Politecnico di Torino

POLITECNICO DI TORINO

Master's degree
in MECHATRONIC ENGINEERING

Master Thesis

Development of on-board algorithms to support the navigation of high-speed planetary rovers

Supervisor:
Prof.ssa Sabrina Corpino

Co-Supervisors:
Prof. Fabrizio Stesina
Guglielmo Daddi

Candidate:
Nicolaus Notaristefano

A.A. 2020/2021

Abstract

The aim of the Master Thesis is to develop, study and validate a path planning algorithm that can support the navigation of rovers for planetary exploration. In particular, the objective is to develop a global path planning algorithm that could be integrated in the rover auto-nav system in order to improve the autonomy and velocity of the planetary rover.

Global path planning using grid-based model of the environment is a well-known problem in AI, planning and robotics with a variety of methods and algorithms proposed so far.

This work presents a deep-learning approach to the path planning problem. In particular, grid maps, containing information about the traversability of the terrain, are suitable input to modern neural networks, such as convolutional neural networks.

The thesis proposes a modern approach, based on the recent advances in deep learning, in order to treat the path planning problem as an Image-to-Image translation problem. Specifically, this thesis applies a Generative Adversarial Network (GAN) and reports the main, obtained results.

The GAN network is trained using a wide enough dataset, which contains a set of grid maps, start and goal nodes, and feasible set of paths generated by any well-known algorithm. This kind of approach allows to generalize the path planning problem. The main advantage is that we can use any algorithm, from the simpler to the more complex one, for the generation of training dataset and in such a way the deep learning architecture, trained properly, replaces the necessity to execute the path-planning on-board classic algorithm using a mathematical model that works only with tensor operations.

In our tests, in order to generate the paths for the training dataset the well-known and effective algorithm A* is used.

The network used, for our application, is a modified version of the Pix2Pix model, introduced by Philip Isola et al. in the paper “Image-to-Image Translation with

Conditional Adversarial Networks" (2018) which is a revisitation of the common Conditional Adversarial Network.

Using this kind of architecture allows us to influence the generation of the image in such a way that is a plausible translation of the input images, which means having a correct path generated given the generator inputs: environment map and start and goal nodes.

We conduct a number of experiments in order to validate the effectiveness of the proposed method, and the results demonstrate how the model adapts well to changes in the position of obstacles in the environment.

This work is a part of a research project called SINAV, sponsored by Italian Space Agency. The objective of SINAV is the development of High-Speed Autonomous Navigation Systems for future missions of robotic planetary exploration.

Contents

Abbreviations

ABC	A* with bounded cost
ACE	Approximate Clearance Evaluation
AI	Artificial Intelligence
ANN	Artificial Neural Network
BFS	Breadth-First Search
BN	Batch Normalization
CFA	Cumulative Fractional Area
CGAN	Conditional Generative Adversarial Network
CIFAR	Canadian Institute For Advanced Research
CNN	Convolutional Neural Network
DB-CNN	Double Branch Convolutional Neural Network
DCGAN	Deep Convolutional Generative Adversarial Networks
DCNN	Deep Convolutional Neural Network
FID	Fréchet Inception Distance
FM	Fast Marching
FM2	Fast Marching Square
FM2*	Heuristic Fast Marching Square
FMM	Fast Marching Method
FMVF	Fast Marching Vectorial Field
FPU	Floating Point Unit
GAN	Generative Adversarial Networks
GESTALT	Grid-based Estimation of Surface Traversability Applied to Local Terrain
HiRISE	High Resolution Imaging Science Experiment
IS	Inception Score
LPA*	Lifelong Planning A*
LRV	Lunar Roving Vehicle
MDP	Markov Decision Process
MER	Mars Exploration Rover
MNIST	Modified National Institute of Standards and Technology
MSL	Mars Science Laboratory
NaN	Not a Number
NN	Neural Network
OFID	Overall Fréchet Inception Distance

PFID	Particular Fréchet Inception Distance
PRM	Probabilistic Road Maps
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
ROI	Region Of Interest
RRT	Rapidly-exploring Random Trees
SGD	Stochastic Gradient Descent
VAE	Variational Autoencoder
VFM	Voronoi Fast Marching
VIN	Value Iteration Network

Chapter 1

Introduction

Historically, Planetary exploration rovers are initially conceived as surface mobility systems to explore safely and efficiently the planetary surface once the man is there. In this perspective, the mobile systems are needed to assist the astronauts in the day-to-day operations, in order to accomplish many tasks ranging from site preparation, construction, and local transportation. This is the idea of the Apollo Lunar Roving Vehicle (LRV) that allows the Apollo astronauts to extend their range of surface activities.

Naturally, in these decades, the idea behind the rover exploration is muted. Many efforts were done in order to make mobile robots as autonomous as possible. This makes not strictly necessary the presence of astronauts for planetary exploration and allows to explore space environments where humans cannot yet go. We have numerous examples like Mars Exploration Rover (MER) mission involving two Mars rovers, Spirit and Opportunity, Mars Science Laboratory (MSL), involving curiosity rover and Mars 2020, involving Perseverance rover.

Improving the level of autonomy of a mobile robot means that the role of humans, in the loop of controllability, is less and less important. This improves the navigation robot efficiency because the problem of communication delay has a minor impact. It is important to note that the level of autonomy strongly impacts the scientific mission return. In particular, the higher the level of autonomy the faster the rover can reach the selected scientific targets. This allows to minimize the time employed to traverse the terrain from one location to another and to increase the time dedicated to perform scientific measurements.

In order to obtain a detailed planetary exploration, the autonomous mobile robot must be capable to move to different locations for a wider exploration area. It must offer adaptability and flexibility in order to reach specific targets for scientific analysis.

Contrary to the traditional terrestrial mobile robots, in the planetary exploration we have a series of additional constraints which must be considered:

- communication problems caused by extensive signal time-of-flight and limited communication windows
- adverse terrain that is characterized by limited features which makes harder the self-localization
- limited power
- hostile ambient conditions.

These constraints have a significant impact in the design and in the methodologies employed by the planetary rovers.

Specifically, the core of an exploration rover is the autonomous navigation system that permits it to traverse the terrain and reach the specific goal, which is typically a target of scientific interest. The navigation system involves four main processes that allow it to sense the environment and build a feasible strategy in order to safe traverse the terrain and react to eventual unexpected situations:

- perception of the environment
- self-localization with respect to landmarks
- path planning
- path traversal.

The rover, through its perception system, is required to construct a geometric map of the local environment around it and localize itself with respect to visual landmarks. Fig.1 shows the general architecture for the autonomous navigation system of a rover.

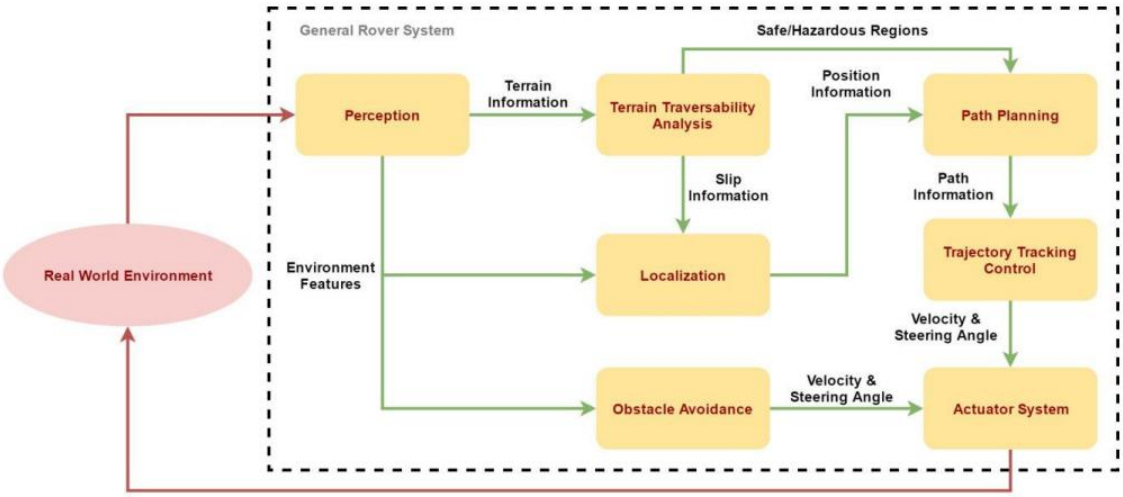


Figure 1 – General architecture of rover navigation system [1]

1.1 Path Planning Problem Statement

The purpose of this section is to properly introduce the problem of path planning presenting all the factors involved in the definition and solution of it.

In order to better explain the problem, we can define:

- the agent (the mobile robot) which is the entity able to perceive the environment through a priori or extrapolated information
- the state as the information which defines the condition of the agent inside the environment
- the action as the operation performed to bring the agent from current state to the next one (a simple state can be the agent coordinates in the environment)

The path planning problem consists in finding a feasible path that allows the agent to reach a given state. Specifically, in a typical approach, given a set of possible feasible paths the path planning aims to find the optimal path which minimizes a specific cost function. So, the latter represents the optimality criterion on the basis of which we operate the choice.

A typical criterion is the obstacle avoidance. The primary objective of a path planning algorithm is to find a collision-free path which allows the rover to avoid all obstacles, considering their shape, size and orientation. Naturally, the path should be as short as possible, and the smoothness of the path should meet dynamics constraints of the mobile robot. Other important factors in mission planning are time, operational

constraints, optimization of the energy resources and the limited on-board computational resources.

The strategy adopted to solve the path planning problem strongly depends on the environment properties. Specifically, the environment could be either completely known (static or dynamic), partially known, unknown. In the first case, the agent has a prior knowledge of the environment, so the path planner can evaluate an initial path to follow, monitoring possible dynamic changes of the environment. If there are changes, as dynamic obstacles, it updates the path consequently. In the second case, the agent extracts environment features using its perception system and it takes decision based on the obtained information. In the third case, having a partial information about the environment, the path planner generates an initial path which is corrected during the motion of the agent using the detailed information extrapolated by the perception system.

Moreover, in the path planning process, depending on the quantity of information that we have about the environment path planning can be divided into two categories: global and local path planning. The first solves the path planning problem using a larger, almost complete, environmental information. In this case, the planner produces a complete path from start location to final location. The second solves the problem while the mobile robot is moving, taking local data of the environment from the perception sensors and usually is performed in unknown or partially known environments.

The path planner can also use both approaches, using the global planner to generate a first raw path and the local planner to meet dynamic and kinematic constraints.

In this work of thesis, we focalize our attention on the global path planning strategy which is the approach chosen for solving the path planning problem.

Currently, the global path planning approach is limited by the fact that we have not a complete knowledge of the environment. In particular, we have only partial global information which are given from satellite images. Considering, HiRISE (High Resolution Imaging Science Experiment), which is an acquisition images device mounted on the Mars Reconnaissance Orbiter, it gives us sensing data for Mars including images with 25 cm/pixel resolution. Although this dataset contains high resolution images it gives limited terrain information about the current conditions at the

surface. Moreover, small rocks remain undetected, and they could be critical hazards to the rovers.

In future, this limitation could be overcome using additional devices, which are at different levels (orbit and flight devices) that allow to explore and capture a larger region, which gives a sufficient detailed global information to use a global path planner. These devices are conceived to work in close cooperation with the exploration rover allowing to have a detailed and always updated global information at different levels. A practical example is Ingenuity, which is a small robotic helicopter which operates on Mars as part of NASA Mars 2020 mission and cooperates with Perseverance rover [2] (Fig. 2). Fig.2 shown rover and copter Ingenuity cooperation whereas Fig. 3 shown a possible architecture of global path planning in the future exploration missions. The less uncertainty in the global path planning strategy can significantly improve the level of autonomy and the rover speed.

Figure 2 - Rover/copter cooperation for Mars exploration [2]

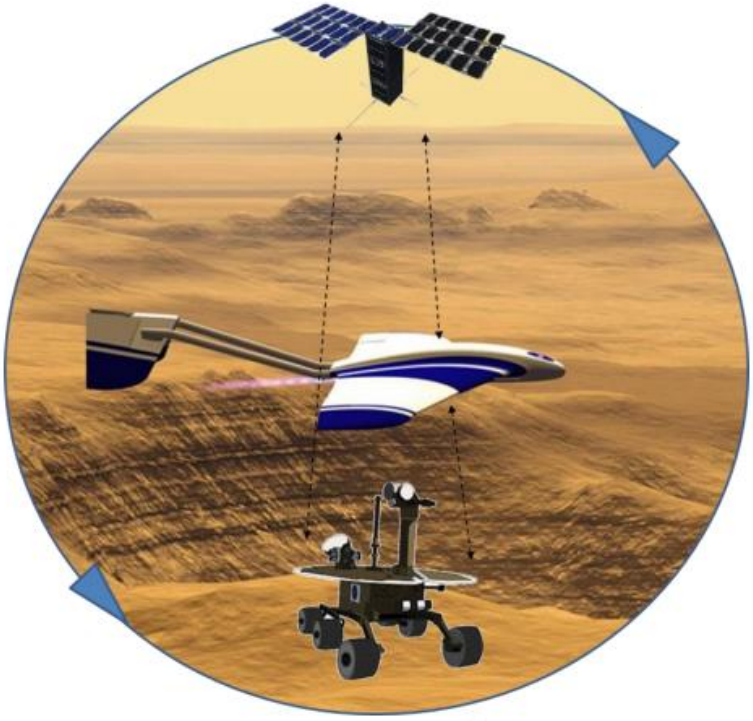


Figure 3 - Global path planning architecture

1.2 Methods and solutions adopted

In this scenario of application, we propose a new solution for global path planning which is based on the new recent developments in deep neural networks. The proposed algorithm treats the path planning problem as an image-to-image translation problem. In particular, given the image representation of the environment, it evaluates a plausible translation of the environment which corresponds to a feasible path to follow. In order to perform a plausible translation, the algorithm required a learning process which aims to minimize a certain loss function that classify how good the network is to relate input (map environment) and output (path).

In order to train the network, we use a learning dataset which contains a set of environments, start and goal locations, and corresponding feasible paths which have been computed using a well-known path planning algorithm. So, the proposed approach is based on imitation learning. The neural network tries to find a solution path of an unknown environment imitating the behaviour of a well-known algorithm. The solution

is an approximated and comparable trajectory with respect to the one that the well-known algorithm would have generated.

This approach is justified by the fact that the learning-based method presented here tries to achieve a behaviour which is effective for exploration. In particular, the proposed solution, after the offline training phase, aims to solve a path planning problem using as information only the real-time acquired global information about the environment. This allows to instantly find the path without problem of convergence time or local minima caused by the complexity of the environment and is advantageous if we have limited computational resources because there is no necessity to have an on-board map component which updates and store maps in the time. Potentially, the used deep learning approach can imitate the behaviour of any well-known algorithm, from the simple to the complex one, without having problem of computationally difficult and local minima.

In our specific case, the proposed algorithm works with a grid map model of the environment, which can be extrapolated, for instance, by the information acquired by the global vision system. The map could be obtained with algorithms capable to represent and learn deep features (shape and location of the obstacles). The implementation is not covered by the work of thesis.

Specifically, our contributions include:

- Proposing as deep learning network a generative adversarial network (GAN) capable to imitate the behaviour of a well-known algorithm that in our specific case is the A* algorithm
- Designing a GAN architecture which is able to predict a feasible path for the planning
- Applying and validating the effectiveness of the proposed method for the solution of path planning problem

1.3 Thesis structure

The work of thesis is organized as follows. In the Chapter 2 we review a wide range of path planning techniques trying to define what is the state-of-the-art. In the Chapter 3 we introduce the theory of the GAN networks and their application in image-to-image

tasks. In Chapter 4 we introduce the proposed GAN architecture used for the path prediction. In the Chapter 5, we apply the GAN network and conduct a series of simulation experiments to show and validate the performance of the GAN-based path planning algorithm. In the last Chapter, we talk about the thesis conclusions and possible future works.

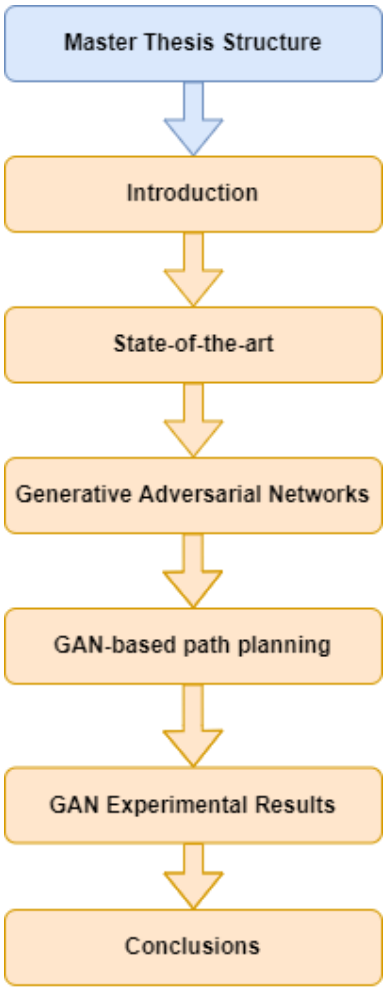


Figure 4 - Master Thesis Structure

Chapter 2

State of the art

The development of Autonomous Path Planning Algorithms is fundamental for the exploration of planetary rovers. This is required by the fact that there are unavoidable communication delays between Earth and other Planets and uncertainty of the environment to explore. So, it becomes impractical to provide real-time control and decision from Earth.

During planetary exploration missions, rovers are required to traverse terrain in order to reach several targets of interest finding optimal paths. The targets are typically location of scientific interest where the rover will acquire and analyse samples. In particular, the path to follow has to be collision free because the surface of the planet contains dangerous area (e.g. rocks and craters), and energy efficient, because the sources of energy are limited [3].

The path planning problem can be divided into two different sub-problems by using a two-level approach [4]. A high-level rough path is provided by the global path planning module. The global path planner finds the optimal path in a larger scale, considering obstacles that the mobile robot, locally, cannot perceive. The path length is the fundamental evaluation metric for the global path because, typically, the shorter the path is the less is the energy consumed. Instead, the local path planner has deep knowledge about local and mobile obstacles and terrain features thanks to on-board perception system. It generates a path that needs to meet dynamics, kinematics and orientation constraints. The combination of the two levels allows the mobile robot to have a high level of automation in order to cover larger distance in a given time period [5].

In this chapter, I will review a wide range of path planning algorithms, giving a clear explanation of what is the state-of-the-art and motivating the choices made to solve the specific path planning problem presented in Chapter 1.

2.1 Local Path Planning

Before the Nasa/JPL Mars 2020 (M2020) mission the state-of-the-art on-board path planner is called GESTALT, which has successfully driven Spirit, Opportunity and Curiosity rovers.

It, based on stereo image pairs captured by the rover's on-board camera, extracts geometric properties of the local 3D environment through a grid-based estimation of surface traversability. In particular, it generates a set of circular arcs of varying curvature to assess the risks of different trajectories. Each arc is evaluated based on three criteria (avoiding hazards, minimizing steering time and reaching the goal). So, for each arc, a vote, based on these 3 criteria, is generated. The arc which minimizes the weighted cost is chosen. This process is repeated until the goal is reached [6].

However, Autonomous Navigation, based on GESTALT, in some situation is not able to reach the goal [5]. More specifically, it frequently fails to find a feasible path when terrain has at least 10% Cumulative Fractional Area (CFA) which is a measure of rock abundance on Mars. It has an excessive conservatism in collision-checking which results in reduced efficiency or inability to find a solution [7].

So, the most recent state-of-the art path planner is called Approximate Clearance Evaluation (ACE) which has been studied to improve the autonomous driving capability demanded by the Mars 2020 rover mission, where the landing site is the Jezero crater with a CFA up to 15-20%. It is significantly less conservative with respect of GESTALT approach [7].

2.2 Global Path Planning

The global path planning method can generate the path under the completely known environment (the position and shape of the obstacle are predetermined). Therefore, the global path planner is typically used to improve the local planner metric evaluation and strategy because it takes into account all the obstacle in the environment [5].

So, it involves two parts: establishment of the environmental model and the path planning strategy.

Moreover, the path planning algorithm must be fed the environment converted into a configuration space (C-space), visibility graph, Voronoi diagram or grid maps [8,9]. In mobile robot navigation, the environment is often represented with grid-based approach. In line with this, we can use a binary representation where the grid contains either an obstacle or free space or traversability representations which reflect the difficulty of traversing the respective area of the environment.

2.2.1 Search-based path planning

The environment information is represented in discrete graph form where a specific cost of action is associated between nodes in the graph. Most of these algorithms are based on the well-known A* algorithm, developed in 1968 [10], which use a heuristic function to estimate the cost from any node to the goal one. This allows to reduce the search space of the original Dijkstra algorithm on which it is based.

However, A* neither enables global constraint satisfaction nor efficient re-planning. It is forced to plan from scratch when any state transition cost happens. Therefore, in dynamic case, where the environment changes, it becomes inefficient because at each alteration of the environment, it should plan the whole path again.

One of the first path-planning algorithm with a replanning capability is Lifelong Planning A* (LPA*) [11] which combines the heuristic search of A* and incremental search of Dynamic SWSF-FP. LPA* still has limitations because it only allows to repeatedly determine optimal path, when the edge costs of the graph change, between a fixed start node and goal node.

So, Koenig and Likhachev use LPA* in order to develop a new algorithm, D* lite [12], which implements the same navigation strategy of [13] but with less complexity and allows to determine the shortest path between current vertex of the mobile robot (robot is moving towards the goal node) and the goal vertex.

In [14] we have a comparison of the path planning performance of A*, D*, LPA* and D* Lite, in static and dynamic environments with different map sizes and complexity. The results show as A* is much faster and more efficient in static environments,

whereas D* Lite is the best algorithm in static environments with a high complexity and in dynamic environments.

2.2.2 Any-angle path planning

Typically, the above-mentioned algorithms use 4 or 8-connected graph obtained from grid-based representation of the environment. Therefore, the paths obtained by these algorithms are often suboptimal. In particular, they are not the true shortest paths, since the mobile robot's motion is restricted to a small set of possible headings which are multiple of $\pi/4$. The path is constrained to pass either from a centre of any grid to the centre of the adjacent grid or through the grid edges.

Field D* [15] extends D* and D* Lite with linear interpolation in order to produce globally smooth and low-cost paths. For instance, considering a standard 2D grid, it locates each node of the graph at the corners of grid cells and improves the cost estimation, from one node to its neighbouring nodes, using interpolation. The results presented in [15] show that this algorithm, in general produces less costly path than regular grid-based planning, but there are rare occurrences where the interpolation assumption is incorrect.

Joseph Carsten et al. propose an extension of Field D* to 3D grids that is the case of robots that navigate in full three-dimensional environments (e.g. aerial vehicles) [16].

In 2005, in order to overcome the local planner GESTALT problems mentioned in the section 2.1 a version of Field D* has been integrated into MER flight software in order to enable simultaneous local and global planning during Auto-Nav. A revised version of Auto-Nav was uploaded to the rovers during the summer of 2006 [5].

Kenny Daniel et al. [17] propose Theta* algorithm which is a variant of A* which allows to propagate information along grid edges without constraining paths to grid edges. They show that Theta* produce shorter paths with respect to both A* with post processing (to obtain smoother paths) and Field D*. Moreover, an incremental version of Theta*, called Incremental Phi*, has been implemented for unknown 2D environments [18].

2.2.3 Path Planning under global constraints

The problem of path planning under global constraints arises from the fact that the path should satisfy a number of different criteria. The optimal path generated should consider several factors like constraints on the space environment, the optimization of the resources of the system, the path length, etc.

A possible solution, in order to consider multiple criteria, could be defining a cost function for each criterion and use as objective function to minimize the weighted sum of the single cost functions. This approach has many limitations because is very complex to understand the relationship between the weights and the solution produced, making hard combining different cost functions to give a desired behaviour. A second, more natural, approach consists in defining a set of constraints directly in the path planning problem, allowing to generate paths that satisfy the constraints over the path.

In line with the second approach Logan and Alechina [19] introduce an extension of A* algorithm called A* with bounded cost (ABC), which allows to specify the set of constraints using inequality constraints.

Another solution, which can work in dynamic environments is the CD* algorithm, introduced by A. Stenz in 2002 [20], that address the combined problem of optimal path planning with global constraints with re-planning using a composite function which is a weighted sum of an objective function and a constraint function.

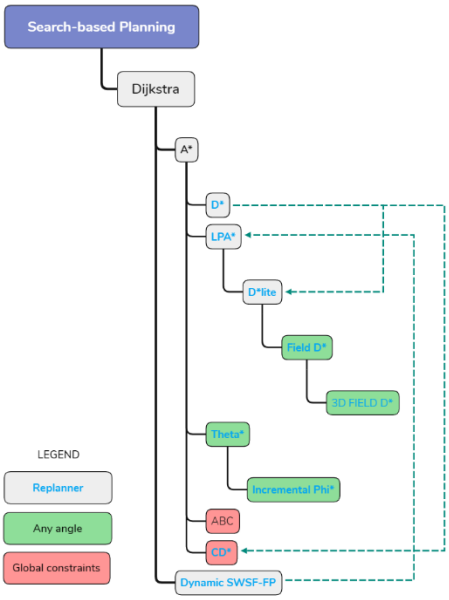


Figure 5 - Search-based Path Planning Classification

2.2.4 Artificial potential field algorithms

Artificial potential fields algorithms represent the mobile robot state as an electric point charge where an attractive field is centred to the global goal and a repulsive field surrounds each obstacle in the environment. In this way, the robot is attracted by the goal and repelled by the obstacles avoiding eventual collisions. The path is obtained simply following the steepest resulting gradient to the goal. This approach allows to obtain the trajectory with little computation, but the main problem is that it is vulnerable to local minima. This means that it fails to find a path or find a path which is sub-optimal [21].

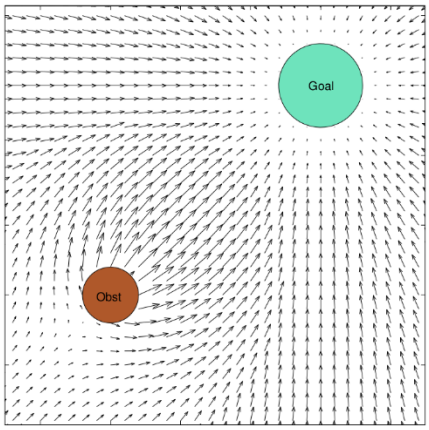


Figure 6 - Potential Field Representation

2.2.5 Path Planning using Fast Marching Method

The Fast-Marching Method is an evolution of the Level Set Methods introduced by Osher and Sethian [22]. Its behaviour takes inspiration from the Fermat's optic principle which describe the propagation of light waves. It claims that: "the path of a beam of monochromatic light follows the path of least time". So, it is an efficient computational numerical algorithm which permits to track and model the motion of a physical wave interface (front). The FM path planning applies this algorithm in order to solve the planning problem. So the path generated corresponds to the path that a wave front would follow from a start location (point where start the propagation) to a target location where the travelling speed along the path is the expansion velocity of the

wavefront. The travelling speed depends on the medium where the wave is expanding according to the Fermat's principle.

Gomez et al. [23] claim how the classic FM algorithm generates paths which are the shortest in length, but they may not be safe, being close to the obstacles. So the path obtained is not the shortest in time because the mobile robot should move slowly in proximity of the obstacles. So, in [23] we have a review of possible methods that can resolve this problem:

- The Voronoi Fast Marching (VFM) where Voronoi Diagram is used in order to obtain a roadmap of the map. So the FM algorithm is used to search a path over this representation. In this case we have that the time spent in the search decreases
- An improved version, called FM2 (Fast Marching Square), which consider additional wave sources centred in the obstacles (FM classical approach considers only one source at the target point).
- A saturated version of FM2 where the map of front propagation speeds resulting applying the fast-marching method is saturated to a maximum allowed speed, which is the maximum speed the robot may receive
- Heuristic version of FM2 (FM2*) which tries to reduce the total number of expanded cells using heuristic function which estimates the cost from a point to the goal.

Garrido et al. [24] apply the fast-marching method to the path planning considering spatial environments. Specifically, the fast marching (FM) has been modified in order to consider the effect of an external vectorial field, obtaining the so called FMVF.

This approach has been chosen mainly to account a series of typical constraints in Mars Rover exploration mission: height change, slope and rugosity of the terrain, composition of the terrain (proportion of sand) and possible slippage of rover wheel due to sand or steep slopes.

Moreover, algorithms with dynamic replanning function, based on FMM, are E* proposed in 2005 by Philippsen and Siegwart [25] based on D* and dynamic fast marching (DFM) proposed in 2009 [26], based on LPA* and D* lite, which have no heuristic.

In 2019, Sinyukov and Padir [27], introduce a new wave-propagation algorithm, which is a special case of the Fast-Marching Method where the front velocity is constant.

Specifically, it works with two-dimensional grid and uses wave front with circular shape.

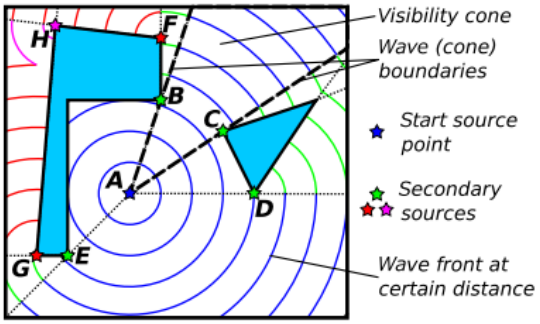


Figure 7 – C-wave waves propagation where point A is the start source of wave and B,C,D,E,F,G,H are secondary sources of waves [27]

This approach operates directly on the grid geometry of the environment using discrete geometric primitives (circular arcs and lines) to represent the wave front instead of using individual vertices. Moreover, this algorithm can be classified as any-angle path planning previously discussed in the section 2.2.2. In its basic implementation, CWave requires only integer arithmetics (CWaveInt) and this allows to have any-angle path planning also on low-cost embedded microcontrollers without floating-point units (FPUs).

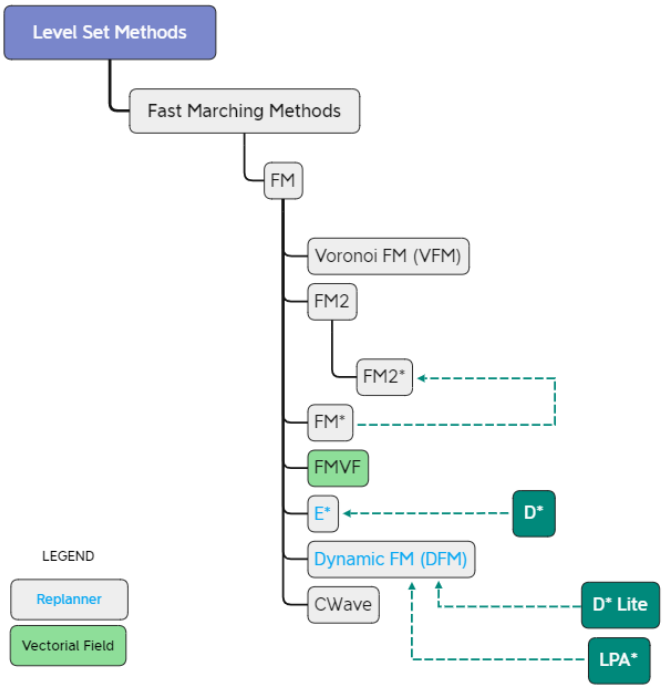


Figure 8 - Wave-based Algorithm Classification

2.2.6 Path Planning using Randomized Techniques

Path planning Randomized Techniques are used in high dimensional C-spaces in order to address the motion planning for mobile robots with many degrees of freedom or for many robots which must simultaneously coordinate their actions. The randomized algorithms use a stochastic approach in order to explore the state space and this allows a rapid exploration of the entire space. However, in order to reach a complete optimal solution, this kind of approach would need an infinite time. So, in practice, time bounds within which the planner process has to finish are defined. As result, the path solutions are generally not the optimal ones.
The most used algorithms, which uses this approach, are Probabilistic Road Maps (PRM) and Rapidly-exploring Random Trees (RRT).

2.2.7 Path Planning using Machine Learning

In recent few years, many results have been achieved trying to solve the path planning problem using machine learning techniques, whereas most of these approaches solve the problem using Neural Networks and Reinforcement Learning [28]:

- Artificial neural networks consist in computational network models which simulate the mechanism of learning in biological organism. The network contains several computational units that are referred to as neurons. Each single neuron can have several inputs which are scaled with particular weights, which affect the function computed by the computational unit. The network can be represented as a graph of nodes (computational units) connected by edges, which propagate the computed values (activation information) from one node to another where the weights are the intermediate parameters. The behaviour of the network can be influenced by changing these parameters. So, the aim is to iteratively train the model adjusting the numerous weights, in order to modify the computed function to obtain the desired behaviour of the system.

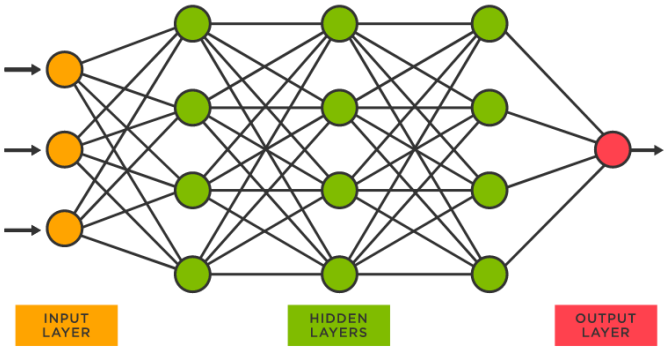


Figure 9 - Graph representation of a simple neural network with 3 hidden layers

In global path planning problem, neural networks are used to represent and learn deep features from environment representation (e.g. orbital images). So according to extracted deep features a path planning strategy can be determined.

Specifically, in recent years, a wide attention was given to Deep Convolutional Neural Networks (DCNNs) which have shown a superior capacity in the representation and in learning capability of deep features, in computer vision field, for tasks such object and action recognition and image segmentation.

Nachuan Ma et al. [29] propose an innovative approach to the path planning, based on unsupervised learning using Conditional Generative Adversarial Networks (CGAN) which uses as input the environment image maps. They claim how A* algorithm tends to consume much time and huge memory usage with the increase of the size of the configuration space. So, they implement a CGAN heuristic version of RRT (rapidly exploring random tree) path planner which allows to overcome the critical issues of the randomized path planner above mentioned and to obtain good result in terms of computational resources spent. They use an architecture based on generator realized with a revisitation of encoder-decoder with skip connections (U-NET) and discriminator realized with a deep convolutional network. The links in the encoder-decoder allow to restore high-resolution details which can be lost during the down-sampling phase. In particular, the generative adversarial network is used, after training phase, to generate region of interests (ROIs) on the environment map over which the well-known RRT can work. In this way, the number of states visited by the RRT algorithm radically decreases and we have a speed up of the convergence to the optimal path. Results shown as they have a much better performance

compared with conventional randomized path planners in terms of time cost, number of nodes and path length.

Tianyi Zhang et al. [30] propose a similar approach with comparable results, using a different GAN architecture. The generator is realized with encoder-decoder where encoding and decoding blocks include convolutional and residual network layers. This is a different method with respect the U-NET to not lose important features used from the decoder layers for the ROI generation. Moreover, they propose a double discriminator, one for the initial and target points pairs, and the other for the region of interest to improve the generator capability to generate feasible region.

Natalia Soboleva et al. [31] propose a path planner, which works on 2D static environments, based on image-to-image translation. The authors claim how classic heuristic search algorithms iteratively explore the search space guided by heuristic function making unnecessary area exploration around the obstacles. So, they propose an alternative method based on state-of-the-art deep learning techniques which avoid unnecessary state-space exploration by construction directly generating path images in response to context input (environment grid-map input).

- Reinforcement learning consists in teaching a system to take an appropriate action using a mechanism of reward and punishment. Specifically, the system is trained to modify its policy, which is the probability distribution associated to the execution of a particular action at a particular state, in such a way to maximize the "reward function". After the agent executes an action, it receives feedback from the environment (immediate reward), which is the evaluation of the action made by the environment. The Reinforcement Learning model is shown in Fig. 10. The system can also be trained using an "Imitation Learning" approach. So, in the training phase, the system tries to modify its policy in order to obtain a behaviour which is most like the expert.

  Moreover, the environment in which the system works is typically stated in the form of a Markov Decision Process (MDP) which provides a mathematical framework for modelling decision making problems. Through the reward function, obtained from the immediate rewards, and value function the MDP realizes the optimal action strategy. The global path planning, being a sequential decision-making process, can

be formulated as MDP and the optimal path planning policy can be found using value function estimation [3].

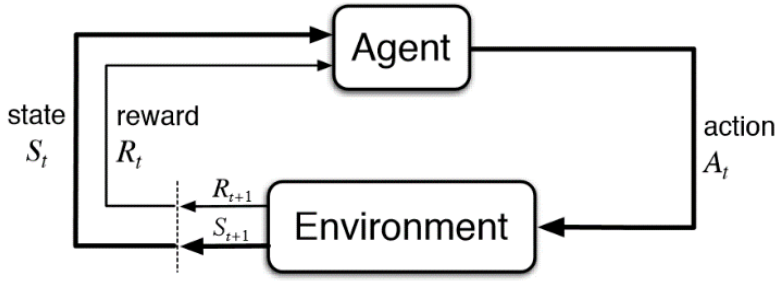


Figure 10 - Model of Reinforcement Learning

Penggang Gao et al. [32] show a reinforcement Q-learning approach to find a collision-free path from starting point to final point. They propose an innovative approach based on random sampling which differs from common Q-learning approaches, which use methods based on grid maps. Collision-free points are generated randomly on the environment map, whereas a point is deleted if it is located on obstacle, until the process reaches the expected number of points. These points are used by the Q-learning as states. This method allows to reduce the number of states and actions for Q-learning saving computational resources. The results show as the algorithm can generate shorter and smoother path compared with BFS (Breadth-First Search) algorithm.

Recently, we have had research progresses in the combined use of Reinforcement Learning and deep neural networks since it enables to tackle decision making tasks which involve complex environments. Specifically, the neural networks (NN) typically consist of CNNs used for feature extraction, and fully connected layers to map the features to probability distribution over actions.

A. Tamar et al. [30] introduce an innovative approach, with respect classic planning algorithm based on RL which uses a NN-based policy that can effectively learn to plan. In particular, the key of their approach is using a specific type of CNN to calculate the value-iteration. Since the value-iteration block is constituted by NN it can be trained using standard backpropagation.

Jiang Zhang et al. [3] propose a combined use of Q-learning Reinforcement Learning and Neural Networks. In particular, they implement a deep Q-learning approach using double branches convolutional network (DB-CNN) which can

obtain optimal paths directly from orbital images of planetary surfaces without environment mapping.

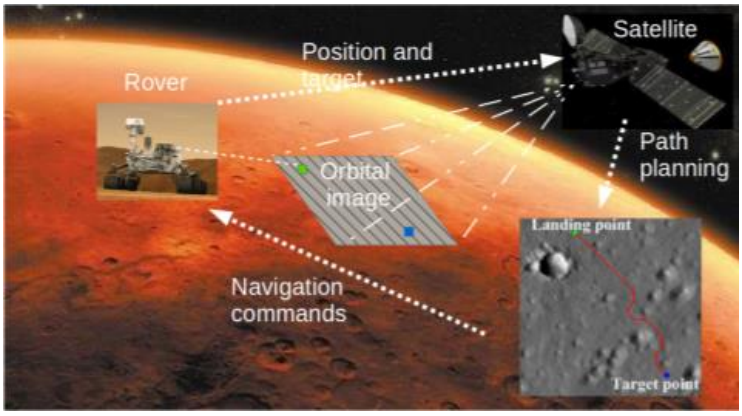


Figure 11 - Global path planning using orbital images

Inspired by the Value Iteration Network [30] Jiang Zhang et al. use the double branch neural network for the value function estimation in the MDP. The value function estimation is obtained from the neural network minimizing the loss function trough the gradient descent method. They evaluate the global path planning performance using two datasets:

- The first dataset consists of 64x64 grid maps with random obstacles and positions of the rover. The map contains 0 (free grid) and1 (obstacles). This approach is widely used to evaluate the path planning performances.
- The second dataset consists of 128x128 Martian surface images from HiRISE.

Results show how this approach can achieve global path planning with higher efficiency, faster convergence and precision with respect VIN which has low training and planning efficiency since it requires, for value function estimation, a long iteration time inside the network.

2.3 Global Path Planning Algorithm Selection

In this chapter we have tried to provide a clear explanation of what is the state-of-the-art of path planning algorithm focalizing our attention on global path planning solutions in

line with the goal of our thesis which is finding innovative solutions for global path planning, based on the recent developments in deep learning.

We have seen how typical algorithms need to explore the state space to pull out a path planning strategy. In particular, we have seen different approaches which allow to optimize the way in which the algorithms explore the state space:

- Adopting heuristic functions which allow to optimize the path considering directly, in the search, what is the final target evaluating the cost of each node to the goal one. This permit to reduce the search space.
- Adopting incremental version of the algorithms in order to not explore the state space, from the scratch, whenever we have a change in the environment (dynamic obstacles). This allows to minimize the number of nodes the algorithm searches when it deals with dynamic environments.

However, using these algorithms, we cannot avoid the unnecessary state space exploration around the obstacles because even if they are minimized, they cannot be eliminated.

As we have seen, using a trained deep neural network for path planning would allow, by construction, to have a direct generation of path planning strategy in response to context inputs (environment maps) with a gain in saved computational resources. Such networks are inherently reactive, and in particular, with a lack of explicit planning computations since the path planning strategy is obtained using a mathematical model which represent the link between input and output.

In machine learning context, the path planning problems are mainly resolved using RL techniques previously mentioned. In particular, we have seen how deep RL is capable of solving complicated planning tasks. However, there are few research works which focalize their attention to use the pure neural networks to find a global path planning strategy.

With these premises we propose a new solution for global path planning which is based on unsupervised generative models which represent the new recent developments in deep neural network in computer vision field. The proposed algorithm treats the path planning problem as an image-to-image translation problem. In particular, is based on imitation learning. So, the neural network tries to find a solution path of an unknown environment imitating the behaviour of a well-known algorithm.

The solution is an approximation of the solution that the well-known algorithm would have generated in the same conditions.

This approach is justified by the fact that the learning-based method tries to achieve a behaviour which is effective for exploration. We no longer have unnecessary state space exploration, but we have a direct mapping between the environment map and the path solution saving computational resources.

Chapter 3

Generative Adversarial Networks

Generative Adversarial Networks (GANs), introduced by Ian Goodfellow et al. [21] in 2014, is an innovative approach to generative modelling, based on deep learning, which uses convolutional neural networks. Generative modelling is a form of unsupervised learning which discovers and learns the patterns in input data in order to generate plausible distribution of data. Thanks to the outstanding data generation capability of GAN methods, the generative models have gained a considerable attention in the unsupervised learning field. This has opened the possibility to new developments in unsupervised learning because most of the machine learning techniques were based on supervised learning approaches, until the introduction of GAN networks.

GAN architecture is based on two neural networks, the generator trained to generate new samples, and the discriminator trained to classify the samples as real (from target domain) or fake (generated). The two networks are trained simultaneously in an adversarial game, where the generator aims to fool the discriminator in the classification of real and fake data.

Practically, GAN are used in many applications, most notably in image-to-image translation tasks, synthesis of images such as the generation of photorealistic photos of people, scenes or objects and computer vision tasks in general [33].

In this Chapter we introduce, in details, the basic theory of GAN and we specialize the GAN network to accomplish the path planning task, introducing the chosen GAN architecture .

3.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a class of machine learning methods which aims to explain the cognitive process in biological organism. They simulate the learning's mechanism of biological organisms representing it through a mathematical model. Generally speaking, the ANN is a complex network composed by interconnected computational units, that are referred to as neurons, with a parallel/series structure. Typically, the network is represented as a graph of nodes (computational units) connected by edges, which propagate the computed values (activation information). The information, processed within the network, propagates from one node to another inspired by the biologic neuron behaviour.

Specifically, each computational unit is connected to another through the weights. The weights can be interpreted as the strengths of synaptic connections in biological organism. Each node can have a multiple number of input nodes, which are scaled with different weights. The computational unit computes a function of the input, using the weights as parameters of the function. Globally, the ANN can influence the network behaviour using the weights as intermediate parameters. So, the learning of the network occurs changing the values of the weights.

In recent years, the artificial neural networks have been applied in many research fields thanks to their excellent capacity in classification, pattern recognition and prediction. Specifically, we would mention a small set of fields, where neural networks have defined a great step forward, like control system (e.g. control of processes and vehicle control) , object recognition and medical diagnoses [34].

3.1.1 Supervised and Unsupervised Learning

Machine learning methods are often divided into Supervised and Unsupervised learning methods. The idea behind the predictive or supervised learning, is training a mathematical model to establish a mapping from inputs x to outputs y. Given a certain input, the prediction of the system's output is done using a training dataset, which comprises multiple samples composed by input variables (X) and output class labels (y)

pairs. At each training iteration, the model is corrected to obtain outputs more like the expected ones. Fig. 12 shows a graphic explanation of supervised learning.

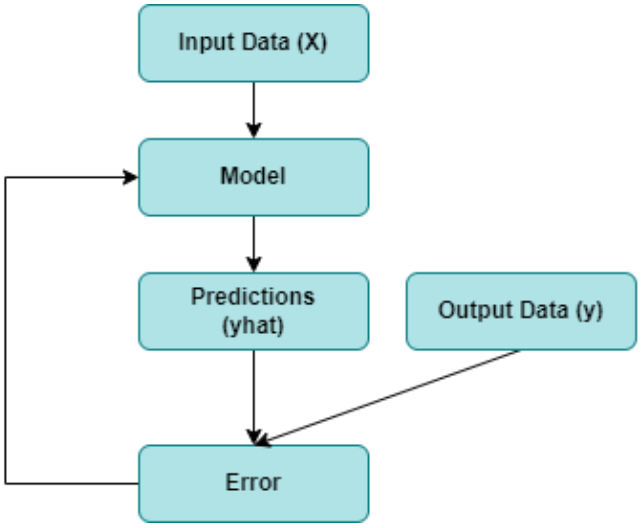


Figure 12 – Example of Supervised Learning

Instead, in unsupervised learning, the model is not provided with pre-assigned input variables (X) and output class labels (y) pairs. Given certain inputs, the objective is to extract "interesting patterns" in the data. In this case the model is not predicting anything and there is no correction of the model. Fig. 13 shows a graphic explanation of unsupervised learning.

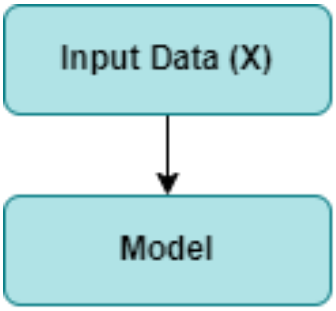


Figure 13 - Example of Unsupervised Learning

A typical application of supervised learning is the classification and regression. In classification the model is developed in such a way it can predict a class label given an example of input variables (discriminative modelling). For instance, the input can be an image which represents a handwritten digit, and the label is a number between zero and nine. Instead, a regression model is used to predict a continuous variable. It performs a mapping between the input data and continuous variables.

Unsupervised models could be used to create or generate new examples in the input distribution (generative modelling). The generative model discovers and learns the patterns in input data in such a way that the model can be used to generate new samples that plausibly could have been drawn from the original dataset. Synthetization of data can be used in applications like open world video game production. The algorithm based on manually created graphical landscapes could generate new worlds.

3.1.2 The Perceptron

In order to introduce the basic architecture of Neural Networks, we present the simplest neural network, the single layer feed-forward network, also known as perceptron, introduced by Rosenblatt in 1958 [35]. It consists in a single input layer and a single output node.

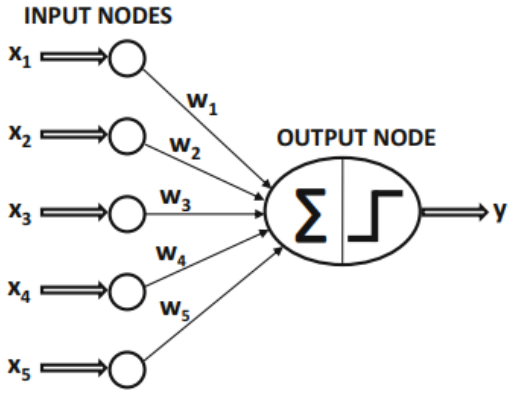


Figure 14 - Perceptron without bias [36]

In the basic architecture of perceptron (Fig. 14) the single arrow represents the connection between the single input node and the output node, and every single connection is associated to a weight with which scaling the input node.

We show the Perceptron behaviour in the context of supervised learning because of its simplicity.

We consider a binary classification problem where each training instance $(\bar{X}, y)$ is composed by:

- $\bar{X} = [x_1, \dots x_d]$ that represents the vector containing d feature variables
- $y \in \{-1, +1\}$ that represents the observed value

The observed value (binary class variable) is given as a part of the training data. The network aims to make a prediction of the class label $\hat{y}$ for the cases in which the class variable is not observed.

The overall architecture is composed by input layer, containing d nodes which transmit the vector of d features $\bar{X} = [x_1, \dots x_d]$, d edges of weight $\bar{W} = [w_1 \dots w_d]$ and output node which computes the function of the d features. Specifically, we consider the linear function $\bar{W} \cdot \bar{X} = \sum_{i=1}^{d} w_i x_i$ followed by the sign of the real value computed. The predicted output is computed as follows:

$$\hat{y} = \text{sign}\{\bar{W} \cdot \bar{X}\} = \text{sign}\left\{\sum_{i=1}^{d} w_i x_i\right\} \tag{3.1}$$

The sign function allows to map a real value to the binary class +1 or -1. Specifically, has the role of activation function. Generally speaking, the most widely known activation functions are the following:

- Identity: $\Phi(v) = v$
- Sign function: $\Phi(v) = \text{sign}\,(v)$
- Sigmoid function: $\Phi(v) = \frac{1}{1+e^{-v}}$
- Tanh function: $\Phi(v) = \frac{e^{2v}-1}{e^{2v}+1}$
- Rectified Linear Unit (ReLU): $\Phi(v) = max\{v, 0\}$
- Hard tanh: $\Phi(v) = max\{min[v, 1], -1\}$

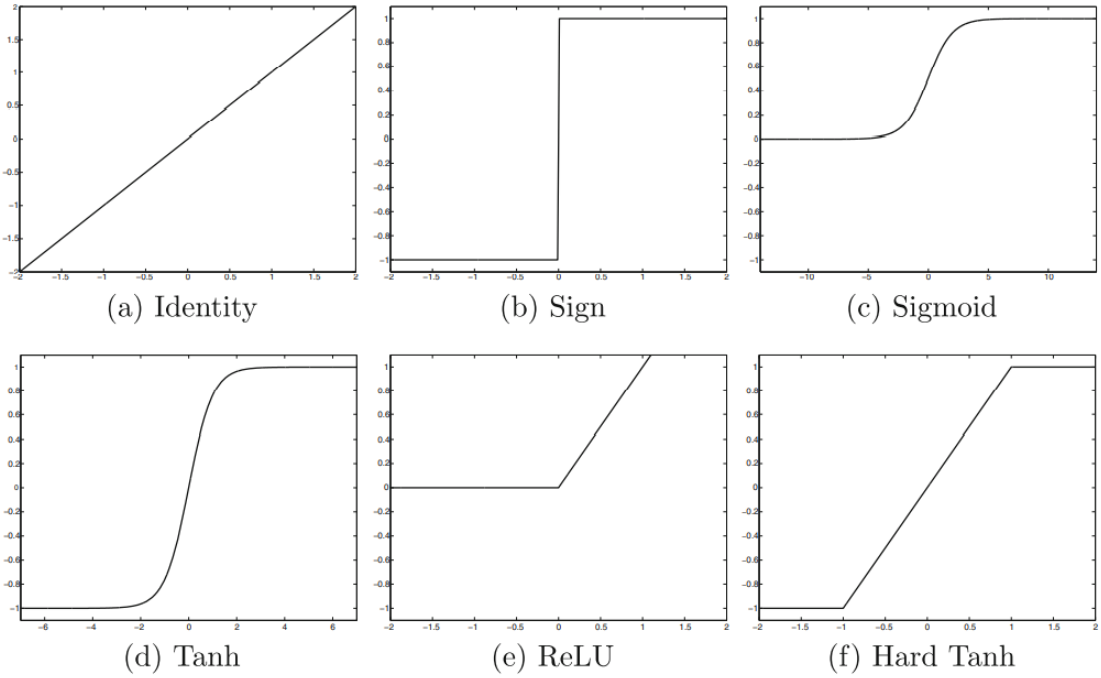


Figure 15- Various activation functions [36]

The error of the prediction with respect the observed value is given by:

$$E(\bar{X}) = y - \hat{y} \tag{3.2}$$

The training of the network is obtained through the backpropagation algorithm which includes two main phases:

- Forward Phase: the inputs from the training instance are fed into neural network which computes, using the current set of weights, the predicted output. The latter is compared to that of the training instance.
- Backward Phase: this phase aims to learn the gradient of the loss function with respect to the different weights.

Specifically, the backward phase consists in a gradient-descent method that minimizes the loss function $L(\bar{W})$. The gradient descent method updates the weights in order to produce a step in the negative direction of the gradient (learn the gradient). At the time t+1, the generic weight is updated as follows:

$$w_i^{(t+1)} = w_i^{(t)} - \alpha \frac{\partial L\left(\bar{W}^{(t)}\right)}{\partial w_i^{(t)}} \tag{3.3}$$

$$\forall i \in \{1 \dots, d\}$$

where α is the learning rate.

Considering data set D which contains feature-label pairs the loss function is computed as follows:

$$L = \frac{1}{2}\sum_{(\vec{X},y)\in\mathcal{D}} (\mathrm{y} - \hat{y})^2 = \frac{1}{2}\sum_{j}\left(\mathrm{y}^{(j)} - \hat{y}^{(j)}\right)^2 \tag{3.4}$$

Imposing that $y = \sum_{i=1}^{d} w_i x_i$ the backward phase computes the derivative of $L(\bar{W})$ with respect to the weights:

$$\frac{\partial \mathrm{L}}{\partial w_i} = \frac{\partial L}{\partial \mathrm{y}^{(j)}}\frac{\partial \mathrm{y}^{(j)}}{\partial w_i} = \sum_{j}\left(\mathrm{y}^{(j)} - \hat{y}^{(j)}\right) * x_i{}^{(j)} \tag{3.5}$$

Considering (3.3) and (3.5) the complete expression for the weight update is the following:

$$w_i^{(t+1)} = w_i^{(t)} - \alpha\sum_{j}\left(\mathrm{y}^{(j)} - \hat{y}^{(j)}\right) * x_i{}^{(j)} \tag{3.6}$$

$$\forall i \in \{1 \dots, d\}$$

Up to now, we have considered the simplest ANN structure but in general, we can have feed-forward neural networks with more and more hidden layers. They are called multi-layer perceptron (MLP) (Fig. 16). For more details you could consult [36].

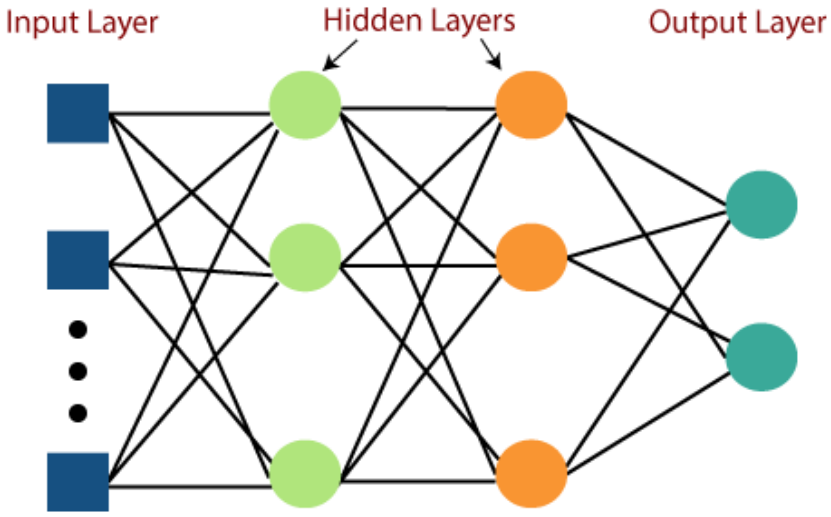


Figure 16 - Example of Multi-Layer Perceptron

3.2 Convolutional neural networks

In deep learning field, Convolutional Neural Networks (CNNs) have become fundamental to solve problems in image domain, like image-to-image translation or image recognition since they are designed to work with grid-structured inputs that present strong spatial dependencies in local regions of the grid. Considering a generic image, adjacent spatial locations (pixels) often have similar color values.

The input data of the network is a two-dimensional grid structure where the individual grid values are referred as pixels. Each pixels defines a spatial location within the image. However, in order to define the number of color channels of the image, an additional dimension (depth) is considered (e.g. RGB image has three color channels). Therefore, if the two-dimensional image has a dimension of 64x64 pixels and a depth of 3, corresponding to RGB color channels, the overall dimension of the input multidimensional array of the network is 64x64x3.

Each layer in the CNN is a 3-dimensional grid structure, where each dimension is respectively height, width and depth. It is worth noting that the depth of a layer in convolutional neural networks differs from the network depth, which represents the number of network layers.

The CNN parameters are organized into sets of three-dimensional structural units, known as kernels or filters. The filter's depth is always the same of the layer to which the kernel is applied whereas the filter's spatial dimension is typically smaller than those of the layer to which is applied.

So, the main difference between a CNN and an ordinary neural network is the convolution operation performed by the network. In particular, the operation places the filter at each possible position in the image or the hidden layer and performs convolution between the filter's parameters and the matching grid-structured volume computing the filter's output which defines a feature in the next layer. So, the spatial relationships within a layer are inherited from the previous layer because each feature value match a local spatial region in the previous layer. The number of filters applied to the layer defines the spatial height and width of the next hidden layer.

In the training, the values stored in the filters are the parameters that are learned by the CNN. Initially, these are random, but at each training iteration the filter adapts its parameters to capture interesting features such as edges or color combinations.

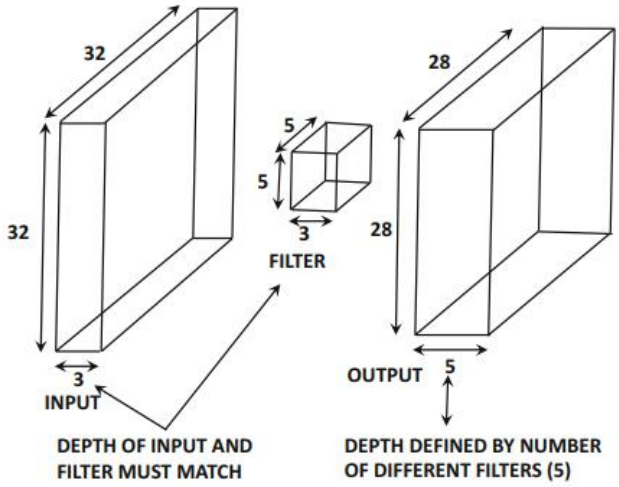


Figure 17 - The convolution between an input layer of size 32 × 32 × 3 and a filter of size 5 × 5 × 3 produces an output layer with spatial dimensions 28 × 28 [36]

3.2.1 Convolution Operation

In order to define the convolution operation, we consider a *p*th filter applied to a *q*th layer. The filter has dimension: $F_q \times F_q \times d_q$. The filter's parameters are defined by a three-dimensional tensor:

$$W^{(p,q)} = \left[w_{ijk}^{(p,q)}\right]$$

where the indices i, j, k define the positions along the height, width and depth of the kernel. The three-dimensional tensor $H^{(q)} = \left[h_{ijk}^{(q)}\right]$ represents the feature maps in the *q*th layer. So we define the convolutional operation from the *q*th layer to the *(q+1)*th layer:

$$h_{ijp}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} h_{i+r-1,j+s-1,k}^{(q)} \qquad \forall i \in \{1 \dots, L_q - F_q + 1\}$$
$$\forall j \in \{1 \dots B_q - F_q + 1\}$$
$$\forall p \in \{1 \dots d_{q+1}\}$$

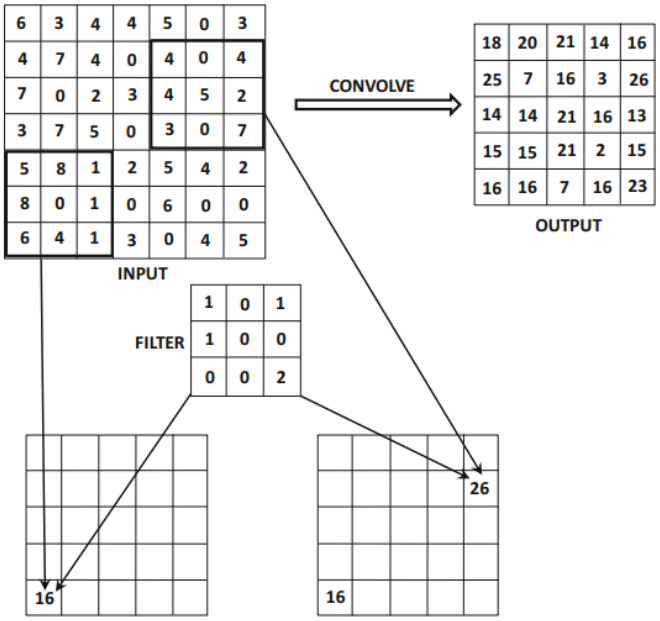


Figure 18 – Example of convolution between 7x7x1 input and 3x3x3 filter with stride of 1 [36]

The convolution operation can be seen as a simple dot product over the entire volume of the kernel which is repeated over each valid spatial position (*i, j*) and for each filter indexed by *p*.

3.2.2 Padding

Applying the convolution operation from the *q*th layer to the *(q+1)*th layer we obtain a size reduction of the *(q+1)*th layer with respect to the previous layer. This size reduction can cause an undesirable effect which consists in the loss of some relevant information along the borders of the image or the feature map (hidden layers).

This problem can be resolved using a method, called Padding, which consists in the addition of $(F_q - 1)/2$ pixels or feature values around the borders of the input image or the feature map. Usually, the value of each padded pixel or feature values is set to 0. When the kernel is applied to the extreme spatial position (along the edges) of the image or feature map, a portion of the filter "sticks out" from the borders and it is superimposed also to the padded portions. These do not contribute, in the convolution, to the final dot product because their values are equal to 0. Therefore, only the portion of the layer where the values are defined contribute to the final dot product.

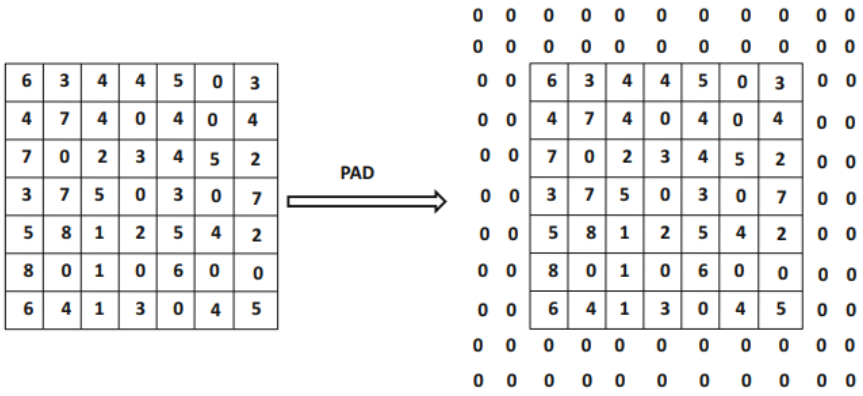


Figure 19 - an example of padding [36]

3.2.3 Strides

We have another parameter which influence the spatial footprint of the image or the hidden layer. If not specified, the convolution is performed moving the filter at every position across the image or feature map (Fig. 20). However, it is not necessary to perform convolution moving the kernel along the spatial locations with a step size of 1. We can manipulate the step size with the stride parameter. Naturally, increasing the stride/step size we reduce the level of granularity of the convolution operation. If not specified, the default value is one which corresponds to the previous case (Fig. 20). If a stride of S_q is used in the qth layer, the output has height of $(L_q - F_q)/S_q + 1$ and a width of $(B_q - F_q)/S_q + 1$ where L_q and B_q are the height and width of the input of the qth layer, respectively.

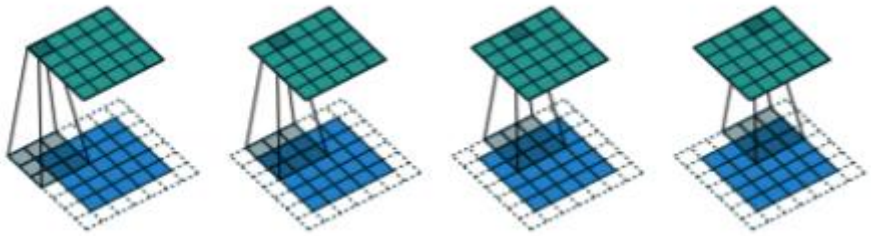


Figure 20 - A 3x3x1 kernel (grey) being passed over a 5x5x1 input image (blue) with padding and strides = 1 to generate the 5x5x1 output (green) [37]

3.2.4 The problem of Overfitting: Data Augmentation

In order to introduce the overfitting problem, we consider the simplest case of a classification problem, using a supervised learning approach. In this scenario, the

network aims to predict a certain class label given a certain unseen input. To do this, the model has to be first trained based on a certain training instance. In the training phase, the network adapts its behaviour based on examples of input-label pairs contained in the training data set trying to give a prediction of the class label equal to the target one. So, we could stop the training phase, when the network predicts the targets on the training data set perfectly. However, fitting a model to a particular training data set does not always guarantee that the network will provide good prediction performance on unseen test data. This creates a gap between training and test data performance which is known as overfitting.

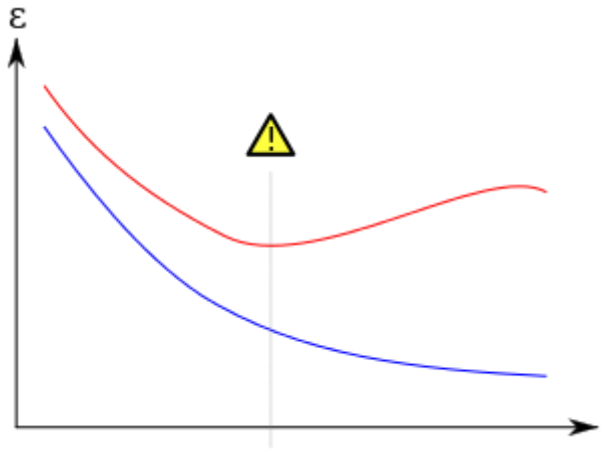


Figure 21 - Overfitting problem

The Fig. 21 shows the training error (blue) and the test error (error). We can see how, during the training phase, the training error and test error decrease until the vertical line, which means that the training and test performance are improving. After the vertical line, we have an overfitting problem, because the test performance starts to degrade, and the training performance continue to improve. In the ideal case, the validation error should continue to decrease with the training error. In this case, we should stop the training before the overfitting starts.

Intuitively, a larger dataset should improve the performance of the deep learning model allowing to resolve the overfitting problem. This technique stands for data augmentation. However, an increased dataset size does not always guarantee an increased amount of relevant information that can be used to improve the performance. Surely, the data set has to be increased trying to increase the diversity in the input data distribution. If we enrich the dataset with input-label pairs which are similar to the ones contained in the original dataset, we do not obtain any benefit.

When we deal with machine learning algorithms designed for image processing, data augmentation procedures are usually based on simple operations such as scaling, rotation, warping or mirroring. Also more sophisticated approaches can be used. For instance, we can use a trained generative model, like Generative Adversarial Networks (GANs), to do data augmentation [38].

3.2.5 Residual Networks

In recent few years, Image Networks have improved their performance in image classification thanks to the development of deep neural networks. A crucial factor, that has contributed to the drastic improvement, is the design of network architectures that have an increased number of layers that leads to have deeper networks. Alex Krizhevsky et al. [39] show as the network depth in CNN is of crucial importance. In particular, in their case the network's performance degrades if a single convolutional layer is removed.

However, Kaiming He et al. [40] show that the training of a deeper network can become very complex. In particular, when the network depth increases, network accuracy gets saturated and degrades rapidly whereas the performance degradation is not caused by overfitting problem. Fig. 22 shows how adding the deeper model leads to higher training and test error.

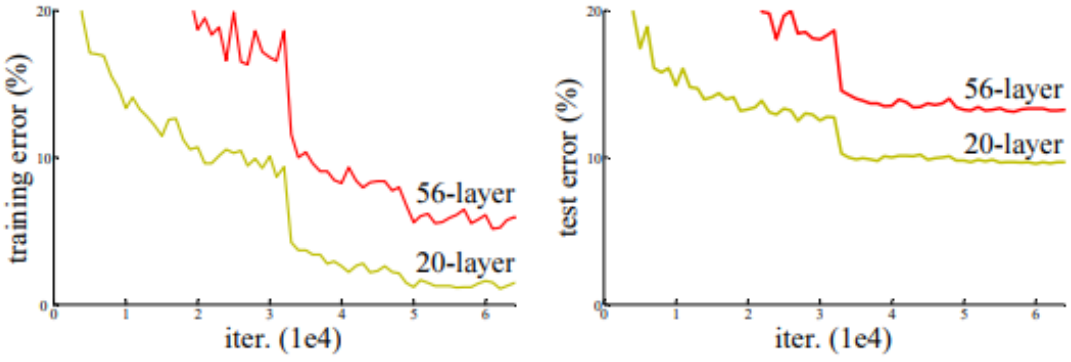


Figure 22 - Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" network[40]

So, they address the degradation problem introducing an architectural change, the residual module. The idea behind the residual module is to create skip-connections between layers i and $(i + r)$ for r>1 where the skip connection simply copies the input of

the layer i and adds it to the output of layer ($i + r$). The architecture is illustrated in Fig. 23.

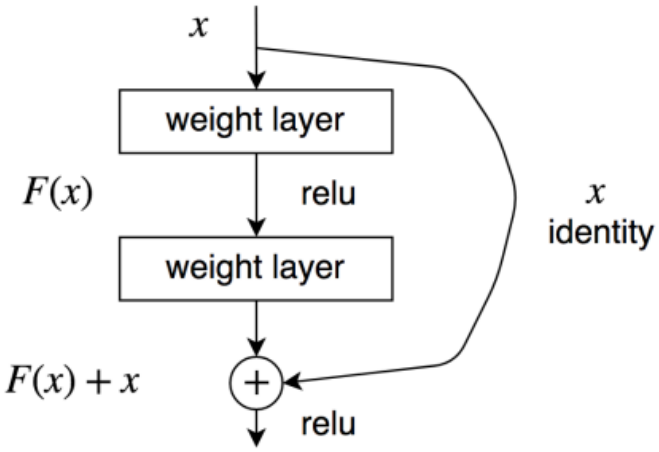


Figure 23 - Residual Module [40]

3.2.6 Variational Autoencoder (VAE)

An auto-encoder is a particular neural network composed of two parts (Fig. 24):

- An encoder which compresses input data in a lower-dimensional representation vector.
- A decoder which decompresses the lower-dimensional representation vector returning to the original domain

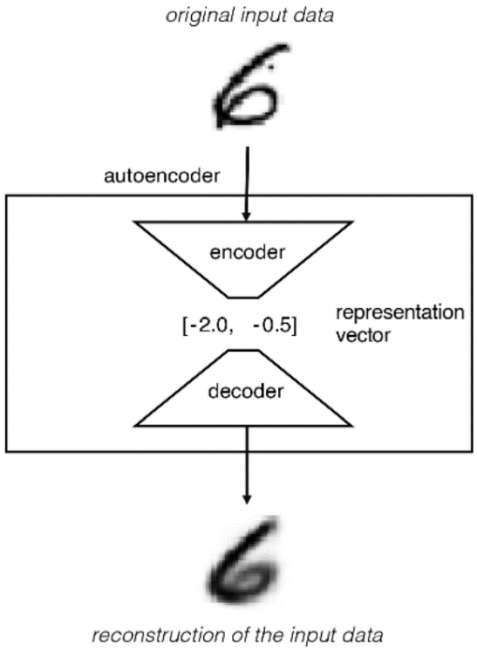


Figure 24 - Autoencoder Architecture [37]

In the training phase, the network's weights are updated in order to minimize the loss between the generated image and the original image.

Moreover, we can define the latent space as the space which contains the lower-dimensional representation vectors. Choosing any point from the latent space, the trained decoder should be able to generate novel images, using this point as input.

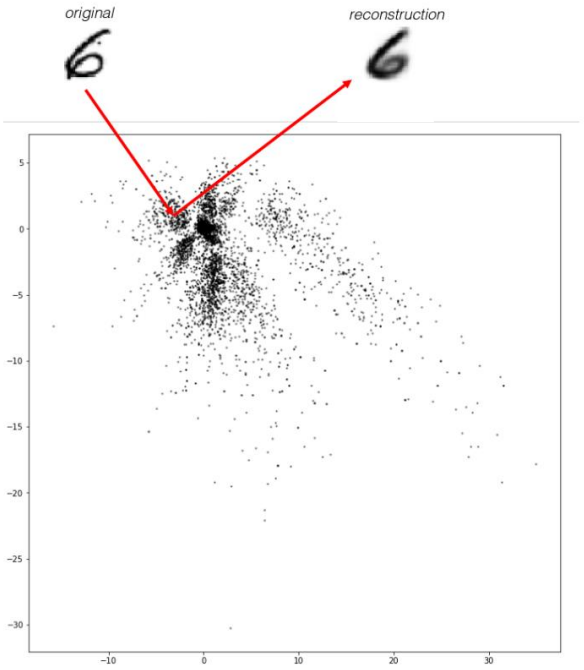


Figure 25 - Latent Space Representation [37]

Variational Autoencoders is a method which uses convolutional neural networks to generate data. The VAEs, like typical autoencoders, consist of an encoder and a decoder. Unlike autoencoders, which map each image to one point in the latent space VAEs map each image to a multivariate normal distribution around a point in the latent space.

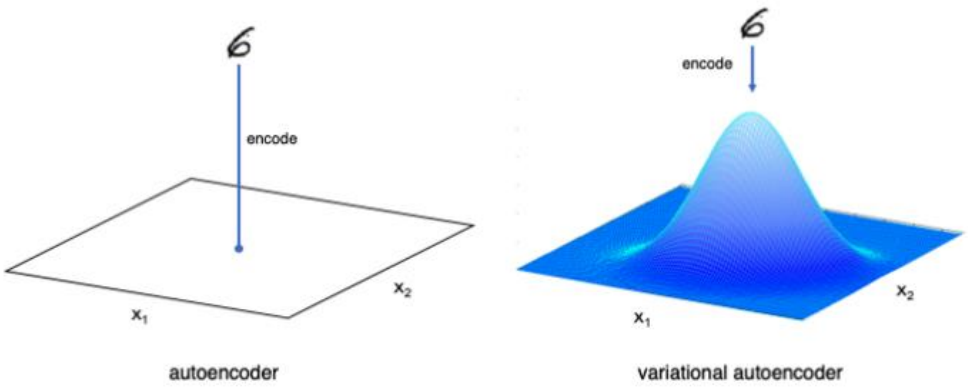


Figure 26 - Difference between autoencoder and variational autoencoder [37]

3.3 Generative Adversarial Networks

The first implementation of Generative adversarial network was published by Ian Goodfellow et al. [21]. The model consists of two neural network models implemented as two multilayer perceptron. The first is a generative model, generator G, and the second is a discriminative model, discriminator D. The generator aims to learn a data distribution p_g over the data x, which it synthesizes using input noise $p_z(\boldsymbol{z})$. The mapping between input and data space is indicated as $G(\boldsymbol{z}; \theta_g)$ where G is a differentiable function (multilayer perceptron) with parameters θ_g. Specifically, the generator input is a fixed-length random vector, sampled from a predefined latent space (e.g. a multivariate normal distribution).

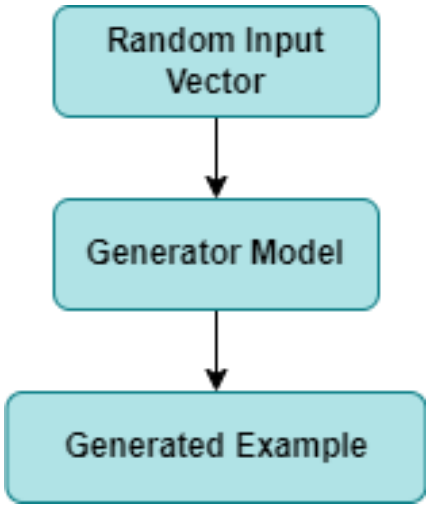


Figure 27 - GAN Generator Model

Also the discriminator is defined as multilayer perceptron $D(\boldsymbol{x}; \theta_d)$ with parameters θ_d. The function estimates the probability that $\boldsymbol{x}$ is taken from the data rather than the generator's distribution p_g. So, it takes as input samples from both p_g and training data distribution and aims to maximize the probability to assign the correct label to training (real label) and generated samples (fake label). The generator has the opposite goal, trying to fool the discriminator in such a way to classify the synthesized data as real. The discriminator and generator networks are trained each training iteration and are competing with each other. Usually, after the training phase, the discriminator model is

discarded because we are interested in the generator model which generates new samples.

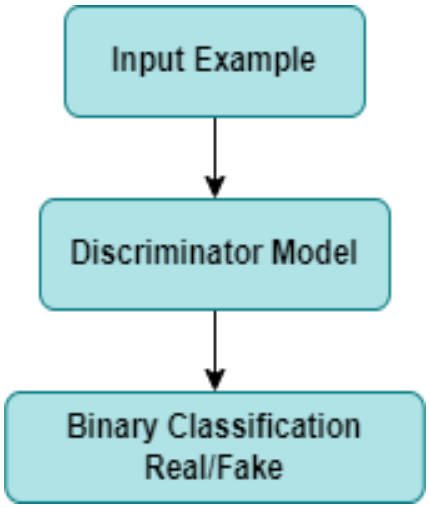


Figure 28 - GAN Discriminator Model

The GAN implemented in [21] was proven on the MNIST handwritten digit dataset. MNIST dataset stands for Modified National Institute of Standards and Technology dataset. It is a dataset composed of 70,000 28x28 grayscale images of handwritten single digits between 0 and 9. The generator G tries to generate numbers using input noise and the discriminative model is trained on both real MNIST data and synthesized data from generator model. In this case the network task is to classify a given image of handwritten digit into a class value, between 0 and 10. From the first publication of GAN, the new GAN architectures have mainly focused on creating visually realistic images and creating a mapping between different image domains (image-to-image translation).

3.3.1 GAN Adversarial Game

The discriminator D and generator G are in adversarial game. Specifically, the two models are trained together. The generator produces new samples in order to fool the discriminator while the discriminator tries to distinguish between real and fake data. At each training iteration the discriminator is updated to improve its capacity in distinguish real and fake samples, and the generator is updated based on the capacity of the generated samples to fool the discriminator.

In this way, the two models are competing against each other where:

- If discriminator successfully discriminates between real and fake samples, no change is needed to the model parameters whereas the generator is penalized with a large update of the model parameters.
- If generator manages to fool the discriminator, no change is needed to the model parameters whereas the discriminator is penalized with a large update of the model parameters.

At the beginning of the training, the generator will synthesize data that will be very different from the real data distribution. Intuitively, we could think that discriminator classification task (fake/real) is particularly easy at the beginning. However, the discriminator is not trained. So, the classification between the two distributions will be a rather challenging task. At each training iteration the discriminative capacity will get better as more examples are presented as well as the generator's fooling capacity trying to generate a data distribution closer to the real distribution.

Specifically D and G play the following two-player minmax game [36]:

$$\min_G \max_D (\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[\log\left(1 - D\left(G(\boldsymbol{z})\right)\right)\right])$$

Where $p_{\boldsymbol{z}}(\boldsymbol{z})$ is a prior distribution on input noise variables and $D(\boldsymbol{x})$ represents the probability that $\boldsymbol{x}$ comes from the data rather than the generator's distribution $p_{\boldsymbol{g}}$

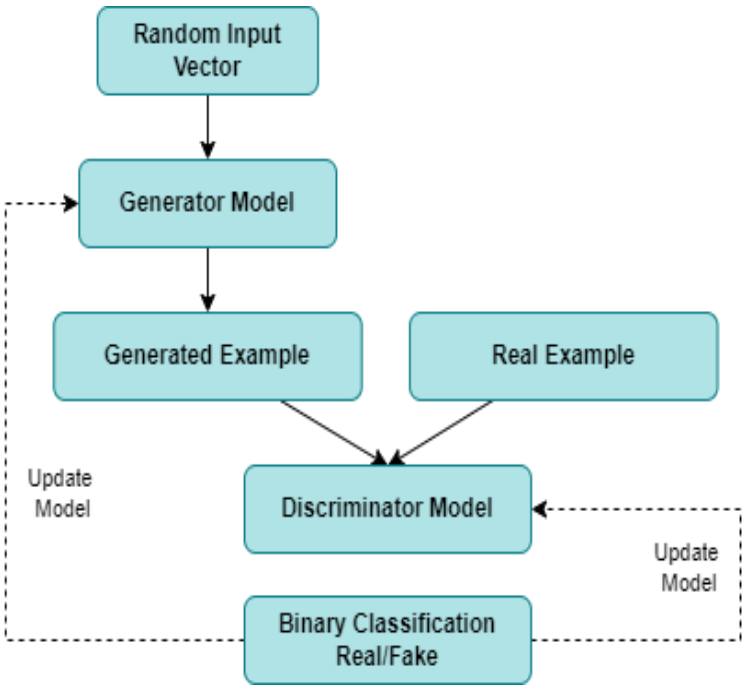


Figure 29 - GAN Model Architecture

3.3.2 GAN Training Algorithm

The GAN training process consists in the training of both discriminator and generator model according to the GAN adversarial game above-mentioned. The algorithm, which allows the network to be trained, is summarized in Algorithm 1. The algorithm is taken from the original paper "Generative Adversarial Networks" of Goodfellow et al [21].

Algorithm 1: Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

The discriminator is trained in order to correctly classify real and fake images. In particular, this is possible maximizing the log of the probability of predicting real images and the log of the inverted probability of fake images, averaged at each batch of samples. So, the aim of the discriminator training is to associate a probability close to 1.0 for real images and a probability close to 0.0 for generated images.

Instead, the generator loss is defined as minimizing the log of the inverted probability of the discriminator's prediction of fake images, averaged at each batch.

According to the authors of the original paper "Generative Adversarial Networks" [21] the loss function of the generator creates some problems. In particular, when the discriminator is good to reject fake images and the generator is not capable to generate adequate images, the generator loss saturates. This means that the loss function doesn't give good gradient information in order to improve the generator weights. Rather than

training G to minimize $\log(1 - D(G(z)))$ they propose to train G to maximize the $\log D(G(z))$. This objective function allows us to have a stronger gradient.

3.3.3 GAN and Convolutional Neural Networks

GAN networks are very often used in image-synthesis-based task. Examples of applications, where GAN networks are employed, are image generation, image-to-image translation and image manipulation [41] which are typical computer vision tasks.
We have seen how Convolutional Neural Networks (CNNs), described in the section 3.2, have become the standard networks in image domain, because are designed to work with grid-structured inputs, which have strong spatial dependencies in local regions of the grid (e.g. 2-dimensional image).
For this reason, in image domain, the GAN networks architecture are typically developed using as generator and discriminator two Deep Convolutional Neural Networks (DCNNs). This kind of architecture was first presented by Alec Radford et al. [42] and is called Deep Convolutional Generative Adversarial Networks (DCGAN). Their work represents one of the most important steps forward in the design and training of stable generative adversarial models.
Considering an image classification task the discriminator takes as input an image which is down sampled until we have as output a binary classification (real/fake). Typically, traditional DCNN uses pooling layers to down sample input image and feature maps. In DCGAN architecture is recommended to downsample using stride convolutions. Convolutional layers can perform a down sampling applying each filter across the input images or input feature maps (in case of hidden layers) using a stride of two instead of the default equal to one. The output of the convolutional layer is a feature map that is smaller with respect to the input. We can use the padding technique, in order not to lose information about the border of the feature map.

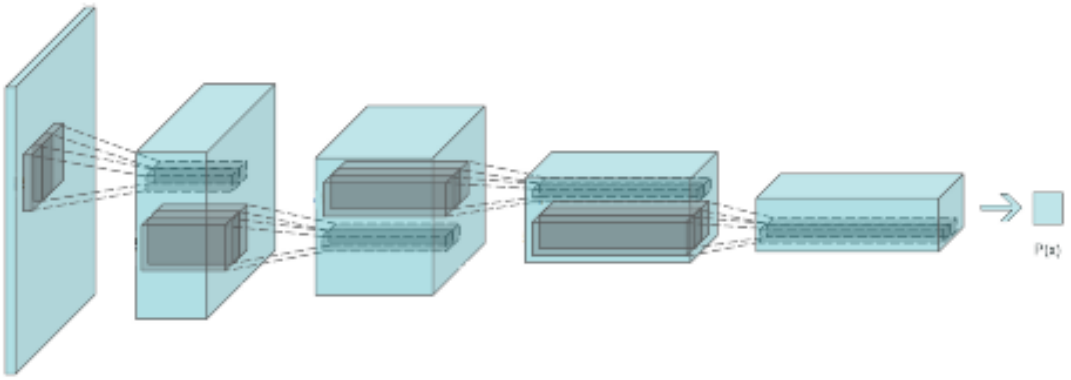


Figure 30 - Example of the discriminator model architecture for the DCGAN

From the other hand, the generator has to synthesize an output image given random input vector designed from the latent space. To achieve this, the generator requires an inverse operation with respect the discriminator model. It needs an up-sampling operation in order to associate to the raw input the detailed image output (generated image). We can obtain this using transpose convolutional layer which performs an inverse convolution operation which can map features to pixels.

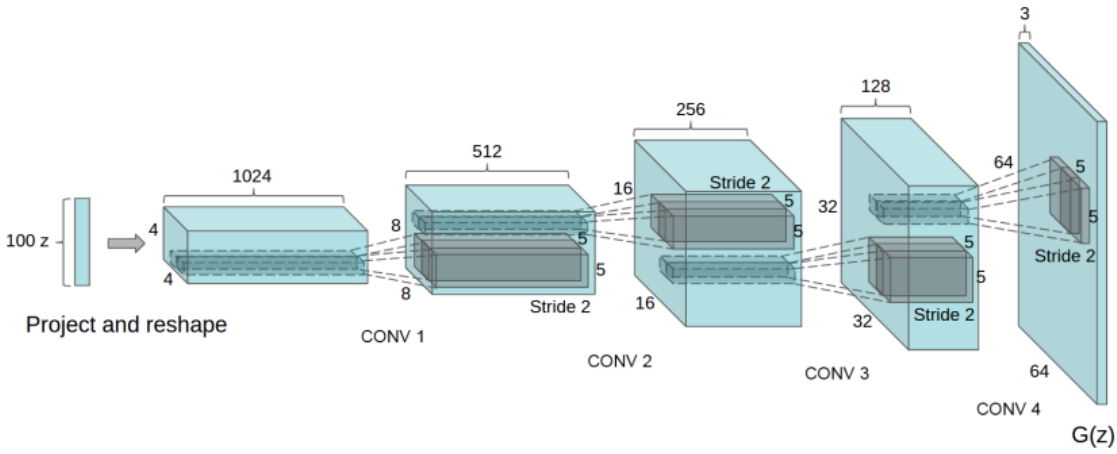


Figure 31 – Example of the generator model architecture for the DCGAN

3.3.3.1 Batch Normalization

One common problem in deep neural networks is what is known as *exploding gradient*. During the training phase, the network's weights are updated based on the error gradient (see section 3.1.2). Since, in the backward phase, the errors are propagated backward through the network, it can happen that the gradient computation grows exponentially

causing large updates to the network weights. Exploding gradients can result in an unstable network that sometimes cause NaN weight values [43].

Usually the problem does not happen immediately, but the training of the network can start and only after some training iterations can result in the exploding of the gradient. Typically, the network's input data are scaled allowing a stable start over the first few iterations. If we used unscaled input, since the weights of the network are initially selected in a random way, we can have huge activation values that results in exploding gradient problem. Indeed, it is recommended scaling the image pixel values from the range [0, 255] to the range [-1,1]. Scaling input does not guarantee that the activations of all following layers are well scaled as well (covariate shift).

A solution that drastically reduces the exploding gradient problem, stabilizing the training process, is the Batch Normalization (BN) introduced by Sergey Ioffe et al. [44]. Specifically a BN layer, at each batch, evaluates the mean and standard deviation of each of its input channels and normalizes subtracting the mean and dividing by the standard deviation. So, it standardizes the activations value from a prior layer to have zero mean and unit variance. For each channel in the preceding layer, the BN layer has two trainable parameters, gamma and beta that defines the scale and the shift, respectively. Algorithm 2 shown the BN process.

Algorithm 2 [44]: Batch Normalizing Transform, applied to activation x over a mini-batch.

Input: Values of x over a mini-batch:
$\mathcal{B} = \{x_{1\dots m}\}$ Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // mini-batch mean

$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$ // mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$ // normalize

$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$ // scale and shift

The BN layer can be place after dense or convolutional layers in order to normalize the output of these layers.

3.3.3.2 Dropouts Layers

We have seen in the section 3.2.4 how it is important that a successful machine learning algorithm does not suffer from overfitting. It must perform well both with training dataset and with test dataset. To counteract this problem, we have seen that a possible solution is data augmentation. However, there are several regularization techniques, which ensure to penalize the model if it starts to overfitting.

One of the most common technique, is using dropout layers, introduced by Geoffrey Hinton in 2012 and presented by Nitish Srivastava et al. [45] in 2014.

The idea behind this approach is very simple. Each dropout layer randomly select a set of computational units of the prior layer and set their output to zero.

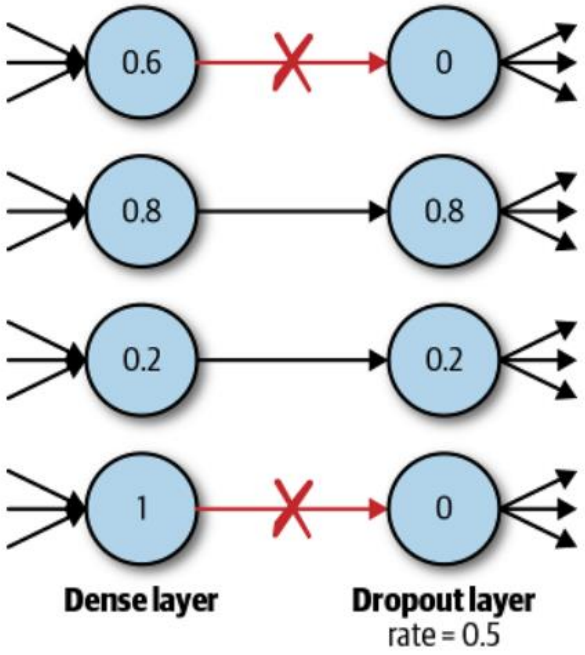


Figure 32 - Dropout Layer [37]

This simple operation allows the network to not be strictly dependent on certain units or groups of units. This makes the model more general when it deals with unseen data drastically reducing the overfitting problem.

In the dropout layer we have not trainable parameters, but we can set the portion of computational units to drop from the preceding layer specifying the dropout rate. For instance, in Fig. 32 we have a rate of 0.5.

3.3.3.3 ReLU and Leaky ReLU

Generally, in deep convolutional neural networks, it has become a best practice using the rectified linear activation unit, ReLU. It is a simple operation that given $L_q \times B_q \times d_q$ feature values in a layer returns $L_q \times B_q \times d_q$ activation values, which are equal to the value provided as input directly, or the value 0.0 if the input is 0.0 or less. The ReLU activation, typically, follows the convolution operation. A. Krizhevsky et al. [46] show how the use of ReLU with respect to other activation functions, like sigmoid and tanh, has advantages in terms of training's speed and accuracy.

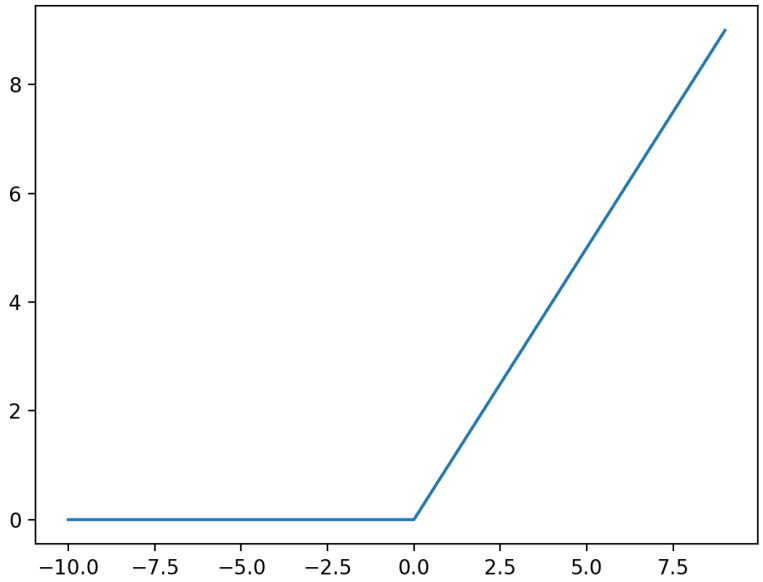


Figure 33 - ReLU function

In generative adversarial networks, a best practice, is using a variation of the ReLU called Leaky ReLU which has a small slope for negative values instead of a flat slope [42]. The slope coefficient is specified before training, and it is not a trainable parameter.

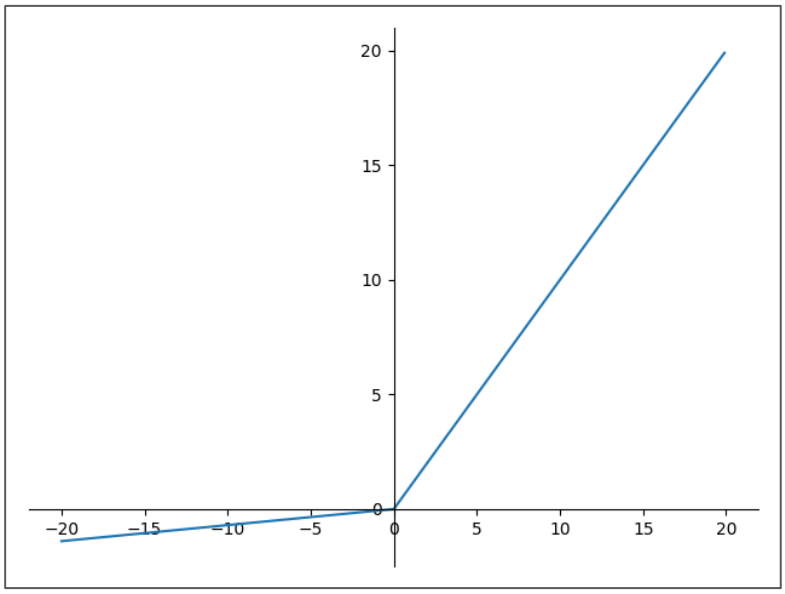


Figure 34 - Leaky ReLU Function

In DCGAN network [42] the generator model has ReLU activations, for all layers expect for the output, which uses Tanh whereas the discriminator adopts Leaky ReLU activations for all layers.

3.3.4 GAN based Image-to-Image Translation

The Image-to-image translation task is a subfield of computer vision, which aims to learn the mapping between an input image and output image. Specifically, the image from a source domain X is translated to a target domain Y. A wide range of problems in computer vision can be treated as image-to-image translation tasks [47][48][49].

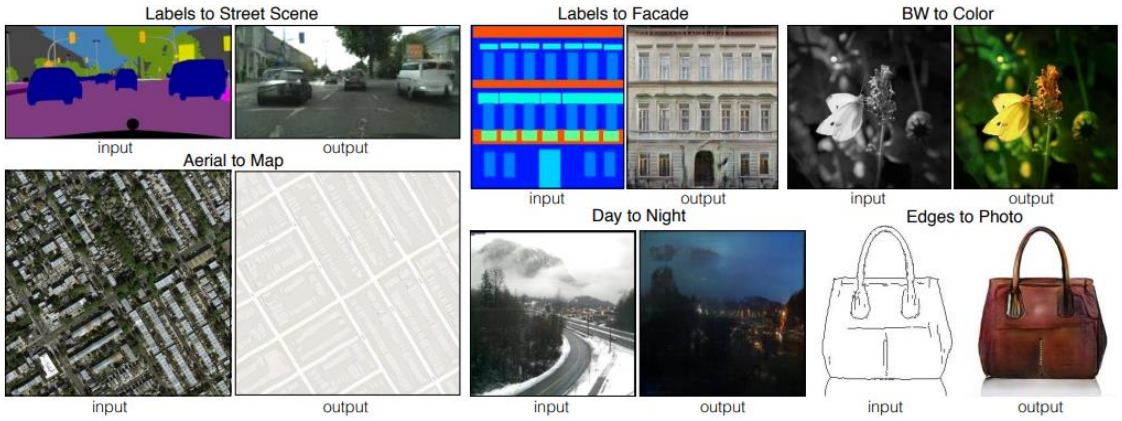


Figure 35 - Computer Vision problems which involves translating an input image into a corresponding output image [47]

The image-to-image translation task using GAN networks requires to use conditional version of GAN (CGAN) whereas, unlike the GAN model above-mentioned, the input to the network is an image instead of random vector.

3.3.4.1 Conditional GAN (CGAN)

The Generative Adversarial Networks are architectures which have as objective the training of a generative model which must be capable to generate images. In order to have a complete control on the generation we should understand the complex relationship between the latent space, the generator and generated images. Conditional GANs allows us to have more control on the generation of the images. In particular, they permit to conditionate the generation of the images given a certain input. In order to do this the generator model is modified adding an additional input (for example the class label) allowing the generation of images of a given type.

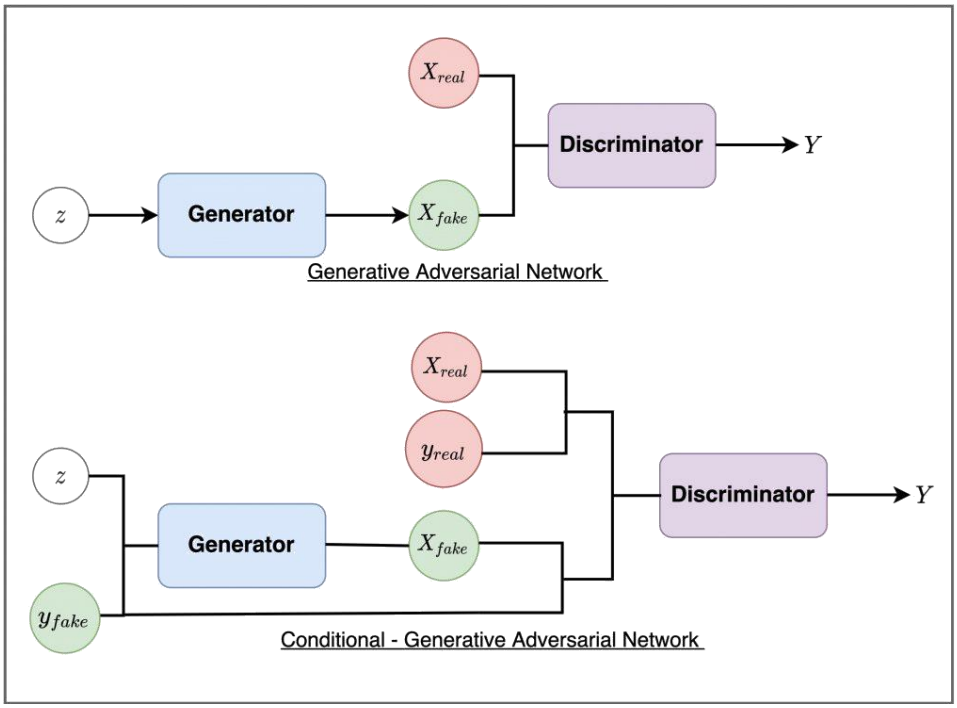


Figure 36 - Difference between original GAN architecture and CGAN architecture

3.3.4.2 GAN Pix2Pix

A common approach used for accomplishing the image-to-image task is the Pix2Pix GAN architecture, introduced by Philip Isola et al [47] in 2018, which is a particular kind of conditional generative adversarial network. The main difference with respect to the CGAN is that the generation of the image is conditioned by a given input image and the loss function is modified in order to have a generated image that is a plausible translation of the input image. In particular, the generator is trained in order to fool the discriminator and minimize the loss between the generated image and the expected target image.

It is worth noting that the GAN Pix2Pix translates the image from one domain to another using a supervised approach, because the generated image are compared to ground truth images contained in the training dataset and a loss is generated by the comparison.

3.3.4.2.1 Generator U-Net Model

Pix2pix architecture uses U-Net, as generator model, which is a revisitation of the common encoder-decoder network. In particular, in a common encoder-decoder model, the encoder takes an image as input and down samples it until a bottleneck layer and the decoder up samples again until it obtains as output the final image of specific size. In the encoder-decoder the bottleneck layer is the lower dimension layer which represents the lower-dimensional representation of the input.

The U-Net model, as encoder-decoder model, uses down sampling layers until a bottleneck layer and up sampling blocks and add a linkage between layers of the encoder and decoder part which have the same size. In order to create the skip-connections the model makes concatenation of same size features maps, along the channel dimension.

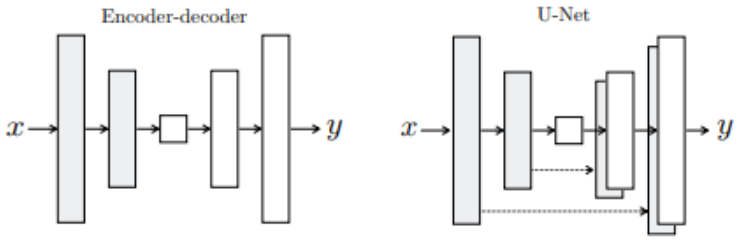


Figure 37 - Encoder-decoder and U-Net [47]

In the Pix2Pix paper [47] the U-Net generator architecture is defined using the notation:

- Encoder: C64-C128-C256-C512-C512-C512-C512-C512
- Decoder: CD512-CD1024-CD1024-C1024-C1024-C512-C256-C128

Where Ck denotes a Convolution – BN – Leaky ReLU layer with k filters and CDk denotes Convolution – BN – Dropout – Leaky ReLU layer with dropout rate of 0.5. In the Convolution layer the kernel size is fixed at 4x4 and the stride at 2. The last layer of the encoder (bottleneck layer) does not use BN and uses ReLU instead of Leaky ReLU.

3.3.4.2.2 Patch-GAN Discriminator Model

PIx2Pix architecture uses Patch-GAN which is a revisitation of the common discriminator model uses by GAN. In particular it is a particular deep convolutional network which aims to classify patches of the input as real or fake instead of classifying the whole input image as real or fake. The output of the network is a simple feature map of a given dimension which contains real/fake predictions, which can be averaged in order to obtain a unique score.

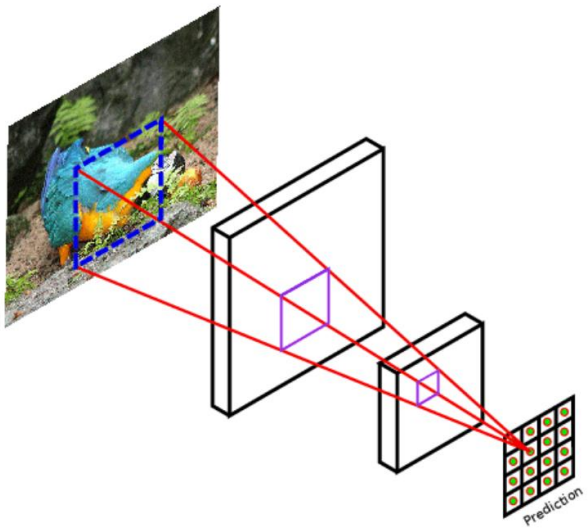


Figure 38 - Patch GAN output feature map

In the Pix2Pix paper [47] the Patch-GAN architecture is defined using the notation: C64-C128-C256-C512, where C stands for a block composed by Convolution – BN – Leaky ReLU layers whereas the first C64 layer does not use BN and the number represent the number of filters. In the Convolution layer the kernel size is fixed at 4x4

and the stride at 2. The slope of the Leaky ReLU is fixed to 0.2 and to obtain a binary (real/fake) output matrix a sigmoid activation function is used to the output layer.

3.3.4.2.3 Modified Loss Functions and training

The training phase of the discriminator model is almost the same above-mentioned (section 3.3.2) . The only modification that we have is that the discriminator loss is halved in order to make the training of the discriminator less fast compared to the generator.

The generator loss is enriched with additional terms with respect the adversarial loss. This term is the L1 distance which is the mean absolute pixel difference between the generated translation of the image input and the expected target image. In the paper of the original Pix2Pix Image-translation the author explains how using L1 loss in place of L2 loss encourages less blurring.

Specifically D and G play the following the new two-player minmax game [47]:

$$G^* = \arg \min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \qquad \mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\| y - G(x, z) \|_1]$$

Where λ is an hyperparameter which quantify the importance of L1 loss with respect to the adversarial loss in training phase (typical values are 10 or 100).

3.3.4.2.4 Adam Stochastic Gradient Descent

In the Pix2pix paper [47] the author proposes to use for the optimization a specific algorithm called Adam Stochastic Gradient descent [50].

The Adam SGD is an optimization algorithm which can be used instead of the classic SGD in order to update and optimize the weights of the networks at each training iteration, based on the training data.

The authors of the original paper of Adam SGD [50] specifies a series of advantages which respect a classic approach:

"*The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is*

also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning " [50]

With respect to the classic SGD, which maintains, during training, a fixed single learning rate for all network weights (parameters), the Adam Optimizer maintains a specific learning rate for each network weight, which can be separately adapted during training phase. As mentioned in the paper the algorithm combines the advantages of two existing approach: AdaGrad and RMSProp optimizers, whereas the first works well with sparse gradients and the second works well in online and non-stationary problems.

Algorithm 3 [50]: Adam algorithm for stochastic optimization. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power of t

---

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Specifically, the algorithm evaluates exponential moving averages:

- mt is the exponential moving average of the gradient gt
- vt is the exponential moving average of the gradient gt^2

where the hyper-parameters β1, β2 ∈ [0, 1) control the exponential decay rates of these moving averages.

The moving averages are estimates of the 1st moment (the mean) and the 2nd raw moment (the uncentered variance) of the gradient where $\widehat{mt}$ and $\widehat{vt}$ are bias-corrected estimates.

At the end of the algorithm the parameters (weights) are updated using the two bias-corrected estimates and α which is referred as step size or learning rate which defines the proportion with which the weights are updated. Smaller value corresponds to more slow learning during training.

The author of the paper [50] shows the effectiveness of Adam solver in deep convolutional networks. He shows Adam performance with CIFAR-10 image recognition dataset:

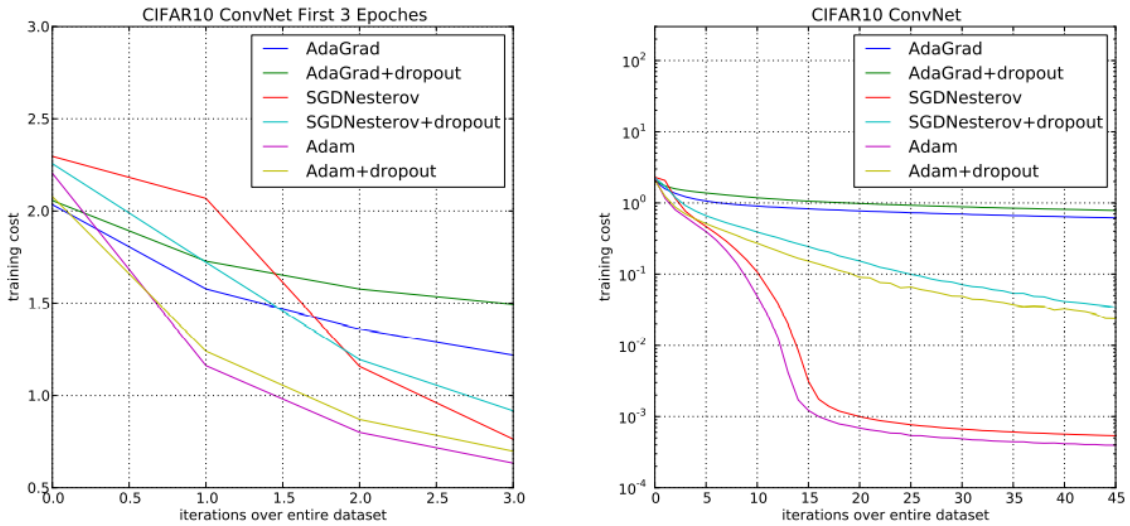


Figure 39 - Adam Solver Performances(Convolutional neural networks training cost. (left) Training cost for the first three epochs. (right) Training cost over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture.)

In the initial iterations, both Adam and Adagrad quickly reduce the training cost. However, Adam and SGD eventually converge considerably faster than Adagrad (Fig. 39)

3.3.5 GAN metrics: Fréchet Inception Distance (FID)

In image-to-image translation problem, implemented trough GAN network, two widely adopted metrics for the quantitative evaluation of the quality of the generated images are the Inception Score (IS) and the Fréchet Inception Distance (FID). The FID is an improvement of IS and it has been shown as FID is consistent with human judgments and is more robust to noise with respect to the IS [51].

We adopt FID as metrics for the evaluation of the quality of the GAN network which allows us to compare the distribution of generated images with the distribution of target real images.

The FID, introduced by Heusel et al. [52], is the squared Wasserstein metric evaluated between two multidimensional Gaussian distributions. In particular the first, is the distribution of some features of the real images and the second is the distribution of some features of the generated images. These features are obtained when the generated images and the target images are fed into Inception V3 network.

$$\mathrm{FID} = \|\mu - \mu_w\|_2^2 + \mathrm{tr}\left(\Sigma + \Sigma_w - 2(\Sigma\Sigma_w)^{1/2}\right)$$

Where the mean and the covariance are evaluated on the activations obtained when we fed the generated and real images into the Inception V3 network.

Lower FID means smaller distances between generated and real data distributions.

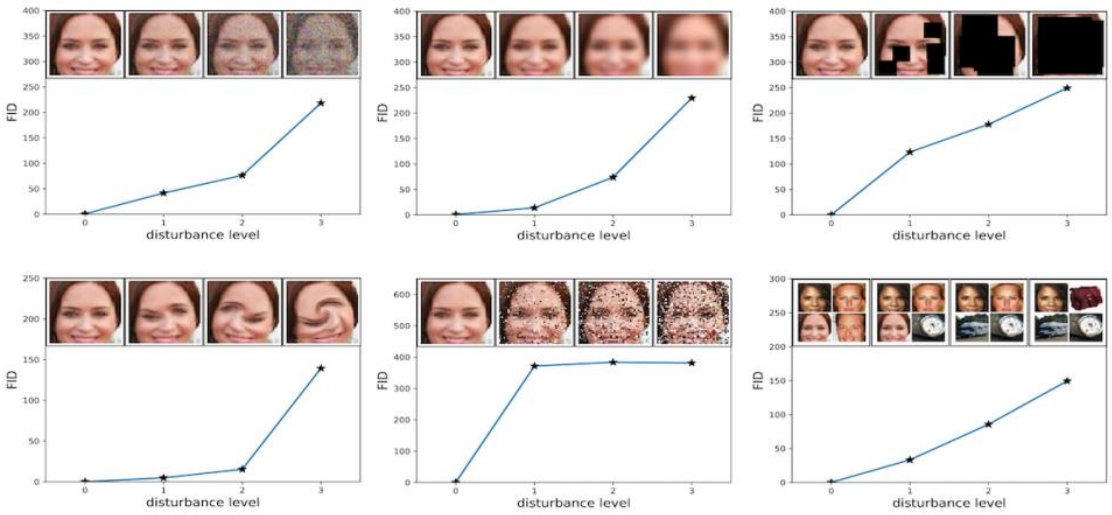


Figure 40 - FID measure sensitive to image distortions [52]

Chapter 4

GAN-based Path Planning

In this chapter, we present the proposed GAN architecture adopted to solve the path planning problem. The problem has been treated as an image-to-image translation task which consists in associating to the image which contains the environment map an image which contains the corresponding feasible path. The model is trained using a large number of samples which contains environment map, start and goal node and corresponding feasible paths which are generated by the well-known algorithm A*. After the training, the model given the input (environment map and start and goal node) returns a plausible translation, which represents the path to follow, whereas the environment is modelled using two-dimensional grid map image which is a suitable input for convolutional neural networks.

4.1 Problem statement

We consider the environment as two-dimensional grid which is composed of obstacle cells (occupied cells), free cells and start and goal cells. The generated path on the grid is a sequence of adjacent free cells which connect start and goal cells. The path planner aims to find a path given a start and goal locations trying to minimize the path length.
Formally, the grid map can be defined as a two-dimensional matrix $A \in \mathbb{R}^{mxn}$, where the occupied cell is indicated with element $a_{ij} = 1$ and the free cell with the element $a_{ij} = 0$. The path planning aims to find a feasible path, π, which is a sequence of feasible moves $e_v = (a_{ij}, a_{kl})$ between adjacent cells, where $a_{ij}, a_{kl} \in$ free cells, which connects the initial cell with the goal cell.

4.2 The A* algorithm

The A* algorithm represents one of the most applied global path planning algorithm, which is an extension of the Dijkstra's algorithm [53], that allows to reduce the Dijkstra's search space using a heuristic function to estimate the cost from any node to the goal node. In particular, the Dijkstra's algorithm searches for the goal cell in all direction. This approach could be efficient if we deal with complex environment, but it becomes very inefficient when the optimal path is very simple, like a straight line to the goal cell.

Considering a generic node n, the A* algorithm follows the best-first approach, which consists of assigning a cost of action at node n by means of an evaluation function $f(n)$ and selecting the action corresponding to the best value. The algorithm considers not only the number of step from the start node, like Dijkstra, but also a heuristic function which gives an indication of the preferred direction to search (towards the goal node). Formally, the evaluation function $f(n)$ is defined as:

$$f(n) = g(n) + h(n) \tag{4.1}$$

where $g(n)$ is the cost from the starting node to the node n and $h(n)$ is the heuristic estimation of the cost from node n to the goal node.

Since the A* is a graph-search algorithm and has to operate with grid maps, which are discretization of the real world, we can define two version of A* which depends on the number of movement allowed to the agent:

- A* with 4-connected graph where 4 movements are allowed along the cartesian directions
- A* with 8-connected graph where 8 movements are allowed along the cartesian directions and diagonal directions.

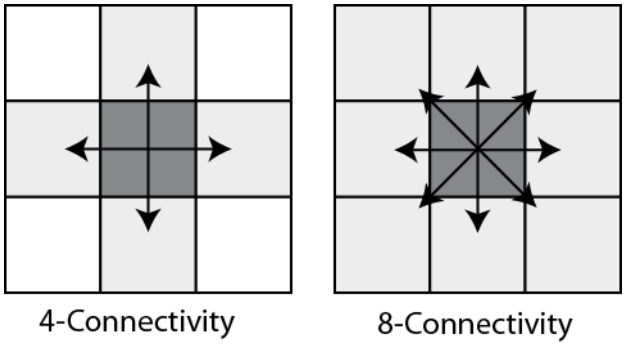


Figure 41 – Allowed movement in 4-connected and 8-connected

In the first case the heuristic function is evaluated using Manhattan distance which is defined as follows:

$$Manhattan(n) = D \cdot (d_x + d_y) \tag{4.2}$$

$$d_x = |n_x - g_x|, d_y = |n_y - g_y| \tag{4.3}$$

where:

- n_x, n_y are the coordinates of the node n
- g_x, g_y are the coordinates of the goal node
- D is a cost parameter associated to cartesian movement between adjacent cells

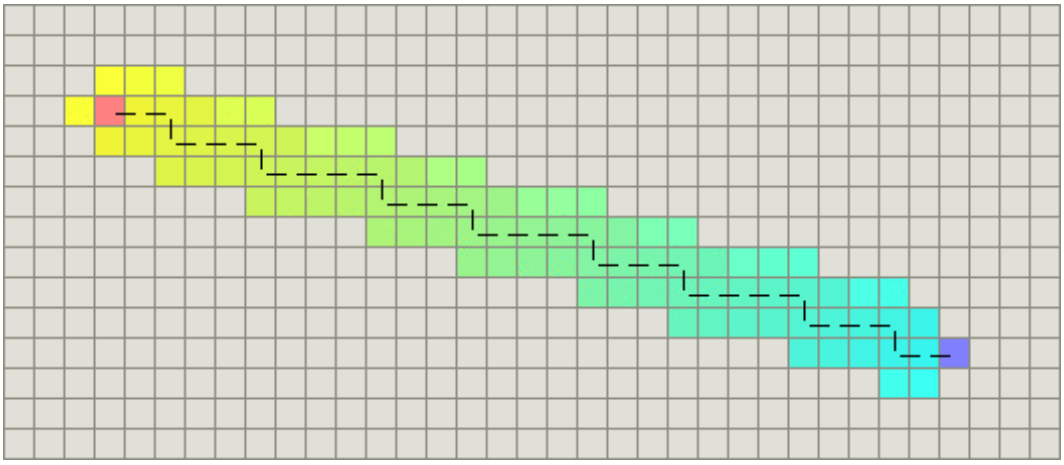


Figure 42 - Example of 4-connected with Manhattan Distance (simplest case without obstacles) [54]

In the second case the heuristic is evaluated using diagonal distance which is defined as follows:

$$diagonal(n) = D \cdot (d_x + d_y) + (D_1 - 2 \cdot D)\min(d_x, d_y) \tag{4.4}$$

where D1 is a cost parameter associated to the diagonal movement.

If $D = 1$ and $D_1 = 1$ the diagonal distance is known as Chebyshev distance whereas if $D = 1$ and $D_1 = \sqrt{2}$ the diagonal distance is known as Octile distance.

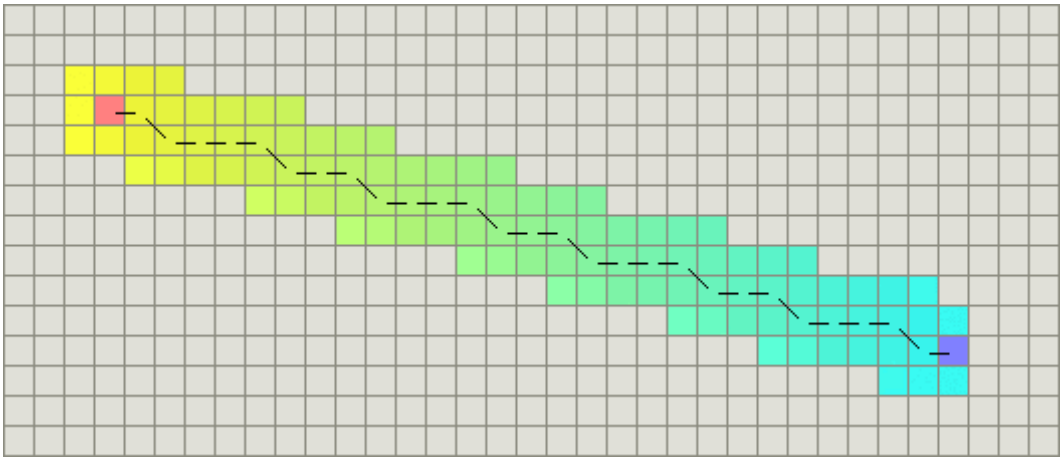


Figure 43 - Example of 8-connected with Octile Distance (simplest case without obstacles) [54]

An alternative to diagonal distance could be the Euclidean distance, which is shorter than diagonal distance. This means that the obtained paths will be the shortest, but A* computation time increases [54].

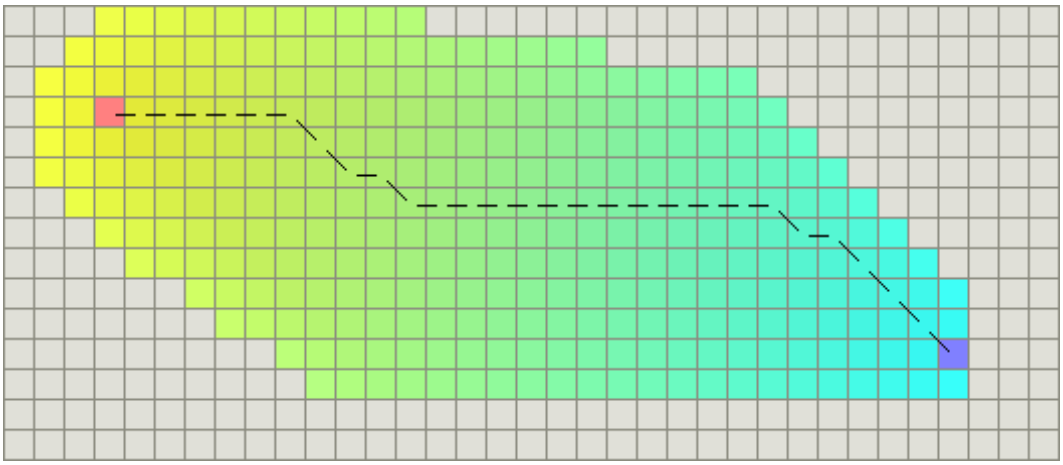


Figure 44 - Example of 8-connected with Euclidean Distance (simplest case without obstacles) [54]

Euclidean distance which is defined as follows:

$$Euclidean(n) = D \cdot \sqrt{\left(d_x{}^2 + d_y{}^2\right)} \tag{4.4}$$

Assuming that S and G are the start and goal node, respectively and $f(n)$ is the evaluation function above mentioned, the A* algorithm, in its conventional form [10], is defined as follows:

Algorithm 4: A* algorithm

```
procedure A_star(S, G)
    OpenSet ← S
    f(S) = g(S) + h(S)
    t ← 0
    while OpenSet ! = ∅ do
        n_next = n_min ∈ OpenSet s.t. f^(t)(n_min) < f^(t)(n_j) ∀j = 1, … , N
        if n_next = G then
            Terminate()
        else
            ClosedSet ← n_next
            start execute expand (n_next)
            for all m ∈ ADJ − NODES(n_next) do
                f^(t)(m) = g^(t)(m) + h^(t)(m)
                if m ∉ ClosedSet then
                    OpenSet ← m
                else if f^(t)(m) < f^(t_i)(m) then
                    OpenSet ← m
                end if
            end for
            end execute expand (n_next)
        end if
        t ← t + 1
    end while
end procedure
```

4.3 Grid-Image Transformation

The implemented GAN network solves the path planning problem treating it as an image-to-image translation task. So, being a network which works with images, we need to map the grid information into image data. We can distinguish 3 classes of grid cells, as mentioned in 4.1 – free cells, occupied cells (obstacles), start and goal cells and path cells. So we map free cells with white pixels (0), occupied cells with black pixels, start cell with red pixel, goal cell with blue pixel and path cells with green pixels. In Fig. 45 we have an example of grid-image transformation for a grid map which contains obstacles, start and goal node and path.

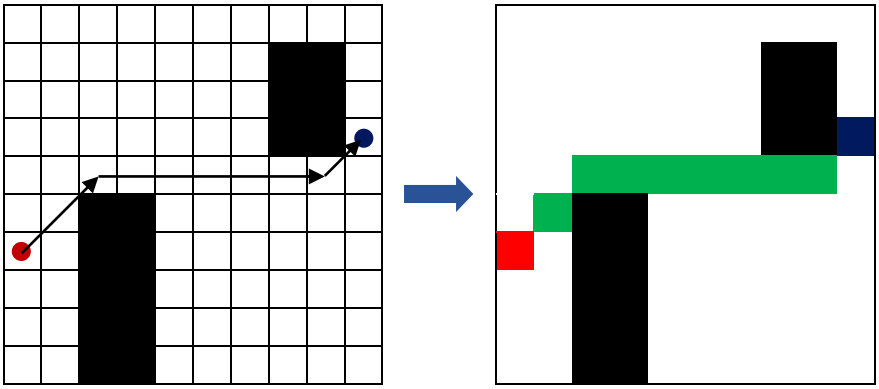


Figure 45 - Grid-Image Transformation for a 10x10 grid map

4.4 Dataset Generation

The dataset consists of a RGB images of dimension 64x64 pixels:

- for the generation of environment maps we use, instead of coordinates, image maps of dimension 64x64 pixels which are randomly generated. The environment is a grid map where the obstacles are denoted with black pixels and the free spaces with white pixels.
- for each map, we generate a certain number of pairs of start and goal nodes, randomly chosen from the free space. The start node is denoted with red pixel whereas the goal node is denoted with blue pixel.
- in order to generate the ground truth for training and validation set, we generate a feasible path, using A* algorithm, for each map and corresponding start and goal nodes. The path is denoted with green pixels.

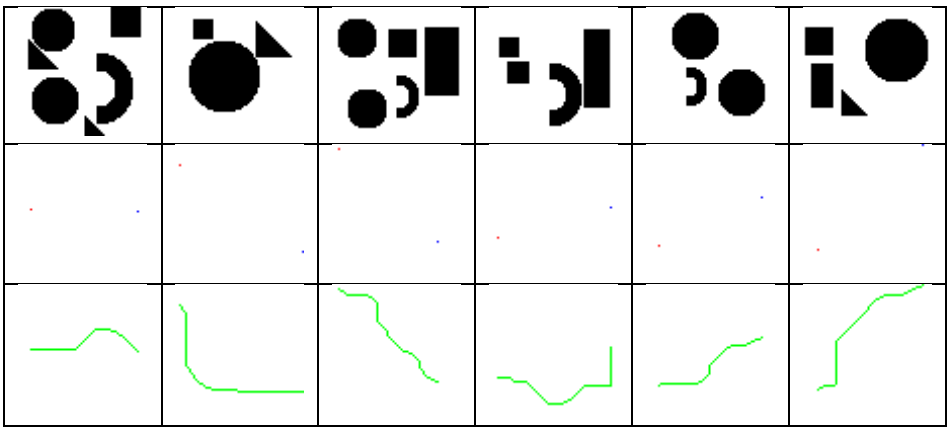


Figure 46 – An example of dataset. For each column, from the top to the bottom, each element represents environment maps, pair of start and goal node and feasible path generated by A algorithm*

4.5 Overall GAN-based Path Planning Architecture

Fig. 47 shows the initial overall architecture of GAN model used for the path planning generation and Fig. 48 and Fig. 49 shows, respectively, the detailed structure of the discriminator and generator.

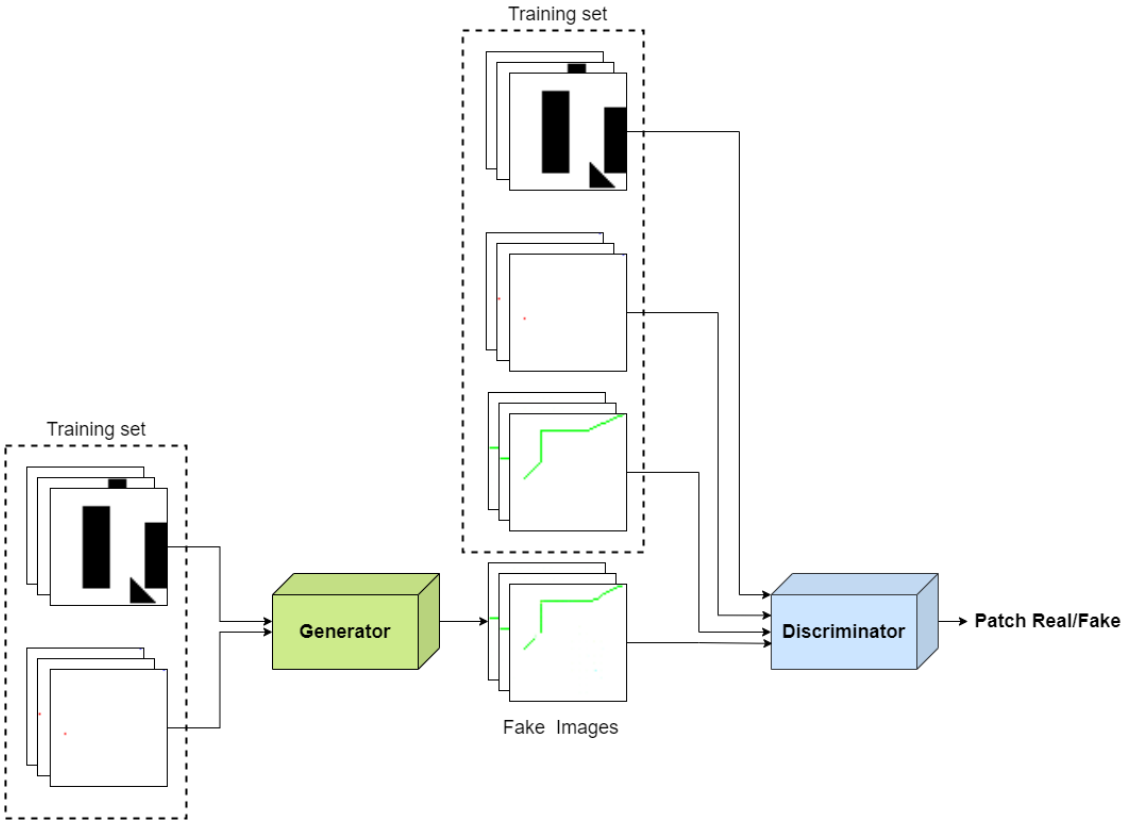


Figure 47 - Overall Architecture of the GAN network for path planning

In our application, we refer to: *S* as the path images space, *M* as the map images space, *P* as the point images space.

In the discriminator the inputs are 64x64x3 dimensional map image *M*, 64x64x3 dimensional points image *P* and 64x64x3 dimensional target path image S, which can be fed by the generator (fake target) or by training set (true target). The output is a 6x6 feature map which contains the fake/true prediction of specific patch of the input.

So, the discriminator architecture is similar to the Pix2Pix model. Specifically:

- the three inputs are concatenated, along channel dimension, and fed into the first block which is a Convolution – Leaky ReLU layer where the convolution uses 4x4 filters with stride of 2 and the slope of the Leaky ReLU is set to 0.2

- from the second to the fourth block we have Convolution – BN – Leaky ReLU layers. In the first two layers the convolution layer uses 4x4 filter with stride of 2 and in the final layer the convolution uses 4x4 filters with stride of 1.
- the final block is a simple Convolution layer, with 4x4 filters and stride of 1, which output is the patch feature map.

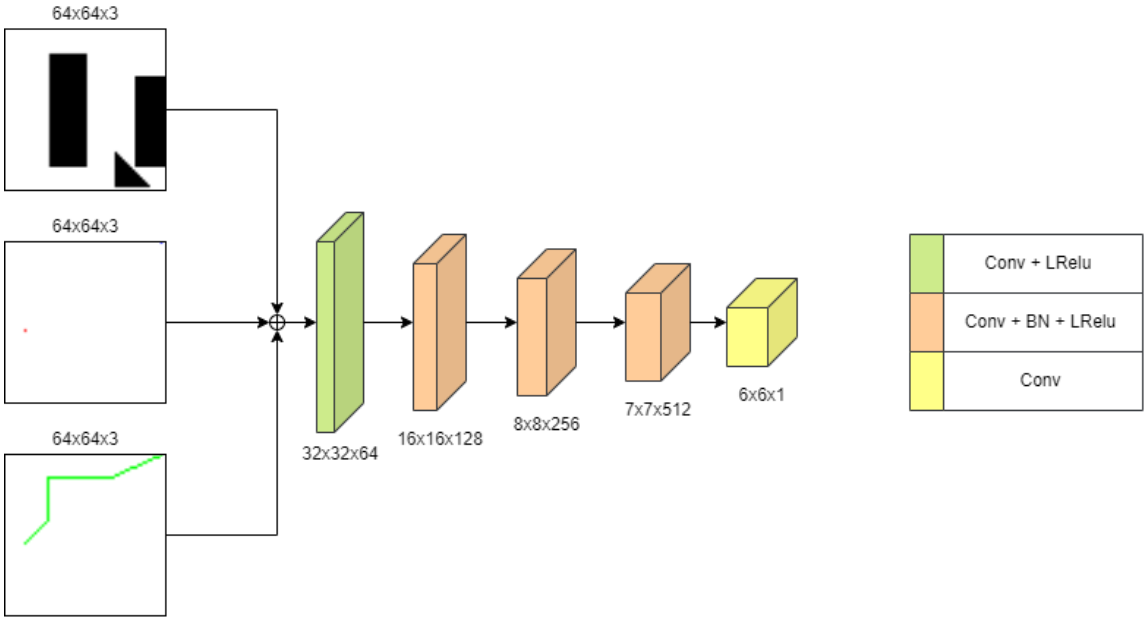


Figure 48 - Discriminator Architecture (to redo)

In the generator the inputs are 64x64x3 dimensional map image *M* and 64x64x3 dimensional points image *P*. The output is 64x64x3 dimensional image which represents the generated path *S*.

The generator obeys to encoder-decoder architecture:

- The encoder aims to extract information about the obstacles location and start and goal positions
- The decoder aims to reconstruct a feasible path, using the information about the environment and start and goal locations.

As in the Pix2Pix architecture, the generator uses U-Net model whereas the skip-connections between same-size encoder and decoder layers are obtained concatenating, along the channel dimension, the output feature map of the encoder layer with the output feature map of the same-size decoder layer.

So, M and P are fed into a 32-channel convolutional layer, respectively. The feature maps obtained are concatenated together along the channel dimension obtaining a

32x32x64 feature map which is processed by subsequent layers which encode and decode obtaining the generated image.

The encoding network architecture is structures as follows:

- the block 1 consists of two layers which are Convolution-LeakyRelu layer with 4x4 filters, stride of 2 and Leaky ReLU slope equal to 0.2.
- the blocks 2 ÷ 4 are Convolution-BatchNormalization-LeakyRelu layers with 4x4 filters, stride of 2 and Leaky ReLU slope equal to 0.2
- the bottleneck layer is a Convolutional-Relu layer with 4x4 filters and stride of 2

The decoding network architecture is structures as follows:

- the blocks 5 ÷ 8 are Deconvolution-BatchNormalization-LeakyRelu layers with 4x4 filters and stride of 2.
- the final block is a Deconvolution-Tanh layer with 4x4 filters and stride of 2. This block compresses the output feature map of the block 8 into 3-channel feature map and activates it by Tanh function obtaining the final generated image.

If we denote the output feature maps from block 1 to block 4 as I_1, I_2, I_3, I_4 and the output feature maps from block 5 to 8 as I_5, I_6, I_7, I_8 we create skip connections concatenating, along the channel I_1 and I_8, I_2 and I_7, I_3 and I_6, I_4 and I_5.

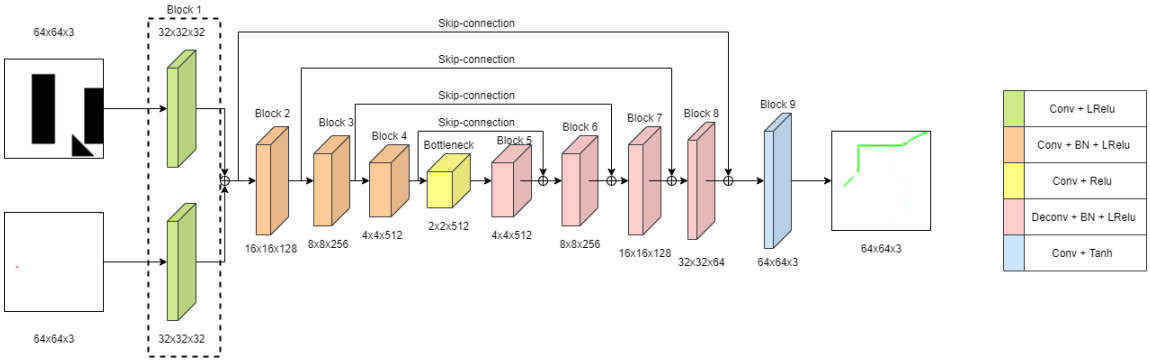


Figure 49 - Generator architecture

4.6 Training algorithm

The network is trained according to the GAN adversarial game using as loss function those used in the original Pix2Pix paper. The algorithm, which allows the network to be trained is the following one.

Algorithm 5: Training algorithm where θ_D and θ_G are the weights of the discriminator and generator respectively and m is the batch size

for Number of training iterations **do**

  for k iterations **do**

    sample $P_{S,\mathcal{M},\mathcal{P}}(s, m, p)$ to get m random samples $\{s_i \dots s_m\}$, $\{m_i \dots m_m\}$, $\{p_i \dots p_m\}$

    Calculate the discriminator error:

$$J_D = -\frac{1}{2m}\left(\sum_{i=1}^{m} \log \mathrm{D}(s_i, m_i, p_i) + \sum_{i=1}^{m} \log\left(1 - D(G(m_i, p_i), m_i, p_i)\right)\right)$$

    Update discriminator parameters using Adam algorithm

    Calculate the generator error:

$$J_G = -\frac{1}{m}\left(\sum_{i=1}^{m} log\ \ D(G(m_i, p_i), m_i, p_i) + \sum_{i=1}^{m} \lambda * \mathcal{L}_{L1}\left(G(m_i, p_i)\right)\right)$$

    Update generator parameters using Adam algorithm

  end

end

Chapter 5

GAN Path Planning Experimental Results

5.2 Training details

For the training and validation of the GAN network we use two datasets of increasing dimensions. The first one is composed by 44500 sets of data whereas we randomly choose, among them, 33000 for the training phase and 11500 for the validation phase. The second one is composed by 110500 sets of data where 99000 are randomly chosen for the training set and 11500 for the validation set.

Table 1 – Summary of the dataset used for the experiments

Dataset ID	**Training Size**	**Validation Size**	**Total Size**	**Dataset Type**
Dataset_01	33000	11500	44500	Original (Fig. 46)
Dataset_02	99000	11500	110500	Original (Fig. 46)
Dataset_03	99000	11500	110500	Modified (Fig. 65)

In order to validate our architecture, we have conducted different training process:

- 50 epochs using for the training and validation phase the smaller dataset
- 150 epochs using for the training and validation phase the smaller dataset
- 50 epochs using for the training and validation phase the bigger dataset
- 50 epochs using for the training and validation phase a modified dataset

Afterwards, we have slightly modified the overall architecture seeking performance improvements.

For the training phase, we used as stochastic gradient descent optimization algorithm the one used in the original Pix-2-Pix architecture [47] which is the Adam optimizer

with parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$ with learning rate of 0.00001 and 0.0001 for the discriminator and generator model, respectively.

5.3 Success metrics

For the quantitative evaluation of the quality of the generated images we have adopted Fréchet Inception Distance (FID) mentioned in 3.3.5.

In our case, we have used the FID metric in two different ways:

- during training, at each epoch, we adopt FID to compare the whole validation ground truth distribution (distribution of target path) with whole distribution generated by the generator model (distribution of generated path), pulling out a unique score. So, we obtain an overall FID for each epoch.
- at the final epoch, we adopt FID to compare each pair of real target and corresponding generated target. In this case we obtain a specific score for each pair of them.

The first way allows us to understand if the network is working in the right way. If the overall FID decreases at each epoch the overall quality of the generated images is improving. While, in the second way we understand, at the final epoch, what is the percentage of path correctly generated with respect the whole validation dataset. From this moment we will refer to the first FID as "Overall FID" (OFID) and to the second as "particular FID" (PFID)

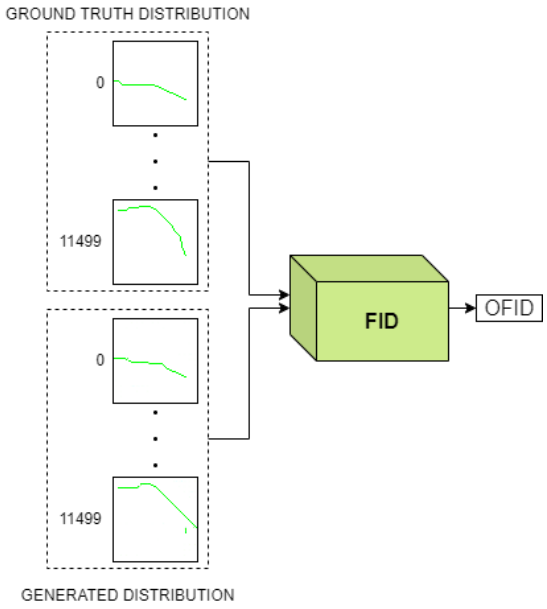


Figure 50 - Overall FID (OFID)

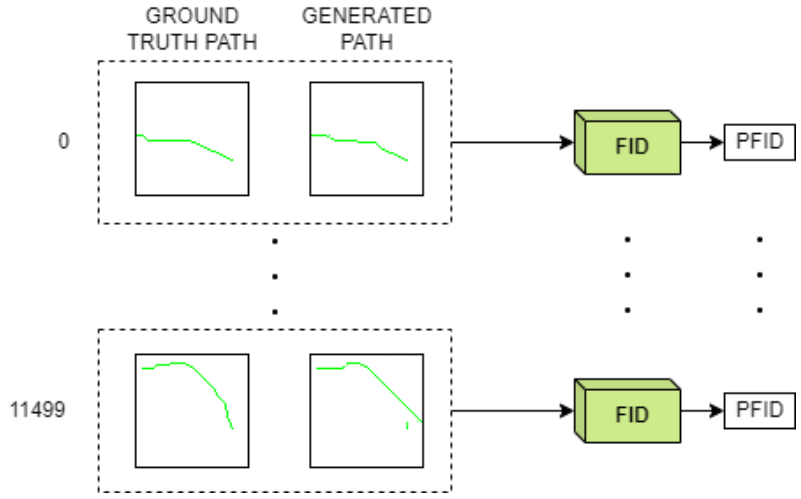


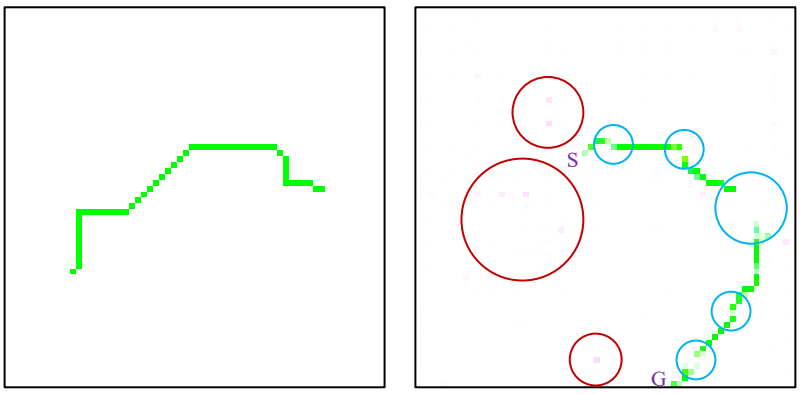
Figure 51 - Particular FID (PFID)

Experimentally, we have found range of PFID which classifies the quality of the path generated:

- PFID > 100: we obtain path which are very different with missing segments and image noise. In this situation the path starts and finishes with different start and goal nodes.
- 60 < PFID < 100: we obtain paths similar to the target one (possibility of little missing segments and image noise) with the same start and goal node.
- PFID < 60: we obtain paths which are very close to the target one

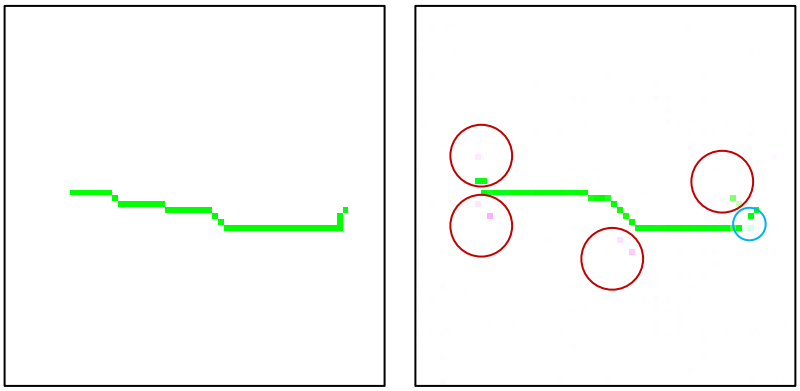
In all experiments that we have conducted the path generated does intersect with any obstacle for all PFID.

In general, there may be cases where we have blurring.



- IMAGE NOISE
- MISSING SEGMENTS
- DIFFERENT START AND GOAL NODE

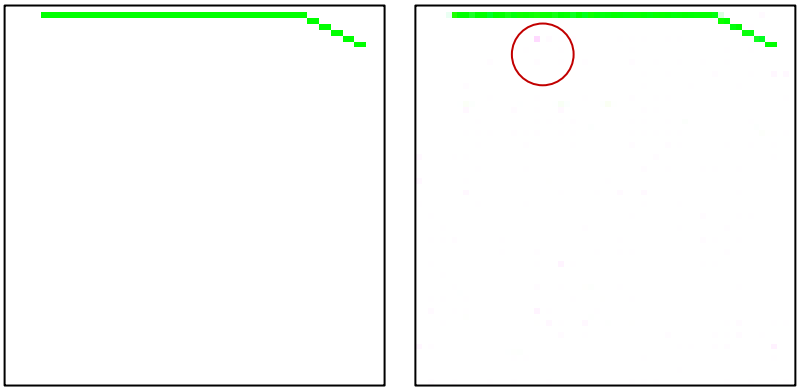
Figure 52 – PFID = 117.93



– IMAGE NOISE

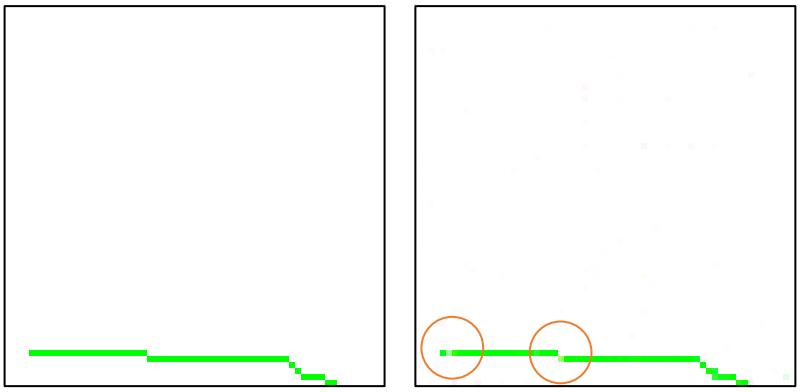
– MISSING SEGMENTS

Figure 53 – PFID = 89.12



– IMAGE NOISE

Figure 54 – PFID = 57.58



– BLURRING

Figure 55 – PFID = 24.15

In our experiments we consider as successful generated all the path which have a PFID values less than 100. So, in order to compare the experiments we introduce a validation

quantity, the success rate. This quantity tells us how many paths are correctly synthesized on the total distribution of generated paths. Considering the paths with a PFID <100 as successful generated, we can easily compute the success rate from the PFID distribution.

5.4 First Experiment – 50 epochs with smaller dataset

We have conducted the first experiment, using for the training phase, the smaller dataset (Dataset_01) with 50 epochs.

Fig. 56 shows the trend of generator and discriminator losses during each epoch. We can see that when the generator loss increases (the generator gets better in fooling the discriminator), the loss of the discriminator loss decreases as a result. This is an indication that the training is successful (see 5.11). The training is stopped before the discriminator and generator losses stabilize.

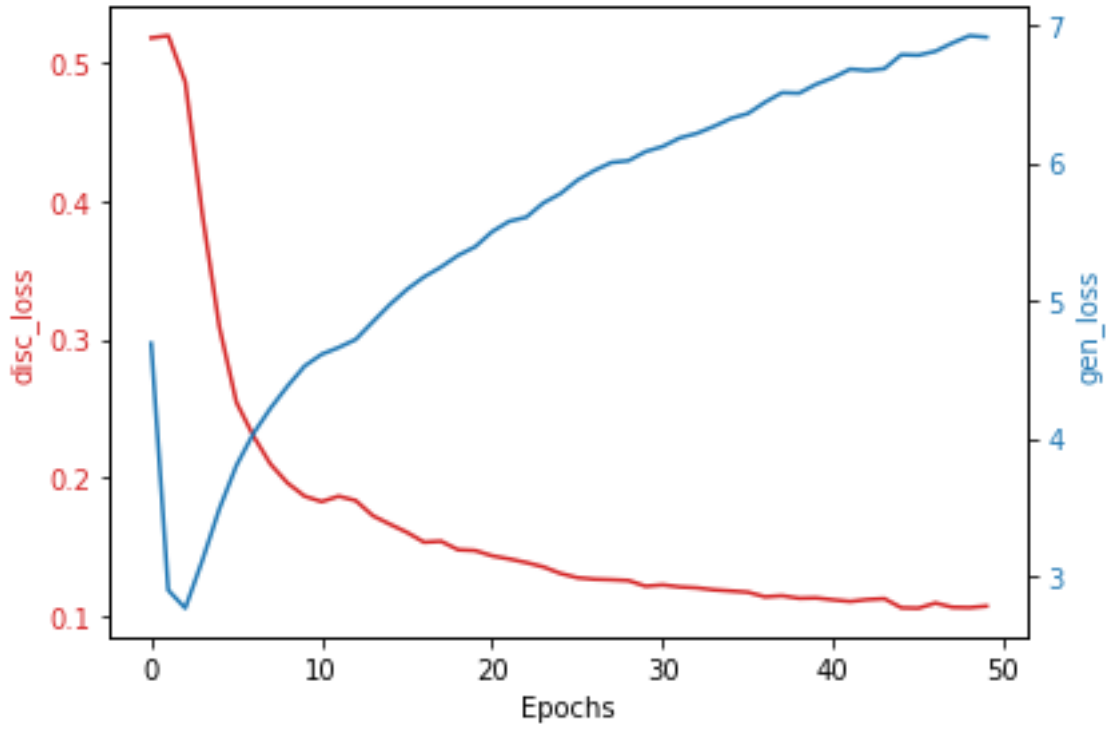


Figure 56 – Generator and Discriminator Losses for 50 epochs with smaller dataset

Fig. 57 shows the trend of the OFID during each epoch. At each epoch, the OFID decreases. This is an indication that the quality of the generated path is improving at

each epoch. After the 40^th epoch the value of OFID starts to stabilize around the value 40. The final value of OFID, for the epoch 50, is 38.31.

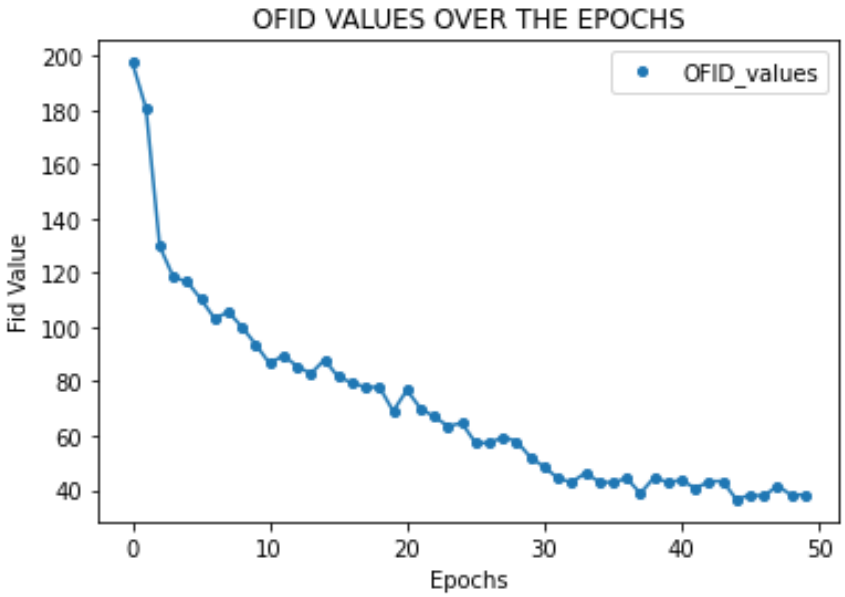


Figure 57 – OFID values for 50 epochs with smaller dataset

For the epoch 50 we have evaluated the PFID value for each pair of target and generated path. Fig. 58 shows distribution of PFID values in the validation dataset. We can see that the model generate path, with PFID value around 100, with the highest frequency. From the distribution of PFID, we can evaluate the success rate which is 38.45%.

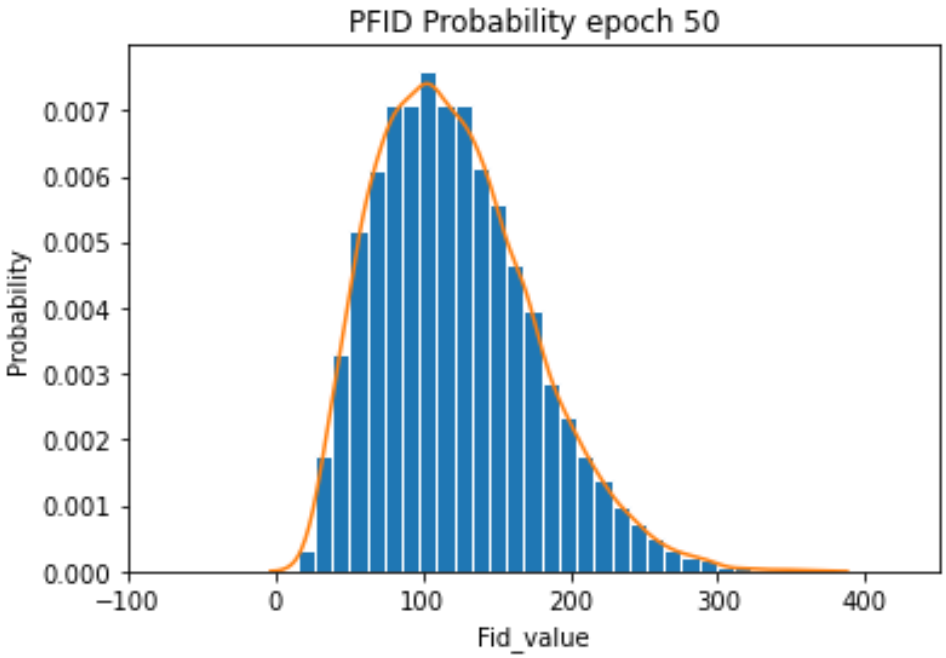


Figure 58 - PFID probability function epoch 50

5.5 Second Experiment – 150 epochs with smaller dataset

We have conducted the second experiment, using for the training phase, the smaller dataset (Dataset_01) with a higher value of epochs. We have increased the number of epochs in order to understand if we can have an improvement of the performances in the generation of the paths.

Fig. 59 shows that after the epoch 50 the discriminator loss continues to decrease, and the generator loss continues to increase accordingly. The training is stopped before the discriminator and generator losses stabilize.

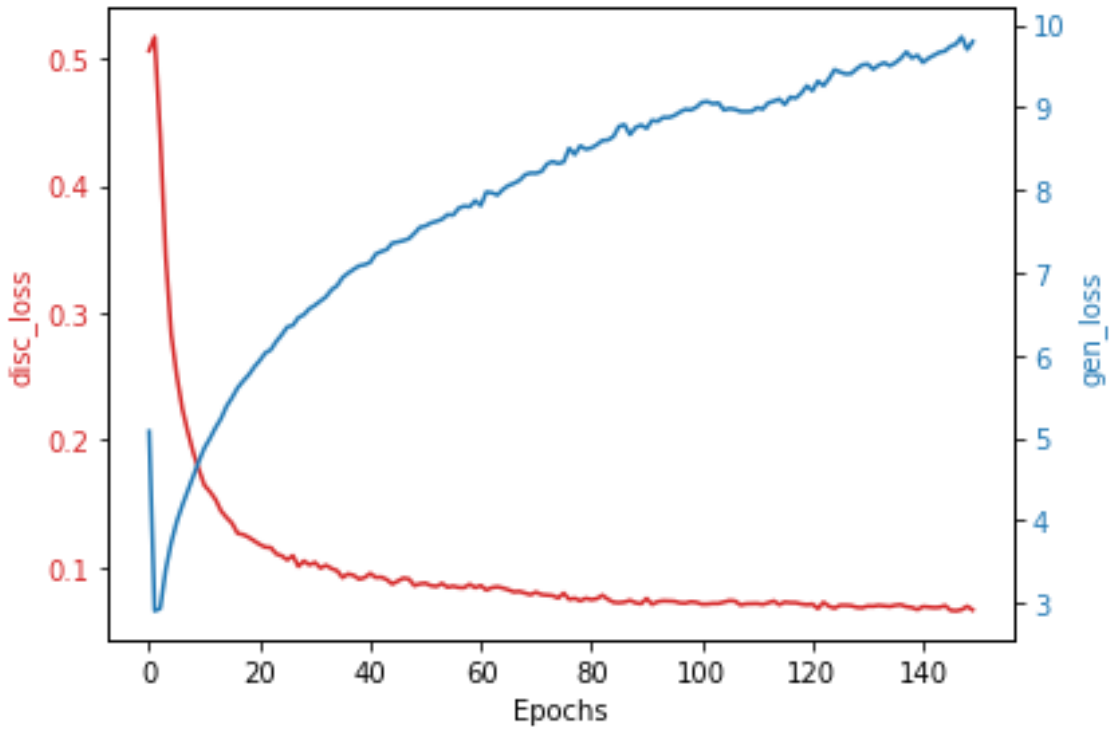


Figure 59 - Generator and Discriminator Losses for 50 epochs with smaller dataset

Fig. 60 shows that, after the epoch 50, the OFID trend stabilizes around a value 30. The final value of OFID, for the epoch 150, is 30.29. When the OFID stabilizes the quality of the generated images does not improve anymore. However, the generator and discriminator losses continue to increase and decrease respectively. So, they do not make us understand if the quality of the generated paths improves or not. The loss functions are not a good indication of the quality of the generated path.

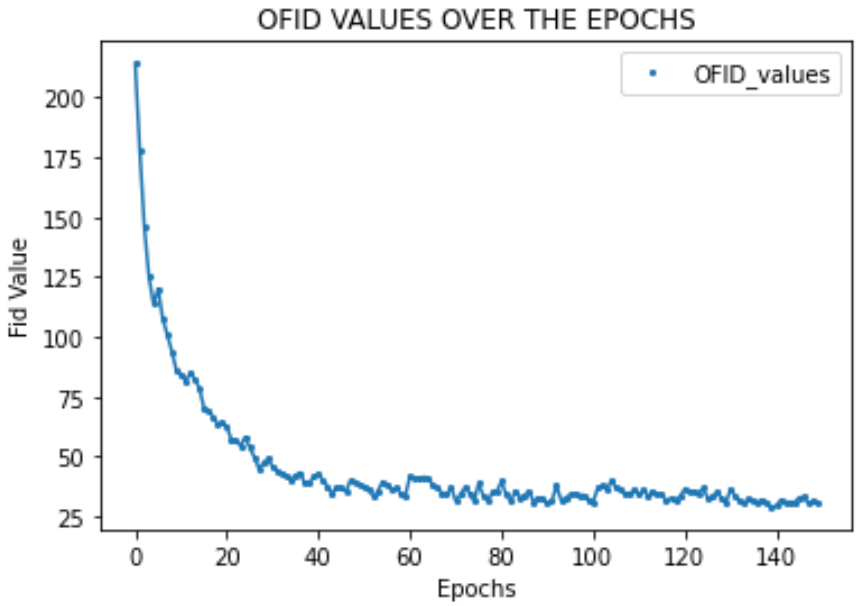


Figure 60 - OFID values for 50 epochs with smaller dataset

Fig. 61 shows the distribution of PFID values in the validation dataset evaluated for the epoch 150. In this case we have that the paths, with a PFID slightly higher than 100, are generated with the highest frequency. The distribution of PFID moves slightly to the left. So, we have that the frequency associated to the generated paths with a PFID values smaller than 100 is increased. So, we have a little improvement of the performances with respect to the previous case, Experiment 1. From the distribution of PFID, we can evaluate the success rate which is 43.68%. So, training the network with a higher number of epochs, we obtain an improvement of the performances.

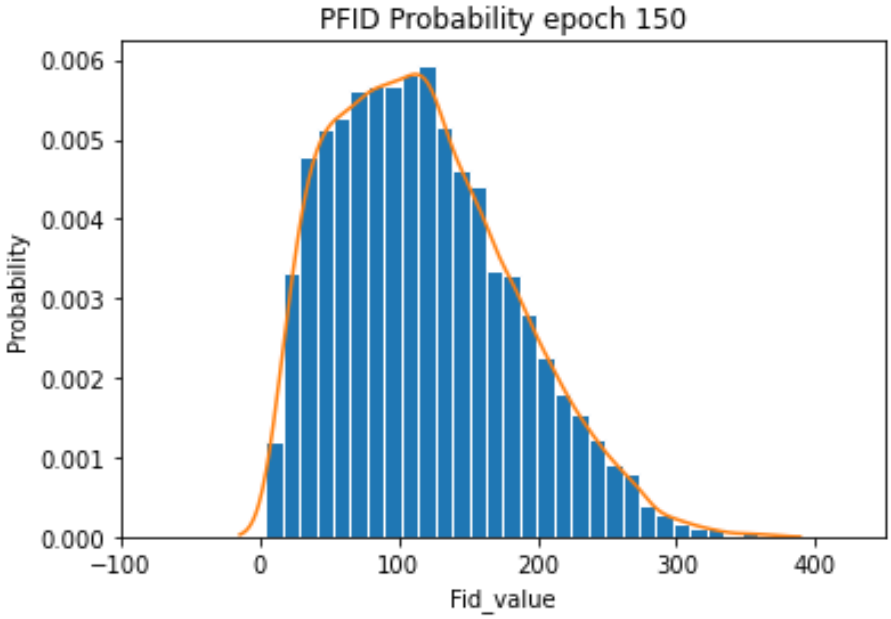


Figure 61 - PFID probability function epoch 150

5.6 Third Experiment – 50 epochs with bigger dataset

The third experiment consists in repeating the first experiment, using for the training phase, the bigger dataset (Dataset_02). We have increased the size of dataset in order to understand if we can have an improvement of the performance, with respect to the first experiment, as in the second experiment.

Fig. 62 shows how, increasing the size of dataset, the generator and discriminator losses increase and decrease, respectively, with a highest slope.

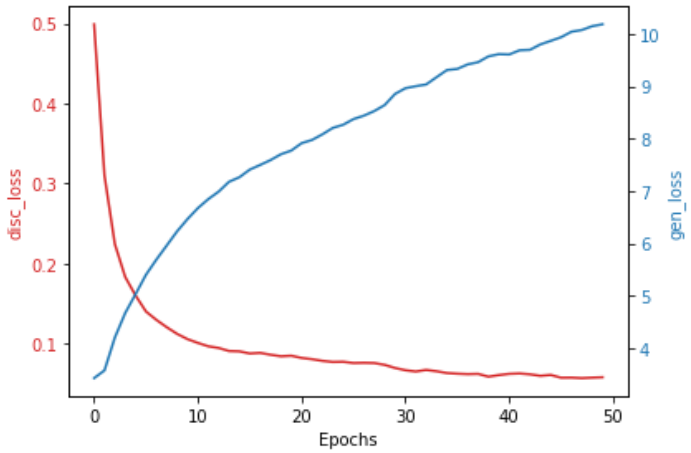


Figure 62 - Generator and Discriminator Losses for 50 epochs with smaller dataset

Fig. 63 shows that the OFID decreases with a higher velocity and at epoch 50 the OFID reaches a smaller value with respect to the first experiment. In this case, the final value of OFID, for the epoch 50, is 29.21 which is almost the same of the one evaluated in the second experiment for the epoch 150.

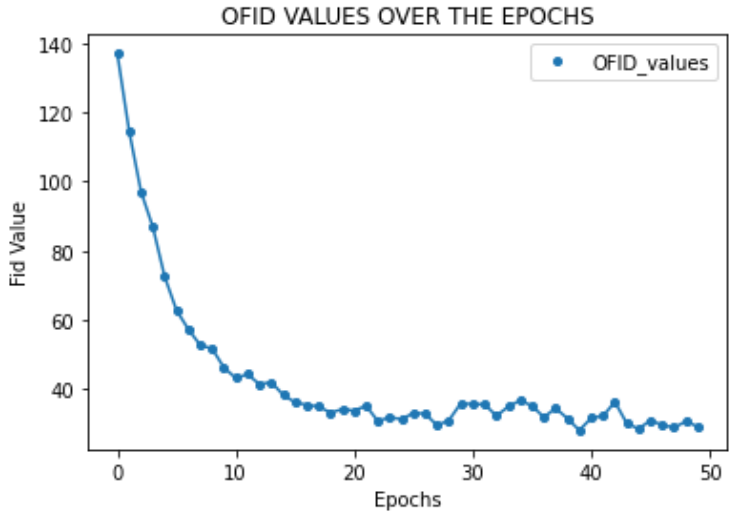


Figure 63 - OFID values for 50 epochs with smaller dataset

Fig. 64 shows that the PFID distribution moves slightly to the left with respect to the first experiment. This means that we have an improvement of the performances with respect to the first experiment. From the distribution of PFID, we can evaluate the success rate which is 47.52%.

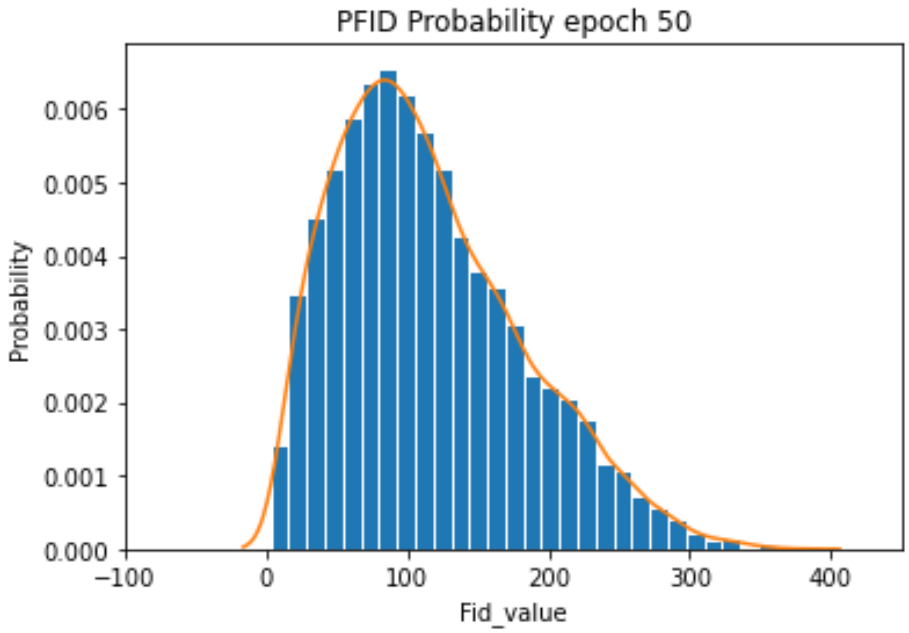


Figure 64 - PFID probability function epoch 50

In order to compare the three experiments we summarize the results as follows:

Table 2: Summary of the results for the first three experiments

Experiments	**Number of Epochs**	**Dataset Type**	**OFID Final Epoch**	**Success Rate**	**Generator Loss Final**	**Discriminator Loss Final**
1	50	Smaller	38.31	38.45%	6.914	0.107
2	150	Smaller	30.29	43.68%	9.799	0.067
3	50	Bigger	29.21	47.52%	10.187	0.058

The comparison between the first two experiments shows how increasing the number of training iterations (epochs) has improved the performances. In particular, if we train the network with a higher number of epochs, we obtain a smaller OFID. This means having a better distribution of PFID, which has increased the number of paths correctly generated passing from a success rate of 38.45% to a success rate of 43.68%.

The comparison between the first and the third experiment shows how increasing the size of the dataset has improved the performance by almost 10%. So, with the same number of training iteration, the capability of the model in generating path in unseen environments has improved. Specifically, we pass from a success rate of 38.45% to a success rate of 47.52%.

The comparison between the second and the third experiment confirm that increasing the size of dataset we have better performance. In particular, we obtain a better success rate with a number of iterations which is reduced by a factor of 3. Specifically, we pass from a success rate of 43.68% to a success rate of 47.52%.

It is worth noting that:

- Considering the training of the model with smaller dataset, if we increase the number of epochs over 150, we have no improvement of the performances.
- Considering the training of the model with bigger dataset, if we increase the number of epochs over 50, we have no improvement of the performances.

So, these results suggest that best way to train this network is to use a bigger dataset.

5.7 Fourth Experiment – 50 epochs with modified dataset

Analysing the results above mentioned we have seen that the best success rate that we have obtained is around the 50%. This means that half of the generated paths are wrong with PFID > 100. The wrong path does not intersect the obstacle but starts and finishes with different start and goal node with respect the one specified in the generator input.

We remind that the generator inputs specify start and goal node using one pixel, respectively. This could suggest that the information associated to start and goal node of the path is too little considering that the filter is moved with a stride of 2. So, for the decoder network, it could be hard using the start and goal information in order to generate the path. In other words, the information relating to start and end points might be lost during the encoding process.

So, we attempt to modify the dataset structure, associating more information to the start and goal nodes. In particular, we associate to each node a greater number of pixels, passing from 1 pixel to 8 pixel (Fig. 65).

The dataset generated has the same size of the bigger dataset in the previous experiments.

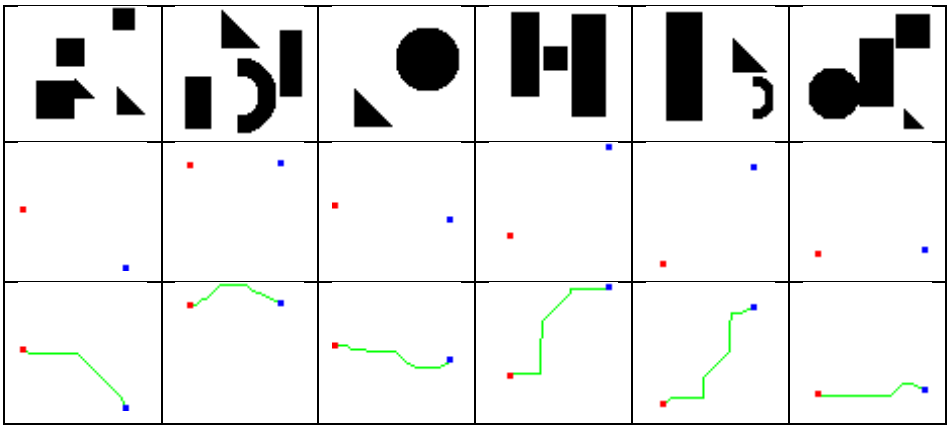


Figure 65 – An example of modified dataset. For each column, from the top to the bottom, each element represents environment maps, pair of start and goal node and feasible path generated by A algorithm with start and goal node*

We have conducted the fourth experiment, using for the training phase, the modified dataset (Dataset_03) with 50 training iterations.

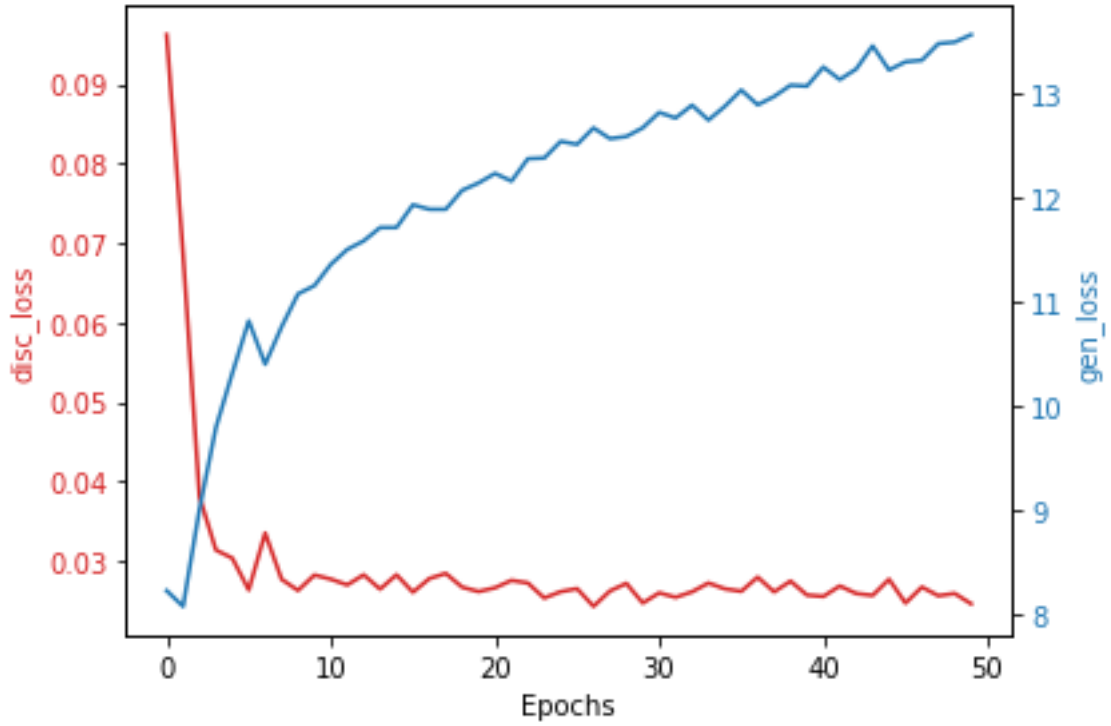


Figure 66 - Generator and Discriminator Losses for 50 epochs with modified dataset

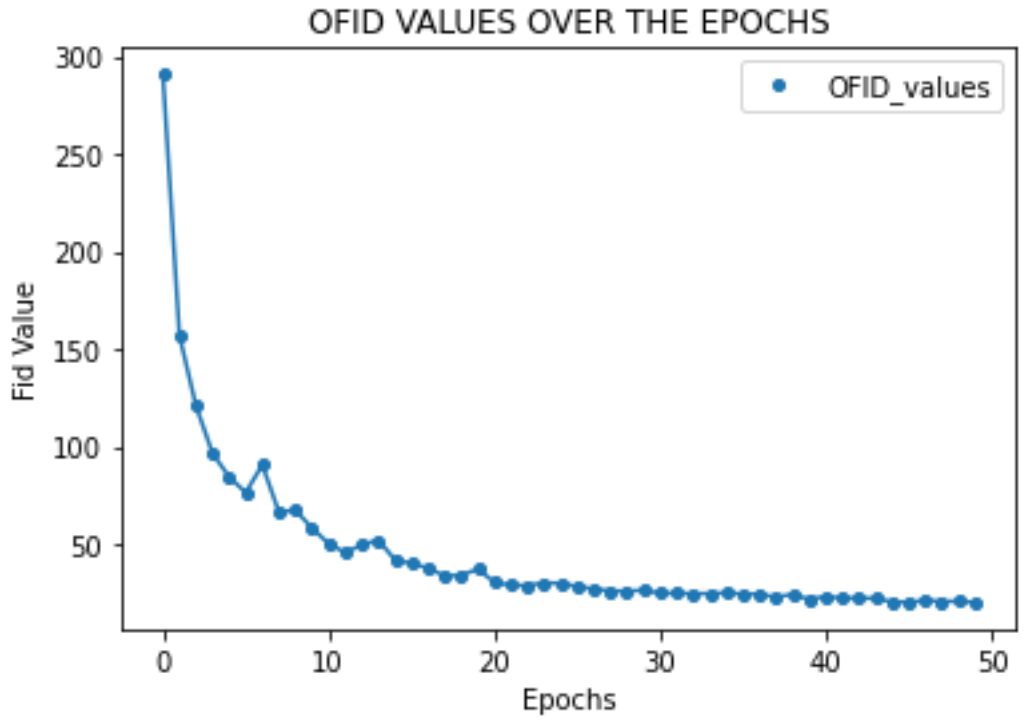


Figure 67 - OFID values for 50 epochs with modified dataset

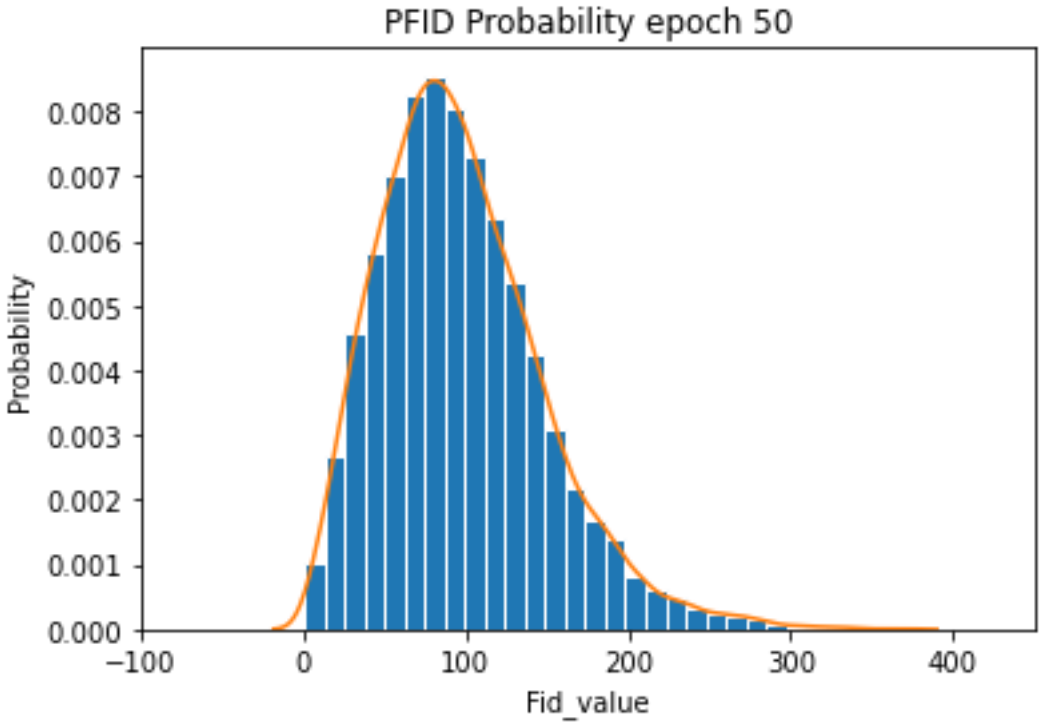


Figure 68 - PFID probability function epoch 50

Table 3: Summary of the results for the third and fourth experiment

Experiments	Number of Epochs	Dataset Type	OFID Final Epoch	Success Rate	Generator Loss Final	Discriminator Loss Final
3	50	Bigger	29.21	47.52%	10.187	0.058
4	50	Modified dataset	20.16	57.51%	13.554	0.025

The comparison between the third and the fourth experiment shows how modifying the dataset has improved the performances. In particular, the success rate has improved by 10% with respect to the third experiment. Specifically, we pass from a success rate of 47.52% to a success rate of 57.51%.

This results suggest that there was a problem related to the representation of start and goal node. So, this dataset is more suitable for this kind of architecture.

5.8 Fifth Experiment – 200 epochs increasing number of layers

In the previous experiments, using the architecture, presented in Chapter 4, we have tried to modify number of epochs, dataset size and dataset structure in order to obtain the best performance from the model. The best success rate that we have obtained is around 60%. Despite the improvements obtained, this results are not satisfactory in order to obtain a reliable path planning strategy. The model returns 40% of the time a wrong path which would lead the rover to a wrong location.

So, we have made an attempt to modify the generator architecture. In particular, we have tried to modify the number of layers within the network in order to see how the number of layers influence the network performances.

For the fifth experiment we have modified the architecture adding an encoder layer and a decoder layer. The detailed network architecture is the following:

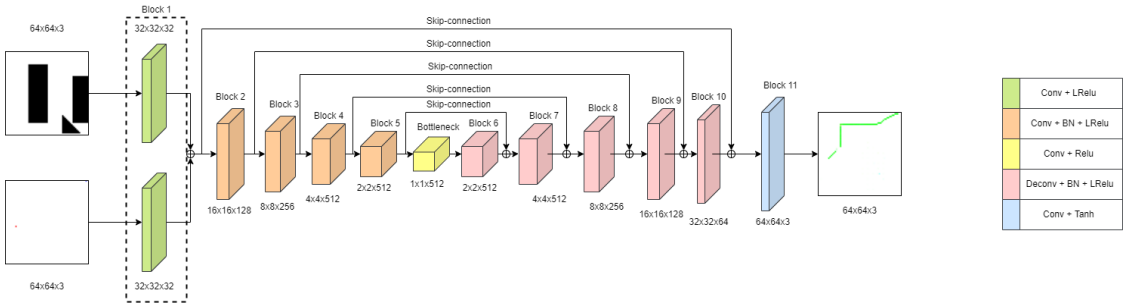


Figure 69 - Generator architecture

Adding an encoder and decoder layer means decrease the size of the output feature map of the bottleneck layer which passes from 2x2x512 to 1x1x512.

Moreover, for the training phase we do not use the modified dataset, but the original one with bigger size (Dataset_02).

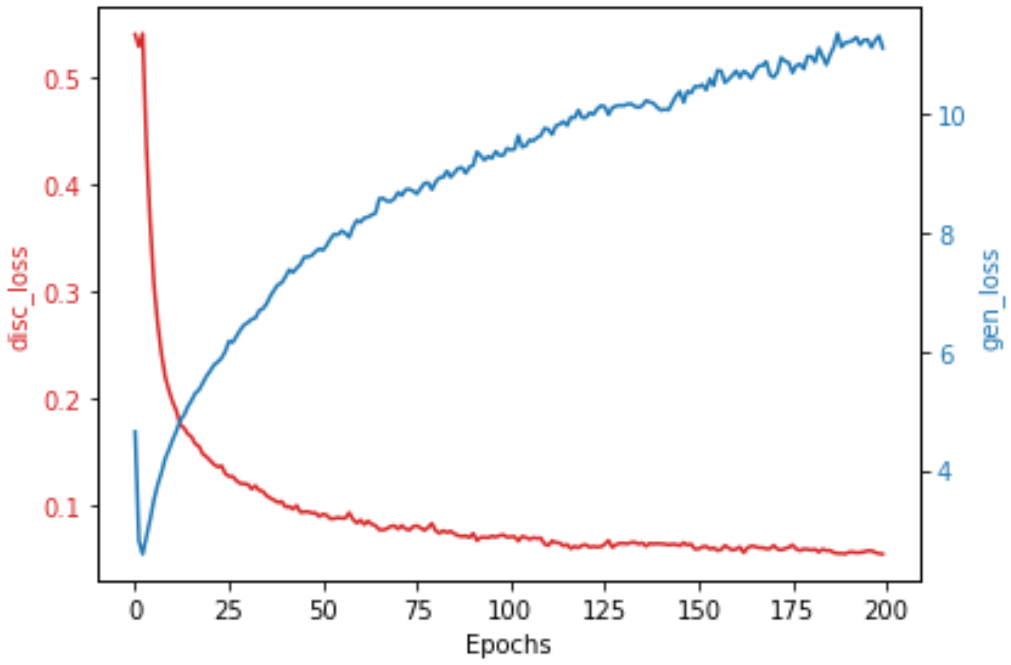


Figure 70 - Generator and Discriminator Losses for 200 epochs with bigger dataset increasing number of layers

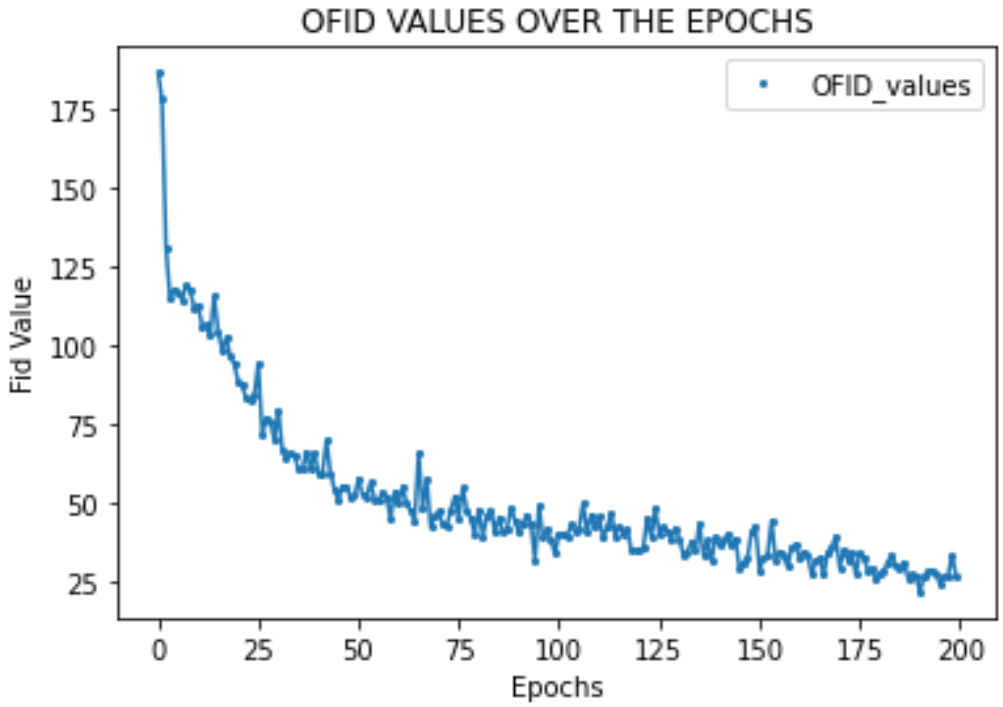


Figure 71 - OFID values for 200 epochs with bigger dataset increasing number of layers

Fig. 71 shows how the OFID decreases with a lower velocity with respect to the third Experiment. The OFID starts to stabilize after the epoch 150 around the value 30.

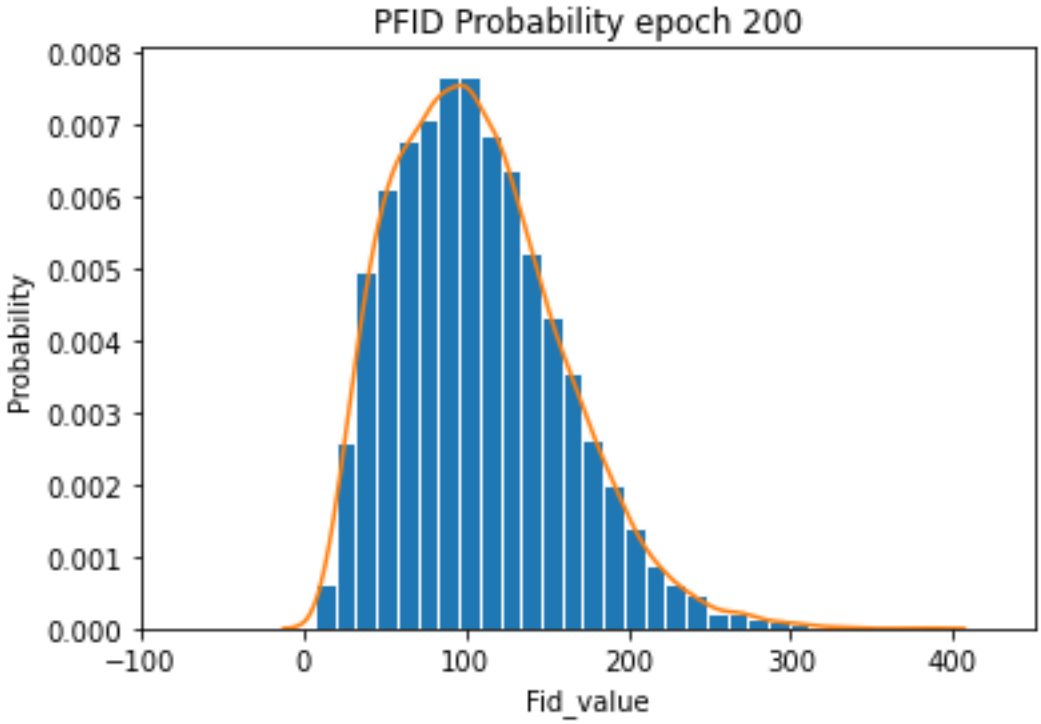


Figure 72 - PFID probability function epoch 200

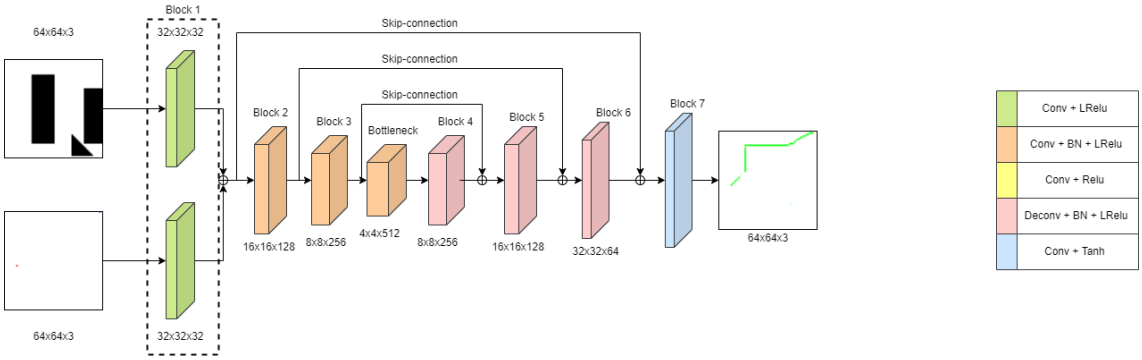
Table 4: Summary of the results for the third and fifth experiment

Experiments	Number of Epochs	Dataset Type	OFID Final Epoch	Success Rate	Generator Loss Final	Discriminator Loss Final
3	50	Bigger	29.21	47.52%	10.187	0.058
5	200	Bigger	27.07	48.62%	11.114	0.054

The comparison between the third and the fifth experiment shows a little improvement with respect to the third experiment. Comparing, the value of OFID, in the fifth experiment, at the epoch 50, we have a value of 52.42 which is much bigger with respect the one we have in the third experiment, 29.21. Increasing the number of layers has increased the necessary time to train the network of three times, getting a very small improvement in the performances. We pass from a success rate of 47.52% to a success rate of 48.62%.

5.9 Sixth Experiment – 80 epochs decreasing number of layers

For the sixth experiment we have modified the architecture removing an encoder layer and a decoder layer. The detailed network architecture is the following:



Removing an encoder and decoder layer means increasing the size of the output feature map of the bottleneck layer which passes from 2x2x512 to 4x4x512.

Moreover, for the training phase we do not use the modified dataset, but the original one with bigger size (Dataset_02).

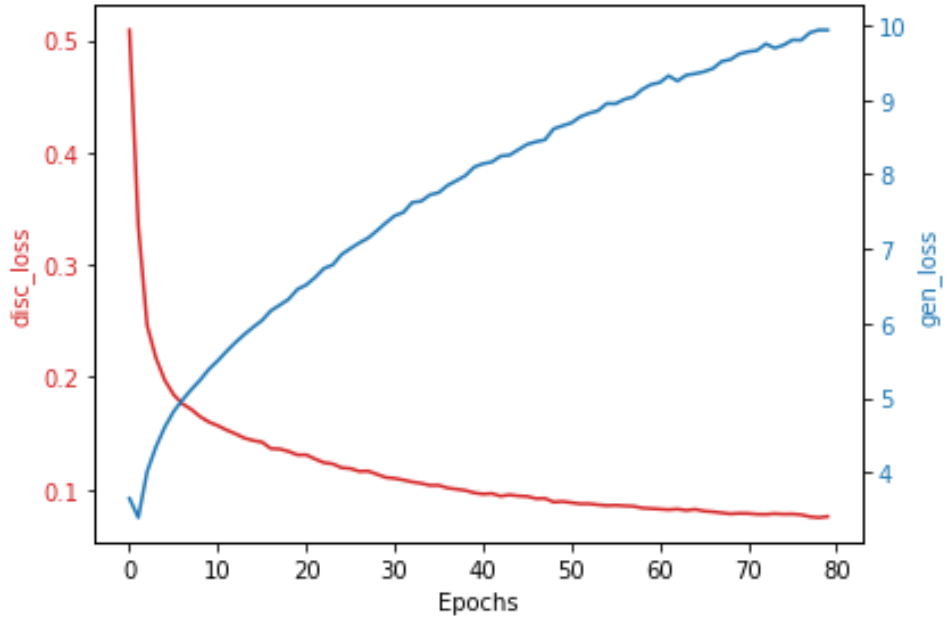


Figure 73 - Generator and Discriminator Losses for 80 epochs with bigger dataset decreasing the number of layers

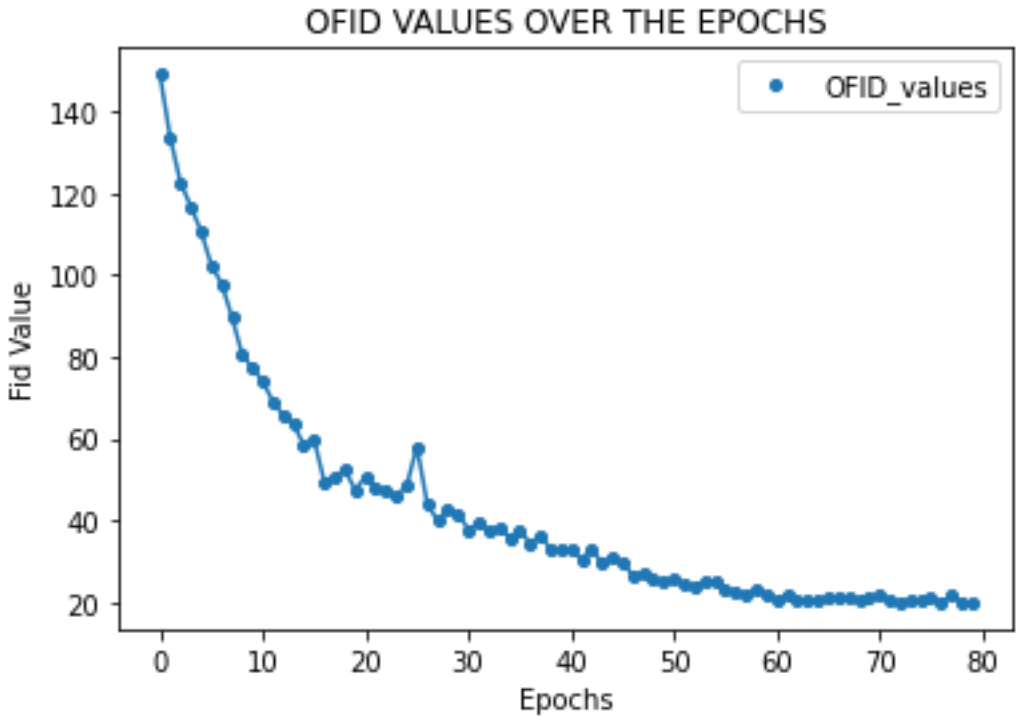


Figure 74 - OFID values for 80 epochs with bigger dataset decreasing number of layers

Fig. 74 shows that the OFID starts to stabilize after the epoch 60 around the value of 20.

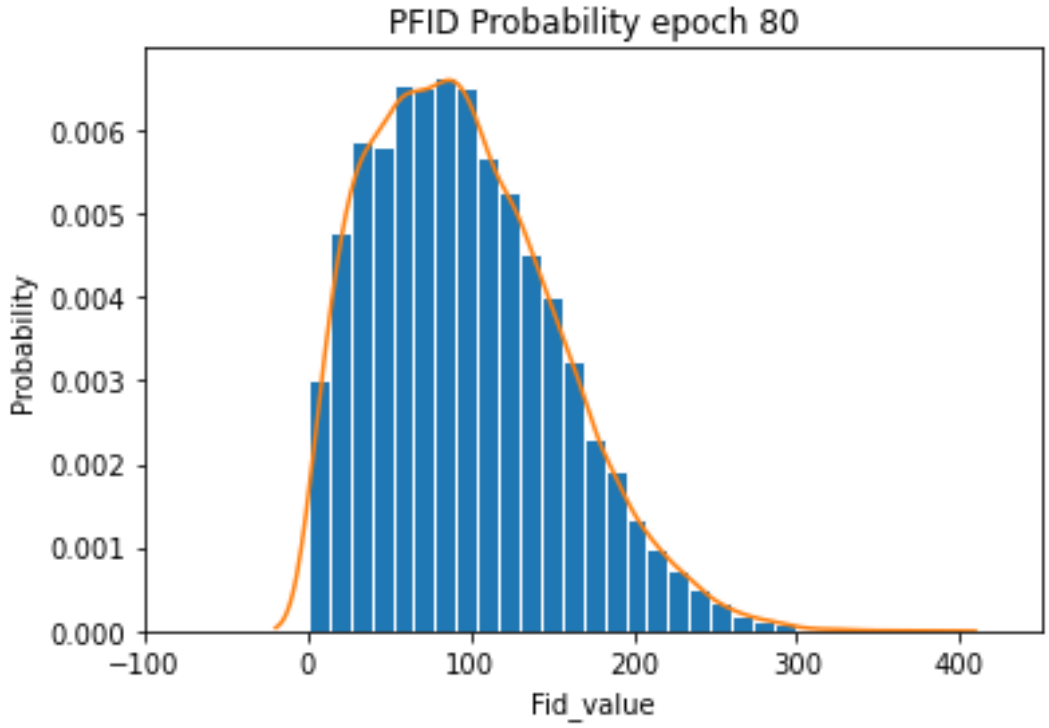


Figure 75 - PFID probability function epoch 80

Table 5: Summary of the results for the third, fifth and sixth experiment

Experiments	**Number of Epochs**	**Dataset Type**	**OFID Final Epoch**	**Success Rate**	**Generator Loss Final**	**Discriminator Loss Final**
3	50	Bigger	29.21	47.52%	10.187	0.058
5	200	Bigger	27.07	48.62%	11.114	0.054
6	80	Bigger	19.88	56.69%	9.936	0.076

The comparison between the third and the sixth experiment shows an improvement with respect to the third experiment of about 10%. We pass from a success rate of 47.52% to a success rate of 56.69%

So, with the original dataset, the best architectural choice is using 3 layers for encoder and decoder layers with a 4x4x512 bottleneck layer. Probably, the reduction of the input to a feature map lower than 4x4x512 makes the network lose relevant information, about the obstacle positions and start and goal node, which can be used from the decoder network to generate the path.

5.10 Seventh Experiment – 150 epochs decreasing number of layers using the modified dataset

The seventh experiment consists in repeating the sixth experiment, using for the training phase, the modified dataset (Dataset_03) in order to see if it is possible to have a further improvement of the performances.

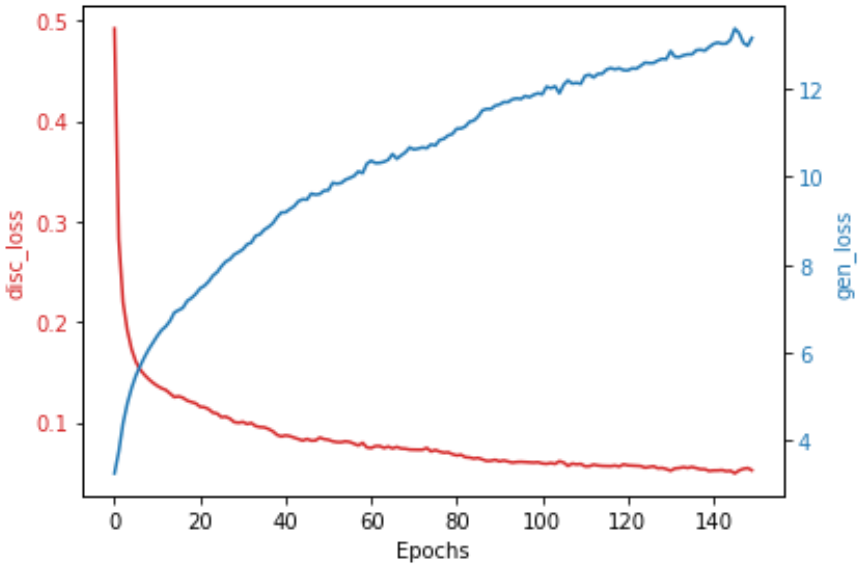


Figure 76 - Generator and Discriminator Losses for 200 epochs with modified dataset decreasing the number of layers

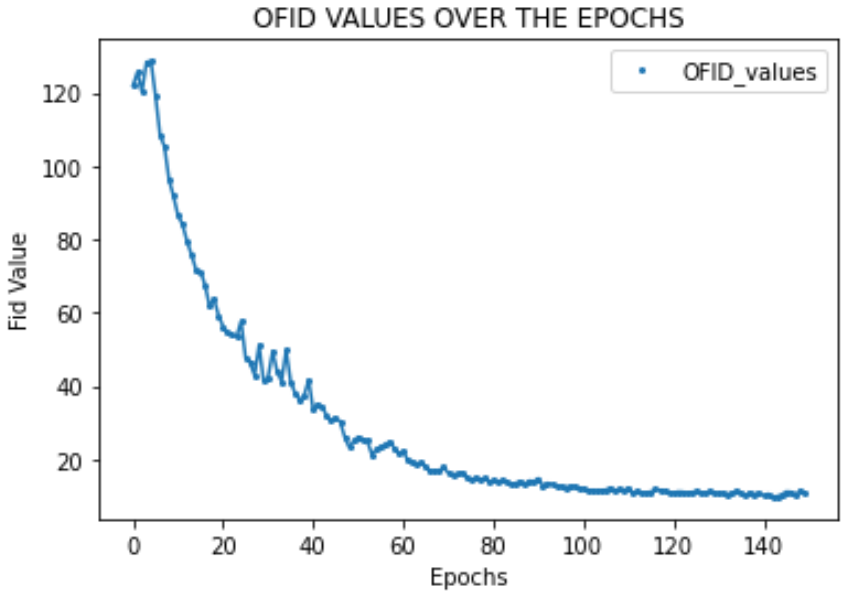


Figure 77 - OFID values for 80 epochs with bigger dataset decreasing number of layers

Fig. 77 shows that the OFID starts to stabilize after the epoch 100 around the value of 10.

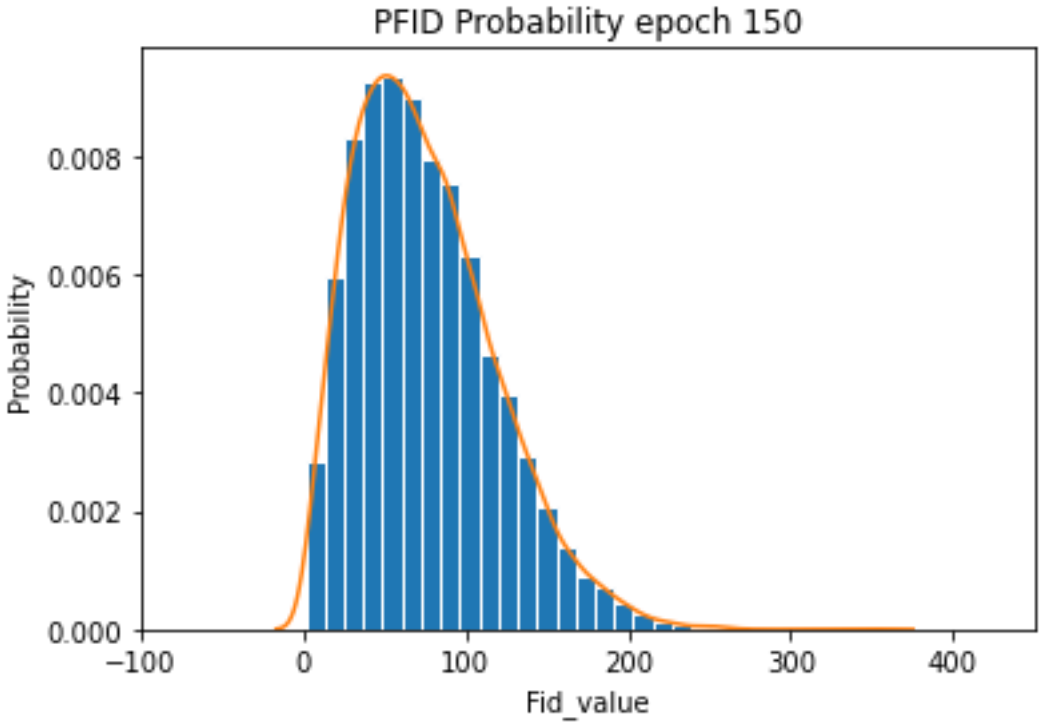


Figure 78 - PFID probability function epoch 150

Table 6: Summary of the results for the sixth and seventh experiment

Experiments	**Number of Epochs**	**Dataset Type**	**OFID Final Epoch**	**Success Rate**	**Generator Loss Final**	**Discriminator Loss Final**
6	80	Bigger	19.88	56.69%	9.936	0.076
7	150	Modified	10.70	73.88%	13.159	0.052

The comparison between the sixth and the seventh experiment shows an improvement with respect to the sixth experiment of about 17%. We pass from a success rate of 56.69% to a success rate of 73.88%

So, with the architecture used, the best choice, for the training, is using the modified dataset.

5.11 Generator loss components

It is worth remembering that the generator loss shown in each experiment is given by the weighted sum of two components, adversarial loss and L1 loss (see section 3.3.4.2.3).

We have seen, in each experiment, that the trend of generator loss and discriminator loss remains very similar. So, in order to analyse the behaviour of the generator loss, we consider the last experiment. Fig. 76 shows the trend of generator and discriminator loss, and Fig. 79 shows the component of the generator loss.

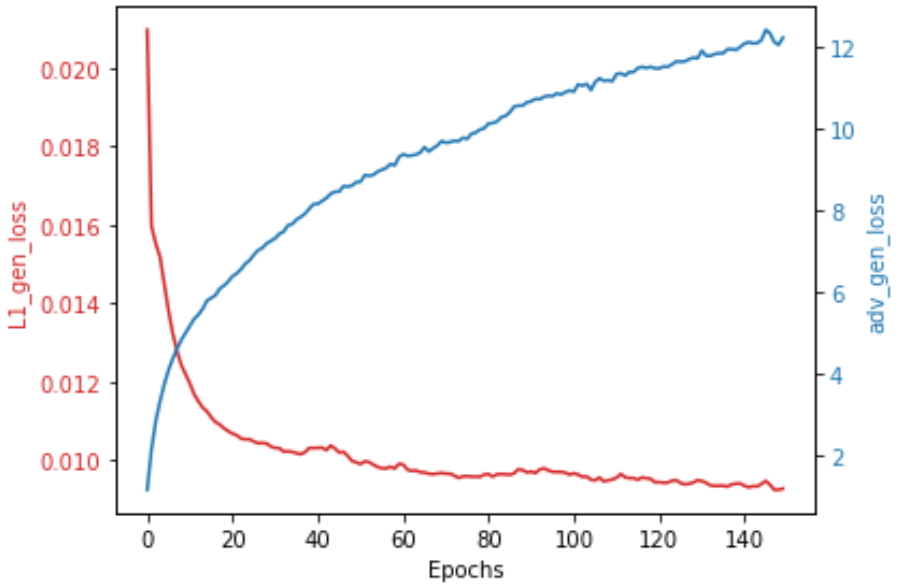


Figure 79 - L1 Generator loss and Adversarial Generator Loss for the seventh experiment

The component which defines the trend, and the module of the overall generator loss is the adversarial generator loss.

The adversarial generator loss increases at each training iteration. This means that, at each epoch, the generator capability to fool the discriminator improves, because the generator aims to maximize the probability to classify as real the generated images.

The L1 loss, which represents the distance between the generated image and the target image, is decreasing. This means that the quality of the generated images is improving at each training iteration.

However, the losses do not always make us understand how the network is working (see 5.5). They cannot be considered as absolute metrics for the evaluation of the network behaviour. For this reason, we have used as metric the FID score.

5.12 Considerations

In the first part of the experiments, the number of epochs, size and database structure have been changed while using the architecture presented in chapter 4. This has allowed to understand the best combination of number of epochs and dataset size and the impact of the dataset structure on the architecture behaviour. From the conducted tests it has been obtained a success rate around 60%.

Since it has not been possible to obtain a greater success rate, from that architecture, architectural changes have been performed in order to obtain a better success rate. In particular, the number of layers of the generator has been modified to see the influence of this factor on the model behaviour. So, decreasing the number of layers we have obtained an improvement in the success rate, achieving a value around the 74%. These improvements have been obtained with a generator architecture with 3 layers for encoder and decoder, respectively with bottleneck size of 4x4x512, using as dataset the modified one. This architecture is the best one we have obtained, considering as input, 64x64 environment map.

Despite improvements have been obtained during the experiments, the final success rate is around 74%. This means that about 26% of the generated paths are wrong.

Considering the scenario of application of the algorithm, which consists in rover planet exploration, this solution for global path planning is not the best one. In particular, the global path planning strategy aims to support the autonomous navigation. So, the rover, in autonomous mode, has to be capable to reach the specified target of scientific interest. The developed global path planning strategy returns 26% of the time a wrong path which would lead the rover to a wrong location.

However, we have seen that all the returned paths do not intersect the obstacle. This means that the algorithm manages to recognize the obstacle edges allowing to avoid the obstacles. So, when a wrong target is reached, we do not risk that the integrity of the rover was compromised, reaching unsafe zones.

It is worth reiterating that we have considered as successful all the paths which have a PFID value < 100. This means that the generated path could contain little missing segments or little image noise. In order to guarantee the continuity of the path, an algorithm that connects these cells with a straight line could be employed with minimal

computational effort (e.g. Bresenham Algorithm [55]). Moreover, in order to remove noise, the image could be convolved with a mask, which represents a low-pass-filter or a smoothing operation. An example can be the Gaussian mask which elements are determined by a Gaussian Function.

The successful generated paths are very similar to the one that A* algorithm would have generated. So, when the algorithm works well, it manages to imitate the algorithm behaviour with which has been trained, in completely unknown environments.

Using this method for the path planning allows, by construction, to have a direct generation of path planning strategy in response to context inputs (environment maps). This method is inherently reactive, and in particular, with a lack of explicit planning computations since the path planning strategy is obtained using a mathematical model which represent the link between input and output.

Moreover, this path planning strategy does not depend on the complexity of the environment. If we consider the A* algorithm, the time needed to solve the path planning problem strictly depends on the number of nodes that the algorithm expands in the search space.

It is worth noting that the architecture has been tested with 64x64 environment map.

Increasing the map size, could get worse the problem related to the representation of the start and goal node. In particular, relatively to the new map dimension, the information related to the start and goal position could be very little and might be lost during the encoding process.

Conclusions

The thesis work aimed to develop a global path planning algorithm to support the autonomous navigation system of a rover for the planet exploration.

In order to solve the path planning problem, we have proposed an algorithm based on recent developments in deep learning networks.

The problem has been treated as an image-to-image translation task. The network, given a certain input (environment map and start and goal node) returns a plausible translation of the input, which consists in the path to follow.

So, using as baseline the Pix2Pix architecture [47], which is a general approach for image-to-image translation, a GAN architecture for the path planning has been implemented. The proposed architecture uses an imitation learning approach to generate the path. Given an unknown environment, the model generates the path trying to imitate the behaviour of the A* algorithm. This means that the network is trained using as ground truth a large set of example paths generated by the A* algorithm.

In Chapter 5, the performances of the proposed architecture has been evaluated. The experiments have been conducted using as input 64x64 environment's maps. In the first part of the experiments, the number of epochs, size and database structure have been changed while using the architecture presented in chapter 4. This has allowed to understand the best combination of number of epochs and dataset size and the impact of the dataset structure on the architecture behaviour. From the conducted tests it has been obtained a success rate around 60%.

Since from the initial architecture it was not possible to obtain a satisfactory success rate, architectural changes have been performed in order to obtain a better success rate. In particular, the number of layers of the generator has been modified to see the influence

of this factor on the model behaviour. The results have shown that decreasing the number of layers improves the performance, obtaining a maximum success rate of about 74%.

However, as highlighted in the Chapter 5, these performances do not allow to have a reliable path planning strategy, applicable to the planet exploration, because the planner returns 26% of the time an incorrect path.

Despite the obtained results, these experiments highlighted the critical issues of the architecture, linked to the success rate, which translates into the limited reliability of the model in the planet exploration application.

Although several experiments have been carried out, to analyse the performance of the architecture, other key aspects need to be analysed. Specifically, the generator architecture could be changed by changing the stride parameter in the encoder and decoder layers passing from 2 to 1 in order to see its impact on the performances. The stride parameter manages the step size of the filter in convolution operation. Decreasing the stride increases the quantity of information acquired by the convolution layer (greater feature map). Naturally, in order to create a lower-dimensional representation (bottleneck layer) we need a greater number of layers. This means having more complex architecture that can cause stability problems in the training phase.

Moreover, the architecture performances should be measured increasing the size of the environment map (e.g. 128x128, 256x256, 512x512).

Acknowledgments

I would like to thank my supervisor Sabrina Corpino and my co-supervisors Fabrizio Stesina and Guglielmo Daddi who have guided and correlated me during the thesis project. Special thanks also go to my family who have always believed in me.

References

[1] C. Wong, E. Yang, X. Yan and D. Gu, "Adaptive and intelligent navigation of autonomous planetary rovers — A survey," 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pp. 237-244, 2017.

[2] T. Sasaki et al. "Where to Map? Iterative Rover-Copter Path Planning for Mars Exploration," in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 2123-2130, April 2020.

[3] J. Zhang, Y. Xia, G. Shen. “A novel learning-based global path planning algorithm for planetary rovers”. Neurocomputing, 361: 69-76, 2019.

[4] P. Gao, Z. Liu, Z. Wu and D. Wang, "A Global Path Planning Algorithm for Robots Using Reinforcement Learning," 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), pp. 1693-1698, 2019.

[5] J. Carsten et al. "Global path planning on board the mars exploration rovers." 2007 IEEE Aerospace Conference. IEEE, 2007.

[6] A. Ellery. “Planetary rovers: robotic exploration of the solar system”. Springer, 2015.

[7] K. Otsu et al. "Fast approximate clearance evaluation for rovers with articulated suspension systems." Journal of Field Robotics 37.5: 768-785. 2020.

[8] T. Lozano-Pérez , and M. A. Wesley. "An algorithm for planning collision-free paths among polyhedral obstacles." Communications of the ACM 22.10. 560-570. 1979

[9] Lozano-Perez, "Spatial Planning: A Configuration Space Approach," in IEEE Transactions on Computers, vol. C-32, no. 2, pp. 108-120, Feb. 1983.

[10] P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968.

[11] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," in IEEE Transactions on Robotics, vol. 21, no. 3, pp. 354-363, June 2005.

[12] S. Koenig, and L. Maxim. "D* lite." Aaai/iaai 15, 2002.

[13] A. Stentz. "The focussed D* algorithm for real-time replanning." IJCAI. Vol. 95, 1995.

[14] B. Yan, et al. "A Comprehensive Survey and Analysis on Path Planning Algorithms and Heuristic Functions." Science and Information Conference. Springer, Cham, 2020.

[15] D. Ferguson, A. Stentz. "Field D*: An interpolation-based path planner and replanner." Robotics research. Springer, Berlin, Heidelberg. 239-253. 2007

[16] J. Carsten, D. Ferguson and A. Stentz, "3D Field D: Improved Path Planning and Replanning in Three Dimensions," 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006.

[17] D. Kenny, et al. "Theta*: Any-angle path planning on grids." Journal of Artificial Intelligence Research 39: 533-579, 2010.

[18] A. Nash, S. Koenig, M. Likhachev. "Incremental Phi*: Incremental any-angle path planning on grids." Twenty-First International Joint Conference on Artificial Intelligence. 2009.

[19] B. Logan , N. Alechina. "A* with bounded costs." AAAI/IAAI. 1998.

[20] A. Stentz. "CD*: A Real-Time Resolution Optimal Re-Planner for Globally Constrained Problems." AAAI/IAAI. 2002.

[21] I. Goodfellow et al. "Generative adversarial nets." Advances in neural information processing systems 27 (2014).

[21] I. Goodfellow et al. "Generative adversarial nets." Advances in neural information processing systems 27, 2014.

[22] S. Osher, J. A. Sethian. "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations." Journal of computational physics 79.1: 12-49. 1988

[23] A. Valero-Gomez, J. V. Gomez, S. Garrido and L. Moreno, "The Path to Efficiency: Fast Marching Method for Safer, More Efficient Mobile Robot Trajectories," in IEEE Robotics & Automation Magazine, vol. 20, no. 4, pp. 111-120, Dec. 2013.

[24] S. Garrido, D. Álvarez, and L. Moreno. "Path planning for mars rovers using the fast marching method." Robot 2015: Second Iberian Robotics Conference. Springer, Cham, 2016.

[25] R. Philippsen and R. Siegwart, "An Interpolated Dynamic Navigation Function," Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pp. 3782-3789, 2005.

[26] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans and D. Lane, "Path Planning for Autonomous Underwater Vehicles," in IEEE Transactions on Robotics, vol. 23, no. 2, pp. 331-341, April 2007.

[27] D. A. Sinyukov, and T. Padir. "CWave: Theory and Practice of a Fast Single-source Any-angle Path Planning Algorithm." Robotica 38.2: 207-234. 2020

[28] M. W. Otte. "A survey of machine learning approaches to robotic path-planning." University of Colorado at Boulder, Boulder (2015).

[29] N. Ma, J. Wang, J. Liu and M. Q. -H. Meng, "Conditional Generative Adversarial Networks for Optimal Path Planning," in IEEE Transactions on Cognitive and Developmental Systems . 2020

[30] A. Tamar et al. "Value iteration networks." arXiv preprint arXiv:1602.02867 (2016).

[31] N. Soboleva, K. Yakovlev. "GAN Path Finder: Preliminary Results." Joint German/Austrian Conference on Artificial Intelligence. Springer, Cham, 2019.

[32] Penggang Gao, Zihan Liu, Zongkai Wu, Donglin Wang. “A Global Path Planning Algorithm for Robots Using Reinforcement Learning”. Proceeding of the IEEE International Conference on Robotics and Biomimetics. 2019

[33] A. Jabbar, X. Li, B. Omar. "A survey on generative adversarial networks: Variants, applications, and training." ACM Computing Surveys (CSUR) 54.8: 1-49. 2021.

[34] O. I. Abiodun et al. "State-of-the-art in artificial neural network applications: A survey." Heliyon 4.11: e00938. 2018

[35] F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6: 386. 1958

[36] C. C. Aggarwal. “Neural Networks and Deep Learning”. Springer. 2018

[37] D. Foster. “Generative Deep Learning: Teaching Machines to Paint, Write, Compose and play”. O'Reilly Media. 2019.

[38] C. Shorten, T. M. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning”. Journal of Big Data 6.1: 1-48. 2019

[39] K. Simonyan, A. Zisserman. “Very deep convolutional networks for large-scale image recognition". arXiv preprint arXiv:1409.1556. 2014.

[40] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770-778, 2016.

[41] Z. Wang, Q. She, T. E. Ward. “Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy”. ACM Computing Surveys (CSUR) 54.2: 1-38. 2021

[42] A. Radford, L. Metz, S. Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. arXiv preprint arXiv:1511.06434. 2015

[43] I. Goodfellow, Y. Bengio, A. Courville. “Deep Learning (Adaptive Computation and Machine Learning series)”. Cambridge Massachusetts: 321-359. 2017

[44] Se. Ioffe, C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. International conference on machine learning. PMLR, 2015.

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. The journal of machine learning research 15.1: 1929-1958. 2014.

[46] A. Krizhevsky, I. Sutskever, and G. Hinton. “Imagenet classification with deep convolutional neural networks”. Advances in neural information processing systems 25: 1097-1105. 2012.

[47] P. Isola, J. Zhu, T. Zhou and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[48] M. Y. Liu, T. Breuel, Jan Kautz. “Unsupervised Image-to-Image Translation Networks”. Advances in neural information processing systems. 2017.

[49] J. Zhu, T. Park, P. Isola and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2242-2251, 2017.

[50] D. P. Kingma, J. Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980. 2014.

[51] A. Borji. "Pros and cons of gan evaluation measures." Computer Vision and Image Understanding 179: 41-65. 2019

[52] A. Gretton et al. "A kernel two-sample test." The Journal of Machine Learning Research 13.1: 723-773. 2012

[53] E. W. Dijkstra. “A Note on Two Problems in Connexion with Graphs”. Numerische mathematik 1.1: 269-271. 1959.

[54] A. Patel. “A*’s Use of the Heuristic”. Ed. by Red Blob Games. Amit’s A* Pages. Stanford University. 2018.
http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html.

[55] J. E. Bresenham, “Algorithm for computer control of a digital plotter”. IBM Systems journal 4.1: 25-30. 1965