ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Scuola di Ingegneria e Architettura di Cesena
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

A COMPARISON AMONG DEEP LEARNING TECHNIQUES IN AN AUTONOMOUS DRIVING CONTEXT

Elaborata nel corso di: Sistemi Intelligenti Robotici

Tesi di Laurea di:
LUCA SANTONASTASI

Relatore:
Prof. ANDREA ROLI

ANNO ACCADEMICO 2016-2017
SESSIONE II

KEYWORDS

Autonomous Driving

Deep Learning

Reinforcement Learning

Whenever I hear people saying AI is going to hurt people in the future I think, yeah, technology can generally always be used for good and bad and you need to be careful about how you build it. If you're arguing against AI then you're arguing against safer cars that aren't going to have accidents, and you're arguing against being able to better diagnose people when they're sick.

[*M. Zuckerberg*]

Introduction

The recent advances in computational power of processors have opened up new opportunities for computational intelligence methods. The potential of algorithms known from decades but still not implemented has been unleashed owing to these recent computational improvements.

These advancements in computational power favoured the expansion of artificial intelligence to domains that were confined to fiction movies in the past.

One of these wonders that has always fed the imagination and dreams of many people is the autonomous car. In the 80's TV series like *Knight Rider* have predicted the possibilities in the near future of such technologies. Nowadays, technology newspapers and blogs are filled by article about how it works but they never go in much detail. The main reason is that car manufactures have always avoided to disclose their secrets. Nevertheless, research papers such as [1] or [2] have paved the way for disseminating these technoglogies, for example by showing successful applications or artificial neural networks in tasks such as traffic sign recognition or lane keeping driver assistant. In particular, the work in [2] exploits deep learning methods, such as Convolutional Neural Networks in order to teach a car to steer and drive using supervised learning with a wide dataset. Deep Learning can be summed up as a sub field of Machine Learning studying statical models called deep neural networks. The latter are able to learn complex and hierarchical representations from raw data, unlike hand crafted models which are made of an essential features engineering step. Especially during the last five years, Deep Learning has made a tremendous impact in computer vision reaching previously unattainable performance on many tasks such as image classification, objects detection, object localization, object tracking, pose estimation, image segmentation or image captioning.

Among the various paradigms in machine learning, a prominent one is *reinforcement learning*, which has recently attracted the attention of scholars and developers and has proven to lead to outstanding results.

Reinforcement learning is a known field in cognitive science that focuses on the study of thinking processes. The main idea is that learning happens with the help of a feedback coming from the outer world as a response to an action. In psychology, the idea of trial-and-error has been expressed by Edward Thorndike in 1911 [3]. So, this very idea existed for a long time until it could become the basis of a machine learning paradigm. In fact, the idea of applying these concepts to computers is not new, as it was already proposed when the notion of computational machine was proposed by Turing [3]. In its early stages, it was first combined with supervised learning, and took its own way in 70s, when the method was formalized and improved.

Lately, DeepMind (former Google Brain Team) made important contributions in artificial intelligence by using reinforcement learning algorithms together with deep learning techniques in neural networks. They have published some state-of-the-art papers that became milestone for the current technological advancements. These include the papers *Playing Atari with Deep Reinforcement Learning* [4] - 2013, where deep Q-learning is used, and the biggest most recent breakthrough, *Mastering the Game of Go with Deep Neural Networks and Tree Search* [5] - 2016, where reinforcement learning Monte Carlo tree search is used with Q-learning. Also, in June 2016, they published a paper *Asynchronous Methods for Deep Reinforcement Learning* [6] that provides a clear comparison of the different deep reinforcement learning algorithms for the asynchronous training and introduces an algorithm - *Asynchronous Advantage Actor-Critic* that surpasses the performances of deep learning approach used for the Atari 2600 games in the paper [4].

In this abundant flow of discoveries in artificial intelligence and deep reinforcement learning, the reuse of new techniques and their further exploration needs to be investigated and studied in more detail. The goal of this work is to dive in this world and show in a specific use case which one behave the best. The specific use case consists of a company which need to start from scratch and want to develop its own ADAS (Advanced Driver Assistance Systems) with no dependence from any of the vendors which provide camera or sensors.

The main objective of this work is to study and analyse the most relevant techniques in deep learning and compare them in the case of autonomous driving. The context of this work is an industrial environment in which a company wants to develop its own Advanced Driver Assistance System with total independence form vendors providing cameras and sensors.

Chapter 1 provides the theoretical background on autonomous driving, machine learning and computer vision.

Chapter 2 illustrates methods, algorithms and neural networks architectures that are best suited for autonomous driving.

Chapter 3 describes all the choices that have been made behind the development of the algorithms. Comparison are shown among programming languages, GPU and CPU, Simulators and Deep Learning Frameworks

All the algorithms and techniques developed in this thesis are described in detail in Chapter 4, while in Chapter 5 experimental results are shown.

Finally, the last chapter summarizes the work done and provides an outlook to future developments.

Contents

List of Figures

Chapter 1

Theoretical Background

In this chapter, an overview of the main concepts at the background of this work is provided and the related technologies that will be dealt with in the following chapters are described. In particular, the work is focused on:

1. **Self Driving Car** (SDC): it is an Unmanned Ground Vehicle capable of sensing its environment and navigating without human input.

2. **Machine Learning** (ML): Machine Learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that machines should be able to learn and adapt through experience [8]. In those section I will also describe *Deep Learning* (DL) and I will introduce *Artificial Neural Networks* (ANNs)

3. **Convolutional Neural Networks** (CNNs): it is a class of deep, feed-forward ANNs that has successfully been applied to analysing visual imagery. Although they belong to *Computer Vision* field, I will treat it on a separate section for the relevance they will have in this work.

4. **Recurrent Neural Networks** (RNNs): class of ANNs where connections between units form a directed cycle. This property allows them to exhibit dynamic temporal behaviour.

5. **Reinforcement Learning** (RL): area of ML inspired by behaviourist psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.

6. **Computer Vision** (CV): discipline that deals with how computers can be made for gaining high-level understanding from digital images or videos.

In the following sections of this chapter, the above mentioned subjects will be described with the aim of providing a background to the modelling, designing and development a Self-Driving Car.

1.1 Self-Driving Car

A *Self-Driving Car* (SDC) or *autonomous car* is an Unmanned Ground Vehicle that is a vehicle capable of sensing its environment and navigating without human input [9].

Autonomous cars use many sensors which are able to perceive the surrounding environment. Sensors used could be radars, laser lights, GPS sensors or vision sensors. Advanced control systems interpret sensory information to plan appropriate navigation paths, as well as to recognise obstacles and relevant traffic signals.[10, 11] Autonomous cars must have control systems that are capable of analysing sensory data to distinguish between different cars on the road.

Many such systems are evolving, but by now no cars permitted on public roads were fully autonomous. They all require a human at the wheel who must be ready to take control at any time.

1.1.1 History and the State of the Art

The first examples of Self-Driving Car has to be found in 1920s were *Houdina Radio Control* demonstrated the radio-controlled *American Wonder* on New York City streets. In 1957, a full size system was successfully

Figure 1.1: *1960 British self-driving car that interacted with magnetic cables that were embedded in the road. It went through a test track at 80 miles per hour (130 km/h) without deviation of speed or direction in any weather conditions, and in a far more effective way than by human control [12, 13, 14]. Image taken from [13]*

demonstrated by RCA Labs and the State of Nebraska on a 400-foot strip of public highway at the intersection of U.S. Route 77 and Nebraska Highway 2, then just outside Lincoln, Nebraska. A series of experimental detector circuits buried in the pavement were a series of lights along the edge of the road. The detector circuits were able to send impulses to guide the car and determine the presence and velocity of any metallic vehicle on its surface[18, 19]. During the years, many examples of autonomous cars were revealed by research laboratories, but more relevant project has been created in the 1990s[12, 13, 14, 20, 21, 22].

In 1995, Carnegie Mellon University's Navlab project completed a 3,100 miles (5,000 km) cross-country journey, of which 98.2% was autonomously controlled, dubbed *No Hands Across America.* This car, however, was semi-autonomous by nature: it used neural networks to control the steering wheel, but throttle and brakes were human-controlled, chiefly for safety reasons.[21, 22]

In 1996, Alberto Broggi of the University of Parma launched the ARGO

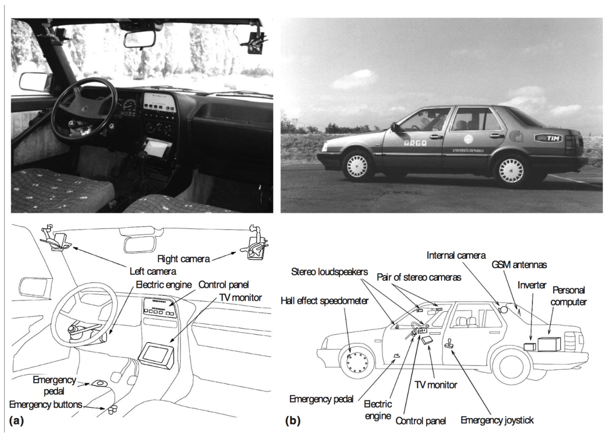


FIGURE 1.2: *ARGO's equipment from an external(a) and internal perspective(b). Image taken from [15]*

Project, which worked on enabling a modified Lancia Thema (see Figure 1.2) to follow the normal (painted) lane marks in an unmodified highway[23, 15].The culmination of the project was a journey of 1,200 miles (1,900 km) over six days on the motorways of northern Italy dubbed *Mille Miglia in Automatico* ("One thousand automatic miles"), with an average speed of 56 miles per hour (90 km/h)[24]. The car operated in fully automatic mode for 94% of its journey, with the longest automatic stretch being 34 miles (55 km). The vehicle had only two black-and-white low-cost video cameras on board and used stereoscopic vision algorithms to understand its environment.

In 2004, DARPA (the Defense Advanced Research Projects Agency) launched the first Grand Challenge event and offered a $1 million prize to any team of engineers which could create an autonomous car able to finish a 150-mile course in the Mojave Desert. No team was successful in completing the course[25]. In October 2005, the second DARPA Grand Challenge was again held in a desert environment. GPS points were placed and obstacle types were located in advance [26]. In this challenge, five vehicles completed the course. In November 2007, DARPA again sponsored Grand Challenge III, but this time the Challenge was held in an urban environment. In this race, a 2007 Chevy Tahoe autonomous car from Carnegie Mellon University earned

Figure 1.3: *Google Driverless Car. Image taken from Grendelkhan,2016*

the 1st place. Prize competitions as DARPA Grand Challenges gave students and researchers an opportunity to research a project on autonomous cars to reduce the burden of transportation problems such as traffic congestion and traffic accidents that increasingly exist on many urban residents[26].

Many major automotive manufacturers, including General Motors, Ford, Mercedes Benz, Volkswagen, Audi, Nissan, Toyota, BMW, and Volvo, are testing driverless car systems as of 2013. BMW has been testing driverless systems since around 2005,[27, 28] while in 2010, Audi sent a driverless Audi TTS to the top of Pike's Peak at close to race speeds.[29] In 2011, GM created the EN-V (short for Electric Networked Vehicle), an autonomous electric urban vehicle[30].

In 2010, Italy's VisLab from the University of Parma, led by Professor Alberto Broggi, ran the VisLab Intercontinental Autonomous Challenge (VIAC), a 9,900-mile (15,900 km) test run which marked the first intercontinental land journey completed by autonomous vehicles. Four electric vans made a 100-day journey, leaving Parma on 20 July 2010, and arriving at the Shanghai Expo in China on 28 October. Although the vans were driverless and mapless, they did carry researchers as passengers in case of emergencies. The experimenters did have to intervene a few times - when the vehicles got caught in a Moscow traffic jam and to handle toll booths[17].

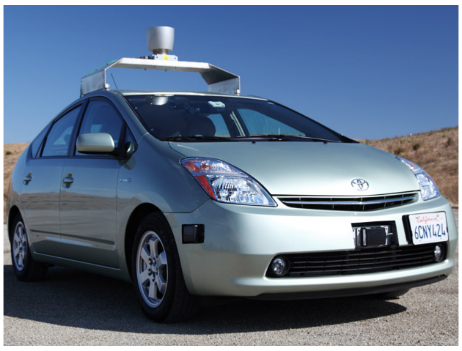


FIGURE 1.4: *In Las Vegas, Chris Urmson, now head of Google's self-driving car project, tested a 2008 Toyota autonomously driven and was in the driver seat in case anything went wrong. Google had previously mapped the area and selected a route for the test, which the DMV agreed to. Image taken from [16]*

On May 1, 2012, a 22 km (14 mi) driving test was administered to a Google self-driving car by Nevada motor vehicle examiners in a test route in the city of Las Vegas, Nevada. The autonomous car passed the test, but was not tested at roundabouts, no-signal rail road crossings, or school zones.[16]

In 2013, on July 12, VisLab conducted another pioneering test of autonomous vehicles, during which a robotic vehicle drove in down town Parma with no human control, successfully navigating roundabouts, traffic lights, pedestrian crossings and other common hazards[31].

Although as of 2013, fully autonomous vehicles are not yet available to the public, many contemporary car models have features offering limited autonomous functionality. These include adaptive cruise control, a system that monitors distances to adjacent vehicles in the same lane, adjusting the speed with the flow of traffic; lane assist, which monitors the vehicle's position in the lane, and either warns the driver when the vehicle is leaving its lane, or, less commonly, takes corrective actions; and parking assist, which assists the driver in the task of parallel parking[32].

In October 2014, Tesla Motors announced its first version of *AutoPilot*. Model S cars equipped with this system are capable of lane control with

FIGURE 1.5: *Driverless electric vans complete 8,000 mile journey from Italy to China. Image taken from [17]*

autonomous steering, braking and speed limit adjustment based on signals image recognition. The system also provide autonomous parking and is able to receive software updates to improve skills over time[33]As of March 2015, Tesla has been testing the autopilot system on the highway between San Francisco and Seattle with a driver but letting the car to drive the car almost unassisted.[34]

Tesla Model S Autopilot system is suitable only on limited-access highways not for urban driving. Among other limitations, Autopilot can not detect pedestrians or cyclists[35]. In March 2015 Tesla Motors announced that it will introduce its Autopilot technology by mid 2015 through a software update for the cars equipped with the systems that allow autonomous driving[34]. Some industry experts have raised questions about the legal status of autonomous driving in the U.S. and whether Model S owner would violate current state regulations when using the autopilot function. The few states that have passed laws allowing autonomous cars on the road limit their use for testing purposes, not the use by the general public. Also, there are questions about the liability for autonomous cars in case there is a mistake[34].

In February 2015 Volvo Cars announced its plans to lease 100 XC90 SUVs fitted with Drive Me Level 3 automation technology to residents of Gothenburg in 2017[36, 37]. The Drive Me XC90s will be equipped with

NVIDIA's Drive PX 2 supercomputer, and will be driven autonomously in certain weather conditions and on one road that loops around the city[38].

In July 2015, Google announced that the test vehicles in its driverless car project had been involved in 14 minor accidents since the project's inception in 2009. Chris Urmson, the project leader, said that all of the accidents were caused by humans driving other cars, and that 11 of the mishaps were rear-end collisions. Over the six years of the project's existence the test vehicles had logged nearly 2 million miles on the road[39, 25].

The first known fatal accident involving a vehicle being driven by itself took place in Williston, Florida on 7 May 2016 while a Tesla Model S electric car was engaged in Autopilot mode. The driver was killed in a crash with a large 18-wheel tractor-trailer. On 28 June 2016 the National Highway Traffic Safety Administration (NHTSA) opened a formal investigation into the accident working with the Florida Highway Patrol. According to the NHTSA, preliminary reports indicate the crash occurred when the tractor-trailer made a left turn in front of the Tesla at an intersection on a non-controlled access highway, and the car failed to apply the brakes. The car continued to travel after passing under the truck's trailer.[40, 41] The NHTSA's preliminary evaluation was opened to examine the design and performance of any automated driving systems in use at the time of the crash, which involves a population of an estimated 25,000 Model S cars[42]. In August 2016 Singapore launched the first self-driving taxi service, provided by nuTonomy[43].

1.1.2 Autonomous instead of automated

Autonomy generally means freedom from external control. Usually, when an agent, or a vehicle or a robot is said to be autonomous, it is meant to be with a certain degree of autonomy because an agent could have dependence from environment or other agents in many different ways. It is not an all-or-nothing issue, but a matter of degree [45]. Because of this reason, autonomous driving has been classified using a system based on six different levels (ranging from

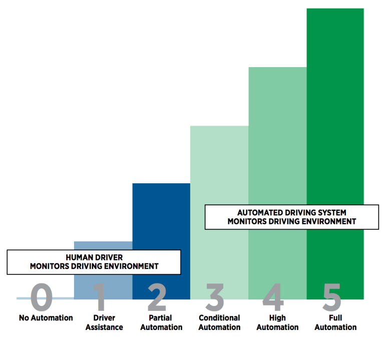


FIGURE 1.6: *SAE Classification. Image taken from [44]*

fully manual to fully automated systems) that was published in 2014 by SAE International, an automotive standardization body, as J3016, Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems.[44] This classification system is based on the amount of driver intervention and attentiveness required, rather than the vehicle capabilities, although these are very loosely related.

These levels are very important because they help to identify the real nature of the vehicle. Actually, there is no commercial Level 4/ Level 5 vehicle on the road, but only some level 3. For example, in 2017 Audi stated that its latest A8 would be autonomous using its *Audi AI*. Billed by Audi as the *traffic jam pilot* it takes charge of driving in slow-moving traffic at up to 60 km/h on motorways where a physical barrier separates the two carriageways. The driver would not have to do safety checks such as frequently gripping the steering wheel. The Audi A8 was claimed to be the first production car to reach level 3 autonomous driving and Audi would be the first manufacturer to use laser scanners in addition to cameras and ultrasonic sensors for their system.[46]

1.1.3 Advantages

Autonomous driving has not a long history since the first real approaches but it may introduce several advantages once deployed for commercial purpose.

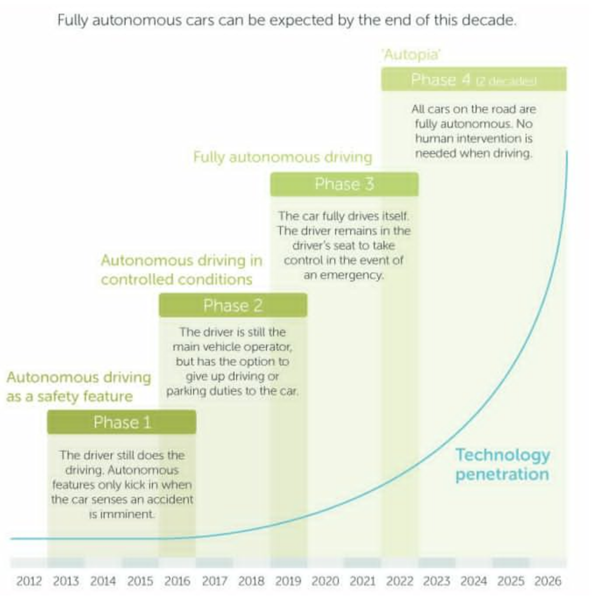


FIGURE 1.7: *Prediction that show the steps that it will be taken to have all cars fully autonomous . Image taken from [47]*

- *Safety*: autonomous driving should reduce to a minimum the risk of an accidental error caused by human distraction or aggressive driving. For human error it is meant rubbernecking, delayed reaction time, tailgating, and other forms of distracted or aggressive driving [47, 48, 49].

- *Welfare*: Autonomous cars could relieve travellers from driving and navigation chores, thereby replacing behind-the-wheel commuting hours with more time for leisure or work;[47, 50] and also would lift constraints on occupant ability to drive, distracted and writing SMS while driving, intoxicated, prone to seizures, or otherwise impaired.[51, 25]. For the young, the elderly, people with disabilities, and low-income citizens, autonomous cars could provide enhanced mobility.[52, 53, 54]

- *Traffic*: Advantages that comes from autonomous driving cars could also include higher speed limits;[55] smoother rides;[56], increased roadway capacity and minimized traffic congestion, due to decreased need for safety gaps and higher speeds[57]. Currently, maximum controlled-access highway throughput or capacity according to the U.S. Highway Capacity Manual is about 2,200 passenger vehicles per hour per lane, with about 5% of the available road space is taken up by cars. It has been estimated that autonomous cars could increase capacity by 273% (8,200 cars per hour per lane)[58].

- *Costs*: Safer driving could reduce the costs of vehicle insurance[59, 58]. Reduced traffic congestion and the improvements in traffic flow due to widespread use of autonomous cars will also translate into better fuel efficiency. [53, 60]

1.1.4 Obstacles

In spite of the benefits related to increased vehicle automation, there are many foreseeable challenge which persist. For example, there are disputes concerning liability, the time needed to turn the existing stock of vehicles from non-autonomous to autonomous,[61] resistance by individuals to forfeit control of their cars,[62] customer concern about the safety of driverless cars,[63] and the implementation of legal framework and establishment of government regulations for self-driving cars.[64]. Other obstacles could be missing driver experience in potentially dangerous situations[65], ethical problems in situations where an autonomous car's software is forced during an unavoidable crash to choose between multiple harmful courses of action[66, 67, 68], and possibly insufficient Adaptation to Gestures and non-verbal cues by police and pedestrians[69].

Possible technological obstacles for autonomous cars are:

- Software reliability.[70]
- A car's computer could potentially be compromised, as could a communication system between cars[71, 72, 73, 74, 75].
- Susceptibility of the car's sensing and navigation systems to different

types of weather or deliberate interference, including jamming and spoofing[69].

- Avoidance of large animals requires recognition and tracking, and Volvo found that software suited to caribou, deer, and elk was ineffective with kangaroos[76].
- Autonomous cars may require very high-quality specialised maps[77] to operate properly. Where these maps may be out of date, they would need to be able to fall back to reasonable behaviours[69, 78].
- Field programmability for the systems will require careful evaluation of product development and the component supply chain.[75]
- Current road infrastructure may need changes for autonomous cars to function optimally [79]
- Cost (purchase, maintenance, repair and insurance) of autonomous vehicle as well as total cost of infrastructure spending to enable autonomous vehicles and the cost sharing model.
- A direct impact of widespread adoption of autonomous vehicles is the loss of driving-related jobs in the road transport industry[59, 80]. There could be job losses in public transit services and crash repair shops. The car insurance industry might suffer as the technology makes certain aspects of these occupations obsolete[53].
- Research shows that drivers in autonomous cars react later when they have to intervene in a critical situation, compared to if they were driving manually[81].

1.2 Machine Learning

There are problems that could be considered very difficult to be formulated and it happens that is extremely difficult to write programs that can solve some type of problems satisfactorily. Sometimes the solution found may be hard to be understood and so they lack of generality. Such algorithm could not be so robust to work when noisy data are used and so they may introduce maintenance problems. A viable solution to this kind of problems is *machine learning.* Generally speaking, a machine learning problem consists of lots of data to be considered in order to obtain an efficient and robust behaviour by the program. Data are fundamental in such kind of programs because they influence learning. Lots of data and correct data are essential part of machine learning algorithm. In fact, such algorithm takes these examples and produces a program that does the job. Usually there may be some task that will require to evaluate data which it has never been considered. A machine learning algorithm is said to be robust if it correctly evaluates also these data. Moreover, machine learning algorithm could provide mechanisms that make the program change on-the-job. This feature when related to human is called *learning by experience.* An important definition was given by Mitchell who said that *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E*[82]. Such definition is relevant because gives a general overview of what is the intrinsic relation between experience and learning. Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning *signal* or *feedback* available to a learning system[83]. These are:

- *supervised learning*: paradigm that uses sets of data already collected in order to provide input-output relationships. Those data provides feedback to the learning process and, as long as it computes, it has to derive a model also for unknown relationship. Usually algorithms that follow such paradigm are able to solve classification problem (a.k.a. pattern recognition) and regression problem.

- *unsupervised learning*: paradigm that has input data which do not have target outputs given. It has to learn how data are organised, discover patterns or build a representation of them. Usually algorithms that follow such paradigm are able to solve clustering type problems.

- *reinforcement learning*: paradigm which addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goal. Usually, input data are generated by the agent's interaction with the environment. A critic provides a reward or a penalty to indicate the desirability of the resulting behaviour. The evaluation from the critic represents the feedback to be used in the learning process. The task of the agent is to learn from this indirect, delayed reward, to choose sequences of actions that produce the greatest cumulative reward.

Machine learning grew out of the quest for AI. Already in the early days of AI as an academic discipline, some researchers were interested in having machines that learn from data. They attempted to approach the problem with various symbolic methods, as well as what were then termed "neural networks"; these were mostly perceptrons and other models that were later found to be reinventions of the generalized linear models of statistics[84].

1.2.1 Deep Learning

During 1980s, machine learning algorithm like Support vector machines and other, much simpler methods such as linear classifiers gradually overtook neural networks in machine learning popularity. Earlier challenges in training deep neural networks were successfully addressed with methods such as unsupervised pre-training, while available computing power increased through the use of GPUs and distributed computing. Neural networks were deployed on a large scale, particularly in image and visual recognition problems. This became known as *deep learning*, although deep learning is not strictly synonymous with deep neural networks. Deep learning is a class of machine learning algorithms that:

- "use a cascade of many layers of non-linear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. The algorithms may be supervised or unsupervised and applications include pattern analysis (unsupervised) and classification (supervised)"(quoted from [85]).

- "are based on the (unsupervised) learning of multiple levels of features or representations of the data. Higher level features are derived from lower level features to form a hierarchical representation.
- "are part of the broader machine learning field of learning representations of data"(quoted from [85]).
- "learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.
- "use some form of gradient descent for training via back-propagation" (quoted from [85])

"These definitions have in common multiple layers of non-linear processing units and the supervised or unsupervised learning of feature representations in each layer, with the layers forming a hierarchy from low-level to high-level features"(quoted from [85]). The composition of a layer of non-linear processing units used in a deep learning algorithm depends on the problem to be solved. Layers that have been used in deep learning include hidden layers of an artificial neural network and sets of complicated propositional formulas[86]. They may also include latent variables organized layer-wise in deep generative models such as the nodes in Deep Belief Networks and Deep Boltzmann Machines[87].

The assumption underlying distributed representations is that observed data are generated by the interactions of layered factors. Varying numbers of layers and layer sizes can provide different amounts of abstraction[88]. Deep learning exploits this idea of hierarchical explanatory factors where higher level concepts are learned from the lower level ones [89, 90]. Deep learning helps to disentangle these abstractions and pick out which features are useful for improving performance[88]. For supervised learning tasks, deep learning methods obviate feature engineering, by translating the data into compact intermediate representations similar to principal components, and derive layered structures that remove redundancy in representation.[91, 85] Deep learning algorithms can also be applied to unsupervised learning tasks. This is an important benefit because unlabelled data are more abundant than labelled data. Examples of deep structures that can be trained in an unsupervised manner are neural history compressors[91, 92] and deep belief networks[88, 87].

1.2.2 Artificial Neural Networks

ANNs are computing systems inspired by the biological neural networks that constitute brains. Their properties and functionality have been successfully replicated in ANNs. An ANN is a layered structure of neurons which has minimum one input layer and one output layer. Each layer has a number of units (neurons) that are connected to the units in another layer. The connections are assigned weights that are used for the unit's *activation*. In order to perform activation, a weighted sum of its inputs is computed at each unit and then passed to an activation function to produce output. There are different activation functions, though, and the choice depends on the type of problem.

Depending on the structure of the network, there are different types of ANNs. For example, a simple case of *feed-forward* ANN without any loops, with an input layer of 4 units, 2 hidden layers, and an output layer with 2 units is presented in the Figure 1.8:

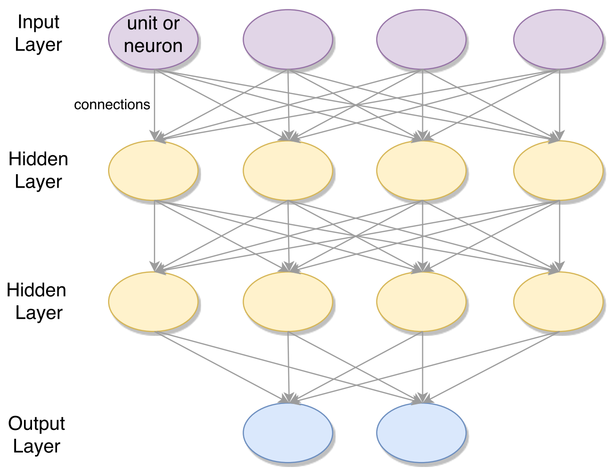


FIGURE 1.8: *Feed-forward ANN*

Some ANNs can be proclaimed as *deep*. This qualifier describes the networks that have many hidden layers, precisely more than two. More hidden layers would compute more abstract representations of the input data, which means richer features. Deep ANNs are harder to train, but they are more powerful and are especially used in modern artificial intelligence applications.

Usually, ANNs learn by an SGD method. More precisely, learning implies the definition of an objective function that describes the performance of the network, and, which is either minimized or maximized. An objective function can be the loss of the network over a set of training examples. In RL, an ANN can use TD errors in computing the loss function and learning the value function, or maximize the reward, or use a policy gradient algorithm [3]. No matter the case, the partial derivatives are required to determine the influence of a weight change on the network's performance, and they can be obtained with the help of the gradient.

In order to find the gradient, an ANN can use the back-propagation algorithm. In the forward pass, the network's units would compute the outputs, whereas in the backward pass - the partial derivatives with respect to each weight. However, the back-propagation algorithm isn't that efficient for the deep ANNs, because of the overfitting problem.

Deep Neural Networks

A special structure of the network's architecture, like that of the deep *convolutional networks* (CNN) would make it possible to use the back-propagation algorithm in deep ANNs, too. CNN is a very important type of ANN that is especially used for finding spatially correlated patterns in images while sharing weights and excluding the need of full connectivity between units. A deep neural network (DNN) is an ANNs with multiple hidden layers between the input and output layers[86, 92]. Similar to shallow ANNs, DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives[93]. The extra layers enable composition of features from lower layers, potentially modelling complex data with fewer units than a similarly performing shallow network.[86]

1.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks are similar to normal neural networks. They are made up of neurons that can learn weights and biases. Each neuron receives some input, performs a dot product, and, sometimes, follows with a non-linearity function. The whole network expresses a single differentiable score function - from raw image pixels to class scores. CNN architectures assume that the inputs are images, which make it possible to encode certain properties into the architecture. It makes the forward function more efficient to implement, and it reduces the number of parameters in the network. [94]

The problem about regular neural networks is that it doesn't scale well to full images. An example is the CIFAR-10 [95], where the images are only of the size $32 \times 32 \times 3$ (32 wide, 32 high, 3 colour channels). A single fully-connected neuron in the first hidden layer of a regular Neural Network would have $32 \cdot 32 \cdot 3 = 3072$ weights. For bigger sizes images, e.g. $200 \times 200 \times 3$, it would lead to neurons that have $200 \cdot 200 \cdot 3 = 120000$ weights. This full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting. Convolutional neural networks take advantage of the fact that the input consists of images. The layers of a CNN have neurons arranged in 3 dimensions: width, height, depth. In the example of the CIFAR-10 [95] the input is a volume of activations, and the volume has the dimensions 32x32x3 (width, height, depth respectively). The neurons in a layer will only be connected to a small region of the previous layer, unlike the fully connected manner in regular NN. In order to visualize this difference, the Figure 1.8 with the feed-forward structure of an ANN can be compared as in the Figure Figure 1.9.

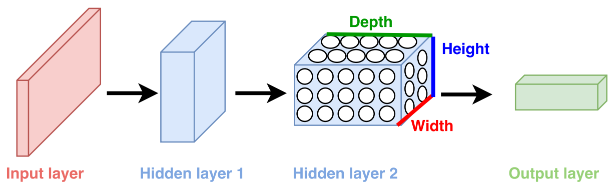


Figure 1.9: *A Convolutional Neural Network arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a CNN transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels). Image taken from [94]*

.

1.3.1 Layers

As described earlier a CNN is a combination of layers and every layer transforms one volume of activations into another through a differentiable function. CNN uses three main types of layers to build the architecture: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks on Figure 1.8). These layers will be stacked to form a full CNN architecture. In reinforcement learning the pooling layer is not used because they buy translation invariance - the network becomes insensitive to the location of an object in the image.

Convolutional Layer

The Convolutional layer is the main building block of a convolutional neural network. It does most of the computational heavy lifting. The convolutional layer consists of filters which learn. Every filter is spatially small (along width and height), but extends to the full depth of the input volume. An example of the first layer in a CNN is a filter with the size $5 \times 5 \times 3$. During the first forward pass, the data is slide/convolved through each filter across the height and width of the input volume, and the dot products between the entries of the filter and the input at any position are computed. As the filter slides over the input volume it produces a 2-dimensional activation map. The activation map shows the responses of that filter at all spatial positions. The network will learn filters that activate when they see some type of visual features - such as an edge of some orientation or a patch of some colours. On higher layers the network will learn to see more complex patterns: it could be honeycomb or wheel-like patterns. On each convolutional layer, there will be an entire set of filters, each layer will produce a separate 2-dimensional activation map. The activation maps will be stacked along the depth dimension and produce the output volume. When dealing with high dimensional inputs like images, it is impractical to connect neurons to all the neurons in the previous volume. A smarter solution is to connect each neuron to only a local region of the input volume instead. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron - equivalently this is the filter size. An illustration of the receptive field can be seen on 1.10.

Spatial arrangement

The connectivity of each neuron in the convolutional layer to the input volume is described by the spatial arrangement. Spatial arrangement describes how many neurons there are in the output volume and how they are arranged. Three hyperparameters control the size of the output volume - depth, stride, and zero-padding.

The first hyperparameter is the *depth.* It corresponds to how many filters

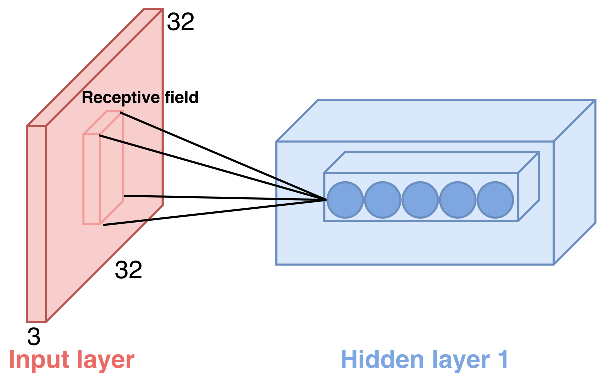


Figure 1.10: *An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer in blue. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all colour channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input. Image taken from [94]*

the convolutional layer uses. Each filter looks for something different in the input. For example, the convolutional layer takes a raw image as an input, then the different neurons along the depth dimension may activate in the presence of various oriented edges or blobs of colours. A set of neurons that look at the same region of the input is called *depth column* or *fibre.*

Another hyperparameter is the *stride.* It defines how the filters slide over the input. If the stride is 1, then the filters move one pixel at a time. When the stride is 2, then the filters move 2 pixels at a time. This will produce spatially smaller output volumes.

The last hyperparameter to control the size of the output volume is the size of zero-padding. Zero-padding pads the input volume with zeros around the border. The good feature of zero-padding is that it controls the spatial size of the output volumes. This is useful to preserve the spatial size of the input volume so that the input and output width and height are the same.

The way to compute the spatial size of the output volume is by using a function of the input volume size (**W**), the receptive field size of the convolutional layer neurons (**F**), the stride with which they are applied (**S**), and the amount of zero padding used (**P**) on the border. The formula for calculating how many neurons "fit" is the following:

$$\frac{W - F + 2P}{S} + 1 \tag{1.1}$$

An example for a 5x5 input and a 3x3 filter with stride 1 and zero-padding 1 the output would be of the spatial size 5x5:

$$\frac{5 - 3 + 2 \cdot 1}{1} + 1 \rightarrow \frac{4}{1} + 1 = 5 \tag{1.2}$$

And with stride 2 the output would be 3x3:

$$\frac{7 - 3 + 2 \cdot 1}{2} + 1 \rightarrow \frac{4}{2} + 1 = 3 \tag{1.3}$$

The visualization is provided on the figure below Figure 1.11:

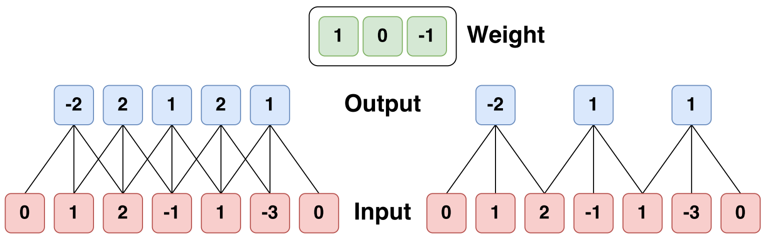


FIGURE 1.11: *Illustration of spatial arrangement. The example is described above. In this example, there is only one spatial dimension (x-axis), one neuron with a receptive field size of F = 3, the input size is W = 5, and there is zero-padding of P = 1.* ***Left:*** *The neuron strides across the input in stride of S = 1.* ***Right:*** *The neuron uses stride of S = 2. The neuron weights are in this example [1,0,-1] (shown on very right), and its bias is zero. These weights are shared across all yellow neurons. Image taken from [94].*

1.3.2 Fully connected Layer

The fully connected layer in CNNs is a traditional Multi-Layer Perceptron. The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron in the next layer as seen on Figure 1.8. The output from the convolutional layers represents a high-level feature of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input images into various classes.

Apart from classification, the fully connected layer is also a cheap way to learn non-linear features. By combining these features, the classification of the network would be even better [96].

1.4 Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNN) are used when the patterns in data change with time. RNNs have a simple structure with a built-in feedback loop allowing it to act as a forecasting engine. RNNs are applied in a large range of applications, from speech recognition to driver-less cars.

In feed-forward neural networks the data flows in one direction only, whereas in RNN the output of the layer is added to the input of the same layer, and this layer represents the whole network. This results in a loop-like network. The flow can be viewed or interpreted as a time passage where at each time-step the same layer receives it's own output from the previous time-step and adds it up to the input part together with the new data received [97].

Unlike feed-forward ANNs, RNNs can work with sequences of data inputs and, subsequently, to output sequences of data in return. Not only RNNs use sequence of data but also these sequences can vary in their size, so different sizes of sequences can be adapted by the RNN dynamically. Another key feature of the RNN is the dependency of the training examples. Unlike feed-forward ANNs, where the training examples are independent of each other, the RNNs treat temporal dependencies, meaning that a sequence of e.g. words is usually dependent on what came before [98]. These new features open a new range of applications like image captioning (single input, sequence output), document classification (sequence input, single output), video frames classification (sequence input, sequence output), demand and supply chain planning forecasting (with added time delay) [97].

In order to understand better the recurrent neuron functionality, the Figure 1.12 presents a comparison between the RNN unit and the linear unit used in feed-forward ANNs.

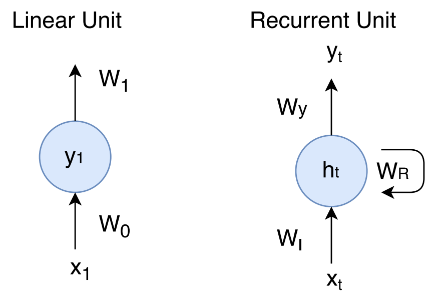


Figure 1.12: *Linear vs. Recurrent unit*

A linear unit's output is the input x_j times the weight matrix W_{ij} which is then passed to an activation function g.

$$y_i = g(\sum_j W_{ij}x_j + b) \tag{1.4}$$

The recurrent unit is composed of a linear unit, but then it adds a recurrent weight W_R, therefore the output $h^{(t)}$ depends on both the input x_t and the activity at the previous time-step. To retrieve an output y_t from the layer, a non linear activation function g_y is applied to the $h^{(t)}$ [98].

$$\begin{aligned} h^{(t)} &= g_h(W_I x^{(t)} + W_R h^{(t-1)} + b_h) \\ y^{(t)} &= g_y(W_y h^{(t)} + b_y) \end{aligned} \tag{1.5}$$

For example, in case of word prediction, the input of a unit can be a word, and the output of that unit would be the predicted next word that follows the one that was input; then, when the next input comes in at the next time-step, the process is applied, but also with the activity at the previous time-step taken into account.

The training of RNNs is different than that of the other ANNs, because RNNs emit an output at each time-step, and, therefore, there is a cost function at each time-step, whereas before, in feed-forward networks, it was necessary to run the input through the whole network in order to get an output comparable to the one and only cost function. Another special characteristic of the RNNs is that the weights W_R are shared across the network. So, the gradients from all the time-steps can be combined together to obtain the weights update and do back-propagation. But, because of the shared weights, the update would scale with the size of the W_R, the back-propagation has to be done all the way until time-step zero, which causes the problem of vanishing and exploding gradients [98].

1.4.1 Gating in RNN

In order to address the problems of training deep RNNs, there are different solutions available. Among the known solutions are the use of the *Root Mean Square Propagation* (RMSProp) optimizer for learning rate adjustment, clipping gradients, ReLu activation functions, special weight initialization, or to use *gating* [98]. Gating is a technique for deciding when to forget the current input, and when to remember it for future time steps [97]. The most famous approaches are *long short-term memory* (LSTM) and *gated recurrent units* (GRU).

LSTM has at its core a memory cell C with a recurrent weight $W_C = 1$ that inherits the activity of the previous time-step. This memory cell has three manipulations: *forget* (flush the memory), *input* (add to the memory), and *output* (retrieve from the memory) [98]. The activity of the memory cell C_t is taken and passed through a tanh activation function and multiplied by the *gate*. The gate is an affine layer, or a 'standard' feed-forward ANN layer, that has as inputs x_t at the current time-step and h_{t-1} of the previous time-step that are together multiplied by the weight W_0, summed and passed through the sigmoid activation function σ to return a vector of numbers between 0 and 1. The gate operation for generating the *output* h_t, or for getting the information from the memory cell C_t, is presented in the following equation:

$$\begin{aligned} h_t &= \sigma(W_0 \cdot [x_t h_{t-1}] + b_0) \odot \tanh(C_t) \\ &= o_t \odot \tanh(C_t) \end{aligned} \tag{1.6}$$

The same approach is used for the *forget* operation f_t. A layer before the final output is introduced, but with different weights, W_f. If the output of the forget gate is 0, means that the memory has been flushed completely, whereas if the output is all 1s then the memory retained everything [98].

The *input* for the next time-step has two affine layers, one for generating new input $\widetilde{C_t}$ for the memory cell with the weights W_C and tanh activation function, and another one i_t that modulates the input and writes it into the memory cell with the weights W_i and the sigmoid activation function σ.

All the components of the LSTM structure are vectors with numbers between 0 and 1, and they can be handled to perform either of the available manipulations. The summarized presentation of the LSTM components with the corresponding affine layers is illustrated in the Figure 1.13 [98].

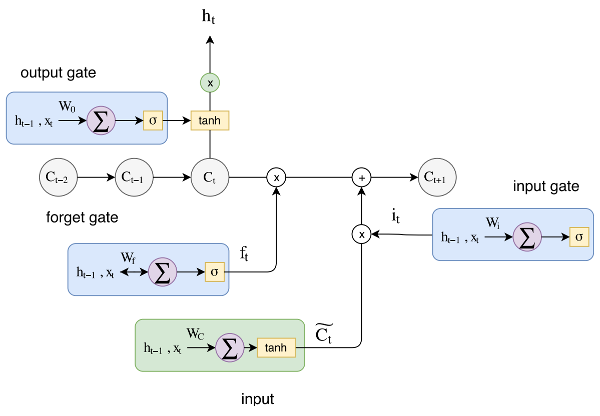


FIGURE 1.13: *LSTM functionality*

Having explained the LSTM functionality, the formula for computing the next time-step activity C_{t+1} is the following:

$$C_{t+1} = f_t \odot C_t + i_t \odot \widetilde{C_t} \tag{1.7}$$

The GRU gating approach is a simplified version of the LSTM. It actually combines all the gates into a single update:

$$\begin{aligned} h_t &= (1 - z_t) \cdot h_{t-1} + z_t \cdot \widetilde{h_t} \\ z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \text{ (update gate)} \\ \widetilde{h_t} &= \tanh(W \cdot [r_t \cdot h_{t-1}, x_t]) \text{ (input gate)} \\ r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \text{ (remember gate)} \end{aligned} \tag{1.8}$$

1.5 Reinforcement Learning

One of the primary goals of Artificial Intelligence is to produce fully autonomous agents that interact with their environments to learn optimal behaviours, improving over time through trial and error. Crafting AI systems that are responsive and can effectively learn has been a long-standing challenge, ranging from robots, which can sense and react to the world around them, to purely software-based agents, which can interact with natural language[99]. One of the approaches which council the best to this paradigm is Reinforcement Learning [3].

Reinforcement learning (RL) is an approach in artificial intelligence for goal-directed learning from interaction and experience. This makes it different from the other approaches in machine learning in which the learner, the decision maker, or the so-called agent, is told what to do. In reinforcement learning the agent tries out different actions in order to understand which of them generates the most reward. The reward is a special term in reinforcement learning and describes the goal in a Markov decision process (MDP) model. Roughly speaking, the MDP model would very well characterize the agent's view of the world, the actions that it can take in the world and its goal.

Reinforcement learning considers the problem of planning in real-time decision making and the models for prediction related to planning. The interactive goal-directed agent is able to operate in an uncertain setup, make decisions despite uncertainty and predict future events. The agent is not necessarily a robot; it can be any component in a larger system in which it interacts directly with the system and indirectly with the system's environment. The environment is everything that the agent interacts with, it is the outer world.

There is a special concern in reinforcement learning which is not present in the other machine learning approaches. It is the issue of balancing exploitation of the knowledge that the agent has and exploration of new information in order to improve the current knowledge base.

A variety of different scientific fields intersects with reinforcement learning, especially mathematics, namely, statistics and optimization, which have an important background contribution to the reinforcement learning methods. Some reinforcement learning methods are able to learn with parametrized approximators which addresses the classical *curse of dimensionality* in operations research and control theory [3]. The relationship between reinforcement learning and optimization can be exemplified by the idea of maximization of the reward signal. Actually, in reinforcement learning the agent intends to maximize the reward, but not necessarily achieves the maximum. Reinforcement learning is also part of the engineering and computer science subjects. The related algorithms have a close resemblance to the biological brain systems of animals and humans due to the reward factor involved, therefore it also binds with the psychology and neuroscience fields.

1.5.1 Elements of an RL problem

A reinforcement learning problem contains at least one of the elements: reward signal, value function, policy, environment model.

The reward signal represents a feedback from the environment as a response to the agent's behaviour in that environment. Therefore the agent cannot change the feedback that it receives, but it can behave accordingly so as to maximize the gained reward signals during its lifetime. The reward signal defines the goal in a reinforcement learning problem [3]. It serves as a problem definition and as a basis for modifying the policy.

The policy maps states to actions so that when the agent is in a specific state, it chooses an action based on the defined policy. A policy is enough to describe the behaviour of the agent and therefore, it is the core of reinforcement learning.

The value function provides values for judging the quality of a state based on the estimated maximum reward it can yield in the long run, in contrast with the reward which expresses only the immediate advantage of being in a specific state.

The model is a representation of the environment's behaviour. In a model-free reinforcement learning (trial-and-error) problem the agent cannot plan its future because it does not have a model basis, whereas in model-based problems the agent can plan its future actions based on the environment's modelled behaviour and expected rewards in certain states.

1.5.2 Markov Decision Processes in RL

The general reinforcement learning problem formulation has the format of a finite MDP. The interaction between the agent and environment happens at each time-step of a sequence of discrete time-steps, $t = 0, 1, 2, 3, ...$, where at each time-step t the agent receives a representation of the world - a state, $S_t \in S$, from a set of possible states S, selects an action A_t from a set of possible actions $A(S_t)$ for the state S_t by implementing a policy π_t, where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$, and in the next time-step $t + 1$ the agent receives a reward signal $R_{t+1} \in R$ from the environment ending up in a new state S_{t+1} [3]. The diagram in Figure 1.14 illustrates the interaction between the agent and the environment.

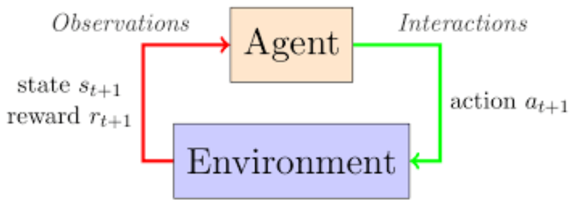


FIGURE 1.14: *Agent - environment interaction. Image taken from* `https://kofzor.github.io/Reinforcement_Learning_101/`

Reinforcement learning methods provide ways to adjust the policy based on the accumulated experience with the goal of maximizing the total cumulative reward in mind. An example for representing the goal in a reinforcement learning problem like that of making a robot learn to walk would be by giving a reward on each time-step proportional to the robot's forward motion. The reward signal is the way of communicating to the agent *what* you want it to achieve, not *how* you want it achieved [3].

A formal definition of the cumulative reward received in the long run is expressed by the expected return G_t, which is a function of rewards sequence $R_{t+1}, R_{t+2}, ..., R_T$ received after the time-step t, where T is the last time-step. In order to express the return more conveniently, the concept of *discounting* is introduced, which determines the current value of the future rewards. The formula generalized for both episodic and continuing tasks is the following:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}, \tag{1.9}$$

where γ is the discount rate, $0 \leq \gamma \leq 1$. In the case of episodic tasks where there is a terminal state after some time-steps, $\gamma = 1$. For the cases in which the process is continuous and the final step is infinite, $T = \infty$.

With the discounting factor, the reward received after k time-steps has the value γ^{k-1} times what it would be worth if it were received immediately [3]. In the extreme point where $\gamma = 0$, it is said that the agent is *myopic*, because it only maximizes over the immediate rewards and not the future rewards, whereas if γ is closer to 1 the agent is far-sighted and sees far into the future considering the future rewards when picking actions.

The *state* that has the Markov property represents all the useful information in order to make a sufficient statistic for the future. With Markov states, we have the best possible basis for choosing an action [3]. The environment's feedback at time-step $t+1$ after a particular action was taken at time-step t depends on the events happened before. If the state has the Markov property instead, then the feedback of the environment depends only on that state, because that state represents all the previous events. In this case, the one-step environment dynamics of a finite MDP can be expressed by the following formula:

$$p(s', r|s, a) = Pr\left\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\right\} \tag{1.10}$$

Based on the formula presented in (1.10) we can also compute the expected rewards for state-action pairs [3],

$$r(s, a) = \mathbb{E}\left[R_{t+1}|S_t = s, A_t = a\right] = \sum_{r \in R} r \sum_{s' \in S} p(s', r|s, a) \tag{1.11}$$

the *state-transition probabilities*

$$p(s'|s, a) = Pr\left\{S_{t+1} = s'|S_t = s, A_t = a\right\} = \sum_{r \in R} p(s', r|s, a) \tag{1.12}$$

and the expected rewards for state-action-next-state triples,

$$r(s, a, s') = \mathbb{E}\left[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'\right] = \frac{\sum_{r \in R} r p(s', r|s, a)}{p(s'|s, a)} \tag{1.13}$$

Value functions estimate how good it is to be in a specific state given the expected return and the policy.

The *state-value function* v_π for the policy π expresses the expected value of a random variable given the followed policy π at any time-step t:

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t|S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \tag{1.14}$$

The *action-value function* q_π for the policy π is the value of taking an action a in a state s while following the policy π:

$$q_\pi(a, s) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{1.15}$$

Value functions have the property of being expressed recursively. The recursive representation is actually the *Bellman equation* and its solution is the value of v_π. It represents the basis of different ways of computing, approximating, and learning v_π. The *Bellman equation* is kind of a look-ahead procedure, where the value of a current state is evaluated by looking ahead at the values that future states can offer. It averages over all the possibilities, weighting each by its probability of occurring. It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way [3]:

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right]), \forall s \in S \tag{1.16}$$

It is a sum over all values of the three variables, a, s', and r. For each triple, we compute its probability, $\pi(a|s)p(s', r|s, a)$, weight the quantity in brackets by that probability, then sum over all possibilities to get an expected value[3].

In finite MDPs, an *optimal policy* π_* is the policy for which its expected return for all the states is greater than or equal to the expected return of all the other policies. There can be many optimal policies, but the evaluation of their optimal state-value functions have the same result. In other words, any optimal policy evaluates to the same *optimal state-value function* v_*, which is presented in the following equation:

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in S \tag{1.17}$$

The *optimal action-value function* q_* evaluates the same for all the optimal policies as well and it is of the following form:

$$q_*(s, a) = \max_\pi q_\pi(s, a), \forall s \in S, a \in A(s) \tag{1.18}$$

Consequently, the optimal value function v_* or the optimal action-value function q_* can lead to an optimal policy and these can be found using the *Bellman optimality equations.*

The *Bellman optimality equation* for v_* is the value of a state on the optimal policy basis, which is the same as the expected return of the best action for that state [3]:

$$\begin{aligned} v_*(s) &= \max_{a \in A(s)} q_{\pi*}(s,a) \\ &= \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right] \\ &= \max_{a \in A(s)} \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_*(s')\right] \end{aligned} \tag{1.19}$$

And the *Bellman optimality equation* for q_* is the following:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') | S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right] \end{aligned} \tag{1.20}$$

For a better understanding of the optimality equations, (1.19) and (1.20), the backup diagrams are provided in Figure 1.15. The backup diagrams illustrate how the update happens by showing the components that are taken into account for the value evaluation.

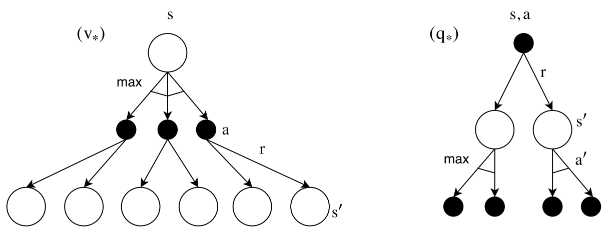


Figure 1.15: *The backup diagrams for v_* and q_**

The Bellman optimality equation generates a system of N non-linear equations, where N is the number of states. It can be simply solved by applying some non-linear methods when the system dynamics ($p(s',r|s,a)$) are known. The solution to the Bellman optimality equation helps in defining the optimal policy; e.g. one can, at any state, choose the action that corresponds to the maximum return, which is also valid in the long term, because the values take into account the reward consequences of all possible future behaviour options [3]. In the case of the action-value pairs, if the system's dynamics are unknown, then the actions would still be optimal because the agent would choose the actions that would maximize q_*.

1.6 Computer Vision

Computer Vision (CV) is a discipline that deals with how computers can be made for gaining high-level understanding from digital images or videos. Computer Vision has a dual goal. It is relevant from biological science point of view, because it aims to come up with computational models of the human visual system, and from the engineering point of view, because of it aims to build autonomous systems which could perform some of the tasks which the human visual system can perform (and even surpass it in many cases). Many vision tasks are related to the extraction of 3D and temporal information from time-varying 2D data such as obtained by one or more television cameras, and more generally the understanding of such dynamic scenes. Of course, the two goals are intimately related. The properties and characteristics of the human visual system often give inspiration to engineers who are designing computer vision systems. Conversely, computer vision algorithms can offer insights into how the human visual system works[100].

1.6.1 History and state of the art

Many researchers involved in Computer Vision agree that the father of Computer Vision is Larry Roberts, who in his Ph.D. thesis (1963) at MIT discussed the possibilities of extracting 3D geometrical information from 2D perspective views of blocks (polyhedra) [100]. Later in that decade, in 1966, MIT involved its students in a summer project, which requires to attach a camera to a computer and having it *describe what it saw* [101]. Many researchers, at MIT and elsewhere, in Artificial Intelligence, followed this work and studied computer vision in the context of the blocks world. Later, researchers realized that it was necessary to tackle images from the real world. Thus, much research was needed in the so called *low-level* vision tasks such as *edge detection* and *segmentation*. A major milestone was the framework proposed by David Marr (1982) at MIT, who took a bottom-up approach to scene understanding [102] . This is probably the single most influential work in computer vision ever because it has found a paradigm which is not easy to substitute or modify.

In the next years, Computer Vision has grown from studies usually originated from various other fields, and consequently there is no standard formulation and no standard solving of *the computer vision problem.* Instead, there exists an abundance of methods for solving various well-defined computer vision tasks, where the methods often are very task specific and seldom can be generalized over a wide range of applications[100]. Many of the methods and applications are still in the state of basic research, but more and more methods have found their way into commercial products, where they often constitute a part of a larger system which can solve complex tasks.

The major contributor of Computer Vision field has been the robotic application field. In fact, robotics and AI usually deals with autonomous planning or deliberation for system which can perform mechanical actions such as moving a robot through some environment. This type of processing typically needs input data provided by a computer vision system, acting as a vision sensor and providing high-level information about the environment and the robot.

A field which plays an important role for Computer Vision is neurobiology, specifically the study of the biological vision system. Over the last century, there has been an extensive study of eyes, neurons, and the brain structures devoted to processing of visual stimuli in both humans and various animals. This has led to a coarse, yet complicated, description of how "real" vision systems operate in order to solve certain vision related tasks. These results have led to a sub-field within computer vision where artificial systems are designed to mimic the processing and behaviour of biological systems, at different levels of complexity. Also, some of the learning-based methods developed within computer vision have their background in biology.

Many of the related research topics can also be studied from a purely mathematical point of view. For example, many methods in computer vision are based on statistics, optimization or geometry. Finally, a significant part of the field is devoted to the implementation aspect of computer vision; how existing methods can be realized in various combinations of software and hardware, or how these methods can be modified in order to gain processing speed without losing too much performance.

1.6.2 Current applications of Computer Vision

The applications of computer vision are numerous and various because of the nature of the problem. Humans use vision to extract features and other relevant informations from the environment which is around them. At the same way computers and machines need to gain informations also from their vision sensors to evaluate the goodness of a certain task or to plan new tasks. Examples of application of Computer Vision are:

- *Agriculture*: include all the application that involve the agriculture application field. CV aims at developing vision-based algorithms to improve efficiency and quality in agricultural applications. In [103], for example, it is described are described two case studies are analyzed dealing with the harvest of radicchio and the post-harvest of fennel. In [104], they have described how to construct a commercial agricultural manipulation for fruit picking and handling without human intervention.
- *Augmented Reality*: in [105], there are explained some of the applications that regards AR and Computer Vision. From an algorithmic point of view, the development of Computer Vision solutions can be split in three layers. At lowest level there is the image acquisition and the processing for basic feature extraction. On top of this there is an intermediate-level vision processing layer: here, the object/feature recognition and tracking can be carried out, including 3D scene modelling, and reconstruction. Finally, at the top level of the processing pyramid there is the so-called high-level vision. Here, the interpretation of the evolving information provided by the intermediate processing layers can be carried out. The acquisition of this high level information can be then used as a feedback

for the intermediate and low-level tasks through representation of

- *Autonomous Vehicles*: include submersibles, land-based vehicles (small robots with wheels, cars or trucks), aerial vehicles, and unmanned aerial vehicles (UAV). The level of autonomy ranges from fully autonomous (unmanned) vehicles to vehicles where computer vision based systems support a driver or a pilot in various situations. Fully autonomous vehicles typically use computer vision for navigation, i.e. for knowing where it is, or for producing a map of its environment (SLAM) and for detecting obstacles. It can also be used for detecting certain task specific events, e. g., a UAV looking for forest fires. Examples of supporting systems are obstacle warning systems in cars, and systems for autonomous landing of aircraft. Several car manufacturers have demonstrated systems for autonomous driving of cars, but this technology has still not reached a level where it can be put on the market. There are ample examples of military autonomous vehicles ranging from advanced missiles, to UAVs for reckon missions or missile guidance. Space exploration is already being made with autonomous vehicles using computer vision, e. g., NASA's Mars Exploration Rover.

- *Biometrics*: Biometrics has the capability to identify or verify each and every individual correctly by using physiological (face, fingerprint, iris, ocular region, palm-print, knuckle, ear, retina, footprint, gait, DNA, dental biometrics etc.) or behavioural characteristics (signature, voice, etc.) possessed by the user. In [106], there is an application of Computer Vision for the recognition of ear that was proposed as a supplement to existing methods.

- *Medical Image Analysis*: characterized by the extraction of information from image data for the purpose of making a medical diagnosis of a patient. Generally, image data is in the form of microscopy images, X-ray images, angiography images, ultrasonic images, and tomography images. An example of information which can be extracted from such image data is detection of tumours, arteriosclerosis or other malign changes. It can also be measurements of organ dimensions, blood flow, etc. This application area also supports medical research by providing new information, e.g., about the structure of the brain, or about the quality of medical treatments.

- *Military applications*: one of the largest areas for computer vision. The obvious examples are detection of enemy soldiers or vehicles and missile guidance. More advanced systems for missile guidance send the missile to an area rather than a specific target, and target selection is made when the missile reaches the area based on locally acquired image data. Modern military concepts, such as "battlefield awareness", imply that various sensors, including image sensors, provide a rich set of information about a combat scene which can be used to support strategic decisions. In this case, automatic processing of the data is used to reduce complexity and to fuse information from multiple sensors to increase reliability.

There are many other applications that are fundamental for Computer Vision. All of these can be classified as problems that require a certain degree of recognition in order to capture a meaning from an image or a sequence of image that comes from a vision sensor.

Chapter 2

AI methodologies for Autonomous Driving

AI researchers and developers are now living in a period of great innovation. Studies related to AI and ANNs are developing very fast and every month new ideas are proposed. The subject is attracting an increasing amount of investments both in research and industry. A prominent field in AI that is currently growing fast is that of autonomous driving.

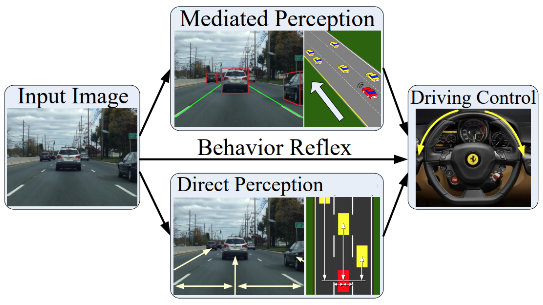


Figure 2.1: *Three paradigms for autonomous driving. Image taken from [107]*

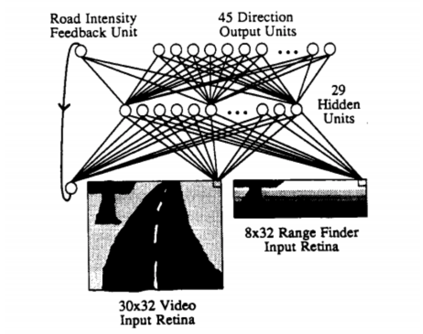
In autonomous driving context, there are three main approaches (see Figure 2.1) that have been proposed to interface the AI to the car and which are based on the way related data are processed[107]. The *Mediated Perception* is an approach where the structure of the environment is assumed to be unknown and the important features are detected using different techniques in the environment. These approaches are usually based on AI engine that processes all the informations and take the driving decisions [107, 108]. Moreover, it combines computer vision, sensor fusion, localization, control theory, and path planning and it is the *classical* approach on which companies are building their own Self-Driving car (e.g. Google).

Another different approach is *Behaviour Reflex* in which an ANNs model is trained to make driving decisions from monitoring the driving behaviour of a human driver in reaction to different driving scenarios [1, 107, 109]. This approach is sometimes called *behavioural cloning* or *end-to-end driving.*

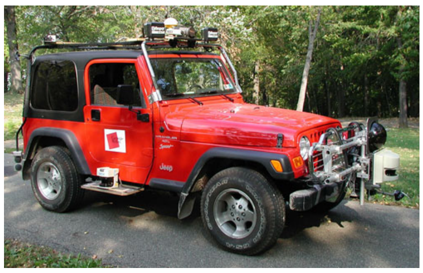
The third approach is called *Direct Perception*, which was proposed in [107]. In this approach, CNN learns by extraction of some preselected features from the scene that the authors believe are important to make driving decisions and subsequently this information is processed by a simple controller to make the corresponding driving decisions. Moreover, this approach assumes full knowledge of the road architecture for training purposes.

2.1 History and State of the art

Traditional vision and robotic techniques have always some trouble when they are requested for autonomous navigation tasks because of the noise and variability associated with real world scenes. A motorway or a city street could offer different condition that are unusual for both of them. In real world, there exist animals, people and other kind of natural factor that may affect how humans drive a car. Traditional image processing and pattern recognition techniques could be a solution to perform and process information well under certain conditions but have problems with others. Systems suffer this type of techniques because the processing performed remains fixed across various driving situations. *ALVINN* (*Autonomous Land Vehicle In a Neural Network*) is one of the first proposal which approach the application of ANNs in autonomous driving context[1]. Specifically, ALVINN is an ANN designed to control the NAVLAB, the Carnegie Mellon autonomous navigation test vehicle (see Figure 2.2b). ALVINN takes images from a camera and a laser range finder as input and produces as output the direction the vehicle should travel in order to follow the road. Successful tests on the NAVLAB indicate that the network can effectively follow real roads under certain field conditions. Even if the neural network powering ALVINN was really well implemented, it was constrained very much by the hardware. ALVINNs original top speed was 3.5mph, which was limited by the amount of computing power they could fit in the vehicle[1].


Road Intensity Feedback Unit
45 Direction Output Units
29 Hidden Units
8x32 Range Finder Input Retina
30x32 Video Input Retina

(a)



(b)

Figure 2.2: *(a) ALVINN architecture. Image taken from[1]. (b) NAVLAB. Image taken from* `https://www.cs.cmu.edu/afs/cs/project/alv/www/`

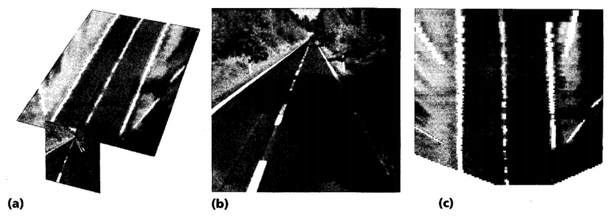
ALVINN was just the beginning of modern autonomous driving systems. In the following subsection, some key milestones in the modern autonomous driving context are described in order to introduce the concepts that are the basis of the current work.

ì

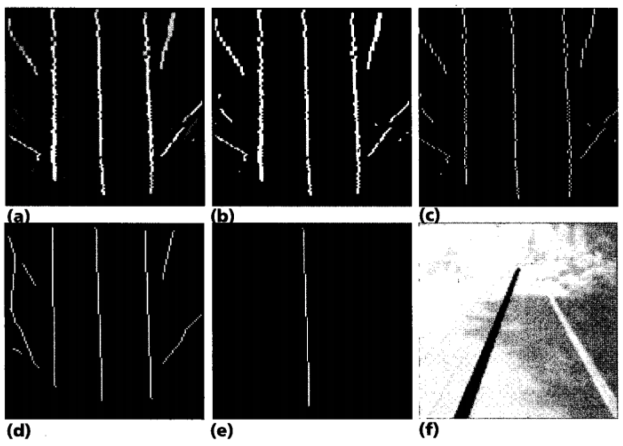
2.1.1 Traditional Image Processing

Another type of project that has affected modern autonomous driving systems was ARGO project[15]. ARGO is a passenger car with a vision-based system for extracting road and environmental information from the acquired images, using different output devices to test the automatic features. It acquires data from only passive sensors like cameras and a speedometer in order to sense the surrounding environment. One of the project's aims was to develop a system inexpensive enough to ease its integration into a large number of vehicles. ARGO's stereoscopic vision system consists of two small (3.2×3.2 cm), low-cost cameras that can acquire pairs of grey-scale images. The angle of view under which a scene is acquired and the distance of the objects from the camera contribute to associate a different information content to each pixel of an image. Image processing must take the perspective effect into account to weigh each pixel according to its information content. They have used a geometrical transform called *inverse perspective mapping*[110] which remove the perspective effect from the acquired image, remapping it into a new 2D domain that homogeneously distributes the information among all the pixels. Assuming the road in front of the vision system is known, IPM makes it possible to obtain a bird's-eye view of the scene (see Figure 2.3a).

The first phase of lane detection developed in ARGO project is to apply IPM to the grey-scale image obtained from vision sensors and obtain constant width lines which represent road markings as illustrated in Figure 2.3b which shows the lane-detection steps for the Figure 2.3a. Then, it searches for dark-bright-dark horizontal patterns of a given size. The result encodes the horizontal brightness transitions and the presence of lane markings. Taking advantage of the lane markings' vertical correlation, the system enhances the result of Figure 2.3b(a) . Different illumination conditions and the non-uniformity of painted road signs necessitate an adaptive threshold' that works on a 3x3-pixel neighbourhood (see Figure 2.3b(b)). The result was thinned (see Figure 2.3b(c)) and scanned row by row to build chains of non-zero pixels (see Figure2.3b(d)). Through an iterative process, the system approximates each chain with a polynomial line made of one or more segments. Polynomial lines then are filtered and selected if it is the best approximation of the results of the previously processed image (see Figure 2.3b(e)). The model



(a)



(b)

FIGURE 2.3: *Images taken from [15]*

assumed for the external environment (a flat road) lets to determine the spatial relationship between image pixels and the 3D world in order derive both the road geometry and the vehicle's lane position[15].

ARGO drove about 2,000 km journey from 1 to 6 June 1998 in Italian motorways. The tour demonstrated that using only visual information and low-cost general purpose hardware it is possible to drive automatically and safely a vehicle under different road and environmental conditions[24].

2.1.2 Deep Learning

In 2007, DARPA Grand Challenge III has been won by Tartan Racing, a team of students from Carnegie Mellon University. In this challenge, DARPA ensured a certain level of success by carefully managing scope: participants agreed to a set of rigorously defined traffic rules, and DARPA eliminated pedestrian and cyclist traffic from the challenge. The DARPA challenge highlighted the need for more advanced computational power and algorithm development. At the time, teams relied heavily on rules-based programming techniques, which meant robotic systems tended to operate only in very constrained environments, around well-behaved road users that would not deviate much from an established set of rules. In 2007, NVIDIA starts developing *CUDA* and related library *cudNN*. CUDA is a parallel computing platform and API model that lets developers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing. This contribute has led the reborn of Deep Learning and DNN. Training a DNNs, which use GPUs, has led to same results in less time due to the increased computational power of GPU against the CPU.

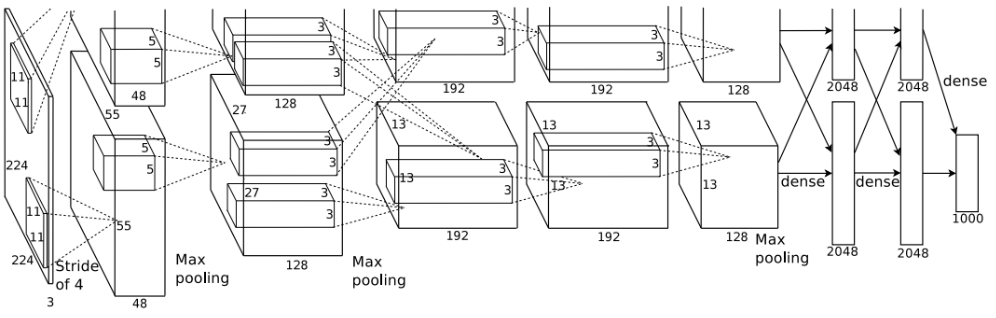
Image Recognition

GPUs give rise to the use of Deep Neural Network in Image and Pattern Recognition. Since 2010, the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a competition where research teams evaluate their algorithms on the given data set, and compete to achieve higher accuracy on several visual recognition tasks. The ImageNet project is a large visual database designed for use in visual object recognition software research, which contains over ten million URLs of images that have been hand-annotated by ImageNet to indicate what objects are pictured.

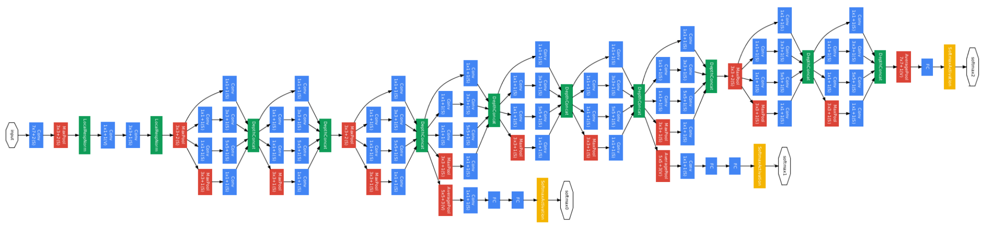
The 2010s saw dramatic progress in image processing. Around 2011, a good ILSVRC classification error rate was 25%. In 2012, *AlexNet* developed by Alex Krizhevsky et al [111] came along. It significantly outperformed all the prior competitors and won the challenge by reducing the top-5 error to 15.3%. The second place top-5 error rate, which was not a CNN variation, was around 26.2%. The network had a very similar architecture as *LeNet* by Yann Lecun et al[112] but was deeper, with more filters per layer, and with stacked convolutional layers. *AlexNet* was trained simultaneously on two Nvidia Geforce GTX 580 GPUs . This success in particular, among others, resulted in a huge increase in popularity of CNNs but also ANNs in general. Not surprisingly, the ILSVRC 2013 winner was also a CNN which became known as *ZFNet*. It achieved a top-5 error rate of 14.8%. It was mostly an achievement by improving the hyper-parameters of *AlexNet* while maintaining the same structure with additional Deep Learning elements as discussed earlier in this essay. The winner of the ILSVRC 2014 competition was *GoogLeNet* from Google [113] which achieved a top-5 error rate of 6.67%.The network used a CNN inspired by *LeNet* but implemented a novel element which is dubbed an inception module. This module is based on several very small convolutions in order to drastically reduce the number of parameters. Their architecture is composed by 22 layer deep CNN and they have reduced the number of parameters from 60 million (*AlexNet*) to 4 million.

However, the runner-up at the ILSVRC 2014 competition is dubbed *VGGNet* developed by Simonyan and Zisserman. It consists of 16 convolutional layers and a very uniform architecture. It only performs $3 \times 33 \times 3$ convolutions and $2 \times 22 \times 2$ pooling all the way through. The weight configuration of the *VGGNet* is publicly available and has been used in many other applications and challenges as a baseline feature extractor. However, *VGGNet* consists of 140 million parameters, which can be a bit challenging to handle. At last, at the ILSVRC 2015 in December, the so-called Residual Neural Network (*ResNet*) by Kaiming He [114] et al introduced a novel architecture with "skip connections" and features heavy batch normalization. Such skip connections are also known as gated units or gated recurrent units and have a strong similarity to recent successful elements applied in RNNs. Thanks to this technique they were able to train a NN with 152 layers while still having lower complexity than *VGGNet*. It achieves a top-5 error rate of 3.57% which beats human-level performance on this dataset.

The residual learning framework bears a strong resemblance with advancements in RNN. In the 2015 ILSVRC challenge, there was a record set by a network which is composed by two parts: the first network is based on the *GoogLeNet* inception architecture and is dubbed Inception-v4 and



(a) *AlexNet. Image take from [111]*



(b) *GoogLeNet. Images taken from [113]*

FIGURE 2.4

a second network is called *Inception-ResNet-v2* which is incorporating the key elements of *ResNet*. Both networks achieve very similar results. While the first inception variation achieves 3.8% top-5 error the latter achieves a slightly better 3.8% top-5 error rate. Furthermore, an ensemble of 3 three *Inception-ResNet-v2* and one Inception-v4 network can achieve an astonishing 3.08% top-5 error rate. All these progresses in image processing and recognition has taken to a reborn of ANNs for image processing. The new rise of such technologies involved many fields like Autonomous Driving one. Image processing tasks let a car learn to recognize other cars, pedestrians and animals and helps to increase the safety of the passengers. Moreover, using deep learning algorithm could also learn how to steer.

NVIDIA model

In 2016, NVIDIA contributed to Autonomous Driving field proposing an architecture for the *end-to-end learning* [2]. The purpose of this approach is that the car automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal. The car was never explicitly trained to detect the outline of roads.This architecture is a great milestone for using Deep Learning in Self-Driving car context.

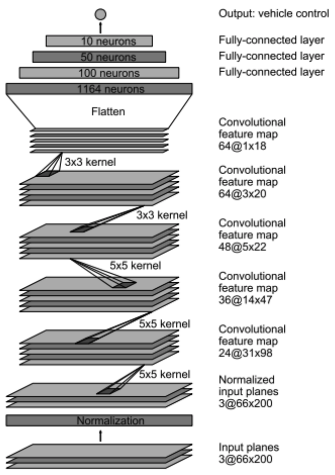


FIGURE 2.5: *CNN architecture. The network has about 27 million connections and 250 thousand parameters. Image taken from [2]*

The network used by NVIDIA is showed in Figure 2.5 consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into YUV planes and passed to the network. The first layer of the network performs *image normalization.* It is included in the network for allowing the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing. The convolutional layers has been designed to perform feature extraction. It has been used strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers. Next, three fully connected layers lead to an output control value which is the inverse turning radius. The fully connected layers are designed to function as a controller for steering. This model has shown a great precision and great results. The learning process associate is sometimes called *Behavioural cloning*. It is a method by which car actions can be captured and reproduced in a computer program. As the car subject performs the skill, its actions are recorded along with the situation that gave rise to the action. A log of these records is used as input to a learning program. The learning program outputs a set of rules that reproduce the skilled behaviour[115]. From there on, NVIDIA has released many tools and hardware useful for developers who approach Self Driving cars especially releasing more powerful graphic cards that enable to develop more powerful DL model.

Other models

Since NVIDIA released its paper, many researchers and companies started developing DNN architectures that resembles the End to End Model that NVIDIA has developed. The increased interest in such field has greatly incremented the number of solution proposed. For example, MOOC courses about Self-Driving Cars and Deep Learning has increased. One of them, Udacity, started a nano-degree course about it and many students has approached it and started also to provide challenges on such subject. One of them, regards the steering problem and the approach to how to steer a car. Many teams have tackled it using a CNNs and RNNs in order to process a sequence-to-sequence mapping from images to steering angles. Various other deep learning neural networks approaches have been proposed for self-driving cars. Some of these approaches use different type of CNN, RNN, or a combination of these architectures. Another approach is to use a combination of CNN and LSTM recurrent neural network like the one illustrated in Figure 2.6. CNN can be used to extract the features from the input image and these features are fed to the LSTM network which is used in the sequence task portion of the system.

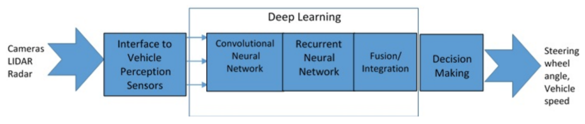
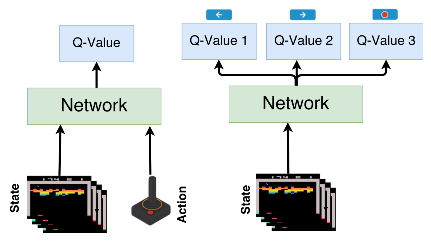

Deep Learning
Cameras
LIDAR
Radar
Interface to Vehicle Perception Sensors
Convolutional Neural Network
Recurrent Neural Network
Fusion/ Integration
Decision Making
Steering wheel angle, Vehicle speed

FIGURE 2.6: *A sketch representing how the Deep Learning is used in Self Driving Car context.*

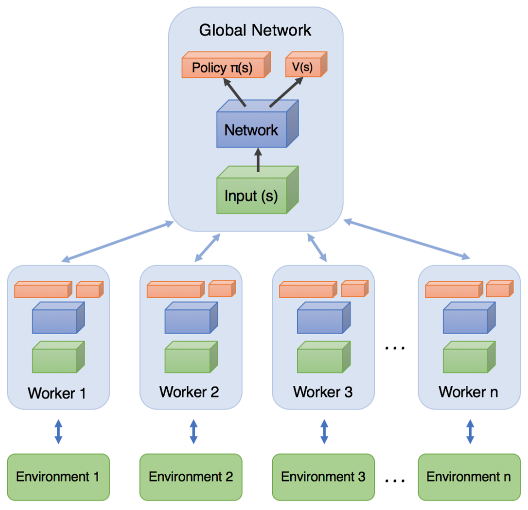
2.1.3 Deep Reinforcement Learning Model

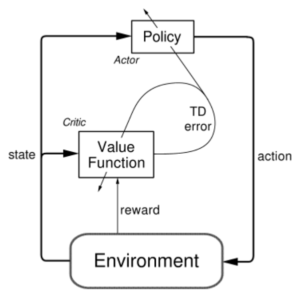
In 2005, Riedmiller introduced the idea of using neural network approximators for the Q function in Q-learning[116]. Mnih et al. introduced the idea of image-based Deep Q-Learning in 2015, when the group at DeepMind successfully used a convolutional neural network (DQN) to learn a Q function which successfully plays various Atari 2600 games at or significantly above professional human player ability [117]. The only inputs to their DQN algorithm were images of the game state and reward function values. Most impressively, this paper utilizes a single learning paradigm to successfully learn a wide variety of games - the generalizability of the approach (while obviously not a single set of learned parameters) is powerful, and is what inspired us to attempt to apply their model to learning a policy for JavaScript Racer. Since that work, DeepMind and others have published numerous extensions to the DQN paradigm. DQN learning approaches have been successfully leveraged for continuous control in addition to discrete control tasks. the simulated car driving task could be classified as one of them.

State-of-the-art autonomous vehicle control algorithms are largely orthogonal to DQN approaches, and since the very simplified game we ended up playing bears few actual similarities to real-world autonomous driving, the substantial body of literature that exists in that field was not especially relevant to our work here. DQN-based approaches to simple video games were much more in the vein of the work done in this paper.



(a) *DQN*





(b) *A3C* (c) *DDPG*

FIGURE 2.7: *Deep Reinforcement Learning architectures.*

Owing to those important steps in Deep RL, also autonomous driving context is going toward this direction. Examples of architectures that could be modelled using DQN-based models are illustrated in Figure 2.7a, Figure 2.7b and Figure 2.7c. *A3C* [6] and *DDPG* [118] are other algorithms implemented by DeepMind. In particular, this work will describe with more detail A3C in future chapters.

2.2 Summary

In this chapter, the main state-of-the-art algorithms in modern autonomous driving have been illustrated, along with the most relevant architectures. Such algorithms are the basis of the current work, which aims at analysing and comparing them so as to provide guidelines and suggestions to improve the current methodologies. In this work, researches will occur to find models or methods that improve the simulated car driving tasks.

Chapter 3

Technologies

In this chapter, the technologies considered in this work will be described. The aim of this chapter is to make the reader aware of the motivation behind what has been implemented.

This project has been developed for testing and trying state of the art methodologies and compare one another in order to find which is the best and if there are points where these algorithms could be improved. Moreover, advantages and drawbacks of each technique will be pointed out.

All the experiment has been done mainly in two different machine:

- 2,5 GHz Intel Core i5 with 3MB of cache and 8GB of RAM, running with Apple macOS Sierra 10.12.6. It has an Intel HD Graphics 4000 1536 MB of dedicated RAM.

- 3.4 Ghz Intel Core i7 with 20 GB of RAM, running Ubuntu Gnome 16.04 LTS,. It has an Nvidia Geforce GTX 1060 with 6GB of dedicated RAM.

3.1 Programming Languages

By now, there are plenty programming language that works well and have many libraries that let to implement Computer Visions tasks and Neural

Networks without problems. In literature, it has been found that the main adopted programming languages adopted for Computer Vision and Neural Networks are MATLAB, C++ and Python.

3.1.1 MATLAB

In the origin, CV was restricted mainly in research area. Researchers usually had used MATLAB because of its impressive power that offer to their customer and its community which has offered many algorithms ready to be used in researches. MATLAB (which stands for **mat**rix **lab**oratory) is a multi-paradigm numerical computing environment. It comes out with a proprietary programming language developed by MathWorks which allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python. It is now used in many fields like Mechanics, Automotive, Automation, Biomedical, etc. MATLAB offers many advantages which are:

- *Powerful matrix library*: in CV it is common to treat an image as a multi-dimensional matrix. Thanks to the power of MATLAB, it is possible to use a heavy dose of linear algebra in various algorithms. MATLAB's linear algebra routines are very powerful and fast if they are used correctly.

- *Toolboxes*: There is a toolbox for everything. MATLAB has an image processing toolbox, a computer vision toolbox, and a statistical and machine learning one that provide implementations of a wide variety of very useful algorithms. The functions usually provide a clean and obvious interface. Many computer vision problems are often set up as optimization problems. The optimization toolbox in MATLAB provides excellent implementations of many optimization algorithms.

- *Visualization and debugging tools*: One of the joys of using MATLAB is that writing code, visualizing results, and debugging happens in one integrated environment which helps the developer to be extremely productive.

- *Works with OpenCV* : You can interface with OpenCV using MATLAB's OpenCV Interface.

- *Great documentation*: MATLAB comes with great documentation and examples that are easily accessible within the IDE.

- *Large research community* : Latest research demos are often shared as MATLAB code.

Nevertheless al the advantages showed, MATLAB has also many drawbacks which are:

- *Cost* : MATLAB is hideously expensive. If you want to obtain a license and use it for CV purpose, you are going to pay maybe more than €10,000. Starting from basic MATLAB (€2,000) adding the computer vision toolbox (€1,250) and the required image processing toolbox (€1000) plus the optimization toolbox (€1,150) and machine learning toolboxes (€1000), it requires about €6400. If the customer wants also to deploy what has implemented, it has to buy the compiler (about €4,500)[119].

- *Learning curve* : MATLAB is a matrix engine. There is a MATLAB way to write code which is different from general purpose programming languages like C++ or Python.

- *Slower runtime* : A typical MATLAB program runs many times slower than a C++ program. Built-in MATLAB routines can be very fast, but the code in MATLAB will usually run much slower. It happens that developers end up coding computationally intensive parts in C and integrating it with MATLAB code using *mex*.

3.1.2 C++

The competitor of MATLAB in the CV fields was *OpenCV* which is a library of programming functions mainly aimed at real-time computer vision[120]. Originally developed by Intel, it was later supported by Willow Garage and then maintained by Itseez Inc. which, since 2016, it has been acquired by Intel. The library is cross-platform and free for use under the open-source BSD license. Nevertheless OpenCV is younger than MATLAB, now it is the most used library in CV field mainly because it supports many of the most famous Deep Learning frameworks and it can be used also in MATLAB. Moreover, OpenCV is written in C++. It is a language that follows an imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation. It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights[121]. C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications,[121] including desktop applications, servers (e.g. e-commerce, web search or SQL servers), and performance-critical applications (e.g. telephone switches or space probes)[122]. C++ is a compiled language, with implementations of it available on many platforms. OpenCV does not have Deep Learning Framework as MATLAB. So alternatives may be found in *eblearn*[123], *Tensorflow*[124] or *Caffe*[125]. The advantages to use C++ could be summarized in properties which are:

- *Free* : Large parts of OpenCV are free and *eblearn*, *Tensorflow* and *Caffe* are open source. It is possible to use them in commercial applications, and you can view the source and fix issues if needed. It is not needed to open source your project if you use them. OpenCV has some parts which are restricted by license: e.g. *surf* and *sift* functions and the cascade filter for recognition are under patent and so they are free for academic use but not for commercial purpose.

- *Optimized library* : The collection of algorithms available in C++ is very rich. OpenCV, for example, has a gigantic library which is also optimized for performance as it supports GPU computing library as OpenCL and CUDA.

- *Platforms and devices* : it is possible to use C++ applications on multiple platforms like desktop application or web application as a back-end. For example, OpenCV (C/C++) is the vision library of choice in many embedded vision applications and mobile apps.

- *Big community* : C++ developers are one of the most extended community in the field of programming language. For OpenCV, there is a big community of developers (about 50,000) that use and support OpenCV. Unlike the MATLAB community that consists of researchers, the C++ community is a mix of people from many fields and industries.

As for MATLAB, also C++ and its library have their drawbacks which are:

- *Difficult for beginners* : C++ is really difficult to learn for beginners due to lack of educational materials. Since no corporation or other permanent entity owns the language, there is no oversight over educational materials.Approaching to it, it requires a great amount of time and usually it makes you learn just a little part of it. Moreover, there are few library well documented. For example, OpenCV (C++) is not one of them and the learning is sometimes daunting. Sometimes you need to have a good understanding of the algorithm, and actually read the paper, because the documentation does not always explain what the parameters mean and how they effect the outcome. The documentation does not always come with sample code, and that makes it harder to understand. The sample code that comes with OpenCV, though very useful, is also not very well documented either.

- *Visualization and debugging*: Debugging and visualizing is hard in any

C++ environment. This is especially true if a new algorithm from scratch. need to be tested and debugged.

3.1.3 Python

The last programming language to be considered is Python. It is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. As interpreted language, Python has a design philosophy that emphasizes code readability (notably using white-space indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java[126]. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software[127] and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

OpenCV comes with a library that also supports Python. Moreover, Python has a set of powerful libraries which make it as one of the best scientific computing language.

Advantages to use Python can be summarized in:

- *Ease of use*: Python is very simple and use construct that are familiar to either Java and C/C++ developer. Simplicity in the constructs and general purpose utilization makes him one of the most preferred first language to learn. Moreover, its simplicity enables programmers to fast prototyping and testing.

- *the language of scientific computing*: A few years back MATLAB was called the language of scientific computing. With OpenCV, *NumPy*[128], *SciPy*[129], *scikit-learn*[130], and *Matplotlib*[131], Python provides a powerful environment for learning and experimenting with Computer Vision and Machine Learning. Moreover it has a wide range of Deep Learning Framework like *Tensorflow*[124], *Keras*[132], *Theano*[133], *CNTK*[134], *Caffe*[125] and *Pytorch* [135] which are the most used framework in this fields.

- *Visualization and debugging*: Visualization using *matplotlib* is about as good as MATLAB. Debugging code in Python is easier than C++, but it does not quite match the super-easiness of MATLAB.

- *Building web backend*: Python is also a popular language for building websites. Frameworks like *Django*, *Web2py*, and *Flask* allow you to quickly put together web apps. It is very easy to use OpenCV in Python and the other libraries along with these web frameworks.

Nevertheless these advantages, Python comes with some drawbacks which are:

- *Slower run time* : Compared to C++, Python programs will typically run slower. Moreover, using GPU libraries (*CUDA* or *OpenCL*) in C++ makes the code runs 10 times faster than the Python implementation.

- *Libraries written in C/C++*: One of the great benefits of an open source library is your ability to modify them to suit your needs. If libraries like *OpenCV* or *Tensorflow* needs a modification, it is needed to modify the C/C++ source.

3.1.4 Conclusions

Once all advantages and drawbacks have been described, it is not so easy to choose the programming language. In a company context, where the product needs to be published, from my point of view, it is recommended to test and try ideas on MATLAB (if the company has the license) or Python and to publish the code in C/C++ version. This modification will lead to better performance for engineers and developers who will prototype faster and for the program which will be faster. as far as concerned this work, the aim is to be a proof of concept for future development and research to be extended. Because of the research work it is going to be done and because of there is no starting point to begin, the choice is Python. One of the main characteristics of Python is that the process that lead to a delivery product is very fast and this is one of the features which suits best for the work it is going to be described. Currently Python has two main version: *2.x* and *3.x*. In short, Python *2.x* is legacy, Python *3.x* is the present and future of the language [136]. For this reason, it has been decided to implement everything done in this work in Python 3.5.

3.2 Deep Learning Frameworks

Once the programming language has been defined, another important step to be considered is the Deep Learning Framework. Given the absence of other valid Computer Vision framework, OpenCV has been chosen as the CV frameworks adopted for this project. The DL framework that will be described are the most used frameworks in Python and represent many of the research and development state-of-the-art algorithms as far as concerned DL field. All the benchmarks, advantages and drawbacks of the DL frameworks proposed are taken from [137, 138, 139].

3.2.1 Theano

Just for historically concerns, I will include *Theano*[133] among the possible choice. In fact, Yoshua Bengio announced on Sept. 28, 2017, that development on Theano would cease [140]. Many academic researchers in the field of deep learning rely on Theano, which is the ancestor of many deep-learning frameworks. It is written in Python. Theano is a library that handles multidimensional arrays, like NumPy. Used with other libraries, it is well suited to data exploration and intended for research.

Numerous open-source deep-libraries have been built on top of Theano, including *Keras*, *Lasagne* and *Blocks*. These libraries attempt to layer an easier to use API on top of Theano's occasionally non-intuitive interface.

Theano is well suited for numerical tasks often encountered when dealing with deep learning. It combines several paradigms for numerical computations, such as matrix operations, symbolic variable and function definitions, and just-in-time compilation to CPU or GPU machine code. Moreover, Theano can compile and optimize the code so that it can run on both CPU and GPU.

Theano is one of the oldest deep learning libraries out there and a lot of other widely used libraries have been built on top of it. But Theano heavily relies on the mathematical side of deep learning and data discovery, having similar features to NumPy or Matlab. This is why it's usually used with other libraries in order to achieve a higher level of abstraction. Moreover is very *fat* compared to other DL frameworks.

3.2.2 Caffe

Caffe [125, 141] is a well-known and widely used machine-vision library that ported Matlab's implementation of fast convolutional nets to C and C++. Caffe is not intended for other deep-learning applications such as text, sound or time series data. Like other frameworks mentioned here, Caffe has chosen Python for its API. It has been developed and maintained by Berkeley Vision and Learning Center.

It performs image classification with convolutional nets, which represent the state of the art. While it is widely cited in papers, Caffe is chiefly used as a source of pre-trained models hosted on its Model Zoo site.

It is a library which is very good for feed-forward networks and image processing and for tuning existing networks. Moreover, it is possible to train models without writing any line of code. Drawbacks that exist when Caffe is used are that, if a new GPU layer is need to be introduced, then it is necessary to write it using C++/CUDA. Many developers have complained that recurrent network are hard to be implemented and that it is a bit heavy in instantiating big networks like *GoogLeNet* or *ResNet*. Caffe does not offer any commercial support and it is probably going to be not maintained anymore due to its slow development and because of Caffe2.

3.2.3 Caffe2

Caffe2[142] is the successor to the original Caffe, whose creator Yangqing Jia now works at Facebook. Caffe2 is the second deep-learning framework to be backed by Facebook after Torch/PyTorch. The main difference seems to be the claim that Caffe2 is more scalable and light-weight. It purports to be deep learning for production environments. Like Caffe and PyTorch, Caffe2 offers a Python API running on a C++ engine. It is possible to use it for commercial purpose but it has no support by now. Moreover this new framework will support ONNX [143], that in the future will let developers to enable framework interoperability.

3.2.4 CNTK

The Microsoft Cognitive Toolkit [134], is a unified DL toolkit that describes neural networks as a series of computational steps via a directed graph. In this directed graph, leaf nodes represent input values or network parameters, while other nodes represent matrix operations upon their inputs. CNTK allows to easily realize and combine popular model types such as feed-forward DNNs, convolutional nets (CNNs), and recurrent networks (RNNs/LSTMs). It implements stochastic gradient descent (SGD, error backpropagation) learning with automatic differentiation and parallelization across multiple GPUs and servers. CNTK has been available under an open-source license since April 2015. It is important to note that there is also a Python framework for CNTK as well. It allows for easy support across mobile operating systems. These advantages make it better for development in industry in many aspects, although the versatility is not as great nor does it allow for as much customization as other frameworks does. The key features which distinguish CNTK library are the flexibility, its goodness for RNN and the distributed training. It does not offer any visualization tool. Moreover its source code is not easily readable. As a distinguishable feature, it has been found that CNTK provides a definition language called NDL (Network Definition Language) [144] which provides a simple way to define a network in a code-like fashion. It contains variables, macros and other concepts.

3.2.5 PyTorch

A Python version of Torch, known as Pytorch, was open-sourced by Facebook in January 2017. PyTorch offers dynamic computation graphs, which let you process variable-length inputs and outputs, which is useful when working with RNNs, for example. In September 2017, Jeremy Howard's and Rachael Thomas's well-known deep-learning course fast.ai adopted Pytorch. Since it's introduction, PyTorch has quickly become the favorite among machine-learning researchers, because it allows certain complex architectures to be built easily. Other frameworks that support dynamic computation graphs are CMU's DyNet and PFN's Chainer. Torch is a computational framework with an API written in Lua that supports machine-learning algorithms. Some version of it is used by large technologies companies such as Facebook and Twitter, which devote in-house teams to customizing their deep learning platforms. Lua is a multi-paradigm scripting language that was developed in Brazil in the early 1990s. Torch, while powerful, was not designed to be widely accessible to the Python-based academic community. PyTorch is currently in beta and it is only supported in Linux and Mac OSX. PyTorch has a team composed by known names in the research community going for it thanks to Facebook AI team who is helping developing it. PyTorch has a great ease of use and modification more than Tensorflow. PyTorch may be used as a NumPy replacement for things not neural network related. Again, for this there is Tensorflow though PyTorch is way more numpy like. PyTorch does not have a visualization tool. It is possible to interactively debug PyTorch.

3.2.6 Tensorflow

In 2015, Google released to the community TensorFlow[124, 145] to replace Theano. The two libraries are quite similar. Some of the creators of Theano, such as Ian Goodfellow, went on to create Tensorflow at Google before leaving for OpenAI.

Like most deep-learning frameworks, TensorFlow is written with a Python API over a C/C++ engine that makes it run faster. TensorFlow runs slower than other frameworks such as CNTK. TensorFlow does not support so-called *inline* matrix operations, but forces you to copy a matrix in order to perform an operation on it. Copying very large matrices is costly in every sense. TensorFlow is about more than deep learning. TensorFlow actually has tools to support reinforcement learning and other algos. Google's acknowledged goal with Tensorflow seems to be recruiting, making their researchers' code shareable, standardizing how software engineers approach deep learning, and creating an additional draw to Google Cloud services, on which TensorFlow is optimized. TensorFlow is not commercially supported by Google because it is a tool for developers and researchers. Like Theano, TensorFlow generates a computational graph (e.g. a series of matrix operations such as $z = sigmoid(x)$ where x and z are matrices) and performs automatic differentiation. Automatic differentiation is important because it is not good to have to hand-code a new variation of back-propagation every time it is experimented a new arrangement of neural networks. In Google's ecosystem, the computational graph is then used by Google Brain for the heavy lifting, but Google has not open-sourced those tools yet. TensorFlow is one half of Google's in-house DL solution. Google introduced Eager, a dynamic computation graph module for TensorFlow, in October 2017. From an enterprise perspective, the question some companies will need to answer is whether they want to depend upon Google for these tools, given how Google developed services on top of Android, and the general lack of enterprise support. As far as concerning visualization and debugging, Tensorflow has Tensorboard which let developers to visualize TensorFlow graph, plot quantitative metrics about the execution of your graph, and show additional data like images that pass through it. It is a bit low-level programming and usually it is used as a Keras backend. Tensorflow has a great community behind and it is one of the most followed repository from the GitHub community[146].

3.2.7 Keras

Keras[132] is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK[147], or Theano. Support for CNTK has been provided by Microsoft in October 2017. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. It is currently maintained by

Keras has been developed following three main guiding principles, which are [132]:

- *User friendliness*: Keras is an API designed for facilitate developers code with more difficult DL frameworks such as those supported. It puts user experience front and center. Keras follows best practices for reducing cognitive load because it offers consistent and simple APIs, it minimizes the number of user actions required for common use cases and it provides clear and actionable feedback upon user error.

- *Modularity*: A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.

- *Easy extensibility*: new modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

- *Work with Python*: No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

Keras is very useful because it supports both convolutional networks and recurrent networks, as well as combinations of the two and it is able to runs seamlessly on CPU and GPU.

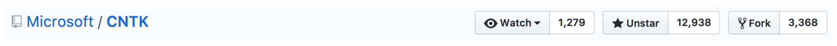
3.2.8 Evaluation and Comparisons

Once all frameworks has been listed and described, a comparison is needed among each of them in order to individuate the best solution for the project. In order to accomplish such task, key feature of DL framework will be considered and ranked.

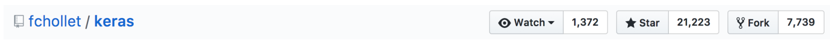
One of the key point which has been fixed for this work is easiness of development and fast prototyping. Performance in this phase is not considered. Moreover, an important key point is the documentation and support by the community. Youngest Deep Learning frameworks are not advantaged by this criteria. For this reason Caffe2 and PyTorch will not be for now a valid choice. In future, when ONNX technology will be refined by Microsoft and Facebook research teams, I believe that this two will lead to great results. For the same reason, framework like Theano and Caffe which are the oldest and less maintained will not be part of this choice. Considering all of this points, it is possible to reduce the choice at Tensorflow, Keras and CNTK.

The first term of comparison is the *community.* As it is possible to see in Figure 3.1, GitHub repository followers gives an idea of how much Tensorflow is followed by research, amateur and professional developer. CNTK is younger than Tensorflow but has a great community and is going to have an even increasing number of followers. Keras has a nice group of followers that improve it and it is going to increase in the next future.

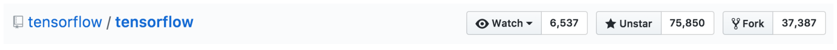
TensorFlow and CNTK are very similar for the simple convolutional neural network example. However, TensorFlow version is easier to experiment with


Microsoft / CNTK

(a)


fchollet / keras

(b)


tensorflow / tensorflow

(c)

Figure 3.1: *(a) CNTK. (b) Keras (c) Tensorflow. Image taken from* `github.com` *on 5th Nov 2017.*

because it is driven by Python. With CNTK, developers need to completely understand how to express things with the configuration file which can be difficult at the beginning. Tensorflow is more than just a Deep Learning library as it provide a valid framework for many ML algorithms. CNTK instead is quite different and it is impossible to reproduce some of the algorithms provided by Tensorflow . In the case of the LSTM recurrent neural network, CNTK version is completely transparent. In the case of Tensorflow, top level idea is very elegant, but very difficult to understand all the details because of the clever use of the variable scoping and variable sharing. But it is easy to experiment with Tensorflow. Documentations on both libraries which is represented by CNTK book[148] and the TensorFlow tutorials[149] are both excellent introductions to the high level concepts. While Tensorflow and CNTK are low-level libraries which are easy to compare one another, Keras represent another type of library that could be considered as the highest level, most user friendly library. It allows users to choose whether the models they build are executed on the chosen backend. Thanks in part to excellent documentation and its relative ease of use, the Keras community is quite large and very active. In January 2017, TensorFlow team announced plans to ship with Keras support built in, so soon Keras will be a subset of the TensorFlow project [150]. Due to the aim of the project and the vision to experiment, the choice is fallen on Keras, for its simplicity and ease of use. Keras, however, needs a choice also for the backend. Keras has three types of backend which are Theano, CNTK and Tensorflow. At the time this thesis has been written, CNTK backend is not fully supported on Keras and the code is maybe not well optimized. In [151, 152] there are described the benchmark which it has been chosen as valid judgement criteria. In [152], results have showed that the accuracies of TensorFlow and CNTK backends are similar across all benchmark tests, while speeds vary a lot. CNTK is a lot (about 2 to 4 times) faster than TensorFlow for LSTM (Bidirectional LSTM on IMDb Data and Text Generation via LSTM), while speeds for other type of neural networks are close to each other.

For reasons that depend on the community dimension, grow possibility, future maintenance of the code, good performance on server-side, good documentation and easiness of visualization through Tensorboard, it has been chosen to use Tensorflow as backend for Keras.

3.3 CPU versus GPU

Traditionally, computing power is associated with the number of CPUs and the cores per processing unit. During the 90s, when WinTel started to invade the enterprise data center, application performance and database throughput were directly proportional to the number of CPUs and available RAM. While these factors are critical to achieving the desired performance of enterprise applications, a new processor started to gain attention – Graphics Processing Unit or GPU.

GPUs remind the video cards that were designed for graphic-intensive games. These were purely optional, which did not influence the buying decision of an average user investing in a PC or server. Nowadays GPUs is a relevant factor for every developer who wants to experiment in faster condition his algorithms. One of the field in which GPU are gaining more and more relevance is Machine Learning and Deep Learning.

All the deep learning algorithms perform complex statistical computations. A simple image translates to few million pixels, which in turn translates to a large matrix of numbers. During the training phase of deep learning, these matrices of numbers are fed as input into the neural network along with the correct classification. For example, by training the neural network with 1000s of cat images, we are going to get a model that can easily recognize a cat visible in a photo. This training process is all about correlating multiple pixels (numbers) to find patterns of a cat image. The correlation involves multiplying millions of matrices with each other to arrive at the right result. To increase the training speed, these operations need to be done in parallel.

Typical CPUs are designed to tackle computations in a sequential order, which means each mathematical operation will have to wait for the previous to complete. A CPU with multiple cores may marginally speed up the calculation by offloading the operations to each core. But, as we know CPUs with multiple cores are prohibitively expensive, making them less optimal for training neural networks.

Instead, GPUs have a processor with thousands of cores capable of performing millions of mathematical operations in parallel. There is a similarity between graphic rendering and deep learning. Both these scenarios deal with a huge number of matrix multiplication operations per second. That's one of the reasons why laptops or desktops with high-end GPUs are preferred for deep learning. Nvidia has a programming model for GPU called CUDA that lets developers build parallel programs. CUDA is available for C/C++ application, Python developers and also in MATLAB[153].

Just as a proof of concept, Nvidia's latest GPUs come with 3,584 cores while Intel's top end server CPUs may have a maximum of 28 cores(see Figure 3.2).

The rise of GPU does not result in the death of CPU. The combination of CPU and GPU along with sufficient RAM makes it a perfect sandbox for deep learning.

In this work, CUDA will be used through Tensorflow Backend, in order to gain an improvement as far as concerned deep neural network training. This is the reason why the work it has been developed in two different PCs. The

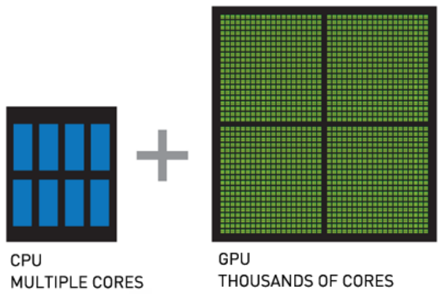


Figure 3.2: *Differences between CPU and GPU*

Desktop PC has inside a Nvidia Geforce GTX 1060 which is supported by CUDA. Another reason is that CPUs training last very long even if is the fastest CPU around. As a confirm. in [154], there is a comparison that will help the reader understand the differences between CPU and GPU in a CNN use case.

3.4 Simulators

One of the difficult part of creating a prototype of a controller for autonomous driving is the one which regards simulation. Simulators are difficult to be implemented because they might lead to oversimplifications. Simulating the reality is impossible for every kind of simulator and so a simulated environment may lead to error when the model is brought in a real scenario. If the simulator does not account for the most relevant any possible noise or error that may occur, then simulated model brought in a real scenario is likely to fail.

Simulators need to respect some specific principles. It needs to have:

- *Server interface*: it is necessary since there are no simulators written in Python and so the communication has to flow from one direction to another and viceversa.
- *Environments that resemble roads*: it is needed in order to enable vision

tasks.

- *Vision sensors*: they are the most important part. The simulator needs to have access to vision sensors in order to grab image to pass to the client.

- *Car dynamics and graphics*: physics engine is not important in this work by now. This is an experimentation of CV and NN. Perhaps, if the game engine on which the simulator is based provides a good physics, the simulation will result improved and more realistic. Another fundamental part is the graphics of the car itself which needs to be realistic. This feature is justified by the possibility of new task that may need the presence of other car in the environment. If the game engine provides car graphics, it will enable to use it in order to test the behaviour of the controller for object recognition tasks.

In this section, all the simulators found that are described respect these features. Data and simulator are strictly related. Data are the relevant part of this project and processing of such data is a fundamental part of every machine learning and deep learning algorithm because they are the mean by which neural network will be trained. To accomplish such task, it has been investigated among different researches to find all the available simulators on which it is possible to experiment implemented models and algorithms.

DeepTesla

One of the choice presented to the reader is DeepTesla[155]. This is not a real simulator for robotics task. It is a visual representation that is used to give a simple demonstration of using convolutional neural networks in end-to-end steering. It has been developed by MIT researchers and consists of a web page which contains many boxes that are used for the configuration of the network. It is possible to find it at `https://selfdrivingcars.mit.edu/deepteslajs/`. It is written in JS and the network is written with *ConvNetJS*. The user can configure the model using a JSON string in a text field and test check the training of the network in real-time. On *GitHub*, a similar project has been released by Lex Fridman (professor who holds *Deep Learning for Self-Driving Cars* course at MIT) in Python which it could be found at `https://github.com/lexfridman/deeptesla`. In this version, video are processed in order to provide a valid dataset. Every video is linked to a CSV file which contains the steering wheel data. It is possible to add a personal model in *model.py* file. After the network has been trained, it is possible to run the model and visualize the result. An image of the video produced is shown in Figure 3.3.

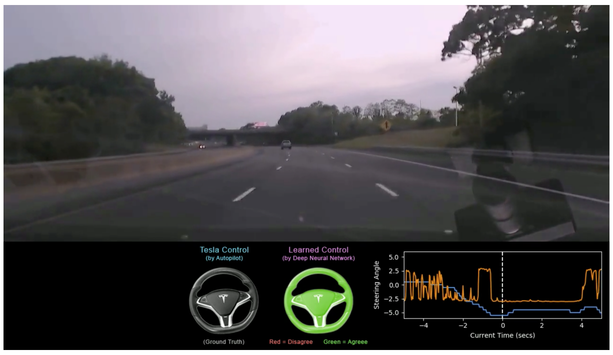


FIGURE 3.3: *Udacity Simulator screen-shot*

Udacity Self-Driving Car Simulator

A *massive open online course* (MOOC) is an online course aimed at unlimited participation and open access via the web. In addition to traditional course materials such as filmed lectures, readings, and problem sets, many MOOCs provide interactive user forums to support community interactions among students, professors, and teaching assistants. *Udacity*[156] is one of them and it is gaining more and more popularity for introducing nanodegree as a set of many courses in a specific field. For the *Self-Driving car Nanodegree* courses[157], developers at Udacity, helped by a wide community, have developed and released a Unity-based simulator for self-driving car. It is possible to find it at `https://github.com/udacity/self-driving-car-sim/`. This simulator is composed by two main track which could be used in two modality. There is a modality which is used for gaining data and it is called *Manual*. It could record a performance of the car in a run and at the end it saves all the images from the three cameras posed in front of the car and all the data such as speed, steering angle and brake. The second modality is *Autonomous* where the simulator enables the server written in C# and let the client receive data from the car and control the car itself. In Figure 3.4, there is a screen-shot taken from a test.

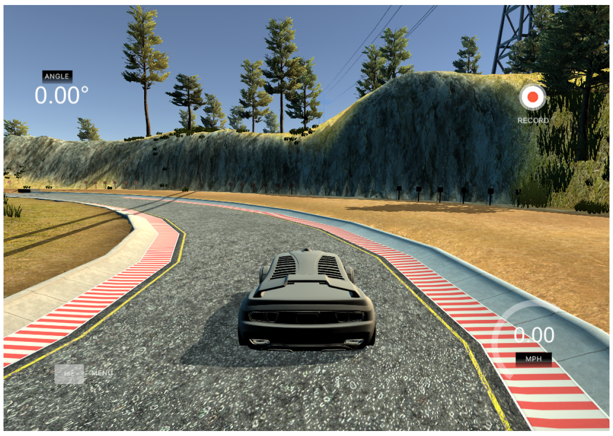


FIGURE 3.4: *Udacity Simulator screen-shot*

This simulator is a great tool for testing the Neural Network architecture model developed. Unity provide a game engine complete from a physics point of view. It is open source and it is possible to clone it using Git and modify as the user needs.

TORCS (The Open Racing Car Simulator)

The Open Racing Car Simulator or *TORCS* is a highly portable multi-platform car racing simulation. It is used as ordinary car racing game, as AI racing game, and as a research platform. The source code of TORCS is licensed under the GPL ("Open Source") [158]. It provides a full 3D visualization, a sophisticated physics engine, and accurate car dynamics taking into account traction, aerodynamics, fuel consumption, etc. Some pictures from TORCS can be found in Figure 3.5.

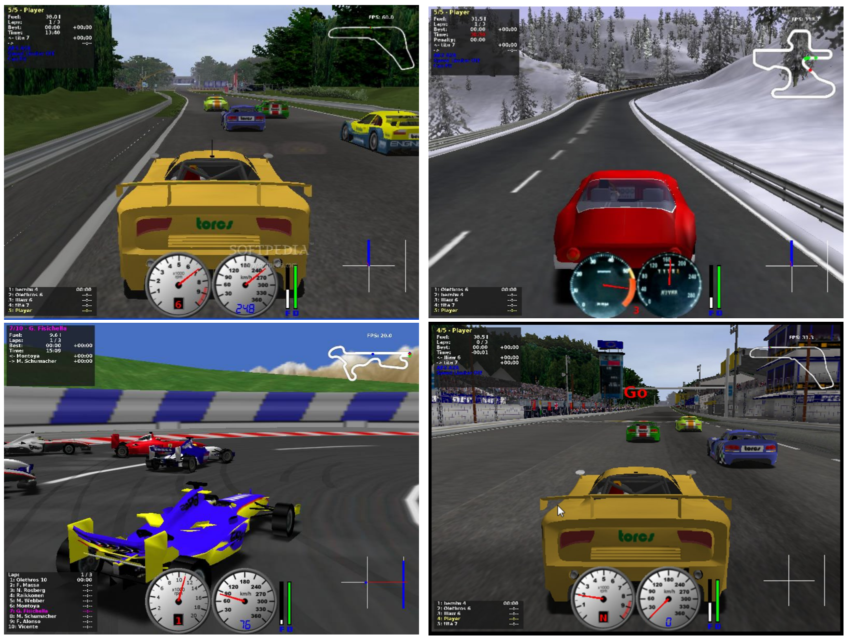


FIGURE 3.5: *Screenshots from the TORCS environment*

In April 2013, Politecnico di Milano organized a competition software for the Simulated Car Racing Championship which is an international competition held at major conferences in the field of Evolutionary Computation and in the field of Computational Intelligence and Games[159]. In this competition they released a TORCS patch, called *scr-patch* that helps to gain information offered by the sensors and environment of TORCS. For enable TORCS to be used along with Python3, the community has been provided scripts called *snakeoil3-gym.py* and *gym-torcs.py* that helps users to interface to TORCS environment. It provides many information to the developer, such as throttle, brake, steering angle, current speed, images, and many other useful information and let the controller drive the car. It is a very realistic driving simulator and it provides interesting features. A drawback consists of images provided by the server which are fixed to a dimension 64 $\times$ 64 pixels because of the usage of an old library which has many issues with buffering while taking image with higher dimension.

3.4.1 CARLA: An Open Urban Driving Simulator

In March 2017, a group of researcher from Intel, Toyota and CVC of Barcelona has started developing *CARLA* (*Car Learning to Act*) [160]. It is an open-source simulator for autonomous driving research that has been developed from the ground up to support development, training, and validation of autonomous urban driving systems. In addition to open-source code and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely. The simulation platform supports flexible specification of sensor suites and environmental conditions.

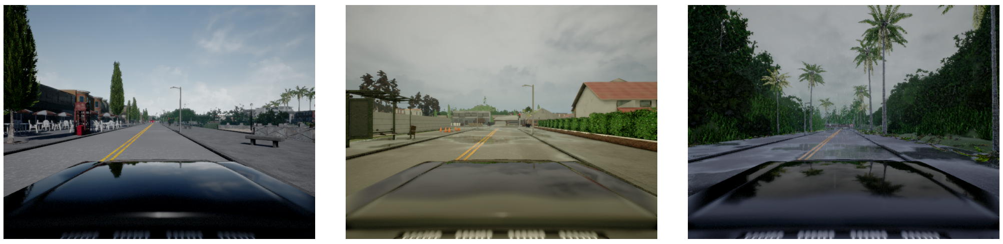


FIGURE 3.6: *Three of the fourteen different weather condition offered by CARLA Simulator.*

It is implemented as an open-source layer over Unreal Engine 4 (UE4), enabling future extensions by the community. The engine provides state-of-the-art rendering quality, realistic physics and an ecosystem of interoperable plugins[160]. CARLA simulates a dynamic world and provides a simple interface between the world and an agent that interacts with the world. To support this functionality, CARLA is designed as a server-client system, where the server runs the simulation and renders the scene. The client API is implemented in Python. The client sends commands and meta-commands to the server and receives sensor readings in return. Commands control the vehicle and include steering, accelerating, and braking. Meta-commands control the behaviour of the server and are used for resetting the simulation, changing the properties of the environment, and modifying the sensor suite. Environmental properties include weather conditions, illumination, and density of cars and pedestrians. When the server is reset, the agent is re-initialized at a new location specified by the client.

The car provides only vision sensors, position sensors, gyroscope and accelerometer sensor. One of the great feature in CARLA is the vision sensor which provide three sensing modalities such as normal vision camera, ground-truth depth, and ground-truth semantic segmentation.

Officially CARLA was published and released to the community on 15th November 2017. It is very young as simulator, and main development regards simulator graphics and server API. In my personal opinion, it has a great potential to be one of the best simulator for autonomous driving application. The first reason is that it is possible to change weather condition at runtime. Second, it provides complete urban environments with vehicles, pedestrians and traffic light. Third, it is designed for application like Reinforcement Learning and Deep Learning. It is easy to configure because it needs just a file for configure the server and the simulator graphics and can be also be configured by API. The client receives from the server the following information about the world and the player's state:

- Player Position: The 3D position of the player with respect to the world coordinate system.
- Player Speed: The player's linear speed in kilometers per hour.
- Collision: Cumulative impact from collisions with three different types of objects: cars, pedestrians, or static objects.
- Opposite Lane Intersection: The current fraction of the player car footprint that overlaps the opposite lane.
- Sidewalk Intersection: The current fraction of the player car footprint that overlaps the sidewalk.
- Time: The current in-game time.

- Player Acceleration: A 3D vector with the agent's acceleration with respect to the world coordinate system.
- Player Orientation: A unit-length vector corresponding to the agent car orientation.
- Sensor readings: The current readings from the set of camera sensors.
- Non-Client-Controlled agents information: The positions, orientations and bounding boxes for all pedestrians and cars present in the environment.
- Traffic Lights information: The position and state of all traffic lights.
- Speed Limit Signs information: Position and readings from all speed limit signs

All this information could enable a Neural Network to be fed by data which are useful in order to train it to accomplish steering tasks.

3.5 Conclusion

This chapter shows the wide availability of different valid framework, language and possible implementations that are available. The work in this thesis will be implemented using Python3. The IDE chosen for Python is *JetBrains PyCharm* (`https://www.jetbrains.com/pycharm/`) which is free for academic purpose. Deep Learning Framework chosen is Keras that will need Tensorflow as backend. This backend will let to run-time visualization using Tensorboard. It will be installed a version that uses GPU for training purpose in order to be faster when the DNN is trained. The technologies chosen will be supported by many Python libraries such as *numpy*, *matplotlib*, *pandas*, etc. The simulators chosen are DeepTesla and CARLA. Even if DeepTesla is not a simulator, it is a great tool to compare and visualize the property and behaviour of deep neural networks model while performing end-to-end steering task. The model which will score the best results will be then compared inside CARLA to understand the goodness of the algorithm in a simulated environment. Udacity Simulator has been excluded because of the absence of information from the environment which has not been developed by now. The reason behind the exclusion of TORCS is that the image provided by TORCS are not good enough to be processed, and the feature extraction from the line in DL algorithm is really hard. Even if CARLA is the younger simulator, it provides a fully set of tool and methodology useful for the work in this thesis.

Chapter 4

Implementation

In Chapter 2, an overview of the state-of-the-art algorithms for autonomous drivng has been provided. These algorithms have been implemented and used in real prototypes. There are algorithms that currently are not used in real commercial prototypes and are discovered only in research context. Checking whether these solutions are better or not compared to the others is one of the key objectives of this thesis and they will be in this chapter.

4.1 Introduction to the problem

The work done consists in implementing and testing some of the state-of-the-art AI and ML algorithm in the context of autonomous driving. It aims to discover which of the model is well suited for autonomous driving tasks and which of the machine learning methodologies improve how the autonomous car percepts the world. The experiment done will be divided in two main part which are:

- End-to-End learning for steering applied on all the algorithms using DeepTesla
- Driving in a CARLA simulated environment to compare machine learning technologies when they are required to use more inputs and enable

sensor fusion. *CNN LSTM* will be compared to *A3C* algorithm in order to obtain a result of reliability and performance of the algorithm in the CARLA simulator. *CNN LSTM* algorithms will be improved in order to give also throttle and brake information.

A *GitHub* repository[161] will be provided. It contains all the data and graph collected during experiment and summaries created using the *Tensorboard* tool.

4.2 Data

Data are the fundamental mean for Deep Learning algorithms, especially for convolutional based one. In autonomous driving context, dataset are growing. The most used and famous are the *DeepDriving* datasets for *TORCS* and the *Udacity* one.

DeepTesla has a set of ten videos that can be used by a developer just splitting the videos frame by frame. It provides a *CSV* file where the steering angle are written and associated to every frame of the videos. Data are characterized by image which are not very clean because of poor light conditions which makes it difficult to recognize line markings. Moreover, traffic conditions are intense and other cars introduce a large quantitative of noise in the images due to the shadow that they create on markings in front of the car itself. The data distribution is illustrated in Figure 4.1.

In Deep Learning context, *data augmentation* is a common procedure too fix a too small dataset that consist of modifying original data with random modification in brightness, contrast or adding some random hole inside the image in order to introduce noise that help the network to generalize and find features. This operation is needed in order to avoid overfitting. For training purpose, four different transformations to augmented data are provided:

- *image brightness modification*: adjust the brightness of the image with a random value. Example is illustrated in Figure 4.2a

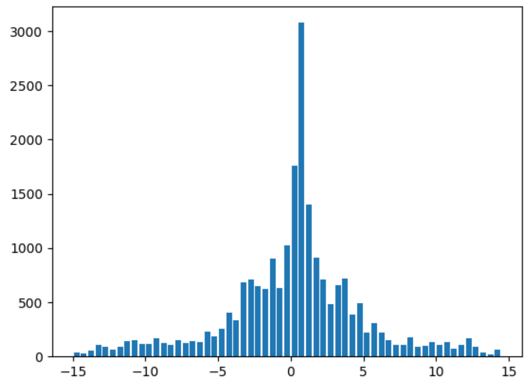
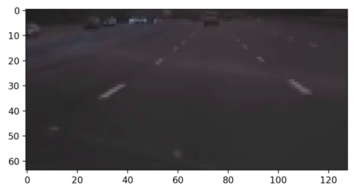


FIGURE 4.1: *DeepTesla training data distribution*

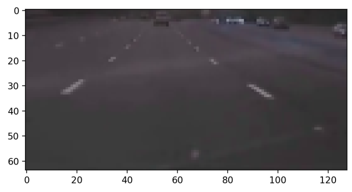
- *horizontal flip*: rotate the image in order to obtain the mirrored image. The steering angle will be then negated in order to follow the image. An example is illustrated in Figure 4.2b
- *random shadow*: applies a random shadow inside the image like showed in Figure 4.2c
- *image blurring*: the image will be blurred to simulate possible front windscreen fogging near the camera. An example is illustrated in Figure 4.2d.

CARLA does not provide data by itself. As the youngest framework to be released, it provides only just some examples. In order to retrieve data which are currently missing, an implementation effort is required for obtaining sufficient data to feed Deep Neural Network during the training. 4000 images and measurements will be provided for each of the fourteen weather conditions. Data distribution provided by CARLA dataset is illustrated in Figure 4.3.

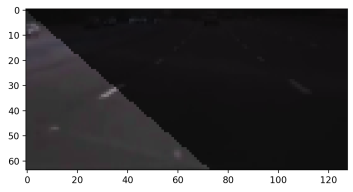
The data set has been split to 70% training data and 10% validation data and 20% for the tests for *DeepTesla* experiment. *CARLA* data set has been split to 70% training data 30% validation data. For CARLA, it has not been



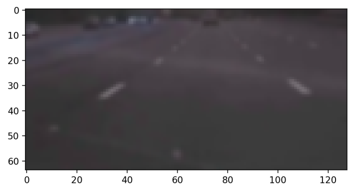
(a) *Brightness*



(b) *Horizontal flip*



(c) *Random shadow*



(d) *Blurring*

considered no test dataset because of the existence of a second Town in which test how the algorithm works.

For the two training experiments, a *BatchGenerator* will be provided, which is a methodology that helps to load huge datasets and pass them to the Neural Network. Large datasets are part of this work and there are two datasets that need a *BatchGenerator*.

4.3 Traditional Image Processing algorithms

Traditional Image Processing procedure is inspired by the methodologies followed in [15]. The slope of the curves and the steering wheel angle will be inferred by the images.

The best library that provide a wide range of traditional CV algorithms is *OpenCV* which will be used for building such algorithm.

In order to accomplish this task, this algorithm will have to:

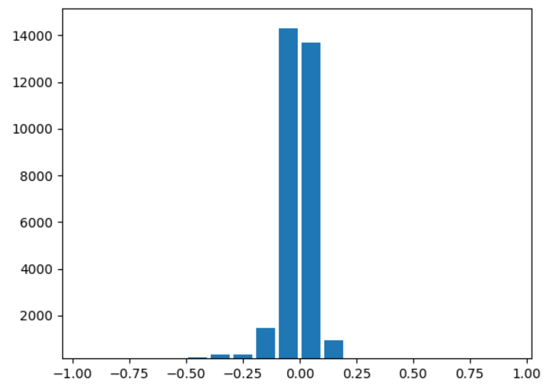


Figure 4.3: *CARLA data distribution*

1. Calibrate the camera and apply a distortion correction to raw images.
2. Transform the perspective of the image
3. Create a threshold binary image in order to find the lines
4. Apply a perspective transform to rectify binary image ("birds-eye view").
5. Detect lane pixels and fit to find the lane boundary.
6. Determine the curvature of the lane and vehicle position with respect to center.
7. Use this data to find the steering angle to apply to the wheel

Camera Calibration

The OpenCV functions $findChessboardCorners$ and $calibrateCamera$ are the backbone of the image calibration. A number of images of a chessboard, taken from different angles with the same camera, comprise the input. Arrays of object points, corresponding to the location (essentially indices) of internal corners of a chessboard, and image points, the pixel locations of the internal chessboard corners determined by $findChessboardCorners$, are fed to *calibrateCamera* which returns camera calibration and distortion coefficients. These can then be used by the OpenCV *undistort* function to undo the effects of distortion on any image produced by the same camera[162]. Generally, these coefficients will not change for a given camera (and lens). Figure 4.4 depicts the corners drawn onto twenty chessboard images using the OpenCV function *drawChessboardCorners*.

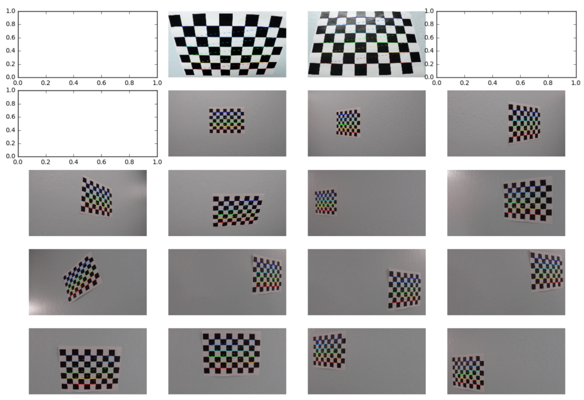


FIGURE 4.4: *Chessboard calibration. Some of the chessboard images don't appear because findChessboardCorners was unable to detect the desired number of internal corners.*

Figure 4.4 depicts the results of applying *undistort*, using the calibration and distortion coefficients, to one of the chessboard images.

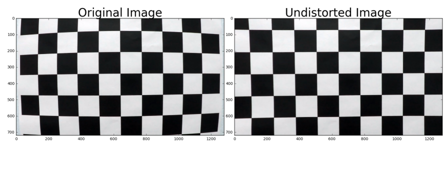


FIGURE 4.5: *Chessboard calibration. Some of the chessboard images don't appear because findChessboardCorners was unable to detect the desired number of internal corners.*

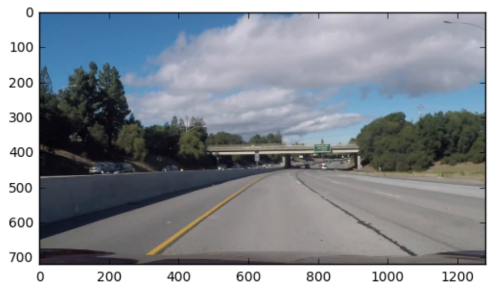
Threshold binary image

The next step is to create a threshold binary image, taking the undistorted image as input. The goal is to identify pixels that are likely to be part of the lane lines. Usually image are rich of noise that makes the lane recognition very hard. In order to have a better vision of the image, it is needed a filtering operation.

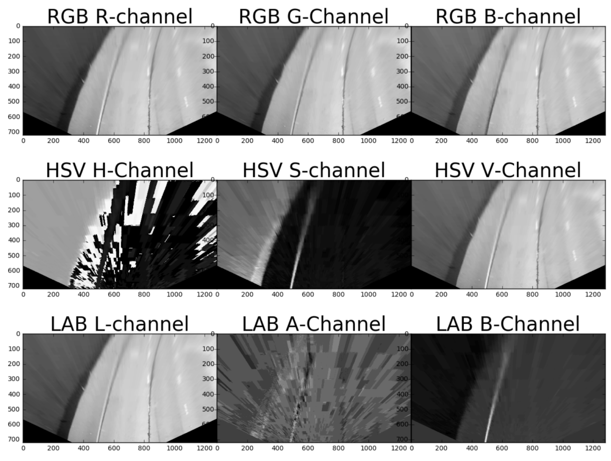
In order to accomplish this operation, all the colour which are not similar to the lane marking are filtered out . For example, Figure 4.6a represents a road in Nevada where the markings are yellow for external marking of the lane and white for internal road markings such discontinued line. In Italy, road markings are all white except for road that are in maintenance which are painted in yellow. Because of this reason, it is useful to combine many filters in order to highlight the line markings. In this work, a combination of a white and a yellow filter has been used and it is applied using the *Sobel operator* which helps to the edge detection. Gradient threshold are applied along the X axis with a directional gradients of 30 and 90 degrees. The reason is that the lines are more or less vertical. After this transformation of the directional gradients, a colour thresholds is applied. In Figure 4.6b, there are represented many of the filters that can be used for this task. In this work, colour thresholds chosen are *R & G channel thresholds* in order to recognize well the yellow lanes, *L channel threshold* to in order to take into account no edges generated by shadows and *S channel threshold* which is useful to separate out white & yellow lanes.

Bird's Eye View

In vision context, Bird's Eye View represents a type of shot that comes from the above and let to see an aerial perspective of the ground. The name comes from the similarity to the view of the ground that birds have when they fly. Considering the case illustrated in 4.6a where a frontal perspective that the car is provided, a perspective transformation is applied. Usually, a frontal image has a perspective of the road lines similar to a trapezoid. This points need to



(a)



(b)

FIGURE 4.6: *4.6a Original image. 4.6b Images obtained by the transformation using the respective filter.*

contains all the road lines in order to Passing the points of the trapezoid which contains both lines to a *OpenCV* function like *getPerspectiveTransform* which it is used for calculating the transformation matrix to be applied to the *warpPerspective* function which will help to have the Bird's Eye View. Figure 4.7 helps to understand what has been done with the Bird's Eye View Transformation applied on the thresholded binary image.

Fit pixels line

The next task is to identify lane lines and fit a second order polynomial to both right and left lane lines. The first of these, which could be named

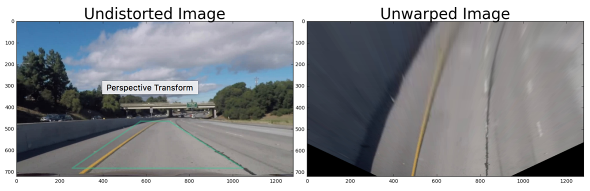
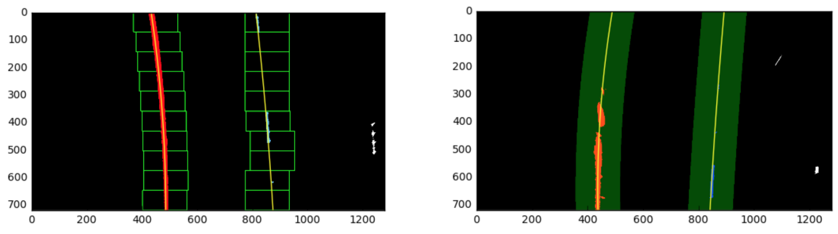


FIGURE 4.7: *Bird's eye view.*

as *Sliding Window Polyfit* computes a histogram of the bottom half of the image and finds the bottom-most x position (or "base") of the left and right lane lines. Originally these locations were identified from the local maxima of the left and right halves of the histogram, but in order to avoid to detect adjacent lines, it has been changed these to quarters of the histogram just left and right of the midpoint. The function then identifies ten windows from which to identify lane pixels, each one centered on the midpoint of the pixels from the window below. This effectively follows the lane lines up to the top of the binary image, and speeds processing by only searching for activated pixels over a small portion of the image. Pixels belonging to each lane line are identified and the *Numpy polyfit* method fits a second order polynomial to each set of pixels. In Figure 4.8a, there is the representation of what the function does. Then a function which apply $polyfit$ using fit from previous frame is called in order to alleviates much difficulty of the search process by leveraging a previous fit and only searching for lane pixels within a certain range of that fit. In Figure 4.8b, there is demonstration where the green shaded area is the range from the previous fit, and the yellow lines and red and blue pixels are from the current image.

Calculate radius of curvature and distance from the center

The radius of curvature is the radius of the circular arc which best approximates the curve at that point. In order to easily understand, it is the radius of the circle which osculate the curve. The radius changes as it



(a) (b)

FIGURE 4.8: *4.8a First function representation. 4.8b Second function representation.*

moves along the curve. In this work, radius of curvature is calculated as:

$$radius = \frac{(1 + (2 * a * y_0 * y_{mpix} + b) * 2) * 1.5)}{|2 * a|} \tag{4.1}$$

where a is the first coefficient (the y-squared coefficient) of the second order polynomial fit, b is the second (y) coefficient, y_0 is the y position within the image upon which the curvature calculation is based (the bottom-most y - the position of the car in the image - was chosen) and y_{mpix} is the factor used for converting from pixels to meters. This conversion was also used to generate a new fit with coefficients in terms of meters.

4.4 Optimizers

In Machine Learning, *gradient descent* is one of the most popular algorithms to perform optimization and the most common way to optimize neural networks. These algorithms are often used as *black-box optimizers* because of the existence of tool like *Keras*, which provides implementations of various algorithms to optimize *gradient descent*. It is a way to minimize an objective function $J(\theta)$ parametrized by a model parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ with regards to the parameters. The learning rate η determines the size of the steps taken to reach a (local) minimum. In simpler words, it follows the direction of the slope of the surface created by the objective function downhill until a valley is reached.

There are three variants of gradient descent, which differ in how much data we use to compute the gradient of the objective function. Depending on the amount of data, we make a trade-off between the accuracy of the parameter update and the time it takes to perform an update.

Batch gradient descent also known as *Vanilla gradient descent* computes the gradient of the cost function with regards to the parameters θ for the entire training dataset:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

As this type of gradient descent needs to compute the gradients for the whole dataset for performing just a single update, it can be very slow and is intractable for datasets that don't fit in memory. Batch gradient descent also does not allow to update any model online, i.e. with new examples on-the-fly.

Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example x(i)x(i) and label y(i)y(i):

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.

While batch gradient descent converges to the minimum of the basin the parameters are placed in, SGD's fluctuation, on the one hand, enables it to jump to new and potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting. However, it has been shown that when the learning rate is slowly decrease, SGD shows the same convergence behaviour as batch gradient descent, almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively.

The last one is *Mini-batch gradient* descent which takes the best of both worlds and performs an update for every mini-batch of neural network training examples:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i,i+n)})$$

Using this type of gradient descent, it lets to reduces the variance of the parameter updates, which can lead to more stable convergence; and could make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient with regard to a mini-batch very efficient. Common mini-batch sizes range between 50 and 256, but can vary for different applications. Mini-batch gradient descent is typically the algorithm of choice when training a neural network and the term SGD usually is employed also when mini-batches are used.

In Keras, optimizers like SGD are used in order to optimize better weights in the neural network. Many algorithms have been developed in Keras which are:

- *ADAM* [7, 86]: it is a method that computes adaptive learning rates for each parameter. Adam algorithms keeps the first order moments m_t and the second order moments g_t of the gradients, and it lets to decay both over time [163]. The formulas on which Adam is based, are:

$$\begin{aligned} m_{t+1} &= \alpha_1 m_t + (1-\alpha_1)\Delta\theta \\ g_{t+1} &= \alpha_2 g_t + (1-\alpha_2)\Delta\theta^2 \\ \hat{m}_{t+1} &= \frac{m_{t+1}}{1-\alpha_1^{t+1}} \\ \hat{g}_{t+1} &= \frac{g_{t+1}}{1-\alpha_2^{t+1}} \\ \theta &\leftarrow \theta - \eta \frac{\hat{m}_{t+1}}{\sqrt{\hat{g}_{t+1}+\epsilon}} \end{aligned} \tag{4.2}$$

- *ADAGRAD*[164]: it is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data. In [165], they have found that *ADAGRAD* greatly improves the robustness of SGD and they have used it for training large-scale neural nets at Google, which learned to recognize cats in Youtube videos. *ADAGRAD* uses a different learning rate for every parameter θ_i at every time step t. One of Adagrad's advantages is that it help to avoid the need to manually tune the learning rate. Most implementations use a default value of 0.01 and leave it at that. *ADAGRAD*'s main weakness is its accumulation of the squared gradients in the denominator. Since every added term is positive, the accumulated sum keeps growing during training. This in turn causes the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge.

- *ADADELTA* [166]: it is an extension of *ADAGRAD* that seeks to reduce the behaviour of the learning rate slope which it is monotonically decreasing. Instead of accumulating all past squared gradients, it process the sum of each gradient g_t which is recursively defined as a decaying average of all past squared gradients. A learning rate for ADADELTA is not needed becuase it proceeds following the previous and current gradient.

- *NADAM*: it is a combination of *ADAM* and Nesterov Accelerated Gradient (NAG) which is a way to give a momentum term this kind of prescience. Nesterov's Accelerated Gradient Descent performs a simple step of gradient descent to go from x_s to y_{s+1}, and then it *slides* a little bit further than y_{s+1} in the direction given by the previous point y_s. Nesterov's Accelerated Gradient is an optimal method for smooth convex optimization. For apply NAG in ADAM, it is required to modify *ADAM* momentum. This demonstrates that momentum involves taking a step in the direction of the previous momentum vector and a step in the direction of the current gradient. NAG then allows us to perform a more accurate step in the gradient direction by updating the parameters with the momentum step before computing the gradient.

- *RMSprop* [167]: it is an adaptive learning rate method proposed by Geoffrey Hinton. *RMSprop* and *ADADELTA* have both been developed in order to solve *ADAGRAD* radically diminishing learning rates problem. RMSProp's "idea is to divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight" [168]. The update formula is given by [6]:

$$\begin{aligned} g &= \alpha g + (1-\alpha)\Delta\theta^2 \\ \theta &\leftarrow \theta - \eta \frac{\Delta\theta}{\sqrt{g+\epsilon}}, \end{aligned} \tag{4.3}$$

  where θ represents the weights shared across all threads, $\Delta\theta$ is the accumulated gradients of the loss with respect to the θ, η is the learning rate, α is the momentum that keeps knowledge of the previous experience, and g is "the moving average of element-wise squared gradients" [6].

The theoretical comparison between these algorithms is needed to understand the properties of convergence to the optimal. As far as concerned this case studies, it has been tried to use *ADAM* and *RMSProp*.

Each of the optimizers that could be chosen will present the problem of identifying a learning rate because a too low learning rate results in slow training, whereas a too big learning rate would cause a lot of noise in the objective function so that it would never converge. Not only the speed of training is an indicator of a good learning rate, but also the plotted loss function. Usually, if the learning rate is smaller, then the loss function will be smoother, whereas if the learning rate is greater than the loss function will look noisy in the graphs. These SGD extensions try to solve the learning rate problem by adapting it for each of the parameters.

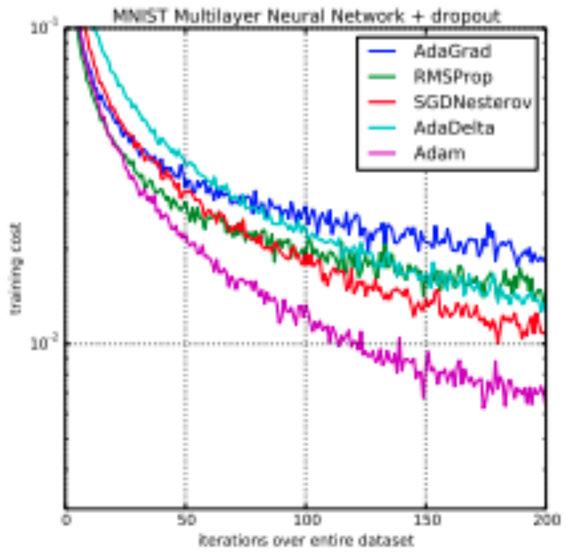


FIGURE 4.9: *A general comparison using MNIST between different optimizer using the same Neural Network. Image taken from [7]*

Nevertheless, in [7], there is a general comparison between all of this algorithm and the result it could be found in Figure 4.9.

The optimizer in this work is *Adam* with a learning rate equal to 0.0001 as suggested by the authors.

4.5 Activation functions

The ability of the neural networks to approximate any functions is directly the result of the non-linear activation functions. Every kind of activation function takes a vector and performs a certain fixed point-wise operation on it. There are three main activation functions, which are:

- *Sigmoid*:
$$y = \sigma(x) = \frac{1}{1 + e^{-x}}$$
It takes a real value and squashes it between 0 and 1. However, when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Thus, the back-propagation algorithm fail

at modifying its parameters and the parameters of the preceding neural layers.

- *Hyperbolic Tangent* (tanh):

  $$y = 2\sigma(2x) - 1$$

  It squashes a real-valued number between -1 and 1. However it has the same drawback than the sigmoid.

- *Rectified Linear Unit*:

  $$y = \max(0, x)$$

  The *ReLU* has become very popular in the last few years, because it was found to greatly accelerate the convergence of stochastic gradient descent compared to the *sigmoid/tanh* functions due to its linear non-saturating form (e.g. a factor of 6 in [111]). In fact, it does not suffer from the vanishing or exploding gradient. An other advantage is that it involves cheap operations compared to the expensive exponentials. However, the ReLU removes all the negative informations and thus appears not suited for all datasets and architectures.

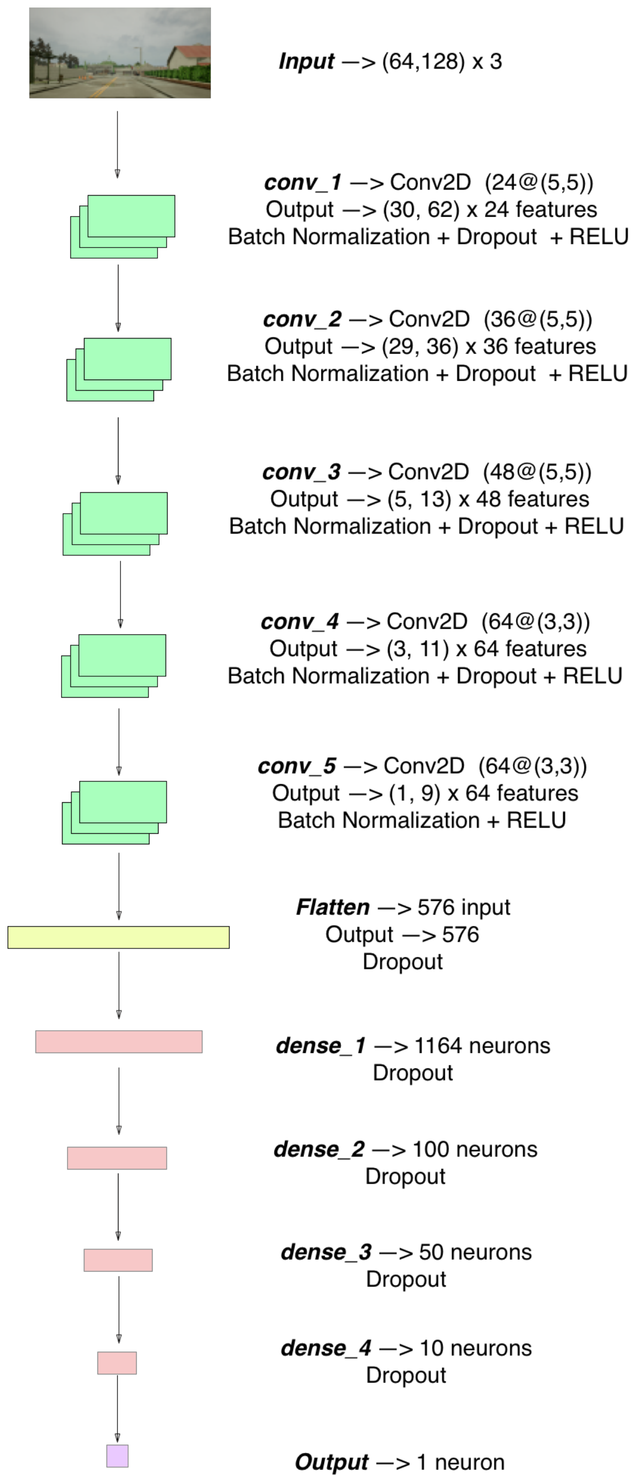
4.6 CNN architecture

This algorithm is inspired by [2]. The methodology adopted for learning is *Behavioral Cloning* which is an attempt to emulate human drivers behaviour using labelled data. In [2], they have used CNN. It is inspired by *AlexNet*, *GoogLeNet* and by Pomerlau work at ALVINN[1]. It learns the features of the road and associate with a value needed to steer the car. This model is composed by five convolutional layer. The first three have a filter size of 5×5, followed by two layers with filter size 3×3. The first three layer use 2×2 sub-sampling, the last two do not use 1×1 sub-sampling. The objective is to learn all the features by the proposed image contained inside the dataset and processed by the Batch Simulators. The proposed implementation is slightly different from the original one. The model includes *ReLU* activation layers after each convolutional or fully connected layer. It has been tried also with *ELU* activation layer but the result has not been as good as with *ReLU*. The weights are initialized using *Glorot uniform initializer*, also called *Xavier uniform initializer* [169]. It draws samples from a uniform distribution within $[-limit, limit]$ where limit is equal to $\sqrt{\frac{6}{(fan_{in}+fan_{out})}}$ where fan_{in} is the number of input units in the weight tensor and fan_{out} is the number of output units in the weight tensor. It has been provided a *BatchNormalization* layer after every convolutional layer in order to normalize data and makes the training be faster. It is a layer which it helps to converge faster. It adds a normalization step (shifting inputs to zero-mean and unit variance) to make the inputs of each trainable layers comparable across features. By doing this it ensures a high learning rate while keeping the network learning. Also it allows activations functions such as *TanH* and *Sigmoid* to not get stuck in the saturation mode (e.g. gradient equal to 0). There is also a *Dropout* layer with drop rate equals to 0.2, which means that 20% of the neurons are deactivated during training. Generally, 0.5 is a good value, but then more training data or epochs are needed for the remaining weights to build robust features. Usually *Dropout* layers are a good mean for preventing *overfitting* effect.

Afterwards, data are *flattened* and feed into three fully connected layer. The number of neurons in those layer are , and . The last neuron predicts the steering angle with no activation function at the end.

The model was trained and validated on different data sets to ensure that the model was not overfitting. For each epoch, the data set was shuffled, and for each batch at the end, too.

In Figure 4.10, there is an image which show how the CNN has been implemented.



Input —> (64,128) x 3
conv_1 —> Conv2D (24@(5,5))
Output —> (30, 62) x 24 features
Batch Normalization + Dropout + RELU
conv_2 —> Conv2D (36@(5,5))
Output —> (29, 36) x 36 features
Batch Normalization + Dropout + RELU
conv_3 —> Conv2D (48@(5,5))
Output —> (5, 13) x 48 features
Batch Normalization + Dropout + RELU
conv_4 —> Conv2D (64@(3,3))
Output —> (3, 11) x 64 features
Batch Normalization + Dropout + RELU
conv_5 —> Conv2D (64@(3,3))
Output —> (1, 9) x 64 features
Batch Normalization + RELU
Flatten —> 576 input
Output —> 576
Dropout
dense_1 —> 1164 neurons
Dropout
dense_2 —> 100 neurons
Dropout
dense_3 —> 50 neurons
Dropout
dense_4 —> 10 neurons
Dropout
Output —> 1 neuron

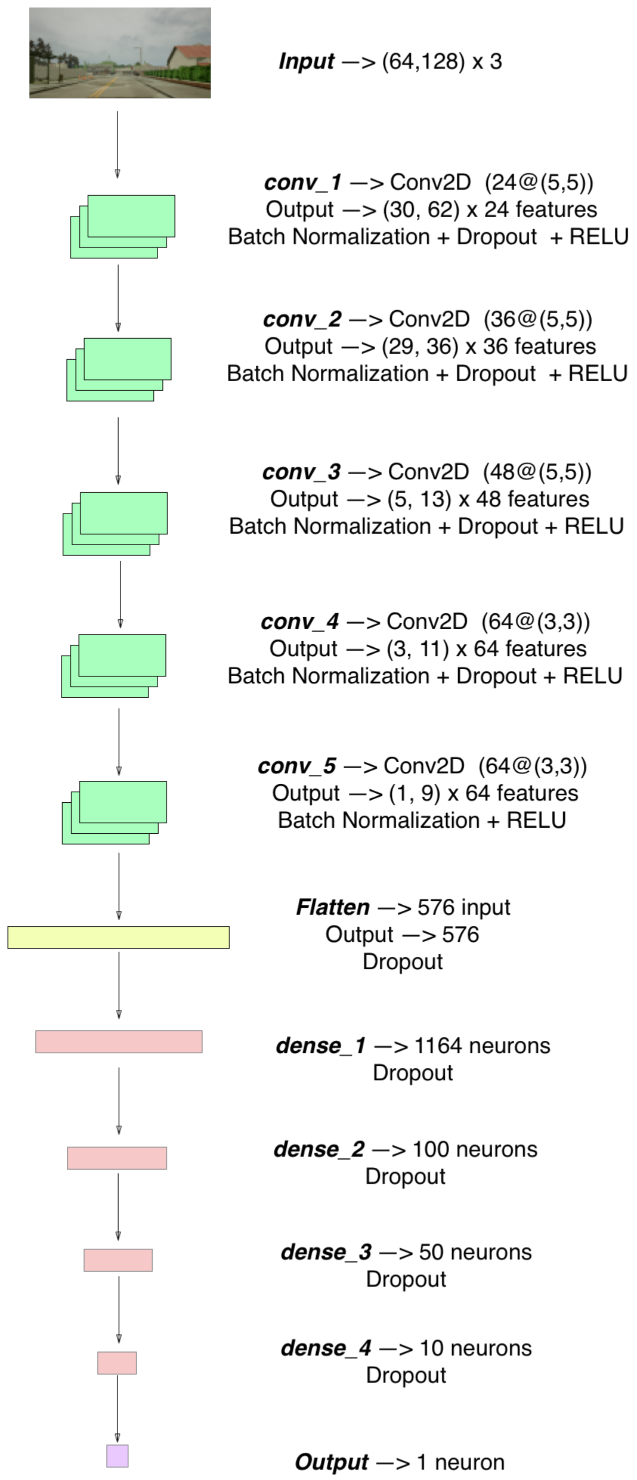
FIGURE 4.10: *Model CNN for DeepTesla implementation*

4.6.1 DeepTesla implementation

For DeepTesla simulator, the network will be fed using original dataset and will be tested the end-to-end learning property using only images as input and steering value as output to compare to DNN output.

This image analysed are high quality images that are taken frame by frame. It will be taken only the portion of space in front of the car which has dimension of 64x128 pixels which is the region of interest of the image for steering values. The evaluated loss function is the *minimum absolute error* which is the measure of comparison used for judging the methodologies in this thesis. All the other measure will be kept in the *Tensorboard* file just for logging purpose.

The model used is illustrated in Figure 4.11.



Input —> (64,128) x 3
conv_1 —> Conv2D (24@(5,5))
Output —> (30, 62) x 24 features
Batch Normalization + Dropout + RELU
conv_2 —> Conv2D (36@(5,5))
Output —> (29, 36) x 36 features
Batch Normalization + Dropout + RELU
conv_3 —> Conv2D (48@(5,5))
Output —> (5, 13) x 48 features
Batch Normalization + Dropout + RELU
conv_4 —> Conv2D (64@(3,3))
Output —> (3, 11) x 64 features
Batch Normalization + Dropout + RELU
conv_5 —> Conv2D (64@(3,3))
Output —> (1, 9) x 64 features
Batch Normalization + RELU
Flatten —> 576 input
Output —> 576
Dropout
dense_1 —> 1164 neurons
Dropout
dense_2 —> 100 neurons
Dropout
dense_3 —> 50 neurons
Dropout
dense_4 —> 10 neurons
Dropout
Output —> 1 neuron

FIGURE 4.11: *CNN model for DeepTesla implementation*

4.7 CNN LSTM architecture

The *CNN* model described previously is only capable of handling a single image, transforming it from input pixels into an internal matrix or vector representation.

The *CNN LSTM* architecture involves using Convolutional Neural Network (CNN) layers for feature extraction on input data combined with LSTMs to support sequence prediction. *CNN LSTMs* are designed and well suited for visual time series prediction problems and the application of generating textual descriptions from sequences of images (e.g. videos). *Activity Recognition* problems like the generation of a textual description of an activity demonstrated in a sequence of images, *Image Description* task like generation of a textual description of a single image or *Video Description* tasks like generating a textual description of a sequence of images are well suited for *CNN LSTMs*.

This architecture is also appropriate for problems that have spatial structure in their input such as the 2D structure or pixels in an image or the 1D structure of words in a sentence, paragraph, or document. Moreover problems that have a temporal structure in their input such as the order of images in a video or words in text, or require the generation of output with temporal structure such as words in a textual description are another type that *CNN LSTM* is able to exploit very well.

"They are a class of models that is both spatially and temporally deep, and has the flexibility to be applied to a variety of vision tasks involving sequential inputs and outputs" (quoted from [170]). This architecture was originally referred to as a *Long-term Recurrent Convolutional Network* or *LRCN* model, although it will be used *CNN LSTM* to refer to *LSTMs* that use a *CNN* to detect features inside the image. Conceptually, there is a single CNN model and a sequence of LSTM models, one for each time step. Each input image needs to apply the CNN model and pass on the output of each input image to the LSTM as a single time step[171].

4.7.1 DeepTesla implementation

It is inspired by the CNN model but three LSTM layers has been added which will keep a temporal series of data. The output will be merged to the output of the CNN. Using this methodology, the network will be able to process temporal data series and know how to react.

This image analysed are the same used for CNN model. It will be taken only the portion of space in front of the car and the image will be reduced to a 64×128 pixels image.

Three LSTM layer has been added between convolutional layers and fully connected layers in order to preserve the sequence order.

The model used is illustrated in Figure 4.12.

4.7.2 CARLA implementation

A CNN needs more input than a single image if task in a simulated environment are required. If a car needs to steer, it has to regulate speed. If current is speed is too high, the car will not be able to steer correctly. The mean used by humans to regulate speed are *throttle* and *brake* pedal. CARLA provide smany data like *RGB images*, *depth images* and many other data on the environments and other agents. RGB images are preprocessed in order to decrease the size and to take only a region of interest. Measurements taken involves *speed*, *throttle*, *brake* and *steering angle* of the vehicle. The new model configuration is shown in Figure 4.13.

4.8 Asynchronous Advantage Actor Critic (A3C)

Asynchronous Advantage Actor Critic (A3C) is a new approach in Deep Reinforcement Learning. In this work, *A3C* will be implemented onto the idea of driver-less cars. A similar achievement has been reached in [172] where the agent was trained using DDPG in order to train an agent to drive in a circuit of TORCS.

The project was mainly inspired by the article [6], and also from the implementation of the A3C into the Doom game elaborated in [173].

The idea behind the *A3C* is very much around the same *actor-critic* approach. It uses both policy and value approximations. The *actor* estimates the policy weights vector for choosing an action and the *critic* estimates the value weights for providing the information about the quality of a state the agent ends up in after making the action according to the policy. So, the actor is about the learned policy and the critic - about the learned value function.

There is an algorithm called *reinforce* and it also uses both policy and value function approximation, but it does not bootstrap. The critic, on the other hand, is a bootstrapping critic, which updates its states based on the value estimates of the next states. So, the initial reinforce algorithm has been changed for the actor-critic method: it's full return was replaced by one-step return. The one-step actor-critic algorithm is presented in Algorithm 1 [3].

Policy gradient methods work for discrete action spaces as well as for continuous. In the discrete actions problem, the policy is estimating the probability of each action in the discrete set. In the continuous actions space, on the other hand, the policy approximates the variance σ^2, and the mean μ of a normal distribution given by:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(x-\mu)^2}{2\sigma^2}), \tag{4.4}$$

where $p(s)$ is the density of the probability at x. This is what makes the basis for the construction of a continuous policy, in which the weights-vector θ is included. The continuous policy formula is provided below:

$$\pi(a|s,\theta) = \frac{1}{\sigma(s,\theta)\sqrt{2\pi}}\exp(-\frac{(a-\mu(s,\theta))^2}{2\sigma(s,\theta)^2}) \tag{4.5}$$

The policy weights vector $\theta = [\theta_\mu, \theta_\sigma]^T$ is composed of two elements, θ_μ and θ_σ, that can be used further in function approximation algorithms. The form that these elements take is shown below:

$$\mu(s,\theta) = \theta_\mu^T\phi(s) \text{ and } \sigma(s,\theta) = \exp(\theta_\sigma^T\phi(s)), \tag{4.6}$$

Algorithm 1 One-step Actor-Critic (episodic)

```
Input: differentiable policy parametrization $\pi(a|s,\theta), \forall a \in A, s \in S, \theta \in \mathbb{R}^n$
Input: differentiable state-value parametrization $\hat{v}(s,w), \forall s \in S, w \in \mathbb{R}_m$
Parameters: $\alpha > 0, \beta > 0$
Initialize policy weights $\theta$ and state-value weights $w$
repeat
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    while $S$ is not terminal do:
        $A \sim \pi(\cdot|S,\theta)$
        Take action $S$, observe $S'$, $R$
        $\delta \leftarrow R + \gamma\hat{v}(S',w) - \hat{v}(S,w)$ (if $S'$ is terminal, then $\hat{v}(S',w) = 0$)
        $w \leftarrow w + \beta\delta\nabla_w\hat{v}(S,w)$
        $\theta \leftarrow \theta + \alpha I\delta\nabla_\theta \log\pi(A|S,\theta)$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
until forever
```

where $\phi(s)$ is the feature vector, which can be, for example, the pixel values vector of an image. In this work, discrete actions space is considered.

An additional feature of the actor-critic method is the *asynchronous* part. Instead of having a single agent training on the GPU, multiple agents are instantiated for training on different CPU threads simultaneously, and they share a global network, which is updated as the agents advance. Another feature of the A3C is the *advantage* element, which is a way of expressing how much better some actions ended up to be, and where the estimation should be improved. The update performed by the A3C is of the form $\nabla_{\theta'}\log\pi(a|s,\theta')A(s,a,\theta,\theta_v)$, and the formula for the advantage is $\sum_{i=0}^{k-1}\gamma^i r_{t+i} + \gamma^k V(s_{t+k},\theta_v) - V(s_t,\theta_v)$, which are both taken from the article [6]. The update formula changes slightly after including the entropy factor in the policy in order to encourage exploration and avoid convergence to an earlier suboptimal solution. The detailed A3C algorithm taken from [6] is listed in Algorithm 2.

Algorithm 2 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```
//Assume global shared parameter vectors $\theta$ and $\theta_v$ and counter $T = 0$
//Assume thread-specific parameter vectors $\theta'$ and $\theta'_v$
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state $s_t$
    repeat
        Perform action $a_t$ according to policy $\pi(a_t|s_t, \theta')$
        Receive reward $r_t$ and new state $s_{t+1}$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal or $t - t_{start} == t_{max}$
    if $s_t$ is terminal then
        $R = 0$
    else
        $R = V(s_t, \theta'_v)$ //bootstrap from last state
    for $i \in \{t - 1, ..., t_{start}\}$ do
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt $\theta'$:
        $d\theta \leftarrow d\theta + \nabla_{\theta'} \log\pi(a_i|s_i, \theta')(R - V(s_i, \theta'_v))$
        Accumulate gradients wrt $\theta'_v$:
        $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i, \theta'_v))^2 / \partial\theta'_v$
    Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$
until $T > T_{max}$
```

4.8.1 A3C on CARLA

The A3C algorithm developed for the CARLA environment uses images as the input states into a ANN structure and the mathematical model of the problem is similar. The CARLA environment is very flexible and that is an advantage as it offers more freedom for changing things and get better results. For example, it is easier to change the reward function and perform action manipulations.

Asynchronous

A3C algorithm implies a multiplicity of agents which are trained simultaneously. They share a global network, but the training environment is different. Each *worker* interacts with its own environment, which is the CARLA Town 1 it is running on. The *worker* drives the car in its own environment until:

- the car collides with an object, pedestrian or a car.
- 10% of the car is over the sidewalk
- 10% of the car is over the other lane.
- if cumulate reward of 100 frame is less then 100.

Every time the run starts, the *worker* will find the car in a different position in order to not overfit the network. The global network is used and continually updated by all the training workers. This facilitates the process of training because every worker improves the most recent version of the global network. Every time a condition is not met or the maximum episode number is reached, it restarts from a new position in the same town.

The global actor-critic network is declared before the workers. Tensorflow has special dedicated methods for enabling multi-threading, which is the *Coordinator*. The workers are declared and initialized, after which each of them is assigned to a Thread, where they are started to run their own actor-critic network. The multi-threading process is stopped with *join()* method when all the workers finished their work. Owing to performance reason, 4 workers has been instantiated to update the global network.

Model

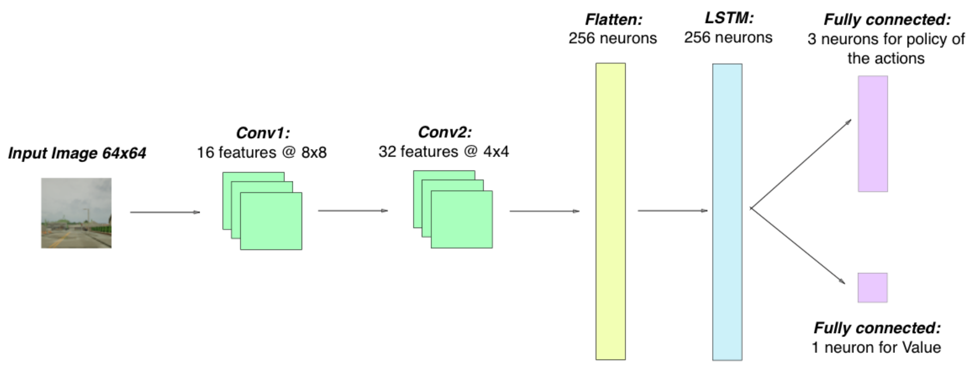


Figure 4.14: *The ANN structure of the A3C implementation into CARLA*

The image passed to the network is smaller than 64×128 (as used in CNN LSTM). The new image size is 64×64. The Actor-Critic Network is composed by a convolutional layer that outputs 16 feature maps of sizes 8×8 each. Next, these are again passed through another convolutional layer that outputs 32 feature maps of size 4×4 each, while taking care of spatial correlations. Then the output is flattened with a fully connected layer and passed to a recurrent layer - basic LSTM, that takes care of the temporal dependencies. Finally, the output of the LSTM layer can be used for the last layers of the ANN. The *value function* is linearly estimated, while the *policy* is estimated and gives the probabilities of each action. These too are then used in the formation of a normal distribution, which is then sampled to get the action to be passed to the environment. The size of the layer is 3 for the policy as it is used for the 3 actions of the environment, namely, *steer*, *throttle*, and *brake*. The final structure of the ANN of the A3C is illustrated in Figure 4.14.

Putting together the A3C theory and the ANN structure described above, a very general illustration of the flow of the program is generated for creating a better understanding of the whole project in Figure 4.15.

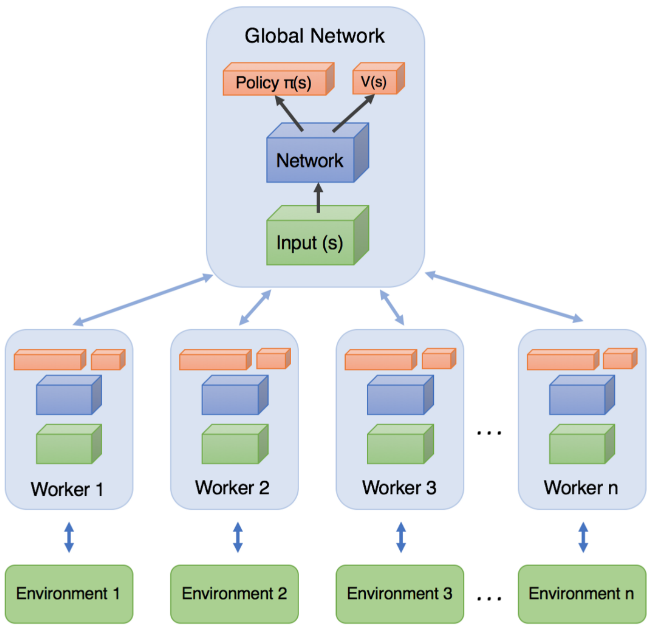


FIGURE 4.15: *The flow of the A3C CARLA program. Image taken from [173]*

Initially, a global network is defined and a number of agents or workers are instantiated to train themselves in their own environment using a copy of the global network. During training, as the first state of the environment is received and passed through all the ANN layers, the worker picks the action with the highest probability given by the output of the discrete policy layer, and then the worker executes the action while the environment returns the next state and the reward. The states keep coming during an episode and the rewards keep accumulating together with the values in an episode buffer. The episode buffer has a specific size, e.g. 100, so that after 100 episodes it becomes full, the global network is updated with the current value estimate - the bootstrap value and the episode buffer is emptied. Nevertheless, at every step, the global network is also updated with the data from the episode buffer without a bootstrap value. The update of the global network happens in a stable way thanks to the episode buffer and it is performed by applying gradients that were computed for a defined loss function which is composed of the value loss and policy loss. The value loss is calculated based on the *Temporal Difference* (TD) error, which is the squared difference between the target value and the estimated value. The policy loss is computed based on the logarithm of the taken actions multiplied by their probabilities and multiplied by the advantages.

Reward

The reward is important in reinforcement learning because the agent must maximize the reward for learning how to act in the environment. Lets take as an example teaching a dog a new trick. You cannot tell it what to do, but you can reward/punish it if it does the right/wrong thing. The dog should figure out what it did to get the reward/punishment, which is known as the credit assignment problem [174]. Similar methods are possible to train the agent how to drive a car.

In this work, the reward formula is based on the one defined by [160]. It is calculated as:

$$r_t = d + 0.05(v) - 0.00002(c) - 2(s) - 2(o).$$

where:

- d is the Euclidean distance in meters between the previous point position and actual point position;
- v is the difference between the actual speed and the previous speed;
- s is the percentage of the car that has intersected the side walk;
- c is the number which indicated the collision damages that the car could increase with the collision with other car, pedestrians or object in the urban environment;

- o is the percentage of the car that has intersected the other lane.

This version of reward is slightly different from [160]. They used distance towards a goal in kilometres, while in this work it has been used the Euclidean distance between points in metres. Moreover, during reward calculation, more penalties are added in case the vehicle is not moving for too much time.

4.9 Conclusion

In this section, the algorithms chosen for each of the experiment are illustrated and described from the idea to model implemented. In the next chapter, results of training and testing will be illustrated and described for both experiments and all of the algorithms will be then compared in each of the experiments.

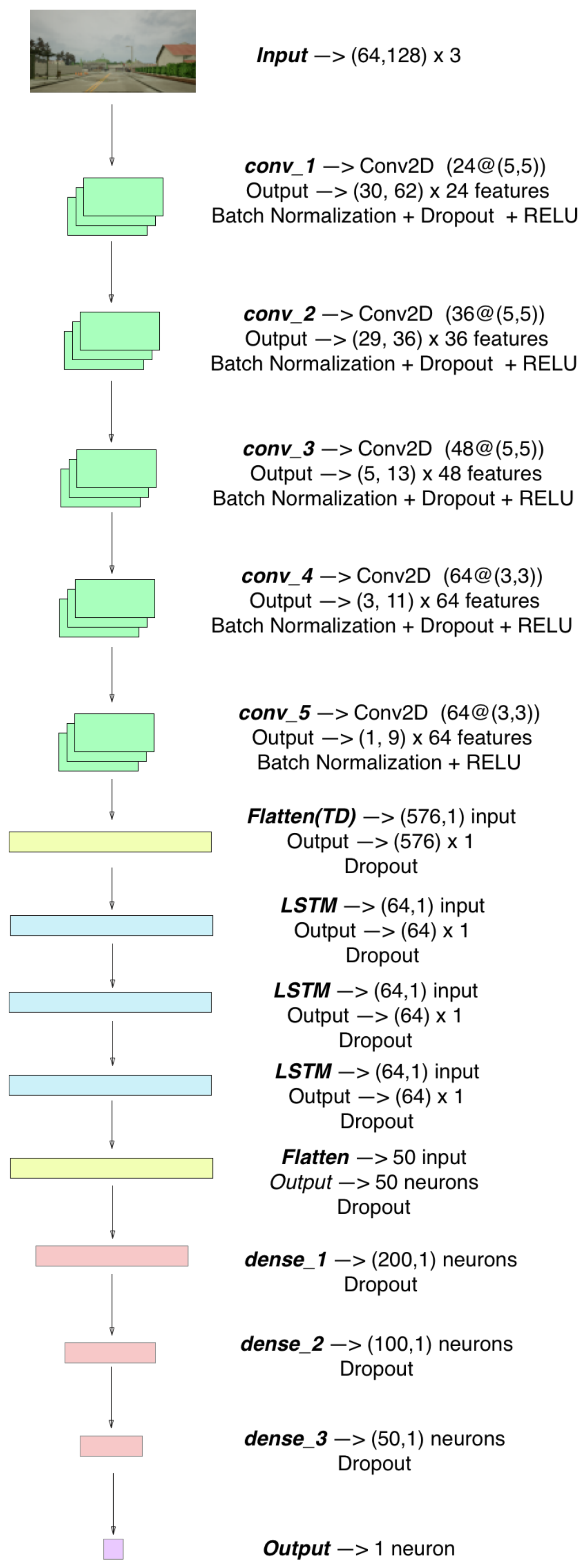

Input —> (64,128) x 3
conv_1 —> Conv2D (24@(5,5))
Output —> (30, 62) x 24 features
Batch Normalization + Dropout + RELU
conv_2 —> Conv2D (36@(5,5))
Output —> (29, 36) x 36 features
Batch Normalization + Dropout + RELU
conv_3 —> Conv2D (48@(5,5))
Output —> (5, 13) x 48 features
Batch Normalization + Dropout + RELU
conv_4 —> Conv2D (64@(3,3))
Output —> (3, 11) x 64 features
Batch Normalization + Dropout + RELU
conv_5 —> Conv2D (64@(3,3))
Output —> (1, 9) x 64 features
Batch Normalization + RELU
Flatten(TD) —> (576,1) input
Output —> (576) x 1
Dropout
LSTM —> (64,1) input
Output —> (64) x 1
Dropout
LSTM —> (64,1) input
Output —> (64) x 1
Dropout
LSTM —> (64,1) input
Output —> (64) x 1
Dropout
Flatten —> 50 input
Output —> 50 neurons
Dropout
dense_1 —> (200,1) neurons
Dropout
dense_2 —> (100,1) neurons
Dropout
dense_3 —> (50,1) neurons
Dropout
Output —> 1 neuron

FIGURE 4.12: *CNN model for DeepTesla implementation*

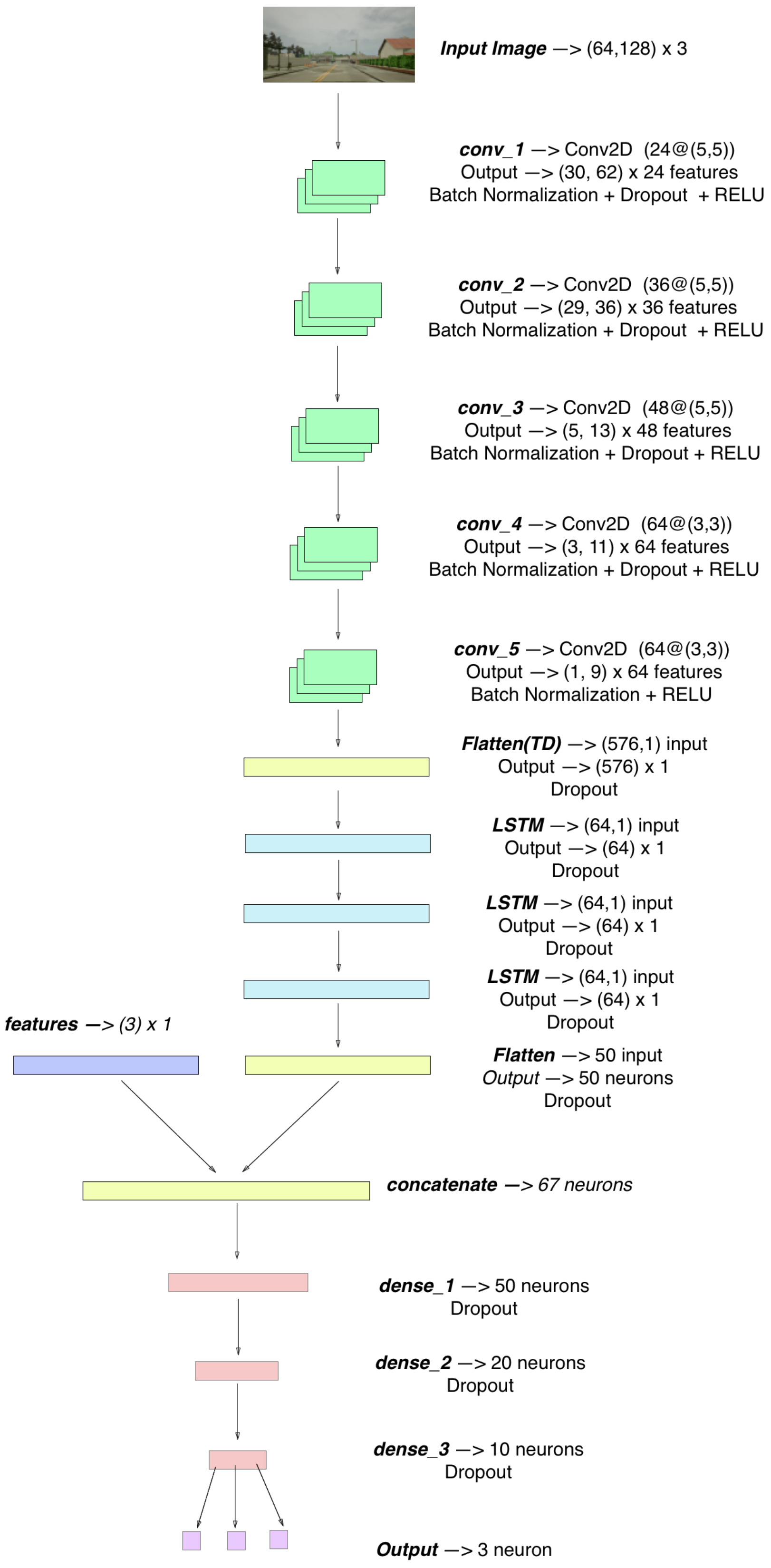

Input Image —> (64,128) x 3
conv_1 —> Conv2D (24@(5,5))
Output —> (30, 62) x 24 features
Batch Normalization + Dropout + RELU
conv_2 —> Conv2D (36@(5,5))
Output —> (29, 36) x 36 features
Batch Normalization + Dropout + RELU
conv_3 —> Conv2D (48@(5,5))
Output —> (5, 13) x 48 features
Batch Normalization + Dropout + RELU
conv_4 —> Conv2D (64@(3,3))
Output —> (3, 11) x 64 features
Batch Normalization + Dropout + RELU
conv_5 —> Conv2D (64@(3,3))
Output —> (1, 9) x 64 features
Batch Normalization + RELU
Flatten(TD) —> (576,1) input
Output —> (576) x 1
Dropout
LSTM —> (64,1) input
Output —> (64) x 1
Dropout
LSTM —> (64,1) input
Output —> (64) x 1
Dropout
LSTM —> (64,1) input
Output —> (64) x 1
Dropout
features *—> (3) x 1*
Flatten —> 50 input
Output —> 50 neurons
Dropout
concatenate *—> 67 neurons*
dense_1 —> 50 neurons
Dropout
dense_2 —> 20 neurons
Dropout
dense_3 —> 10 neurons
Dropout
Output —> 3 neuron

FIGURE 4.13: *CNN model for CARLA implementation*

Chapter 5

Results and Discussion

After all the experiments and algorithms have been explained and motivated, in this chapter the summary of all the results obtained is presented and discussed. Every experiment has been run using JetBrains IDE. Most of them, especially *CNN* and*CNN LSTM* for DeepTesla and *CNN LSTM* for CARLA use GPU, while Traditional computer vision algorithm and A3C are CPU-based. They are implemented using *Python3* with *Tensorflow 1.3* and *Keras 2.OpenCV 3* has been used for image preprocessing and for traditional computer vision algorithm. *Matplotlib* has been used for process image in graphs.

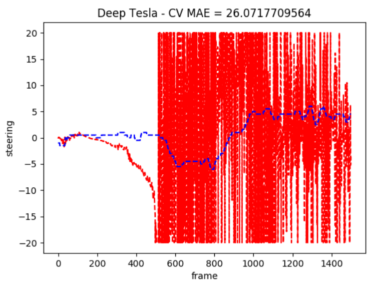
All the results found, the graphs and *Tensorboard* related to the current project are available inside the repository[161].

5.1 First Experiment - DeepTesla

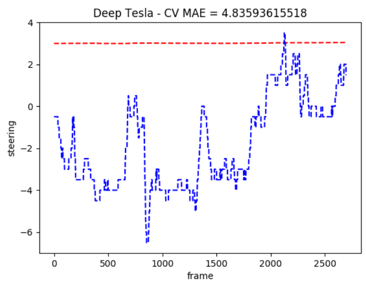
In this experiment, a modification and revision of the original project is needed to adapt it to the experiment. The visualization mechanism is left as is. CNN models and CNN LSTM model are all written using Keras which is not supported in the original version. Support to Keras models will be added and also to CV models.

Traditional Image Processing algorithms results

The processing of the image provides as output steering angles which will be used to steer the car. One of the curse of traditional image processing is that it needs a camera with high resolution image which let the car identify correctly the line markings provided. The results could be seen in Figure 5.1a and Figure 5.1b.



(a) *CV results from test 1*



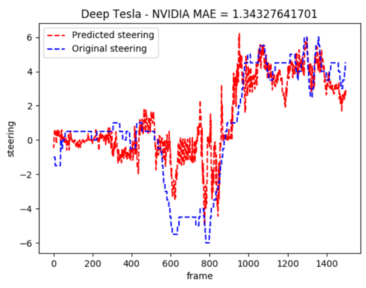
(b) *CV results from test 2*

FIGURE 5.1: *Results obtained during the evaluation of Traditional Image Processing solution using DeepTesla.*

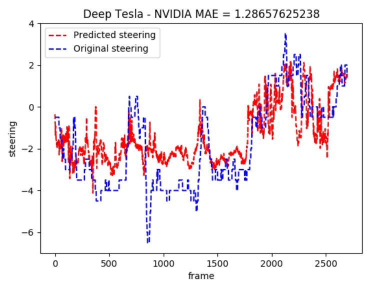
CNN model result

For this algorithm, different configuration of CNN has been tried for searching the best possible validation accuracy and minimum mean absolute error during validation. For this reason, the model obtained has gained the validation loss accuracy as it could be seen in [161].

Results are illustrated in Figure 5.2a and Figure 5.2b.



(a) *CNN results from test 1*

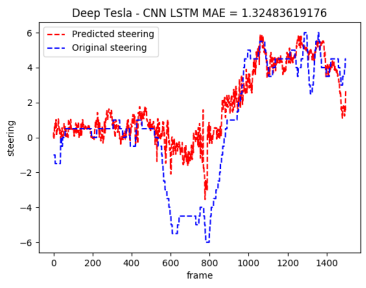


(b) *CNN results from test 2*

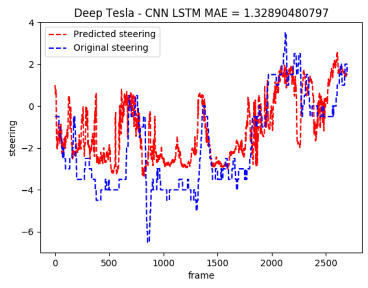
Figure 5.2: *Figure 5.2a and Figure 5.2b are the results obtained during the evaluation of CNN solution using DeepTesla.*

CNN LSTM model result

For this type of network, consideration to be done are similar to CNN model. The only differences between the models are the three LSTM layer which should result in an improvement in the performance of the network. Results are illustrated in Figure 5.3a and Figure 5.3b.



(a)



(b)

Figure 5.3: *Figure 5.3a and Figure 5.3b are the results obtained during the evaluation of CNN LSTM solution using DeepTesla.*

5.1.1 Discussion

Traditional computer vision algorithm has obtained the worst results among the various algorithm. When light condition are very poor and the quality is not high, this type of algorithm seems to not be the best. Many scholars and researchers have always tried to solve the problem related to changing of light condition and poor resolutions. In this case, results are caused by the video frames which contains lane marking that are discontinuous and not well marked. The algorithm so is not able to recognize right lane or left lane. Moreover, in some part of the video there is also the reflection of the vision sensor used while recording which does not help. This result is not very good as the predicted steering is often outside the range defined. If a road with no markings is taken in consideration, this algorithm would not be able to detect the limit of the road. Moreover, CV solution, as it has been implemented, is very slow because it takes about 500ms to obtain a single steering angle from one frame. The solution proposed could be improved in order to be more efficient using GPU with OpenCV, but by now there are errors during the compilation of the OpenCV version.Traditional CV approach, as considered, is not sufficient to be proposed as a solution for predict steering angles.

CNN solution has attained a good result. In fact, as the resulting graph from the test could demonstrate, the predicted steering line is very similar to original steering. As a note, original steering is dependent on the behaviour of the driver obtained during the recording of the data and so it happens that there are odd peaks which the network does not recognize that influence the average MAE. Moreover original data are approximated and does not respect the real steering angle. Considering the range of value of steering that in this case is $[-15; 15]$, and the average MAE found which in both case is 1.32 approximately, it has about 95% of accuracy in the result found for the test case which is a great score considering the difficulty of the image and the detection of the feature. Moreover, loss training curve has a good slope which means that data augmentation and batch normalization have avoided overfitting effect.

CNN LSTM solution seems to have resolved this problems. The temporal sequence seems to affect mainly the amplitude of the peaks of the prediction which seems to be reduced compared to the CNN solution. However the consideration are similar to CNN as far as concerned learning curve slope and MAE. The average between the MAE seems to be better for *CNN LSTM*.

5.2 Second Experiment - CARLA

In this experiment, a different approach to autonomous driving needs to be tested. Results obtained in CNN LSTM and A3C will be explained and compared in this section.

5.2.1 CNN LSTM algorithm results

For this algorithm, the consideration done in DeepTesla experiment are the same. The network will take as input the speed and the image, while it provides as output throttle, steering and brake value. Those value are then passed to the controller which will send the control value as predicted by the CNN LSTM. The results of the training and validation are showed in Figure ??.

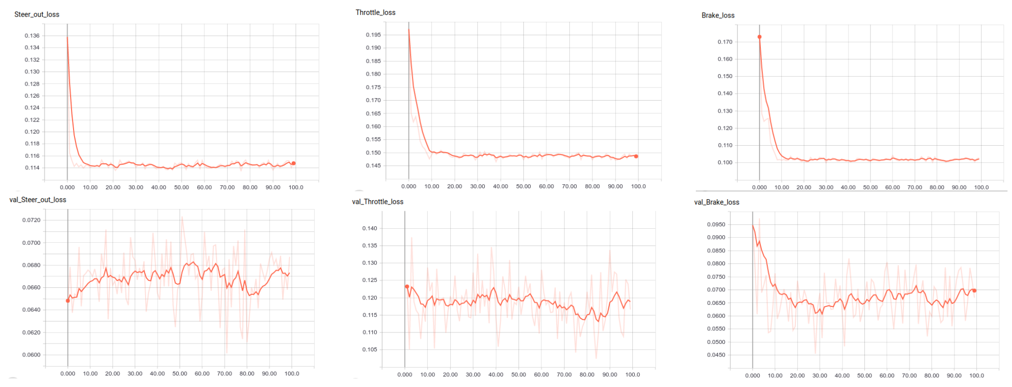


FIGURE 5.4: *Training(up) and validation(down) losses*

5.2.2 A3C algorithm results

A3C algorithm has been trained using 4 working agents that constantly update a Global Agent. The training has been done in Town 1 where the agents has learned to steer, do not pass the lane and avoid collision as the reward specification wants. The agent are divided in 4 environments with different weather conditions.

In Figure 5.5, there is a picture taken during training. In Figure 5.6c, they are displayed the rewards obtained during the training session for all of the 4 training agents. Moreover *Policy Loss* and *Value loss* are displayed in Figure 5.6a and Figure 5.6b respectively. The graphs shows the result obtained from the average value of the losses among all of the agents.

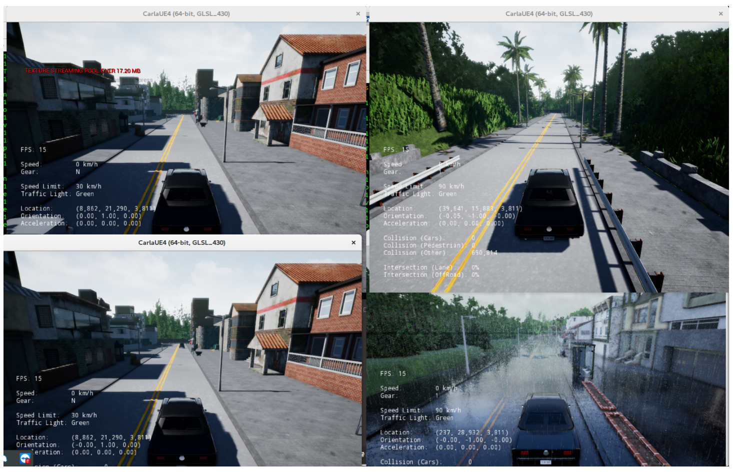
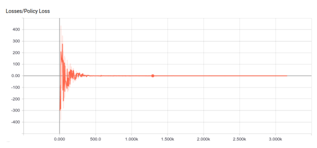
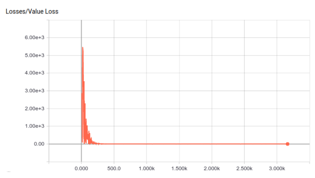

CarlaUE4 (64-bit, GLSL_430)
CarlaUE4 (64-bit, GLSL_430)
CarlaUE4 (64-bit, GLSL_430)

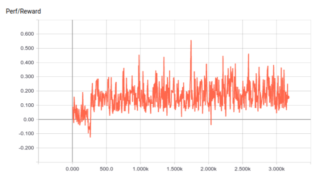
FIGURE 5.5: *Four agents working in different conditions*



(a) *Policy Loss*



(b) *Value loss*



(c) *Reward loss*

5.2.3 Discussion

All the video of the tests done after the training has been collected inside the repository.

CNN LSTM has shown many critical issues in this experiment. It was not able to steer well in some cases during the testing even if steering losses in the training were good enough. It seems to be very sensible to hyperparameter and data. Wrong configuration of the network could lead to wrong prediction and errors that lead to unsatisfactory result. Probably, a wrong evaluation has led to wrong hyper-parameters that affected the bad behaviour and poor learning of the network. But, as the parameters and all the dimension are similar to the original developed in [2], probably the error in the evaluation has to be related to data, which probably are not good. Data are the fundamental mean that drives supervised learning methodology as *Behavioural Cloning*. In many competitions, participants have got more than 100 GB of data to feed their Deep Neural Network. Collect data, classify and clean data is a long work that requires experience and can affect the data. In this work, the context is related to a fictitious company which has few data available and wants to develop its own ADAS using AI methodologies. As said, this seems to be not the right choice as the sensibility of the CNN LSTM to data and hyperparameter choice affect the behaviour.

A3C, instead, has shown great performances. Every reinforcement learning algorithms requires many iterations to attain sufficiently good results. The job has been stopped after 3000 episodes as the cars starts to perform well in the environment which corresponds to 4 hours of training. The agents, initially are not able to accelerate or brake and they do not steer. When they learn to accelerate, they start to approach how to go straight on the way. The next step, that has affected all the agents is steering which is the most difficult task. The limit imposed has not been sufficient to make the agent learn to avoid other cars and pedestrians. A3C has outperformed CNN LSTM as it has given great results in about the same times CNN LSTM has been trained. As previously explained, sensibility of CNN in supervised learning case studies is very high and wrong specification could lead to error. This concept seems to not affect A3C as it adapts correctly to the environment even if a part of him contains a CNN. Deep Reinforcement learning, using A3C, seems to be the best methodology to learn a NN in autonomous driving context.

One of the main differences that it is highlighted during testing phase is the collision avoidance behaviour of the agent which uses A3C. The A3C solution in fact seems to have acquired in less time this behaviour while CNN LSTM agent has not. It prefers to take less reward for the episode than take negative reward. The main evidence of this concept is highlighted by the video *A3C - vehicle stop - position 20.mp4* in [161]. Moreover, it happens during execution of the agent that CNN LSTM sometimes turns without reason or crash into crash rail. A proof is reported respectively in video *LCRN - curve 2 - position 40.mp4* and *LCRN - curve 1 - position 21.mp4* in [161].

Conclusions

In this work, we have surveyed the history of the main artificial intelligence technologies for autonomous driving, with emphasis on the most recent advances in deep learning. Many of the most relevant methodology existent in the state-of-the-art technologies that enables autonomous driving has been explained using examples. All the state-of-the-art algorithms currently used in ADAS systems have been illustrated and described with some relevant notions from machine learning and artificial intelligence field. Programming language, processors, deep learning frameworks and simulators that enables autonomous driving have been compared, described and selected for a comparison. To summarize, Python 3 has been chosen as the programming language used. The IDE adopted is *JetBrains PyCharm* which is free for academic usage. *Python* choice enabled the usage of *Tensorflow* (v1.3) with *Keras* (v2) as Deep Learning frameworks. Simulator chosen is CARLA, a young simulator currently in development which is born to be adopted in vision tasks and autonomous driving context. Then, all the algorithms developed have been described with related motivations that has lead to choose it. The algorithms implemented in current work are:

- *Traditional vision approach*
- *CNN*
- *CNN LSTM*
- *A3C*

The work has been divided in two different experiments, which are:

- *Vision task using DeepTesla*: Traditional vision approach has been compared to CNN and CNN LSTM approach in order to compare the end-to-end steering behaviour just using the vision camera sensor.
- *Sensor fusion and driving using CARLA*: CNN LSTM has been compared to A3C in CARLA to compare the behaviour in a simulated environment using sensors as input for the network.

The work undertaken in this thesis has been challenging with respect to several aspects. Many of the used frameworks or library are new and deployed only in Linux environments and they will probably be released on other operating systems in the next years. Some libraries and frameworks crash while building and they are supported only by community.

Results found seem to be very promising for future developments. A3C seems to be the best choice when a developer starts from scratch without data. It is a very good algorithm that applies the reinforcement learning paradigm using parallelism of the learning. As an improvement for future development, the reward function needs to consider also when an agent tries to pass an intersection with a red semaphore. A penalisation may be added as to improve the movement of the vehicle in an urban environment. CARLA will probably help to accomplish such task, but by now it does not provide any API that let to retrieve the nearest semaphore/traffic signals.

A3C is run in a discrete version. [118] has proposed a methodology that uses mean and variance for accomplish such task in a continuous fashion. As a future improvement, it could help to obtain relevant results using continuous control. However, reinforcement learning, even if powerful, it is not used in official real world autonomous driving systems.

CNN and CNN LSTM instead, are currently in development and used in some of the ADAS like *NVIDIA Drive PX series* on which [2] is based. Even Comma AI, a new startup of the Silicon Valley has provided in the original code, a network that resemble [1] and NVIDIA architectures. CNN is a great network for feature detection and suites very well in recognising lane markings. However, one of the great problems of these type of networks is that they need a great amount of data with labels and sometimes, just a little change could be dangerous for the recognition. In this work, the agent which has used CNN LSTM in CARLA has not been able to steer the vehicle in some cases. In other episode proposed during testing, CNN LSTM agent was not able to keep the lane and to stay on the road. Moreover, it does not recognize vehicles or pedestrians. A possible solution is to use pre-trained DNNs able to output if a vehicle or pedestrians is in the scene. Another possibility, however, which was initially considered, is using the *Semantic Segmentation* camera provided by CARLA. It classifies every object in the view by displaying it in a different colour according to the object class. E.g., pedestrians appear in a different colour than vehicles and sidewalk. The CARLA server provides an image with the tag information encoded in the red channel. A pixel with a red value of x displays an object with tag x. However this image is actually available but there are several bugs[1] in the actual code that make this solution impractical.

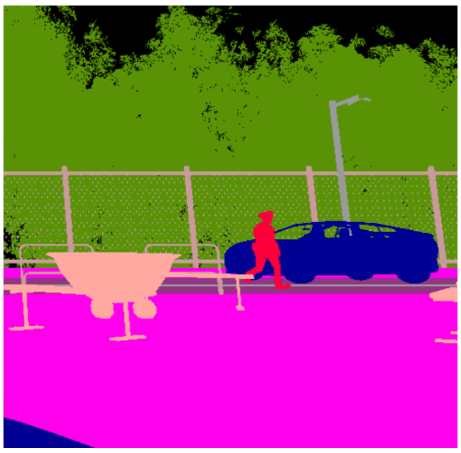


FIGURE 5.7: *Example of semantic segmentation.*

The collection of data and training using this data requires a long work

[1]`https://github.com/carla-simulator/carla/issues/4`

and it is sometimes hard to create a perfect network that is able to accomplish the task really well. The dependence from hyperparameters and data does not help CNN. [175] has provided a simple asynchronous optimisation algorithm which utilises a fixed computational budget to jointly optimise a population of model and their hyperparameter to maximise performances. It discovers a schedule of hyperparameters rather than following the general sub-optimal strategy of trying to find a single fixed set to use for the whole course of training. Using this algorithm could help to find the right hyperparameters that maximise the performance of the network. For this reasons, LiDAR and sensors are cited among the technologies that an autonomous car could use in the near future along with images and CNN. Google for example has adopted the sensor fusion to combine high resolution maps to LiDAR data in order to learn a network how to steer a car and when. In Figure 5.8, there is a representation that is a reconstruction done using points of a One of the drawbacks especially of the LiDAR is the cost. which is now decreasing but it is always over twenty thousands dollar.

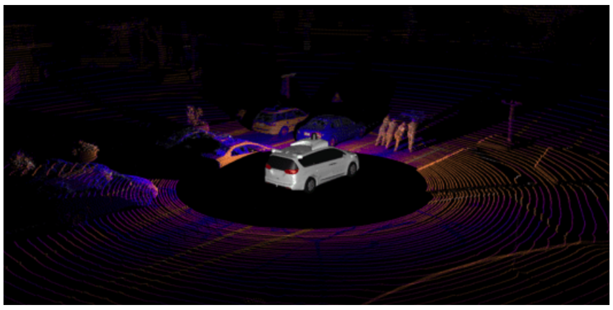


FIGURE 5.8: *Representation of the environment reconstruction using LiDAR technology. Image taken from [176]*

CNN are becoming obsolete to object detection and recognition tasks and they will not used any more. In fact, Sabour et al. [177] has recently released a paper that describes *Capsule Network*. "A *capsule* is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an object part"(quoted from

[177]).In the paper, they describe the case of MNIST recognition and it has been demonstrated how it outperform CNN. In the near future, new type of algorithm will be used instead of CNN also for feature detection. One of the aspects that mostly regards and sometimes frighten people is the *ethics*. What will the car do in case brake does not work and it percepts obstacle in front of it? Could it kill the driver or the nearest pedestrians in order to save the driver? The question is not so easy and there are studies on this problem since the rise of robotics. The problem is sometimes called *The Trolley problem.* It is is a thought experiment in ethics. The general form of the problem is this:

There is a runaway trolley barreling down the railway tracks. Ahead, on the tracks, there are five people tied up and unable to move. The trolley is headed straight for them. You are standing some distance off in the train yard, next to a lever. If you pull this lever, the trolley will switch to a different set of tracks. However, you notice that there is one person tied up on the side track. You have two options:

- *Do nothing, and the trolley kills the five people on the main track.*
- *Pull the lever, diverting the trolley onto the side track where it will kill one person.*

Which is the most ethical choice?[178, 179].

In my honest opinion, I think it is an unhelpful problem because not enough data are available to know at any given point, with what amount of certainty is the car going to kill anybody. Fatal accidents in autonomous driving cars have not happened yet in any meaningful numbers, so the necessary data does not exist to even work on the problem. Of course, in this work, there is no solution to this problem. In a future possibility that the *trolley problem* will happen, a paradigm like reinforcement learning could help to find new behaviours for trying to save lives. Reinforcement learning is one of the best learning approaches because it is adaptive and it could be used in order to learn how to act in difficult situation. Neural Networks are the right mean for such work because they can be used when it is not known a technology or algorithm that implement it but it is known how it has to behave in a situation. Simulators like CARLA combined with great car models could provide systems that are able to simulate really well the behaviour of a real car also in a simulated environment and could also help the development of improved and powerful simulator. If a simulator is realistic, the robot will succeed otherwise, it could be very difficult for it. Learning is the most interesting procedure that has attracted many neuroscientist and engineers since the mechanisms of neurons is known. But learning has not to stop only to cars. Fully autonomous driving is not possible by now. The reasons are many but the main is that people are not ready yet. Law needs to be similar among various states. Motorways, extra-urban and urban road are not good enough to such technologies. Many road are not paved and many other does not have markings. Fully autonomous car mean safety driving but safety has to come also from the roads. When the environment will be ready, the agent and fully autonomous driving will be possible. IoT could provide a great benefit to roads and create vehicle network for manage traffic jam conditions in a better way and help the vehicles to get out of the jam. Road is the next big thing that needs to learn. It needs to learn from the car and from the atmospheric agents in order to be more and more safe. A road which is able to learn and help the car to drive could resolve many safety problems but it needs many years to come out. For road, it is not meant the asphalt, but an artificial ecosystem composed by asphalt, sidewalk, lane, semaphore, intersection, signals and so on, which need to be connected by themselves and need to be managed. In 2017, humans have only SAE-3 vehicles which are the most autonomous vehicle on the road. They will probably be the most autonomous until a road revolution will not be taken.

Bibliography

[1] D. A. Pomerleau, "ALVINN: An Autonomous Land Vehicle in a Neural Network," in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1989, pp. 305–313. [Online]. Available: http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf

[2] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[3] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 2nd ed. Cambridge, MA, USA: MIT Press, 1998.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[8] SAS, "Machine Learning. What it is and why matters." [Online]. Available: https://www.sas.com/en_us/insights/analytics/machine-learning.html

[9] S. K. Gehrig and F. J. Stein, "Dead reckoning and cartography using stereo vision for an autonomous car," in *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No.99CH36289)*, vol. 3, 1999, pp. 1507–1512 vol.3.

[10] T. Lassa, "The Beginning of the End of Driving." [Online]. Available: http://www.motortrend.com/news/the-beginning-of-the-end-of-driving/

[11] E. T. P. on Smart Systems Integration (EPoSS), "European Roadmap Smart Systems for Automated Driving." [Online]. Available: https://www.smart-systems-integration.org/public

[12] J. Reynolds, "Cruising into the future." [Online]. Available: http://www.telegraph.co.uk/motoring/4750544/Cruising-into-the-future.html

[13] "How the first "driverless car" was invented in Britain in 1960." [Online]. Available: https://uk.news.yahoo.com/how-the-first--driverless-car--was-invented-in-britain-in-1960-093127757.html#TAIMeVa

[14] “1960 Citroen DS19 - "Driverless car".” [Online]. Available: https://www.flickr.com/photos/homer----simpson/7768062772/

[15] A. Fascioli, A. Broggi, and M. Bertozzi, “ARGO and the MilleMiglia in Automatico Tour,” *IEEE Intelligent Systems*, vol. 14, pp. 55–64, 1999. [Online]. Available: https://www.computer.org/csdl/mags/ex/1999/01/x1055.html

[16] M. Harris, “How Google’s Autonomous Car Passed the First U.S. State Self-Driving Test,” IEEE Spectrum. [Online]. Available: https://spectrum.ieee.org/transportation/advanced-cars/how-googles-autonomous-car-passed-the-first-us-state-selfdriving-test

[17] “Driverless electric vans complete 8,000 mile journey from Italy to China.” [Online]. Available: http://www.dailymail.co.uk/sciencetech/article-1324515/Driverless-vans-8-000-mile-test-drive-Italy-China.html

[18] D. Bartz, “Autonomous Cars Will Make Us Safer.” [Online]. Available: https://www.wired.com/2009/11/autonomous-cars/

[19] N. Hicks, “Nebraska tested driverless car technology 60 years ago.” [Online]. Available: http://journalstar.com/news/local/govt-and-politics/nebraska-tested-driverless-car-technology-years-ago/article_a702fab9-cac3-5a6e-a95c-9b597fdab078.html

[20] E. D. Dickmanns, *Dynamic Vision for Perception and Control of Motion.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007. [Online]. Available: http://www.springer.com/us/book/9781846286377

[21] T. Jochem, D. Pomerleau, B. Kumar, and J. Armstrong, “PANS: A Portable Navigation Platform.” [Online]. Available: http://www.cs.cmu.edu/~tjochem/nhaa/navlab5_details.html

[22] D. Pomerleau and T. Jochem, "Look, Ma, No Hands." [Online]. Available: https://www.cmu.edu/news/stories/archives/2015/july/look-ma-no-hands.html

[23] C. Albanesius, "Google Car: Not the First Self-Driving Vehicle." [Online]. Available: https://www.pcmag.com/article2/0,2817,2370598,00.asp

[24] A. Broggi, "The ARGO Project," PDF. [Online]. Available: http://www.argo.ce.unipr.it/ARGO/english/flyer_en.pdf

[25] D. Dudley, "The Driverless Car Is (Almost) Here." [Online]. Available: http://www.aarp.org/home-family/personal-technology/info-2014/google-self-driving-car.html

[26] S. Thrun, "Toward Robotic Cars," *Commun. ACM*, vol. 53, no. 4, pp. 99–106, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721679

[27] D. Neil, "Who's Behind the Wheel? Nobody." [Online]. Available: https://www.wsj.com/articles/SB10000872396390443524904577651552635911824

[28] J. Carfrae, "An automated adventure at the wheel of a driverless BMW." [Online]. Available: https://www.thenational.ae/business/an-automated-adventure-at-the-wheel-of-a-driverless-bmw-1.371963

[29] Audi, "Autonomous Audi TT Pikes Peak Self-Driving Test Car." [Online]. Available: https://youtu.be/IFwIlflmk2Y

[30] DesignBoom, "EN-V electric networked car concept by GM begins pilot testing." [Online]. Available: https://www.designboom.com/technology/en-v-electric-networked-car-concept-by-gm-begins-pilot-testing/

[31] A. Broggi, "PROUD Car Test 2013." [Online]. Available: http://vislab.it/proud/

[32] D. P. Howley, “The Race to Build Self-Driving Cars.” [Online]. Available: https://www.laptopmag.com/articles/high-tech-cars-go-mainstream-self-driving-in-car-radar-more

[33] “This is Tesla’s D: an all-wheel-drive Model S with eyes on the road.” [Online]. Available: https://www.theverge.com/2014/10/9/6955357/this-is-tesla-s-d-an-all-wheel-drive-car-with-eyes-on-the-road

[34] A. M. Kesslermarch, “Elon Musk Says Self-Driving Tesla Cars Will Be in the U.S. by Summer.” [Online]. Available: https://www.nytimes.com/2015/03/20/business/elon-musk-says-self-driving-tesla-cars-will-be-in-the-us-by-summer.html?hpw&rref=automobiles&action=click&pgtype=Homepage&module=well-region&region=bottom-well&WT.nav=bottom-well&_r=0

[35] S. Abuelsamid, “Tesla Autopilot Fatality Shows Why Lidar And V2V Will Be Necessary For Autonomous Cars.” [Online]. Available: https://www.forbes.com/sites/samabuelsamid/2016/07/01/first-tesla-autopilot-fatality-demonstrates-why-lidar-and-v2v-probably-will-be-necessary/#453a1f302d91

[36] T. Stevens, “Inside Volvo’s self-driving car: Improving driver safety without the driver.” [Online]. Available: https://www.cnet.com/roadshow/news/a-ride-in-volvos-autonomous-car-how-the-next-step-in-driver-safety-requires-replacing-the-d

[37] C. Ziegler, “Volvo will run a public test of self-driving cars with 100 real people in 2017.” [Online]. Available: https://www.theverge.com/2015/2/23/8091455/volvo-drive-me-self-driving-car-test-2017

[38] K. Korosec, “Volvo Expands Its Self-Driving Car Experiment to China.” [Online]. Available: http://fortune.com/2016/04/06/volvo-self-driving-china/

[39] J. Titcomb, "Google blames careless humans after first driverless car injury." [Online]. Available: http://www.telegraph.co.uk/technology/google/11745772/Google-blames-careless-humans-after-first-driverless-car-injury.html

[40] D. Yadron and D. Tynan, "Tesla driver dies in first fatal crash while using autopilot mode." [Online]. Available: https://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk

[41] "Self-Driving Tesla Involved in Fatal Crash." [Online]. Available: https://www.nytimes.com/2016/07/01/business/self-driving-tesla-fatal-crash-investigation.html

[42] NHTSA, "ODI Resume - Investigation: PE 16-007." [Online]. Available: https://www-odi.nhtsa.dot.gov/acms/cs/jaxrs/download/doc/UCM530776/INOA-PE16007-7080.PDF

[43] J. M. Watts, "World's First Self-Driving Taxis Hit the Road in Singapore - Singapore's nuTonomy debuts autonomous cabs, beating the likes of U.S. tech giants Uber and Google." [Online]. Available: https://www.wsj.com/articles/worlds-first-self-driving-taxis-hit-the-road-in-singapore-1472102747

[44] SAE, "Automated Driving," PDF. [Online]. Available: http://www.sae.org/misc/pdfs/automated_driving.pdf

[45] J. Collier, "What is Autonomy?" 2002. [Online]. Available: http://cogprints.org/2289/

[46] M. McAleer, "Audi's self-driving A8: drivers can watch YouTube or check emails at 60km/h." [Online]. Available: https://www.irishtimes.com/life-and-style/motors/audi-s-self-driving-a8-drivers-can-watch-youtube-or-check-emails-at-60km-h-1.3150496

[47] B. Zhang, "Autonomous Cars Could Save The Save The US $1.3 Trillion Dollars A Year," Infographic. [Online]. Available: http://www.businessinsider.com/morgan-stanley-autonomous-cars-trillion-dollars-2014-9?IR=T

[48] J. Miller, "Self-Driving Car Technology Benefits, Potential Risks, and Solutions." [Online]. Available: http://www.theenergycollective.com/jemiller_ep/464721/self-driving-car-technology-s-benefits-potential-risks-and-solutions

[49] R. Whitwam, "How Google's self-driving cars detect and avoid obstacles." [Online]. Available: http://www.extremetech.com/extreme/189486-how-googles-self-driving-cars-detect-and-avoid-obstacles

[50] T. Cowen, "Can I See Your License, Registration and C.P.U.?" [Online]. Available: http://www.nytimes.com/2011/05/29/business/economy/29view.html

[51] T. Gosman, "Along for the ride: How driverless cars can become commonplace." [Online]. Available: http://www.brandunion.com/insight/2016-07-24/5689/along-for-the-ride-how-driverless-cars-can-become-commonplacerl

[52] P. Stenquist, "In Self-Driving Cars, a Potential Lifeline for the Disable." [Online]. Available: https://www.nytimes.com/2014/11/09/automobiles/in-self-driving-cars-a-potential-lifeline-for-the-disabled.html?_r=0

[53] D. Curry, "Will elderly and disabled gain most from autonomous cars?" [Online]. Available: http://readwrite.com/2016/04/22/autonomous-cars-elderly-disabled-drivers-google-tl4/

[54] J. M. Anderson, N. Kalra, K. D. Stanley, P. Sorensen, C. Samaras, and O. A. Oluwatola, *Autonomous Vehicle Technology: A Guide*

for Policymakers. RAND Corporation, 2014. [Online]. Available: https://www.rand.org/pubs/research_reports/RR443-2.html

[55] T. Simonite, "Self-Driving Motorhome: RV of the Future?" [Online]. Available: http://vogeltalksrving.com/2014/11/self-driving-motorhome-rv-of-the-future/

[56] "Get ready for automated cars." [Online]. Available: http://www.chron.com/opinion/editorials/article/Get-ready-for-automated-cars-3857472.php

[57] "Data Shows Google's Robot Cars Are Smoother, Safer Drivers Than You or I." [Online]. Available: https://www.technologyreview.com/s/520746/data-shows-googles-robot-cars-are-smoother-safer-drivers-than-you-or-i/

[58] E. Ackermann, "Study: Intelligent Cars Could Boost Highway Capacity by 273% ." [Online]. Available: https://spectrum.ieee.org/automaton/robotics/artificial-intelligence/intelligent-cars-could-boost-highway-capacity-by-273

[59] M. Ramsey, "Self-Driving Cars Could Cut Down on Accidents, Study Says." [Online]. Available: https://www.wsj.com/articles/self-driving-cars-could-cut-down-on-accidents-study-says-1425567905

[60] J. Piper, "Self-Driving Cars Could Cut Greenhouse Gas Pollution." [Online]. Available: https://www.scientificamerican.com/article/self-driving-cars-could-cut-greenhouse-gas-pollution/

[61] R. Adhikary, "Feds Put AI in the Driver's Seat." [Online]. Available: http://www.technewsworld.com/story/83102.html?rss=1

[62] G. Nichols, "NHTSA chief takes conservative view on autonomous vehicles." [Online]. Available: http://www.zdnet.com/article/nhtsa-chief-takes-conservative-view-on-autonomous-vehicles/

[63] "New Allstate Survey Shows Americans Think They Are Great Drivers - Habits Tell a Different Story." [Online]. Available: http://www.prnewswire.com/news-releases/new-allstate-survey-shows-americans-think-they-are-great-drivers---habits-tell-a-different-st html

[64] "Remembering When Driverless Elevators Drew Skepticism." [Online]. Available: http://www.npr.org/2015/07/31/427990392/remembering-when-driverless-elevators-drew-skepticism

[65] "Will Regulators Allow Self-Driving Cars In A Few Years?" [Online]. Available: https://www.forbes.com/sites/quora/2013/09/24/will-regulators-allow-self-driving-cars-in-a-few-years/#4a2a5e105c9d

[66] C. Newton, "Reliance on autopilot is now the biggest threat to flight safety, study says." [Online]. Available: https://www.theverge.com/2013/11/18/5120270/reliance-on-autopilot-is-now-the-biggest-threat-to-flight-safety

[67] P. Lin, "The Ethics of Autonomous Cars." [Online]. Available: https://www.theatlantic.com/technology/archive/2013/10/the-ethics-of-autonomous-cars/280360/

[68] T. Worstall, "When Should Your Driverless Car From Google Be Allowed To Kill You?" [Online]. Available: https://www.forbes.com/sites/timworstall/2014/06/18/when-should-your-driverless-car-from-google-be-allowed-to-kill-you/#25f75555fa5brl

[69] A. Skulmowski, A. Bunge, K. Kaspar, and G. Pipa, "Forced-choice decision-making in modified trolley dilemma situations: a virtual reality and eye tracking study," *Frontiers in Behavioral Neuroscience*, vol. 8, p. 426, 2014. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnbeh.2014.00426

[70] L. Gomes, “Hidden Obstacles for Google’s Self-Driving Cars.” [Online]. Available: https://www.technologyreview.com/s/530276/hidden-obstacles-for-googles-self-driving-cars/

[71] “Carlo van de Weijer on real intelligence,” YouTube. [Online]. Available: https://www.youtube.com/watch?v=I6sWZMR9OZM&feature=youtu.be&t=32s

[72] “Hackers find ways to hijack car computers and take control.” [Online]. Available: http://business.financialpost.com/technology/hackers-find-ways-to-hijack-car-computers-and-take-control

[73] P. E. Ross, “A Cloud-Connected Car Is a Hackable Car, Worries Microsoft.” [Online]. Available: https://spectrum.ieee.org/tech-talk/transportation/advanced-cars/a-connected-car-is-a-hackable-car

[74] R. Moore-Colyer, “Driverless cars face cyber security, skills and safety challenges.” [Online]. Available: https://www.v3.co.uk/v3-uk/analysis/2394924/driverless-cars-face-cyber-security-skills-and-safety-challenges

[75] J. Petit and S. E. Shladover, “Potential cyberattacks on automated vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 546–556, April 2015.

[76] “The Challenges Facing Autonomous Vehicles.” [Online]. Available: http://auto-sens.com/the-challenges-facing-autonomous-vehicles/

[77] N. Zhou, “Volvo admits its self-driving cars are confused by kangaroos.” [Online]. Available: https://www.theguardian.com/technology/2017/jul/01/volvo-admits-its-self-driving-cars-are-confused-by-kangaroos

[78] J. Boyd, “Mitsubishi Electric joins race to make maps for self-drive cars.” [Online]. Available: http://www.atimes.com/article/mitsubishi-joins-race-build-maps-cars-not-drivers/

[79] G. Garvin, "Automakers say self-driving cars are on the horizon." [Online]. Available: http://www.tampabay.com/news/business/autos/automakers-say-self-driving-cars-are-on-the-horizon/2171386

[80] E. Badger, "5 confounding questions that hold the key to the future of driverless cars." [Online]. Available: https://www.washingtonpost.com/news/wonk/wp/2015/01/15/5-confounding-questions-that-hold-the-key-to-the-future-of-driverless-cars/?utm_term=.3e02e99f35fe

[81] M. Ufberg, "WHOOPS: THE SELF-DRIVING TESLA MAY MAKE US LOVE URBAN SPRAWL AGAIN." [Online]. Available: https://www.wired.com/2014/10/tesla-self-driving-car-sprawl/

[82] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[83] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003.

[84] W. S. Sarle, "Neural Networks and Statistical Models," 1994.

[85] L. Deng and D. Yu, "Deep Learning: Methods and Applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014. [Online]. Available: http://dx.doi.org/10.1561/2000000039

[86] Y. Bengio, "Learning Deep Architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1561/2200000006

[87] G. E. Hinton, "Deep belief networks." [Online]. Available: http://www.scholarpedia.org/article/Deep_belief_networks

[88] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Trans. Pattern Anal. Mach.*

Intell., vol. 35, no. 8, pp. 1798–1828, Aug. 2013. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2013.50

[89] A. G. Ivakhnenko, V. Lapa, and PURDUE UNIV LAFAYETTE IND SCHOOL OF ELECTRICAL ENGINEERING, *Cybernetic Predicting Devices*, ser. JPRS 37, 803. Purdue University School of Electrical Engineering, 1965. [Online]. Available: https://books.google.it/books?id=l38DHQAACAAJ

[90] A. G. Ivakhnenko, “Polynomial Theory of Complex Systems,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-1, no. 4, pp. 364–378, Oct 1971.

[91] J. Schmidhuber, “Learning complex, extended sequences using the principle of history compression,” *Neural Computation*, vol. 4, no. 2, pp. 234–242, March 1992. [Online]. Available: http://ieeexplore.ieee.org/document/6795261/

[92] ——, “Deep Learning in Neural Networks: An Overview,” *CoRR*, vol. abs/1404.7828, 2014. [Online]. Available: http://arxiv.org/abs/1404.7828

[93] C. Szegedy, A. Toshev, and D. Erhan, “Deep Neural Networks for Object Detection,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2553–2561. [Online]. Available: http://papers.nips.cc/paper/5207-deep-neural-networks-for-object-detection.pdf

[94] A. Karpathy, “Convolutional Neural Networks for Visual Recognition,” http://cs231n.github.io/convolutional-networks/, accessed: 22-05-17.

[95] A. Krizhevsky, “The cifar-10 dataset,” https://www.cs.toronto.edu/~kriz/cifar.html, accessed: 22-05-17.

[96] U. Karn, "An Intuitive Explanation of Convolutional Neural Networks," https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/, accessed: 22-05-17.

[97] DeepLearning.TV, "Recurrent Neural Networks - Ep. 9 (Deep Learning SIMPLIFIED)," https://www.youtube.com/watch?v=_aCuOwF1ZjU, accessed: 22-05-17.

[98] Nervana, "(2) Recurrent Neural Networks," https://www.youtube.com/watch?v=Ukgii7Yd_cU, accessed: 22-05-17.

[99] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A Brief Survey of Deep Reinforcement Learning," *CoRR*, vol. abs/1708.05866, 2017. [Online]. Available: http://arxiv.org/abs/1708.05866

[100] T. Huang, "Computer Vision: Evolution And Promise," 1996. [Online]. Available: https://cds.cern.ch/record/400313

[101] S. Papert and M. I. of Technology. Artificial Intelligence Laboratory, *The Summer Vision Project*, ser. AI memo. Massachusetts Institute of Technology, Project MAC, 1966. [Online]. Available: https://books.google.it/books?id=qOh7NwAACAAJ

[102] D. Marr, *Vision: A Compuctational Investigation into the Human Representation and Processing of Visual Information.* New York, NY, USA: Henry Holt and Co., Inc., 1982.

[103] A. Milella, G. Reina, and M. Foglia, "Computer vision technology for agricultural robotics," *Sensor Review*, vol. 26, no. 4, pp. 290–300, 2006. [Online]. Available: https://doi.org/10.1108/02602280610692006

[104] G. Muscato, M. Prestifilippo, N. Abbate, and I. Rizzuto, "A prototype of an orange picking robot: past history, the new robot and experimental results," *Industrial Robot: An International*

Journal, vol. 32, no. 2, pp. 128–138, 2005. [Online]. Available: https://doi.org/10.1108/01439910510582255

[105] V. Lepetit, “On computer vision for augmented reality,” in *2008 International Symposium on Ubiquitous Virtual Reality*, July 2008, pp. 13–16.

[106] M. Burge and W. Burger, “Ear biometrics in computer vision,” in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol. 2, 2000, pp. 822–826 vol.2.

[107] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “Deepdriving: Learning affordance for direct perception in autonomous driving,” in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV ’15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 2722–2730. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2015.312

[108] A. Jazayeri, H. Cai, J. Y. Zheng, and M. Tuceryan, “Vehicle detection and tracking in car video based on motion model,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 583–595, June 2011.

[109] K. Öfjäll, M. Felsberg, and A. Robinson, “Visual autonomous road following by symbiotic online learning,” in *2016 IEEE Intelligent Vehicles Symposium (IV)*, June 2016, pp. 136–143.

[110] M. Bertozzi, A. Broggi, and A. Fascioli, “Stereo inverse perspective mapping: theory and applications,” *Image and Vision Computing*, vol. 16, no. 8, pp. 585 – 590, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885697000930

[111] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Commun.*

ACM, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386

[112] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[113] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[114] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[115] C. Sammut, *Behavioral Cloning.* Boston, MA: Springer US, 2010, pp. 93–97. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_69

[116] M. Riedmiller, *Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 317–328. [Online]. Available: https://doi.org/10.1007/11564096_32

[117] DeepMind, "Publications of DeepMind in reinforcement learning," https://deepmind.com/research/publications/?author=D+Silver, accessed: 01-06-17.

[118] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[119] MATLAB, "MATLAB pricing." [Online]. Available: https://it.mathworks.com/pricing-licensing.html

[120] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Realtime computer vision with opencv," *Queue*, vol. 10, no. 4, pp. 40:40–40:56, Apr. 2012. [Online]. Available: http://doi.acm.org/10.1145/2181796.2206309

[121] "Bjarne Stroustrup - The Essence of C++," accessed: 04/11/17. [Online]. Available: https://www.youtube.com/watch?v=86xWVb4XIyE

[122] B. Stroustrup, "C++ Applications," accessed: 04/11/17. [Online]. Available: http://www.stroustrup.com/applications.html

[123] "eblearn," accessed: 04/11/17. [Online]. Available: https://sourceforge.net/projects/eblearn/

[124] "Tensorflow," accessed: 04/11/17. [Online]. Available: https://www.tensorflow.org/

[125] "Caffe," accessed: 04/11/17. [Online]. Available: http://caffe.berkeleyvision.org/

[126] S. McConnell, *Code Complete*, ser. DV-Professional. Microsoft Press, 2009. [Online]. Available: https://books.google.it/books?id=3JfE7TGUwvgC

[127] "History and License," accessed: 04/11/17. [Online]. Available: https://docs.python.org/3/license.html

[128] "NumPy," accessed: 04/11/17. [Online]. Available: http://www.numpy.org/

[129] "SciPy," accessed: 04/11/17. [Online]. Available: https://www.scipy.org/

[130] "scikit-learn," accessed: 04/11/17. [Online]. Available: http://scikit-learn.org/stable/

[131] “Matplotlib,” accessed: 04/11/17. [Online]. Available: https://matplotlib.org/

[132] “Keras,” accessed: 04/11/17. [Online]. Available: https://keras.io/

[133] “Theano,” accessed: 04/11/17. [Online]. Available: http://deeplearning.net/software/theano/

[134] “CNTK,” accessed: 04/11/17. [Online]. Available: https://cntk.ai

[135] “PyTorch,” accessed: 04/11/17. [Online]. Available: http://pytorch.org/

[136] “Should I use Python 2 or Python 3 for my development activity?” accessed: 05/11/17. [Online]. Available: https://wiki.python.org/moin/Python2orPython3

[137] “Deep Learning Libraries,” accessed: 05/11/2017. [Online]. Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/deep_learning_libraries.html

[138] “Benchmarking State-of-the-Art Deep Learning Software Tools,” accessed: 05/11/2017. [Online]. Available: https://github.com/hclhkbu/dlbench

[139] S. Shi, Q. Wang, P. Xu, and X. Chu, “Benchmarking state-of-the-art deep learning software tools,” *CoRR*, vol. abs/1608.07249, 2016. [Online]. Available: http://arxiv.org/abs/1608.07249

[140] “MILA and the future of Theano,” accessed: 30/09/17. [Online]. Available: https://groups.google.com/forum/#!msg/theano-users/7Poq8BZutbY/rNCIfvAEAwAJ

[141] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.

[142] “Caffe2,” accessed: 05/11/2017. [Online]. Available: https://caffe2.ai/

[143] J. Q. Candela, "Facebook and Microsoft introduce new open ecosystem for interchangeable AI frameworks," Accessed: 05/11/2017. [Online]. Available: https://research.fb.com/facebook-and-microsoft-introduce-new-open-ecosystem-for-interchangeable-ai-frameworks/

[144] "Network Description Language," accessed: 05/11/2017. [Online]. Available: https://github.com/Microsoft/CNTK/blob/master/Documentation/Documents/Network%20Description%20Language.md

[145] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[146] Accessed: 05/11/2017. [Online]. Available: https://github.com/search?q=stars:%3E1&s=stars&type=Repositories

[147] Accessed: 05/11/2017. [Online]. Available: https://docs.microsoft.com/en-us/cognitive-toolkit/Using-CNTK-with-Keras

[148] D. Yu, A. Eversole, M. Seltzer, K. Yao, O. Kuchaiev, Y. Zhang, F. Seide, Z. Huang, B. Guenter, H. Wang, J. Droppo, G. Zweig, C. Rossbach, J. Gao, A. Stolcke, J. Currey, M. Slaney, G. Chen, A. Agarwal, C. Basoglu, M. Padmilac, A. Kamenev, V. Ivanov, S. Cypher, H. Parthasarathi, B. Mitra, B. Peng, and X. Huang, "An introduction to computational networks and the computational network toolkit," Tech. Rep., October 2014. [Online].

Available: https://www.microsoft.com/en-us/research/publication/an-introduction-to-computational-networks-and-the-computational-network-toolkit/

[149] "Tensoflow documentation," accessed: 05/11/2017. [Online]. Available: https://www.tensorflow.org/api_docs/

[150] R. Thomas, "Big deep learning news: Google Tensorflow chooses Keras," accessed: 05/11/2017. [Online]. Available: http://www.fast.ai/2017/01/03/keras/

[151] M. Woolf, "Benchmarking CNTK on Keras: is it Better at Deep Learning than TensorFlow?" accessed: 05/11/2017. [Online]. Available: http://minimaxir.com/2017/06/keras-cntk/

[152] "Keras backend benchmark," accessed: 05/11/2017. [Online]. Available: https://github.com/szilard/benchm-dl/blob/master/keras_backend.md

[153] "CUDA in MATLAB," accessed: 04/11/17. [Online]. Available: https://developer.nvidia.com/matlab-cuda

[154] "cnn-benchmarks," accessed: 05/11/2017. [Online]. Available: https://github.com/jcjohnson/cnn-benchmarks/blob/master/README.md

[155] MIT, "Deeptesla." [Online]. Available: https://selfdrivingcars.mit.edu/deeptesla/

[156] Udacity, "Udacity official website." [Online]. Available: https://www.udacity.com/

[157] ——, "Self-driving car nanodegree." [Online]. Available: https://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd013

[158] B. Wymann and E. Espié, "TORCS official website," http://torcs.sourceforge.net/, accessed: 24-05-17.

[159] D. Loiacono, L. Cardamone, and P. L. Lanzi, “Simulated car racing championship: Competition software manual,” *CoRR*, vol. abs/1304.1672, 2013. [Online]. Available: http://arxiv.org/abs/1304.1672

[160] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[161] L. Santonastasi, “Audrey summaries data.” [Online]. Available: https://github.com/lucosanta/audrey-summaries-data

[162] OpenCV, “Camera calibration and 3d reconstruction,” accessed: 12/11/2017. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[163] Quora, “What are differences between update rules like AdaDelta, RMSProp, AdaGrad and AdaM?” https://www.quora.com/What-are-differences-between-update-rules-like-AdaDelta-RMSProp-AdaGrad-and-AdaM, accessed: 24-05-17.

[164] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2021068

[165] A. Coates, B. Carpenter, C. Case, S. Satheesh, B. Suresh, T. Wang, D. J. Wu, and A. Y. Ng, “Text detection and character recognition in scene images with unsupervised feature learning,” in *2011 International Conference on Document Analysis and Recognition*, Sept 2011, pp. 440–445.

[166] M. D. Zeiler, “ADADELTA: an adaptive learning rate method,” *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[167] G. Hinton, "Overview of mini-batch gradient descent." [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[168] Wikipedia, "RMSProp," https://en.wikipedia.org/wiki/Stochastic_gradient_descent#RMSProp, accessed: 24-05-17.

[169] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html

[170] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *CoRR*, vol. abs/1411.4389, 2014. [Online]. Available: http://arxiv.org/abs/1411.4389

[171] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," *CoRR*, vol. abs/1506.04214, 2015. [Online]. Available: http://arxiv.org/abs/1506.04214

[172] B. Lau, "Using Keras and Deep Deterministic Policy Gradient to play TORCS," https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html, accessed: 16-05-17.

[173] A. Juliani, "Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C)," https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c accessed: 17-05-17.

[174] K. Murphy, “A brief introduction to reinforcement learning,” http://www.cs.ubc.ca/~murphyk/Bayes/pomdp.html, accessed: 01-06-17.

[175] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, “Population Based Training of Neural Networks,” Nov. 2017. [Online]. Available: http://arxiv.org/abs/1711.09846

[176] D. Silver, “Headlines from Google, Uber, and Waymo,” 2017. [Online]. Available: https://medium.com/self-driving-cars/headlines-from-google-uber-and-waymo-d4bee7325291

[177] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 3859–3869. [Online]. Available: http://papers.nips.cc/paper/6975-dynamic-routing-between-capsules

[178] P. Foot, “The problem of abortion and the doctrine of double effect,” *Oxford Review*, vol. 5, pp. 5–15, 1967.

[179] J. J. Thomson, “Killing, Letting Die, and the Trolley Problem,” *The Monist*, vol. 59, no. 2, pp. 204–217, 1976.