phase, the layers of the default VGG16 model were frozen, having only the newly added layers released for training. After 20 pseudo-epochs, all the network model is released and the second phase begins. The training proceeds, but now adjusting the weights of all the layers. This technique takes advantage of the pre-trained weights of the model with another dataset to calculate the initial weights of the new layers. Releasing all the layers for training in the last step can be considered a fine-tuning of the weights calculated initially.

The ColANet dataset was used to train the presented model. The training and test accuracies were measured during all the training procedures. In order to prove that the network must be trained, the test set was evaluated before any training. Then the results were evaluated both after training the classifier (keeping the default ImageNet weights) and after the fine-tuning. The results are shown in table 4.1.

During the model training, in the first phase, an accuracy of 92,73% was reached with the test data. After the fine-tuning, the final calculated weights allowed to get an accuracy of 94,55%. Due to the similarity with the training accuracies, it can be assumed that the model was not memorizing nor overfitting the training data. In figure 4.7 is depicted the evolution of the accuracy and loss values over all the pseudo-epochs.

These results confirm that it is possible to train different models with this dataset. However, it is important to note that the default TensorFlow training/validation methods for imaging were used. Therefore, these results may lead to a false sensation of high performance, which will not translate to good results in a practical application. In addition, the videos contain many similar frames that may end up in training and validation. These



a. Training classifier

b. Full training

Figure 4.7: Training results of the VGG model on the ColANet dataset.

issues will be later explored in the chapter 5.

4.2 BallNet Dataset

The ColANet dataset has the advantage of being generic and having many different types of collisions in different scenarios provoked by multiple causes. However, it has few samples for each example. For a fully generic algorithm, a larger dataset with millions of entries per case would be necessary. Therefore, a second dataset was created to study one concrete case and better understand the importance of data for the developed algorithms. It consists of an incoming collision, where a ball was thrown at the UAV. It contains 575 videos recorded by a Parrot Bebop 2. This new dataset was entitled BallNet dataset. Some video frames are depicted in figure 4.8.

Figure 4.8: Visualization of 10 frames from 4 different collision videos on BallNet Dataset.

The BallNet dataset uses the same principles and annotation techniques as ColANet.

Furthermore, this dataset is also made available under the same server at `https://ballnet.qa.pdmfc.com/`. This dataset has 575 videos, which represent a total of 20,000 images. If necessary, the two datasets can be used for training a given model. However, it is helpful to have them separated for research purposes, allowing a faster comparison of a use-case with multiple occurrences and multi use-cases with sparse instances.

CHAPTER 5

Dynamic Collision Avoidance

To prevent a collision with a dynamic obstacle (such as an animal) or an incoming object (such as a thrown ball), a UAV needs to detect them and execute a safe maneuver to avoid them. In this chapter, the DCA problem is deeply studied. Initially, a set of evaluation metrics is presented, defining parameters and formulas used throughout the work. Afterwards, a novel solution using DL is presented and evaluated.

5.1 Classification Evaluation metrics

This section describes the performance metrics that are used in ML classification tasks evaluation of results presented in this chapter and on chapter 6.

5.1.1 Confusion matrix

In a classification problem, there is a true output y and a model-generated predicted output $\hat{y}$ for each data point. The confusion matrix is $K \times K$, where K is the number of classes. It shows the number of correct and incorrect predictions made by the classification model compared to the actual outcomes in the data.

In the context of binary classification problem with two possible classes (positive and negative classes), the results for each instance point can be assigned to one of four categories:

- True Positive (TP): the label y is positive and prediction $\hat{y}$ is also positive.
- True Negative (TN) : the label y is negative and prediction $\hat{y}$ is also negative.
- False Positive (FP) : the label y is negative but prediction $\hat{y}$ is positive.
- False Negative (FN) : the label y is positive but prediction $\hat{y}$ is negative.

The confusion matrix has the following form:

$$\begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix} \tag{5.1}$$

In multiclass problems, a given matrix row corresponds to a specific value for the "truth". Moreover, it is possible to normalize the confusion matrix by dividing each value by the sum of values in the row the value belongs. In this way, each value is between 0 and 1. This is interesting in case of class imbalance to have a more visual interpretation of which class is being misclassified.

The following metrics are computed from the confusion matrix without normalization.

5.1.2 Accuracy

The accuracy measures how close the prediction is to the real value, and for binary classification can be computed by:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

The generalization to multiclass problems is the ratio between correctly predicted labels and the total number of predictions. A classifier usually gives a set of predicted labels in decreasing order of probability, and the label with the highest probability is the predicted label (for example, a softmax classifier). Thus, the ratio between the number of cases in which correct labels are in the top k predicted labels and the total number of predictions is the top-k accuracy. However, one should be aware that accuracy is not always a good metric if a dataset is highly unbalanced. One can illustrate this based on an example. Assume a highly unbalanced dataset where 95% of the data points are not collision and 5% of the data points are collision. Then, a naive classifier that predicts no collision, regardless of input, would quickly achieve 95% accuracy. For this reason, balancing the dataset or considering other metrics such as precision, recall, and f1-score is also relevant.

5.1.3 Precision, Recall and f_1-score

Regarding precision, in binary classification,

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

The generalization to multiclass problems is to consider columns of the confusion matrix. Given that a given row of the matrix M corresponds to a specific value for the "truth", it can be presented as:

$$Precision_i = \frac{M_{ii}}{\sum_j M_{ij}} \tag{5.4}$$

and this is specific for a class i. Precision is the fraction of events where we correctly declared i out of all instances where the algorithm declared i.

For recall, in binary classification,

$$Recall = \frac{TP}{TP + FN} \tag{5.5}$$

Here again, the generalization to multiclass problems is to consider columns of the confusion matrix. Given that a given row of the matrix corresponds to a specific value for the "truth", it is obtained:

$$Recall_i = \frac{M_{ii}}{\sum_j M_{ji}} \tag{5.6}$$

and this is specific for a class i. Conversely to Precision, recall is the fraction of events where we correctly declared i out of all of the cases where the true of state of the world is i.

For f_1-score we have:

$$\text{f}_1\text{-score}_i = 2 \times \frac{Precision_i \times Recall_i}{Precision_i + Recall_i} \tag{5.7}$$

which is the harmonic mean of precision and recall.

5.1.4 Conditional Average

Whenever multiple algorithms are developed with minor modifications, the conditional average of a given metric $\overline{x}_m$ is a powerful observation. This value can be calculated by:

$$\overline{x}_m[\phi] = \frac{\sum_N \overline{x}_m[\upsilon == \phi]}{N[\upsilon == \phi]} \tag{5.8}$$

In CNNs, this is a good approach to test multiple combinations of parameters. For example, the influence of a parameter υ with the value ϕ can be estimated when observing all the results obtained with a given value and comparing it with all the combinations of that specific parameter in different tests.

5.2 Deep Learning for Collision Avoidance

In this work, it is proposed a novel solution that utilizes DL to handle the DCA problem. In order to simplify the task, a Neural Network Pipeline (NNP) with three blocks was developed. The first block is the Feature Extraction (FE) per frame. The second block handles the video temporal information (stream) with RNNs and the input of multiple SEQ feature vectors. Finally, the third block receives the result of the last RNN and uses a Feedforward Neural Network (FNN) to output a decision, which can be the collision detection or an escape trajectory. The proposed architecture is represented in Figure 5.1. These blocks will be further detailed in the following subsections. Implementations,

visualization functions, and further information can be found at `https://github.com/dario-pedro/uav-collision-avoidance/tree/master/train-models`.

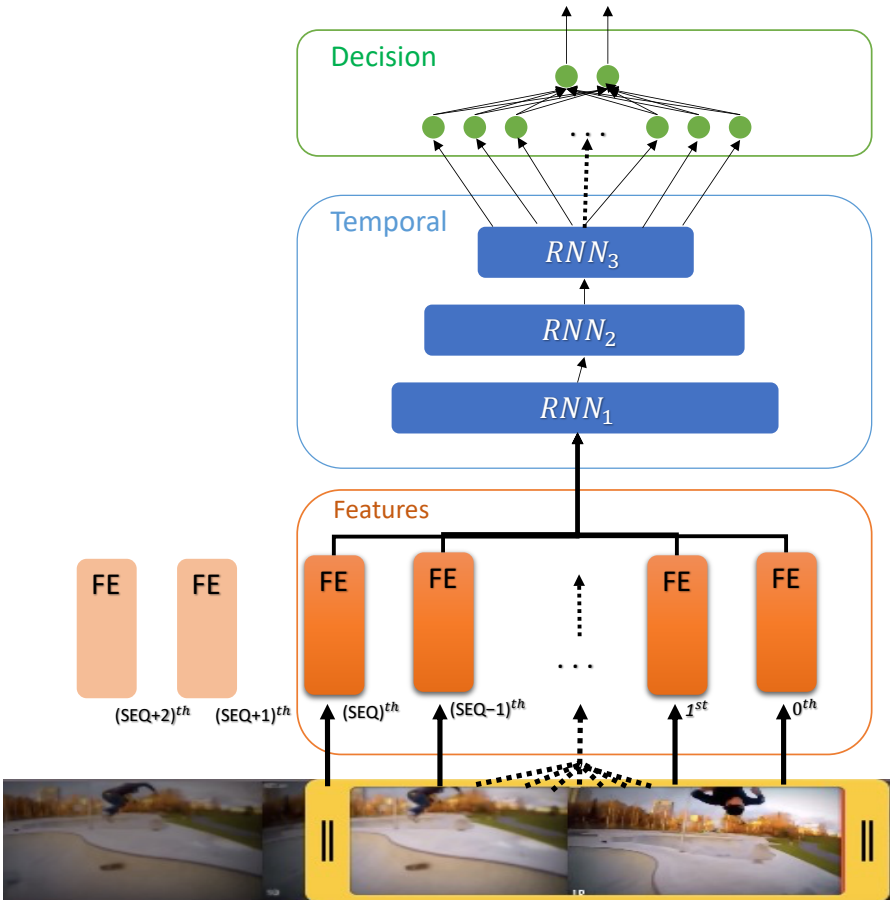


Figure 5.1: Proposed dynamic collision avoidance neural networks architecture.

5.2.1 Feature Extraction

The process of FE can be summarized in processing each frame with a CNN, which produces a feature vector, that can be interpreted as the frame key features that will ultimately be used by the NNP to detect a collision. Advances in CNNs have made this type of models ideals for FE tasks (Khan et al., 2020). Two models were tested, initially a VGG and afterwards, looking for a resource optimized CNN, the MobileNetV2 (MNV2) (Sandler et al., 2018a).

The VGG is an homogeneous and straightforward topology. This model is simple to understand at the cost of an over-dimensioned number of parameters, increasing computational requirements. The MNV2 architecture is based on an inverted residual structure where the input and output of the traditional residual block are thin bottleneck layers opposite to residual models, which use expanded representations. The initial layers are lightweight depth-wise convolutions to filter features for the middle expansion layer

Listing 5.1: Dynamic Collision Avoidance—processing the latest video frame.

```
SEQ_LEN = 25
features_queue = deque(maxlen=SEQ_LEN) # Double-ended queue

def dcaProcessFrame(videoFrame):
  # Resize image to cnn input size
  img = video_frame.resize(224,224,3)

  # ML libs predict functions outputs arrays
  cnn_pred = cnn_model.predict(img)[0]

  # Shift add the image features to the features queue
  features_queue.append(cnn_pred)

  # Check if enough images have been seen
  if(len(features_queue) >= SEQ_LEN ):
    rnn_pred = rnn_model.predict(features_queue)[0]
    return decision_model.predict(rnn_pred)[0] # return result
  else:
    return 0 # return no collision
```

(Sandler et al., 2018b). This model was selected because it has the best trade-off between accuracy and computation for a low-power processor as the present in UAVs (Howard et al., 2017; Ma et al., 2018; Sandler et al., 2018b). The model receives as input a 224×224×3 image and outputs 7×7×1280, which was converted into a 1280 feature vector by applying a 2D Global Average Pooling (Boureau et al., 2010).

5.2.2 Temporal Correlation and Decision

The temporal correlation of data features from each frame is obtained by applying a RNN. In this work, a 3-depth blocks LSTM architecture is proposed, which receives a sequence φ of input vectors. By default $\varphi = 25$, representing one second of video at a 25 frames/second rate (the average video framerate of the videos recorded on the ColANet dataset. Multiple combinations of wide and depth of LSTMs blocks are explored, always using dropout and batch normalization. Moreover, the last RNN layer is connected into a FNN with four neurons that are finally connected to two output neurons.

In a live scenario, the architecture is executed using a sliding window approach where the feature queue always contains the last 25 feature vectors and is fed into the RNN. Whenever a new video frame is available, the frame is processed by the FE, and the new feature vector is added to the features queue by shifting the previous values. Furthermore, the result of the RNN and FNN for a set of 25 feature vector array is the prediction for the last frame. Algorithm 5.1 presents the necessary sequential actions to process a new video frame, where it is assumed that all models have been previously loaded.

The procedure is optimized when compared with solutions such as conv3d (Tran et al., 2015), which apply convolutions to a 3D space. In the proposed architecture, only the last frame needs to be processed by the CNN and introduced with a shift into a deque array that is passed to the RNN. Afterwards, the RNN and FNN are triggered, which will output a prediction.

5.2.3 Training and results

To train the proposed architecture, the ColANet dataset was used. This is a video dataset of collisions and can output a classification target (collision or no collision) or a regression target, consisting of 6 values ranging from 0 to 1 representing directions front, back, left, right, up, and down. To simplify, initially, the training results are oriented to the classification problem. Afterward, the regression problem is solved by adding four additional output neurons and target each for an avoidance axis, training each neuron to the regression task. The ML frameworks TensorFlow and Keras were used to facilitate the construction and training of such networks (Shanmugamani, 2018). In this section, initially, the FE models are trained with output neurons and fine-tunned with some parameters combinations being compared. Finally, the main problems found along the process are discussed, such as the model complexity and data selection.

5.2.3.1 Feature Extrator based on VGG Train

Initially, a version using a VGG with 16 layers (figure 4.7) was tested (Simonyan and Zisserman, 2015). The results appear to be promising, but the model is too complex for standard drones to compute. For this reason, a model that require less processing was tested, the MNV2. Nevertheless, the results are presented because in the future new hardware architectures might turn possible the fitting of such architecture of UAVs computers.

To use the VGG model for FE, the same procedure that was explained in chapter 4.1.1 was followed, generating a model that outputs a 1x2 probability vector (collision or no collision).

The dataset used is highly biased for no collision frames, having $\approx 16k$ frames of no collisions and only $\approx 3k$ frames of collisions. In order to reduce this effect (in contrast to what was presented in the previous chapter), the exact value of collision and no collision frames were considered. Furthermore, to have a good test set, 5% of the video was initially left out, and the process of retaining the same amount of collision and no collision frames was applied. Note that most of the libraries split the training and validation sets by a percentage of the shuffled frames by default. This leads to highly unfair testing due to the correlation of the frames' data within the same video. Many frames have almost the same information. These can be partitioned into the training and validation set, highly increasing the accuracy of the validation set, leading to a false performance evaluation. In chapter 4, the default training routine was used to test the validity of the dataset,

Table 5.1: Training classifier accuracy of VGG16 on ColANet dataset.

Pre train	**Trained classifier**		**Fully trained**	
Test	Train	Test	Train	Test
58,18%	93,89%	79,85%	94,75%	78,10%

but for the rest of the work, a pre-selection of training and validation videos was done, preventing that frames from the same video are used both in training and validation.

In order to have a baseline, the test set was evaluated before any training (using the pre-trained weights). Then the results were evaluated after training the classifier (keeping the default ImageNet weights) and after the fine-tuning. The results are shown in table 5.1.

In figure 5.2 is depicted the evolution of the accuracy and loss values over all the pseudo-epochs. During the model training, in the first phase, it can reach an accuracy of 79,85% with the test data when training. After the fine-tuning, the final calculated weights allowed to get an accuracy of 78,10%. The neural network quickly obtained the final validation accuracy on the first five epochs, and the test did not evolve much more on the rest. The training kept increasing which hints us that it has a slight overfit. The non fine-tuned results achieve better results on validation and worse on the train, which might conclude that fine-tuning such a model makes him memorize most of the training data, generalizing worse the entire dataset. If we compare these results with the previous training results of table 4.1, it is possible to observe that the results on the training data are similar. However, the results on the test data achieve a higher percentage on the table 4.1. The most plausible reason is that the old approach of dividing the training and test set was creating a test set that was too similar to the training set, allowing the model to decorate.

5.2.3.2 Feature Extrator based on MNV2 Train

Moving to a lighter model, the MNV2 model was obtained with Tensorflow using a transfer learning approach (Pan and Yang, 2010), with the weights pre-trained on the ImageNet dataset. First, it was chosen which layer of MNV2 is used for FE. The very last classification layer [1] is not very useful. Instead, it is common practice to use the very last layer before the flatten operation. This layer is called the "bottleneck layer". The bottleneck features retain much generality as compared to the final/top layer. This can be done by specifying not to include the top layers, loading a network that does not include the classification layers at the top, which is ideal for FE.

Afterwards, all the layers are frozen before compiling the model, which prevents

[1] on "top", as most diagrams of machine learning models go from bottom to top

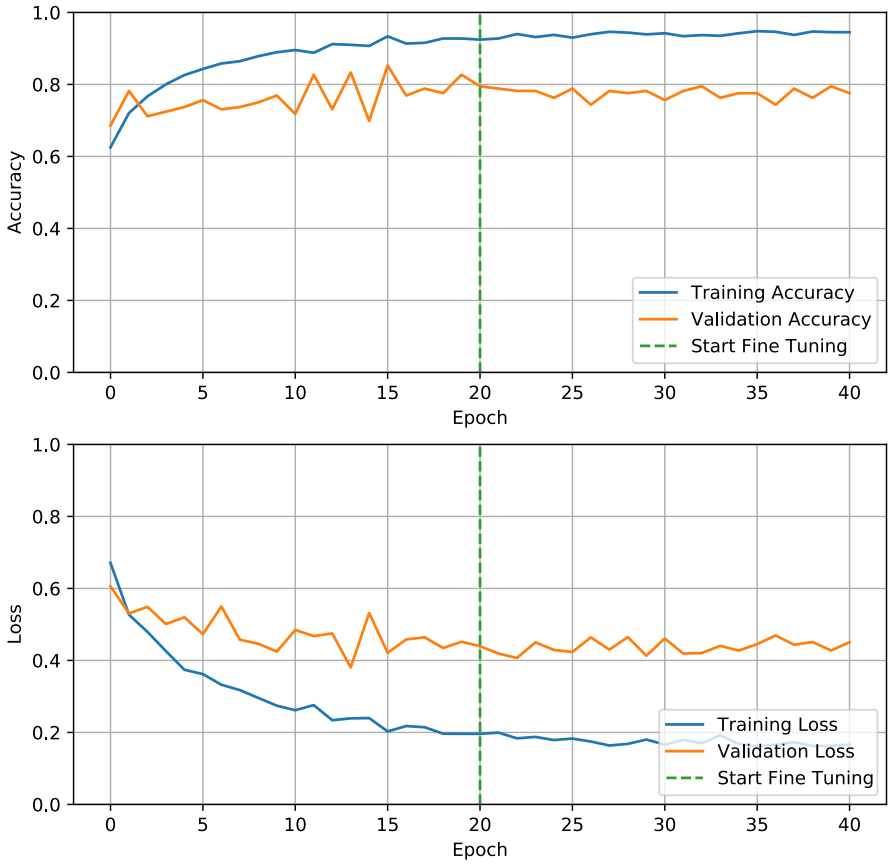


Figure 5.2: Training evolution of the FE based on VGG16 model.

On the first 20 epochs only the classification neurons are trained. Afterwards, the entire model is fine-tuned.

weights from being updated during training. Then, a classification block is added, composed of a Global Average Pooling 2D layer to convert the features to a single 1280-element vector per image, and a Dense layer to convert these features into a single prediction per image. Adding an activation function is unnecessary because this prediction will be treated as a logit or a raw prediction value. Positive numbers predict class 1; negative numbers predict class 0. This last classification layer was trained to give it some knowledge of the objective goal, using a binary cross-entropy loss and an Adam optimizer (Kingma and Ba, 2015) with $1x10^{-4}$ learning rate and $1x10^{-6}$ decay rate. For our classification problem, the loss can be depicted as the equation 5.9,

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \tag{5.9}$$

where w refer to the model parameters (weights), N are input values (images),y_i is the true label and $\hat{y}_i$ is the predicted label. The accuracy metric is given by the equation 5.10.

Table 5.2: Training classifier results of MNV2 on ColANet dataset.

	Pre train	Trained classifier		Fully trained	
	Test	Train	Test	Train	Test
Accuracy	50,01%	68,25%	64,30%	94,34%	80,29%
Loss	0.843	0.588	0.635	0.221	0.439

$$\text{accuracy}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} 1(\hat{y}_i = y_i) \tag{5.10}$$

The model was trained in two steps. On the first step, all layers except the newly added classification block were frozen. Using this setup, the model was trained for 20 epochs. On the second step, using the previous network, the fine-tuned version of the MNV2 base model was generated. For this, all the layers were unfrozen. It is important to note that the first step is mandatory because, if a randomly initialized classifier is added on top of a pre-trained model and attempt to train all layers jointly, the magnitude of the gradient updates will be too large (due to the random weights from the classifier). Moreover, the pre-trained model will forget what it has learned before (the transferred knowledge).

The training results are presented in figure 5.3. At the center of the graph, a green dashed line separates the two steps. As expected, the model starts with $50,01\%$ validation accuracy and finishes the first train with approximately 68.25%. When the whole model is fine-tuned, the network manages to reach $80,29\%$ validation accuracy. It is possible to detect slight overfitting towards the end of the training because both accuracy and loss start diverging when comparing training and validation lines. Not only that, but the training accuracy is close to 100% with a significant gap to the validation accuracy. On table 5.2 the summary of the results is presented.

The fine-tuning of the MNV2 network gives a model that is highly oriented to the collision classification problem. On the one hand, this is good because it allows CNN to prioritize some features, but on the other hand, it is terrible because it generalizes the world worse. For this reason, the training of the RNN+FNN blocks will be presented in two versions. The first with the FE with only the general knowledge transferred from ImageNet, and the second with the FE fine-tuned in the ColANet dataset.

5.2.3.3 DCA Model Train

Initially, the RNN+FNN blocks are trained with feature data from the CNN, which has not been fine-tuned with ColANet data. In order to prepare the data for the RNN+FNN model, some constraints must be placed:

1. The input data must be an array of φ length (input sequences), which is the first

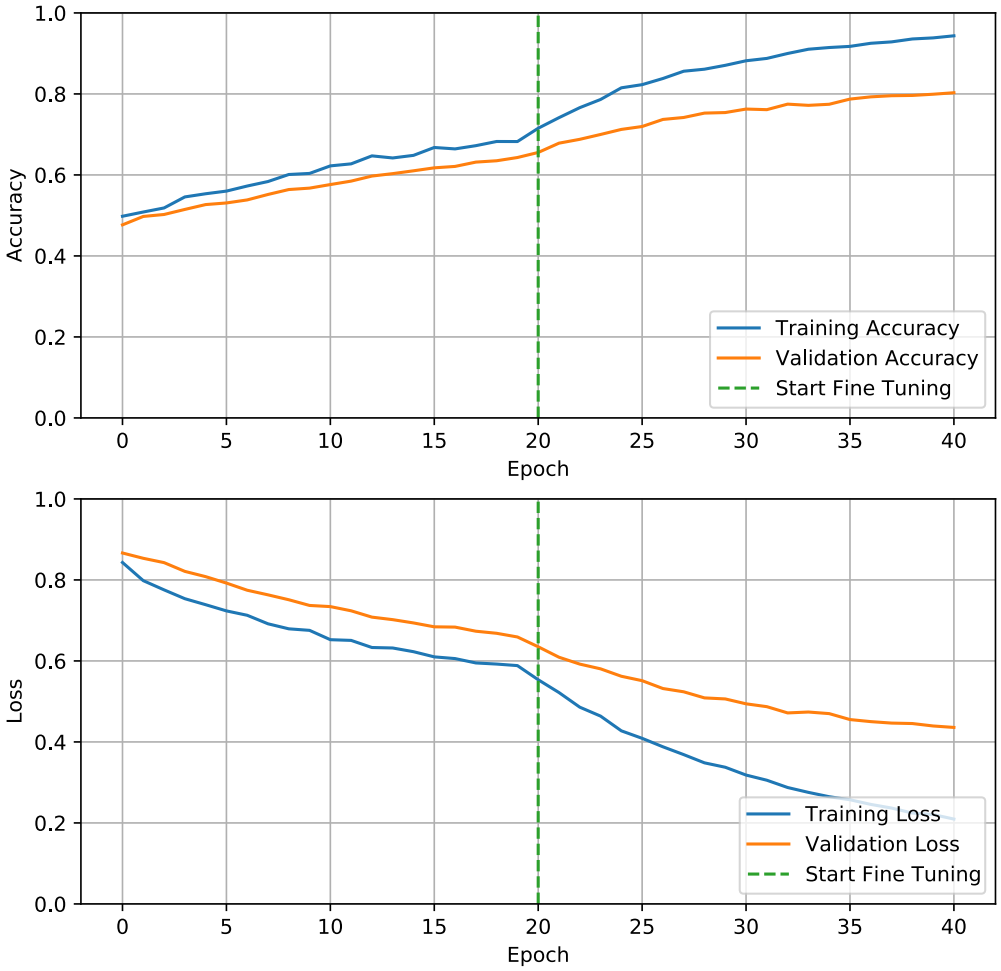


Figure 5.3: Training evolution of the FE based on MNV2 model.

On the first 20 epochs only the classification neurons are trained. Afterwards, the entire model is fine-tuned.

set of tests it was considered to be 25 (average Frames Per Second (FPS)). Any value between 20 and 50 achieved similar results.

2. The input sequences must only contain frames belonging to the same video. Working with video data on GPUs is not a trivial task, and generating video sequences adds an overhead. The dataset is seen by the model as a continuous stream of data, and this constrain must be enforced, so that the model does not learn jumps between videos (false knowledge).

3. The last frame target label is the target for the entire input sequence.

Since this data is easier to analyze using two output neurons, the Sparse Categorical Cross Entropy Loss was selected. Nevertheless, on Tensorflow equation 5.9 applies. The Adam optimizer (Kingma and Ba, 2015) with $1x10^{-4}$ learning rate and $1x10^{-6}$ decay rate was used. Furthermore, in order to understand the influence of the number of LSTMs

and dense layers, a total of $7^4(2401)$ networks were trained, using a combination of $[64, 32, 16, 8, 4, 2, 1]$ units per layer. The same number of sequences with collisions and no collisions was pre-processed, and 5% of the videos were placed on the validation test, which means ≈ 6000 train feature sequences and ≈ 300 validation feature sequences (the value vary because each video has a different number of collision frames).

In both cases (default FE and fine-tuned FE), the networks were trained on a Nvidia GeForce RTX 2070 with Compute Unified Device Architecture (CUDA) libraries. They trained for 20 epochs with a batch size of 32, resulting in an average train time per model $\mu = 130$ s and a standard deviation $\sigma = 1$ s. This represents approximately 87 h 45 m to train the desired models variants per FE used. An Adam optimizer with a learning rate of 1×10^{-3} and a decay rate of 1×10^{-6} was considered. Furthermore, a dropout of 40% and batch normalization was added to the outputs of LSTMs layers.

A detailed analysis of the results is presented on the appendix B, which use the FE directly. The same exercise was done with the FE fine-tuned, and the resulting table and graphs were similar (as a global result). The main results are summarized bellow.

The most important metric is the validation accuracy because it represents the accuracy of the network on unknown data. Nevertheless, looking at the best scoring networks on the validation accuracy metric can be misleading for multiple reasons. A network can enter a local minima point where the value of validation accuracy is higher than the training accuracy (sometimes considered underfitting). Also, all the top-best scoring slightly increased accuracy on the last epoch, which boosted the results.

Regarding the test with the default FE, the LSTM layers with 16 units produced the best results on average, but having all layers with 16 units does not generate a good model. The influence of each unit value can be observed in table 5.3. The values represent conditional averages with the parameters. Furthermore, per each row (layer), the values are highlighted from best to worst using green to red colors, respectively.

Table 5.3: Validation accuracy mean with variation of the number of units per layer using Sequences from the default MNV2.

Layers	**Units**						
	64	**32**	**16**	**8**	**4**	**2**	**1**
LSTM 1	60.90%	65.91%	69.26%	70.91%	70.87%	68.02%	61.98%
LSTM 2	68.08%	66.65%	67.90%	67.29%	66.57%	66.23%	65.14%
LSTM 3	65.76%	65.62%	65.68%	67.79%	66.84%	67.31%	68.84%
Dense	66.41%	66.95%	67.29%	67.27%	66.20%	66.73%	67.01%

From the results, it is possible to conclude that the initial LSTM block requires more units than the following layers. Regarding the last dense layer, there are no obvious candidates, since on average, all networks returned similar validation accuracy results. However, the training accuracy was higher with the increase of dense units, which could mean that it started overfitting easily.

The training evolution graph of the DCA model with parameters $N_{L1} = 4$, $N_{L2} = 2$, $N_{L3} = 32$ and $N_{D1} = 4$ is represented on figure 5.4. This is a good example as training and validation accuracies evolve together. This model achieves a final validation accuracy score of 92,14 %. The average of the top 10 best scoring models on the validation accuracy metric using the default FE is 92,29 %.

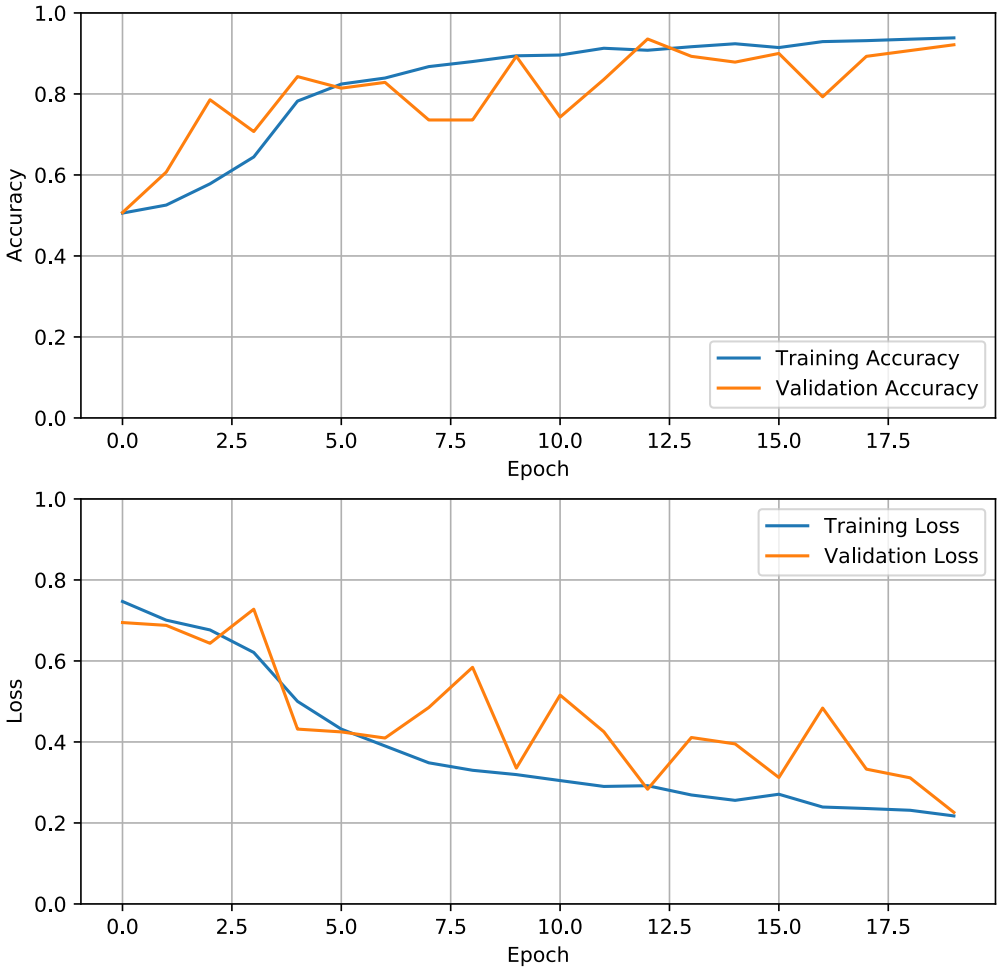


Figure 5.4: Training evolution graph of the DCA model with parameters $N_{L1} = 4$, $N_{L2} = 2$, $N_{L3} = 32$ and $N_{D1} = 4$.

On almost all of the training evolution graphs, it is possible to observe that the models tend to start overfitting around the 6th epoch since the accuracy validation line starts stabilizing around 80% where the training accuracy keeps increasing to values above 95%.

At this point, some considerations can be made regarding image and temporal data. The image data trained CNN produced a score of 80% accuracy on unseen data, whereas the RNN managed to use temporal features from an untrained MNV2 and reached an accuracy of above 90%. This leads us to believe that temporal information has more importance in the collision avoidance problem or even in any video-related classification

problem. Having this note pointed out, it is time to use the trained CNN to generate the features for a new round of RNN trains and analyzes. For this, the same procedure was conducted, but this time with the features arrays being generated using the fine-tuned MNV2.

A more extended training of the algorithm would not necessarily improve performance due to the overfitting effect. This is visible in figure 5.5, where the model was left running 1000 epochs.

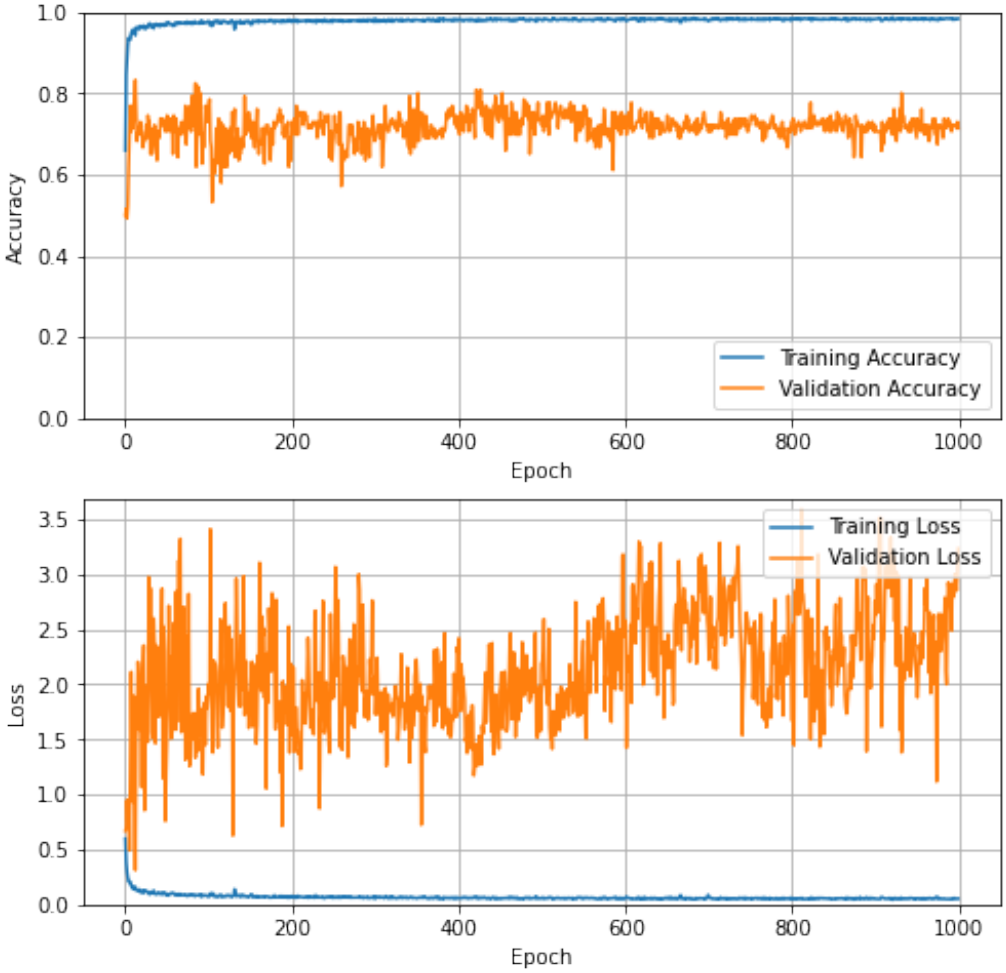


Figure 5.5: Training evolution graph of the DCA model with 1000 epochs, which overfits the network.

On table 5.4 is represented the average influence of each unit value on the validation accuracy of the models trained with data from the fine-tuned MNV2. Comparing with table 5.3 the results were on average 2.7 % higher. Nevertheless, when looking for the top-scoring models, the results are similar. This suggests that the fine-tune increased the values for the worse case by pointing a worse RNN+FNN into the right direction, but on the top results, when the key is generalization, it did not make much difference.

The dataset is relatively small and diverse, which makes training and generalization

Table 5.4: Validation accuracy mean with variation of the number of units per layer using Sequences from the fine-tuned MNV2.

Layers	Units						
	64	32	16	8	4	2	1
LSTM 1	63.99%	68.52%	72.12%	73.60%	73.17%	71.21%	64.10%
LSTM 2	70.46%	70.47%	68.94%	70.12%	70.62%	68.33%	67.77%
LSTM 3	67.48%	68.82%	69.65%	69.88%	70.24%	70.60%	70.05%
Dense	69.82%	69.74%	68.60%	69.44%	69.98%	69.32%	69.82%

tasks hard. This can be observed on the graph of figure 5.6, where it has presented the accuracy results of all models. On the abscissa is the training accuracy, and on the ordinate is the validation. It is possible to visualize that a few models were stuck on a local minimum and could not reach a value higher than 50% on validation. Furthermore, many models achieved near-perfect accuracy during training but a lower value on validation, which hints that the dataset has little data and the model is overfitting. Nevertheless, many models achieve a performance higher than 85%, proving that this is a reasonable solution for building models for the collision avoidance problem.

Figure 5.6: Accuracy results of DCA model with fine-tuned FE.

It is possible to conclude that this is a valid approach for the DCA problem. The ColANet is a relatively recent dataset and still has a low number of UAV collision videos

(less than 100), making the training task harder due to the model's tendency to overfit. As the number of available videos increases, the possibility of using this algorithm on daily UAVs will increase.

In order to have the neural network estimating the collision vector, the last two neurons of the FNN need to be swapped for six neurons that perform regression for each axis.

5.2.3.4 Repeating the process on BallNet

The ColANet is a good dataset for validating the solution and applicability for different use-cases. However, to compare the algorithm with SoA algorithms, a concrete use-case is better because it allows a direct comparison. For this reason, the scenario of avoiding a thrown ball present in BallNet was considered.

Following the same steps as in the previous subsections, initially, the FE was trained using the MNV2 model. The training results of the added classifier are seen in Figure 5.7 for the first 20 epochs (before fine-tuning). Using only single frame knowledge, the model predicted collisions with 54,4% accuracy at the end of the 20 epochs (validation accuracy). Following that, a refined version of the MNV2 base model was trained. To do this, all layers were unfrozen. It is vital to emphasize that the first step is needed if a randomly initialized classifier is applied on top of a pre-trained model. Afterwards, all layers are jointly trained. If this was not done by this steps, the magnitude of the gradient updates will be too high (due to the classifier's random weights), and the pre-trained model would forget what it has learned (the transferred knowledge).

The fine-tuned FE's training results are the last 20 epochs of Figure 5.7 and obtained a final validation precision of 66.8%. The disparity between training and validation began to increase on the final epochs, indicating the beginning of over-fitting, and no further epochs were trained as a result.

Afterwards, the temporal block was tackled. The Adam optimizer (Kingma and Ba, 2015) was used, with a learning rate of 1×10^{-4} and a decay rate of 1×10^{-6}. Figure 5.8 shows the training results of the RNN with the FNN classification layer with moved FE weights and fine-tuned FE.

Table 5.5 summarizes the outcomes of the proposed models pipelines. It is possible to infer that it is a viable solution for detecting incoming collisions, but more research, datasets, and testing are needed. On unseen data, the trained MNV2 achieved an accuracy of 66,8 %, while the complete NNP using temporal features from an untrained MNV2 achieved an accuracy of over 89 %. This confirms the theory that temporal information is important in the collision detection problem (or possibly in any video-related classification problem).

Fine-tuning the MNV2 improved the results of the NNP, but it is a slight trade-off between generalization and dataset performance. The presented dataset is relatively small and narrow, with a limited range of environments and variability. Because of the

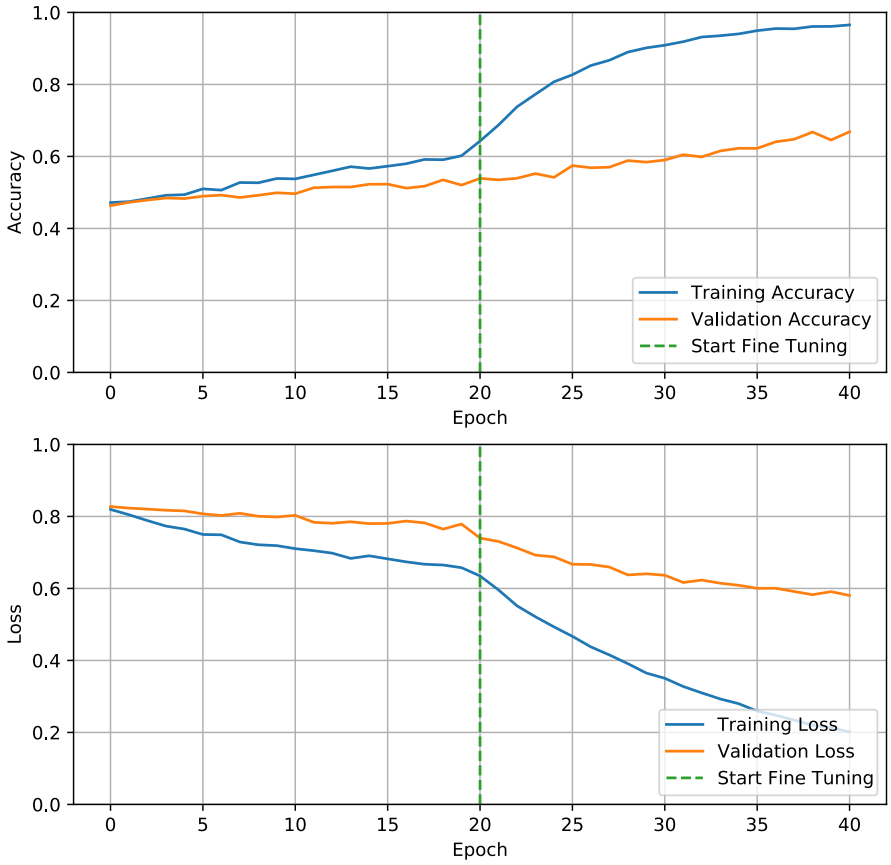


Figure 5.7: Training Feature Extraction based on MobileNetV2 model. On the first 20 epochs, only output neurons are trained. Afterwards, the entire model is fine tuned.

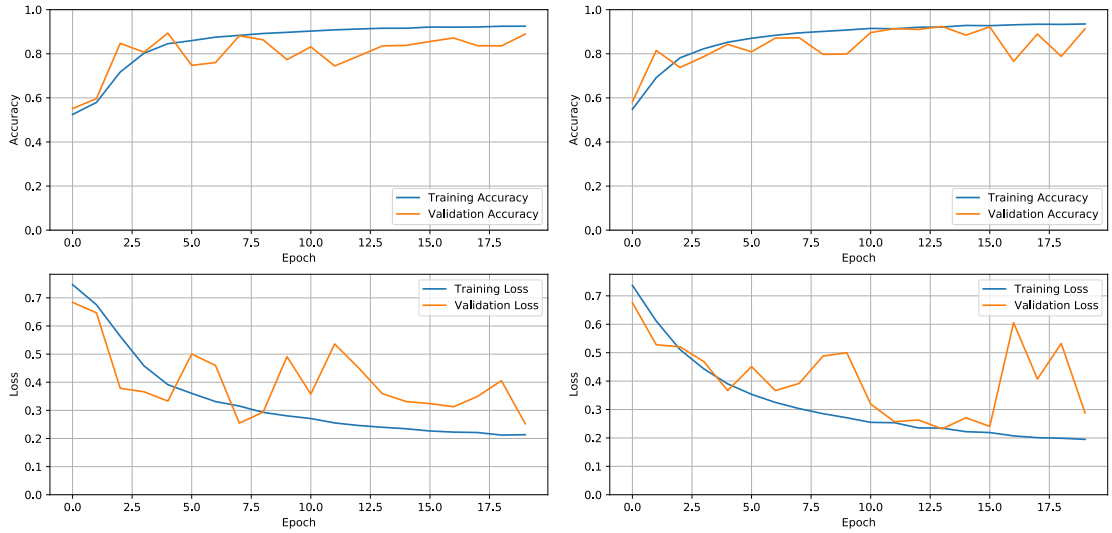
model's proclivity to overfit, this makes preparation more difficult. Nevertheless, the models can be further generalized and show better results as the amount of available UAV video datasets grows.

Table 5.5: Collision avoidance trained models' results comparison on the BallNet dataset.

Metrics	FE_1 MNV2	Fine-Tuned MNV2	NNP w/ MNV2	NNP w/ Fine-Tuned MNV2
Training Accuracy	64.6%	97,4%	92,6%	93,4%
Validation Accuracy	54,4%	66,8%	89,4%	91,4%

5.2.4 Features Grad-CAM

The results of the proposed DNN composition seemed promising, but meanwhile, these models achieved high performance. Unfortunately, their lack of decomposability into individually intuitive components makes them difficult to interpret (Lipton, 2018). As a result, when today's smart systems fail, they often fail spectacularly shamefully without warning or explanation, leaving a user staring at an inconsistent output, wondering why the system did what it did. This is even more worrying when evaluating algorithms



(a.) First iteration

(b.) Second iteration

Figure 5.8: Training evolution graph of the NNP models.

a) First iteration - Using FE with the default MNV2 weights (ImageNet weights); b) Second iteration - Using FE fine-tuned weights.

intended to run on autonomous vehicles.

A way to minimize this effect is by enforcing Interpretability. To build trust in intelligent systems and move towards their meaningful integration into UAVs, it is clear that it is needed to build 'transparent' models that can explain their predictions.

For this reason, the Gradient-weighted Class Activation Mapping (Grad-CAM) was integrated (Selvaraju et al., 2019). As it can be observed in figure 5.9, this algorithm receives an image and a class of interest as input. The image is forward propagated through the CNN, part of the model, and then, through task-specific computations, it obtains a raw score for the category. The gradients are set to zero for all classes except the desired class (collision), which is set to 1. This signal is then backpropagated to the rectified convolutional feature maps of interest, which are combined to compute the coarse DNN localization (blue heatmap) which represents where the model has to look to make the particular decision. Finally, a pointwise multiplication to the heatmap with guided backpropagation to get Guided Grad-CAM visualizations which are both high-resolution and concept-specific.

The Grad-CAM was applied to the MNV2 with its two flavors. The first, with the pre-trained weights from ImageNet, and the second with the fine-tunned weights on the ColANet dataset(section 5.2.3.2). This provides insights into which areas of the network's image are relevant for the collision classification and a visual representation of the influence of fine-tuning a network. The Grad-CAM outputs a $9x9$ matrix with normalized weights from 0 to 1, which can be rescaled to the input image size, equalized to the output values, and generate a superimposed visualization. On the figure 5.10 is illustrated:

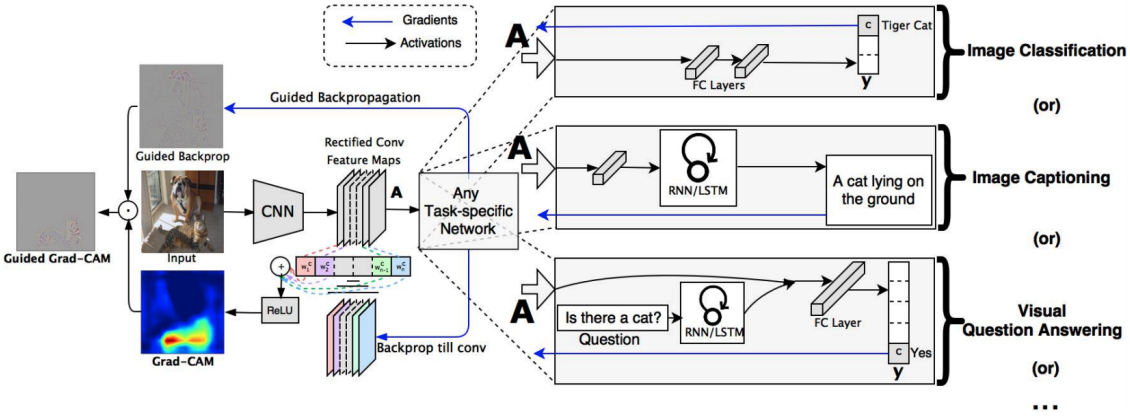


Figure 5.9: Grad-CAM pipeline (Selvaraju et al., 2019).

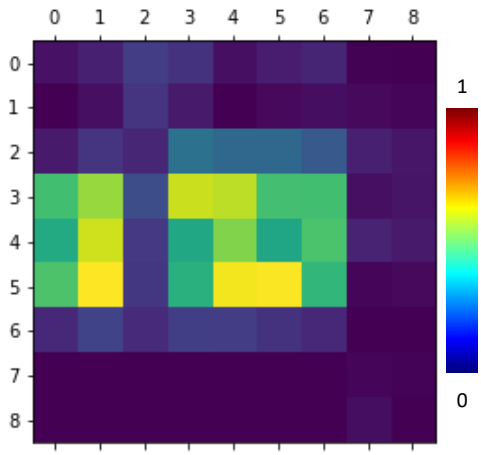
throwing a ball at the drone at image a, the result of the Grad-CAM $9x9$ matrix using the MNV2 with the pre-trained weights at image b; the superimposed visualization of the Grad-CAM result b and the input image a; the result of the Grad-CAM $9x9$ matrix using the MNV2 with the fine-tunned weights at image d; and finally the superimposed visualization of the Grad-CAM result d and the input image a. Observing the results of the Grad-CAM using the fine-tuned MNV2, it is possible to verify that the region where the ball is present attracts more attention from the network, being the main reason for its output. In contrast, the MNV2 with pre-trained weights seem to be focusing on the person rather than the ball.

This technique was applied to the remaining ColANet dataset. A couple of examples are presented in figure 5.11. The images a, b, and c have most of the focus on the incoming collision object, whereas image d has two attention hot-spots: the first on the colliding snow goose and the second on the background windmill. This is a good revelation that spatio-image processing alone is incapable of handling the collision detection task, and for this sort of case, the temporal information might be relevant. For example, the windmill will not be approaching the camera, whereas the snow goose will be at high speed.

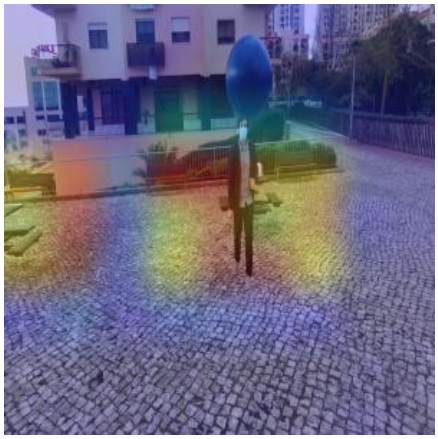
The use of Grad-CAM helps to pock holes in the black box of the trained CNN models. Furthermore, it provides a visual and straightforward way to detect issues and possible problems and areas where the algorithms might fail. The snow goose frame is a good example where the CNN is not focusing on the main features, and that cannot provide a valid output alone (on real use-cases deployment).

In the author's opinion, a true AI system should not only be producing above SoA outputs (decisions), but also be able to reason about its beliefs and actions, being capable of explaining them, and of making humans trust and use it. Of course, the necessary trust level varies per application, but everything that involves safety must have methods to provide answers.

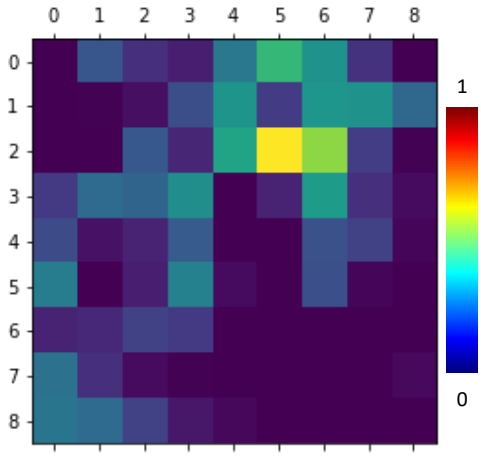
a. Input frame - imminent ball collision.



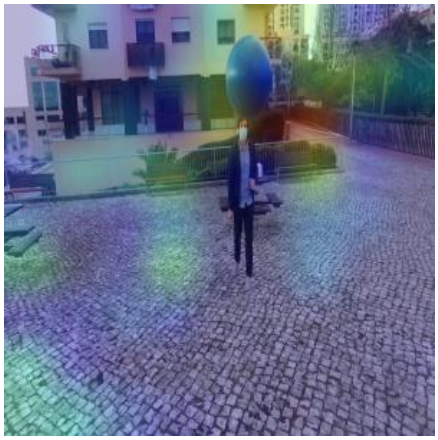
b. MNV2 pre-trained - Grad-CAM result.



c. Superimposed Visualization of *b* in *a*



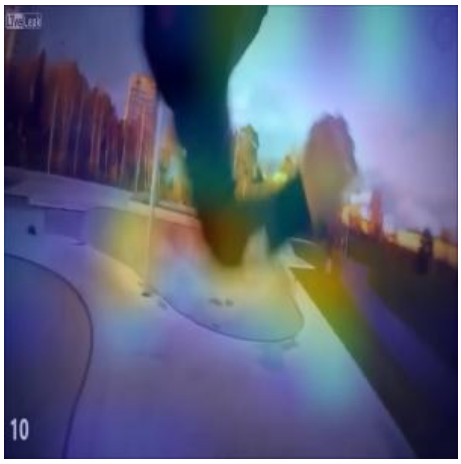
d. MNV2 fine-tunned - Grad-CAM result.



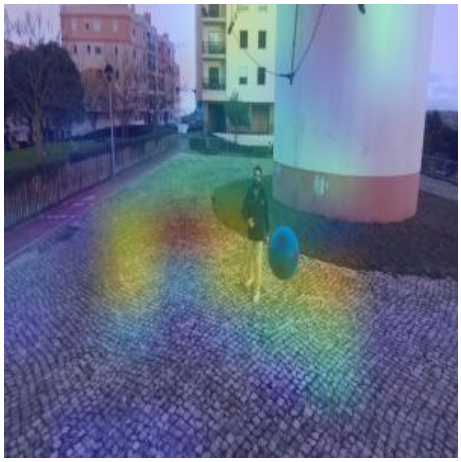
e. Superimposed Visualization of *d* in *a*

Figure 5.10: Grad-CAM results from a MobileNetV2 (pre-trained and fine-tuned) given as input a frame with an imminent ball collision.

a. Collision with a thrown ball (subject 1).



b. Collision with a skater arm.



c. Collision with a thrown ball (subject 2).

d. Collision with a snow goose.

Figure 5.11: Grad-CAM Superimposed Visualization in multiple images, using a MobileNetV2 fine-tuned.

5.3 Object Motion Estimation

The NNP presented is capable of detecting collisions or estimating escape trajectories. Nevertheless, for practical scenarios, some federal organizations fear to deploy algorithms that are only based on NN architectures because they lack results explanation (Schlögl et al., 2019). Furthermore, if the task of the AI algorithm is simplified to the collision prediction, it increases its performance. For this reason, an Object Motion Estimator (OME) that utilizes Optical Flow (OF) and that can run on a parallel thread was developed.

The OF is defined as the change of light in the image, e.g., the retina or the camera sensor, associated with the motion of the scene relative to the eyeball or the camera. In a bio-inspired sense, shifts in the light captured by the retina result in a movement perception of the objects projected onto the retina. In the technical context of computer vision, a set of video frames contain the observer's movement and the environment combined.

There are three frames with a face in motion on figure 5.12, which are differentiated by

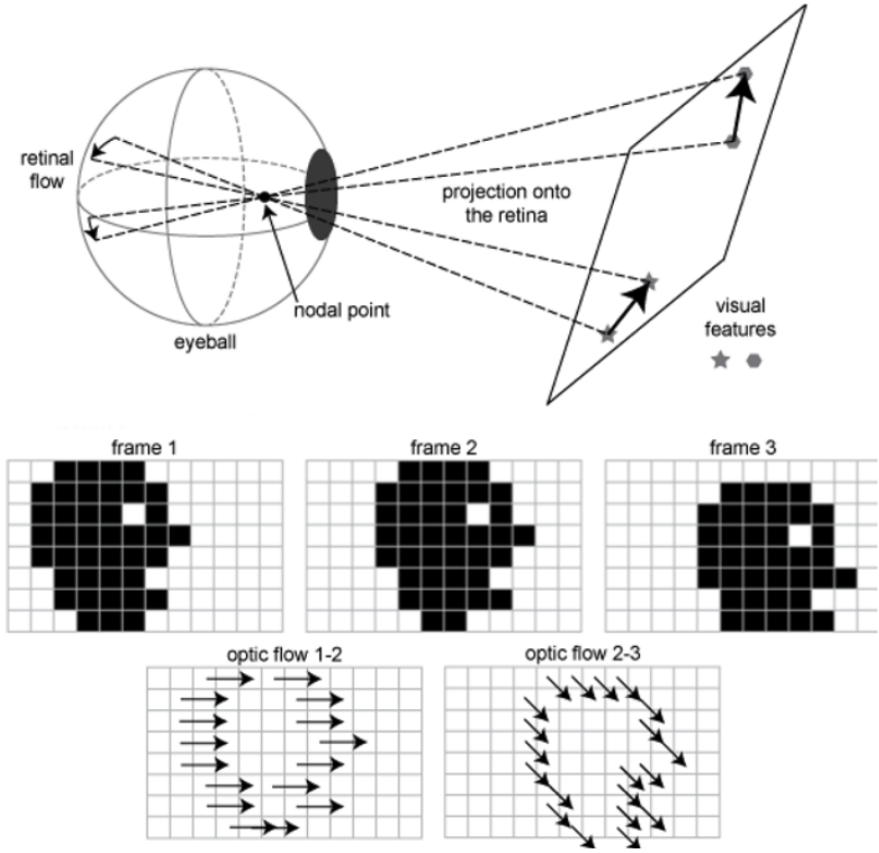


Figure 5.12: Bio-inspired representation of the Optical Flow (Raudies, 2013).

spatial and temporal analysis of the light source captured by the camera. Computing the OF captures the changes to these frames via a vector field. Using the first and the second frames, it is possible to compute the OF 1-2, capturing the pixel movements between these frames. In principle, OF pixel algorithms in the first image look for a neighboring pixel in the second image with the same brightness.

The OF methods calculate the motion at every pixel position between two image frames, which are taken at times t and $t + \Delta t$. These methods are called differential since they are based on local Taylor series approximations of the image values (Horn and Schunck, 1981). The following intensity calculation can be given as follows (equation 5.11):

$$I(x,y,t) = I(x + \Delta x,\ y + \Delta y,\ t + \Delta t) \tag{5.11}$$

where Δx, Δy and Δt are the motion vectors between the two image frames, (x,y,t) is the pixel location at a given time and I(x,y,t) is the pixel intensity.

Assuming the movement is minimal, the I(x,y,t) image constraint with the Taylor series can be developed to:

$$I(x + \Delta x,\ y + \Delta y,\ t + \Delta t) = I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t \tag{5.12}$$

Equation 5.12 provides the following results:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0 \tag{5.13}$$

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \tag{5.14}$$

where V_x and V_y are the x and y velocity components or OF of $I(x,y,t)$ and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are the derivatives of the image at (x,y,t) in the corresponding directions. By applying equation 5.14 and replacing the image derivatives by I_x, I_y and I_t, it is possible to obtain the equation 5.15:

$$I_x V_x + I_y V_y = -I_t \tag{5.15}$$

This is an equation with two variables (V_x,V_y), therefore it cannot be solved. This is known as the OF algorithm aperture problem (Raudies, 2013) and can be seen in Figure 5.13.

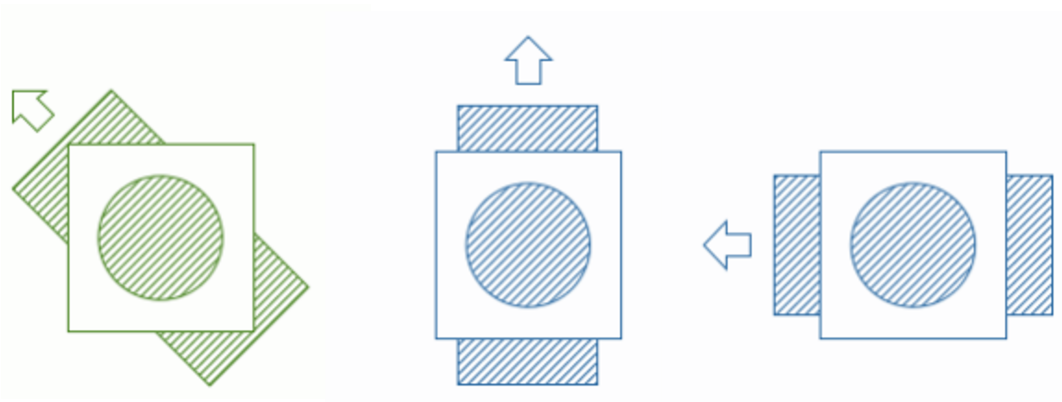


Figure 5.13: Optical Flow Aperture issue. The observer might experience the same view even if the object is moving in different directions.

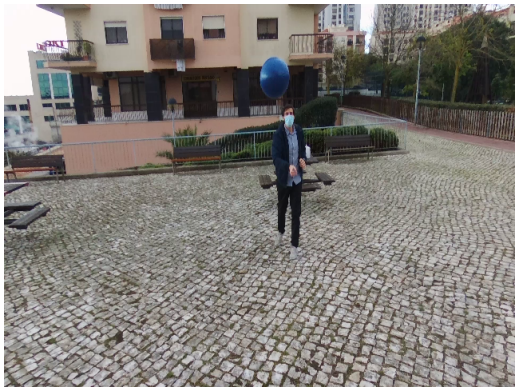
This implies that the OF image cannot be calculated. A further collection of equations is required to find the OF, with additional restrictions. These additional conditions are added by all OF methods for estimating the actual flow. Several algorithms have been further developed, expanding the optical flow capabilities. Some of these techniques are categorized as global methods (Meinhardt-Llopis et al., 2013), local methods (Farnebäck, 2003) and regional matching (Abràmoff et al., 2000):

- Global methods: have the main advantage to output a smooth and regularized flow, using global information that provides accurate time derivatives. On the other hand, they are done by a slow iterative method and have unsharp boundaries.
- Local methods: probably the most used because they are easy and fast to calculate, with accurate time derivatives, which provide the best tradeoff between accuracy

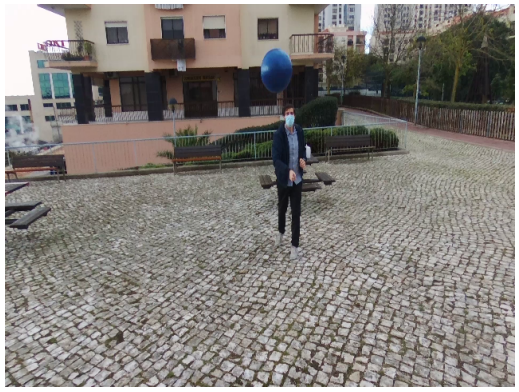
and computational efficiency. Nevertheless, they introduce errors in the boundaries between regions.

- Region-based matching: the fastest of the approaches, but has inaccurate time derivatives.

NVIDIA Turing GPUs include dedicated functions for the OF computing (CUDA libraries to process the algorithm on GPU). It uses sophisticated algorithms to generate highly accurate flow vectors that are robust for frame-to-frame variations in intensity and track true object motion. Computation is significantly faster than other methods with comparable accuracy. The NVIDIA library for the pyramidal version of the Lucas-Kanade method, which computes the optical flow vectors for a sparse feature set, was used to estimate the object's movement. The result of this algorithm on two frames at $t-1$ and t is illustrated at figure 5.14. On the figure 5.14 c, it is possible to observe the magnitude and direction of the flows matrix, each represented by a red arrow.



a. First Image at $t-1$ time.



b. Second Image at t time.

c. Optical Flow between images

Figure 5.14: Optical Flow result from frames $t-1$ and t. The red arrows represent the magnitude and direction of each flow.

Calculating the OF of an image generates a matrix of flows that can be used to estimate the object's flow. Nevertheless, some flows are outliers, and others are tracks of the object and parts of the background that were covered and unveiled. In literature, there are multiple algorithms capable of clustering data. Moreover, none of them is tailored for the concrete case of low-processing, highly variable object flows. For this reason, a novel algorithm that filters and agglomerates flow in groups, outputting an aggregated flow result, to obtain the closest object actual flow is proposed. This technique is entitled as Optical Agglomerated Flow (OAF).

5.3.1 Optical Flow Clustering

In the following subsections the flow vectors representation and normalization of the flow data (5.3.1.1), appropriate distance measures (5.3.1.2), and clustering algorithms used for evaluation (5.3.1.3), are addressed. Finally, different techniques of Optical Flow Clustering (OFC) are compared with the proposed OAF algorithm.

5.3.1.1 Flow Vectors

To obtain the feature space χ, a four-dimensional vector space with N feature vectors $f = \begin{pmatrix} x & y & u & v \end{pmatrix}^T$ is considered, where $p = \begin{pmatrix} x & y \end{pmatrix}^T$ are image pixel location coordinates and $\psi = \begin{pmatrix} u & v \end{pmatrix}^T$ are velocity vectors.

The $\mathfrak{Re}(v)$ is the magnitude of a vector and $\Theta(v) = \angle(v_i, v_j)$ is the angle between any two vectors v_i and v_j, with $1 \leq i, j < N$. The N function vectors are drawn at random from dense optical flow fields obtained for the measurement duration using a real-time variational method recently published Zach et al., 2007. Flow vectors with the smallest magnitude are discarded (by default 10%). Random sampling is used for statistical purposes so that clustering can be processed in milliseconds. We omit time details from the function vectors since the examined video sequences are comparatively small. By subtracting the average and dividing by the standard deviation, we normalize the image position coordinates x and y, and the velocity components u and v.

5.3.1.2 Flow Distances and Dimension reduction

Three distance measures are defined: $D(i, j) := D(f_i, f_j)$ between any two feature vectors, with $1 \leq i, j < N$:

- Euclidian: $D_E(i, j) = \sqrt{\sum_K^k \left(k_i - k_j\right)^2}$
- Manhattan: $D_{Mt}(i, j) = \sum_K^k \left|k_i - k_j\right|$
- Mahanalobis: $D_{Mh}(i, j) = \sum_K^k \left(k_i - k_j\right)^{\Sigma^{-1}} \left(k_i - k_j\right)^T$, where Σ is the covariance matrix between the components of the feature vectors.

Dimension reduction assists in data compression and reduces computation time. It also aids in the removal of some unnecessary functions. Furthermore, it reduces the time needed for clustering computation. For this reason, some dimension reduction techniques were also integrated:

- Isomap is a low-dimensional embedding approach commonly used to compute a quasi-isometric, low-dimensional embedding of a series of high-dimensional data points. Centered on a rough approximation of each data point's neighbors on the manifold, the algorithm provides a straightforward procedure for estimating the intrinsic geometry of a data manifold. Isomap is highly efficient and can be applied to a wide variety of data sources and dimensionalities (Tenenbaum et al., 2000).
- Multidimensional Scaling (MDS) is a technique for displaying the degree of resemblance between particular cases in a dataset. MDS is a method for converting the information about the pairwise 'distances' among a collection of vectors into a structure of points mapped into an abstract Cartesian space (Kruskal, 1964a; Kruskal, 1964b; O'Connell et al., 1999).
- T-distributed Stochastic Neighbor Embedding (t-SNE) is a mathematical method for visualizing high-dimensional data by assigning a position to each data point on a two or three-dimensional map. Its foundation is Stochastic Neighbor Embedding. A nonlinear dimensionality reduction technique is well-suited for embedding high-dimensional data for visualization in a two- or three-dimensional low-dimensional space. It models each high-dimensional object by a two- or three-dimensional point in such a way that identical objects are modeled by neighboring points and dissimilar objects are modeled by distant points with a high probability (Maaten, 2014; Van Der Maaten and Hinton, 2008).

5.3.1.3 Flow Clustering

In order to generate the region of interest of the incoming object, the following clustering methods have been implemented:

- Kmeans is a vector quantization clustering technique that attempts to divide n observations into c clusters, with each observation belonging to the cluster with the closest mean (cluster centers or cluster centroid), which serves as the cluster's prototype (MacQueen, 1967). As a consequence, the data space is partitioned into Voronoi cells (Wu et al., 2007).
- Agglomerative Ward (AW) is an Agglomerative Clustering technique that recursively merges the pair of clusters that minimally increase the wards distance criterion (Müllner, 2011). Ward suggested a general agglomerative hierarchical clustering procedure in which the optimal value of an objective function is used to pick the pair of clusters to merge at each node.

- Agglomerative Average (AA) is a clustering technique that recursively merges pairs of clusters, ordered by the minimum average distance criterion, which is the average of the distances between each observation (Murtagh and Legendre, 2014).

5.3.1.4 Optical Agglomerated Flow

Additionally to the SoAclustering techniques, a novel algorithm, entitled OAF was developed, which is finely tailored for collision detection. To process the OAF , initially it's necessary to calculate the image normalization factor (Equation 5.16), ρ:

$$\rho = W^2 + H^2 \tag{5.16}$$

where W is the image width and H is the image height. Then, the OF matrix is obtained, by computing the flow between $frame_{t-1}$ and $frame_t$. The resulting flows need to be filtered to reduce noise and ensure that only meaningful flows are considered for agglomeration. The value should be normalized by ρ to compare to the flow threshold, ϕ_T. A standard value for ϕ_T is 1%, varying mostly with the camera stabilization (which induces noise). This filtering can be obtained by Equation 5.17.

$$\sqrt{\frac{x_w^2 + y_h^2}{\rho}} \geq \phi_T \tag{5.17}$$

The next step is an iterative procedure. It starts by considering two flows f_0 and f_1, from which it is obtained the current position $P_{r0} = (x_0, y_0)$ and $P_{r1} = (x_1, y_1)$, the flow ending position (x_{0d}, y_{0d}) and (x_{1d}, y_{1d}) , which is $(x_n + f_{nx}, y_n + f_{ny})$. An example of the contemplated flows and positions is illustrated in figure 5.15.

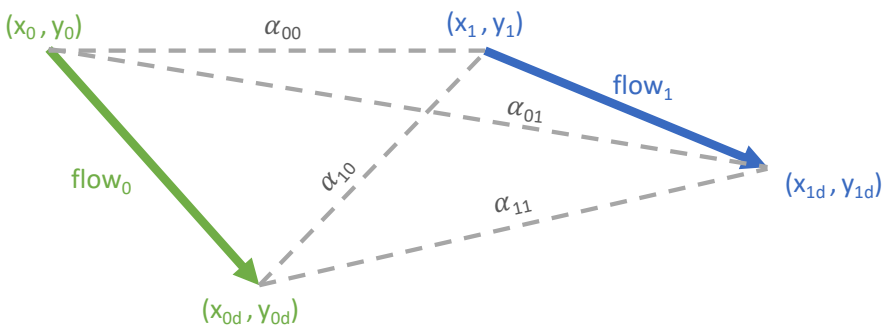


Figure 5.15: The α distances obtained from flows magnitude and directions vectors.

Using this positions, it's possible to calculate the α distances:

$$\begin{bmatrix} \alpha_{00} & \alpha_{01} \\ \alpha_{10} & \alpha_{11} \end{bmatrix} = \begin{bmatrix} \sqrt{\frac{(y_0-y_1)^2+(x_0-x_1)^2}{\rho}} & \sqrt{\frac{(y_0-y_{1d})^2+(x_0-x_{1d})^2}{\rho}} \\ \sqrt{\frac{(y_{0d}-y_1)^2+(x_{0d}-x_1)^2}{\rho}} & \sqrt{\frac{(y_{0d}-y_{1d})^2+(x_{0d}-x_{1d})^2}{\rho}} \end{bmatrix} \tag{5.18}$$

The α distances presented in equation 5.18 are used to verify if two flows can be merged by comparing their values with $\alpha_{threshold}$. If the calculated value is below the $\alpha_{threshold}$, it is a valid flow to be merged. Whenever the values of the flows increase greatly and share the same direction, all the α distances might be larger than the $\alpha_{threshold}$, but still represent a flow from the same object. For this reason, it is important to calculate the distance of the centers of both flows D_c and the radius of the enclosing circumferences R_{fn} (Equations 5.19 and 5.20):

$$D_c = \sqrt{\frac{\left(\left|y_0 - y_{0d}\right| - \left|y_1 - y_{1d}\right|\right)^2 + \left(\left|x_0 - x_{0d}\right| - \left|x_1 - x_{1d}\right|\right)^2}{2}} \tag{5.19}$$

and

$$\begin{cases} R_{f0} = \sqrt{\frac{(y_0 - y_{0d})^2 + (x_0 - x_{0d})^2}{2}} \\ R_{f1} = \sqrt{\frac{(y_1 - y_{1d})^2 + (x_1 - x_{1d})^2}{2}} \end{cases} \tag{5.20}$$

On figure 5.16 is represented the intersection of the enclosing circumferences, which can be verified by the condition $R_{f0} > D_c + R_{f1} or R_{f1} > D_c + R_{f0}$.

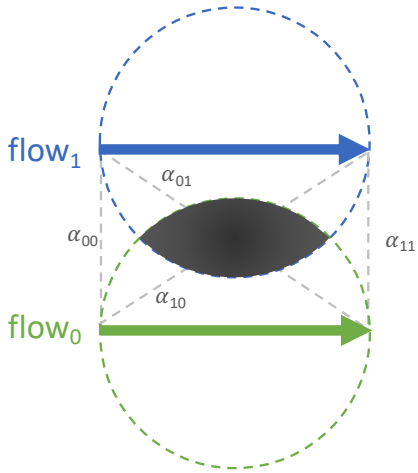


Figure 5.16: Intersection of the enclosing circumferences generated by the obtained flows.

Whenever the α distance or the intersection of the enclosing circumferences is verified, calculate $\{y_{min}, x_{min}; y_{max}, x_{max}\}$ of the considered positions and merge Flows f_0 and f_1. The merge can be obtained by the Equations 5.21 and 5.22:

$$P_{r0}(x,y) = \frac{P_{r0} \cdot v_{f0} + P_{r1} \cdot v_{f1}}{v_{f0} + v_{f1}} \tag{5.21}$$

$$F_{r0}(x,y) = \P_{r1} - \P_{r0} \tag{5.22}$$

Then, the f_1 is removed from the flow list. For a given region group, this process is iterated considering the next P_{r1} the left value of the list. When no flows can be

agglomerated, that flow is stored, and the next two flows are considered. This process is executed through the entire list of flows and only stops whenever no flows are merged in a full search. The final result are regions containing the group of flows and the cumulative flow values at the center of the regions. On figure 5.17 is represented the result of the OAF on the flows processed and previously represented on the figure 5.14.

Figure 5.17: Optical Agglomerated Flow enclosing circumferences.

The output of the OAF are regions that can be considered as moving objects. The incoming colliding object is considered a region with the bigger area (supposedly closer to the camera). For example, in Figure 5.18, the hand of the person throwing the ball towards the UAV has produced a region with flows, which is smaller than the flow produced by the ball, and that needs to be discarded. By the incoming colliding object flow is possible to calculate an escape trajectory v, which is the perpendicular vector $v^{\perp}$, giving preference to rising solutions. Note that a perpendicular 2D vector $v^{\perp}$ always has two solutions, (90° and −90°). For a UAV, it is usually safer to go up; therefore, dodging objects by rising the UAV is considered safer. The OAF algorithm is depicted in pseudo-code in Algorithm 5.2, and a open-source version is made available at `https://github.com/dario-pedro/uav-collision-avoidance`.

5.4 Hybrid Collision Avoidance

The OFC and the collision detection network can be executed in parallel threads, being continuously predicting if there is going to be a collision and estimating the motion of the image objects. The combination of the two techniques is entitled hybrid collision

Listing 5.2: Optical Agglomerated Flow Algorithm.

```
# threshold to filter the flows
flowThreshold = 1

# N value for normalize the flows with image width and image height
N = math.sqrt(pow(w, 2) + pow(h, 2))

# threshold to filter the alpha distances
distanceThreshold = 15

# Callback for Optical Agglomerated Flow function
# it should be called whenever a new frame is obtained
def OAF(frame1, frame2):
  # obtain the optical flow from the 2 frames
  flows = cv2.cuda_OpticalFlow.calc(frame1, frame2)

  # filter meaningful flows
  flows = filterFlows(flows, flowThreshold)

  aggregating = True # control variable
  regions = [] # object regions

  while aggregating: # stop when there is no flows to merge
    aggregating = False

    for i in len(flows)-1:
      for j in len(flows)-1:
        if(i != j): # don't compare with self
          # calculate the flows
          alphas = calculateAlphas(flows[i], flows[j])

          # radius and centers
          rac = calculateRaC(flows[i], flows[j])

          if(validAggregate(alphas, rac, distanceThreshold)):
            # force a full scan
            aggregating = True

            # merge flows into flow[j]
            mergeFlows(flows[j], flows[i)

            if flows[j] not in regions:
              # new region found - append it
              regions.append(flows[j])

            del flows[i] # remove the merged flow
            break # go back to the first for cycle
```

Figure 5.18: Optical Agglomerated Flow with two regions.

avoidance.

Whenever a possible collision is detected, the algorithm can use the latest available escape trajectory to dodge the incoming object. If the processing time of the DCA is greater then one fps, an additional thread must be added, which keeps reading the frames in parallel and updating the frame buffer that is used by the DCA. This thread makes sure that both the DCA and the OFC are using the latest frames, with no lost frames, or variable sequencing.

A pseudo-code example is shown in the algorithm 5.3.

5.5 Challenges

Most of the challenges regarding the models' training were left out from the previous sections for a clear understanding of the results. Nevertheless, it is important to highlight the most significant challenges for those who intend to reutilize and further continue this work. This is usually not presented on papers and represents a big part of the work of a AI engineer.

The first challenge is setting up GPU CUDA libraries and frameworks compatibility. In this work, it was used the development docker version of TensorFlow, and which setup with the following versions:

- Python Version 3.6.9.
- CUDA Version 10.1.243.

Listing 5.3: Hybrid Collision Avoidance Algorithm.

```
# message publisher
reactiveCmdPub = ros.Publisher()

# last known escape vector
escapeVector = {x: 0.0, y: 0.0, z: 0.0}

# Optical Flow Clustering will run in parallel and
# will contantly update the escapeVector var
opticalFlowThread = threading.Thread(target=OFC)

# Callback for Hybrid Collision Avoidance function
# it should be called whenever a new frame is obtained
def hca(videoFrame):

  # Use the DCA algorithm to detect collisions
  if(dcaProcessFrame(videoFrame)):
    reactiveCmdPub.pub(escapeVector)
```

- Tensorflow Version 2.2.0-dev20200311 (which already contains Keras build-in).
- OpenCV Version 4.4.0 (compiled with CUDA)

Another challenge is to manage the models training on GPU, which implies that the data will be temporally on vRAM. The major problem arises when training large models (like VGG or con3D from Facebook). Furthermore, large datasets (such as an image or video dataset), cannot be kept in RAM while the model is training. On this point, solutions such as the data-generators should be used to fetch data, modify it if required, and then present it to the model.

The lack of data for a given problem is an issue transversal to multiple AI tasks. The most commonly used solution is Data Augmentation, but it is not a trivial task on video datasets, and there is still much to explore. For example, modifying frame by frame produces transitions that are not natural on the standard videos, and the frameworks' data-generators do not have a solution for video data.

Data selection and train/test splitting are tasks that affect the outcomes immensely and are typically left out of papers. For example, the ColANet dataset is highly biased for the no collision data. This is quite natural, and most of the time, a typical UAV flies freely without colliding with objects. Feeding this data directly to a model will make it believe that outputting 'no collision' is the best solution, as it will have top performance. Using this model in the real world would be completely useless. Some solutions with sci-learn tend to influence the loss in biased datasets, but the results are worse than a balanced dataset.

Choosing the proper configuration of Dropout, Learning Rate, Learning Decay, Batch Normalization, and optimizers is still a trial and error process. For example, if the network has a low dropout, for being a small dataset, the LSTM units will decorate the dataset. On the other hand, if a high value of Dropout is considered, it will have problems memorizing, never achieving a satisfactory result.

Chapter 6

Applications and Results

The modules presented in the previous sections can be integrated and combined in different ways, depending on the use case. Some of the results were presented alongside the algorithm's description for a better comprehension of the work. Nevertheless, in this chapter, the deployment of the architecture running the DCA for the incoming ball scenario is explored. Initially, all the solution is validated in a simulation environment. Afterward, two setups are tested: on an external ground station sending commands to a Parrot Bebop 2, and directly on the OBC of a HEIFU drone (Pedro et al., 2021). For the simulation environment, a sample dataset, with around 60 videos generated inside the simulation environment (Gazebo), was set up to test the DCA in a controlled environment, working inside the FFAU. Afterwards, the BallNet dataset was used to train and test the DCA on the live scenario.

6.1 Simulation

The simulation provides an environment that allows testing the FFAU, with a focus on the DCA, without the risk of losing an aircraft or colliding with an obstacle. The selected simulator was Gazebo due to the integration with ROS, and it is the capability to reproduce events in similar timings as they would occur on live tests.

The object textures and the background environments on Gazebo are elementary when compared to the real world. For this reason, the first step was to fine-tune the model with a small dataset that was created just for the simulation tests. This dataset consists of 60 videos with incoming thrown balls. In addition, a ROS node was implemented, which spawns objects with an oblique throw movement, which is helpful to simulate random thrown objects to the drone.

A pre-trained model was fine-tuned with this dataset during three epochs. Training

for longer quickly overfits the model because of the small number of videos and the lack of pixel differences between the videos and between the frames of each video.

Afterwards, a simple world was created, where the ball spawned and was thrown to the UAV from different angles. A representation of the results outputted by the NNP and OAF and represented on Figures 6.1 and 6.2 respectively.

a. Normal/Free flight frame (no collision label).

b. Incoming collision frame (collision label)

Figure 6.1: Neural Network Pipeline detecting an incoming ball on the Gazebo simulator.

In around 95% of the test, the UAV dodged the ball with success, and it was surprising

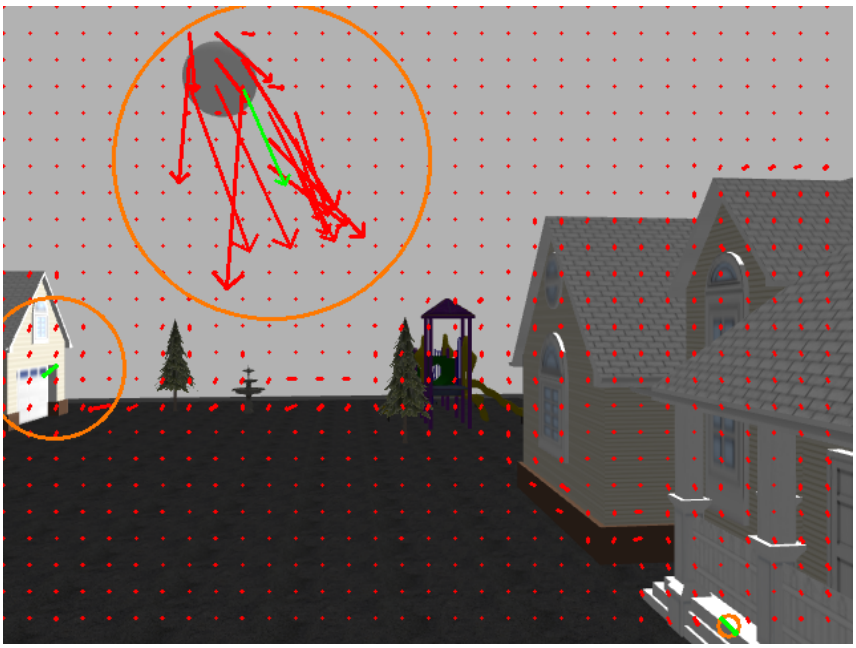


Figure 6.2: Optical Agglomerated Flow result, being executed after a collision was detect by the Neural Network Pipeline.

that it was capable of dodge new objects (e.q. a thrown car) that were not present on the training data. This setup allows validating the processing pipelines of the algorithm and the behavior of the UAV on the request of an avoidance reaction. Furthermore, it proved that fine-tuning the algorithm to a new set of images allows it to learn the new environment and be executed on it.

6.2 Real Application

After the algorithm validation on the simulation, the solution was tested in a similar situation in the real world. For this, the scenario of a human throwing a ball to the UAV was selected because it is one of the few scenarios that other researchers are also conducted tests on (Falanga et al., 2019).

An aspect that was perceived while integrating the algorithm on the UAV was the frame rate variability. The DCA was trained with a constant frame rate, and for that reason, an oscillating frame rate would bring an additional parameter. For example, a video with half the frame rate was shown to the algorithm, and the performance dropped approximately 30%. To solve this issue, an additional feature was implemented and added to the training process of the RNN of the DCA. While constructing the sequences of feature vectors, some frames were dropped with a given frequency, constructing sequences with variable frame rates. This presents the variability to the algorithm and also augments the dataset. Surprisingly, this technique maintained the original dataset frame rate performance while dealing with frame rate oscillation and achieving the same performance on a video with half the frame rate as one of the original frame rates.

On figure 6.3 (a) a frame processed by the algorithm is represented, where the algorithm outputted 'No Collision', and on the figure 6.3(b) detected a 'Collision'.

a. Normal/Free flight frame (no collision label).

b. Incoming collision frame (collision label)

Figure 6.3: Neural Network Pipeline detecting if there is an incoming collision.

The avoidance of a thrown ball was tested to validate the algorithm in a similar use case to previous research works (Falanga et al., 2019). A Parrot Bebop 2 was connected to

a Legion with a 2060 GPU, running the proposed framework.

At first, the trained NNP weights were used, but the results were worse than expected. After some troubleshooting, the authors realized that a constant framerate of approximately 29 fps was obtained by using the recorded videos. However, when working with the live streaming video from the UAV, a high framerate variance was experienced, oscillating between 5 to 30 fps. Also, the compression algorithm used by the Parrot Bebop 2 in livestream mode reduces the video quality, creating another significant difference compared to the trained NNP. Finally, the transmission delay is also an issue, and, for simplification, it will be left out of this paper.

To solve this issue, a set of image augmentation techniques were applied to the dataset. First, it randomly exposed the model to videos at variable framerate (dropping frames) and compressed frames (constant within the video, variable per epoch). Then, it applied the most traditional augmentations, such as rotation, translation, and zoom (yet again, constant per video). After training and deploying this new model, normal behavior was obtained. The NNP result is far from perfect, but this is because the score is measured at the frame level. On the testing results, the NNP often detects a collision a few frames before or after the ideal moment annotated on the dataset, lowering the score. However, this is not critical, as long as the detection is obtained at a moment closed enough for the dodge routine.

Some of the latest UAVs in the market (e.g., Skydio 2, HEIFU (Metz, 2016; Pedro et al., 2021)) already have NVIDIA Single Board Computer (SBC)s, being capable of running such algorithms directly on the aircraft, therefore being capable of running the proposed architecture directly on the UAV. The NNP and OME were integrated with an NVIDIA Jetson Nano used by HEIFU) to run the entire algorithm pipeline, which took an average of 0.18 s, demonstrating that it is a viable option for SoA UAVs.

In order to study the proposed OME algorithm, eight frames from eight different videos (a total of 64 frames) were selected. Figure 6.4 illustrates these frames, which were manually annotated with a red mask on the ball, allowing the creation of a ground truth mask filtering by the red color. As a result, the NNP were correctly outputting collision for all the selected frames; therefore, it is possible to evaluate the performance of the OME algorithm (the most critical part of the dodging trajectory estimation).

The OME is capable of calculating escape trajectories for the closest detected object. For each frame illustrated in Figure 6.4, the previous frame and the current frame were fed into the OME algorithm that outputted a region for the incoming object. Using this output and the ground truth mask GT, it is possible to calculate the True Positive (TP) and False Positive (FP) (normalized by the object size of the object $\sum GT_f$) areas. For a given frame f and a different OME algorithm i, where & is the bitwise AND between matrix, it can calculated by Equation (6.1):

Figure 6.4: Set of 64 ground truth frames with collisions for the OME results discussion.

$$\begin{cases} TP_{fi} = \frac{GT_f \ \& \ OME_{fi}}{\sum GT_f} \\ FP_{fi} = \frac{\overline{GT_f} \ \& \ OME_{fi}}{\sum GT_f} \end{cases}. \tag{6.1}$$

In Figure 6.5, it is possible to observe the TP for each frame that is under analysis. The top-5 performing algorithms were picked for better visualization. The OAF and the Agglomerative algorithms without dimension reduction were the ones that achieved the higher results. Moreover, Figure 6.6 is depicted the processing time in milliseconds of the algorithms per frame. The application of dimension reduction does not reduce processing time, according to the results depicted in Figure 6.6. This is most likely due to the low complexity of the applied clustering algorithms. Thus, the OFC appears as a trade-off between accuracy and processing time.

Just analyzing the TP results might be misleading because an algorithm might be detecting bigger regions, which always encapsulate the incoming object and, therefore, outperform other algorithms. On the other hand, when analyzing the FP, some algorithms detect a bigger area around the object, which intuitively is justified by the object's movement, which generates a flow vector between the previous and the current frame. The analyzed metrics should not penalize these algorithms because, ultimately, the goal is to estimate the incoming object correctly. For this reason, a new metric entitled $FPPerformance$ is introduced. The $FPPerformance$ takes into consideration the impact

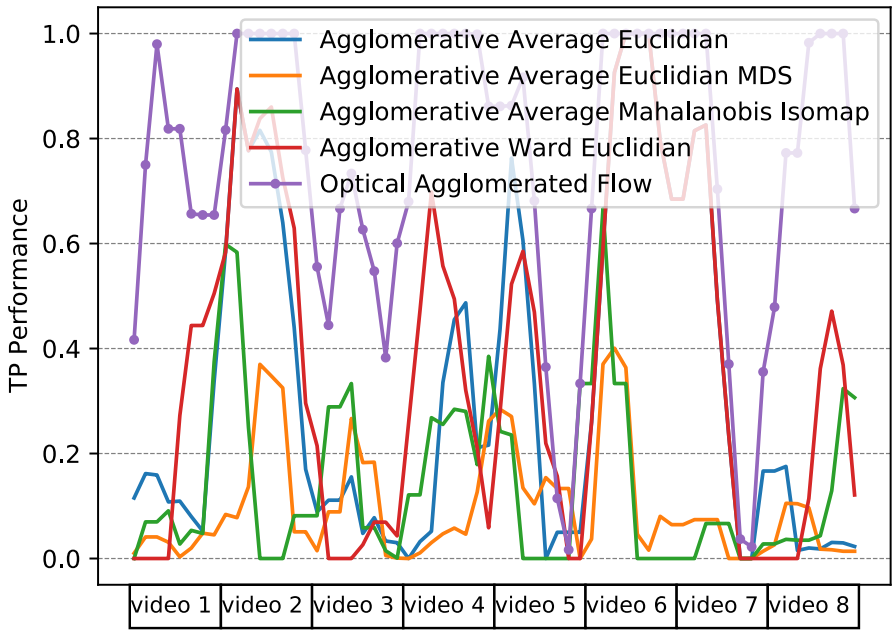


Figure 6.5: Graph analyses of the TP results in the selected video frames.

of the error in the decision. For this, after computing the $\overline{GT_f}$ & OME_{fi}, for each point of the resulting mask, it calculates the distance to the nearest point on the object. This new matrix of distances is named FD. Afterward, the $FP_{Performance}$ can be calculated by:

$$FP_{Performance} = \frac{\sum FD_{fi}}{\sum GT_f \ \& \ OME_{fi}}. \tag{6.2}$$

Using Equation 6.2, it is possible to plot the results in Figure 6.7, which illustrate the FP Performance on the multiple frames. The OAF generates very few points outside the region of the incoming object; therefore, it is always below 0.15 FP Performance, which is a desirable threshold. Values above this threshold might lead to agglomeration of other objects and, therefore, a miss perception of another direction, which could lead to a wrong escape trajectory.

All the results have been summarized in Table 6.1, which is ordered by decreasing mean FP performance. The TP, FP, and the processing time are presented as the mean of all frames. Furthermore, the Root Mean Square Error (RMSE), and the min/max FP performance results are also presented. It is possible to conclude that for the object trajectory estimation, the proposed solution is capable of solving the problem with approximately 2% error. When the algorithm is running live, with continuous frames being fed to the algorithm, the error should statistically decrease because it was measured per frame (Huang

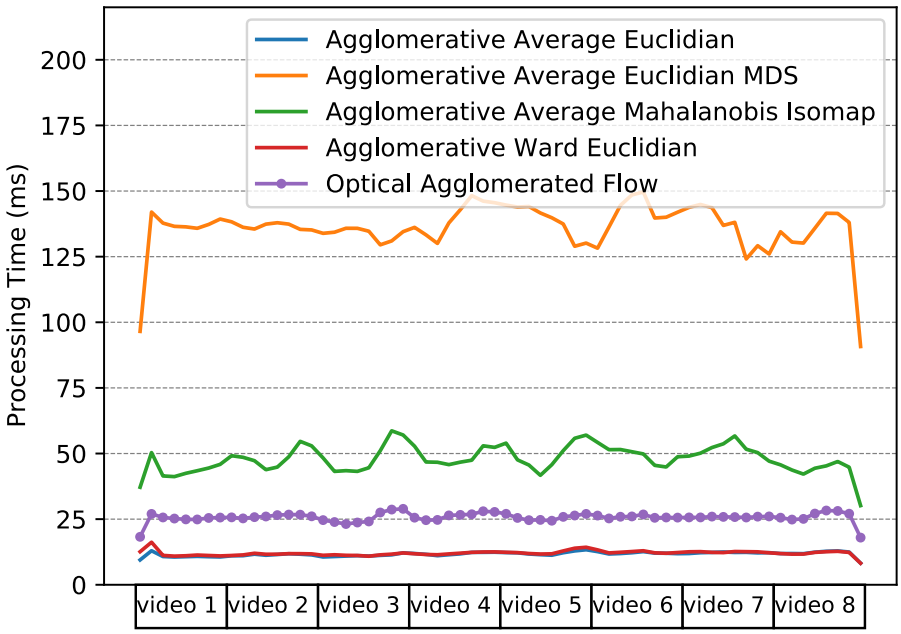


Figure 6.6: Graph analyses of the processing time in the selected video frames.

et al., 1997; Kazemi et al., 2021). Further studies are required to analyze the performance impact with the speed variability of the incoming object, the speed variability of the observer, especially angular movements that might induce false trajectories (Wang and Lin, 2002). In addition, environments with multiple moving objects require some attention, as might be the case in most crowded cities.

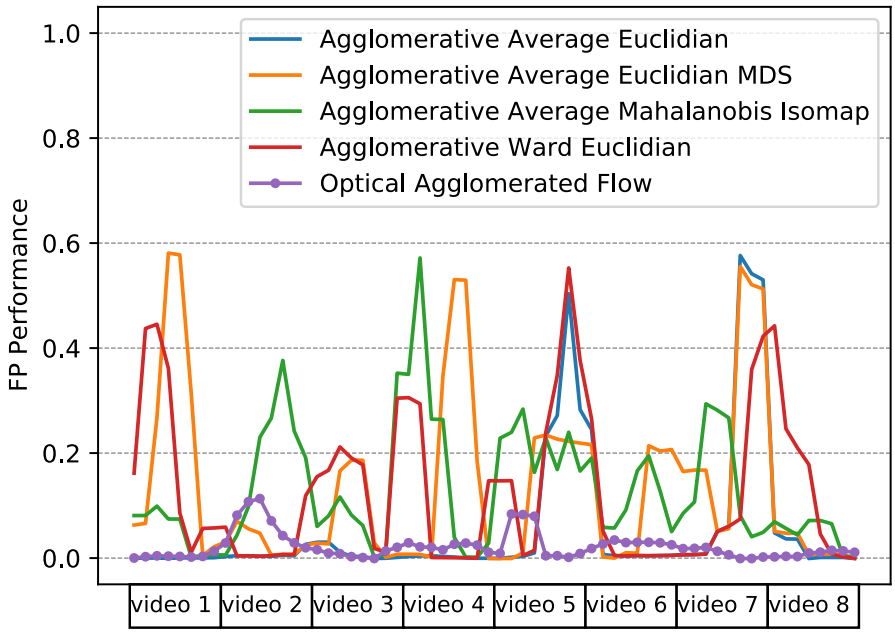


Figure 6.7: Graph analyses of the FP Performance in the selected video frames.

Table 6.1: Results comparison of the OME algorithm in the selected and annotated 64 frames.

Algorithm	Distance Metric	Dimension Reduction	Mean TP	Mean FP	RMSE	FP Perf. (Min / Mean / Max)	Mean Time (ms)
OFC Agglomeration	- -	- -	0.76	3.40	0.50	0.00 / 0.02 / 0.06	25.99
AA (Murtagh and Legendre, 2014)	Euclidian	- -	0.33	0.96	0.43	0.00 / 0.06 / 0.22	11.84
AA (Murtagh and Legendre, 2014)	Manhattan	- -	0.33	0.96	0.43	0.00 / 0.06 / 0.22	8.69
AA (Murtagh and Legendre, 2014)	Mahalanobis	- -	0.33	1.29	0.46	0.00 / 0.10 / 0.37	33.67
AW (Müllner, 2011)	Euclidian	- -	0.37	1.82	0.47	0.00 / 0.13 / 0.27	12.23
AW (Müllner, 2011)	Manhattan	- -	0.37	1.82	0.47	0.00 / 0.13 / 0.27	9.11
AA (Murtagh and Legendre, 2014)	Mahalanobis	Isomap (Tenenbaum et al., 2000)	0.14	2.03	0.41	0.05 / 0.13 / 0.24	48.51
AA (Murtagh and Legendre, 2014)	Euclidian	MDS (Kruskal, 1964b)	0.10	1.43	0.37	0.02 / 0.14 / 0.28	137.63
AA (Murtagh and Legendre, 2014)	Manhattan	MDS (Kruskal, 1964b)	0.13	1.36	0.39	0.01 / 0.14 / 0.37	138.84
Kmeans (MacQueen, 1967)	Manhattan	- -	0.43	2.43	0.48	0.01 / 0.15 / 0.35	25.01
AA (Murtagh and Legendre, 2014)	Euclidian	Isomap (Tenenbaum et al., 2000)	0.09	2.18	0.48	0.06 / 0.16 / 0.24	26.11
AA (Murtagh and Legendre, 2014)	Manhattan	Isomap (Tenenbaum et al., 2000)	0.09	2.18	0.48	0.06 / 0.16 / 0.24	21.95
Kmeans (MacQueen, 1967)	Euclidian	- -	0.41	2.58	0.48	0.01 / 0.16 / 0.45	29.10
AA (Murtagh and Legendre, 2014)	Mahalanobis	MDS (Kruskal, 1964b)	0.13	1.81	0.39	0.02 / 0.17 / 0.48	159.93
AW (Müllner, 2011)	Mahalanobis	- -	0.33	2.49	0.46	0.00 / 0.18 / 0.48	34.07
Kmeans (MacQueen, 1967)	Mahalanobis	- -	0.38	3.08	0.47	0.01 / 0.19 / 0.50	49.76
Kmeans (MacQueen, 1967)	Mahalanobis	Isomap (Tenenbaum et al., 2000)	0.18	4.67	0.44	0.07 / 0.23 / 0.41	64.07
AW (Müllner, 2011)	Mahalanobis	Isomap (Tenenbaum et al., 2000)	0.18	3.91	0.46	0.07 / 0.24 / 0.40	48.91
AA (Murtagh and Legendre, 2014)	Mahalanobis	t-SNE (Van Der Maaten and Hinton, 2008)	0.16	2.92	0.42	0.06 / 0.24 / 0.37	1 057.79
AW (Müllner, 2011)	Mahalanobis	t-SNE (Van Der Maaten and Hinton, 2008)	0.17	3.15	0.43	0.06 / 0.25 / 0.39	1 058.15
Kmeans (MacQueen, 1967)	Mahalanobis	t-SNE (Van Der Maaten and Hinton, 2008)	0.19	3.51	0.43	0.06 / 0.26 / 0.36	1 075.77
AA (Murtagh and Legendre, 2014)	Euclidian	t-SNE (Van Der Maaten and Hinton, 2008)	0.13	2.97	0.45	0.12 / 0.29 / 0.43	1 017.39
AA (Murtagh and Legendre, 2014)	Manhattan	t-SNE (Van Der Maaten and Hinton, 2008)	0.13	2.97	0.45	0.12 / 0.29 / 0.43	1 039.02
Kmeans (MacQueen, 1967)	Euclidian	Isomap (Tenenbaum et al., 2000)	0.10	4.37	0.49	0.07 / 0.29 / 0.44	41.11
Kmeans (MacQueen, 1967)	Euclidian	t-SNE (Van Der Maaten and Hinton, 2008)	0.13	3.43	0.45	0.10 / 0.30 / 0.47	1 034.63
Kmeans (MacQueen, 1967)	Manhattan	Isomap (Tenenbaum et al., 2000)	0.08	4.70	0.47	0.07 / 0.31 / 0.49	36.56
Kmeans (MacQueen, 1967)	Manhattan	t-SNE (Van Der Maaten and Hinton, 2008)	0.13	3.43	0.46	0.10 / 0.31 / 0.47	1 056.44
AW (Müllner, 2011)	Euclidian	t-SNE (Van Der Maaten and Hinton, 2008)	0.11	3.31	0.47	0.09 / 0.32 / 0.50	1 017.83
AW (Müllner, 2011)	Manhattan	t-SNE (Van Der Maaten and Hinton, 2008)	0.11	3.31	0.47	0.09 / 0.32 / 0.50	1 039.30
AW (Müllner, 2011)	Euclidian	Isomap (Tenenbaum et al., 2000)	0.11	4.97	0.49	0.08 / 0.33 / 0.52	26.52
AW (Müllner, 2011)	Manhattan	Isomap (Tenenbaum et al., 2000)	0.11	4.97	0.49	0.08 / 0.33 / 0.52	22.40
AW (Müllner, 2011)	Manhattan	MDS (Kruskal, 1964b)	0.06	4.15	0.44	0.20 / 0.42 / 0.63	139.31
AW (Müllner, 2011)	Euclidian	MDS (Kruskal, 1964b)	0.07	4.77	0.45	0.21 / 0.45 / 0.90	138.09
AW (Müllner, 2011)	Mahalanobis	MDS (Kruskal, 1964b)	0.08	4.92	0.50	0.17 / 0.52 / 0.99	160.31
Kmeans (MacQueen, 1967)	Manhattan	MDS (Kruskal, 1964b)	0.03	6.08	0.42	0.36 / 0.54 / 0.78	158.46
Kmeans (MacQueen, 1967)	Mahalanobis	MDS (Kruskal, 1964b)	0.04	5.84	0.44	0.24 / 0.54 / 0.82	179.07
Kmeans (MacQueen, 1967)	Euclidian	MDS (Kruskal, 1964b)	0.03	6.33	0.42	0.30 / 0.58 / 1,00	158.76

CHAPTER 7

Conclusions and Future Work

This chapter summarizes the main contributions of this thesis. It also proposes some guidelines for future research on the theme, highlighting the points where the developed application can be improved.

7.1 Conclusions

This thesis focused on a framework for handling the behavior of individually or networked unmanned aerial vehicles, ensuring reliability and safety in the flight, regardless of the environmental conditions and unexpected events. In addition, the work explored in-depth the collision avoidance task with deep learning techniques.

In an initial phase, a research process was carried out related to autonomous vehicles, the different types of collision avoidance, and the most well-known Artificial Intelligence algorithms in the literature. A comparative analysis was always carried out regarding the performance of the presented deep learning algorithms. For the training of the envisioned algorithms, two datasets were proposed, namely the ColANet and the BallNet. These datasets were made available as open-source datasets to accelerate and facilitate the development of new collision avoidance algorithms.

Furthermore, a safer architecture for UAVs' navigation was presented. The core elements are implemented on top of ROS framework and make use of its communication mechanism to implement all the framework connections. Additionally, there were several modules developed that can be adjusted and reused to fit new purposes.

The main innovation added to this architecture features a block responsible for collision avoidance with dynamic objects (such as a thrown ball). This block uses a NNP composed of a CNN for feature extraction, a RNN for temporal analyses, and a FNN for outputting the detection. Different models of CNN were explored, and from that

inspection, the best trade-off between performance and computation resources lead to the selection of the MobileNetV2 as the ideal model used in practical tests. A similar process was conducted for the RNN, testing multiple combinations of LSTMs in deep and width.

Whenever this NNP outputs an incoming collision, an escape vector is calculated by a OME algorithm running in parallel. The OME uses the optical flow between the previous and the current frame and a clustering algorithm to estimate the trajectory of the incoming object. A novel OF clustering algorithm for this use-case was introduced, which was named OFC, that outperforms the state-of-art techniques.

The NNP can be thought of as a combination of different NN from a theoretical point of view. But in practice, it's a black box that receives input images and outputs data regarding collisions. To understand a little bit more what's going on inside the NNP, the Grad-CAM was adapted to be executed on this model. This allowed the perception of which areas of the network's are relevant for the collision classification. Also, a visual representation of the influence of fine-tuning a network was obtained. The Grad-CAM output $9x9$ matrixes with normalized weights from 0 to 1, which were rescaled to the input image size, equalized to the output values, and produced a superimposed visualization.

All the modules presented in the framework for remote and fully autonomous UAVs were developed and integrated, connected to a cloud-based platform, named *beXStream*. The proposed architecture enabled remote control of UAVs via Internet, in a portable, extensible, open-source platform that manages containerized workloads and services, which facilitated scalability, configuration, and automation. Furthermore, the actions were sent through the platform. At the same time, safety was enforced by the FFAU, providing constant feedback and telemetry to the pilot, which could take control of the autonomous UAVs when desired.

To train the NNP, the BallNet dataset with videos of subjects throwing balls at a UAV was used. The videos were annotated and converted into several images. The NNP demonstrated an on-time detection, which allows the UAV to estimate a trajectory and to dodge the incoming ball. Both the results on the NNP and the OME demonstrated promising results, achieving 9% error on frame collision detection (multiple consecutive frames drop this percentage) by the NNP, and approximately 2% error on the trajectory estimation by the OME. The tackled use case is just an introduction to the capabilities of the proposed technique, as it is only a scenario that consists of a thrown ball and presents a dataset for that purpose. The NNP knowledge can also be transferred to other scenarios with the enlargement of the dataset.

Furthermore, this solution only requires a simple monocular camera, which can be found in most commercial UAVs. The benefits of using these cameras are their small size, reduced weight, lower power consumption, flexibility, and mounting simplicity. On the other hand, they depend highly on weather conditions and might lack image clarity depending on the background color contrast.

Regarding the proposed algorithms, the drawbacks identified are the processing requirements of the NNP, which are still not available in most out-of-shelf UAVs. On the professional categories, it is still a significant amount of computational processing and power consumption. In addition, the OME might be accurate in processing the object trajectory, but any minor error can compromise the dodging maneuver. Finally, fast reactions might be dangerous if flying in cluttered environments or if the UAV has considerable dimensions.

The proposed approach can be applied to standard UAVs using regular video sensors compared with the current state-of-art. Even though in this work, only the collision of an incoming thrown ball was deeply put to the test, the author believes that the algorithm can be easily adapted to multiple use-cases (with static or dynamic obstacles) by increasing the dataset scenarios. The solutions in the literature are, in comparison, harder to apply or unable to handle fast-moving objects.

7.2 Future Work

During this thesis, multiple dead ends were encountered. For these, possible paths to be explored were envisioned. For this reason, this work can be further improved with updates on the modules presented, possibly even entering new areas of research. The list below summarizes some of the key innovative ideas that will drive future work:

- Optimized DCA. The DCA module can be explored in greater depth, as this area still has many unsolved problems. The dataset must be enlarged, and the proposed algorithm needs to be optimized to run faster. The concept can be optimized by exploring different feature extractors, variations on the sequence size with which the RNN runs and different types of RNN;
- Testing the DCA algorithm on real UAVs in autonomous missions;
- Framework modules variants. Different implementations of the proposed framework should be developed, allowing a performance evaluation and comparison.
- Spatio-temporal Grad-CAM. How would it be a good temporal thermal feature visualization?
- Optical Flow with depth estimation (using a depth camera), allowing the estimation of the distance to the object, therefore adjusting the escape speed and facilitating the selection of the nearest object.
- CUDA implementation of the OFC algorithm to speed up computation time.
- Live tests on HEIFU hexacopters with the algorithms taking advantage of the onboard GPU.

- Edge multi-tenant computing. Whenever a UAV is flying in a different country, the server should be instantiated in the vicinities, minimizing the communication delay;

Bibliography

Abràmoff, M. D., W. J. Niessen, and M. A. Viergever (2000). "Objective quantification of the motion of soft tissues in the orbit." In: *IEEE Transactions on Medical Imaging*. ISSN: 02780062. DOI: 10.1109/42.887614.

Acuña, P. (2016). "Kubernetes." In: *Deploying Rails with Docker, Kubernetes and ECS*. Apress. DOI: 10.1007/978-1-4842-2415-1_3.

Ahmad, J., H. Farman, and Z. Jan (2019). "Deep Learning Methods and Applications." In: *SpringerBriefs in Computer Science*. DOI: 10.1007/978-981-13-3459-7_3.

Aizenberg, I. N., N. N. Aizenberg, and J. Vandewalle (2001). "Multi-Valued and Universal Binary Neurons: Theory, Learning, and Applications." In: *IEEE Transactions on Neural Networks*. ISSN: 19410093. DOI: 10.1109/TNN.2001.925572.

Akyildiz, I. F., T. Melodia, and K. R. Chowdury (2007). *Wireless multimedia sensor networks: A survey*. DOI: 10.1109/MWC.2007.4407225.

Amazon.com Inc. (2015). "Determining Safe Access with a Best- Equipped, Best-Served Model for Small Unmanned Aircraft Systems." In: *NASA UTM 2015: The Next Era of Aviation*.

Amirante, A., T. Castaldi, L. Miniero, and S. P. Romano (2014). "Janus: A general purpose WebRTC gateway." In: *Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications, IPTComm 2014*. ISBN: 9781450321242. DOI: 10.1145/2670386.2670389.

Andrew, A. M. (2001). "Multiple View Geometry in Computer Vision." In: *Kybernetes*. ISSN: 0368492X. DOI: 10.1016/S0143-8166(01)00145-2.

Ardupilot (2013). *ArduCopter - Open Source Flight Controller*. Online; accessed 2 February 2022. URL: https://ardupilot.org/.

Arel, I., D. Rose, and T. Karnowski (2010). "Deep machine learning-A new frontier in artificial intelligence research." In: *IEEE Computational Intelligence Magazine*. ISSN: 1556603X. DOI: 10.1109/MCI.2010.938364.

Ayodele, T. O. (2010). "Types of Machine Learning Algorithms." In: *New Advances in Machine Learning*. ISSN: 978-953-307-034-6. DOI: 10.5772/56672.

Azevedo, F., A. Oliveira, A. Dias, J. Almeida, M. Moreira, T. Santos, A. Ferreira, A. Martins, and E. Silva (2017). "Collision avoidance for safe structure inspection with multirotor UAV." In: *2017 European Conference on Mobile Robots, ECMR 2017*. ISBN: 9781538610961. DOI: 10.1109/ECMR.2017.8098719.

Baccouche, M., F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt (2011). "Sequential deep learning for human action recognition." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783642254451. DOI: 10.1007/978-3-642-25446-8_4.

BBC (2016). *'Drone' hits British Airways plane approaching Heathrow Airport*. Online; accessed 2 February 2022. URL: https://www.bbc.com/news/uk-36067591.

— (2017). *Drone collides with commercial aeroplane in Canada*. Online; accessed 2 February 2022. URL: https://www.bbc.com/news/technology-41635518.

Bell, S., P. Upchurch, N. Snavely, and K. Bala (2013). "OpenSurfaces." In: *ACM Transactions on Graphics*. ISSN: 07300301. DOI: 10.1145/2461912.2462002.

— (2015). "Material recognition in the wild with the Materials in Context Database." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298970.

Bengio, Yoshua, Courville, Aaron, Vincent, and Pascal (2013). "Representation learning: A review and new perspectives." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. ISSN: 01628828. DOI: 10.1109/TPAMI.2013.50.

Bengio, Y. (2009). "Learning Deep Architectures for AI." In: *Foundations and Trends® in Machine Learning*. ISSN: 1935-8237. DOI: 10.1561/2200000006.

— (2013). "Deep learning of representations: Looking forward." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783642395925. DOI: 10.1007/978-3-642-39593-2_1. arXiv: 1305.0445.

Bengio, Y., P. Simard, and P. Frasconi (1994). "Learning Long-Term Dependencies with Gradient Descent is Difficult." In: *IEEE Transactions on Neural Networks*. ISSN: 19410093. DOI: 10.1109/72.279181.

Berg, A. C., T. L. Berg, H. Daume, J. Dodge, A. Goyal, X. Han, A. Mensch, M. Mitchell, A. Sood, K. Stratos, and K. Yamaguchi (2012). "Understanding and predicting importance in images." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467312264. DOI: 10.1109/CVPR.2012.6248100.

Boiman, O. and M. Irani (2007). "Detecting irregularities in images and in video." In: *International Journal of Computer Vision*. DOI: 10.1007/s11263-006-0009-9.

Boulanger-Lewandowski, N., Y. Bengio, and P. Vincent (2012). "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription." In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*. ISBN: 9781450312851. arXiv: 1206.6392.

Boureau, Y. L., J. Ponce, and Y. Lecun (2010). "A theoretical analysis of feature pooling in visual recognition." In: *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*. ISBN: 9781605589077.

Bouvrie, J. (2006). *Notes on convolutional neural networks*. Online; accessed 2 February 2022. DOI: http://dx.doi.org/10.1016/j.protcy.2014.09.007. arXiv: 1102.0183. URL: http://cogprints.org/5869/.

Brisset, P. and G. Hattenberger (2008). "Multi-UAV Control with the Paparazzi System." In: *Conference on Humans Operating Unmanned Systems (HUMOUS'08)*. hal: hal-00938715.

Brostow, G. J., J. Fauqueur, and R. Cipolla (2009). "Semantic object classes in video: A high-definition ground truth database." In: *Pattern Recognition Letters*. ISSN: 01678655. DOI: 10.1016/j.patrec.2008.04.005.

Brostow, G. J., J. Shotton, J. Fauqueur, and R. Cipolla (2008). "Segmentation and recognition using structure from motion point clouds." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISSN: 03029743. DOI: 10.1007/978-3-540-88682-2-5.

Burgard, W., M. Bennewitz, D. Tipaldi, Spinello, and Luciano (2019). *Techniques for 3D Mapping*. Online; accessed 2 February 2022. URL: http://ais.informatik.uni-freiburg.de/teaching/ss14/robotics/slides/17-3dmapping.pdf.

Canada, C. (2017). *Drone that struck plane near Quebec City airport was breaking the rules | CBC News*. Online; accessed 2 February 2022. URL: http://www.cbc.ca/news/canada/montreal/garneau-airport-drone-quebec-1.4355792.

Canziani, A., A. Paszke, and E. Culurciello (2016). "An Analysis of Deep Neural Network Models for Practical Applications." In: *Computer Vision and Pattern Recognition*. eprint: 1605.07678. URL: http://arxiv.org/abs/1605.07678.

Caron, C. (2017). "After Drone Hits Plane in Canada, New Fears About Air Safety." In: *The New York Times*. URL: https://www.nytimes.com/2017/10/17/world/canada/canada-drone-plane.html.

Caterini, A. L. and D. E. Chang (2018). "Recurrent neural networks." In: *SpringerBriefs in Computer Science*. DOI: 10.1007/978-3-319-75304-1_5. arXiv: 1808.06170.

Cawley, G. C. and N. L. Talbot (2010). "On over-fitting in model selection and subsequent selection bias in performance evaluation." In: *Journal of Machine Learning Research* 11, 2079–2107. ISSN: 15324435.

Chao, H., Y. Cao, and Y. Chen (2010). "Autopilots for small unmanned aerial vehicles: A survey." In: *International Journal of Control, Automation and Systems*. ISSN: 15986446. DOI: 10.1007/s12555-010-0105-z.

Chen, X., A. Golovinskiy, and T. Funkhouser (2009). "A benchmark for 3D mesh segmentation." In: *ACM SIGGRAPH 2009 papers on - SIGGRAPH '09*. ISBN: 9781605587264. DOI: 10.1145/1576246.1531379.

Chitta, S., I. Sucan, and S. Cousins (2012). "MoveIt!" In: *IEEE Robotics and Automation Magazine*. DOI: 10.1109/MRA.2011.2181749.

Chollet, F. (2017). "Xception: Deep learning with depthwise separable convolutions." In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.195. arXiv: 1610.02357.

Christian (Sept. 2016). *Object detection in 3D point clouds*. Ed. by F. U. Berlin. Freie Universitat Berlin.

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio (2014). "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." In: *arXiv*. eprint: 1412.3555 (cs.NE).

Cireşan, D. C., A. Giusti, L. M. Gambardella, and J. Schmidhuber (2012). "Deep neural networks segment neuronal membranes in electron microscopy images." In: *Advances in Neural Information Processing Systems*. ISBN: 9781627480031.

Cireşan, D. C., U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber (2011). "Flexible, high performance convolutional neural networks for image classification." In: *IJCAI International Joint Conference on Artificial Intelligence*. ISBN: 9781577355120. DOI: 10.5591/978-1-57735-516-8/IJCAI11-210.

Cireşan, D. C., U. Meier, L. M. Gambardella, and J. Schmidhuber (2010). "Deep, big, simple neural nets for handwritten digit recognition." In: *IEEE Neural Computation*. Vol. 22. MIT Press, pp. 3207–3220. DOI: 10.1162/NECO_a_00052.

Cloud Native Computing Foundation (2019). *What is Kubernetes - Kubernetes*. https://www.cncf.io/. Online; accessed 2 February 2022.

Combe, T., A. Martin, and R. Di Pietro (2016). "To Docker or Not to Docker: A Security Perspective." In: *IEEE Cloud Computing*. ISSN: 23256095. DOI: 10.1109/MCC.2016.100.

Cordts, M., M. Omran, S. Ramos, T. Scharwächter, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele (2015). "The Cityscapes Dataset." In: *CVPR Workshop on The Future of Datasets in Vision*.

Cortes, C. and V. Vapnik (1995). "Support-Vector Networks." In: *Machine Learning*. ISSN: 15730565. DOI: 10.1023/A:1022627411411.

Csáji, B. C. (2001). "Approximation with artificial neural networks." In: *MSc. thesis on Faculty of Sciences Eötvös Loránd University*. DOI: 10.1.1.101.2647.

Dahl, G. E., T. N. Sainath, and G. E. Hinton (2013). "Improving deep neural networks for LVCSR using rectified linear units and dropout." In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. ISBN: 9781479903566. DOI: 10.1109/ICASSP.2013.6639346.

Dai, A., A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner (2017). "ScanNet: Richly-annotated 3D reconstructions of indoor scenes." In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.261.

Darmatasia and M. I. Fanany (2017). "Handwriting recognition on form document using convolutional neural network and support vector machines (CNN-SVM)." In: *2017 5th International Conference on Information and Communication Technology, ICoIC7 2017*. ISBN: 9781509049127. DOI: 10.1109/ICoICT.2017.8074699.

Darwish, A. (2018). "Bio-inspired computing: Algorithms review, deep analysis, and the scope of applications." In: *Future Computing and Informatics Journal*. ISSN: 23147288. DOI: 10.1016/j.fcij.2018.06.001.

Dauphin, Y. N., H. De Vries, and Y. Bengio (2015). "Equilibrated adaptive learning rates for non-convex optimization." In: *Advances in Neural Information Processing Systems*. arXiv: 1502.04390.

Dauphin, Y. N., A. Fan, M. Auli, and D. Grangier (2017). "Language modeling with gated convolutional networks." In: *34th International Conference on Machine Learning, ICML 2017*. ISBN: 9781510855144. eprint: 1612.08083.

Dave Hershberger, David Gossow, J. F. (2009). *rviz - ROS Wiki*. Online; accessed 2 February 2022. URL: http://wiki.ros.org/ROS.

Davis, J., R. Lambert, J. Davis, and R. Lambert (2014). "Vehicles." In: *Engineering in Emergencies*. DOI: 10.3362/9781780441139.017.

Dechter, R. (1986). "Learning While Searching in Constraint-Satisfaction-Problems." In: *Aaai*.

Delalleau, O. and Y. Bengio (2011). "Shallow vs. deep sum-product networks." In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011*. ISBN: 9781618395993.

Deng, H., G. Runger, and E. Tuv (2011). "Bias of importance measures for multi-valued attributes and solutions." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783642217371. DOI: 10.1007/978-3-642-21738-8_38.

Deng, L. and D. Yu (2013). "Deep Learning: Methods and Applications Foundations and Trends R in Signal Processing." In: *Signal Processing*. ISSN: 1932-8346. DOI: 10.1561/2000000039. arXiv: 1309.1501.

DeVries, T. and G. W. Taylor (2019). "Dataset augmentation in feature space." In: *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*. arXiv: 1702.05538.

Dixon, C. (2016). *What's Next in Computing?* Online; accessed 2 February 2022. URL: https://www.medium.com.

Domingos, P. (2012). "A few useful things to know about machine learning." In: *Communications of the ACM*. DOI: 10.1145/2347736.2347755.

— (2015). *The Master Algorithm*. Ed. by P. B. LTD. 2017th ed. Penguin Books. ISBN: 9780141979243.

Dong, C., C. C. Loy, K. He, and X. Tang (2016). "Image Super-Resolution Using Deep Convolutional Networks." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. ISSN: 01628828. DOI: 10.1109/TPAMI.2015.2439281. eprint: 1501.00092.

Einser, J. (2013). "Deep Learning of Recursive Structure: Grammar Induction." In: *International Conference on Learning Representations 2013*.

Erhan, D., Y. Bengio, A. Courville, and P. Vincent (2009). "Visualizing higher-layer features of a deep network." In: *Technical Report, Univeristé de Montréal*.

Everingham, M., S. M. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman (2014). "The Pascal Visual Object Classes Challenge: A Retrospective." In: *International Journal of Computer Vision*. ISSN: 15731405. DOI: 10.1007/s11263-014-0733-5.

Falanga, D., S. Kim, and D. Scaramuzza (2019). "How Fast Is Too Fast? the Role of Perception Latency in High-Speed Sense and Avoid." In: *IEEE Robotics and Automation Letters*. ISSN: 23773766. DOI: 10.1109/LRA.2019.2898117.

Fang, H., S. Gupta, F. Iandola, R. K. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. C. Platt, C. L. Zitnick, and G. Zweig (2015). "From captions to visual concepts and back." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298754.

Farnebäck, G. (2003). "Two-frame motion estimation based on polynomial expansion." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISSN: 16113349. DOI: 10.1007/3-540-45103-x_50.

Fukushima, K. (1980). "Neocognition: a self." In: *Biol. Cybernetics*. ISBN: 0340-1200. DOI: 10.1007/BF00344251. eprint: arXiv:1011.1669v3.

Fukushima, K. and S. Miyake (1982). "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition." In: DOI: 10.1007/978-3-642-46466-9_18.

Galceran, E. and M. Carreras (2013). "A survey on coverage path planning for robotics." In: *Robotics and Autonomous Systems*. ISSN: 09218890. DOI: 10.1016/j.robot.2013.09.004.

Gallup, D., J. M. Frahm, P. Mordohai, and M. Pollefeys (2008). "Variable baseline/resolution stereo." In: *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. ISBN: 9781424422432. DOI: 10.1109/CVPR.2008.4587671.

Garcia-Garcia, A., S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez (2018). "A survey on deep learning techniques for image and video semantic segmentation." In: *Applied Soft Computing Journal*. ISSN: 15684946. DOI: 10.1016/j.asoc.2018.05.018.

Gardner, M. W. and S. R. Dorling (1998). "Artificial neural networks (the multilayer perceptron) - a review of applications in the atmospheric sciences." In: *Atmospheric Environment*. ISSN: 13522310. DOI: 10.1016/S1352-2310(97)00447-0.

Geiger, A., P. Lenz, C. Stiller, and R. Urtasun (2013). "Vision meets robotics: The KITTI dataset." In: *International Journal of Robotics Research*. ISSN: 02783649. DOI: 10.1177/0278364913491297.

Gent, E. (2017). *AI Is Easy to Fool*. Online; accessed 2 February 2022. URL: https://singularityhub.com/2017/10/10/ai-is-easy-to-fool-why-that-needs-to-change/.

Gers, F. A., N. N. Schraudolph, and J. Schmidhuber (2003). "Learning precise timing with LSTM recurrent networks." In: *Journal of Machine Learning Research*. ISSN: 15324435. DOI: 10.1162/153244303768966139.

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks." In: *Journal of Machine Learning Research*.

Goertzel, B. (2015). "Are there deep reasons underlying the pathologies of today's deep learning algorithms?" In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783319213644. DOI: 10.1007/978-3-319-21365-1_8.

Goglia, J. (2017). *NTSB Finds Drone Pilot At Fault For Midair Collision With Army Helicopter*. Online; accessed 2 February 2022. URL: https://www.forbes.com/sites/johngoglia/2017/12/14/ntsb-finds-drone-pilot-at-fault-for-midair-collision-with-army-helicopter/#22b9b11f7b3f.

Goh, H., N. Thome, M. Cord, and J. H. Lim (2013). "Top-down regularization of deep belief networks." In: *Advances in Neural Information Processing Systems 26*. Ed. by C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger. Lake Tahoe, United States, pp. 1878–1886. URL: https://hal.archives-ouvertes.fr/hal-00947569.

Gomez, F. J. and J. Schmidhuber (2005). "Co-evolving recurrent neurons learn deep memory POMDPs." In: *GECCO '05: Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pp. 491–498. DOI: 10.1145/1068009.1068092.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press. ISBN: 3540620583, 9783540620587. DOI: 10.1016/B978-0-12-391420-0.09987-X. eprint: arXiv:1011.1669v3. URL: http://www.deeplearningbook.org.

Grill-Spector, K., K. S. Weiner, J. Gomez, A. Stigliani, and V. S. Natu (2018). "The functional neuroanatomy of face perception: From brain measurements to deep neural networks." In: *Interface Focus*. ISSN: 20428901. DOI: 10.1098/rsfs.2018.0013.

Group, E. (2018). *Autonomous Underwater Vehicle Auvs Solutions*. Online; accessed 2 February 2022. URL: https://www.ecagroup.com/en/autonomous-underwater-vehicle-auvs-solutions.

Grun, F., C. Rupprecht, N. Navab, and F. Tombari (2016). "A Taxonomy and Library for Visualizing Learned Features in Convolutional Neural Networks." In: *Computer Vision and Pattern Recognition*. eprint: 1606.07757.

Gu, J., Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen (2018). "Recent advances in convolutional neural networks." In: *Pattern Recognition*. ISSN: 00313203. DOI: 10.1016/j.patcog.2017.10.013.

Gu, S. and L. Rigazio (2015). "Towards deep neural network architectures robust to adversarial examples." In: *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*. arXiv: 1412.5068.

Guo, Y., Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew (2016). "Deep learning for visual understanding: A review." In: *Neurocomputing*. ISSN: 18728286. DOI: 10.1016/j.neucom.2015.09.116.

Hackel, T., J. D. Wegner, and K. Schindler (2016). "Contour detection in unstructured 3D point clouds." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.178.

Hamel, P. and D. Eck (2010). "Learning features from music audio with deep belief networks." In: *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010*. ISBN: 9789039353813.

Han, D., J. Kim, and J. Kim (2017). "Deep pyramidal residual networks." In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.668. eprint: 1610.02915.

Han, J., D. Zhang, G. Cheng, N. Liu, and D. Xu (2018a). "Advanced Deep-Learning Techniques for Salient and Category-Specific Object Detection: A Survey." In: *IEEE Signal Processing Magazine*. DOI: 10.1109/MSP.2017.2749125.

Han, S., J. Pool, J. Tran, and W. Dally (2015). "Learning both Weights and Connections for Efficient Neural Network." In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., pp. 1135–1143. URL: http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf.

Han, W., R. Feng, L. Wang, and G. Lang (July 2018b). "Adaptive Spatial-Scale-Aware Deep Convolutional Neural Network for High-Resolution Remote Sensing Imagery Scene Classification." In: pp. 4736–4739. DOI: 10.1109/IGARSS.2018.8518290.

Hanhart, P., Y. He, Y. Ye, J. Boyce, Z. Deng, and L. Xu (2018). "360-Degree Video Quality Evaluation." In: *2018 Picture Coding Symposium, PCS 2018 - Proceedings*. ISBN: 9781538641606. DOI: 10.1109/PCS.2018.8456255.

Hanin, B. and M. Sellke (2017). "Approximating Continuous Functions by ReLU Nets of Minimal Width." In: *arXiv, University of Cambridge*. eprint: 1710.11278 (stat.ML).

Hartmann, K. and K. Giles (2016). "UAV exploitation: A new domain for cyber power." In: *International Conference on Cyber Conflict, CYCON*. ISBN: 9789949954483. DOI: 10.1109/CYCON.2016.7529436.

He, K., X. Zhang, S. Ren, and J. Sun (2015). "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. ISSN: 01628828. DOI: 10.1109/TPAMI.2015.2389824.

— (2016). "Deep residual learning for image recognition." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385.

Hermann, A., F. Drews, J. Bauer, S. Klemm, A. Roennau, and R. Dillmann (2014). "Unified GPU voxel collision detection for mobile manipulation planning." In: ISSN: 21530866. DOI: 10.1109/IROS.2014.6943148.

Hert, S., S. Tiwari, and V. Lumelsky (1996). "A terrain-covering algorithm for an AUV." In: *Autonomous Robots*. ISSN: 09295593. DOI: 10.1007/BF00141150.

Hinton, G. E. (2012). "A practical guide to training restricted boltzmann machines." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISSN: 03029743. DOI: 10.1007/978-3-642-35289-8-32.

Hinton, G. E., S. Osindero, and Y. W. Teh (2006). "A fast learning algorithm for deep belief nets." In: *Neural Computation*. ISSN: 08997667. DOI: 10.1162/neco.2006.18.7.1527.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov (2012). "Improving neural networks by preventing co-adaptation of feature detectors." In: *CoRR* abs/1207.0580. arXiv: 1207.0580 [cs.NE]. URL: http://arxiv.org/abs/1207.0580.

Hochreiter, S. (1998). "The vanishing gradient problem during learning recurrent neural nets and problem solutions." In: *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*. ISSN: 02184885. DOI: 10.1142/S0218488598000094.

Hochreiter, S. and J. Schmidhuber (1997). "Long Short-Term Memory." In: *Neural Computation*. ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735.

Horn, B. K. and B. G. Schunck (1981). "Determining optical flow." In: *Artificial Intelligence*. ISSN: 00043702. DOI: 10.1016/0004-3702(81)90024-2.

Hornung, A., K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard (2013). "OctoMap: An efficient probabilistic 3D mapping framework based on octrees." In: *Autonomous Robots*. ISSN: 09295593. DOI: 10.1007/s10514-012-9321-0.

Hotz, G. (2020). *Comma.ai – Introducing openpilot*. Online; accessed 2 February 2022. URL: https://comma.ai/.

Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam (2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." In: *arXiv*. eprint: 1704.04861 (cs.CV).

Hrabar, S. (2008). "3D path planning and stereo-based obstacle avoidance for rotorcraft UAVs." In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. ISBN: 9781424420582. DOI: 10.1109/IROS.2008.4650775.

— (2011). "Reactive obstacle avoidance for rotorcraft UAVs." In: *IEEE International Conference on Intelligent Robots and Systems*. ISBN: 9781612844541. DOI: 10.1109/IROS.2011.6048312.

— (2012). "An evaluation of stereo and laser-based range sensing for rotorcraft unmanned aerial vehicle obstacle avoidance." In: *Journal of Field Robotics*. ISSN: 15564959. DOI: 10.1002/rob.21404.

Hu, J., L. Shen, and G. Sun (2018a). "Squeeze-and-Excitation Networks." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781538664209. DOI: 10.1109/CVPR.2018.00745. eprint: 1709.01507.

Hu, Y., G. Wen, M. Luo, D. Dai, J. Ma, and Z. Yu (2018b). "Competitive Inner-Imaging Squeeze and Excitation for Residual Network." In: *Computer Vision and Pattern Recognition*. arXiv: 1807.08920 [cs.CV].

Huang, G., Z. Liu, L. Van Der Maaten, and K. Q. Weinberger (2017). "Densely connected convolutional networks." In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.243. arXiv: 1608.06993.

Huang, G., Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger (2016). "Deep Networks with Stochastic Depth." In: *European Conference on Computer Vision*. ISBN: 978-3-319-46493-0.

Huang, H., D. Dabiri, and M. Gharib (1997). "On errors of digital particle image velocimetry." In: *Measurement Science and Technology*. ISSN: 09570233. DOI: 10.1088/0957-0233/8/12/007.

Huang, S., N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel (2019). "Adversarial attacks on neural network policies." In: *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*. arXiv: 1702.02284.

Hubel, D. H. and T. N. Wiesel (1962). "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." In: *The Journal of Physiology*. ISSN: 14697793. DOI: 10.1113/jphysiol.1962.sp006837.

— (1968). "Receptive fields and functional architecture of monkey striate cortex." In: *The Journal of Physiology*. ISSN: 14697793. DOI: 10.1113/jphysiol.1968.sp008455.

Ioffe, S. and C. Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *32nd International Conference on Machine Learning, ICML 2015*. ISBN: 9781510810587. arXiv: 1502.03167.

Ivakhnenko, A. G. (1971). "Polynomial Theory of Complex Systems." In: *IEEE Transactions on Systems, Man and Cybernetics*. ISSN: 21682909. DOI: 10.1109/TSMC.1971.4308320.

Ivakhnenko, A. G. and V. G. Lapa (1965). "Cybernetic predicting devices." In: *CCM Information Corporation*.

Jaderberg, M., K. Simonyan, A. Zisserman, and K. Kavukcuoglu (2015). "Spatial transformer networks." In: *Advances in Neural Information Processing Systems*. arXiv: 1506.02025.

Janczukowicz, E., A. Braud, S. Tuffin, G. Fromentoux, A. Bouabdallah, and J. M. Bonnin (2015). "Specialized network services for WebRTC: TURN-based architecture proposal." In: *Proceedings of the 1st Workshop on All-Web Real-Time Systems, AweS 2015 - In Conjunction with EuroSys 2015*. ISBN: 9781450334778. DOI: 10.1145/2749215.2749218.

Janoch, A., S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell (2011). "A category-level 3-D object dataset: Putting the Kinect to work." In: *Proceedings of the IEEE International Conference on Computer Vision*. ISBN: 9781467300629. DOI: 10.1109/ICCVW.2011.6130382.

Jarrett, K., K. Kavukcuoglu, M. Ranzato, and Y. LeCun (2009). "What is the best multi-stage architecture for object recognition?" In: *Proceedings of the IEEE International*

Conference on Computer Vision. ISBN: 9781424444205. DOI: 10.1109/ICCV.2009.5459469.

Ji, S., W. Xu, M. Yang, and K. Yu (2013). "3D Convolutional neural networks for human action recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. ISSN: 01628828. DOI: 10.1109/TPAMI.2012.59.

Jia, Y., E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell (2014). "Caffe: Convolutional architecture for fast feature embedding." In: *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*. ISBN: 9781450330633. DOI: 10.1145/2647868.2654889. eprint: 1408.5093.

Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "ImageNet: A large-scale hierarchical image database." In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. ISBN: 978-1-4244-3992-8. DOI: 10.1109/CVPRW.2009.5206848.

Jiang, J. and Y. Ma (2020). "Path planning strategies to optimize accuracy, quality, build time and material use in additive manufacturing: A review." In: *Micromachines*. DOI: 10.3390/MI11070633.

Joseph, L. (2015). *Mastering ROS for Robotics Programming*. Ed. by P. Publishing. Packt Publishing. ISBN: 9781783551798. URL: \url{https://www.packtpub.com/hardware-and-creative/mastering-ros-robotics-programming}.

Kalman, B. and S. Kwasny (1992). "Why tanh: choosing a sigmoidal function." In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. Vol. 4, 578–581 vol.4. DOI: 10.1109/IJCNN.1992.227257.

Kalman, R. E. (1960). "A New Approach to Linear Filtering and Prediction Problems." In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D, pp. 35–45.

Kamilaris, A. and F. X. Prenafeta-Boldú (2018). "Deep learning in agriculture: A survey." In: *Computers and Electronics in Agriculture*. DOI: 10.1016/j.compag.2018.02.016.

Kamiński, B., M. Jakubczyk, and P. Szufel (2018). "A framework for sensitivity analysis of decision trees." In: *Central European Journal of Operations Research*. ISSN: 16139178. DOI: 10.1007/s10100-017-0479-6.

Kaplan, E. (2021). *influxDB*. Online; accessed 2 February 2022. URL: https://www.influxdata.com/.

Karpathy, A., G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. F. Li (2014). "Large-scale video classification with convolutional neural networks." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781479951178. DOI: 10.1109/CVPR.2014.223.

Kawaguchi, K., J. Huang, and L. P. Kaelbling (2019). "Effect of depth and width on local minima in deep learning." In: *IEEE Neural Computation*. DOI: 10.1162/neco_a_01195. eprint: 1811.08150.

Kazemi, M., M. Ghanbari, and S. Shirmohammadi (2021). "A review of temporal video error concealment techniques and their suitability for HEVC and VVC." In: *Multimedia Tools and Applications*. ISSN: 15737721. DOI: 10.1007/s11042-020-10333-6.

Kemp, A. W., A. Stuart, and J. K. Ord (1994). "Kendall's Advanced Theory of Statistics." In: *The Statistician*. ISSN: 00390526. DOI: 10.2307/2348968.

Keras (2019). *Keras Image Preprocessing*. Online; accessed 2 February 2022. URL: https://keras.io/preprocessing/image/.

Khan, A., A. Sohail, and A. Ali (2018). "A New Channel Boosted Convolutional Neural Network using Transfer Learning." In: eprint: 1804.08528 (cs.CV).

Khan, A., A. Sohail, U. Zahoora, and A. S. Qureshi (2020). "A survey of the recent architectures of deep convolutional neural networks." In: *Artificial Intelligence Review* 53.8, pp. 5455–5516. ISSN: 1573-7462. DOI: 10.1007/s10462-020-09825-6. URL: https://doi.org/10.1007/s10462-020-09825-6.

Kingma, D. P. and J. L. Ba (2015). "Adam: A method for stochastic optimization." In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. arXiv: 1412.6980.

Kitani, K. M., B. D. Ziebart, J. A. Bagnell, and M. Hebert (2012). "Activity forecasting." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783642337642. DOI: 10.1007/978-3-642-33765-9_15.

Knight, W. (2017). "DARPA is funding projects that will try to open up AI's black boxes." In: *MIT Technology Review*.

Koenig, N. and A. Howard (2004). "Design and use paradigms for Gazebo, an open-source multi-robot simulator." In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISBN: 0780384636. DOI: 10.1109/iros.2004.1389727.

Konno, T. and M. Iwazume (2018). "Icing on the Cake: An Easy and Quick Post-Learnig Method You Can Try After Deep Learning." In: *arXiv*. eprint: 1807.06540 (cs.LG).

Koubaa, A., A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui (2019). "Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey." In: *IEEE Access*. ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2924410. arXiv: 1906.10641.

Kovacs, L. (2016). "Visual Monocular Obstacle Avoidance for Small Unmanned Vehicles." In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. ISBN: 9781467388504. DOI: 10.1109/CVPRW.2016.114.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2017). "ImageNet classification with deep convolutional neural networks." In: *Communications of the ACM*. ISSN: 15577317. DOI: 10.1145/3065386.

Kruskal, J. B. (1964a). "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis." In: *Psychometrika*. ISSN: 00333123. DOI: 10.1007/BF02289565.

— (1964b). "Nonmetric multidimensional scaling: A numerical method." In: *Psychometrika*. ISSN: 00333123. DOI: 10.1007/BF02289694.

Kuchar, J. K. and L. C. Yang (2000). "A Review of Conflict Detection and Resolution Modeling Methods." In: *IEEE Transactions on Intelligent Transportation Systems*. DOI: 10.1109/6979.898217.

Kuen, J., X. Kong, G. Wang, and Y. P. Tan (2017). "DelugeNets: Deep Networks with Efficient and Flexible Cross-Layer Information Inflows." In: *Proceedings - 2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017*. ISBN: 9781538610343. DOI: 10.1109/ICCVW.2017.117. arXiv: 1611.05552.

La, H. J. and S. D. Kim (2010). "A service-based approach to designing cyber physical systems." In: *Proceedings - 9th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2010*. ISBN: 9780769541471. DOI: 10.1109/ICIS.2010.73.

Lai, K., L. Bo, X. Ren, and D. Fox (2011). "A large-scale hierarchical multi-view RGB-D object dataset." In: *Proceedings - IEEE International Conference on Robotics and Automation*. ISBN: 9781612843865. DOI: 10.1109/ICRA.2011.5980382.

Laptev, I. and T. Lindeberg (2003). "Space-time interest points." In: *Proceedings of the IEEE International Conference on Computer Vision*. DOI: 10.1109/iccv.2003.1238378.

Larsson, G., M. Maire, and G. Shakhnarovich (2019). "FractalNet: Ultra-deep neural networks without residuals." In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. arXiv: 1605.07648.

Laskar, M. N. U., L. G. S. Giraldo, and O. Schwartz (2018). "Correspondence of Deep Neural Networks and the Brain for Visual Textures." In: eprint: 1806.02888 (q-bio.NC).

Le, Q. V. (2013). "Building high-level features using large scale unsupervised learning." In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. ISBN: 9781479903566. DOI: 10.1109/ICASSP.2013.6639343. arXiv: 1112.6209.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (1989). "Backpropagation Applied to Handwritten Zip Code Recognition." In: *IEEE Neural Computation*. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

— (2008). "Backpropagation Applied to Handwritten Zip Code Recognition." In: *IEEE Neural Computation*. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

LeCun, Y., L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. a. Muller, E. Sackinger, P. Simard, and V. Vapnik (1995). "Learning algorithms for classification: A comparison on handwritten digit recognition." In: *Neural networks: the statistical mechanics perspective*.

Lecun, Y., Y. Bengio, and G. Hinton (2015a). "Deep learning." In: *Nature*. DOI: 10.1038/nature14539.

— (2015b). "Deep Learning (Tutorial Slides)." In: *Advances in Neural Information Processing Systems 28 (NIPS 2015)*. ISSN: 0028-0836. DOI: 10.1038/nature14539. arXiv: arXiv:1312.6184v5.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE*. ISSN: 00189219. DOI: 10.1109/5.726791.

LeCun, Y., K. Kavukcuoglu, and C. Farabet (2010). "Convolutional networks and applications in vision." In: *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*. ISBN: 9781424453085. DOI: 10.1109/ISCAS.2010.5537907.

LeCun, Y. A., L. Bottou, G. B. Orr, and K. R. Müller (2012). "Efficient backprop." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISSN: 03029743. DOI: 10.1007/978-3-642-35289-8-3.

Lee, C. Y., P. Gallagher, and Z. Tu (2018). "Generalizing Pooling Functions in CNNs: Mixed, Gated, and Tree." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. ISSN: 01628828. DOI: 10.1109/TPAMI.2017.2703082.

Lee, C. Y., P. W. Gallagher, and Z. Tu (2016). "Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree." In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016*. arXiv: 1509.08985.

Li, C. J. and H. Ling (2015). "Synthetic aperture radar imaging using a small consumer drone." In: *IEEE Antennas and Propagation Society, AP-S International Symposium (Digest)*. ISBN: 9781479978151. DOI: 10.1109/APS.2015.7304729.

Li, Z., Y. Fan, and W. Liu (2015). "The effect of whitening transformation on pooling operations in convolutional autoencoders." In: *Eurasip Journal on Advances in Signal Processing*. ISSN: 16876180. DOI: 10.1186/s13634-015-0222-1.

Lin, M., Q. Chen, and S. Yan (2014a). "Network in network." In: *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*. arXiv: 1312.4400.

Lin, T. Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick (2014b). "Microsoft COCO: Common objects in context." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. DOI: 10.1007/978-3-319-10602-1_48.

Lipton, Z. C. (2018). "The mythos of model interpretability." In: *Communications of the ACM*. ISSN: 15577317. DOI: 10.1145/3233231. arXiv: 1606.03490.

Litman, T. (2014). "Autonomous Vehicle Implementation Predictions: Implications for Transport Planning." In: *Transportation Research Board Annual Meeting*. ISSN: 10769757. DOI: 10.1613/jair.301. arXiv: 9605103 [cs].

Liu, W., Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi (2017). "A survey of deep neural network architectures and their applications." In: *Neurocomputing*. ISSN: 18728286. DOI: 10.1016/j.neucom.2016.12.038.

Liu, X., Z. Deng, and Y. Yang (2019). "Recent progress in semantic image segmentation." In: *Artificial Intelligence Review*. ISSN: 15737462. DOI: 10.1007/s10462-018-9641-3.

Lu, Z., H. Pu, F. Wang, Z. Hu, and L. Wang (2017). "The expressive power of neural networks: A view from the width." In: *Advances in Neural Information Processing Systems*. arXiv: 1709.02540.

Ma, N., X. Zhang, H.-T. Zheng, and J. Sun (2018). "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design." In: *The European Conference on Computer Vision (ECCV)*.

Maas, A. L., R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts (2011). "Learning word vectors for sentiment analysis." In: *ACL-HLT 2011 - Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. ISBN: 9781932432879.

Maaten, L. van der (2014). "Accelerating t-SNE using tree-based algorithms." In: *Journal of Machine Learning Research*. ISSN: 1532-4435.

MacQueen, J (1967). "Some methods for classification and analysis of multivariate observations." In: *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*.

Magree, D., J. G. Mooney, and E. N. Johnson (2014). "Monocular visual mapping for obstacle avoidance on UAVs." In: *Journal of Intelligent and Robotic Systems: Theory and Applications*. ISSN: 09210296. DOI: 10.1007/s10846-013-9967-7.

Mao, X. J., C. Shen, and Y. B. Yang (2016). "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections." In: *Advances in Neural Information Processing Systems*. arXiv: 1603.09056.

Marblestone, A. H., G. Wayne, and K. P. Kording (2016). "Toward an Integration of Deep Learning and Neuroscience." In: *Frontiers in Computational Neuroscience*. DOI: 10.3389/fncom.2016.00094.

Marcus, G. (2015). "Is "Deep Learning" a Revolution in Artificial Intelligence?" In: *New Yorker*.

— (2018). *In defense of skepticism about deep learning*. Online; accessed 2 February 2022. URL: https://medium.com/@GaryMarcus/in-defense-of-skepticism-about-deep-learning-6e8bfd5ae0f1.

Marr, D. (1980). "Visual information processing: the structure and creation of visual representations." In: *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*. ISSN: 09628436. DOI: 10.1098/rstb.1980.0091.

Matos-Carvalho, J. P., D. Pedro, L. M. Campos, J. M. Fonseca, and A. Mora (2020). "Terrain classification using w-k filter and 3d navigation with static collision avoidance." In: *Advances in Intelligent Systems and Computing*. ISBN: 9783030295127. DOI: 10.1007/978-3-030-29513-4_81.

Meinhardt-Llopis, E., J. Sánchez Pérez, and D. Kondermann (2013). "Horn-Schunck Optical Flow with a Multi-Scale Strategy." In: *Image Processing On Line*. ISSN: 2105-1232. DOI: 10.5201/ipol.2013.20.

Merz, T. and F. Kendoul (2011). "Beyond visual range obstacle avoidance and infrastructure inspection by an autonomous helicopter." In: *IEEE International Conference on Intelligent Robots and Systems*. ISBN: 9781612844541. DOI: 10.1109/IROS.2011.6048249.

Metz, J. (2016). *The AI Revolution: Why Deep Learning Is Suddenly Changing Your Life*. Online; accessed 2 February 2022. URL: https://fortune.com/.

Mikolov, T., M. Karafiát, L. Burget, C. Jan, and S. Khudanpur (2010). "Recurrent neural network based language model." In: *Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010*.

Mnih, V., N. Heess, A. Graves, and K. Kavukcuoglu (2014). "Recurrent models of visual attention." In: *Advances in Neural Information Processing Systems*. arXiv: 1406.6247.

Montúfar, G., R. Pascanu, K. Cho, and Y. Bengio (2014). "On the number of linear regions of deep neural networks." In: *Advances in Neural Information Processing Systems*. arXiv: 1402.1869.

Morar, A., F. Moldoveanu, and E. Gröller (2012). "Image segmentation based on active contours without edges." In: *Proceedings - 2012 IEEE 8th International Conference on Intelligent Computer Communication and Processing, ICCP 2012*. ISBN: 9781467329514. DOI: 10.1109/ICCP.2012.6356188.

Mousavi, S. S., M. Schukat, and E. Howley (2018). "Deep Reinforcement Learning: An Overview." In: *Lecture Notes in Networks and Systems*. DOI: 10.1007/978-3-319-56991-8_32.

Mueggler, E., C. Forster, N. Baumli, G. Gallego, and D. Scaramuzza (2015). "Lifetime estimation of events from Dynamic Vision Sensors." In: *Proceedings - IEEE International Conference on Robotics and Automation*. DOI: 10.1109/ICRA.2015.7139876.

MultiWii (2020). *MultiWii project related stuffs*. Online; accessed 2 February 2022. URL: http://www.multiwii.com/.

Murtagh, F. and P. Legendre (2014). "Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion?" In: *Journal of Classification* 31.3, 274–295. ISSN: 1432-1343. DOI: 10.1007/s00357-014-9161-z. URL: http://dx.doi.org/10.1007/s00357-014-9161-z.

Müllner, D. (2011). *Modern hierarchical, agglomerative clustering algorithms*. Online; accessed 2 February 2022. URL: https://arxiv.org/abs/1109.2378.

Nair, V. and G. E. Hinton (2010). "Rectified linear units improve Restricted Boltzmann machines." In: *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*. ISBN: 9781605589077.

Najafabadi, M. M., F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic (2015). "Deep learning applications and challenges in big data analytics." In: *Journal of Big Data*. ISSN: 21961115. DOI: 10.1186/s40537-014-0007-7.

Nelli, F. and F. Nelli (2018). "Deep Learning with TensorFlow." In: *Python Data Analytics*. DOI: 10.1007/978-1-4842-3913-1_9.

Nguyen, A., J. Yosinski, and J. Clune (2015). "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298640.

Nguyen, Q., M. C. Mukkamala, and M. Hein (2018). "Neural networks should be wide enough to learn disconnected decision regions." In: *35th International Conference on Machine Learning, ICML 2018*. ISBN: 9781510867963. eprint: 1803.00094.

Nwankpa, C., W. Ijomah, A. Gachagan, and S. Marshall (2020). "Activation Functions: Comparison of trends in Practice and Research for Deep Learning." In: *Conference: 2nd International Conference on Computational Sciences and Technology, (INCCST) 2020*. arXiv: 1811.03378 [cs.LG].

O'Connell, A. A., I. Borg, and P. Groenen (1999). "Modern Multidimensional Scaling: Theory and Applications." In: *Journal of the American Statistical Association*. ISSN: 01621459. DOI: 10.2307/2669710.

Oh, K. S. and K. Jung (2004). "GPU implementation of neural networks." In: *Pattern Recognition*. ISSN: 00313203. DOI: 10.1016/j.patcog.2004.01.013.

Olshausen, B. A. and D. J. Field (1996). "Emergence of simple-cell receptive field properties by learning a sparse code for natural images." In: *Nature*. ISSN: 00280836. DOI: 10.1038/381607a0.

P. Valavanis, K. (Jan. 2007). *Advances in Unmanned Aerial Vehicles: State of the Art and the Road to Autonomy*. Springer, Dordrecht. ISBN: 978-1-4020-6113-4. DOI: 10.1007/978-1-4020-6114-1.

Pan, S. J. and Q. Yang (2010). *A survey on transfer learning*. DOI: 10.1109/TKDE.2009.191.

Pascanu, R., T. Mikolov, and Y. Bengio (2012). "Understanding the exploding gradient problem." In: *Proceedings of The 30th International Conference on Machine Learning*. ISSN: 1045-9227. DOI: 10.1109/72.279181. eprint: arXiv:1211.5063v2.

Paul, S. (1998). "Real-Time Transport Protocol (RTP)." In: *Multicasting on the Internet and its Applications*. DOI: 10.1007/978-1-4615-5713-5_16.

Pedro, D., S. Tomic, L. Bernardo, M. Beko, R. Oliveira, R. Dinis, and P. Pinto (2018). "Localization of static remote devices using smartphones." In: *IEEE Vehicular Technology Conference*. ISBN: 9781538663554. DOI: 10.1109/VTCSpring.2018.8417726.

Pedro, D., P. Lousa, A. Ramos, J. Matos-Carvalho, F. Azevedo, and L. Campos (2021). "HEIFU - Hexa Exterior Intelligent Flying Unit." In: Computer Safety, Reliability, and Security. SAFECOMP 2021 Workshops, pp. 89–104. ISBN: 978-3-030-83906-2.

Pedro, D., A. Mora, J. Carvalho, F. Azevedo, and J. Fonseca (2020a). "ColANet: A UAV Collision Avoidance Dataset." In: ISSN: 1868422X. DOI: 10.1007/978-3-030-45124-0_5.

Pedro, D., J. P. Matos-Carvalho, F. Azevedo, R. Sacoto-Martins, L. Bernardo, L. Campos, J. M. Fonseca, and A. Mora (2020b). "FFAU—Framework for Fully Autonomous UAVs." In: *Remote Sensing* 12.21. ISSN: 2072-4292. DOI: 10.3390/rs12213533. URL: https://www.mdpi.com/2072-4292/12/21/3533.

Perazzi, F., J. Pont-Tuset, B. McWilliams, L. V. Gool, M. Gross, and A. Sorkine-Hornung (2016). "A benchmark dataset and evaluation methodology for video object segmentation." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.85.

Poiesi, F. and A. Cavallaro (2017). "Detection of fast incoming objects with a moving camera." In: *British Machine Vision Conference (BMVC)*. DOI: 10.5244/c.30.146.

Popovski, P., C. Stefanovic, J. J. Nielsen, E. de Carvalho, M. Angjelichinoski, K. F. Trillingsgaard, and A.-S. Bana (2019). "Wireless Access in Ultra-Reliable Low-Latency Communication (URLLC)." In: *IEEE Transactions on Communications*. ISSN: 0090-6778. DOI: 10.1109/tcomm.2019.2914652. arXiv: 1810.06938.

Potluri, S., A. Fasih, L. K. Vutukuru, F. A. MacHot, and K. Kyamakya (2011). "CNN based high performance computing for real time image processing on GPU." In: *Studies in Computational Intelligence*. ISSN: 1860949X. DOI: 10.1007/978-3-642-24806-1_20.

Powering the world's robots (2007). Online; accessed 2 February 2022. URL: https://www.ros.org/.

PX4 Open Source Autopilot (2019). *PX4 Autopilot User Guide*. Online; accessed 2 February 2022. URL: https://px4.io/.

Quadros, A., J. P. Underwood, and B. Douillard (2012). "An occlusion-aware feature for range images." In: *Proceedings - IEEE International Conference on Robotics and Automation*. ISBN: 9781467314039. DOI: 10.1109/ICRA.2012.6225239.

Raina, R., A. Madhavan, and A. Y. Ng (2009). "Large-scale deep unsupervised learning using graphics processors." In: *Proceedings of the 26th Annual International Conference on Machine Learning*. DOI: 10.1145/1553374.1553486.

Ramasamy, S., R. Sabatini, A. Gardi, and J. Liu (2016). "LIDAR obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid." In: *Aerospace Science and Technology*. ISSN: 12709638. DOI: 10.1016/j.ast.2016.05.020.

Ranzato, M., F. J. Huang, Y. L. Boureau, and Y. LeCun (2007). "Unsupervised learning of invariant feature hierarchies with applications to object recognition." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 1424411807. DOI: 10.1109/CVPR.2007.383157.

Raudies, F. (2013). "Optical FLow." In: *Optic flow*. DOI: doi:10.4249/scholarpedia.30724. URL: http://scholarpedia.org/article/Optic{_}flow.

Rawat, W. and Z. Wang (2017). "Deep convolutional neural networks for image classification: A comprehensive review." In: *IEEE Neural Computation*. DOI: 10.1162/NECO_a_00990.

Rawlinson, K. (2016). *Drone hits plane at Heathrow airport, says pilot*. Online; accessed 2 February 2022. URL: https://www.theguardian.com/uk-news/2016/apr/17/drone-plane-heathrow-airport-british-airways.

Ronneberger, O., P. Fischer, and T. Brox (2015). "U-net: Convolutional networks for biomedical image segmentation." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783319245737. DOI: 10.1007/978-3-319-24574-4_28.

Ros, G., L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez (2016). "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban

Scenes." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.352.

Roy, A. G., N. Navab, and C. Wachinger (2018). "Concurrent spatial and channel 'squeeze & excitation' in fully convolutional networks." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783030009274. DOI: 10.1007/978-3-030-00928-1_48. arXiv: 1803.02579.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei (2015). "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision*. ISSN: 15731405. DOI: 10.1007/s11263-015-0816-y.

Russell, B. C., A. Torralba, K. P. Murphy, and W. T. Freeman (2008). "LabelMe: A database and web-based tool for image annotation." In: *International Journal of Computer Vision*. ISSN: 09205691. DOI: 10.1007/s11263-007-0090-8.

Ryan, A., M. Zennaro, A. Howell, R. Sengupta, and J. K. Hedrick (2004). "An overview of emerging results in cooperative UAV control." In: *Proceedings of the IEEE Conference on Decision and Control*. DOI: 10.1109/cdc.2004.1428700.

Saab (2018). *Saab Australia*. Online; accessed 2 February 2022. URL: https://saab.com/region/saab-australia/.

Sabatini, R., A. Gardi, and M. A. Richardson (2014). "LIDAR Obstacle Warning and Avoidance System for Unmanned Aircraft." In: *International Journal of Mechanical, Aerospace, Industrial and Mechatronics Engineering*, pp. 718–729.

Salakhutdinov, R. and H. Larochelle (2010). "Efficient learning of Deep Boltzmann Machines." In: *Journal of Machine Learning Research*.

Samangouei, P., M. Kabkab, and R. Chellappa (2018). "Defense-Gan: Protecting classifiers against adversarial attacks using generative models." In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*. arXiv: 1805.06605.

Sandler, M., A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen (2018a). "MobileNetV2: Inverted Residuals and Linear Bottlenecks." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781538664209. DOI: 10.1109/CVPR.2018.00474. arXiv: 1801.04381.

— (2018b). "MobileNetV2: Inverted Residuals and Linear Bottlenecks." In: *arXiv*. eprint: 1801.04381 (cs.CV).

Scellier, B. and Y. Bengio (2016). "Towards a Biologically Plausible Backprop." In: *Arxiv*. arXiv: 1602.05179.

Scherer, D., A. Müller, and S. Behnke (2010). "Evaluation of pooling operations in convolutional architectures for object recognition." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 3642158242. DOI: 10.1007/978-3-642-15825-4_10.

Schlögl, S., C. Postulka, R. Bernsteiner, and C. Ploder (2019). "Artificial intelligence tool penetration in business: Adoption, challenges and fears." In: *Communications in Computer and Information Science*. ISBN: 9783030214500. DOI: 10.1007/978-3-030-21451-7_22.

Schmidhuber, J. (2015). *Deep Learning in neural networks: An overview*. DOI: 10.1016/j.neunet.2014.09.003.

Selvaraju, R. R., M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra (2019). "Grad-CAM : Visual Explanations from Deep Networks." In: *International Journal of Computer Vision*. arXiv: arXiv:1610.02391v4.

Shanmugamani, R. (2018). *Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and keras*. ISBN: 9781788295628. DOI: 10.1007/978-1-4842-4261-2_3.

Sharma, A. and S. K. Muttoo (2019). "Spatial image steganalysis based on ResNeXt." In: *International Conference on Communication Technology Proceedings, ICCT*. ISBN: 9781538676349. DOI: 10.1109/ICCT.2018.8600132.

Shen, F., C. Shen, X. Zhou, Y. Yang, and H. T. Shen (2016a). "Face image classification by pooling raw features." In: *Pattern Recognition*. ISSN: 00313203. DOI: 10.1016/j.patcog.2016.01.010. arXiv: 1406.6811.

Shen, X., A. Hertzmann, J. Jia, S. Paris, B. Price, E. Shechtman, and I. Sachs (2016b). "Automatic portrait segmentation for image stylization." In: *Computer Graphics Forum*. ISSN: 14678659. DOI: 10.1111/cgf.12814.

Sheridan, T. B. (1993). "Telerobotics, automation, and human supervisory control." In: *IEEE Technology and Society Magazine*. ISBN: 0262193167. DOI: 10.1109/MTAS.1993.232279.

Shin, H. C., H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers (2016). "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning." In: *IEEE Transactions on Medical Imaging*. ISSN: 1558254X. DOI: 10.1109/TMI.2016.2528162. arXiv: 1602.03409.

Shorten, C. and T. M. Khoshgoftaar (2019). "A survey on Image Data Augmentation for Deep Learning." In: *Journal of Big Data*. ISSN: 21961115. DOI: 10.1186/s40537-019-0197-0.

Silberman, N., D. Hoiem, P. Kohli, and R. Fergus (2012). "Indoor segmentation and support inference from RGBD images." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783642337147. DOI: 10.1007/978-3-642-33715-4_54.

Simonyan, K., A. Vedaldi, and A. Zisserman (2014). "Deep inside convolutional networks: Visualising image classification models and saliency maps." In: *2nd International Conference on Learning Representations, ICLR 2014 - Workshop Track Proceedings*. arXiv: 1312.6034.

Simonyan, K. and A. Zisserman (2015). "Very deep convolutional networks for large-scale image recognition." In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. arXiv: 1409.1556.

Society of Automotive, E. (2014). "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems (J3016 Ground Vehicle Standard)." In: *SAE International*.

Song, S., S. P. Lichtenberg, and J. Xiao (2015). "SUN RGB-D: A RGB-D scene understanding benchmark suite." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298655.

Srinivas, S., R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. S. Kruthiventi, and R. V. Babu (2016). "A taxonomy of deep convolutional neural nets for computer vision." In: *Frontiers Robotics AI*. ISSN: 22969144. DOI: 10.3389/frobt.2015.00036. arXiv: 1601.06615.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). "Dropout: A simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research*. ISSN: 15337928.

Srivastava, R. K., K. Greff, and J. Schmidhuber (2015a). "Highway Networks." In: *ICML 2015 Deep Learning workshop*. arXiv: 1505.00387 [cs.LG].

— (2015b). "Training very deep networks." In: *Advances in Neural Information Processing Systems*. arXiv: 1507.06228.

Statnikov, A., C. F. Aliferis, D. P. Hardin, and I. Guyon (2011). "Support Vector Clustering." In: *A Gentle Introduction to Support Vector Machines in Biomedicine*. DOI: 10.1142/9789814335140_0009.

Stewart, P. and P. Stewart (2019). "YouTube." In: *The Live-Streaming Handbook*. DOI: 10.4324/9781315209883-5.

Sundermeyer, M., R. Schlüter, and H. Ney (2012). "LSTM neural networks for language modeling." In: *13th Annual Conference of the International Speech Communication Association 2012, INTERSPEECH 2012*. ISBN: 9781622767595.

Sze, V., Y. H. Chen, T. J. Yang, and J. S. Emer (2017). "Efficient Processing of Deep Neural Networks: A Tutorial and Survey." In: *Proceedings of the IEEE*. DOI: 10.1109/JPROC.2017.2761740.

Szegedy, C., S. Ioffe, V. Vanhoucke, and A. A. Alemi (2017). "Inception-v4, inception-ResNet and the impact of residual connections on learning." In: *31st AAAI Conference on Artificial Intelligence, AAAI 2017*. arXiv: 1602.07261.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). "Going deeper with convolutions." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298594. arXiv: 1409.4842.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2016). "Rethinking the Inception Architecture for Computer Vision." In: *Proceedings of the IEEE Computer*

Society Conference on Computer Vision and Pattern Recognition. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.308. arXiv: 1512.00567.

Tellman, J. and T. V. News (2018). *First-ever recorded drone-hot air balloon collision prompts safety conversation*. Online; accessed 2 February 2022. URL: https://mavicpilots.com/threads/drone-collides-with-hot-air-balloon.49300/.

Tenenbaum, J. B., V. De Silva, and J. C. Langford (2000). "A global geometric framework for nonlinear dimensionality reduction." In: *Science*. ISSN: 00368075. DOI: 10.1126/science.290.5500.2319.

The Economist (2010). "A special report on managing information: Clicking for gold." In: *The Economist*.

Tong, T., G. Li, X. Liu, and Q. Gao (2017). "Image Super-Resolution Using Dense Skip Connections." In: *Proceedings of the IEEE International Conference on Computer Vision*. ISBN: 9781538610329. DOI: 10.1109/ICCV.2017.514.

Tran, D., L. Bourdev, R. Fergus, L. Torresani, and M. Paluri (2015). "Learning spatiotemporal features with 3D convolutional networks." In: *Proceedings of the IEEE International Conference on Computer Vision*. ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.510. arXiv: 1412.0767.

UAV, A. (2019). *Types of Drones: Multi-Rotor vs Fixed-Wing vs Single Rotor vs Hybrid VTOL*. Online; accessed 2 February 2022. URL: https://www.auav.com.au/articles/drone-types/.

Valavanis, K. P. and G. J. Vachtsevanos (2015). "Handbook of unmanned aerial vehicles." In: *Handbook of Unmanned Aerial Vehicles*. ISBN: 9789048197071. DOI: 10.1007/978-90-481-9707-1.

Van Der Maaten, L. J. P. and G. E. Hinton (2008). "Visualizing high-dimensional data using t-sne." In: *Journal of Machine Learning Research*. ISSN: 1532-4435.

Viebke, A., S. Memeti, S. Pllana, and A. Abraham (2019). "CHAOS: a parallelization scheme for training convolutional neural networks on Intel Xeon Phi." In: *Journal of Supercomputing*. ISSN: 15730484. DOI: 10.1007/s11227-017-1994-x.

Vincent, P., H. Larochelle, Y. Bengio, and P. A. Manzagol (2008). "Extracting and composing robust features with denoising autoencoders." In: *Proceedings of the 25th International Conference on Machine Learning*. ISBN: 9781605582054. DOI: 10.1145/1390156.1390294.

Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2015). "Show and tell: A neural image caption generator." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298935.

Waizenegger, W., I. Feldmann, and O. Schreer (2011). "Real-time patch sweeping for high-quality depth estimation in 3D video conferencing applications." In: *Real-Time Image and Video Processing 2011*. ISBN: 9780819484086. DOI: 10.1117/12.872868.

Wang, F., M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang (2017). "Residual attention network for image classification." In: *Proceedings - 30th IEEE Conference*

on Computer Vision and Pattern Recognition, CVPR 2017. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.683. arXiv: 1704.06904.

Wang, H. and B. Raj (2017). "On the Origin of Deep Learning." In: *arXiv*. eprint: 1702.07800 (cs.LG).

Wang, T., D. J. Wu, A. Coates, and A. Y. Ng (2012). "End-to-end text recognition with convolutional neural networks." In: *Proceedings - International Conference on Pattern Recognition*. ISBN: 9784990644109.

Wang, Y. and S. Lin (2002). "Error-resilient video coding using multiple description motion compensation." In: *IEEE Transactions on Circuits and Systems for Video Technology*. ISSN: 10518215. DOI: 10.1109/TCSVT.2002.800320.

Wang, Y., L. Wang, H. Wang, and P. Li (2019). "End-to-End Image Super-Resolution via Deep and Shallow Convolutional Networks." In: *IEEE Access*. ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2903582. eprint: 1607.07680.

Waymo (2018). *Waymo One*. Online; accessed 2 February 2022. URL: https://waymo.com/.

Willee, H. (2005). *Introduction*. Online; accessed 2 February 2022. URL: https://mavlink.io/en/.

Woo, S., J. Park, J. Y. Lee, and I. S. Kweon (2018). "CBAM: Convolutional block attention module." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783030012335. DOI: 10.1007/978-3-030-01234-2_1. eprint: 1807.06521.

Wu, C. J., D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang (2019). "Machine learning at facebook: Understanding inference at the edge." In: *Proceedings - 25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019*. ISBN: 9781728114446. DOI: 10.1109/HPCA.2019.00048.

Wu, L., M. A. Garcia, D. Puig, and A. Sole (2007). "Voronoi-based space partitioning for coordinated multi-robot exploration." In: *Journal of Physical Agents*. ISSN: 18880258. DOI: 10.14198/JoPha.2007.1.1.05.

Xiang, Y., W. Kim, W. Chen, J. Ji, C. Choy, H. Su, R. Mottaghi, L. Guibas, and S. Savarese (2016). "Objectnet3D: A large scale database for 3D object recognition." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783319464831. DOI: 10.1007/978-3-319-46484-8_10.

Xiao, J., A. Owens, and A. Torralba (2013). "SUN3D: A database of big spaces reconstructed using SfM and object labels." In: *Proceedings of the IEEE International Conference on Computer Vision*. ISBN: 9781479928392. DOI: 10.1109/ICCV.2013.458.

Xie, S., R. Girshick, P. Dollár, Z. Tu, and K. He (2017). "Aggregated residual transformations for deep neural networks." In: *Proceedings - 30th IEEE Conference on Computer*

Vision and Pattern Recognition, CVPR 2017. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.634. arXiv: 1611.05431.

Xu, B., R. Huang, and M. Li (2016). *Revise Saturated Activation Functions*. arXiv: 1602.05980 [cs.LG].

Xu, B., N. Wang, T. Chen, and M. Li (2015). "Empirical Evaluation of Rectified Activations in Convolutional Network." In: *CoRR* abs/1505.00853. arXiv: 1505.00853 [cs.LG]. URL: http://arxiv.org/abs/1505.00853.

Yang, Z., F. Gao, and S. Shen (2017). "Real-time monocular dense mapping on aerial robots using visual-inertial fusion." In: *Proceedings - IEEE International Conference on Robotics and Automation*. ISBN: 9781509046331. DOI: 10.1109/ICRA.2017.7989529.

You, Y., A. Buluç, and J. Demmel (2017). "Scaling deep learning on GPU and knights landing clusters." In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. DOI: 10.1145/3126908.3126912.

Zach, C, T Pock, and H Bischof (2007). "A Duality Based Approach for Realtime TV-L1 Optical Flow." In: *Pattern Recognition*. ISBN: 978-3-540-74933-2.

Zagoruyko, S. and N. Komodakis (2016). "Wide Residual Networks." In: *British Machine Vision Conference 2016, BMVC 2016*. DOI: 10.5244/C.30.87. eprint: 1605.07146.

Zeiler, M. D. and R. Fergus (2014). "Visualizing and understanding convolutional networks." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. ISBN: 9783319105895. DOI: 10.1007/978-3-319-10590-1_53. arXiv: 1311.2901.

Zhang, H. Y., W. M. Lin, and A. X. Chen (2018). "Path planning for the mobile robot: A review." In: *Symmetry*. ISSN: 20738994. DOI: 10.3390/sym10100450.

Zhang, N., M. Paluri, M. Ranzato, T. Darrell, and L. Bourdev (2014). "PANDA: Pose aligned networks for deep attribute modeling." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781479951178. DOI: 10.1109/CVPR.2014.212. arXiv: 1311.5591.

Zhang, X., J. Zou, K. He, and J. Sun (2016). "Accelerating Very Deep Convolutional Networks for Classification and Detection." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. ISSN: 01628828. DOI: 10.1109/TPAMI.2015.2502579. arXiv: 1505.06798.

Zhang, X., Z. Li, C. C. Loy, and D. Lin (2017). "PolyNet: A pursuit of structural diversity in very deep networks." In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.415. arXiv: 1611.05725.

Zhao, B., B. Wu, T. Wu, and Y. Wang (2017). "Zero-Shot learning posed as a missing data problem." In: *Proceedings - 2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017*. ISBN: 9781538610343. DOI: 10.1109/ICCVW.2017.310. eprint: 1612.00560.

Zhao, J., B. Liang, and Q. Chen (2018). "The key technology toward the self-driving car." In: *International Journal of Intelligent Unmanned Systems*. DOI: 10.1108/IJIUS-08-2017-0008.

Zheng, H., J. Fu, T. Mei, and J. Luo (2017). "Learning Multi-attention Convolutional Neural Network for Fine-Grained Image Recognition." In: *Proceedings of the IEEE International Conference on Computer Vision*. ISBN: 9781538610329. DOI: 10.1109/ICCV.2017.557.

Zhong, S.-h., Y. Liu, and Y. Liu (2011). "Bilinear deep learning for image classification." In: *Proceedings of the 19th ACM international conference on Multimedia - MM '11*. ISBN: 9781450306164. DOI: 10.1145/2072298.2072344.

Zhou, B., A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba (2016). "Learning Deep Features for Discriminative Localization." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.319. eprint: 1512.04150.

Zhou, B., A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva (2014). "Learning deep features for scene recognition using places database." In: *Advances in Neural Information Processing Systems*.

Zhu, S.-C. and D. Mumford (2006). "A Stochastic Grammar of Images." In: *Foundations and Trends® in Computer Graphics and Vision*. ISSN: 1572-2740. DOI: 10.1561/0600000018.

Appendix A

Machine Learning

This appendix contains a detailed analyses on SoA Machine Learning that are the pillars for this thesis. This thesis will focus mainly on DL, but it's important to understand the different fields inside AI and organize them in such a way that it's possible to find parallelism between subsets and extract advantages from their differences (Domingos, 2012).

AI is a field of computer science that aims to make computers achieve human-style intelligence. As represented in figure A.1, ML is a subset of AI, which contains a subset that try to replicate the human-brain called NN. NN contains large neural models which finally get us to the field of DL.

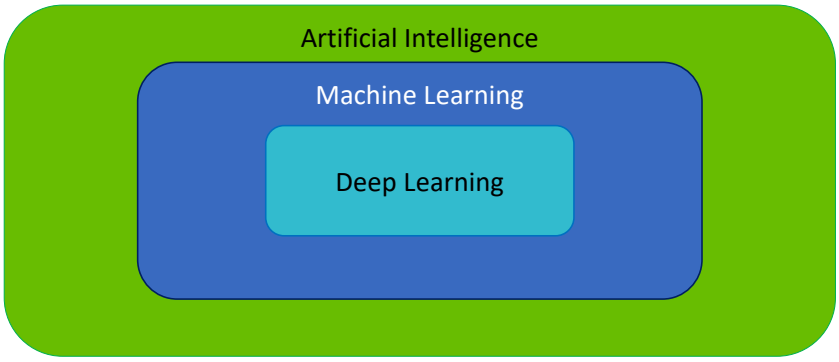


Figure A.1: Artificial Intelligence

ML is a set of related techniques in which computers are trained to perform a particular task rather than by explicitly programming them. ML algorithms can be used to infer relationships and extract knowledge from gathered data.

A NN is a construction type in ML inspired by the network of neurons (nerve cells) in

the biological brain. NN are a fundamental part of DL, and will be covered in this thesis.

Finally there is DL, which is a subfield of ML, that uses multi-layered neural networks. Often, ML and DL are used interchangeably.

This thesis will first go through the three main ways of learning, and then try to cluster the ML techniques into five clusters (Domingos, 2015). There are main approaches for learning algorithms are Supervised Learning (SL), Unsupervised Learning (UL) and Reinforcement Learning (RL) (Ayodele, 2010).

- SL consist in obtaining outcome variables (or dependent variables) which are predicted from a given set of predictor variables (data features). Using these set of variables, a function that maps inputs to desired outputs is generated in what is called the training process. The process finishes when the model achieves a desired level of accuracy on the training data. Examples of SL algorithms are: K-Nearest Neighbor (KNN), Random Forest, Decision Tree and Logistic Regression. A sample representation of a SL workflow is illustrated on figure A.2.

  On figure the dataset is divided by colors. After training the algorithm correctly classifies each object by it's characteristic color.

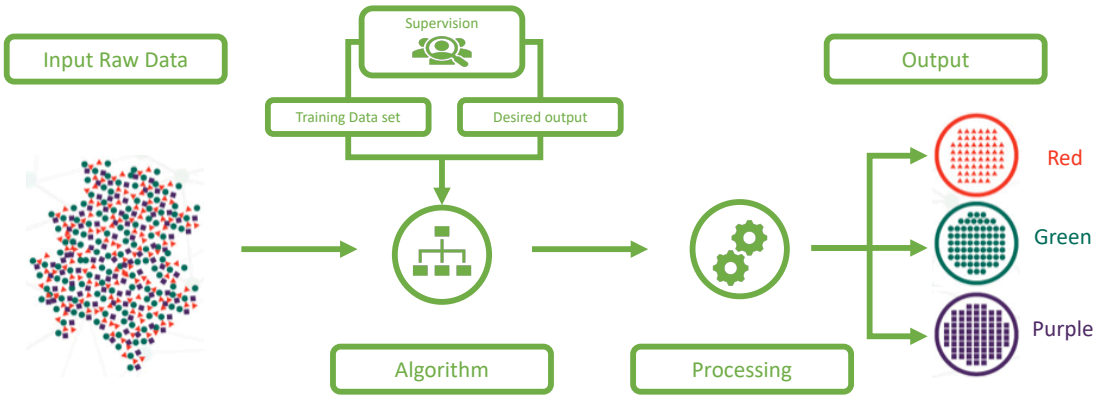


Figure A.2: Supervised Learning.

- UL is a data-driven knowledge discovery approach that can automatically infer a function that describes the structure of the analyzed data or can highlight correlations in the data forming different clusters of related data. A UL workflow is depicted at figure A.3. Examples of algorithms include: K-Means, Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Apriori.

  On figure A.3 no information is given to the algorithm, and it has to discover that the input contains objects of different shapes and colors. Afterwards, he will group the different objects according to it's similarities. At the end, the output should be 3 clusters, with the different colors. Notice that in the case of SL the algorithm was able to recognize that a given object belong to the color class, whereas on the UL it just as the concept of the object belonging to a different category.

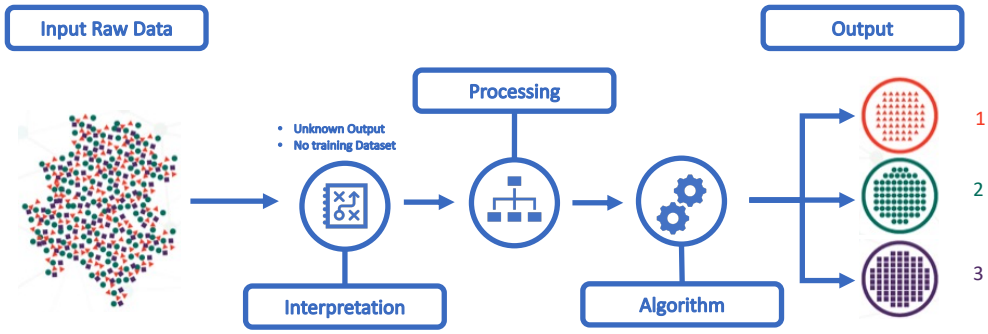


Figure A.3: Unsupervised Learning.

- Reinforcement Learning algorithms are trained to make specific decisions. The goal is to discover which actions lead to an optimal policy. This is done by learning from past experiences, as represented at figure A.4. As an example, a target policy is set, for instance the delay of a set of flows in an Software Defined Networking (SDN). Then an algorithm results in actions on the SDN controller that change the configuration and for each action a reward is received, which increases as the in-place policy gets closer to the target policy. Ultimately, the algorithm will learn the set of configuration updates (actions) that result in such target policy (e.g. Markov Decision Process).

  Compared to SL and UL, RL is slightly different, in the sense he does not intend to map the input to the output. For example, he can try to take actions towards the region area that contains the maximum number of red objects, but he does it undefinability, until he reaches an end function.

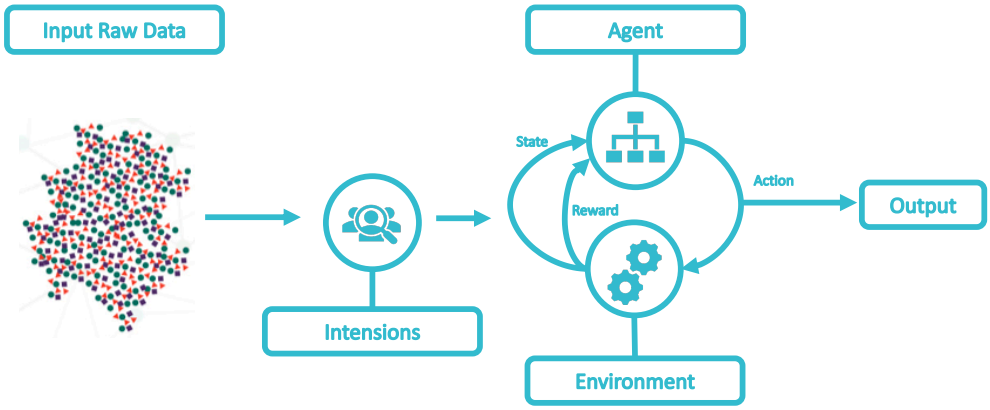


Figure A.4: Reinforcement Learning.

As described in (Domingos, 2012), there are 12 important key points that should be

kept in mind when working with ML:

1. Learning = Representation + Evaluation + Optimization.
    a) Representation - A classifier must be represented in a formal language that the computer can handle. Creating a set of classifiers the learner can learn is crucial.
    b) Evaluation - An Evaluation function is needed to distinguish good classifiers from bad ones.
    c) Optimization - A method to search among the classifiers in the language for the highest scoring one. The choice of optimization technique is key to the efficiency of the algorithm.
2. It is generalization that matters.
3. Data alone is not enough.
4. Overfitting has many faces.
5. Intuition fails in high dimensions.
6. Theoretical guarantees are not what they seem.
7. Feature engineering is the key.
8. More data beats a cleverer algorithm.
9. Learn many models not just one.
10. Simplicity does not imply accuracy.
11. Representable does not imply learnable.
12. Correlation does not imply Causation .

A.1 Machine Learning Fields

ML has many subfields, branches, and special techniques. To over simplify — in SL you know what you want to teach the computer, while UL is about letting the computer figure out what can be learned. SL is the most common type of ML and will be the focus of our work.

The majority of ML algorithms can be clustered into five clusters (Domingos, 2015), as summarized on table A.1 :

The symbolist cluster represent algorithms who believe in discovering new knowledge by filling in the gaps in the knowledge that you already have. They are the ones that most relate to computer science in the five clusters. Their master algorithm is inverse deduction.

Table A.1: Different types of Machine Learning.

Cluster	Origins	Strength	Main Algorithm
Symbolist	Logic & philosophy	Structure Inference	Inverse deduction
Connectionists	Neuroscience	Estimating Parameters	Neural Networks
Evolutionaries	Evolutionary biology	Weighing Evidence	Genetic programming
Bayesians	Statistics	Structure Learning	Probabilistic Inference
Analogizers	Psychology	Mapping to Novelty	Kernel Machines

For the symbolist, learning is the inverse of deduction, which means that learning is the induction of knowledge. In practical terms, they try to create general rules from specific facts. On figure A.5, is a simplistic representation of a typical symbolist algorithm, a decision tree. On this example, a character classifier is presented, where the output are 4 different possible groups. A decision tree has multiple types of nodes (Kamiński et al., 2018). On figure A.5, decision nodes are represented in green and end nodes are represented in blue. The green arrows represent a true evaluation at the node, while the red arrows represent a false evaluation. A decision tree is a flowchart-like structure in which each internal node represents a "test"on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

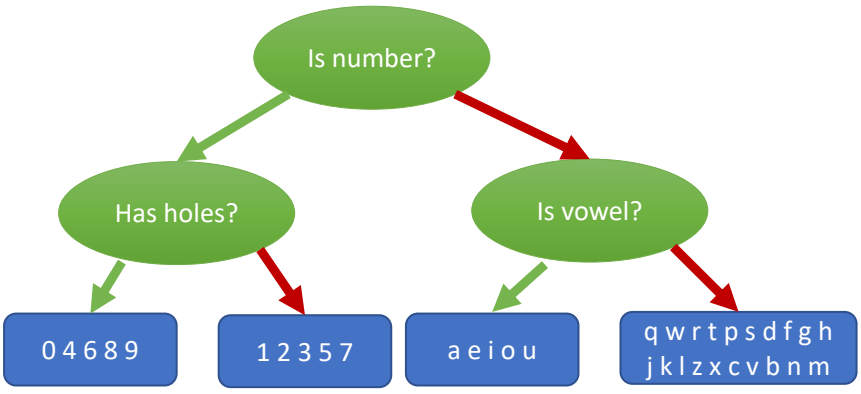


Figure A.5: Symbolist Representation

In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

Among decision support tools, decision trees (and influence diagrams) have several advantages, such as:

- Are simple to understand and interpret. People are able to understand decision tree models after a brief explanation.

- Have value even with small datasets. Important insights can be generated based on experts describing a situation (its alternatives, probabilities, or costs) and their preferences for outcomes.
- Help determine worst, best and expected values for different scenarios.
- Use a white box model. If a given result is provided by a model.
- Can be combined with other decision techniques.

On the other hand, decision trees have some disadvantages:

- They are unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal decision tree.
- They are often relatively inaccurate. Many other predictors perform better with similar data. This can be remedied by replacing a single decision tree with a random forest of decision trees, but a random forest is not as easy to interpret as a single decision tree.
- For data that include categorical variables with different number of levels, information gain in decision trees is biased in favor of those attributes with more levels (Deng et al., 2011).
- Calculations can get very complex, particularly if many values are uncertain and/or if many outcomes are linked.

The evolutionaries, have origins in the evolutionary biology. The main algorithm of this school, is the genetic programming and consist on replicating the process of genetic evolution.

As it is illustrated in figure A.6, it starts from a population of unfit (usually random) elements, and they are iteratively fit for a particular task by applying operations analogous to natural genetic processes to the population. It is essentially a heuristic search technique that searches for an optimal or at least suitable element.

The typical operations of a Genetic algorithm are:

1. **Selection**: the fittest elements for reproduction (crossover) and mutation are selected according to a predefined fitness measure, usually proficiency at the desired task.
2. **Crossover**: involves swapping random parts of selected pairs (parents) to produce new and different offspring that become part of the new generation of elements.
3. **Mutation**: consists involves substitution of some random part of a element with some other random part of another element.

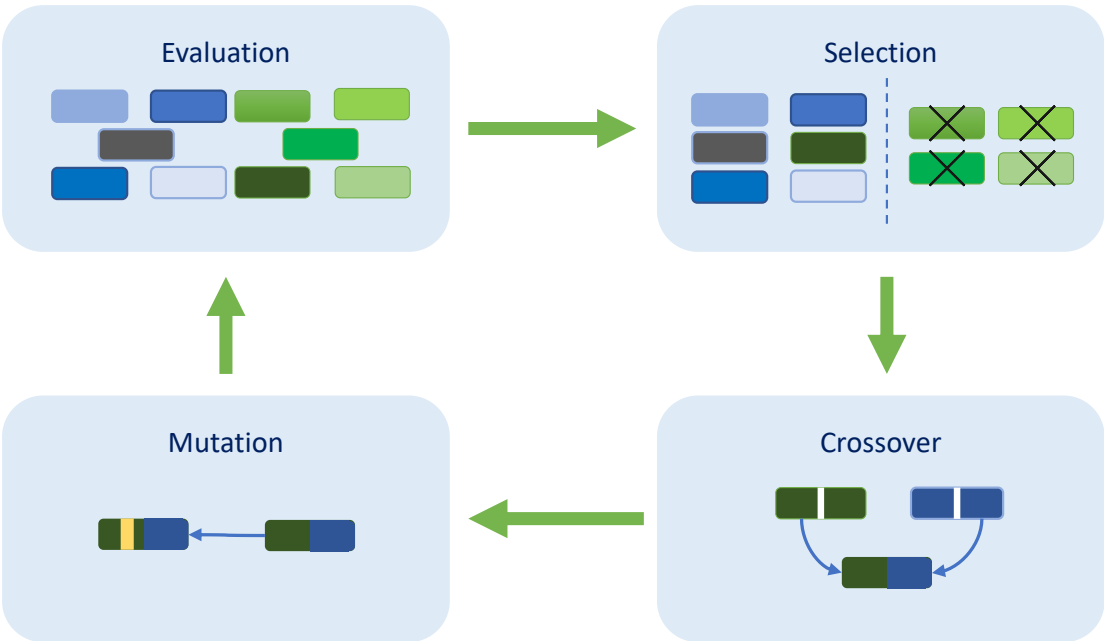


Figure A.6: Genetic Algorithm Representation.

Some combinations, usually the best ones, are directly copied from the current generation to the new generation, which is usually called elitism. Then the selection and other operations are recursively applied to the new generation of elements.

Typically, members of each new generation are on average more fit than the members of the previous generation, and the best-of-generation element is often better than the best-of-generation elements from previous generations. Termination of the recursion is when some individual element reaches a predefined proficiency or fitness level. A branch of Genetic algorithms are considered to be evolutionary bio-inspired, such as Genetic Bee Colony Algorithm (GBCA), Fish Swarm Algorithm (FSA), Cat Swarm Optimization (CSO), Whale Optimization Algorithm (WOA), Artificial Algae Algorithm (AAA), Elephant Search Algorithm (ESA), Chicken Swarm Optimization Algorithm (CSOA), Moth Flame Optimization (MFO) and Grey Wolf Optimization (GWO) (Darwish, 2018).

It can be considered that the main advantages of genetic algorithms are:

- It can find fit solutions in less time. (fit solutions are solutions which are good according to the defined heuristic).
- The random mutation guarantees to some extent that a wide range of solutions is generated.
- Coding them is really easy compared to other algorithms.

On the other hand, the drawbacks of genetic algorithms are:

- It is really hard for people to come up with a good heuristic which actually reflects what the algorithm should do.
- It might not find the most optimal solution to the defined problem in all cases.
- Its also hard to choose parameters like number of generations, population size or stopping condition. When the model is being worked, even though the heuristic was right, it might be hard to realize it because it's running for a few generations.

The bayesians come from statistics and most of their algorithms are extensions and reformulations of the equation A.1. In probability theory and statistics, Bayes' theorem describes the probability of an event, based on a priori knowledge that may be related to the event. The theorem shows how to change a priori probabilities in view of new evidence to obtain a posteriori probabilities.

The core stone is the equation A.1 (Kemp et al., 1994), on which A and B are events, $P(A|B)$ is a conditional probability of the likelihood of event A occurring given that B is true, $P(B|A)$ is the conditional probability of the likelihood of event B occurring given that A is true and finally $P(A)$ and $P(B)$ are the probabilities of observing A and B independently of each other. This is known as the marginal probability.

$$\begin{cases} P(A|B) = P(A)\frac{P(B|A)}{P(B)} \\ P(B) \neq 0 \end{cases}, \tag{A.1}$$

Some advantages to using Bayesian analysis include the following:

- It provides a natural and principled way of combining prior information with data, within a solid decision theoretical framework. You can incorporate past information about a parameter and form a prior distribution for future analysis. When new observations become available, the previous posterior distribution can be used as a prior. All inferences logically follow from Bayes' theorem.
- It provides inferences that are conditional on the data and are exact, without reliance on asymptotic approximation. Small sample inference proceeds in the same manner as of a larger dataset. Bayesian analysis also can estimate any functions of parameters directly, without using the "plug-in"method (a way to estimate functionals by plugging the estimated parameters in the functionals).
- It obeys the likelihood principle. If two distinct sampling designs yield proportional likelihood functions for, then all inferences about should be identical from these two designs. Classical inference does not in general obey the likelihood principle.
- It provides a convenient setting for a wide range of models, such as hierarchical models and missing data problems. Markov Chain Monte Carlo (MCMC), along with other numerical methods, makes computations tractable for virtually all parametric models.

There are also disadvantages to using Bayesian analysis:

- It does not tell you how to select a prior. There is no correct way to choose a prior. Bayesian inferences require skills to translate subjective prior beliefs into a mathematically formulated prior. If you do not proceed with caution, you can generate misleading results.
- It can produce posterior distributions that are heavily influenced by the priors. From a practical point of view, it might sometimes be difficult to convince subject matter experts who do not agree with the validity of the chosen prior.
- It often comes with a high computational cost, especially in models with a large number of parameters. In addition, simulations provide slightly different answers unless the same random seed is used. Note that slight variations in simulation results do not contradict the early claim that Bayesian inferences are exact. The posterior distribution of a parameter is exact, given the likelihood function and the priors, while simulation-based estimates of posterior quantities can vary due to the random number generator used in the procedures.

The analogizers actually have influences from a lot of different fields, being psychology probably the most important to them. The core algorithm for the analogizers is the kernel machines as known as SVM (Cortes and Vapnik, 1995), as is exemplified at figure A.7.

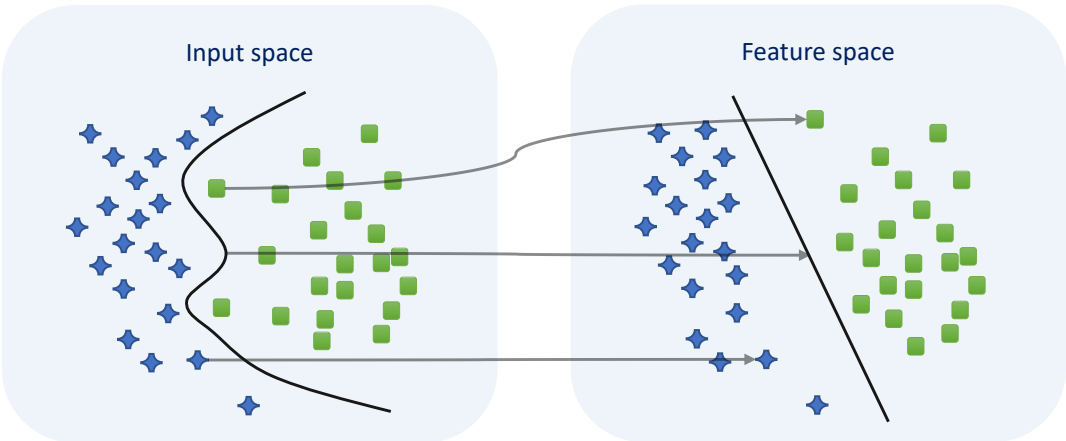


Figure A.7: Support Vector Machine Representation.

Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting).

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data is unlabeled, SL is not possible, and an UL approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups (Statnikov et al., 2011).

In general terms, the main advantages of SVMs can be described as:

- SVMs are very good when there's not much information about the working data.
- Works well with even unstructured and semi structured data like text, images and trees.
- The kernel trick is the major advantage of SVM. With an appropriate kernel function, it's possible to solve any complex problem, with few parameters.
- Unlike in neural networks, SVM is not solved for local optima.
- It scales relatively well to high dimensional data.
- SVM models have generalization in practice, the risk of over-fitting is less in SVM.

The main SVM disadvantages are (Cawley and Talbot, 2010):

- Choosing a "good" kernel function is not easy.
- Long training time for large datasets.
- Difficult to understand and interpret the final model, variable weights and individual impact.
- Since the final model is not so easy to see, small calibrations cannot be done to the model hence its tough to incorporate our business logic.

Finally, the last group of the ML cluster are the connectionists, which have origins in neuroscience, because they're trying to take inspiration from how the brain works. This is the cluster that will be further analyzed, and will give additional details in the next section. On figure A.8 a brief representation of the connectionists algorithm, a neural network.

In the seek for knowledge in SL (in particular on DL), it's explored ways to interconnect different types of DL architectures in order to solve yet unsolved problems, such as

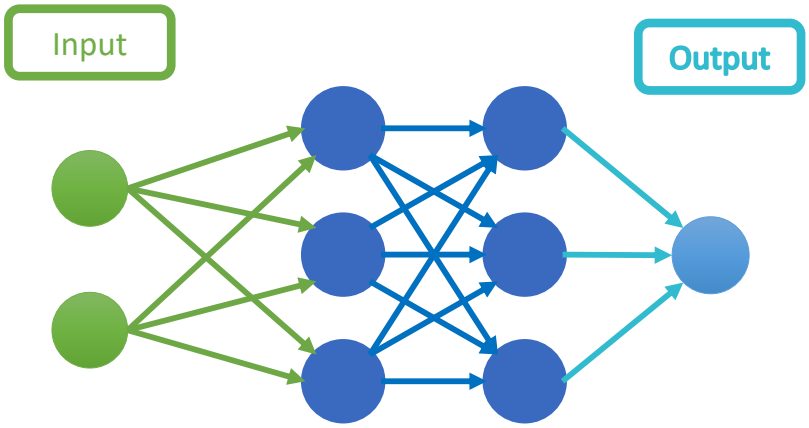


Figure A.8: Connectionists Representation.

video context awareness classifiers in collision detectors, data fusion and correlation, or even clinical future estimation.

Computer vision has become ubiquitous in our society, with applications in search, image understanding, apps, mapping, medicine, drones, and self-driving cars. Core to many of these applications are visual recognition tasks such as image classification, localization and detection. Recent developments in NN approaches have greatly advanced the performance of these SoA visual recognition systems. This next subsection is a deep dive into details of the DL architectures with a focus on learning end-to-end models and datasets for these tasks, particularly image classification. This part of the work, will give detailed resume about neural networks and gain a detailed understanding of cutting-edge research in computer vision that is later reused and applied to create our collision avoidance algorithm.

A.2 Deep Learning

ANNs were inspired by information processing and distributed communication nodes in biological systems. ANNs have several differences from biological brains. Specifically, neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plastic) and analog (Marblestone et al., 2016; Olshausen and Field, 1996; Scellier and Bengio, 2016).

The term DL was introduced to the ML community by Rina Dechter in 1986, (Dechter, 1986; Schmidhuber, 2015) and to artificial NN by Igor Aizenberg et. al in 2000, in the context of Boolean threshold neurons (Aizenberg et al., 2001; Gomez and Schmidhuber, 2005).

DL is part of a broader family of ML methods based on artificial NN (Lecun et al., 2015a; Schmidhuber, 2015).

The first general, working learning algorithm for supervised, deep, feedforward, multilayer perceptrons was published by Alexey Ivakhnenko and Lapa in 1965 (Ivakhnenko and Lapa, 1965). A 1971 paper described a deep network with 8 layers trained by the group method of data handling algorithm (Ivakhnenko, 1971).

The work on DL in computer vision was slightly hibernated, until in 1989, Yann LeCun et al. applied the standard backpropagation algorithm, which had been around as the reverse mode of automatic differentiation since 1970. The impact of DL in industry began in the early 2000s, when CNNs already processed an estimated 10% to 20% of all the checks written in the US (Lecun et al., 2015a).

DL architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases superior to human experts (Ahmad et al., 2019; Arel et al., 2010; Guo et al., 2016; Mousavi et al., 2018; Schmidhuber, 2015).

Modern CNNs are considered as one of the best techniques for learning image and video content showing SoA results on image recognition, segmentation, detection, and retrieval related tasks (Cireşan et al., 2012; Liu et al., 2019). The success of CNN has captured attention beyond academia. In industry, companies such as Google, Microsoft, AT&T, Facebook and PDM have developed active research groups for exploring new architectures of CNN (Deng and Yu, 2013).

In DL, each level learns to transform its input data into a slightly more abstract and composite representation. In an image recognition application, the raw input is a matrix of pixels, where the first representational layer may abstract the pixels and encode edges, the second layer may compose and encode arrangements of edges, the third layer may encode a nose and eyes and the fourth layer may recognize that the image contains a face. Moreover, a DL process can learn which features to optimally place in which level on its own (Bengio et al., 2013; Lecun et al., 2015b).

The *deep* in *deep learning* refers to the number of layers through which the data is transformed. More precisely, DL systems have a substantial Credit Assignment Path (CAP) depth. The CAP is the chain of transformations from input to output. CAPs describe potentially causal connections between input and output. For a feedforward neural network, the depth of the CAPs is that of the network and is the number of hidden layers plus one (as the output layer is also parameterized). For recurrent neural networks, in which a signal may propagate through a layer more than once, the depth is potentially unlimited (Schmidhuber, 2015). No universally agreed upon threshold of depth divides shallow learning from DL, but most researchers agree that DL involves $depth > 2$. CAP of depth 2 has been shown to be a universal approximator in the sense that it can emulate any function (Hinton et al., 2006). Beyond that more layers do not add to the function approximator ability of the network. Deep models are able to extract better features than

shallow models and hence, extra layers help in learning features.

The DL architectures are often constructed with more layers then the necessary, which helps to disentangle these abstractions and pick out which features improve performance.

CNN topology is divided into multiple learning stages composed of a combination of the convolutional layer, non-linear processing units, and subsampling layers (Jarrett et al., 2009). As shown in Figure A.9, the architecture of a typical CNN model is structured as a series of layers. Each layer performs multiple transformations using a bank of convolutional kernels (filters) (LeCun et al., 2010). All the components involved in such architecture will be later described in section A.3. Convolution operation extracts locally correlated features by dividing the image into small slices (similar to the retina of the human eye), making it capable of learning suitable features. Output of the convolutional kernels is assigned to non-linear processing units, which not only helps in learning abstraction but also embeds non-linearity in the feature space. The non-linearity outputs different patterns of activations for different responses, which facilitates the learning of semantic in different images. This is usually followed by subsampling, which helps in compressing the results and also makes the input invariant to geometrical distortions (LeCun et al., 2010; Scherer et al., 2010).

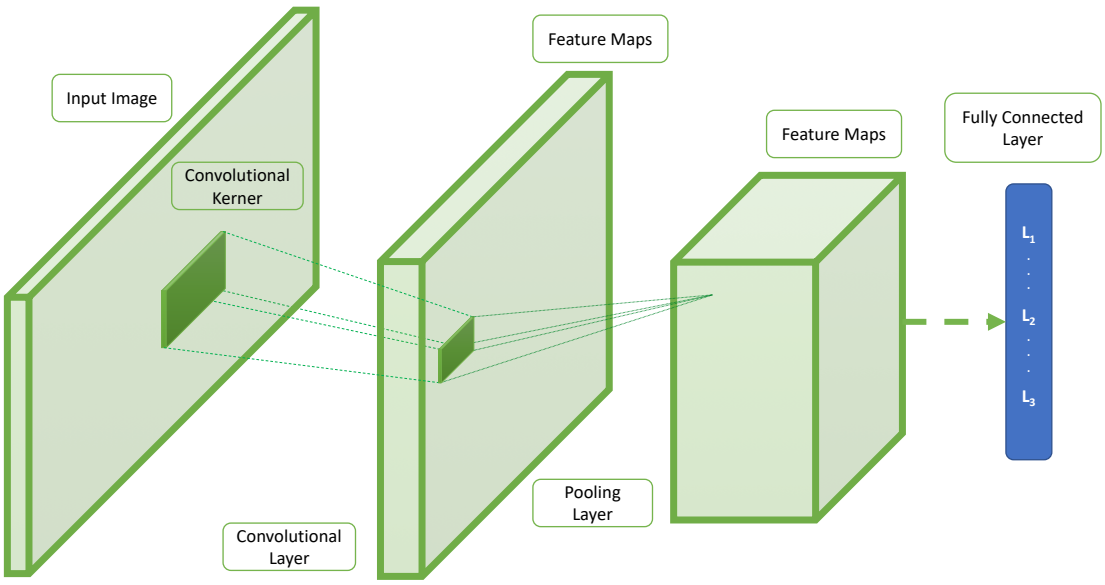


Figure A.9: The architecture of a standard Convolutional Neural Network model.

The work conducted by Hubel and Wiesel's (Hubel and Wiesel, 1962; Hubel and Wiesel, 1968) inspired the initial architectural designs of CNNs, following the basic structure of primate's visual cortex. As illustrated in figure A.10, the first steps can be considered in 1980, with the initial work in Neocognition like networks (Fukushima, 1980). Using this knowledge, Yann LeCun (LeCun et al., 1989) proposed a grid-like

topological data, which displayed the hierarchical feature extraction ability of CNNs.

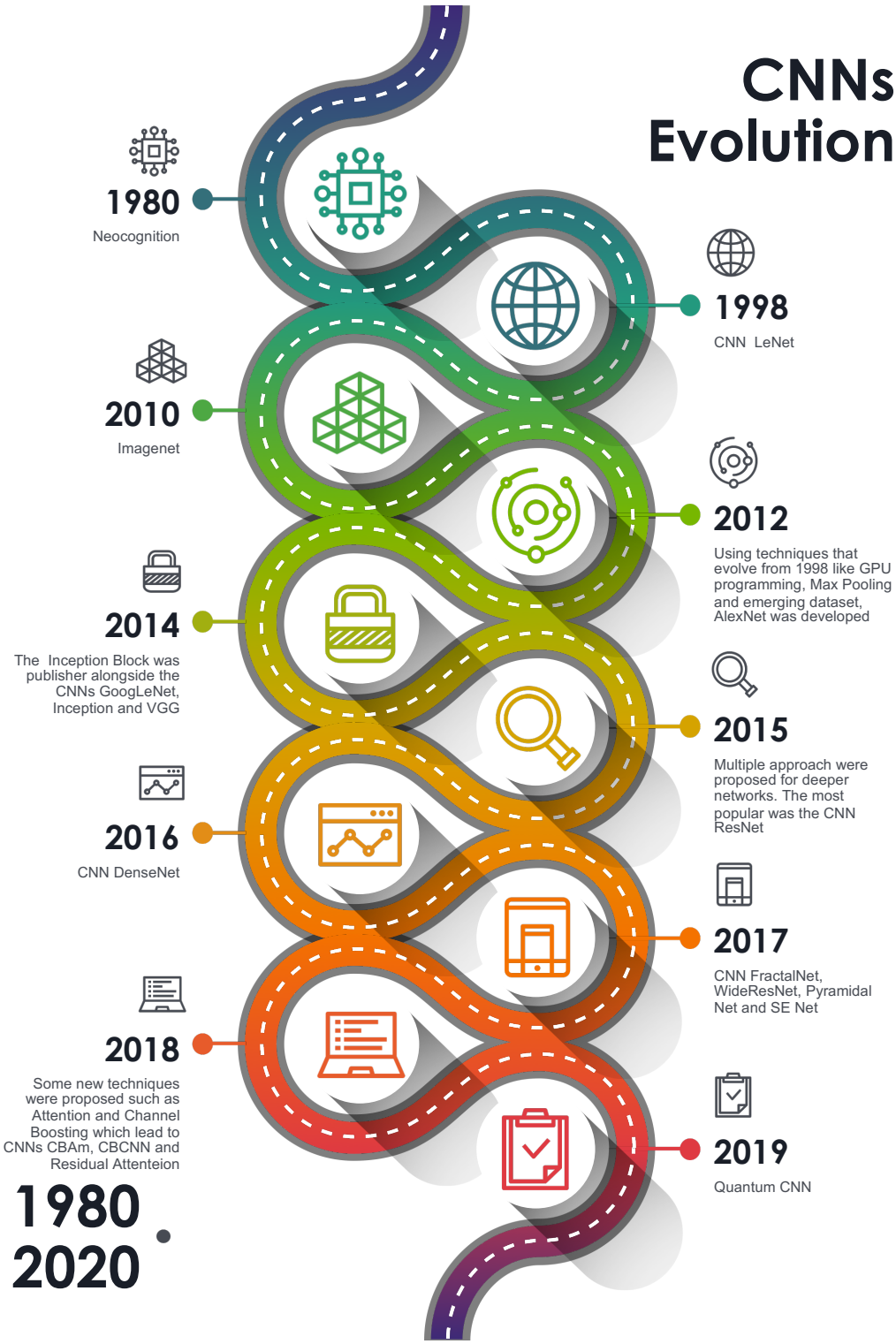


Figure A.10: Convolutional Neural Networks evolution over the years.

This hierarchical organization emulates the deep and layered learning process of the Neocortex in the human brain, which extract features from the underlying world (Bengio,

2009). The engineered process in CNN resemblance with V1-V2-V4-IT/VTC primate's ventral pathway of visual cortex (Laskar et al., 2018). The retinotopic area provide input to primates visual cortex, where contrast normalization and multi-scale highpass filtering is performed by the lateral geniculate nucleus. Afterwards, different regions of the visual cortex categorized as V1, V2, V3, and V4 classify and detect information. The V1 and V2 areas of the visual cortex can be imagined as the convolutional, and subsampling layers, whereas inferior temporal region are similar to the final layers of CNN, which makes inference about the image (Grill-Spector et al., 2018).

CNN training is similar to standard NN, where the weights are regulated with backpropagation algorithm, iterating over multiple input images. In backpropagation, the objective is to minimize a cost function, similar to the response based learning of human brain(Najafabadi et al., 2015).

The revolution of the use of CNNs for image understating and segmentation occurred when it was discover that the results could be improved by tweaking with layers depth (Krizhevsky et al., 2017). Deep CNN architectures have advantage over shallow architectures when dealing complex learning problems. Using multiple linear and non-linear neurons in a layer wise mode, enhances this deep networks with the ability to learn representations at different levels of abstraction. Additionally, advances in hardware enabled the renewed the interest. In 2009, Nvidia was involved in what was called the big bang of deep learning, as DNN were trained with Nvidia GPUs (Dixon, 2016). That year, Google Brain used Nvidia GPUs to create capable DNNs. While there, Andrew Ng determined that GPUs could increase the speed of DL systems by about 100 times (The Economist, 2010). In particular, GPUs are well-suited for the matrix/vector math involved in ML (Darmatasia and Fanany, 2017; Oh and Jung, 2004). GPUs speed up training algorithms by orders of magnitude, reducing running times from weeks to days (Cireşan et al., 2010; Raina et al., 2009). Specialized hardware and algorithm optimizations can be used for efficient processing (Sze et al., 2017).

Significant additional impacts in image or object recognition were noticed from 2011 to 2012. Although CNNs trained by backpropagation had been around for decades, and GPU implementations of NNs for years, including CNNs, fast implementations of CNNs with max-pooling on GPUs in the style of Ciresan and colleagues were needed to progress on computer vision (Cireşan et al., 2011; LeCun et al., 2008; Oh and Jung, 2004).

Image classification was then extended to the more challenging task of generating descriptions (captions) for images, often as a combination of CNNs and LSTMs (Fang et al., 2015; Vinyals et al., 2015; Zhong et al., 2011).

Some researchers assess that the October 2012 ImageNet victory anchored the start of a deep learning revolution that has transformed the AI industry (Metz, 2016).

Multiple improvements in CNNs learning strategy and architectures have been presented to make CNNs scalable to large and complex problems. These innovations can be divided as regularization, structural reformulation, parameter optimization and computation efficiency. Major innovations in CNN have been proposed since 2012 and were

mainly due to restructuring of processing units and designing of new blocks. Zeiler and Fergus (Zeiler and Fergus, 2014) presented the concept of layer-wise visualization of features, which shifted the trend towards features extraction at low spatial resolution in deep architecture such as VGG (Simonyan and Zisserman, 2015). Currently, most of the new architectures are built upon the principle of simple and homogeneous topology as it was presented by VGG. However, Google group introduced an interesting idea of split, transform, and merge, which is known as an *inception block*. The inception block gave the concept of branching within a layer, which allows features abstraction at different spatial scales (Szegedy et al., 2015). In 2015, the concept *connections skips* was introduced by ResNet (He et al., 2016). Afterwards, this concept was used by most of the succeeding NN, such as Inception-ResNet, WideResNet and ResNext (Szegedy et al., 2017; Xie et al., 2017; Zagoruyko and Komodakis, 2016).

Towards the improvement of learning capacities of CNNs, different design such as WideResNet, Pyramidal Net, Xception have been proposed, exploring the effect of transformations of additional cardinality and increase in width (Han et al., 2017; Xie et al., 2017; Zagoruyko and Komodakis, 2016). The focus of research moved from parameter optimization and connections optimization towards improved architectural design (layer structure) of the network. This change resulted in many new architectural blocks such as channel boosting, spatial and channel wise exploitation and attention based information processing (Khan et al., 2018; Wang et al., 2017; Woo et al., 2018).

The overfitting problems are raised by the added layers of abstraction, which allow them to model rare dependencies in the training data. Regularization methods such as Ivakhnenko's unit pruning or weight decay or sparsity can be applied during training to combat overfitting (Bengio et al., 2013). Alternatively dropout regularization technique randomly omits neurons from the hidden layers during training. This helps to exclude rare dependencies (Dahl et al., 2013). Finally, data can be augmented via methods such as cropping and rotating such that smaller training sets can be increased in size to reduce the chances of overfitting, which will be detailed in A.4.

The learning computation time comes from the many training parameters of the standard DNNs, such as the size (number of layers and number of neurons per layer), the learning rate, and initial weights. Sweeping through the parameter space for optimal parameters may not be feasible due to the cost in time and computational resources. Various tricks, such as batching (computing the gradient on several training examples at once rather than individual examples) (Hinton, 2012) speed up computation. Large processing capabilities of many-core architectures (such as GPUs or the specialized CPUs such as Intel Xeon Phi) have produced significant speedups in training, because of the suitability of such processing architectures for the matrix and vector computations (Viebke et al., 2019; You et al., 2017).

In the recent years, many different surveys were conducted on CNNs that depicted and compared their basic components. The survey reported by Gu (Gu et al., 2018) has reviewed the famous models from 2012-2015 along with their core blocks. There are also

other similar surveys in literature that discuss different algorithms of CNN and focus on applications for demonstration of results (Guo et al., 2016; LeCun et al., 2010; Liu et al., 2017; Najafabadi et al., 2015; Srinivas et al., 2016). The following subsections of A.3 tried to aggregate this information in a concise, yet vast and wide explanation of the field, detailing building blocks, data sets and models.

A.3 Basic CNNs Building Blocks

For most of the perception applications, CNN is considered as the most widely used ML technique. A typical block diagram of an ML system was shown in Figure A.9. Since, SoA CNNs possesses both good feature extraction and strong discrimination ability, the most common task are feature extraction and classification.

The most common CNN architecture is composed of alternated layers of convolution and pooling followed by one or two fully connected layers at the end. In some cases, the fully connected layers are swapped with global average pooling layer. In addition to the various learning stages, different regulatory units, such as batch normalization and dropout are also incorporated to optimize CNN performance (Bouvrie, 2006). The structure of CNNs components play a fundamental role in new architectures designs and thus achieving enhanced performance. This subsection briefly describes and discusses the role of these components in CNN architecture.

A.3.1 Convolutional Layer

A convolutional layer (sometimes denominated conv layer) is composed of a set of convolutional kernels (where each neuron act as a kernel). These kernels are linked with a small area of the image known as a *receptive field*. The image is divided into small blocks (receptive fields) and convoluted with a specific set of weights (multiplying elements of the filter with the corresponding receptive field elements) (Bouvrie, 2006). This operation have similarities of a convolutional, but they are mathematically different. Convolution layer operation can expressed as follows:

$$C_l^k = P_{x,y}^k * K_l^k \tag{A.2}$$

On equation A.2, the input pixel of the image is represented by $P_{x,y}$, x, y shows spatial locality and K_l^k represents the lth convolutional kernel of the kth layer. Dividing the image into small blocks helps extracting local pixel correlations. Different set of features within the image are extracted by sliding convolutional kernel on the whole image with the same set of weights. This weight sharing on the kernels of convolution operation makes CNN parameters efficient when compared to fully connected NN. The convolution operation may further be categorized into different types based on the type and size of filters, type of padding, and the direction of convolution (Lecun et al., 2015a). If the kernel is symmetric, the convolution operation becomes a correlation operation (Goodfellow et al., 2016).

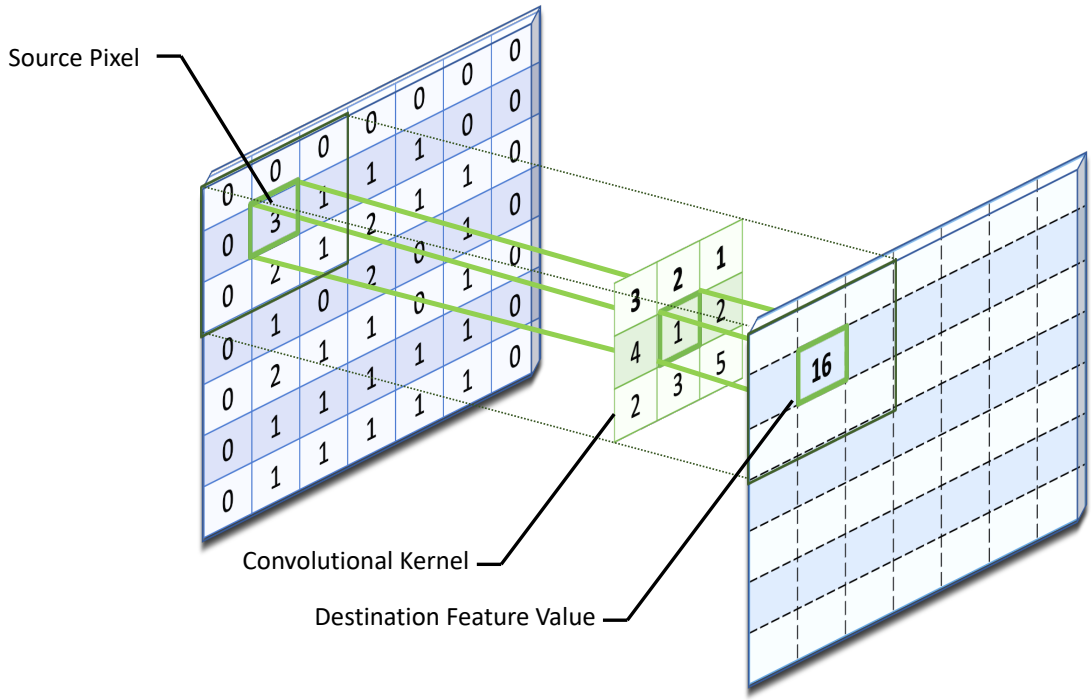


Figure A.11: Convolutional layer destination feature value calculation example.

On figure A.11 is represented an example of the kernel sliding over a source pixel. Initially, the center element of the kernel is placed over the source pixel. Afterwards, the destination pixel, C_l^k is then calculated with the weighted sum of itself and nearby pixels. On this example, the resulting destination feature value can be calculated as :

$$C_l^k = 0*3+0*2+0*1+0*4+3*1+1*2+0*2+2*3+1*5 = 16 \tag{A.3}$$

A.3.2 Pooling Layer

The convolution operation outputs feature maps. Once features values are calculated, its exact location becomes less important as long as its approximate position relative to others is preserved. Pooling or downsampling like convolution, is a local operation. It sums up similar information in the neighborhood of the receptive field and outputs the dominant response within this local region (Lee et al., 2018; Lee et al., 2016).

On equation A.4 is represented the pooling operation in which Z_l represents the lth output feature map, $C_{x,y}^l$ represents the lth input feature map, whereas $f_p(x)$ defines the type of pooling operation.

$$Z_l = f_p(C_{x,y}^l) \tag{A.4}$$

The use of pooling operation extracts a combination of features, which are invariant to translational shifts and distortions (Ranzato et al., 2007; Scherer et al., 2010). Reduction

in the size of feature map to invariant feature set not only reduces network complexity and also increases generalization by reducing overfitting. The most common types of pooling formulations are (He et al., 2015; Wang et al., 2012):

- Max pooling.
- Average pooling.
- L2 pooling.
- Overlapping pooling.
- Spatial pyramid pooling

A.3.3 Activation Function

On classification problems, activation functions are used as a decision function, helping to differentiate complex classes. The selection of the activation function can also accelerate the learning process. For CNNs, activation functions of the convolved feature map is defined in equation can be defined as:

$$T_l^k = f_A(C_l^k) \tag{A.5}$$

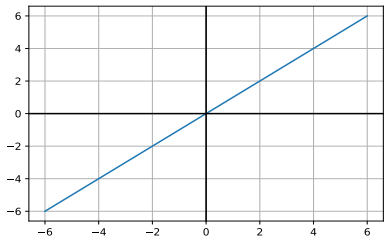
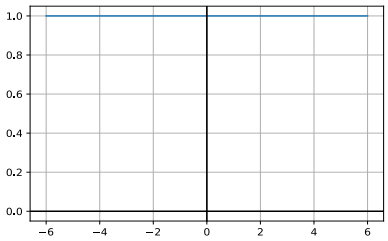
On equation A.5, C_l^k is the output of a convolution operation, which is mapped to an activation function $f_A(x)$. This activation function adds non-linearity and returns the resulting output T_l^k for k^{th} layer. In academia, different activation functions such as sigmoid, tanh, maxout, ReLU, and variants of ReLU such as leaky ReLU, Exponential Linear Unit (ELU), and Parametric ReLU (PReLU) (LeCun et al., 2012; Wang et al., 2017; Wang et al., 2012; Xu et al., 2015), are used to inculcate nonlinear combination of features. However, ReLU and its variants are preferred over others activations as it helps in overcoming the vanishing gradient problem (Hochreiter, 1998; Nwankpa et al., 2020).

Many improvements to the learning progress were only possible due to the research of new activation functions. The backpropagation this functions derivatives, so it's also important to have a clear idea of the activation functions derivatives, because backpropagation is a leaky abstraction (it might use a credit assignment scheme with non-trivial consequences).

A.3.4 Linear

The linear activation function, as described in table A.2, is the most basic activation function. It can be seen as a straight line function where activation is proportional to input (which is the weighted sum from neuron). For the derivative graph, a value of $m = 1$ was considered.

Table A.2: Activation Function Linear resume.

	Function	Derivative
Formula	$R(z,m) = z * m$	$R'(z,m) = m$
Graph		
Python code	`def linear(z,m):` `return m*z`	`def linear_der(z,m):` `return (m * z ) / z`

The main advantages of using a linear activation function can be described as:

- It gives a linear value, for range of activations, which can be used in both regression and classification.
- It's possible to utilize multiple neurons together, and do simple classifications afterwards, such as considering the max value fired.

On the other hand, the linear activation function as some disadvantages, such as:

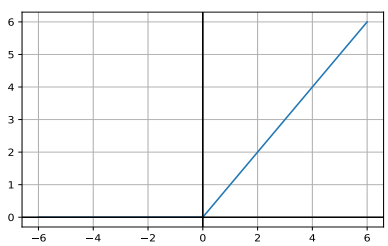
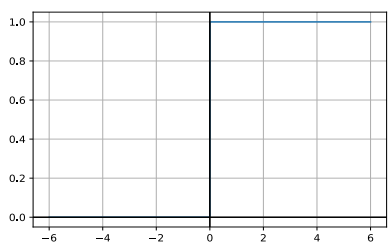
- The derivative is a constant. This has a negative impact on the backpropagation, because the gradient has no relationship with x.
- It's not possible to utilize gradient descent for leaning, because it's going to be on constant gradient.

A.3.5 ReLU

ReLU is the most used activation function in nowadays applications, mainly because the formula is deceptively simple: $max(0, z)$. Despite its name and appearance, it's not linear and provides the same benefits as the traditional Sigmoid but with better performance due to it's computational simplicity. This activation function has been summarized in table A.3.

The advantages of using ReLU are quite trivial to understand, but it was a big breakout on the CNNs (Wang et al., 2017; Wang et al., 2012; Xu et al., 2015). The main ones can be considered as:

Table A.3: Activation Function ReLU resume.

	Function	**Derivative**
Formula	$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix}$	$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ 0 & z < 0 \end{Bmatrix}$
Graph		
Python code	`def relu(z):` `return np.where(z >= 0, z, 0)`	`def relu_der(z):` `return np.where(z >= 0, 1, 0)`

- It avoids and rectifies vanishing gradient problem that were present on the antecedent activation functions.
- It is less computationally expensive than the tanh and sigmoid because it involves simpler mathematical operations.

Due to its popularity, several researchers have detected some disadvantages in this technique, and have proposed alternative versions (Wang et al., 2017; Wang et al., 2012; Xu et al., 2015). Some of these disadvantages are:

- The range of ReLU is $[0, \infty]$. This means it has no positive boundary, which makes the classification problem harder, and can force the CNN to overshot.
- It should only be used within Hidden layers of a Neural Network Model. There's no advantage of cropping the negative values in the output layer.
- Some gradients can be fragile during training and get discarded. Usually, when this happens, the neuron will update the weights to values which produce negative x results. When this values are passed to the ReLU it will always returns 0, and due to it's derivative, it is never again updated, which can be considered a 'dead neuron'.
- In another words, f activations in the region ($x < 0$) of ReLU, gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons which go into that state will stop responding to variations in error/ input (simply because gradient is 0, nothing changes). This is called dying ReLU problem. Some studies conducted to SoA CNN realized that in many architectures, more then 90% of the network is composed of 'dead neuron' (Han et al., 2015).

A.3.6 ELU

The activation function ELU usually converges to zero in a few epoch, which generate fast training and produce more accurate results. Different to other activation functions, ELU uses an α constant which needs to be positive number. As analyzed in section A.3.5, ELU is similar to ReLU, except on the negative inputs region. Both functions are a identity function for positive inputs. On the other hand, ELU becomes smooth slowly until its output equal to $-\alpha$ whereas ReLU swaps to 0 instantaneously, as presented in table A.4.

Some of the benefits of ELU are (Xu et al., 2015):

- ELU becomes smooth slowly until its output equal to $-\alpha$ whereas ReLU sharply smoothes.
- ELU is a strong alternative to ReLU.
- Unlike to ReLU, ELU can produce negative outputs.

Nonetheless, for $x > 0$, the ELU activation function can also start overshot with the output range of $[0, \infty]$ (Xu et al., 2015).

A.3.7 LeakyReLU

LeakyRelu is yet another variant of ReLU. Instead of being 0 when $z < 0$, a leaky ReLU allows a narrow, non-zero, constant gradient α (usually the value $\alpha = 0.01$ is considered). However, the consistency of the benefit across tasks is presently unclear. This activation function has been summarized in table A.5. Even thought that usually the value $\alpha = 0.01$ is considered, for a better graphical representation of the 'leaking' property, a value of $\alpha = 0.1$ has been considered.

Leaky ReLUs are a clear attempt to fix the dead neurons problems of ReLU. By having a narrow negative slope, it allow the gradient to always have an opportunity to train the network, and the possibility to tweak the weights to place it in the positive x region. Nevertheless, it possess linearity, so it shouldn't be used for the classification tasks (Wang et al., 2017; Wang et al., 2012).

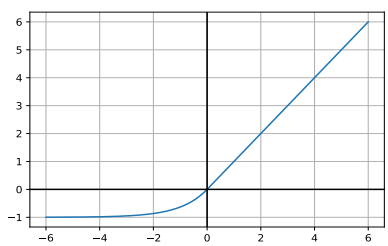
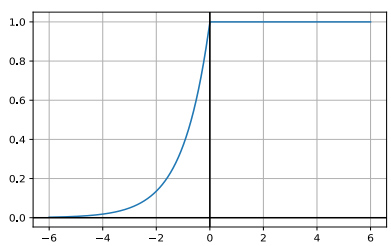
A.3.8 Sigmoid

The Sigmoid activation function receives as input a real value and outputs a value between 0 and 1, as can extrapolated from table A.6. This makes it easy to apply because it contains the most desired proprieties for an activation function. It's non-linear, continuously differentiable, monotonic, and has a well defined output range (LeCun et al., 2012).

The main advantages of the sigmoid function can be described as (LeCun et al., 2012):

- It is nonlinear function, which if combined multiple times, represents a complex output space easier then a linear function.

Table A.4: Activation Function ELU resume.

	Function	**Derivative**
Formula	$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha * (e^z - 1) & z <= 0 \end{Bmatrix}$	$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha * e^z & z <= 0 \end{Bmatrix}$
Graph		

Python code	`def elu(z,alpha):`
	`return np.where(z >= 0, z,↩`
	`↪ alpha*(np.exp(z) -1))`
		`def elu_der(z,alpha):`
		`return np.where(z >= 0, 1,↩`
		`↪ alpha*np.exp(z))`

Table A.5: Activation Function LeakyReLU resume.

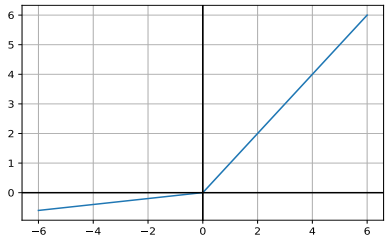
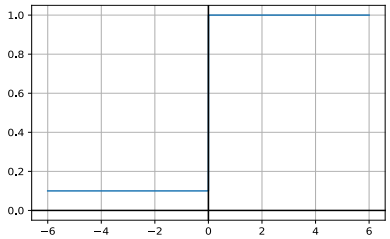
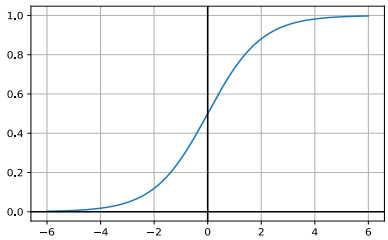
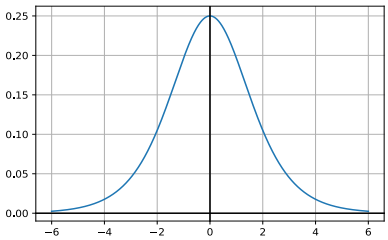
	Function	Derivative
Formula	$R(z) = \begin{cases} z & z > 0 \\ \alpha * z & z <= 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ \alpha & z < 0 \end{cases}$
Graph		
Python code	`def leakyrelu(z, alpha):` `return np.where(z >= 0, z, alpha * z)`	`def leakyrelu_der(z, alpha):` `return np.where(z>=0, 1, alpha)`

Table A.6: Activation Function Sigmoid resume.

	Function	Derivative
Formula	$S(z) = \frac{1}{1+e^{-z}}$	$S'(z) = S(z) \cdot (1 - S(z))$
Graph		
Python code	`def sigmoid(z):` `return 1.0 / (1 + np.exp(-z))`	`def sigmoid_der(z):` `return sigmoid(z)*(1-sigmoid(z))`

- Produces an analog activation with step function reassembly.
- It has a smooth gradient.
- The step like shape, gives good results in classification applications.
- The output of the activation function is always going to be in range $[0,1]$ compared to $[-\infty,\infty]$ of linear like function. This prevents the output from overshooting.

On the other hand, some disadvantages have been identified by researchers (LeCun et al., 2012), in concrete:

- At the extremes of the sigmoid function, the y values fluctuations are ignored in the X response.
- Also, on the extremes, the gradient tend to 0, which generates the problem of vanishing gradients (Hochreiter, 1998).
- Optimization is not trivial, because the output is not zero centered. On the zero region, the gradient is higher which make the updates flow in different directions.
- Random weight initialization, can make the network to refuse to learn drastically slow.

A.3.9 Tanh

Tanh activation function is similar to sigmoid, but with the output zero centered, as it is presented in table A.7. Usually tanh is prefered over sigmoid, due to it's center (Kalman and Kwasny, 1992; Xu et al., 2016).

Kalman calculated a function based on tanh (Kalman and Kwasny, 1992). On his study, he concluded that for deep networks, the gradient is stronger for tanh than sigmoid (due to the derivatives being steeper), which leads to a faster inference. Nonetheless, both sigmoids and tanh don't address the vanishing gradient problem.

A.3.10 Softmax

Finally, the last activation function this dissertation will look into is the Softmax. It calculates the probabilities distribution of the event over N different events, where the N is the size of the output array. This function is a quite different from the previous presented in it's conception. The probabilities of each target class over all possible target classes is calculated utilizing an N dimensional vector of arbitrary real values and producing another N dimensional vector with real values in the range $[0,1]$ that add up to 1.0. This is demonstrated in equation A.6.

$$S(z): \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{bmatrix} \rightarrow \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_N \end{bmatrix} \tag{A.6}$$

Since these output are already a probabilities from $[0,1]$, the results can be directly mapped to target classes, and the training not only maximize a value, but also maximize the disparity of triggers.

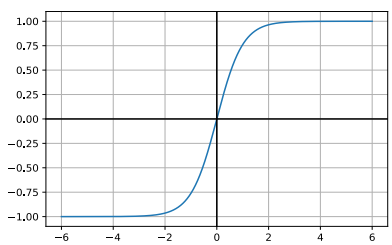
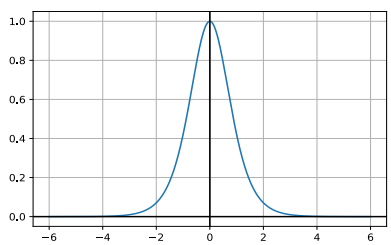
From a mathematical point of view, where the rest of the functions could be analyzed from a escalar view, softmax is fundamentally a vector function. It takes a vector as input and produces a vector as output. In other words, it has multiple inputs and multiple outputs. Therefore, it's not possible to represent 'the derivative of softmax'. For this reason, in this activation function is presented more detail.

Since softmax has multiple inputs, with respect to which input element the partial derivative should be computed. Thus, it's necessary to find the partial derivatives:

$$\frac{\partial S_i}{\partial z_j} \tag{A.7}$$

This is the partial derivative of the ith output with respect to the jth input. A shorter way to write the partial derivative that will be used going forward is $D_j S_i$. Since softmax is a $\mathbb{R}^N \rightarrow \mathbb{R}^N$ function, the most general derivative computed for it is the Jacobian matrix:

Table A.7: Activation Function Tanh resume.

	Function	**Derivative**
Formula	$T(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$T'(z) = 1 - T(z)^2$
Graph		
Python code	`def tanh(z):` `return (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))`	`def tanh_der(z):` `return 1 - np.power(tanh(z), 2)`

$$DS = \begin{bmatrix} D_1 S_1 & \cdots & D_N S_1 \\ \vdots & \ddots & \vdots \\ D_1 S_N & \cdots & D_N S_N \end{bmatrix} \tag{A.8}$$

Computing for arbitrary i and j:

$$D_j S_i = \frac{\partial S_i}{\partial z_j} = \frac{\partial \frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}}}{\partial z_j} \tag{A.9}$$

Note that no matter which z_j is yielded, the derivative of the denominator $\sum_{k=1}^{N} e^{z_k}$, will always yell e^{z_j}. This is not the case for numerator e^{z_i}. The derivative of e^{z_i} with respect to z_j is e^{z_j} only if $i = j$, because only then e^{z_i} has z_j anywhere in it. Otherwise, the derivative is 0.

In result, it's obtained that $D_j S_i$ can be calculated by:

$$D_j S_i = \begin{Bmatrix} S_i(1 - S_j) & i = j \\ -S_i S_j & i \neq j \end{Bmatrix} \tag{A.10}$$

In ML literature, the term gradient is commonly used to stand in for the derivative. Gradients are only defined for scalar functions (such as the functions described in the previous sections). For vector functions like softmax it's imprecise to present it as a gradient. The Jacobian is the fully general derivate of a vector function. Nevertheless, for the sake of coherence, a resume table A.8 is presented. Keep in mind that both the graphical representation and direct code derivative in Python, the results don't express much importance. For being a vectorial function, the resulting value $S(z)$ for a given z will be highly depend of the number N of the the z array, that for this representation 60 points from $[-6, 6]$ were considered. In practical applications the Jacobian Matrix is calculated, and for graphical representation, the probability of the target class is usually preferred.

The basic practical difference between Sigmoid and Softmax is that while both give output in $[0, 1]$ range, softmax ensures that the sum of outputs along channels (as per specified dimension) is always 1, which enables them to be directly mapped to classes probabilities estimation. Sigmoid just makes outputs between $[0, 1]$.

Hence, if a one hot encoding scheme is being used, where one channel has probabilities of one class and other channel has probabilities of another, then Softmax activation is preferred.

A.3.11 Batch Normalization

Batch normalization is used to address the issues related to internal covariance shift within feature maps. The internal covariance shift is a change in the distribution of hidden units' values, which slow down the convergence (by forcing learning rate to

small value) and requires careful initialization of parameters. Batch normalization for a transformed feature map T_l^k can be represented as:

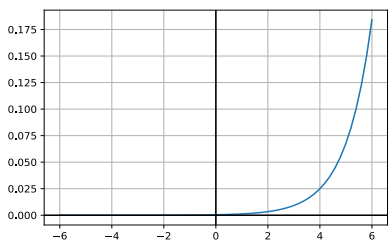
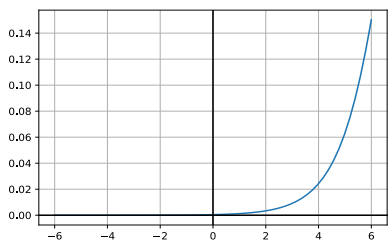
$$N_l^k = \frac{C_l^k - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \tag{A.11}$$

In equation A.11, N_l^k represents normalized feature map, C_l^k is the input feature map, μ_B is the mean and $sigma_B^2$ depict the variance of a feature map for a mini batch respectively. Batch normalization unifies the distribution of feature map values by bringing them to zero mean and unitary variance (Ioffe and Szegedy, 2015). Furthermore, it smooths the flow of gradient and acts as a regulating factor, which thus helps in improving generalization of the network.

A.3.12 Dropout

The Dropout technique introduces regularization in the network, which ultimately reduces overfitting by randomly skipping some units or connections with a certain probability. In DNNs, multiple connections that learn a non-linear relation are sometimes co-adapted, which reduces generalization (Hinton et al., 2012). This random dropping of some connections or units force all neurons to be utilized, by making thinned network architectures trains, and finally one representative network with all weights. This selected architecture is then considered as an approximation of all of the proposed networks (Srivastava et al., 2014).

Table A.8: Activation Function Softmax resume.

	Function	**Derivative**
Formula	$S(z_i) = \frac{e^{z_i}}{\sum_1^j e^{z_j}}$	$S'(z_i) = \begin{Bmatrix} S_i * (1 - S_j) & i = j \\ -S_i * S_j & i \neq j \end{Bmatrix}$
Graph		

Python code	`def softmax(x):`
	`return np.exp(x) / np.sum(np.exp(x), axis=0)`
		`def softmax_der(x):`
		`sm = softmax(x)`
		`return sm * (1 - sm)`

A.3.13 Fully Connected Layer

Fully connected layers are used at the end of the networks for classification or regression purposes. It takes input from the previous layer and globally analyses output of all the preceding layers (Lin et al., 2014a). This makes a non-linear combination of selected features, which are used for the classification of data (Rawat and Wang, 2017). For being a process that crosses all values, the number of operations and weights involved usually surpasses the rest of the entire network.

A.4 Data Augmentation

Data augmentation is an effective technique for improving the accuracy of CNNs (Shorten and Khoshgoftaar, 2019). Usually Data Augmentations uses transformations such as flipping, color space augmentations, and random cropping. These transformations encode many of the invariance that present challenges to image recognition tasks. Some more advance data augmentations techniques are GAN-based augmentation, neural style transfer, and meta-learning schemes (DeVries and Taylor, 2019; Konno and Iwazume, 2018). This section will explain how the common augmentation algorithms works, illustrate experimental results, and discuss disadvantages of the augmentation technique.

Some frameworks such as Keras (Keras, 2019) provide ways to perform Data Augmentation on the fly, rather than performing the operations on your entire image dataset in memory. The API is designed to be iterated by the deep learning model training process, creating augmented image data for the algorithm on run-time. This reduces memory overhead, but adds some additional computation during model training, which result in a longer training time.

In Keras, the Image Data Generator (IDG) calculate the statistics required to actually perform the transforms to the image data. The data generator itself is in fact an iterator, returning batches of image samples when requested. In the most used ML frameworks, when data augmentation is applied, instead of calling the fit function on the model, it's necessary to call the fit generator function and pass in a IDG with the desired length of an epoch, as well as the total number of epochs on which to train.

The MNIST dataset (LeCun et al., 1998) was used in order to have a common set of example images. On figure A.12 a set of nine images is represented to have a base of comparison for Image Augmentation algorithms.

A.4.1 Feature Standardization

Standardization typically means data rescaling, in order to have a mean of $\mu = 0$ and a standard deviation of $\sigma = 1$ (unit variance). Feature Standardization allows to normalize pixel values across an entire dataset. It mirrors the type of standardization often performed for each column in tabular dataset (Shen et al., 2016a). Usually this is done by performing the equation A.12:

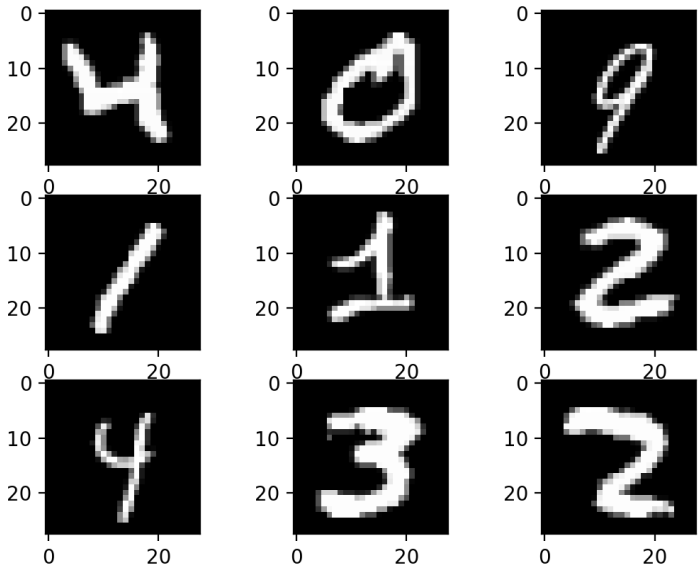


Figure A.12: Point Of Comparison.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{A.12}$$

On Keras framework, this is achieved by setting the feature-wise center and feature-wise standard normalization arguments on the IDG class. Applying feature standardization on the images of figure A.12, it's possible to achieve the result represented on figure A.13, which result in images seemingly darkening and lightning different digits.

A.4.2 ZCA Whitening

A whitening transform of an image is a linear algebra operation that reduces the redundancy in the matrix of pixel images. Less redundancy in the image is intended to better highlight the structures and features in the image to the learning algorithm (Li et al., 2015).

Considering N data point in $\mathbb{R}^n$, the covariance matrix is $\Sigma \in \mathbb{R}^{n \times n}$ estimated to be:

$$\hat{\Sigma}_{jk} = \frac{1}{N-1} \sum_{i=1}^{N} (x_{ij} - \bar{x}_j) \cdot (x_{ik} - \bar{x}_k) \tag{A.13}$$

In equation A.13, $\bar{x}_j$ denotes the jth component of the estimated mean of the samples x. Any matrix $W \in \mathbb{R}^{n \times n}$ which satisfies the condition $W^T W = C^{-1}$ whitens the data. Typically, image whitening is performed using the Principal Component Analysis (PCA) technique. More recently, an alternative called Zero-phase Component Analysis (ZCA) shows better results and results in transformed images that keeps all of the original

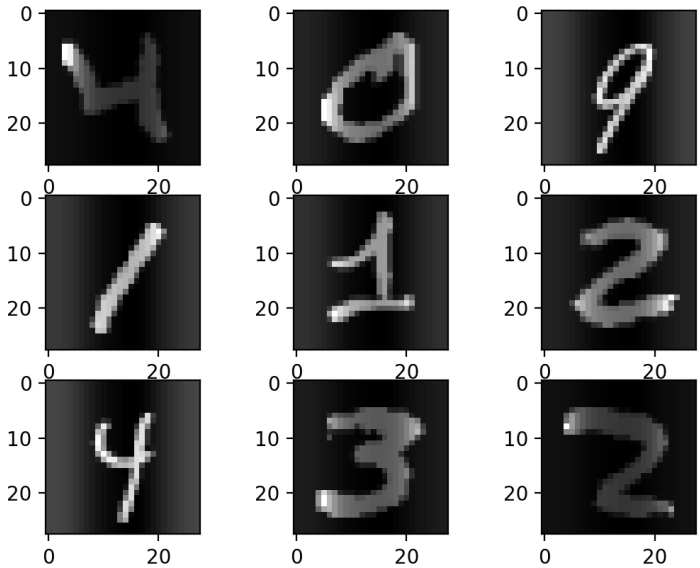


Figure A.13: Feature Standardization.

dimensions and unlike PCA, resulting transformed images still look like their originals. To execute a ZCA $W = M^{-\frac{1}{2}}$.

Using a ZCA Whitening transform on the sample images, the same general structure is maintained and how the outline of each digit is highlighted, as illustrated on A.14.

A.4.3 Random Shifts

Objects in images may not be centered in the frame. They may be off-center in a variety of different ways. To solve this problem during training, a common technique is to train the deep learning networks to expect and handle off-center objects by artificially creating shifted versions of the training data. For example, Keras and Tensorflow supports separate horizontal and vertical random shifting of training data by the width shift range and height shift range arguments.

Running this example creates shifted versions of the digits, as represented on A.15. Again, this is not required for MNIST as the handwritten digits are already centered, but it is useful on more complex problem domains.

A.4.4 Random Flips

Another image data augmentation technique that improves the performance is to randomly flip the training images. On figure A.16 it can be seen it's result over the sample images. On this example (MNIST dataset), flipping digits is not useful as they require the correct left and right orientation, but this may be useful for images of objects in a scene that can have different orientation.

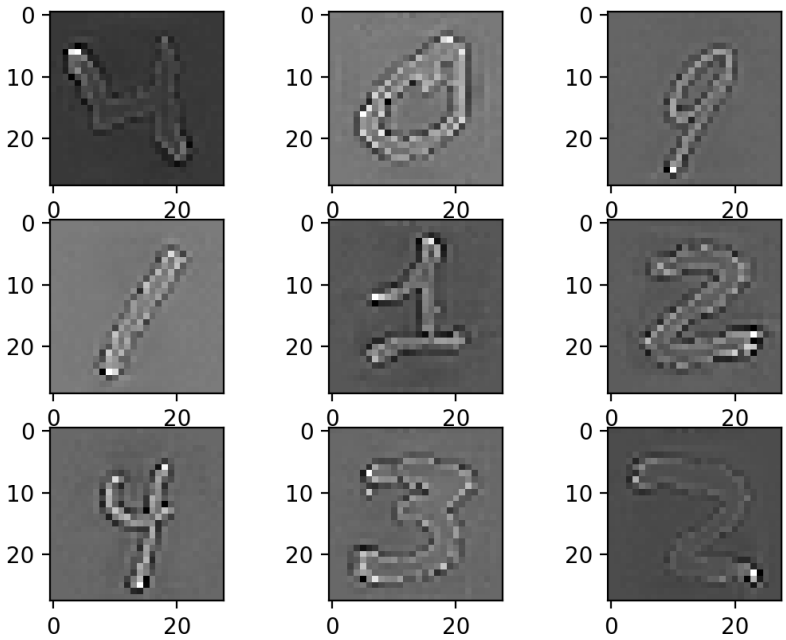


Figure A.14: ZCA Whitening.

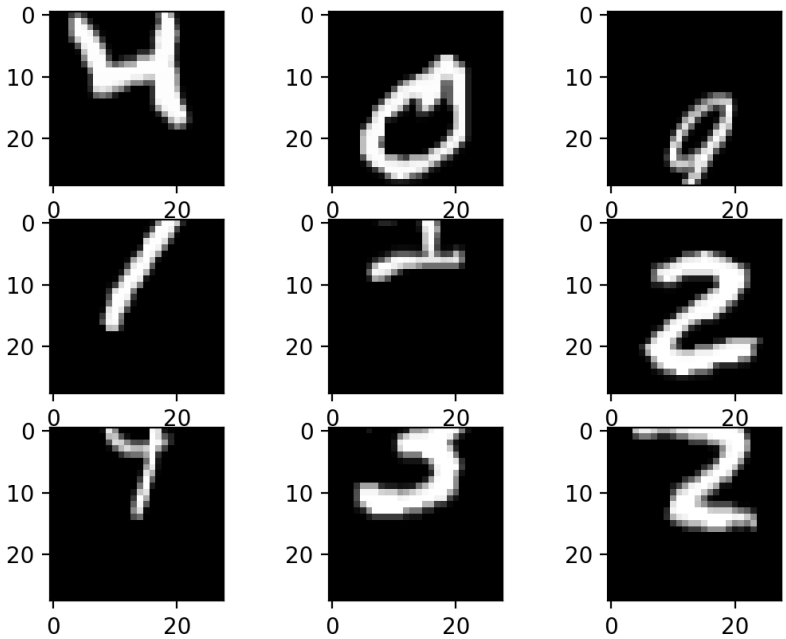


Figure A.15: Random Shifts.

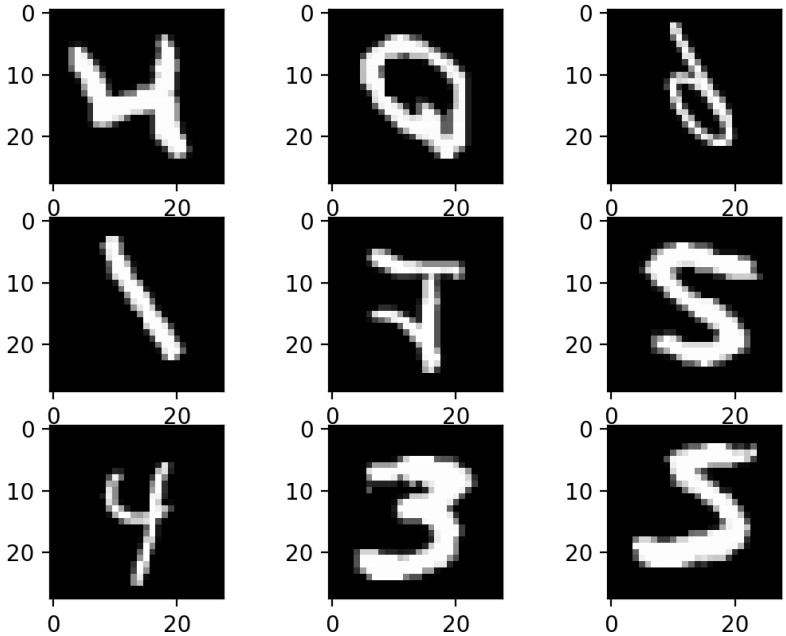


Figure A.16: Random Flips.

A.4.5 Random Rotations

Sometimes images in the dataset may have different rotations in the scene. In those cases, it's helpful to train the model capable of handling images rotations by artificially and randomly rotating images from the dataset during training.

As seen on figure A.17 the images have been rotated left and right up to a limit of 180 degrees. This is not helpful on this problem because the MNIST digits have a normalized orientation, but this transform might be of help when learning from photographs where the objects may have different orientations. Not only that but it might to some incorrect labeling. For example, the digit 9 at the top right corner is transformed into a 6 but will remain labeled as a 9 possibly leading to a worse model.

A.4.6 Additional Augmentations

When doing runtime data augmentations it's important not to use multiple techniques without a clear idea of the augmented results. As an example of this, it can be observed in figure A.18 where it was applied random shifts, ZCA whitening, standard normalization, random flips and zoom (between $\left[\frac{1}{2}, 2\right]$). It's questionable if the data represented after augmentation is valid, or if require the model to learn that the number 2 is a black square (bottom right image).

Additionally, some common data augmentations techniques are the rescaling and filling mode. Both this methods are usually applied after the rest of data augmentations

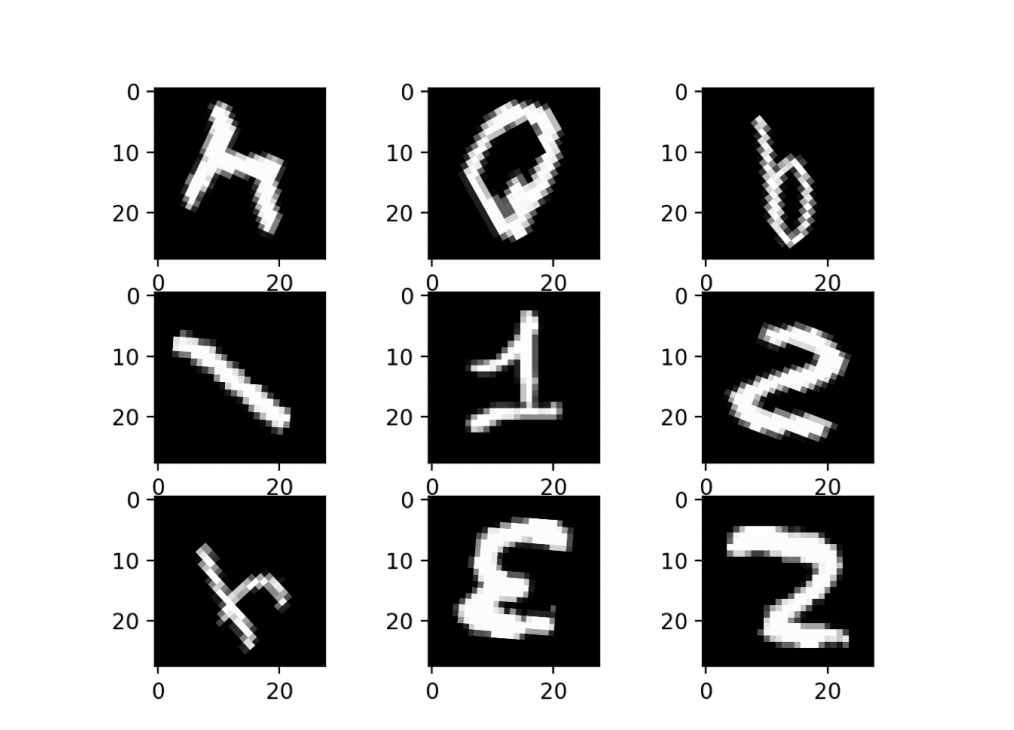


Figure A.17: Random Rotations.

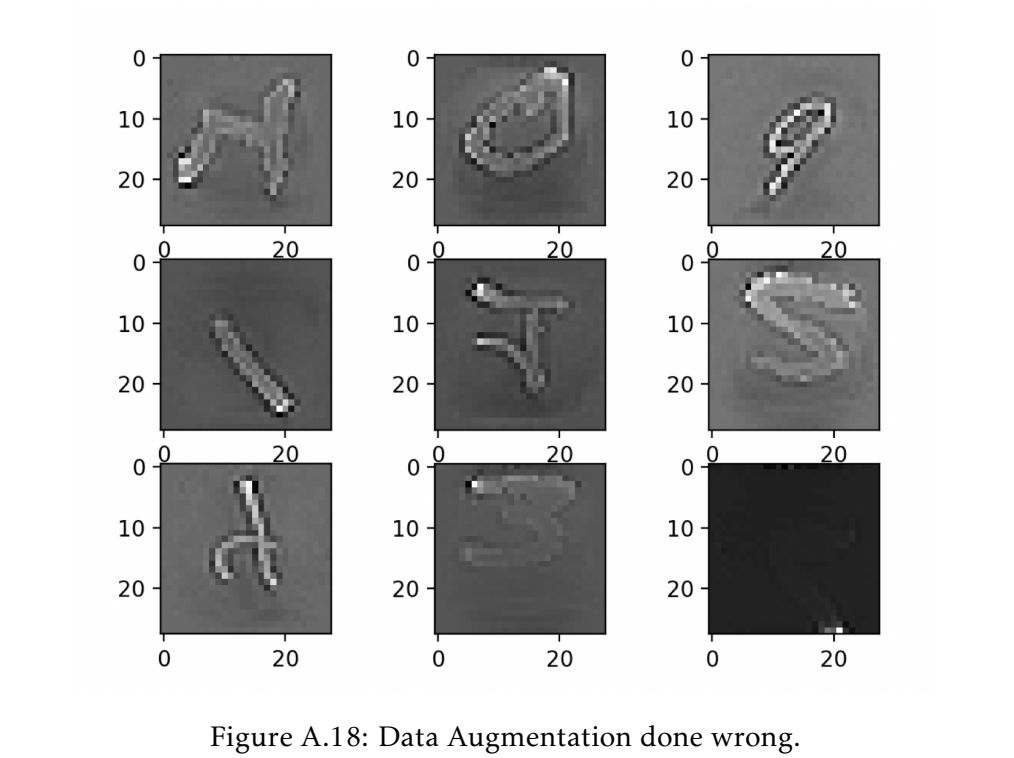


Figure A.18: Data Augmentation done wrong.

techniques. Filling mode can have different flavors points outside the boundaries of the input are filled according to the given mode:

- Constant: The outside is filled with a predefined value.

$$\left[kkkk|abcd|kkkk\right](p_v al = k) \tag{A.14}$$

- Nearest: The outside is filled with the nearest value of the last pixel.

$$\left[aaaa|abcd|dddd\right] \tag{A.15}$$

- Reflect: The outside is filled with a reflection of the values, sometimes called mirror filling.

$$\left[dcba|abcd|dcba\right] \tag{A.16}$$

- Wrap: The outside is filled with the opposite values, like the image was a cylinder and the content is wraping around.

$$\left[abcd|abcd|abcd\right] \tag{A.17}$$

Image data is unique in the way that is possible to review the data, create transformed copies and quickly get an idea of how the dataset may be perceive it by the working model. Training DNNs comes with experience, and the quality of the results are interlinked with the tweaks done to the data. For that reason, in conclusion of this section, it's summarized some tips for getting the most from image data preparation and augmentation for DL.

- Review the dataset and do some work with it before starting to train models. In most cases, only a few images actually benefit the training process of your model when augmented, such as the need to handle different shifts, rotations or flips of objects in the scene.

- Inspect augmentations. It is one thing to intellectually know what image transforms to use, but in practical cases, it is very different to look at examples results. Reviewing images both with individual augmentations as well as the full set of augmentations planned may unveil ways to simplify or further enhance your model training process.

- Lastly, it's important to evaluate a suite of transforms. Trying more than one image data preparation and augmentation scheme. Often it's possible that the results of a data preparation scheme are different that what was initially envisioned and the data augmentations are not beneficial.

Appendix B

Dynamic Collision Avoidance Training Results

This appendix contains the results of training the 3-depth LSTM model varying the units per layer. The inputs are feature vectors from the ColANet dataset that were produced using a untrained MobileNetV2 with ImageNet weights. In order to fully study the influence of different units on each layer of the RNN, different numbers of neuron units on each layer was tested with values ranging from $2^0(1)$ to $2^6(64)$, resulting in $7^4(2401)$ combinations of networks. The same number of sequences with collisions and no collisions was pre-processed and 5% of the videos were placed on the validation test, which means ≈ 6000 train feature sequences and ≈ 300 validation feature sequences (the value vary because each video have a different number of collision frames).

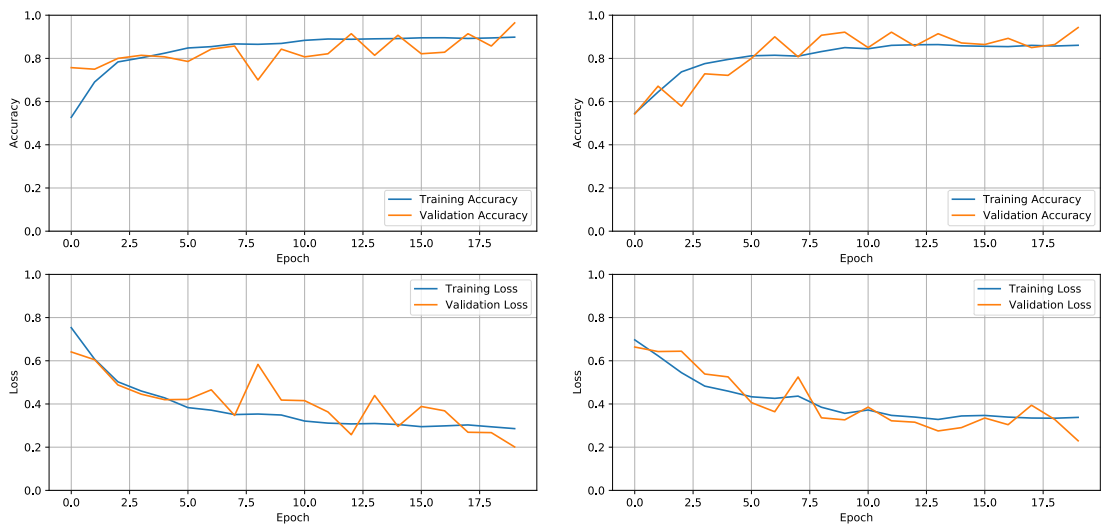
The networks were trained on a Nvidia GeForce RTX 2070 with CUDA libraries. They trained for 20 epoch with a batch size of 32 having an average train time per model $\mu = 130,01$ s and a standard deviation $\sigma = 1,2$ s. This gives us a total of approximately 87 h 45 m to train the desired models variants. A Adam optimizer with learning rate of 1×10^{-3} and a decay rate of 1×10^{-6} were considered. Furthermore a dropout of 40% and batch normalization was added to the outputs of LSTMs layers.

The most important metric is the validation accuracy, because it represents the accuracy of the network on unknown data. Nevertheless, looking at the best scoring networks on the validation accuracy metric can be misleading by multiple reasons. A network can enter a local minima point where the value of validation accuracy is higher the training accuracy (sometimes considered under fitting). Also, all the top best scoring had a slight increase of accuracy on the last epoch, which boost the results. For this reason, it is illustrated the average of the best 5, 10, 20, 50, and 100 networks on table B.1. The loss is also an important metric, which should float inversely proportional with accuracy.

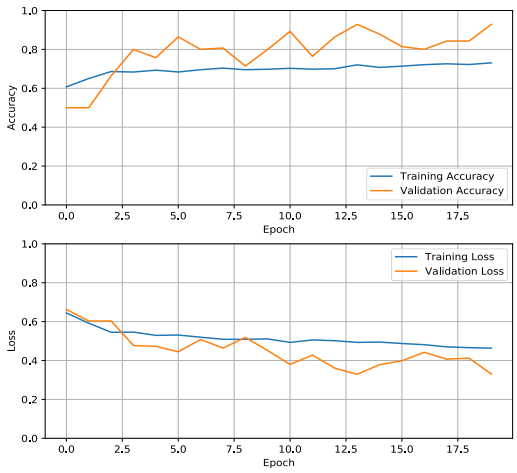
Table B.1: Average of the best scoring models regarding validation accuracy on training and validation results for accuracy and loss.

Averages	Training accuracy	Validation accuracy	Training loss	Validation loss
Top 1	89.83%	94.43%	0.286	0.201
Top 5	79.81%	93.12%	0.398	0.283
Top 10	82.60%	92.29%	0.364	0.287
Top 20	85.14%	90.99%	0.320	0.289
Top 50	88.07%	88.41%	0.269	0.370
Top 100	88.14%	86.15%	0.266	0.446

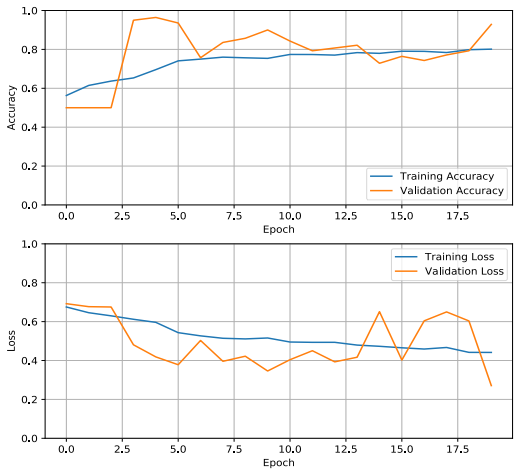
The figures on B.1 illustrate the 8 best scoring networks ordered by validation accuracy. The dataset is small, with high variance in the data, which make the accuracy results on the validation set to oscillate during training. On the best model training scenarios, both the training and validation accuracy grow organically with a small increase on the last epoch. Good examples of this types of training flows are the graphs of the models Top7 LSTM1=4 LSTM2=2 LSTM3=32 Dense=4 and Top8 LSTM1=4 LSTM2=4 LSTM3=8 Dense=8 .



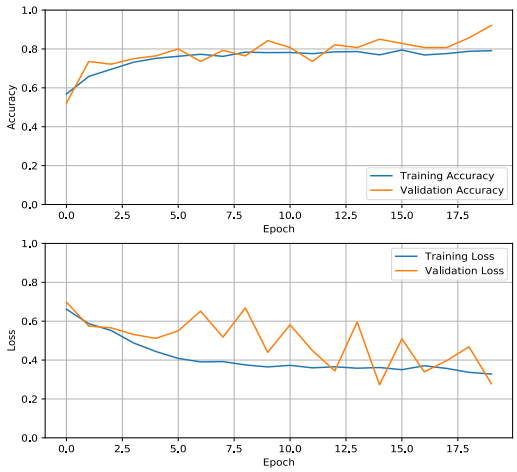
a. Top1 - LSTM1=2 LSTM2=8 LSTM3=32 Dense=4 b. Top2 LSTM1=2 LSTM2=8 LSTM3=32 Dense=4



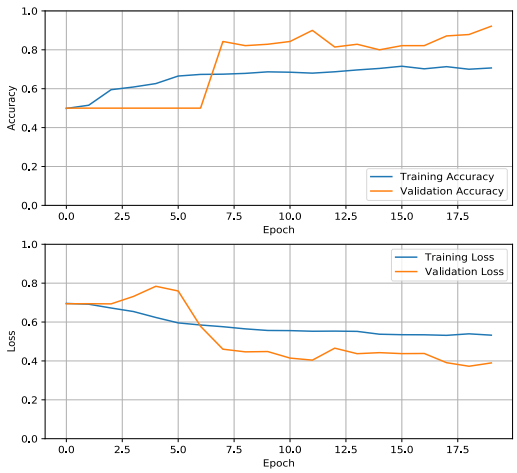
c. Top3 LSTM1=1 LSTM2=8 LSTM3=1 Dense=2



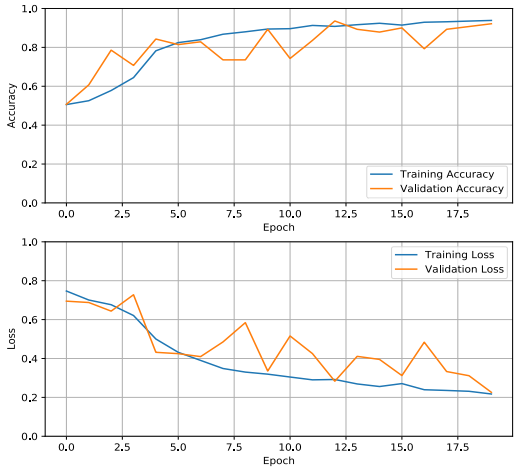
d. Top4 LSTM1=1 LSTM2=1 LSTM3=2 Dense=1



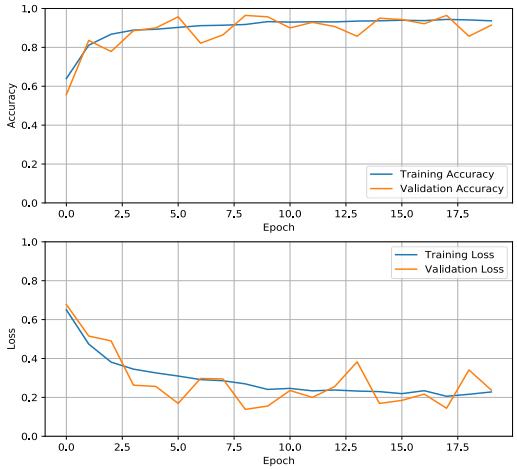
e. Top5 LSTM1=8 LSTM2=64 LSTM3=1 Dense=32



f. Top6 LSTM1=4 LSTM2=8 LSTM3=1 Dense=1



g. Top7 LSTM1=4 LSTM2=2 LSTM3=32 Dense=4



h. Top8 LSTM1=4 LSTM2=4 LSTM3=8 Dense=8

Figure B.1: Top 8 of the best scoring models regarding validation accuracy.

For each model, is presented the training results evolution in terms of accuracy and loss.

In pursuance of a better understanding of amount of units per layer, it was calculated the average of results when a given layer L has a given number of units U. In case of LSTM layers, the units are LSTM neurons and on the dense layer the units are a matrix vector multiplication. The results of this are presented in tables B.2, B.3 B.4 and B.5. On each layer, it was highlighted from green to red, the best results to worse. Keep in mind that on the loss tables a lower value represents a better result, and for this reason the lower values appear in green.

There are multiple conclusions that can be taken from these tables. First it's clear that with a low number of units the model as difficulties learning, and tends to present bad results during training, which are then passed to the validation tests. On the other hand, when the model has a higher amount of units per layer, the training results are better, but tend to overfit, which leads to bad validation results and be seen by the gap between the training results and validation results. Given this, the best models seen to be a combination of 4 to 16 in the first two LSTM layers and from 1 to 8 on the last LSTM layer. The number of neurons in the dense layer seen indifferent, but the models that achieved better results were the models with 8 or 16 neurons on the dense layer.

Table B.2: Training accuracy mean with variation of the number of units per layer.

Layers	**Units**						
	64	**32**	**16**	**8**	**4**	**2**	**1**
LSTM 1	89.23%	88.99%	90.03%	90.02%	88.88%	85.82%	74.46%
LSTM 2	88.30%	87.55%	86.60%	87.68%	86.10%	85.75%	85.45%
LSTM 3	90.51%	89.70%	89.93%	90.47%	87.71%	83.28%	75.89%
Dense	90.71%	89.48%	90.12%	88.73%	87.39%	82.14%	78.89%

Table B.3: Validation accuracy mean with variation of the number of units per layer.

Layers	**Units**						
	64	**32**	**16**	**8**	**4**	**2**	**1**
LSTM 1	60.90%	65.91%	69.26%	70.91%	70.87%	68.02%	61.98%
LSTM 2	68.08%	66.65%	67.90%	67.29%	66.57%	66.23%	65.14%
LSTM 3	65.76%	65.62%	65.68%	67.79%	66.84%	67.31%	68.84%
Dense	66.41%	66.95%	67.29%	67.27%	66.20%	66.73%	67.01%

On figure B.2 it's presented the accuracy results of all models. On the abscissa is the training accuracy and on the ordinate the validation. It's possible to visualize that a few models were stuck on a local minima and couldn't reach a value higher then 50% on validation. Furthermore, many models achieved near perfect accuracy during training, but a lower value on validation, which hint us that the dataset as few data, and the model

Table B.4: Training loss mean with variation of the number of units per layer.

Layers	Units						
	64	32	16	8	4	2	1
LSTM 1	0.234	0.237	0.220	0.223	0.242	0.282	0.412
LSTM 2	0.241	0.251	0.263	0.251	0.273	0.280	0.289
LSTM 3	0.201	0.211	0.212	0.217	0.260	0.326	0.420
Dense	0.189	0.206	0.201	0.226	0.256	0.336	0.433

Table B.5: Validation loss mean with variation of the number of units per layer.

Layers	Units						
	64	32	16	8	4	2	1
LSTM 1	1.233	1.101	1.020	0.928	0.898	0.929	0.876
LSTM 2	1.015	1.047	0.976	0.983	0.989	0.997	0.980
LSTM 3	1.158	1.118	1.119	1.082	1.017	0.836	0.660
Dense	1.102	1.059	1.103	1.052	1.066	0.876	0.730

is overfitting. Nevertheless, many models achieve a performance higher than 85%, which proves that this is a valid solution to build models for the collision avoidance problem.

Figure B.2: Training and Validation accuracy results mapped into a xy plane.