**ABSTRACT:**

There is a continuous demand for intelligent software models that enhances the subsystems of autonomous driving. The recognition of specific elements in the environment during driving is specifically important for the awareness of the automobile or the automobile network, but it is also quite challenging. Tunnels and bridges are amongst the most prominent and commonly found pieces of infrastructure on the road so, it is obvious that we should strongly consider their presence or absence in the surroundings during the decision-making process. There are multiple motivations for obtaining this information: headlights correction, ventilation and temperature setting, sensors calibration, windscreen wipers usage reduction, speed adjustment, alerts regarding maximum allowed height, and even strategizing. In the following paper we will focus on different methods and techniques based on convolutional neural networks that can be used to resolve this problem of classification, and we will conclude which one is best for which situation. We will also present the process of developing a complete environment that allows the user to train or validate models, analyze statistics, and visualize the results.

Table of Contents

[1. Introduction 8](#_Toc43067478)

[1.1. Context 8](#_Toc43067479)

[1.2. Motivation 10](#_Toc43067480)

[1.3. Problem definition 11](#_Toc43067481)

[1.4. Solution construction 11](#_Toc43067482)

[1.5. Article Structure 11](#_Toc43067483)

[1.6. Personal contribution 12](#_Toc43067484)

[2. Theoretical Background 12](#_Toc43067485)

[2.1. Image processing 13](#_Toc43067486)

[History 13](#_Toc43067487)

[Introduction 14](#_Toc43067488)

[Image analysis 14](#_Toc43067489)

[Image enhancement 15](#_Toc43067490)

[Image compression 15](#_Toc43067491)

[Image restoration 16](#_Toc43067492)

[Image processing and artificial intelligence 16](#_Toc43067493)

[2.2. Machine Learning and Artificial intelligence 17](#_Toc43067494)

[Artificial intelligence 17](#_Toc43067495)

[Machine learning 18](#_Toc43067496)

[Supervised learning 19](#_Toc43067497)

[Unsupervised learning 20](#_Toc43067498)

[Semi-supervised learning 21](#_Toc43067499)

[Reinforcement learning 21](#_Toc43067500)

[2.3. Artificial Neural Networks 22](#_Toc43067501)

[History 22](#_Toc43067502)

[The Neuron and Backpropagation 23](#_Toc43067503)

[Activation function 24](#_Toc43067504)

[Loss function 25](#_Toc43067505)

[Layers 26](#_Toc43067506)

[Types of ANN 26](#_Toc43067507)

[2.4. Convolutional Neural Networks 27](#_Toc43067508)

[Description 27](#_Toc43067509)

[Convolution layer 28](#_Toc43067510)

[Pooling layer 29](#_Toc43067511)

[Fully Connected layer and Loss layer 29](#_Toc43067512)

[Overfitting and Underfitting 29](#_Toc43067513)

[2.5. Active Learning 30](#_Toc43067514)

[3. Literature Overview 31](#_Toc43067515)

[3.1. State of the art/Related work 31](#_Toc43067516)

[A. Fast Vision-Based Road Tunnel Detection [27] 31](#_Toc43067517)

[B. Road Tunnel Entrance Recognition System [28] 33](#_Toc43067518)

[C. Tunnel detecting device for vehicle and ... [30] 35](#_Toc43067519)

[3.2. Implementation and results 35](#_Toc43067520)

[A. Fast Vision-Based Road Tunnel Detection [27] 36](#_Toc43067521)

[B. Road Tunnel Entrance Recognition System [28] 36](#_Toc43067522)

[C. Tunnel detecting device for vehicle and ... [30] 37](#_Toc43067523)

[3.3. Differences and comparison 37](#_Toc43067524)

[A. Fast Vision-Based Road Tunnel Detection [27] 37](#_Toc43067525)

[B. Road Tunnel Entrance Recognition System [28] 38](#_Toc43067526)

[C. Tunnel detecting device for vehicle and ... [30] 38](#_Toc43067527)

[3.4. Conclusions 38](#_Toc43067528)

[4. Application and results 39](#_Toc43067529)

[4.1. General presentation 39](#_Toc43067530)

[4.2. Methodology 40](#_Toc43067531)

[4.3. Proposed approaches 41](#_Toc43067532)

[4.4. Data 42](#_Toc43067533)

[Complete flow 42](#_Toc43067534)

[Collecting the Data 42](#_Toc43067535)

[Extracting the information 43](#_Toc43067536)

[Image Preprocessing 44](#_Toc43067537)

[4.5. Application 45](#_Toc43067538)

[Environment 45](#_Toc43067539)

[Frameworks and libraries 45](#_Toc43067540)

[User Interface 45](#_Toc43067541)

[Architecture 48](#_Toc43067542)

[4.6. Models comparison 50](#_Toc43067543)

[Models architecture 51](#_Toc43067544)

[Training of the models 51](#_Toc43067545)

[The best model 51](#_Toc43067546)

[4.7. CNN Scope Strategies 52](#_Toc43067547)

[Supervised tunnel binary classification 53](#_Toc43067548)

[Supervised tunnel multiclass classification 54](#_Toc43067549)

[Untested strategies 54](#_Toc43067550)

[Methods results 55](#_Toc43067551)

[Active learning strategy 56](#_Toc43067552)

[Comparison of chosen models 57](#_Toc43067553)

[5. Conclusion and Extensibility 59](#_Toc43067554)

[5.1. Experiments & usability 59](#_Toc43067555)

[5.2. Limitations 59](#_Toc43067556)

[5.3. Future work 60](#_Toc43067557)

[5.4. Comparison to State of the art 61](#_Toc43067558)

[Acknowledgements 61](#_Toc43067559)

[Bibliography 62](#_Toc43067560)

Table of Figures

[Figure 2‑1 [55] – first digital image 13](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067434)

[Figure 2‑2 [60] – grayscale representation 14](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067435)

[Figure 2‑3 [56] - face recognition 15](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067436)

[Figure 2‑4 [57] – image enhancement 15](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067437)

[Figure 2‑5 [59] – compression example 16](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067438)

[Figure 2‑6 [58] - image restoration 16](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067439)

[Figure 2‑7 [53] – dataset not linearly separable 19](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067440)

[Figure 2‑8 [53] - clustering method 20](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067441)

[Figure 2‑9 [61] – the neuron inputs and outputs 24](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067442)

[Figure 2‑10 [61] - activation functions 25](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067443)

[Figure 2‑11 [66] – simplified architecture 26](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067444)

[Figure 2‑12 [62] – CNN layers visualization 28](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067445)

[Figure 2‑13 [63] – main layers mechanics 28](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067446)

[Figure 2‑14 [64] – pooling 29](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067447)

[Figure 2‑15 [65] – two learned equation on the same dataset 29](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067448)

[Figure 2‑16 – active learning steps for augmenting a database 31](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067449)

[Figure 3‑1 [27] – the tunnel separated in states 32](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067450)

[Figure 3‑2 [27] – vertical and horizontal darkness histogram 33](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067451)

[Figure 3‑3 [27] – vertical and horizontal brightness histogram 33](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067452)

[Figure 3‑4 [28] – “Consecutive tunnel lights” method 34](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067453)

[Figure 3‑5 [28] – “Tunnel arch detection” method 34](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067454)

[Figure 3‑6 [28] – “Elliptic black hole” method 34](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067455)

[Figure 3‑7 [28] – “Template matching” method 35](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067456)

[Figure 3‑8 [30] - patent for tunnel detection 35](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067457)

[Figure 3‑9 [27] – results by sequence 36](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067458)

[Figure 3‑10 [28] – results by method 36](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067459)

[Figure 3‑11 [28] – results for methods and hybrids 37](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067460)

[Figure 4‑1 – simplified architecture of the project 39](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067461)

[Figure 4‑2 – possible classification approaches 41](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067462)

[Figure 4‑3 – The steps of our data 42](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067463)

[Figure 4‑4 – labelling interface 43](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067464)

[Figure 4‑5 – visualization of the cropped area 44](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067465)

[Figure 4‑6– the user interface 46](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067466)

[Figure 4‑7 -active example changes 47](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067467)

[Figure 4‑8 – active learning querying 47](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067468)

[Figure 4‑9 – the image-probability animation visualization 48](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067469)

[Figure 4‑10 - stats for the training phase 52](#_Toc43067470)

[Figure 4‑11 – stats for the performance on the first dataset 52](#_Toc43067471)

[Figure 4‑12 – detailed statistics on the new test dataset 54](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067472)

[Figure 4‑13 - stats for the performance on the updated test dataset 55](#_Toc43067473)

[Figure 4‑14 - stats for the training phase 55](#_Toc43067474)

[Figure 4‑15 – active learning performance visualization 57](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067475)

[Figure 4‑1 – Raspberry Pi experiment project schematics 59](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067476)

[Figure 5‑2 - real-time image of a prediction 59](file:///C:\Users\extra\OneDrive\Desktop\modelLicentaFINALRaduGalan.docx#_Toc43067477)

# Introduction

We will begin by presenting a little motivation for choosing this field of research, we will continue with the current background of the project, in general terms, and then proceed by explaining what the stated problem is and approximately how we propose to resolve it.

## Context

We most definitely live in times of great progress in all fields of research, but one in particular, has always intrigued me through its potential strength of changing the world for good. These days we are witnessing an overwhelming number of limitations given by human capabilities, ranging from individual power of acquiring expertise in certain subfields - all the way to the quality, frequency, or even individual willingness of communicating between one another. Allow me to better explain myself by providing a couple of useful examples.

Scientist have evolved quite interesting over the centuries, because from mere thinkers and accidental inventors they have become great academic researchers and engineers there was certainly a long way to go. Almost since we first started studying the world around us we had the tendency to classify everything we have done about it. In ancient Greece for example the first physicians were not just mathematicians, astronomers or healers, but were above anything else thinkers. They were usually practicing a little of everything, from physics and philosophy to strategist or political advisor depending on whether there were times of war or not. Of course, with time science branched into Social, Natural, Formal and Applied sciences and thinkers slowly began choosing to specialize in a specific field, given the fact they could no longer cope with the amount of information available. Coming closer to the present, it was acknowledged by E.T. Bell that “The Last Universalist” [1](not in general sciences, but specifically in the subfield of mathematics) was Jules Henry Poincare (b. 1789 – d. 1857). Almost 2 centuries later the situation changed a lot, not even after a lifetime of study one cannot declare himself an “Artificial Intelligence Universalist” (which is subfield of Theoretical Computer Science, subfield of General Computer Science, subfield of Logic). It is becoming increasingly hard for human individuals to reach those frontiers of knowledge and the size of the subfield covered by each is extremely tiny related to the whole, which gives significant less perspective to the individual. This is one of the human limitations that Artificial Intelligence could one day begin to resolve.

Our communication methods as humans, although evolving to unimaginable scale, don’t fundamentally change anything but the quantity of messages and the speed of information transfer. The quality and efficiency though, remain only slightly altered due to the emergence of new words. This is not our fault, but merely a natural restraint attributable to the ever-increasing population and social clustering by location and culture. Although we are internet-connected, we are still not inter-connected and we will probably never be, at least as long as our strong ego will be the one responsible for the evolution of societies. We often work as individual components although we essentially form a complex biological structure, so called superorganism [2]. This represents yet another human limitation with potential to be resolved by the emerging science of Artificial Intelligence.

Neither communication, nor expertise is a problem for A.I. given enough resources. Computers have already evolved computational as strong as the brain of a mice, although the law of More [3] is slightly off with the predictions, the inevitable of surpassing human’s brain powers (and even the brains of all humans) is only delayed. Moreover, all the communication methods and protocols directly in the hardware or through the internet lack all the big disadvantages of human interactions.

We have a long way to go until reaching those peaks of technological development, but until then we are confronting with unlimited opportunities of optimizations and quality improvements for our everyday life. Currently, one well-known challenge, that is of the highest interest and is in continuous development, is autonomous driving. Certainly, most people agree that in high amounts driving becomes tedious and exhausting, and we can most likely find better use of that wasted time. The increasing number of auto vehicles populating the world is becoming alarmingly high and this is a serious reason of concern from an environmentally point of view, but also from a travel efficiency perspective.­ Also another strong argument, is that by improving the quality of transport systems, we will decrease the number of accidents and, in return, elevate the safety of traveling.

Nonetheless traveling is crucial in our lives and regardless of which reason we choose out of the hundreds that exist, we will conclude that autonomous driving is essential, and perhaps even mandatory, for a bright future of our society. This is only one of the many subfields making up the present artificial intelligence industry that will change the future of our lives, our cities and our societies in the most wonderful and futuristic ways we can only hope and imagine.

## Motivation

Having the intention of contributing as much as possible to the development of artificial intelligence towards the previously mentioned ideals, I decided to pursue this subfield of autonomous driving hoping I can bring us just a little closer to the future.

We constantly find ourselves in the situation where a task is so complicated, that it needs to be divided into multiple smaller, simpler sub-tasks. This is very much the case when talking about autonomous driving since we are definitely in a need of an intelligent agent. Using a perception constructed from inputs (cameras, ultrasonic distance, inter-communication and other sensors), it needs to take a certain action (accelerate, break, steer etc.) whilst following a specific goal (autonomous driving).

The states of the environment are defining for the complexity and the difficulty of the problem. It is only partly accessible through the perception of the agent and the actions made may alter it, which makes it dynamic. The fact that it is not an enclosed, controlled environment forces it to be non-deterministic. Since it is strictly correlated to the precision of the sensors, it can be described as continuous. Most of the laws regarding the way the environment works and reacts to actions are known. The history of the environment is obviously very important and that makes it sequential. The numbers of agents present may vary depending on whether we are talking about only one self-driving vehicle or there is an entire network of them. We are of course referring here to the main characteristics of the environment: Observability, Staticness, Deterministicness, Discreteness, Knowledge, Episodicness and Agency [4].

Considering the impact and the importance of autonomous driving I decided to choose an auxiliary system used in the current autonomous driving system, I am talking about Tunnel recognition. It has many uses in any driving software and not only.

First of all, it can be used to prepare the camera adjustment (also for any other sensors) before the entrance in the tunnel so that the “Blinding effect” due to the sudden brightness change in the environment doesn’t cause so many problems. This can reduce the lack of accuracy during the first seconds after entering a tunnel and the first seconds after exiting the tunnel.

Second of all, tunnel recognition can be used in the process of strategizing, which is a very important part of the decision-making process. For example, information that the car is currently inside a tunnel or that the car is approaching a tunnel can influence the decision of overtaking another vehicle because of the usual tightness and lack of visibility in a tunnel.

Another very good use of the tunnel recognition can be to adjust the front lights of the vehicle. We are talking about the simple turning on of the head lights or direction adjustment of the lights, because if the tunnel has two-way circulation it can be dangerous if the front lights create a blind spot for any vehicle coming from upfront. Yet another adjustable system from inside the car can be the air circulation (which should be set to recirculation due to the relatively polluted air inside tunnels, and also to a certain temperature in order to avoid formation of dew on windows) or the windscreen wipers (which should not waste energy inside a tunnel for this task).

Also, if the car is connected with the environment or it has access to an updated database containing tunnel and bridges specifications, the information can be used if the vehicle has a certain height that could exceed a limit and create a dangerous situation.

## Problem definition

The problem should be defined as simple as possible as to be properly generalized and pragmatically applied to a multitude of situations. We want to predict approaching infrastructure on the road (tunnels and bridges) from a moving point of view located on a vehicle and using sensors present on the vehicle.

## Solution construction

Our solution involves an intelligent agent that receives the input from the sensors (on-board camera) as grayscale images and will predict whether there is an incoming tunnel (or bridge). The intelligent agent will run on the server (it is not designed to work real-time on a vehicle, not yet) and will consist of a convolutional neural network already trained on a database formed from similar data. We will create an environment where we can easily construct and test different networks to choose the best one for each situation or database.

## Article Structure

We already presented our personal opinion and motivation for starting this project, and also explained what the problem is and how we plan to solve is. We will continue by constructing a theoretical background containing most of the information an under-graduate (and even non-technical individual) would need to be able to fully understand the concepts, the usage and the results (by going through image processing, artificial intelligence, machine learning, and convolutional neural networks). After that we will present the already existing methods that try to solve this problem (or at least a similar one), and then present our method. We will go through the entire process of creating this project, from gathering the data to analyzing the results. In the end we will draw conclusions and prepare the next steps for this project.

## Personal contribution

We managed to create multiple algorithms that make predictions related to those road infrastructures (tunnel/no tunnel, tunnel entrance/inside the tunnel/tunnel exit) and also create a complete environment (with a user interface and active connection to a server) that allows the user to validate, test and even train models whilst gathering information about the whole process and most of the components. We also created a hardware implementation using a Raspberry Pi to test the algorithm in the real-world.

Most of the work is gravitating around the idea of using artificial intelligence to recognize tunnels and bridges, which is certainly a new approach (at least judging from the publicly documented work). We succeeded in taking advantage of neural network architectures and optimally solve this problem of recognition using multiple methods and strategies, while also documenting and properly comparing them all. The solving begins with a simple binary classification, evolves into a multi-class classification, and finally culminates with the active learning strategy.

# Theoretical Background

In this chapter we are going to cover most of the knowledge needed in order to understand the practical part of the project. From image processing to machine learning, we will try to briefly present the theoretical notions for an individual who already has quite a good grasp on basic general subjects of computer science (data base system, object oriented programming, data structure etc.), but also on basic mathematics ( mathematical analysis, linear algebra and statistics ).

## Image processing

### History

Digital image processing is a ramification of computer science that describes any algorithm that processes an image in digital form. It is a subfield of digital signal processing and it has started as a consequence of the conversion from analog images to digital images. In 1957 the first digital image that was taken (although  “[Bartlane cable picture transmission system](https://en.wikipedia.org/wiki/Bartlane_cable_picture_transmission_system)” was doing similar work sending photos across the Atlantic through a cable ), more exactly scanned, stored, and recreated in digital pixels was showed on the Standards Eastern Automatic Computer ([SEAC](https://en.wikipedia.org/wiki/SEAC_(computer))) at [NIST](https://en.wikipedia.org/wiki/NIST).



Figure ‑ [55] – first digital image

Naturally the following years the industry evolved, and in the 1960s research was active at multiple locations in America: at [Bell Laboratories](https://en.wikipedia.org/wiki/Bell_Laboratories), the [Jet Propulsion Laboratory](https://en.wikipedia.org/wiki/Jet_Propulsion_Laboratory), [Massachusetts Institute of Technology](https://en.wikipedia.org/wiki/Massachusetts_Institute_of_Technology), [University of Maryland](https://en.wikipedia.org/wiki/University_of_Maryland,_College_Park), and a few other research facilities. The main purpose at first was image quality enhancement, but it gained momentum and ended up being used in a wide range of domains: [satellite imagery](https://en.wikipedia.org/wiki/Satellite_imagery), [wire-photo](https://en.wikipedia.org/wiki/Wirephoto) standards conversion, [medical imaging](https://en.wikipedia.org/wiki/Medical_physics), [character recognition](https://en.wikipedia.org/wiki/Character_recognition). The wide range of applicability continues to grow even today, we are strongly relying on this science for all the multimedia systems out there, for the field of astronomy and space exploration, for the increasing capabilities of A.I systems to automatize many of our needs (especially in medicine, transportation, and entertainment).

### Introduction

One of the most essential characteristics of a digital image is the way it is stored in the memory, because while implementing any sort of image processing method that will be the first stage. Although every image format uses a certain form of optimization or even archiving the information, we could simplify the image to a multidimensional array. Since most images are 2D we can safely assume that its ‘height’ and ‘width’ will define the size of the first 2 dimensions of the image. In this imagined board every squared box represents one individual pixel and its value will determine the color.

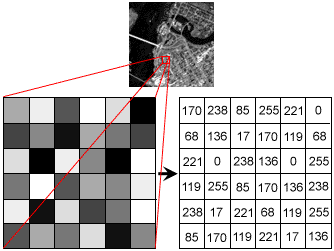


Figure ‑ [60] – grayscale representation

Mathematically we can define this grayscale image:

,

where ‘minGray’ and ‘maxGray’ represent the limits of the domain of values for the grays (it varies depending on the image quality, bigger interval means a wider array of colors).

Not all images are grayscale, if we are dealing with a colored image it will most probably contain multiple layers (usually 3 or 4, depending on the format: RGB, CMYK etc.). So the mathematical form of a RGB image will be:

,

where ‘minRGB’ and ‘maxRGB’ represent the limits of the domain of values for the colors (it varies depending on the image quality, bigger interval means a wider array of colors).

It is widely agreed that there are 4 main types of operations to be applied to an image: analysis, enhancement, compression and restoration. [5]

### Image analysis

This covers a very extensive array of operations, from scanning a barcode to face recognition, but generally it refers to the automatic extraction of information from an image with the intention to use that information in order to better understand the given image.

The applications are vast, but some of the more popular can be mentioned: medicine (detecting certain conditions or investigating one of the numerous types of scanning available: MRI, radiography, electrography, photoacoustic etc.), astronomy (calculating distances, sizes of solar objects, identifying new planets, or proving theoretical physics), security (eye or face recognition), defense (satellite image analysis and recognition), microscopy, optical character recognition, material science, robotics and last but definitely not least machine vision (household robots, industrial line machinery, autonomous driving etc.).

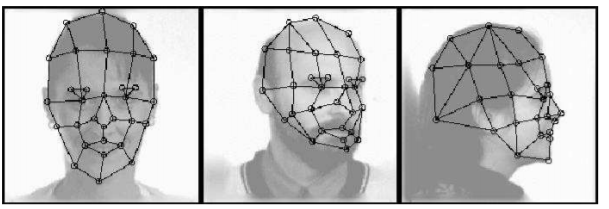


Figure ‑ [56] - face recognition

The techniques are numerous and complex, but to name a few: image recognition, image segmentation (splitting up the imaging into different areas of interest), video-tracking and motion detection (identifying moving parts using consecutive images) and many others.

### Image enhancement

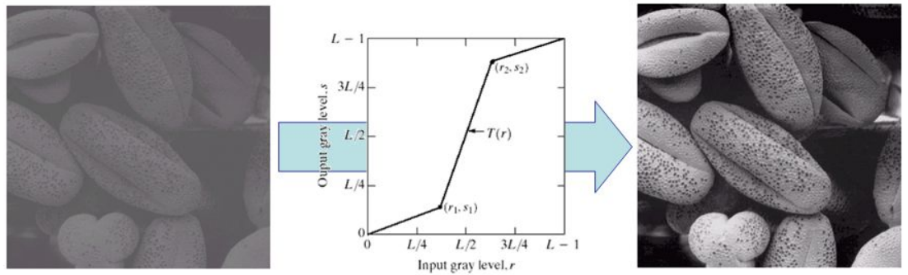
This type of techniques is also used to extract useful information from an image, but they try manipulating the structure of the image so that certain characteristics are highlighted. The fact that some aspects of the photo are pointed out can make it easier to analyze or recognize those aspects (for a human or another algorithm). These enhancements happen to be some of the most popular image filters used by the general public on the social media, but this is not their main purpose. ‘Deblurring’, histogram equalization, contrast and saturation enhancement are just some of the simple examples used to increase image quality.

Figure ‑ [57] – image enhancement

### Image compression

There are unlimited reasons to want your images to occupy as little space as possible: the limited physical storage space (happens on personal devices), the loading, downloading and uploading speed of the image (especially important for web applications), the need to store as many images as possible (essential for intelligent algorithms that need to be fed as much data as possible) and last but not least the money and time to make any of the task above come true.

We need compression of any type of file and that’s why archiving programs have gained so much popularity and also why the image compression techniques have been intensely researched. There are 2 big types of compression: lossy (examples: Discrete Cosine Transform, wavelet transform, color space reduction chroma subsampling) and lossless (examples: predictive encoding, entropy encoding, DEFLATE, adaptive dictionary algorithms). The difference between the two is related to whether or not you are losing information during de compression process (if the image decompressed is of a worse quality, or if it loses some detail). Depending on the problem you must decide what you are more interested in space or quality, because there is definitely going to be a trade-off there.



Figure ‑ [59] – compression example

Simplified compression usually either decreases the number of pixel and/or colors, either simplify the structure using mathematical models.



Figure ‑ [58] - image restoration

### Image restoration

This procedure means that an image does contain the desired information in a form clear enough due to some kind of disturbance (corruption of the file, noise, blur, bad mid-focus). This is different from enhancement due to the fact that another attribute is usually traded for another (for example reducing some noise may lower the resolution of the image). Also this kind of filter is not very scientific accurate, but manages to make images more pleasant to the viewer using techniques like: deconvolution and inverse filters, the Wiener filter etc.

### Image processing and artificial intelligence

There is a very interesting relationship between those two relatively different fields, perhaps a relationship of symbiosis because we can find that artificial intelligence uses image processing techniques in some of its algorithms and we can also identify intelligent methods in some of the image processing techniques. Due to this intertwinement there is a big confusion between image processing, artificial intelligence and computer vision.

Functionalities like object detection or image classification (and mostly anything from the field of computer vision) were realized, before the emergence of artificial intelligence, only using different mathematical models and certain filters. This was basically pattern or template detection using simple techniques from image processing, and to be able to upgrade those methods to the ones using artificial intelligence it would be necessary to assign a couple of connectors. Most computer vision algorithms today use neural networks (convolutional neural networks, recurrent neural networks, generative adversarial network etc.) and these require some form of input. The input is achieved using classical image processing operation like filtering or edge detection and the results will most certainly be better than the ones from the mathematic pattern detection (or at least it will have taken less time).

A very important part of any neural network is image processing, and more precisely it is used in two phases. The first one, as we have already mentioned, is in the preprocessing of the image by applying filter (or detecting edges, constructing skeleton, reconstructing etc.). The second one is right in the heart of the network and it is called a convolution (see Chapter 2.4).

In conclusion, machine learning can be used as a technique to solve image processing problems but combined with some image processing algorithm it can achieve amazing results as we can see in the field of computer vision.

## Machine Learning and Artificial intelligence

### Artificial intelligence

The origin of artificial intelligence has profound roots in the history of humanity under some form or another, varying from gods to advanced machinery or science fiction. It is very interesting to observe how the antiquity thinkers envisioned similar creation as the 21­st century scientists. One example is “Talos” from Greek mythology who was tasked with protecting Crete Island and was defeated by “unplugging” him from his source of power [6]. The Golem is a clay made intelligent legendary automata which have been given powers by the gods under the form of a piece of paper with writings on it (the symbol of knowledge at the time) [7]. It only seems that we are becoming the gods our most illustrious ancestors envisioned by creating the most unimaginable inventions ever. This is at the same time a great power, a power given by almost unlimited knowledge, but also burdens us with colossal amounts of responsibility.

Thinking evolved, and through the mysticism given by the human mind, turned into alchemy and magic. In Goethe’s famous second part of the tragedy: “Faust”; a similar character is created “Homunculus the artificial man”, through more of a biological process similar to cloning [8]. Modern science fiction automata followed, and it only seems inevitable that this artificial intelligence-based creature will one day come into existence by our hand.

Although the subject has long been on our minds the moment when it has been finally materialized into an organized thought reform was after the creation of programmable computers. “The field of [AI](https://en.wikipedia.org/wiki/Artificial_intelligence) research was founded at a [workshop](https://en.wikipedia.org/wiki/Dartmouth_workshop) held on the campus of [Dartmouth College](https://en.wikipedia.org/wiki/Dartmouth_College) during the summer of 1956. Those who attended would become the leaders of AI research for decades. Many of them predicted that a machine as intelligent as a human being would exist in no more than a generation and they were given millions of dollars to make this vision come true.” [9]

After immense financial investment, two ‘winter’ crisis and continuous research by the brightest minds in the world the field of artificial intelligence has developed to the state in which we find ourselves today. A.I., as it is famously abbreviated, is now used in any field of the industry and we can easily identify it in any segment of our daily life. The term is used for any intelligent machine that can mimic cognitive functions normally associated with humans, such as learning and problem solving. [10]

Currently another defying characteristic of an A.I is given by our ability to distinguish it from an actual human being, this is called the Turing Test [11]. Nonetheless this basic terminology (A.I.) has been intensively discussed and debated in the last years and we do not intend to go into detail since it is not our objective.

### Machine learning

It is often associated directly, or even confused with, artificial intelligence, but actually describes a more specific ramification of artificial intelligence. The denomination can be given to any system that uses information from a data source or from its own data-transmitting sensors (as experience) to improve its ability of decision or prediction making. Generally, if it learns by itself to identify certain patterns that have an effect on the problem, rather than being given the patterns, then we can safely assume that we are encountering a machine learning algorithm.

It is widely used in the industry and in the research fields, movie/video provider’s recommendation systems, speech recognition, physics simulations for fluids mechanics, animations and gaming, financial market analysis, bioinformatics, and robotics just to name a few.

All of the methods have the same purpose: making the most accurate prediction possible, and for this to happen the existence of a properly defined environment is needed. It is based on a mathematical model, but for the sake of understanding we are going to simplify things.

If we are talking about a prediction, we are either referring to predicting a number (called regression), either to predicting a class (called classification). Both situation require a main function that receives the input data – x (data from the environment regarding its current state) and gives back an output – y (the actual result which can be a number or a class): y = f(x).

It is mostly accepted that there are 3 main classes of machine learning judging by the manner in which the algorithms calculate the function f(x) and we are going to continue by examining each of them.

### Supervised learning

If the machine learning algorithm learns using human intervention in any phase of the learning stage, then we are talking about a supervised learning algorithm. Part of the is usually the source of learning for the algorithm, labeled prior to training by humans, and could be defined as pairs of (input, expected output): (x1, y1), (x2, y2), ..., (xN, yN). This data is used in multiple stages of the algorithm: a part of it will be used to train the algorithm, another part of it will be used in the evaluation of the algorithm. Using the same data for both stages may not reflect the actual accuracy of the algorithm.

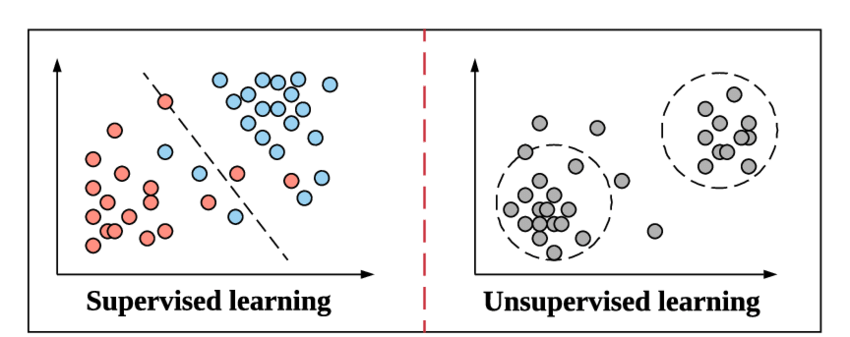


Figure ‑ [53] – dataset not linearly separable

This process can be compared with a learning toddler, we are constantly showing him a spider (it can be an object, a situation, or an information) and telling him explicitly that it is a “spider”. We would expect after some time for the toddler to be able to name a creature on sight by himself as spider or as something else.

Advantages of the methods are that we can decide what the classes are if we have a desired layout, the accuracy is very high and learns exactly what the humans want it to learn and it can usually be simplified to a mathematical formula.

A couple of disadvantages are related to the enormous need for data and the fact that the results are dependent on the quality of the labels. It implicates a lot of human intervention, it often overfits the data, and it is not generally that capable of handling very complex problems.

### Unsupervised learning

The machine doesn’t use human assistance during the learning process, and instead, what it actually tries to do is to use a certain loss or fitness function (that determines how bad or well the predictions are) to constantly improve itself. This approach makes the program less susceptible to our subjective believes and perception, and instead it tries to find its own logic and believes. The algorithms usually work based on a clustering method, trying to create a number of classes with certain features that encapsulate the given elements. The results are usually quite interesting because the machine finds either different classes than how we are imagining things, or it focuses on different features than we usually do. This can often end up being very useful when we are trying to find a different perspective on our problem.

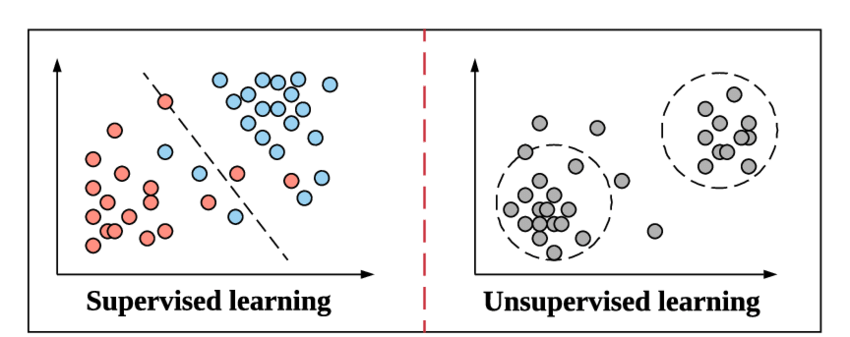


Figure ‑ [53] - clustering method

The algorithm receives only the input with no prior explanation (x1,), (x2), ..., (xN) and must find associations, similarities and differences between the input based on which the f(x) functions.

We can compare this type of learning with the way infants learn to speak in their first years of life. Babies do not understand the concept of letter or written words, they have no prior knowledge of our language and our way of thinking, yet they need to learn what every word means without anyone telling them. The infant seems to learn in a statistical method based on the frequency sounds (or phonetic groups) as this notorious paper states [12]. Based on how often two sounds are put together in speaking they tend to be associate by them more, and sounds first become words without them knowing what they mean (it is probable that the sounds are also associated with what the infant is seeing, hearing, smelling at that moment too, but with no actual meaning), and only afterwards the meaning of the words is understood and memorized.

Among the advantages are the fact that it is faster and it manages to find unforeseen links in the data (this is particularly helpful for scientist), but it is much harder to be used to identify the same features as a human would do and requires a lot of data.

### Semi-supervised learning

This specific type of learning is actually the result of a hybridization between supervised and unsupervised learning. The purpose for their combination is strongly related to the problems posed by limited data bases. Many problems do not offer large databases of millions and millions of labeled entries available for training, and to be able to solve those problems we need to use the best of these two methods.

We start with an initial small database of labelled data which we feed to the algorithm, then it will be then able to predict with a decent accuracy (usually a low accuracy) any new data. The goal is to gain better accuracy and increase our database, and we are going to do this by taking advantage of the trust of the algorithm on certain predictions (the percentage of certainty that he is right). If the trust is high, then it is most likely that he is right, and we are going to add the input with the predicted label into the second generation of the database. If the trust is beneath our chosen threshold, then we are going to predict the label manually, and add the input with the manually added label to the second-generation database. The algorithm can train again on the new database and it will most likely acquire a better accuracy.

This case can also be compared to a child’s, or rather, a responsible child, that ,although, he has been thought what a spider is (exactly like in the example from supervised learning) will come and ask his parents every once in a while if what he has found is a spider or not (because he is not sure), thus reinforcing the correct prediction and retraining on the weak points (unfortunately this involves the child coming across more spiders than a mother probably prefers).

The advantages of these techniques are that less data can be used to achieve the same results and less human labor is required, but on the other side bad behaviors can be reinforced if the trust is high.

### Reinforcement learning

This category of techniques relies on a very interesting psychological concept of “Classical Conditioning” first introduced by Pavlov [13]. He first popularized the effect that positive and negative feedback has on the behavior of animals, and human alike.

With this in mind a learning algorithm can learn what a good thing is by being punished and rewarded (or mathematically speaking, by reducing a cost function or achieving a higher score). It seems very similar to how we train any animal to do a particular task, and it’s widely used to create algorithms that learn to play games (or game-theory problems).

## Artificial Neural Networks

We will start by presenting a little history related to the beginning of the neural network simulations and then proceed by examining the defining components of the artificial neural networks in the present day.

### History

Artificial neural networks were born it a very interesting way that all started in the 1940’ with the numerous neural scientist that were trying to find out how exactly does our brain work. They conducted numerous experiments and at some point, most of them realized that those experiments (that basically tried to simulate a neural network using electrical hardware components) could be used at a number of different problems, like visual recognition.

In the year 1949 Donald Hebb constructed the support for what the next generations of scientists would use in the computer science and neuroscience fields [14]. The famous saying “Neurons that fire together, wire together” was born and the notion of neural plasticity (the property of neurons to form new connection, enhance or drop the old, already existing ones based on the frequency of which they are activated) gained momentum in the scientific community.

In the following years many scientist, among whom neurophysiologist Warren McCulloch and mathematician Walter Pitts, continued to make important discoveries and conduct experiment after experiment [15].

The initial desire was to be able to create an accurate simulation of the brain using hardware, although now we know they ended not up achieving that goal. That partly happened because a new architecture began to gain popularity: von Neumann, based on which our current computers are designed and constructed, and this architecture is not the most efficient if we want to simulate large numbers of operations done asynchronously (like in the brain) [16].

From the 1970s till the present day the research has been mostly focusing on artificial neural network simulated through software, mostly because of the increasing computational power available with which useful little classification and regression problems have been solved. There have been two ‘winters’ in which the research process has basically collapsed due to economic and technological reasons, but it has never died and now it is gaining more and more interest from both the scientific community, as well as from the multinational private companies.

### The Neuron and Backpropagation

Artificial neural networks or ANN are one of the most used supervised techniques of learning at the moment. At the core of this technique stands, very intuitively, the neuron and its origins go way back in the 1950s. Frank Rosenblatt used the theory created by his predecessors: McCulloch and Pitt to create the “Perceptron” [17]. This actually consisted of a set of neurons connected and able to adjust the weights of their connections using input data to classify simple shapes. It worked, but its limitations created a stop to the research in this direction (it could only work on linearly separable classes and could not learn a simple task like the exclusive OR logic function).

Backpropagation was invented and assured once again that neural networks are the best inspiration for a learning algorithm (the method is discussed in more detail in Chapter 2.3 – Backpropagation) [18].

The perceptron works in a precisely elegant manner, we will need to define ourselves a couple of variables first:

* , the learning rate of the perceptron (the bigger ‘r’ the faster the weights change)
* , the input vector of the neuron
* , the output results
* , training set compose of ‘s’ samples where:
  + , the input vector with ‘n’ dimensions
  + , the desired output
  + , value of ‘i’-th feature of the ‘j’-th training input vector
* , the ‘i’-th value of the weight vector
* , the first value or bias

The algorithm starts with the null or random initialization of the inputs, and the initialization of the threshold. For every input vector and the current network, we calculate the results:

,

and then update the weight:

, for all feature: .

Simplified, the neural network wants to produce a simple answer using the input. Every input is preprocessed and then given to the first layers of neurons. Each neuron only knows to receive signals from its connections in the first layer ( the input layer ), and then if enough signals are received and a certain threshold is reached, the neuron will also fire and send a signal to the next layer. The communication on a regular neural network is directional, every layer of neurons only sends signals to the next layer of neurons. This is the reason for which the networks must not contain any cycle, or bidirectional connection. We can chose to hardcode every neuron’s weight, threshold and connection to obtain a specific result from our input, or we can use certain intelligent techniques that with lots of data, will train themselves (tweak the weights, biases, filters, connections etc.) in order to best describe the dataset. (as it can be observed in Figure 2-9 and figure 2-11).

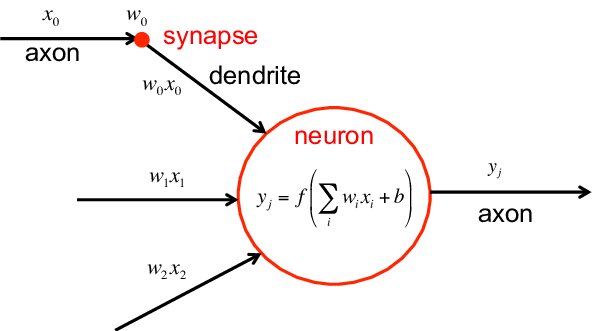


Figure ‑ [61] – the neuron inputs and outputs

### Activation function

The activation function is inspired by our anatomy once again. Cells like muscle cells, neurons, endocrine cells and others possess the property of ‘action potential’, better known as ‘nerve impulse’ or ‘spike’ (in the case of the neurons). Every neuron receives signals by means of voltage spiking, signals that are further transmitted along the axon of the neuron. The voltage travels because a chain reaction is created by the polarization and depolarization across the membrane. Sodium and potassium gated ion channels will quickly open and close creating a magnetic driven ‘wave’ that sends the information toward the axon terminal, that will send the signal onward. This voltage spike will need to exceed a certain threshold potential in order to cause this chain reaction, otherwise the signal will be too weak. [19]

This is the main inspiration for the activation function of the neuron, which will either return a ‘True’ value (if the threshold is exceed),or will return a ‘False’ value (if the threshold is not exceeded). But the activation function does not need to be just a ‘Binary Step Function’ ( Heaviside's Unit Step Function) [20], it may not even be limited to sending True/False signals. Non-discrete values are actually more popular due to the power of learning complex patterns with fewer neurons, their continuous threshold is given by many activation function (view Figure 2-10).

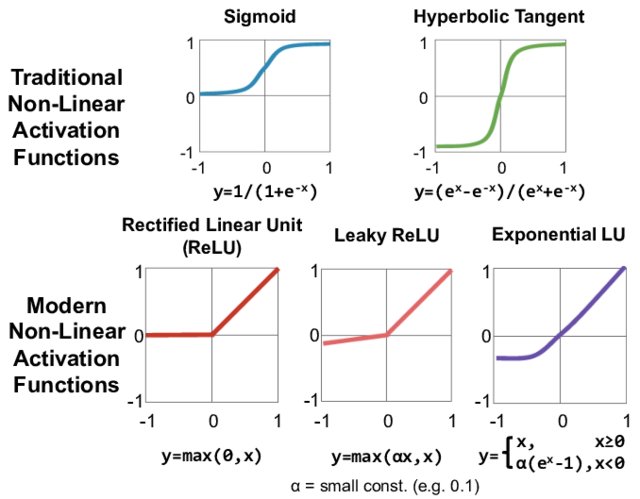


Figure ‑ [61] - activation functions

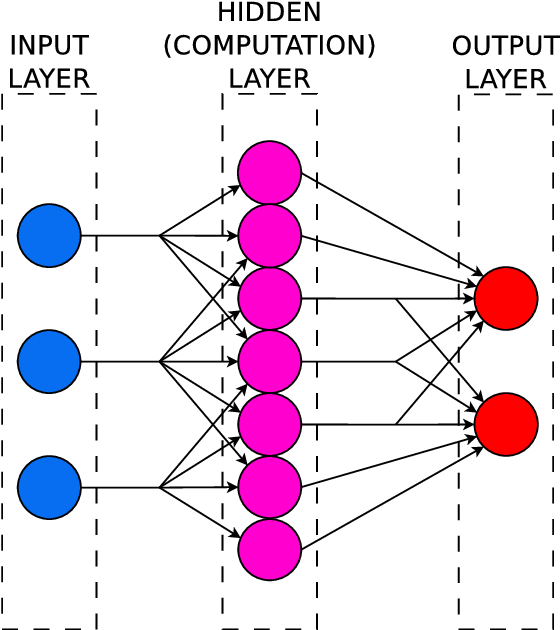
### Loss function

Every artificial neural network tries to make prediction that are good, or at least good enough. The ability to define how well a network is doing on a particular prediction comes down to this function. Stochastic gradient descent usually works well with backpropagation, but this optimization method needs an error gradient. After a set of predictions has been made the loss function determines how well the network did, and then it will calculate how to tweak the weights for the next iteration (the worse the prediction, the more severe the gradient change of the weights). [21]

Perhaps the most important characteristic of the problem that determines what kind of loss function would work best regards the output layer. The main types of problems (and one of the specific loss function for each) that we are considering are: Regression problem (Mean Squared Error Loss) , Binary classification (Binary Cross-Entropy), Multi-class classification (Multi-Class Cross-Entropy Loss). [22]

### Layers

Figure ‑ [66] – simplified architecture



There are three main types of layers: input, hidden and output, and each of them contains different nodes. First, the information from the outside world is provided to the input nodes that, without any additional computation, just pass it next to the hidden nodes. Those are going to the ‘hard work’ for the network and can be organized in either a ‘Single layer perceptron’, or a ‘Multi-layer perceptron’ depending on whether it has only one hidden layer or not [17]. In the last layer we find the output nodes that work similarly, but the major difference is that the output of each one is the predicted class (or value if the problem is a regression).

### Types of ANN

There exist a multitude of variations based on these few principles, all of them have having these former mentioned characteristics in common (or at least a subset of them). We will mention some of the most popular and widely used artificial neural networks and briefly explain them.

A Multilayer Perceptron  is one of the simplest ANN and the main features are the fact that it has more than three hidden layers, that it uses nonlinear activation function and most importantly that is fully-connected (every node from one layer is connected with every node from the next layer). [23]

A Recursive Neural Network (RNN) is based on structural prediction of the input. The same network with the same weights runs multiple times on parts of the input only to achieve a better topological view of the input. It is used mostly in natural language processing, but also in some computer vision segmentation techniques [24].

A Recurrent Neural Network (also RNN) constitutes a common variation of the classic ANN that introduces the possibility of a directed cycle. This is often used in the situation where the previous state of the environment is also very important for the prediction (because each layer also takes into account its previous state). Some popular usage of this architecture are document summarizer and handwriting recognition [25].

Last but definitely not least a Convolutional Neural Network (CNN) is the one we are going to use in this project. It uses the feed forward method, weights and biases to classify images or recognize objects, but the most important characteristic is given by the various types of layers that it contains in order to manipulate the image, categorize small sections of it and then make sense of the whole picture. We are going to further explain this type of ANN in the next subchapter (2.4) [26].

## Convolutional Neural Networks

We are going to present the CNN mechanics, explain how and why it works, and then we will also discuss the particularities of every type of layer. In the end we will try to see the CNN with more pragmatic eyes by discussing pros, cons and certain particularities.

### Description

First of all, CNN is a variation of the Multi-layer perceptron, but with a couple of changes that we are going to discuss immediately. Its application must be considered beforehand because they have led CNNs to become what it is today, and we are talking of course about computer vision. Although it is also used in natural language processing, finances and recommendation systems, the most important applications are in computer vision (object recognition and image classification).

Because an image is the input, we must take into account the huge number of potential input parameters if we were to only use regular MLP. Let us take as example an image of 2 megapixels, that translates to a resolution of 1920x1080, and if there are 3 color channels (RGB) we will end up with over 6.2 million input parameters. This is a huge number and a waste of potential computational power, also the neural network will most likely end up not learning properly and will lack the perspective (ability to consider the whole picture whilst recognizing an object or class).

A very important concept of image processing: filters, could significantly simplify the whole process of learning. Instead of taking every single pixel color by itself, the network will take sections of the image and apply filters to it in order to make sense of it, then it will divide the current sections into smaller ones and so on. Ideally this process, if the right filters are chosen and there is enough data, the network would manage to divide the image in sections (usually squared sections) and first recognize simple geometrical figures (straight edges, curved edges, point, circles lines etc.) and then gradually put them together into subcomponents of different sizes and shapes (tire, headlight, windscreen, car door etc.) and finally into different components (sedan, truck, pickup, suv. etc.). This concept enables current CNNs to learn so many different categories with one single neural network (with tens of layers and millions of parameters).

Before CNN made its debut, the formerly mentioned filters were hard coded into the network, but this was inefficient (because it is usually very dependent on the problem) and a very tedious work. CNN will use filters but will try to learn all of them by itself in the training phase, but only filters being applied on parts of the image is not enough. Next we will discuss each different type of layer and what is its contribution to solving this problem [26]

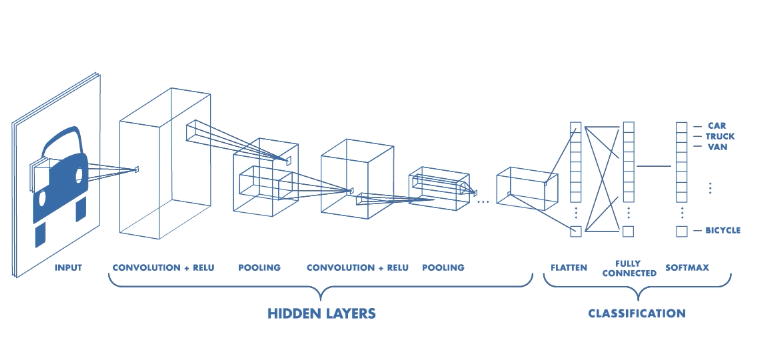


Figure ‑ [62] – CNN layers visualization

### Convolution layer

We are beginning with essential component of the CNN: the convolutional layer. It strongly relies on the filter (sometimes called kernels) that will be learned during the training. The dot product between the filter and the input of the neuron will result into a bidimensional activation function (or activation map). One very important matter is choosing how to split the input between the neurons, and then we encounter a series of questions: ‘How big should the input (and the filter) be ?’, ‘Should there be any overlapping of the inputs (if so, how big) ?’, ‘How many inputs should there be?’. In order to answer all these question, we must analyze our particular problem and only then we could conclude what would be the best choice, but usually it is discovered empirically.

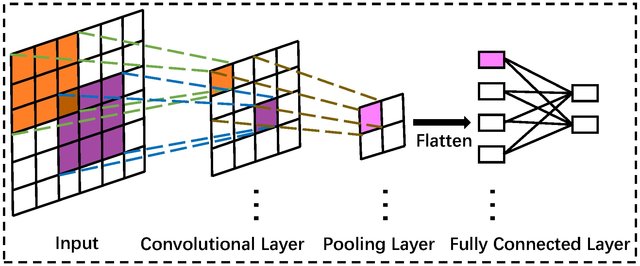


Figure ‑ [63] – main layers mechanics

### Pooling layer

This layer’s main purpose is down-sampling the image in order to simplify computations, but also give the network perspective regarding a bigger surrounding area of the image. Gradually decreasing the perspective, we manage to achieve local connectivity.

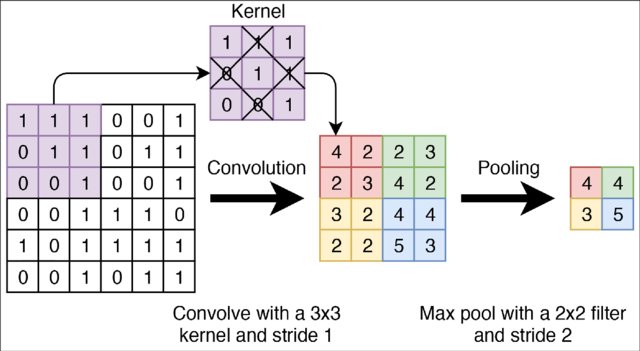


Figure ‑ [64] – pooling

Achieving smaller size and keeping quality is a great problem in image processing (image compression), but this is not exactly the same situation since we intend to propagate the essence of information (which might not always be the same with the simplification of information). In Figure 2-14 we have an example of Max-Pooling, simply keeping the biggest value from the map, but there are many other techniques: L2-norm pooling, average pooling etc.

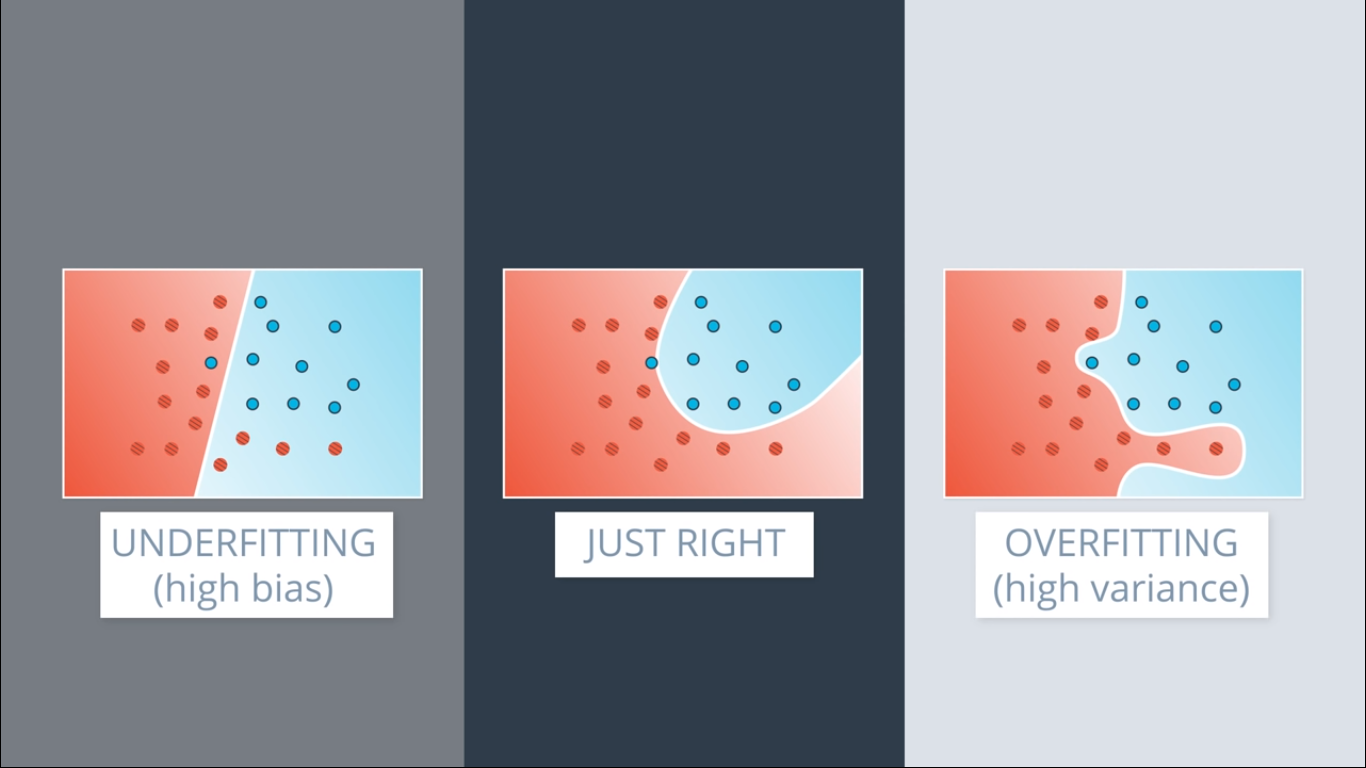
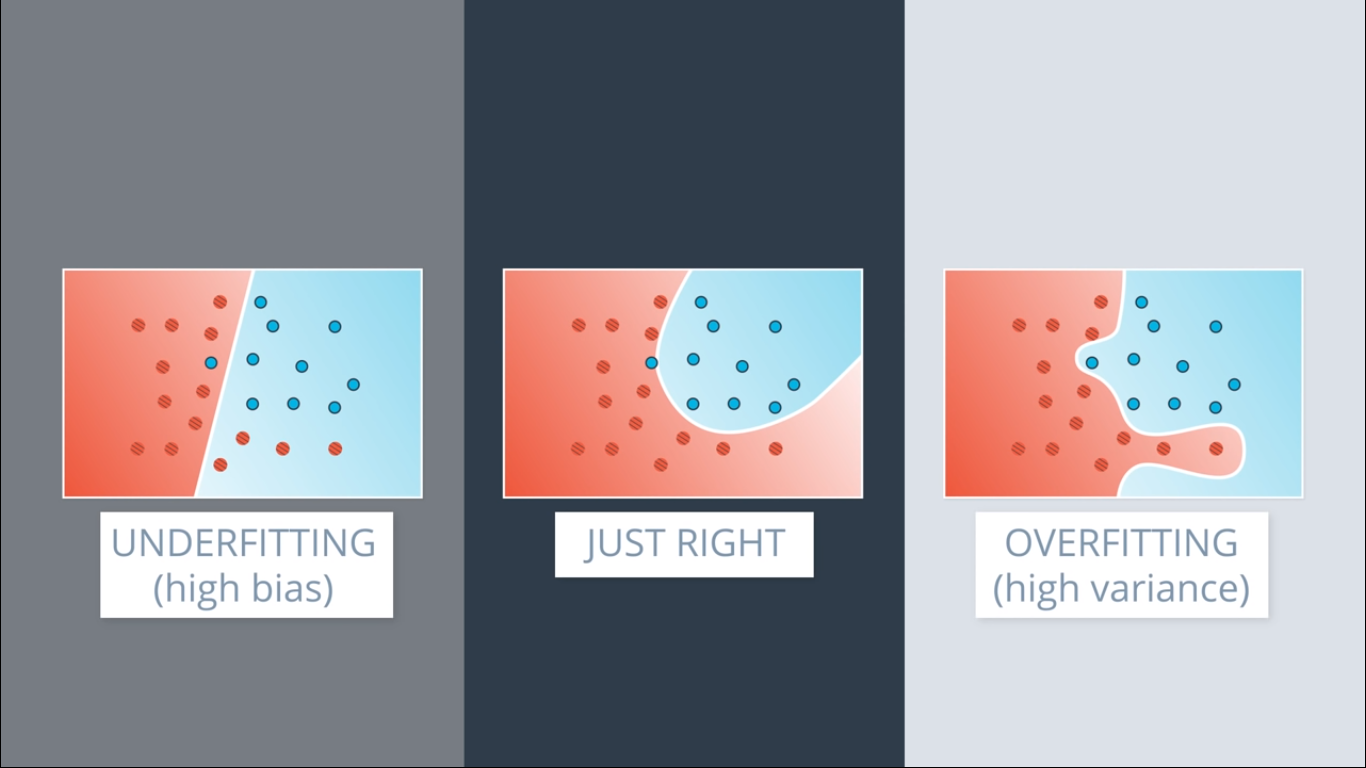
### Fully Connected layer and Loss layer

The last couple of layers are usually fully connected in order to give the network the chance to put all the information together and be able to make a correct prediction. It will take as input the output from the hidden layer which already has a reduced size containing the essential information in each area of the image. (see Figure 2-13)

The very last layer is the loss layer that determines the penalization manner for the training phase using the aforementioned loss function.

### Overfitting and Underfitting

There are many things to take into account when deciding what layers to use, what configuration to choose for each one, and in what order to assemble them. If we do not succeed to make the right choices for the setup of the network we will end up in one of two situations: overfitting or underfitting the data.



Overfitting

Underfitting

Figure ‑ [65] – two learned equation on the same dataset

The cause for both the problems is actually the fact that the network is only learning from a subset of information from the actual complete dataset of information. The test input (for real world predictions) is often not the same as any other input encountered during training and also the actual training data is much smaller than the testing data (because of the computational power and available time for training a network).

Overfitting happens if the model learns very good the training data set (high variance), then it will be very specific about the boundaries between the classes (as we can see in the Figure 2-15). This is often not desired because anomalies can be present in the data and alter the predictions. Many factors can drive a model to overfit: the model is too complex for the desired learned equation, there is not much data available. We can try a multitude of methods or improvements to reduce overfitting: Cross-validation, early stopping of the training, pruning, regularization (by reducing the error, adding weight decay, adding drop-out chance for a neuron, L1/L2 regularization), adding more data to the training set (using bootstrapping or augmentation), removing features, or even upgrading our current convolutional neural network to an ensemble learning.

Underfitting occurs when the network does not manage to learn well enough the data set and it ends up over-simplifying the problem (as can be seen in Figure 2-15). This doesn’t happen as often, and when it does it is because there is too much data, the model we are using (or the learned equation) is too simple relative to the complexity of our problem. We can solve this problem using multiple techniques: adding more data, increasing the training error (penalty from loss function), adding more features, or increasing the complexity of the network.

## Active Learning

Training a model requires collosal amounts of computational power, time and data. Data is perhaps the most important one and without it you cannot train an efficient model with all the computational power and time in the world. Collecting data is complciated and expensive, but that is not as big of a problem as preparing it. Labelling is one of the most tedious and time-consuming activity needed in order to train a neural network. More over if the labelling process requires someone with professional expertise it can also become really expensive. Active learning is one of the best strategies to use when having a few labelled data entries but enough unlabelled ones.

The way it works is relatively simple and although each active learning strategy works differently, all of them have some details in common. They all seek to train a model on the data already existent, and then using this model they analyze the unlabelled data and come up with a list of data entries that should be labelled manually. The user is interogated and he provides the information needed. After that, the initially unlabelled dataset has become labelled and the user did not had to do all the work. The initial labelled dataset can now be augmented and the model can be retrained. In theory the model should become better with each iteration because its good predictions are reinforced and the bad ones are corrected.

The strategies vary mostly by the way they choose the entries to be labelled, but also based on the model it is applied to. We are going to explain one of the strategies (the one we are going to use) called ‘uncertainty sampling’. The data labelled by the model is split by a pre-set threshold (as can be seen in Figure 2-16) based on precision or trust. If the model is not sure if it is the right prediction, then it is going to ask the user about it. After this we can augment our database and retrain the model.

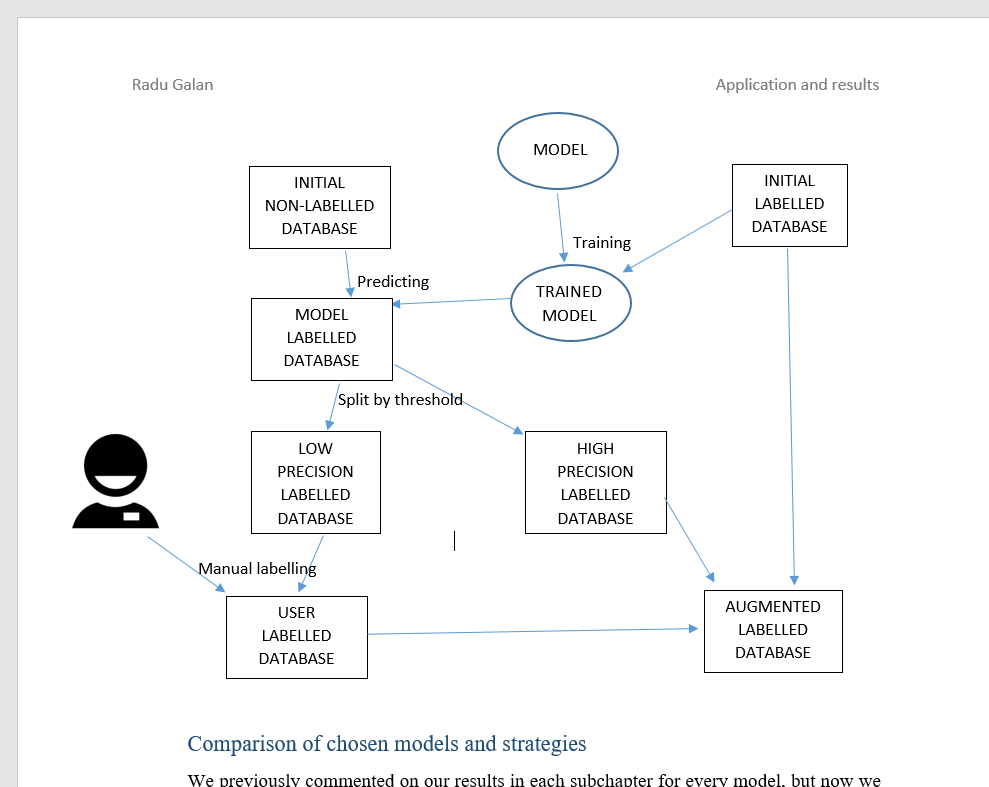


Figure ‑ – active learning steps for augmenting a database

# Literature Overview

We are comparing different strategies for detecting Tunnels and by this we understand any form of algorithm, technique or applied method that can classify data as to conclude if the environment contains or not some form of tunnel. The classification can be both binary and multi-classification, it can use any form of data as long the data is gathered from a moving target (the vehicle) and the tunnel in cause is a Road Tunnel.

## State of the art/Related work

In this chapter we will present all the methods analyzed and explain the pros and cons.

### Fast Vision-Based Road Tunnel Detection [27]

The intentions of the paper are to create a tool that uses images to recognize tunnels and then passes that information to an “Advanced Driver Assistance Systems” which can better fructify the information to predict GPS error due to signal loss inside tunnel and also the creation of a protocol for adapting the sensors to the low-light conditions.

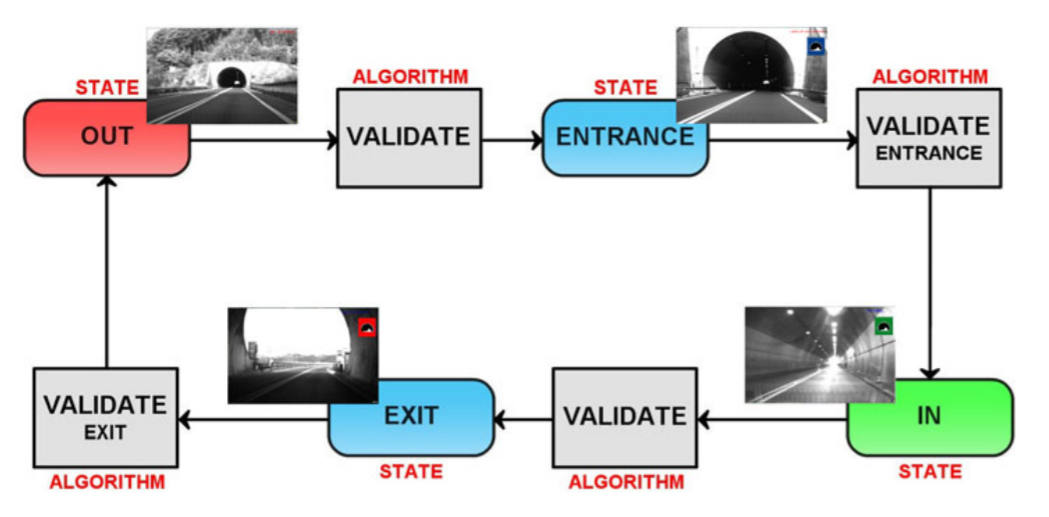
The algorithm uses four states: outside the tunnel, at the entrance, inside, at the exit. The grayscale images received as input are reduced in size for the following computations. The method takes advantage of the fact that the four states always come in the same order. For example, after you recognize that you are inside, you only need to check if you are still inside or approaching the exit, because the algorithm is supposedly running live on a vehicle. They have observed this property of order in the states of the classification.

Figure ‑ [27] – the tunnel separated in states

After the entrance is validated there is no need to check whether you are inside the tunnel or not, because the next state is always the exit. Here we can observe one of the limitations of the algorithm. It cannot tell from the environment if the vehicle is inside a tunnel or not, it only guesses that is still inside a tunnel using temporal information. If the images were not in perfect order or the recording would begin while inside a tunnel the algorithm would not predict properly until the exit of the tunnel.

The algorithm that predicts the entrance and exit is fairly interesting and simple. Using the histogram of the image and a dynamic threshold it can recognize a pattern. When high contrast is detected on both the vertical and horizontal histogram, their intersection will constitute the actual bounding box for the tunnel entrance/exit. If the contrast constitutes a darker portion it is an entrance and if it constitutes a brighter portion, then it must be an exit.

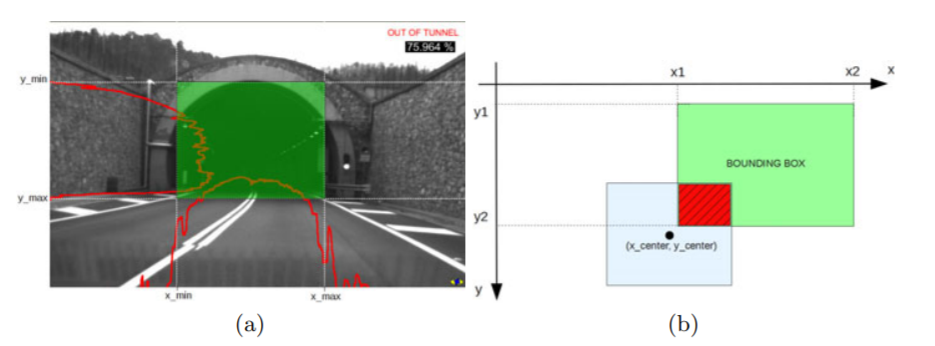


Figure ‑ [27] – vertical and horizontal darkness histogram

in creating the bounding box on a tunnel entrance

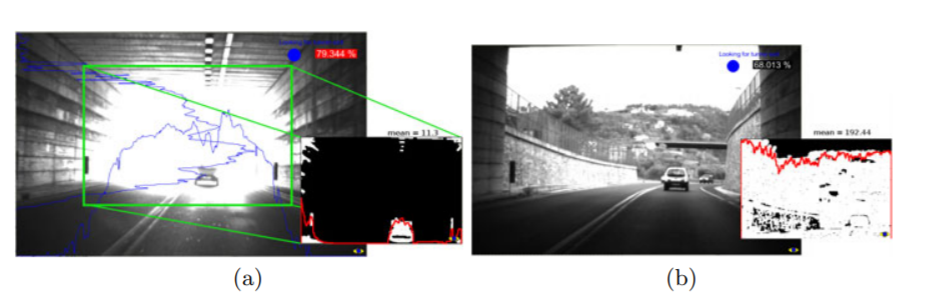


Figure ‑ [27] – vertical and horizontal brightness histogram

in creating the bounding box on a tunnel exit

### Road Tunnel Entrance Recognition System [28]

This is a whole master thesis that covers the recognition of tunnel entrances through a diverse number of techniques, most of them are using pattern recognition and image processing. The motivations of the paper are somewhat similar to the aforementioned article [27], but with more concern on the safety side ( trying to reduce the risk of accidents in tunnels due to visibility loss and the formation of dew ). It uses as a foundation the work of another article [29]

They use multiple methods of image processing that intend to capture disjunct characteristics of general tunnels. Afterwards they use the individual methods together to create hybrid methods that use multiple ideas and patterns. Colored images constitute the database.

The first method applies edge recognition and generalized Hough transformation to identify potential signature shape for a tunnel entrance (this is the “Tunnel arch detection” method). What it actually does after identifying all the remarkable edges in an image is to calculate which of those have the most similarities (or the least error) compared with a circle.



Figure ‑ [28] – “Consecutive tunnel lights” method

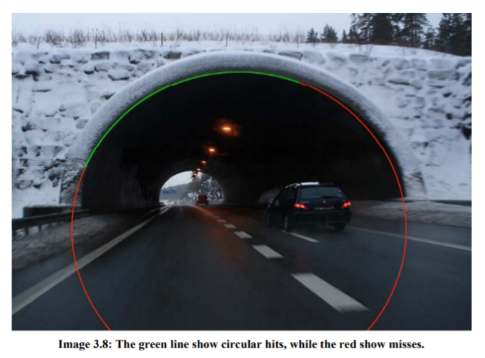


Figure ‑ [28] – “Tunnel arch detection” method

“Consecutive tunnel lights” is the second method that tries to identify light sources on the upper part of the image, this usually indicates that there is a tunnel which is often also a frequent exception for the first method. This is a bold assumption, but at least for the given dataset it works just fine. It is not based on edge detection, but on a method that identifies sudden brightness changes.

“Elliptic black hole” method is designed to identify the tunnel entrance using the contrast of darkness created inside, specifically any dark areas are compared with an ellipsoid. It uses a simplified form of gradient descent for a regression by adjusting the formula of the ellipsoid. It also takes advantage of a lane tracking algorithm to acquire better segmentation and it only works for circular tunnel entrances.

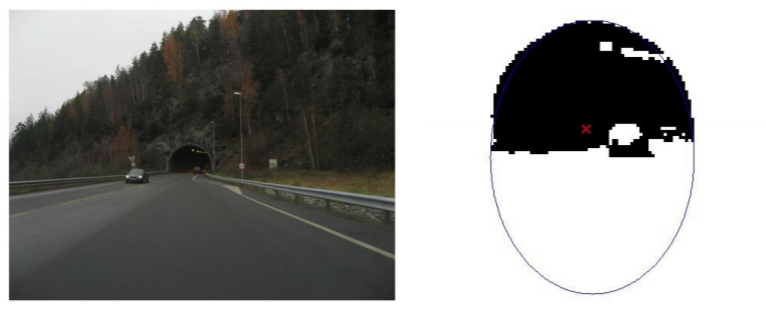


Figure ‑ [28] – “Elliptic black hole” method

“Template matching” searches exactly for certain templates in the image (tunnel hole and tunnel lights) pixel-wise. This method relies strongly on the fact that most tunnels respect certain condition of design and architecture, conditions defined as patterns and written as mathematical functions in the code.

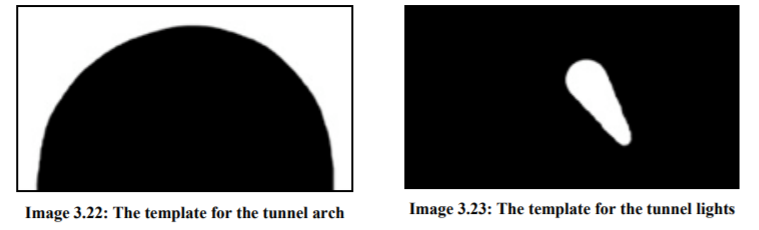


Figure ‑ [28] – “Template matching” method

Afterwards he experiments with a couple of hybrid methods. Considering how radically different those methods are it is safe to assume that their prediction could almost be disjunctive, therefore combining them could increase the performance significantly.

### Tunnel detecting device for vehicle and ... [30]

The third and final reference, found out that recognizing tunnels using a camera is a patented project in the USA about which we were unable to find many details, but it seems as if the inventor used a camera fixated on the room mirror with which he would be able to detect potential tunnels using brightness differences given by a certain horizontal limit of the image. This limit splits the image into “obliquely upper area and forward area”. The process is rather complex, but basically uses a user-initialized threshold of brightness difference that could predict if a tunnel is coming within a certain distance.

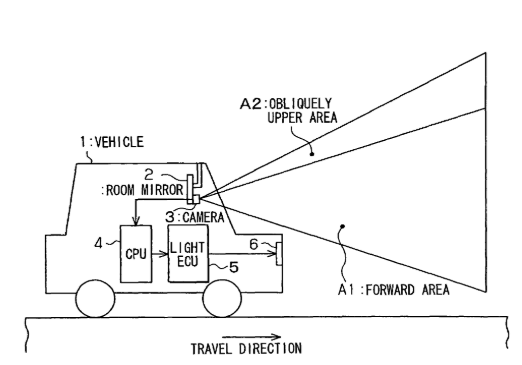


Figure ‑ [30] - patent for tunnel detection

## Implementation and results

We try to present objectively the results of each study focusing on the precision of the prediction, the database and runtime of the method.

### Fast Vision-Based Road Tunnel Detection [27]

There have been little documented testing situation (3 sequences – for 43 minutes), all of them during the day ( from obvious reasons ) and presented relatively many false positives ( twin tunnels, bright reflection on entrances, consecutive tunnels, tall trees and building on the side of the road ). Although the accuracy is great, the method is not versatile or scalable and the results are working for a limited array of cases. The thing that is truly impressive is the run time (750 nano-seconds), which means this routine can run on 30 frames along with thousands of other similar routine in this “Advanced Driver Assistance System”.

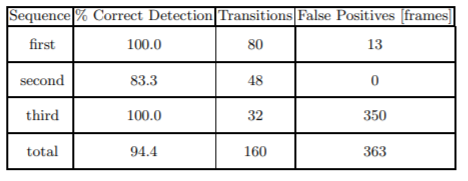


Figure ‑ [27] – results by sequence

### Road Tunnel Entrance Recognition System [28]

The different methods give somewhat similar results that average around 70% with various types of false positives. Each data set consist of 100 images from 3 recording session, also the formation of the training set is not very well explained, but from what we have understood there is no actual training, only manual improvement based on some images and only in some of the methods ( for the “Elliptic black hole” method especially).

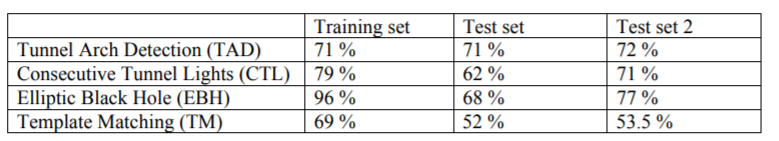


Figure ‑ [28] – results by method

The average time to run on one frame (on the used system, which is considerably weaker than the one used in the first article) is in the time frame of (0.5 seconds, 1.2 seconds].

Although the time to run the method is twice as high, both hybrid methods presented in the article achieve a good 10 percentage of extra accuracy. Disjunction between the CTL method and all the others seems to be the greatest and consequently these two hybrids perform better on the data set.

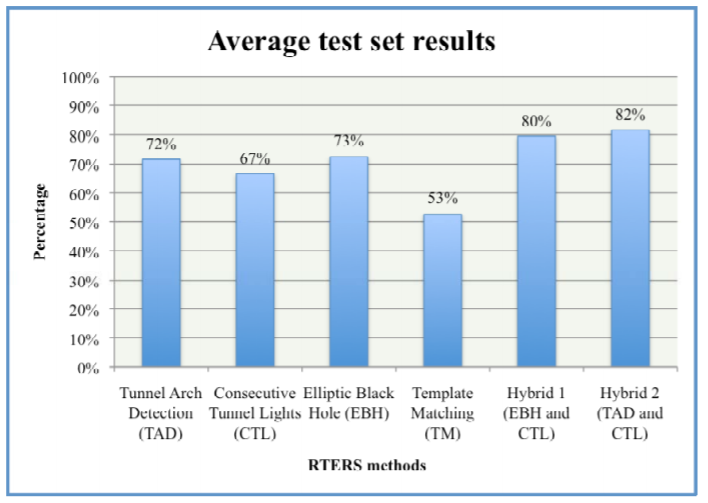


Figure ‑ [28] – results for methods and hybrids

### Tunnel detecting device for vehicle and ... [30]

No official results, accuracy or dataset has been publicly released about the mentioned subject and no other online sources have been identified to research, debate or investigate the study.

## Differences and comparison

### Fast Vision-Based Road Tunnel Detection [27]

The algorithm presents couple of interesting features and manages to subtract a simplified information very fast. Speed is one of its most important strengths based on the fact that the algorithm is only one optimized pass through all the pixels combined with the calculation of a threshold formula. 750 nanoseconds is insanely fast comparative to a learning algorithm. Another advantage is that being simple is also easy to implement.

The method also poses some disadvantages regarding the versatility of the program. It is limited to daytime condition due to the fact that is uses brightness, also it confronts many false positives and false negatives for the reason that similar brightness situations occur often in normal conditions (buildings, trees). Adaptability is also a big issue given that the algorithm does not learn, but follows a given pattern. The idea of states, although practical it only works in certain conditions and creates a lot of exceptions. It supposedly delivers an 80-90% accuracy, but it must be mentioned that the testing condition were quite limited and selective.

### Road Tunnel Entrance Recognition System [28]

The article studies plenty of methods that mostly use a mathematical base and image processing techniques and not only presents interesting result for every method, but goes further and tries to combine the best of each method into a hybrid logic-based method that gives considerably better results with up to 80% accuracy.

The data set of this article is much smaller in size and only uses daylight conditions in limited area around Oslo mostly during the wintertime. The runtime of the methods is quite high ( 0.4-1.2 seconds for normal methods, and 1.3-1.7 seconds for the hybrids ), but the fact that it is running on a quarter of the computational capabilities ( comparing to the algorithm used by [27] ) has to be considered.

### Tunnel detecting device for vehicle and ... [30]

Although the idea is certainly interesting, it may seem rather simplistic after the other two articles and the fact that we were not able to identify any official research or statistic on the capabilities of the technique leads me to believe that the method was not practically implemented ( or at least the results were not publicly reported ). The algorithm would most certainly be fast if well optimized, but the quality of the prediction would be surely unreliable (the author even admits that bridges create a strong false positive).

## Conclusions

We have presented and compared 3 main researched methods found in the academic environment that one way or another manages to solve the same problem: tunnel detection. The methods used were rather interesting and different even though all of them relied on some form of pattern/template for the tunnel entrance at least. Mathematics and image processing were the underlying fields that provided the actual method’s inspiration.

It should be mentioned that two more papers were identified that resolved ( directly or indirectly ) the same problem.

One of the methods used high quality image data provided by moving satellites fed into a neural network to detect roads, bridges, and tunnels. The idea is based on the fact that two road segments identified and split by a relatively small space may contain a bridge or a tunnel. [31]

The second mentioned research with similar objective was a paper that uses Radar Signals to identify iron tunnel taking advantage of reflection and diffraction of radar sensors. [32].

Considering all the methods analyzed it is not obvious which of the main 3 is best. We do not possess enough conclusive data for the third article [30], and we can only compare the other two. Out of these two a more interesting approach and most probably a better accuracy over a large array of data has the Road Tunnel Entrance Recognition System [28], but the statistical better one regarding accuracy and data base size (not to forget the huge difference in runtime) is the Fast Vision-Based Road Tunnel Detection [27].

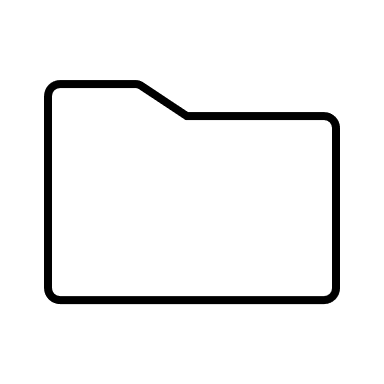
# Application and results

In this chapter we will begin by presenting how we intend to work and what we will work on, also we will analyze the methodology and every step on the way.

## General presentation

Our problem is obviously tunnel recognition using computer vision, but there are so many ways available to solve this that we decided it is only fit to create a general environment for our problem (also for similar problems). We will then compare the different methods implemented and try to experiment with them as much as possible in order to assimilate knowledge relative to both the database and the intelligent methods. The environment will be designed for our system specifically, and for this we will need to first present the system we are working with.

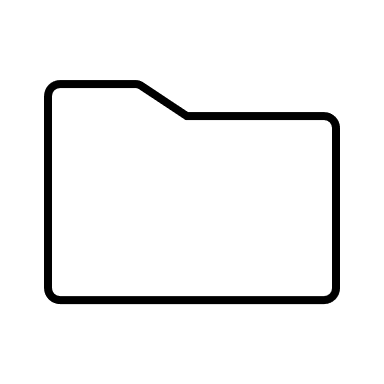
Figure ‑ – simplified architecture of the project



**Server**

-image sequences

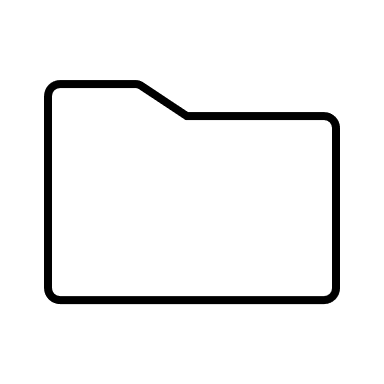
-computing power



**Shared**

-code share

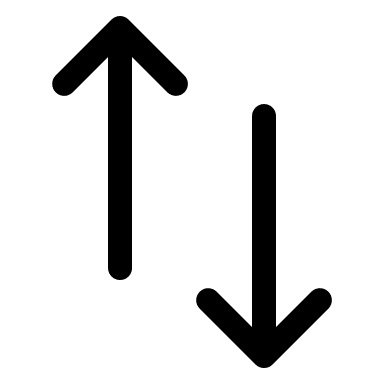
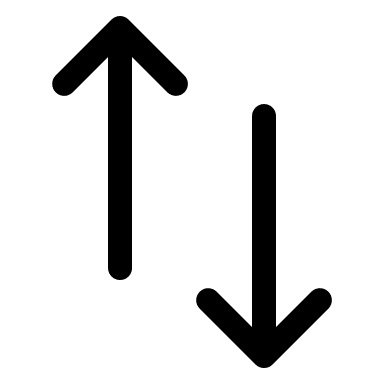
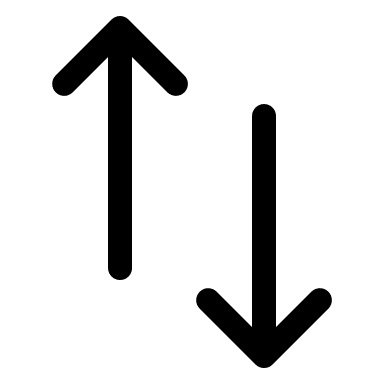
-IO files access



**Local**

-the IDE

-the display



The server is where we have all available computing power (used for training the model and for predicting) and all the image sequences (all the data used).

On the local environment (the laptop) we are going to do all the coding and the visualization (for labelling and also results analysis).

To be able to do all the coding in an IDE (PyCharm) we had to create a shared space between the server and the local system. This space was used to upload the code automatically and run it from the IDE, but also to make all the transfer of files between the two systems easier.

## Methodology

The first step was obviously a very rigorous research, but right after that we started thinking about all the approaches possible with the intention of solving this problem in the fastest and most accurate way. We made a list with all of them by taking into consideration a multitude of factors: the availability of images, the manual labels on image sequences already existing, certain restrictions, available technologies and libraries for creating the neural networks.

After testing the environment and making sure everything is in place we tried training models with real data. The main part of the work will come right after gathering data, preparing the environment, and testing the different models, when we actually get to compare the results. Afterwards we selected the best model architecture for our problem (based on theoretical facts and empirical arguments) with which we continued experimenting by varying parameter setups and using different data sets.

After establishing a couple of definitive working networks, there was now only room for little improvements and adjustments.

## Proposed approaches

The first and most obvious approach would be the Convolutional Neural Network with binary classification (tunnel or no tunnel), or an alternative CNN with multi-classification (no tunnel, entrance of a tunnel, inside a tunnel and exit from the tunnel).

CLASSIFICATION

TUNNEL

BRIDGE

2.BINARY

(tunnel/no tunnel)

3.BINARY

(bridge/no bridge)

1.MULTIPLE

(start/inside/exit/none)

TUNNEL & BRIDGE

4.MULTIPLE

(on top/under/none)

5.TERNARY

(tunnel/bridge/none)

6.MULTIPLE COMBINED

(start / inside/ exit of tunnel)

(bridge/none)

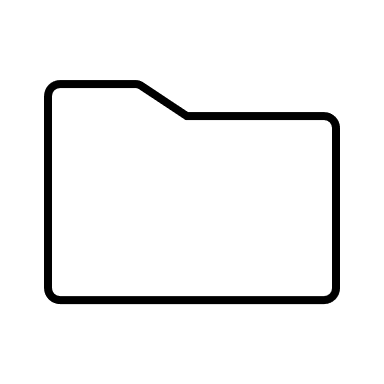
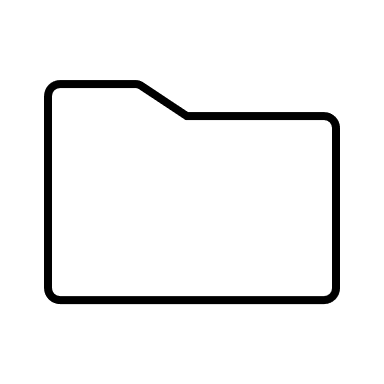
Figure ‑ – possible classification approaches

An entire new branch could be constructed around the fact that there are strong similarities between tunnels and bridges (especially wide ones). Here we could encounter a lot of false positives, and most certainly we should acknowledge this and try to include bridges images into the classification.

Among the variations considered in Figure 4-2 we also have “bridges multi classification” in two instances, but it seemed highly unlikely to perform well due to the lack of features that could be discovered and the fact that in our sequences it turned out to be complicated to classify bridges even for humans (in particular images with the car on top of a bridge). Otherwise the usage of bridges (in a binary manner: bridge/no bridge) will turn out to be extremely useful in reducing the number false positives of the tunnel recognition network.

More important variations that are considered after having a working network are the usage of active learning or ensemble learning. They would most likely improve the performance of the model.

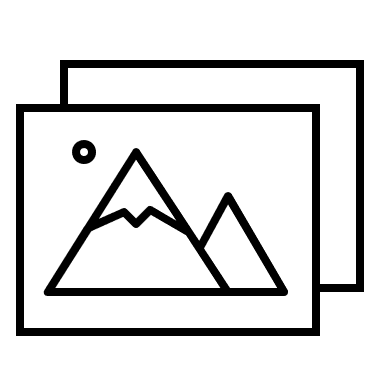
## Data



**Server**

-extract images

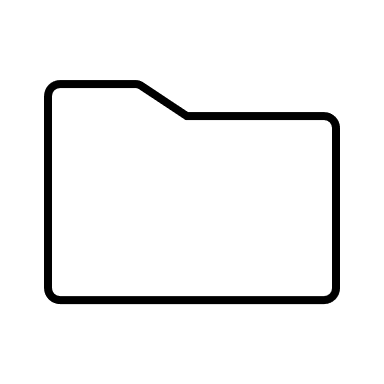
-transfer images



**Local**

-convert paths

-label images



**Server**

-extract by class

-construct database

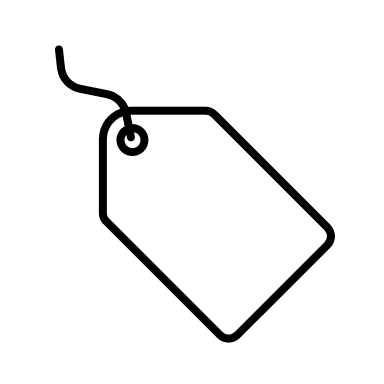
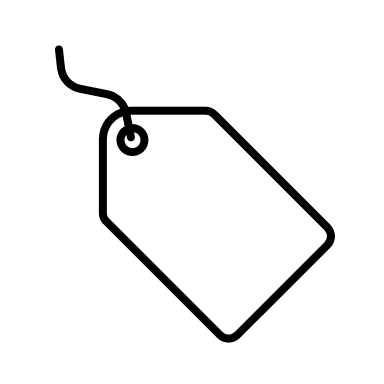
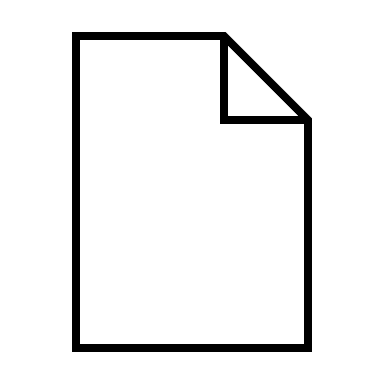
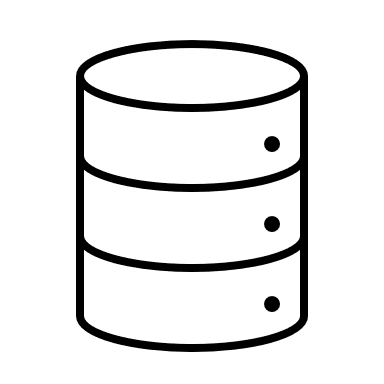


Figure ‑ – The steps of our data

### Complete flow

Originally the data is stored on the server as sequences of images and individual images needed to be extracted from them. We exported every third frame in a grayscale format. The images were copied on the local memory for visualization and the labelling process.

After converting the image paths and labelling all the images a file with all the labels had to be sent back on the server where all the neural networks would have faster access to it.

The output file contained images ordered by time and with a manually added label, they were transformed into 3 individual files with data ready to be used for the training, validation and testing of the model.

With this database of image paths prepared (not the images themselves), the model had to be explained how to collect the images and from where (by overriding some functions of the Data Generator). Afterwards the images would be preprocessed all in the same manner and they would serve as input for the convolutional neural network.

### Collecting the Data

As we previously mentioned we use images extracted from sequences, but first we had to select the relevant sequences. The main filter used in selecting the sequences was the presence of a tunnel in it and the compatibility with our system. Although there was a lot of other information in an image we only needed a grayscale image (one layered image). Out of the couple of thousands of sequences we extracted about 500 sequences with tunnels in it, and another 500 sequence containing bridges. That sums up to about 140’000 images from tunnel sequences and over 300’000 images from bridges sequences.

### Extracting the information

After the images were present locally, we had to manually label each image and of course that due to the time constraints we were not able to label all of them. During several labelling sessions (about 3 different batches) we summed up to 50’000 manually labelled images.

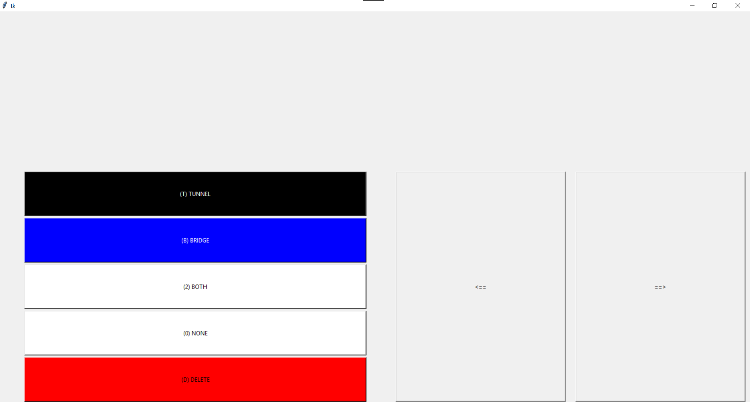


Figure ‑ – labelling interface

We used a simple labelling interface (as can be seen in Figure 4-4), which given a path with all the images (stored locally) will print them one by one waiting for an input from the user (with the label). The labelling interface has multiple commands and functionalities:

* Label current image (and load the next image afterward) using hot-keys
* Skip an image (this will leave the image with no label and will not be exported)
* Automatic save and export of the labelled data on exit
* Keeps track of labelling progress even on the restart of the interface

The available labels were: TUNNEL/BRIDGE/NONE/BOTH/DELETE. To improve the labelling process ‘None’ and ‘Both’ were added to be able to use all labels for binary and ternary classification. A variation of this interface has been implemented for the tunnel multi-classification labelling containing the following labels: NONE/ENTRY/INSIDE/EXIT.

From those images we had to extract the information using the CNN and for this to properly happen (and also to be able to validate it accurately) we had to rigorously create the 3 files with data (training data, validation data and testing data). In order to achieve a correct validation and testing of the model, the images must not be present in two different files, and moreover no sequence of images should be split between two different files. We do this to avoid a common evaluation error that happens when we test the model on data too similar to the data it has also been trained on. Even though the images are not exactly the same, the similarity between one image and another in the same sequence (3 or more frames apart) could be considerably higher than two images in different sequences.

After separating the images into 3 different files we had to consider the unbalance of the classes. We could use weights for each class whilst training the model or we could simply remove some images. Although we only used sequences where tunnels appeared, there were plenty of negative frames (before the tunnel showed and after exiting the tunnel, same for bridges), in fact we ended up with more of the latter (~45% tunnel images and ~55% no tunnel images, ~10% bridge images and ~90% no bridge images). We will remove negative images until we achieve a balanced dataset.

Another important matter to consider is the order in which the images are given to the model, because it is known that if a neural network learns all the positive entries and then all the negative ones it will have bigger variance and will probably overfit. To avoid this, we will shuffle each dataset file individually.

### Image Preprocessing

After we collected our data and grouped it in three distinct files, one for each stage of the model (training, validation, testing) we considered the images themselves and how we can help the network to better find the right features in the image. We use filters/effects on images before giving them to the network, and this stage is called image preprocessing. We have experimented with multiple filters, but on the architecture that we applied the best results came in when we used no effects. This most probably happened because the models were already trained on very large datasets that consisted of a very wide range of images (preprocessing could not be used so easily). We experimented with random filters (not randomly chosen, but random ratios/values/intensities): brightness, contrast, resizing, rotations, and even Lagrange filter, but most of them ended up not contributing much in optimizing the model, because all tunnels usually look the same, they don’t rotate and they come in different sizes by default (as the car approaches the entry).

In the end we used cropping to remove some of the imperfections caused by processing and visual defects on the sides of the images caused by turning the image into a flat wide format. All the image had to be cropped to the exact same size since the neural network must know beforehand the expected format.



Figure ‑ – visualization of the cropped area

We applied a normalization operation on every image using the maximum pixel value, mean value and standard deviation to limit the variation in input (especially helpful for very sunny days where glare appears or at night).

## Application

### Environment

Our system is basically split between the local device and the server, so there are two individual setups. Whilst we cannot go into too much detail, the server contains an Anaconda environment (conda 4.5.13) with python 3.6.8, cuda 10.1.105 and cudnn 10.1\_v7.6. Locally we are using as IDE the PyCharm 2019.3.5 that can connect to a shared location and directly run scripts on the server using a SSH connection.

### Frameworks and libraries

We are using plenty of libraries and packages for python in order to optimize the process as much as possible, the most important of them are: albumentation 0.2.2 (for image filters) [33], cudatoolkit 9.2 [34], cudnn 7.6.0 [34], keras 2.2.4 (used for creating the neural networks) [35], opencv 3.4.2 (used for CRUD operation with the images) [36], matplotlib 3.1.0 (used for some of the graphical results) [37], numpy 1.15.2 (used for work with matrices and mathematics) [38], pandas 0.23.4 (used for certain data manipulation and analysis) [39], pillow 5.3.0 (used for image processing) [40], scikit-learn 0.20 (used for a couple of statistical interpretations of data) [41], tensorflow 1.13.1 (the backend used by keras, used in overriding a couple of keras methods) [42], paramiko 2.7.1 (used to open and maintain a SSH connection and communication means between the local machine and the server) [43]. Locally we also used pysimplegui 4.19.0 (for the user interface) [44] and tkinter 0.1.0 (for the labelling gui) [45].

### User Interface

Although the user interface was not a priority, it was used for two purposes: the labelling process and the visualization process. The first one has been explained (in chapter 4.4 Data – Extracting the information) and the designated user is only the developer (author of the paper) with the sole purpose of preparing the data. The latter could be used by anyone with access to the server because it communicates through SSH during the predicting functionality.

As we can see in the Figure 4.6 there are a lot of functionalities available. Although it began only as a simple way of visualizing the results, it slowly turned into a feauture-rich desktop application that can do everything one might need after training the model. Some of the key features are:



Figure ‑– the user interface

* Select a model (select a classifier category and then a particular model already trained)
* View details about a model (compare the models by analyzing many particularities)
* Select an image database (choose a local path of a .csv file)
* View details about the database (a wide range of information regarding data exploration)
* Visualize properties of the database (analyze the graphical plots generated using some of the details about the database)
* Predict results (using a model and a database return all the predictions of the images)
* Analyze results (statistics of the results, confusion matrix and other metrics)
* Visualize the results (generate an animation with all the given frames and the prediction)
* Select a result file (choose an already existing result file generated beforehand and load)



Figure ‑ -active example changes

* Set the plot location (choose between plotting the statistics in a new window or in the same window)
* Apply active learning (with a given model and 3 datasets – for first training, unlabeled predictions and validation) for one epoch or more

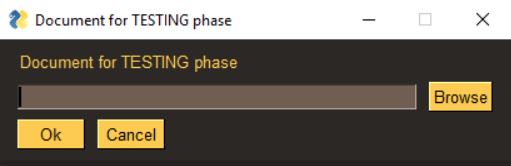


Figure ‑ – active learning querying

We will discuss the functionalities, but only shallowly since they are not the most important part of the study. Perhaps the best manner in which one could comprehend the intended usage of the interface is through a use case that would cover all the main functionalities.

The user selects the desired model from the list of pre-trained neural networks on the main database (they are divided by the classification strategy) and now the system will display details about the selected model. The user will select an image database path and after uploading the file, the system will display all the details about the uploaded images. The user will clicks “Predict” button and the system will send a message to the server about the task. While the results are loading the user will visualize data about the country distribution of the data (in a new window with a generated chart). The system prints out a message that the predictions are done and saved in a certain file. The system displays details about the results of the predictions automatically (confusion matrix, metrics, and visualizing methods). The user clicks “Start visualization” and the system opens an animation with all the frames and the probabilities predicted for each frame. The user loads a different result file and the system will load all the details and metrics (also the visualization methods) for the given results.

The interface is largely separated into 3 components: the classifier-model, the image database, the prediction. The first component uses an already known list of all the models that are currently trained and upon selection it simply populates all the details fields. The second component receives from the user an input file path, it reads it completely and then it calculates a couple of information and generate the necessary data for all the graphic plots available by category/property. It uses not only the given path, but also a complete list of all the sequences ever extracted that contains essential details. The last component can be populated by two means; either predict new results, this requires a model and a database already selected, or load a previously generated result file. Both methods will load at the end details about the results’ details: confusion matrix, metrics of the evaluation (which will make sense only on labelled data), and it will unlock the visualization methods.

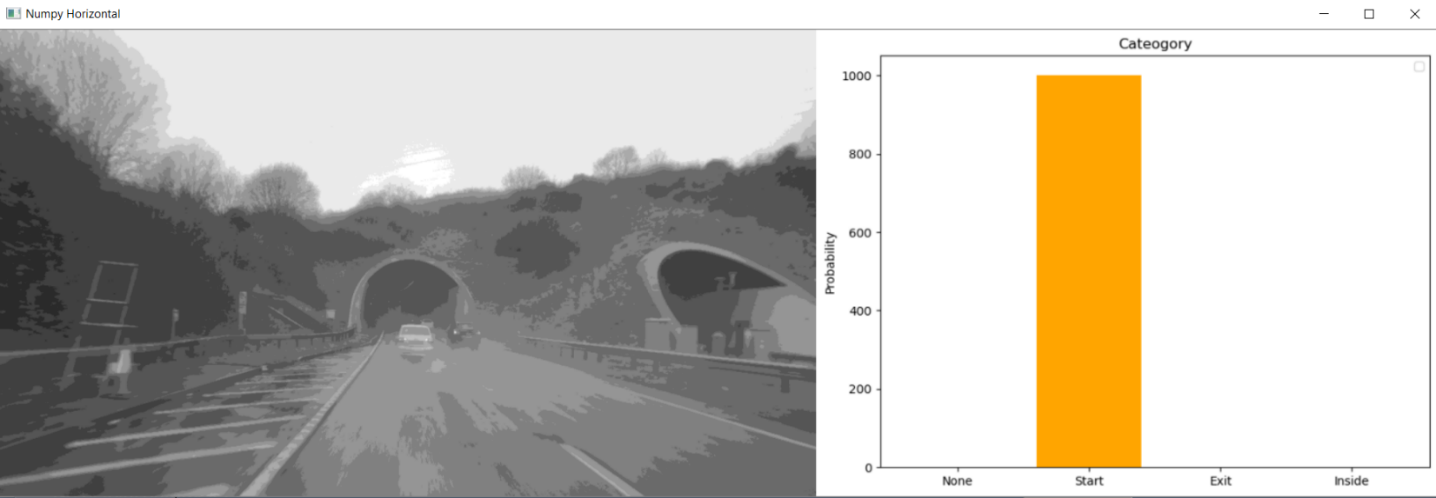


Figure ‑ – the image-probability animation visualization

The active learning functionality will be activated by confirming a model stored in the category “Active learning”. The user can then select the initial database and press the button “Begin training”. The user will need to provide again input: the path to the unlabeled database (used as augmentation), and the path to the validation database (used for comparing models). The server will be contacted through an SSH connection and it will respond by sending results files and statistics that are going to be displayed in real time on the interface.

### Architecture

As it can be seen so easily from Figure 4.1 and Figure 4.3 and as it was thoroughly explained in the corresponding subchapters, the main characteristic of our application’s architecture is the fact that it is distributed. The server deals with storage and computations, and the local environment is used for visualization and programming. The work has been separated in three big projects, each of them responsible for a specific phase.

First, we have the preprocessing project that contains a series of scripts that were run every time we added new data to the project (this project runs on the local environment only). After having available the complete list of 140’000 images (more exactly server paths) that are tunnel related we will have to go through the following steps:

* Copy the image files on the local memory
* Get a list file with all the image paths converted to local addresses
* Run the labelling interface with that file as input, label all images or only a part and close the window
* Generate from the resulted file with paths and labels a file for every label in another location with converted paths to server address
* Using the resulted files created 3 files (train, validate and test data) with independent sequences and using 60/30/10 percentage (60% train data, 30% validate data, 10% test data)

Now that we have successfully created the database, the second project can be used to create the models. The database must be transferred to the server because this project runs exclusively remote (all the code has been written in a shared location). We are using transfer learning to import the first layers that detect the most generalized features, and then add a couple of dense fully connected layers that will make sense of the features and classify them correctly. There is only one algorithm here that can be run with different parameters (the arguments for the compilation of the neural network, the arguments for the data generator, the arguments for the evaluation of the network), and it is composed of a couple of main components:

* Data generator (defines the preprocessing operations, how the data is read and processed in batches) – overridden class
* Model construction (initialization of the model, loading the weights, adding the new layers, fitting the data, saving the model)
* Model testing (it predicts the test data and then generates metrics for comparison)

The third and final project is the user interface and it runs on the local machine, but also accesses the server remotely using the SSH protocol. The project is client-server between the GUI run locally and the server that can run the prediction algorithm. The local part of the project is structured simply by the Module-View-Controller architecture, but has two controllers: one runs the operations that do not need serious computations locally, and the other one implements a communication interface with the server in order to run the CNN.

The prediction algorithm will work on another thread and this will allow the user to access the rest of the components at the same time. It will open an SSH connection with the server and begin sending the commands needed in order to predict the results:

* + module load conda/4.5.13 cuda/10.1.105 cudnn/10.1\_v7.6 – it will load the packages
  + *source activate condaEnvironment* – it will open the environment (already created)
  + taskset –cpu-list 12 python predict.py -model\_name VGG16.2 –frame\_folder data/testData4.csv –result\_file /results/resultFile4.csv – it will start a task on a specified CPU with the command “python...” (the GPU is also specified, but in the python code), command that will run a given script with the given parameters

This last command is rather interesting because as simple as it is, it actually contains the final product of the entire project. What it will actually do is use the given model (it will load the weights and configure it with the already tested parameters that work best) and dataset (which will be preprocessed as presented in Chapter 4.4 Data -Image preprocessing) to generate the best prediction and store them in the given location.

A training algorithm has been designed in a very similar manner to the one previously explained. Both those algorithms have been used in creating a new class that works as an interface for the computational operations. With a beautiful abstraction we managed to reduce the interface to a few rudimentary methods: train, predict, validate, splitByThreshold, augmentDataBase. All these methods facilitate very easy implementation of the Active Learning and Ensemble Learning strategies.

## Models comparison

With the whole environment complete, we can begin testing and experimenting with all kinds of model architectures and datasets. We can then properly compare the models and based on metrics and statistics choose one that works best and use it in the following iterations.

All the used models have been constructed following the steps presented in the previous Chapter 4.5 Application and are based on transfer learning using Keras. The first layers that are imported are not altered and we import weights pretrained, but the second part of the network will consist of manually added layers that are trained using our data.

### Models architecture

In order to find the architecture that best fits our data we will definitely need to experiment with multiple architectures. For a wide and accessible variety, we are going to choose the following public and efficient pretrained architectures: InceptionV3 (created by Google with 42 layers and over 23 million parameters, factorizing convolution, introduced batch normalization, grid size smoothing and regularizations ) [46] ,VGG16 (build on top of AlexNet with 16 layers and over 100 million parameters uses convolutional layers with ReLu, max pooling and fully connected layer with ReLu) [47] , ResNet50 (created by Microsoft it introduced the concept of skipping connections or residual information, with 50 layers and about 23 million parameters) [48], DenseNet121(contains dense blocks and pooling operations, it seems as an extension of ResNet with iterative sums with 121 layers and about 7 million parameters) [49]. Although all four models are very interesting and have many particularities we are not going to go into much detail.

### Training of the models

The general order of the operations made to train the models is as follows:

* Initialize all the parameters (some of them are calculated: mean, standard deviation)
* Create the classifier, keep the first ‘x’ layers (one of the parameters), add the last dense layers and an activation layer to output the predicted class, and compile it
* Using some of the parameters create the Data Generator, then initialize a couple of callbacks (Early Stopping, checkpoint saving)
* Fit the model on the data and save it after completion

The general order of testing the models is relatively similar: initialize parameters, load the model, create the data generator and the predict generator, predict all the data and using the results calculate metrics, confusion metrics, and export.

### The best model

After experimenting with every model on different parameters setups (custom chosen depending on the architecture of each one) we have come up with the following results:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| MODEL | Data Size (train/valid) | Param. Trainable | Param. Total | Train Time (avg epoch) | Epochs | Train Acc | Valid Acc |
| InceptionV3 | 10600/5300 | 262’530 | 22’065’314 | 940 sec | 10 | 93% | 85% |
| ResNet50 | 10600/5300 | 820’310 | 23’408’022 | 990 sec | 5 | 95% | 49% |
| DenseNet121 | 10600/5300 | 133’506 | 7’168’962 | 940 sec | 10 | 95% | 92% |
| VGG16 | 10600/5300 | 2’425’730 | 14’780’610 | 995 sec | 10 | 99.55% | 99.13% |

Figure ‑ - stats for the training phase

Right after the training phase we could see that ResNet50 did not complete its 10 epochs, but instead stopped early due to lack of improvement. A little overfit appeared for every model because of the reduced size of the image database relative to the number of parameters. Even though this happened we can safely approximate what model would fit best on our data, but before any conclusion let us test every model on a new dataset composed of about 1400 images from a couple of different sequences:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MODEL | Test Size (p/n) | Class Distr. (p/n) | Positive  Precision | Negative Precision | Pos. F1-sc | Neg. F1-sc | Weighted Average Precision | False Pos. | False Neg. | |
| InceptionV3 | 717/771 | 48/52 | 93% | 75% | 77% | 84% | 84% | 37 | 244 |
| ResNet50 | - | - | - | - | - | - | - | - | - |
| DenseNet121 | 717/771 | 48/52 | 97% | 88% | 91% | 92% | 92% | 19 | 105 |
| VGG16 | 717/771 | 48/52 | 100% | 99% | 100% | 100% | 100% | 1 | 4 |

Figure ‑ – stats for the performance on the first dataset

The conclusion is rather obvious, that VGG16 outperformed all other three architectures for our problem and the given dataset, but the same comparison was also done on the multi-classification for tunnels problem with a bigger dataset (over 40k) and the results turned out similar ( VGG16 – 91% , ResNet – 69%, DenseNet 81%).We will attempt further improvement on the next strategies and techniques using the VGG16 architecture.

## CNN Scope Strategies

At this point we have trained a working model on one of the strategies, but with the sole purpose of choosing one architecture only for usage in the other strategies due to time constraints and constraints on the availability of the computation cluster. It does not worth the effort to attempt every strategy with each of the models since we already know which model can learn our database images best.

In the following subchapters we are going to shortly explain each strategy and if we attempted an implementation we will explain how it performed and under what circumstances, otherwise we will justify why we decided not to implement a model.

### Supervised tunnel binary classification

This model takes in data labelled as follows: 1 if a tunnel is in the image, 0 otherwise. It seeks to identify tunnel entrances, tunnel insides, and tunnel exits under one positive label. This does not simplify the learning process because if the models learn the same features, then theoretically the only differences between this and multi-classification of tunnels should be in the last fully connected layer.

We began the augmentation of the original dataset (from Subchapter 4.6 Models comparison - The best model), that initially had about 15’000 images, to almost 40’000 images with the same file ratio (60-30-10; train-validate-test). Also, we separated some particularly different sequence for the testing set only and obtained satisfying results (VGG16.2 in Figure 4-12 and Figure 4-11).

Although there are clear signs of overfitting on the training data it is rather interesting that on the testing data (particularly selected to be a little harder to recognize) it performed with over 5 percent better than in the validation stage (an average of 94.5).

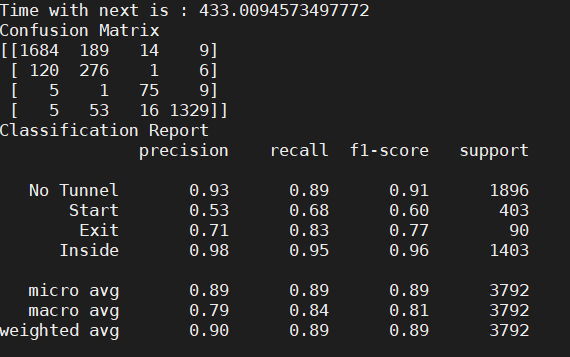
Another significant improvement has been made based on the fact that most of the false positives were happening under three conditions: extreme lighting (a stroboscopic effect or glare that can be extremely difficult to ignore or learn), extreme darkness (the lights on the side of the road would sometimes appear as if they are lights on the ceiling of the tunnel, but this can be challenging to identify even for a human), and the presence (and quantity) of images containing bridges in the datasets (due to their similarity). We added 2500 images that are known to contain bridges only to the training and validation data set with the intention of correcting the false positives aforementioned. This is VGG16.4 (found also 1 in Figure 4-12 and Figure 4-11) and even though we expected a decrease in the number of false positives, the exact opposite happened and the number of false positive increased. On a closer look (using the visualization methods explained on Chapter 4.5 Application - User Interface) we observed something interesting: overall, the network predicted slightly better, but relatively light oscillation in the probability caused no statistical improvement. This oscillation can be eradicated using a simple averaging algorithm.

### Supervised tunnel multiclass classification

This model uses 4 distinct labels: 0 for no tunnel, 1 for tunnel entrance, 2 for inside the tunnel, 3 for tunnel exit. Although it should not be very different during training, it can possibly learn to recognize the tunnel better because it is penalized differently on every part of the tunnel, and since the previous model had issues especially with the entrance of the tunnel and inside the tunnel we should be seeing a slight improvement.

For this strategy to be implemented a whole new dataset had to be constructed, but we will optimize the labelling process by only labelling the positive images as one of the three classes: tunnel entrance, inside the tunnel and tunnel exit. We created the same database of about 40’000 images labelled accordingly and trained VGG16.1. Although we added numbers to Figure 4-11 and Figure 4-12, they do not really emphasize the improvement. In Figure 4-10 we have a better view of what really changes, and by visualization (on this test data and also on other unlabelled data) we can gain an even better understanding. Although statistically only the false positives number are getting better (smaller) this actually gives us a really powerful inside view. We can conclude even only using this limited testing set and visualization that actually the biggest issue is not with the model, but with the labelling data process. We are going to discuss this thoroughly later (in subchapter 5.2 Limitations). The predictions are rather relative and hard to statistically verify them, but visualizing the data we can obviously quantify the accuracy of the model.

Figure ‑ – detailed statistics on the new test dataset



### Untested strategies

We did not attempt to implement the following possible strategies: supervised bridge binary classification, supervised bridge-multiclass classification, supervised bridge-tunnel multiclass classification, supervised bridge-tunnel ternary classification, because all of them require a significant augmentation of the data base (labelling bridges). Moreover multiple bridge classification would most likely not work as it has been previously discussed in subchapter 4.3. Proposed approaches.

### Methods results

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MODEL | Test Size (p/n) | Class Distr. (p/n) | Positive  Precision | Negative Precision | Pos. F1-sc | Neg. F1-sc | Weighted Average Precision | False Pos. | False Neg. | |
| VGG16.2 | 1898/1894 | 50/50 | 92% | 98% | 95% | 94% | 95% | 170 | 38 |
| VGG16.4 | 1898/1894 | 50/50 | 89% | 98% | 93% | 93% | 93% | 230 | 36 |
| VGG16.4 | 1896/403/90/1403 | 50/10/ 3/37 | 87% | 93% | 88% | 91% | 90% | 200 | 228 |

Figure ‑ - stats for the performance on the updated test dataset

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| MODEL | Data Size (train/valid) | Param. Trainable | Param. Total | Train Time (avg epoch) | Epochs | Train Acc | Valid Acc |
| VGG16.2 | 23000/11500 | 2’425’730 | 14’780’610 | 2570 sec | 10 | 99.7% | 89.12% |
| VGG16.4 | 24500/12000 | 2’425’730 | 14’780’610 | 2650 sec | 10 | 99% | 93% |
| VGG16.1 | 23000/11500 | 2’425’730 | 14’780’610 | 2300 sec | 10 | 99.3% | 91.9% |

Figure ‑ - stats for the training phase

In the first table (Figure 4-11) we have metrics from the testing phase, in the second table (Figure 4-12) we have information about the training process, validation process, and general details about the model. It should be considered that for VGG16.1 the database was the same but partitioned differently because the sequences are randomly split between the final files (training, validation, and testing database).

Another important factor is computational time and our models manage to predict an image in 0.1 seconds on average (VGG16.2: 0.095 seconds, VGG16.1: 0.11 seconds, VGG16.4:0.098 seconds). The predicting algorithm has been tested on a GeForce GTX 1080 Ti.

### Active learning strategy

The implementation was complicated due to the restrictions and limitations posed by the SSH connectivity to the server, but by creating a simplified interface we integrated all the necessary methods. We automated the process of active learning, we initialized a class with a model name and three input files (the first training database, the unlabeled database for the augmentation and a validation database for comparing). The model is trained on the first database and then it makes predictions on the second one. The predictions with a high certainty will be added directly to the training set, and the other will be labelled by a user and then added to the training set. The model will be retrained on the extended data set and then the process can repeat if another unlabeled database is given as input. After each training phase the model will be tested on the validation data set. Since all the models will be tested on the same data and for all of them the data is new, we can guarantee the fairness of the results.

The models could not train for too much time because there were time limitations to the SSH connection and this complicated the process, but we managed to completely train 2 versions. The first version went through 5 iterations using datasets with an average size of 100 images augmented per iteration and a validation dataset of 456 images. That means the last iterations trained on a little over 600 images. The second version trained for 6 iterations with over 900 images on average per iterations (a total of about 6000 images) and was tested on a validation dataset of 5746.

As we can see in Figure 4-13 the results are quite positive for such a little dataset. Using the same validation data set (only reduced for version 1) as the original algorithm: supervised tunnel binary classification, we see that we obtain very similar results and with a very different setup and limited resources. Every individual model trains the same as the original algorithm only it takes far less data (600/40’000, respectively 6000/40’000), trains for way less time (5 and 15 minutes compared to one hour each epoch) and for fewer epochs (maximum 5 compared to 10). The most important fact is that they needed way less labelling and human intervention (they required 225 labelled images out of 500 total images, respectively 1150/6000). We have some huge advantages here, very little effort for pretty good accuracies, but the models could have been trained for more time and with more data. This model might be lacking the robustness, consistency of the original one and it certainly has bigger variations between iteration, but with more resources it will most likely surpass its predecessor.

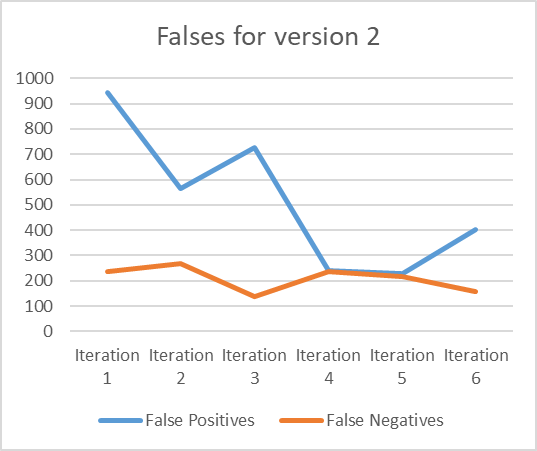
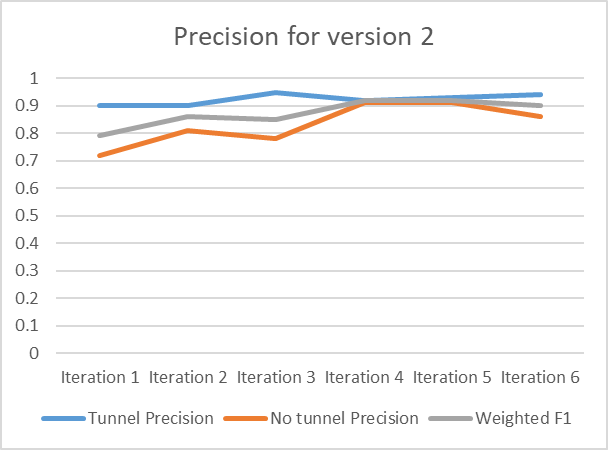
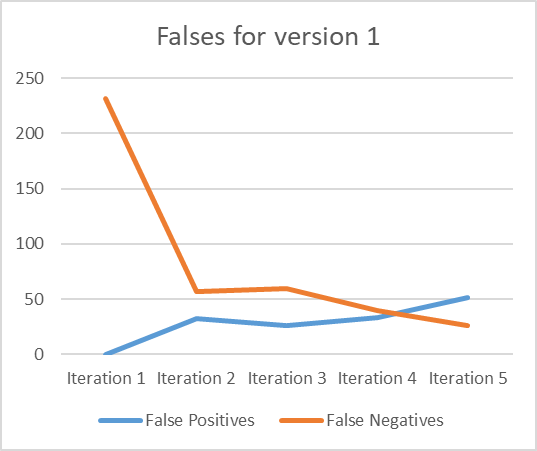
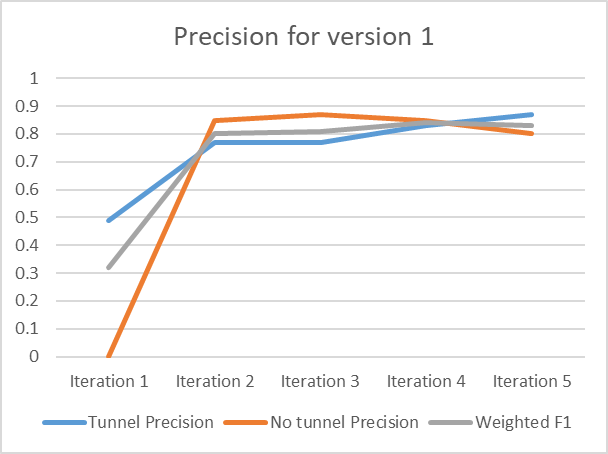


Figure ‑ – active learning performance visualization

### Comparison of chosen models

We previously commented on our results in each subchapter for every model, but now we will make a short summary and compare the methods. We made continuous improvements with each new strategy implemented, even if we cannot properly quantify the improvement with anything other than visualization techniques. It must be considered that the ultimate task in this problem is to recognize tunnels on the street with as great an accuracy and as few false positives as possible. The false positives remained the same, but the situation in which they appeared changed. Instead of predicting false positives while going under a bridge or during the night, they were predicted more in the time interval right before a tunnel entrance or exit was labelled. This is now a labelling issue and our model should recognize most tunnels (from a sequence instead an individual image) out there and nothing more.

It has been observed that the original binary tunnel classification model although having a great accuracy, it lacked versatility. It had certain cases in which it consistently predicted badly (nighttime with bad illumination, daytime with direct sunlight), but it managed to recognize every individual tunnel sooner or later. The two improvements implemented for this method seemed to be solving the problem, although at the cost of accuracy (which we should not forget that is very strongly related to the human labelling standards).

With the second classification it was proved that using multi-class for each of the tunnel section the areas in which the model failed (at least statistically) were the entrance and exit. That confirmed our concerns that human labelling was the problem, and also confirmed our hopes that the model was in did predicting very well overall, but not exactly at the same pace with the human.

The last method was a strategy implemented on the untrained model from supervised binary tunnel classification using active learning. This attempt proved to improve immensely the information usage from the given data, the model evolved at a much quicker rate and with very little resources comparative to its predecessor. Moreover, it seemed that training in only a fraction of the original time it came close to the accuracy of the original model.

# Conclusion and Extensibility

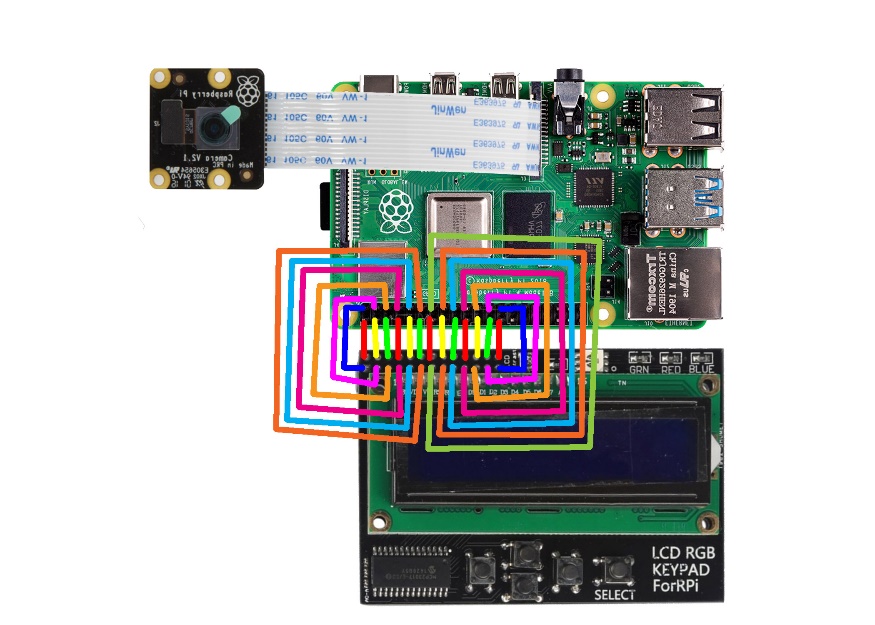


Figure ‑ – Raspberry Pi experiment project schematics

## Experiments & usability

We intended to create a tunnel recognition device using a Raspberry Pi [50], Python, a camera module and a display module to show the live results. Unfortunately, we were not able to properly run the keras algorithm on the Raspberry Pi OS (formerly Raspbian) operating system because there are not yet compatible (at least for this particular device) and it would have been almost impossible to make it work on the GPU. Instead we managed to run it on the CPU (using tensorflow lite) [51].

The system is relatively simple: firstly, the camera takes photos on a time interval, then the CNN analyzes the newly added images and generates a prediction, afterwards the LCD display prints out the label and the probability. The speed at which it runs on the processor is considerably slower (about 5-10 seconds per frame), while on a GPU it could predict one image in 0.1 seconds.



Figure ‑ - real-time image of a prediction

Another experiment with an interesting usability is the user interface itself (Subchapter 4.5 Application – User interface), that makes testing a model a lot easier and enables all sorts of metrics and visualizations in order to best understand the performance of a model. It could be extended easily to also train the models on given data sets.

## Limitations

Even though there are a great number of improvements compared to any other published prediction technology of this kind, there are of course a couple of limitations. The two main considerable limitations are related to the manner in which the labelling is done, and related to time constraints.

Let us explain in more detail the problem related to labelling. In the begining we needed to decide how to label the data, because the fact that all the images were from the perspective of a vehicle on the road can make the situation quite complicated. If we consider a highway that heads towards a tunnel and it is straight for a few hundred meters before the entrance, then that means the tunnel will teoretically always be in the image, bigger at times and smaller at other times. This situation forces us to set up some standards that need to be respected when labelling, standards regarding the distance until a tunnel is considered visible. Since this is entirely contextual we decided to label as positive any image that contains a clear tunnel, aproximately centered, and that covers a certain portion of the image (width of tunnel is at least 10% to 15% out of the total width of the image). The same concept is applied for the exit from a tunnel. This standard is quite relative and open to considerable human errors. That does not mean the model will not learn, because our model predicts almost perfectly the inside of the tunnel and the only problems appear with “Start” and “Exit” labells. The only disadvantage is the fluctuation/oscillation that appears when going from one label to another (right at that time when the entrance/exit is going from 10% of the image to 20%), due to the fact that it didn’t learn a very rigorous boundary between the labels.

The second problem is quite simple and is related to the time needed to predict one image. On average the algorithm predicts the class of an image in about 0.095 seconds, that means it can go up to 10 frames per second. It must be considered that this computation took place on a very powerful GPU and it could not run efficiently in real-time on a car unless the car has special feature-specific equipment. It is rather obvious that if we want to achieve greater precision and accuracy we must sacrifice computation time, so this might not be the best solution for any system.

## Future work

The project can be extended in multiple ways depending on further interests. The first and most rudimentary upgrade would be the augmentation of the database since there is already data available. For this we would only need to label more data, and then train the model on the new database.

The second manner in which to extend the project is also strongly related to the neural network. We could implement different strategies of active learning or even ensemble learning in order to better recognize the particularities of the tunnels. Using ensemble learning will most definitely work considering that we already know our networks learned different particularities about tunnels, but did not learn to associate them with the classes perfectly.

A third way in which we could extend our project is using the user interface. We could implement more functionalities in the GUI to allow the user to also train networks with certain parameters and visualize in real time the networks and their evolution.

Another way of better predicting the entrance and exit from the tunnel can rely on supplementary information for each image related to the distance between the vehicle and the obstacle ahead (the tunnel). Using a simple regression algorithm, we could rectify the oscillations that happen between labels.

Scalability is also possible considering the intelligent agent actually is contained in a simple script that can run on a single GPU. The software could be externalized using a client-server approach based on the HTTP protocols and could easily become accessible to any online vehicle.

## Comparison to State of the art

The comparison to the other identified works is not particularly difficult. The quality of the predictions is obviously much better (as the metrics and statistics show), but also the reliability of the results is way greater (as the database is way bigger and has more variation without any limitation given by daytime, location or environmental conditions). Although our model gives much better results, some of the other works out there run considerably faster. The trade-off between precision and speed was expected, but it is not unacceptable considering that our model can still run at a speed of several frames per second.

Comparing the active learning strategy implemented with the state of the art the gap is not that big anymore. There is far less data needed and training takes place much faster, but the accuracy does not change significantly. Although improvement in prediction time per frame is not increased, the overall time needed to get the model trained is greatly reduced.

# Acknowledgements

The research regarding “Chapter 3: Literature overview” has been done by the author alone. It only acknowledges work posted online publicly under any form and before the date of 04 – April – 2020. It cannot be guaranteed that other similar work (in progress or under certain confidentiality restriction) don’t exist, furthermore I suspect their existence due to the highly competitive industry of autonomous driving.

I must acknowledge the contribution and assistance of my supporting mentors, that provided guidance, resources throughout the whole process.

# Bibliography

|  |  |
| --- | --- |
| [1] | B. E.T., Men of Mathematics, Simon & Schuster, 1986. |
| [2] | G. Vince, "Homni: The new superorganism taking over Earth," *BBC Future,* 2014. |
| [3] | M. Gordon, "IEEE Technical Digest 1975," *Inter Coporation,* 1975. |
| [4] | P. N. a. S. J. Russell, Artificial Intelligence: A Modern Approach. Third Edition, Prentice Hall, 2009. |
| [5] | G. V. M. Eduardo A.B. da Silva, "Digital Image Processing," *The Electrical Engineering Handbook,* 2005. |
| [6] | A. Rhodius, Argonautica, Ancient Greece, 3rd century BC. |
| [7] | M. Kressel, 36 Days of Judaic Myth: Day 24, The Golem of Prague, 2015. |
| [8] | J. W. v. Goethe, Faust II, 1829. |
| [9] | A. Kaplan and M. Haenlein, Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence, Business Horizon, 2019. |
| [10] | S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach (3rd ed.), Upper Saddle River, New Jersey, 2009. |
| [11] | A. Turing, Computing Machinery and Intelligence, 1950. |
| [12] | J. L. E. M. W. A. a. J. R. S. Katharine Graf Estes, Can Infants Map Meaning to Newly Segmented Words?, 2013. |
| [13] | P. IP, Conditional Reflexes, New York: Dover Publications, 1927. |
| [14] | D. Hebb, The Organization of Behavior, New York, 1949. |
| [15] | W. McCulloch and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics,* pp. 115-133, 1943. |
| [16] | J. von Neumann, "First Draft of a Report on the EDVAC," 1945. |
| [17] | F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review,* pp. 386-408, 1958. |
| [18] | D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *Nature. 323 ,* p. 533–536, 1986. |
| [19] | A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology,* pp. 500-544, 1952. |
| [20] | R. Bracewell, ""Heaviside's Unit Step Function, H(x)."," in *The Fourier Transform and Its Applications, 3rd ed.* , New York, McGraw-Hill, 2000, pp. 61-65. |
| [21] | R. O. Duda, P. E. Hart and D. G. Stork, Pattern Classification - Second edition, New York: WILEY-INTERSCIENCE, 2001. |
| [22] | K. Murphy, Machine Learning: A Probabilistic Perspective, London: The MIT Press, 2012. |
| [23] | T. T. R. F. J. Hastie, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, New York: Springer, 2009. |
| [24] | A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transactions on Neural Networks,* pp. 714-735, 1997. |
| [25] | S. Dupond, "A thorough review on the current advance of neural network structures," *Annual Reviews in Control,* pp. 200-230, 2019. |
| [26] | M. Valueva, N. Nagornov, P. Lyakhov, G. Valuev and N. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation," in *Mathematics and Computers in Simulation*, 2020, pp. 232-243. |
| [27] | B. M., B. A., B. G. and M. L., "Fast Vision-Based Road Tunnel Detection.," in *Image Analysis and Processing*, Berlin, Heidelberg, Springer, 2011. |
| [28] | N. Christoffer, Road Tunnel Entrance - master thesis, Oslo: University of Oslo, 2009. |
| [29] | C. C., S. H.C. and S. W., "Tunnel Entrance Recognition for video-based Driver," *Technical University Munich,* 2006. |
| [30] | T. Takeshi, Tunnel detecting device for vehicle and light control device for vehicle, USA: United States Patent Service, 2005-2006. |
| [31] | N. Ghasemloo, M. R. Mobasheri, A. M. Zare and M. M. Eftekhari., "Road and Tunnel Extraction from SPOT Satellite Images Using Neural Networks," *Journal of Geographic Information System,* p. 6, 2012-2013. |
| [32] | S. Lee, B.-h. Lee, J.-E. Lee, H. Sim and S.-C. Kim, "Iron Tunnel Recognition Using Statistical Characteristics of Received Signals in Automotive Radar Systems," in *2018 19th International Radar Symposium (IRS)*, Bonn, Germany, 2018. |
| [33] | A. P. E. K. V. I. I. a. A. K. A. Buslaev, "Albumentations: fast and flexible image augmentations," *ArXiv e-prints,* 2018. |
| [34] | P. a. F. F. H. NVIDIA and Vingelmann, *NVIDIA and Vingelmann, Péter and Fitzek, Frank H.P.,* https://developer.nvidia.com/cuda-toolkit, 2020. |
| [35] | F. Chollet, *keras,* {https://github.com/fchollet/keras, 2015. |
| [36] | G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools,* 2000. |
| [37] | J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in science & engineering,* vol. 9, pp. 90-95, 2007. |
| [38] | T. E. Oliphant, A guide to NumPy, Trelgol Publishing USA, 2006. |
| [39] | W. a. o. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, Austin, TX, 2010. |
| [40] | A. Clark, "Pillow (PIL Fork) Documentation," in *readthedocs*, https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf, 2015. |
| [41] | F. a. V. G. a. G. A. a. M. V. a. T. B. a. G. O. a. B. M. a. P. P. a. W. R. a. D. V. a. o. Pedregosa, "Scikit-learn: Machine learning in Python," *Journal of machine learning research,* vol. 12, 2011. |
| [42] | M. a. B. P. a. C. J. a. C. Z. a. D. A. a. D. J. a. D. M. a. G. S. a. I. G. a. I. M. a. o. Abadi, "Tensorflow: A system for large-scale machine learning," in *12th Symposium on Operating Systems Design and Implementation*, 2016. |
| [43] | Paramiko, "Paramiko.org," 2020. [Online]. Available: http://www.paramiko.org. |
| [44] | PySimpleGUI, "PySimpleGUI," [Online]. Available: https://pysimplegui.readthedocs.io/en/latest/. [Accessed 2020]. |
| [45] | F. Lundh, "An introduction to tkinter," *www.pythonware.com/library/tkinter/introduction/index.htm,* 1999. |
| [46] | V. V. S. I. J. S. Z. W. Christian Szegedy, "Rethinking the Inception Architecture for Computer Vision," Cornell University, 2015. |
| [47] | A. Z. Karen Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Cornell University, 2015. |
| [48] | X. Z. S. R. J. S. Kaiming He, "Deep Residual Learning for Image Recognition," Cornell University , 2015. |
| [49] | Z. L. L. v. d. M. K. Q. W. Gao Huang, "Densely Connected Convolutional Networks," Cornell University, 2018. |
| [50] | Raspberry Pi Foundation, "raspberrypi.org," June 2020. [Online]. Available: https://www.raspberrypi.org/documentation/. |
| [51] | A. A. P. B. E. B. C. C. C. G. S. C. A. D. D. M. D. S. G. I. G. .. Martín Abadi, "Large-Scale Machine Learning on Heterogeneous Systems," Software available from tensorflow.org., 2015. |
| [52] | S. Raschka, Python Machine Learning, Packt Publishing Ltd., 2015. |
| [53] | B. Qian, J. Su and Z. Wen, Orchestrating Development Lifecycle of Machine Learning Based IoT Applications: A Survey, 2019. |
| [54] | A. Rosenfeld, Picture Processing by Computer,, New York: Academic Press, 1969. |
| [55] | R. A. Kirsch, "The first scan done," SEAC NIST, USA, 1957. |
| [56] | X. Lu, Image Analysis for Face Recognition, Michigan, East Lansing: Michigan State University, 2008. |
| [57] | R. Maini and H. Aggarwal, "A Comprehensive Review of Image Enhancement," *JOURNAL OF COMPUTING, vol. 2, issue 3,* 2010. |
| [58] | D. Kundur and a. D. Hatzinakos, "A Novel Blind Deconvolution Scheme for," *IEEE TRANSACTIONS ON SIGNAL PROCESSING,* pp. 375-390, 1998. |
| [59] | J. Barta, "Image Compression Examples," [Online]. Available: http://www.pagetutor.com/image\_compression/. |
| [60] | S. Yeung, "Introduction to computer vision," 2015. [Online]. Available: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html. |
| [61] | V. Sze, Y.-H. Chen, T.-J. Yang, J. S. Emer and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *IEEE 105,* 2017. |
| [62] | D. Jamil, "Getting into Convolution Neural Networks," 3 May 2020. [Online]. Available: https://medium.com/@D3nii/convolution-neural-network-9320fa32fe9c. |
| [63] | J. Lu, J. Feng, Z. Fan, L. Huang, C. Zheng and W. Li, Automated Strabismus Detection based on Deep neural networks for Telemedicine Applications., 2018. |
| [64] | W. V.-v. d. V. Lambers and K. Lambers, Learning to Look at LiDAR: The Use of R-CNN in the Automated Detection of Archaeological Objects in LiDAR Data from the Netherlands, 2019. |
| [65] | K. Patel, "Overfitting vs Underfitting in Neural Network and Comparison of Error rate with Complexity Graph," 14 september 2019. [Online]. Available: https://towardsdatascience.com/overfitting-vs-underfitting-ddc80c2fc00d. |
| [66] | S. Davies, "Learning in spiking neural networks," %0 Conference Proceedings, 2013. |