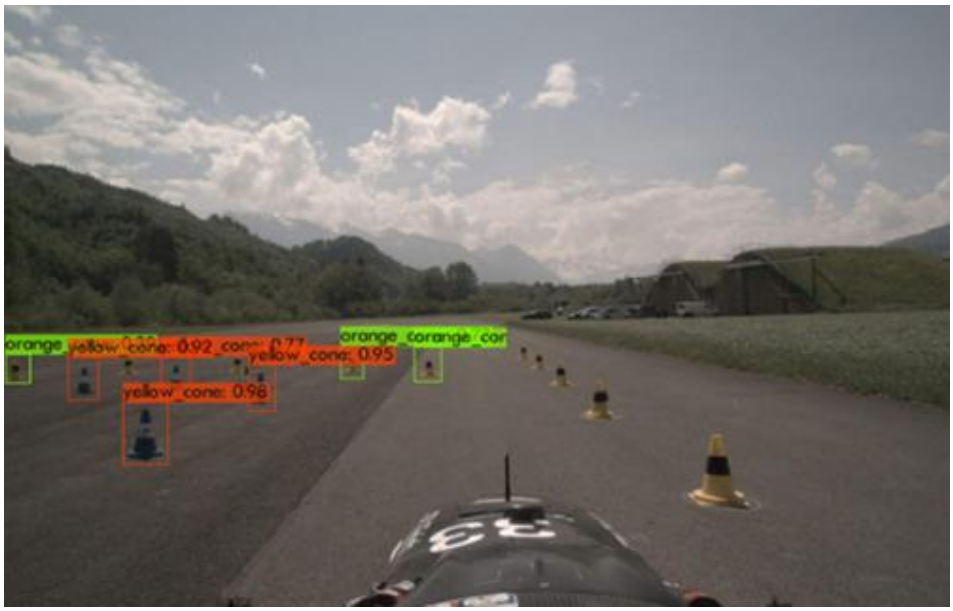
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Implementing Real Time Object Recognition with YOLOv4 for Effective Cone Tracking»



Των φοιτητών
Ανδρέα Οικονόμου & Δημήτρη-Μάρτιν Αραμπατζή
Αρ. Μητρώου: 113765 & 144229

Επιβλέπων
Περικλής Χατζημίσιος
Καθηγητής

Ημερομηνία 15 Ιανουαρίου 2021

Βεβαιώνουμε ότι είμαστε οι συγγραφείς αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχαμε για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχουμε καταγράψει τις όποιες πηγές από τις οποίες κάναμε χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνουμε ότι αυτή η εργασία προετοιμάστηκε από εμάς προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Περίληψη

Η παρούσα πτυχιακή εργασία πραγματεύεται την αναγνώριση αντικειμένων σε πραγματικό χρόνο με τη χρήση της 4ης έκδοσης του You Only Look Once (YOLOv4). Η εργασία αρχικά παρέχει μια εισαγωγή σε βασικές πτυχές της Τεχνητής Νοημοσύνης και της μηχανικής μάθησης και εξετάζει την εφαρμογή του YOLOv4 και προτύπων μηχανικής μάθησης που συνεπάγεται λήψη αποφάσεων υψηλού επιπέδου στα αυτόνομα οχήματα. Επιπλέον η εργασία εξετάζει την ιστορική εξέλιξη των αυτόνομων οχημάτων από τις βασικές τους τεχνολογίες, έως τα 6 επίπεδα αυτοματισμού που καθορίζονται σήμερα. Η εργασία επίσης αναλύει άλλες τεχνολογίες που ασχολούνται με την αναγνώριση αντικειμένων και εξετάζει το YOLOv4 κυρίως σχετικά με τη χρήση του για την παρακολούθηση κώνων ενός αυτόνομου οχήματος με δυνατότητα αυτο-βελτίωσης.

Ο στόχος της πτυχιακής εργασίας είναι να συζητήσει τα πλεονεκτήματα και τα μειονεκτήματα συγκεκριμένων μοντέλων τεχνητής νοημοσύνης και να εστιάσει στην εφαρμογή τους. Στα πλαίσια της εκπόνησης δόθηκε η ευκαιρία έγινε συνδυασμός της τεχνητής νοημοσύνης, της μηχανικής μάθησης και της λήψης αποφάσεων με βάση την αναγνώριση αντικειμένων σε πραγματικό χρόνο. Η εργασία καταλήγει στο συμπέρασμα ότι αυτός ο εξαιρετικά απαιτητικός τομέας τεχνολογίας είναι σήμερα πιο προσιτός ακόμη και σε ερασιτέχνες προγραμματιστές και παρόλο που εξελίσσεται εκπληκτικά γρήγορα, οι αλγόριθμοί του μπορούν να βρουν εφαρμογή σε καθημερινές χρήσεις. Η εργασία παρέχει πληροφορίες κατάλληλες για περαιτέρω έρευνα σχετικά με την προηγμένη μηχανική μάθηση και μπορεί να οδηγήσει σε καλύτερη κατανόηση του τρόπου λειτουργίας των αλγορίθμων και συνεπώς συμβάλλει, μεταξύ άλλων, στην ανακάλυψη νέων χρήσεων αναγνώρισης αντικειμένων σε πραγματικό χρόνο. Παρά τον πολύπλευρο χαρακτήρα της τεχνητής νοημοσύνης και της μηχανικής μάθησης που συνεπάγεται πολλές προκλήσεις και ηθικά ερωτήματα, οι άνθρωποι θα πρέπει να εξοικειωθούν με την παρουσία των αλγορίθμων αυτών και οι επιστήμονες θα πρέπει να συνεχίσουν αδιάκοπα να μελετούν κάθε πιθανή χρήση που μπορεί να κάνει τον κόσμο μας ένα καλύτερο μέρος.

Preface

During the academic year 2017-2018 we participated in the University team namely “FINEAS Racing Team”. FINEAS was supervised by Prof. Periklis Chatzimisios and was involved with autonomous vehicles. Motivated by our involvement, we decided to expand our knowledge of this sector, specifically on the field of machine learning. The primary goal of our thesis is to contribute, even in the slightest, to the systems in question, given the fact that the same process is used in every aspect of machine learning. As a result, we came to better understand the way algorithms function for the process of object recognition in real time and we hope, in this way, to encourage and help other researchers and maybe also other university students to utilize the process we went through, for further implementation. We chose to write our thesis in English so as to make it more comprehensible and, therefore, accessible to a wider public interested in the field of real time object recognition.

«Implementing Real Time Object Recognition with YOLOv4 for Effective Cone Tracking»

«Andreas Oikonomou & Dimitris-Martin Arabatzis»

Abstract

This B.Sc. thesis provides several insights into real time object recognition with You Only Look Once (YOLOv4). The thesis initially introduces certain fundamental aspects of Artificial Intelligence (AI) and machine learning and addresses the application of YOLOv4 to achieve machine learning standards that entail high level decision making to autonomous vehicles. The thesis then reviews the historical evolution of autonomous vehicles from the basic technologies involved, to the 6 automation levels that nowadays are defined. The thesis also analyses the technologies that deal with object recognition and examines closely YOLOv4 how it can be used to cone tracking by an autonomous vehicle capable of self-improvement.

The aim of the thesis is to discuss the advantages and disadvantages of certain artificial intelligence models and focus on their application. As a result, we have the opportunity to approach AI, machine learning and decision making based on real time object recognition. We came to the conclusion that this extremely demanding field of technology is nowadays more accessible even to amateur programmers and even though it evolves surprisingly fast, its algorithms can find implementation in every day uses. The thesis can have significant implications for further research into advanced machine learning and can lead to a better understanding of how algorithms work and therefore contribute, among else, to help discover new uses of real time object recognition. Despite the multifaceted character of AI and machine learning that entail certain challenges and moral questions, people should get accustomed to its presence in their lives and scientists should continue relentlessly to study every possible use that may make our world a better place.

Acknowledgments

I would like to thank Christina for being patient with me while I was writing this B.Sc. thesis during the pandemic of COVID 19. Moreover, a big thank you to my parents who they listen to me carefully every time I talk to them about my findings, even though they do not understand anything.

- Andreas Oikonomou

Table of contents

Abbreviations

ADS	Automated Driving System
ADAS	Advanced Driver Assistance System
AI	Artificial Intelligence
ANN	Artificial Neural Network
AV	Autonomous Vehicle
B.Sc. thesis	Bachelor of Science thesis
CNN	Convolutional Neural Network
FPS	Frames Per Second
GOPS	Giga Operations Per Second
CPU	Central Processing Unit
GPS	Global Positioning System
GPU	Graphical Processing Unit
IHU	International Hellenic University
LiDAR	Light Detection and Ranging sensor
mAP	mean Average Precision
ML	Machine Learning
NNA	Neural Network Accelerator
RGB	Red Green Blue
SSD	Single Shot multibox Detector
YOLO	You Only Look Once

1 Introduction

The Fourth Industrial Revolution also known as the second IT revolution or Industry 4.0, cited as such by Klaus Schwab [1] is characterized by disruptive technologies referred to as cyber-physical systems, one of which is Artificial Intelligence (AI). AI is defined as “developing computer programs to solve complex problems by applications of processes that are analogous to human reasoning process” [2]. It is presented by Poole and Mackworth [3] as “the study of the design of intelligent computational agents”. According to their definition, AI is the field that studies “the synthesis and analysis of computational agents that act intelligently”. A direct outcome of AI is machine learning. Although intelligence and learning has been so far the field of research for psychologists and zoologists, AI has led engineers to explore computational models conceived by scientists in their effort to explain human and animal learning to achieve machine learning. Therefore, Nils J. Nilson [4] defines machine learning as a change of the machine in “its structure, program, or data, based on its input or in response to external information. In such a manner that its expected future performance improves”.

Real object recognition (as a response to external information combined to advanced problem solving and decision-making abilities) can lead computers’ perception to action. This AI perception-action cycle is the foundation of the data flow needed to build an autonomous vehicle. According to Chris Schwarz [5] the idea of autonomous vehicles is as old as the automobile itself with one of the first efforts dating back in 1958 in Nebraska. But it is in the 1990s that autonomous vehicles became a controversial issue and as they evolve there are even more significant and complex challenges to be solved. One of these challenges has to do with algorithms that continuously develop to become more and more effective and reliable.

In May 2018, a collective effort from students from different departments from Alexander Technological Education Institute of Thessaloniki (ATEITHE), created "FINEAS Racing Team" with the goal of participating in "Formula Student" races, conducted by the Institution of Mechanical Engineers. "Formula Student" is the world's biggest competition for students that study engineering of any kind. Every year competitions are held all across the globe. Founded by the "Society of Automotive Engineers" in 1981, the first competition in Europe took place in 1998. At present there are more than 600 teams from universities all over the world competing with their self-constructed race cars. The winner is not necessarily the team with the fastest car, but the one with the best package regarding construction, performance, financial planning and sales arguments.

Figure 1.1: FINEAS Racing team logo

The competition consists of Static and Dynamic disciplines. In the Static events the engineers have to present their car and their development process to well-known judges from the economy, the automotive industry and prestigious racing series like Formula One. The disciplines upon we are being judged are Engineering design, Cost and Business plan. From all three Static events a team can get a maximum of 475 points. Moving further, the Dynamic events reveal the driving performance of the race car prototypes. From the disciplines Acceleration, Skid pad, Autocross and Track drive & efficiency, a team can get a maximum of 575 points which adds up to 1000 points which is also the highest score a team could get.

A lot has happened in the automotive industry since the competition's debut in 1981 which pushed the "Formula Student" to explore frontiers and add even more modern challenges. In 2010 for example there was created the electric class and after that the competition followed up with actual technical trends and created the new driverless class, which was raced for the first time in 2017. The goal in the driverless class is for the teams to build autonomous formula-style race cars which are capable of completing different disciplines without a driver. This kind of innovation led a group of young students, who are also automotive enthusiasts, to the creation of "FINEAS Racing Team", the first ever fully autonomous race team in Greece with the purpose of competing in the driverless class of "Formula Student". It must be also pointed out that since 2019 the competition has changed some of its dynamic disciplines to driverless only events.

The team (figure 1.2), that was created under the supervision of Prof. Periklis Chatzimisios and the coordination of one of the writers of this thesis, was composed of students from the Departments of Computer Science, Electronics as well as the Vehicles Department of ATEITHE. Being autonomous of course meant that at its core this partnership had a strong team of computer scientists that were working on the software of their first race car. The software had a lot of components and a lot more AI algorithms. At the top of this software stack everything starts from the ability that our algorithms give to the race car to understand its environment, take decisions and act upon those perceptions. To take this capacity to the fullest, our team decided to use a complex system of sensors and sensory data. A part of those sensors is the cameras of the system that help to offer "vision" to the vehicle. It is this problem that triggered our research. We needed a solution to help our race car understand objects in real time from its cameras and use that knowledge to path a perfect race line through the competition's race tracks.

Figure 1.2 FINEAS team members in a team meeting

This being our fundamental query for this thesis, firstly we are going to provide more information about autonomous vehicles. Starting with a historical evolution overview to comprehend how we arrived at the present point to be able to talk about self-driving cars and vehicles in general. We are going to have a look at the technologies and the sensors used by almost every manufacturer, in any possible combination, to achieve the best results in making a vehicle as driver independent as it can be. We will also present and describe the six automation levels that the Society of Automobile Engineers (SAE) has set globally. Secondly, Convolutional Neural Networks and their evolution and variations will be examined and analyzed starting from the basic theory underlying Convolutional Neural Networks to the YOLO algorithm which is also the one that we are going to utilize to achieve this thesis's purpose and FINEAS team's goal as well. In addition, we will try to exhibit the benefits that YOLO has for our specific use case and prove why it is the best algorithm currently available to use for implementing real time object recognition.

Moving on from theory to practice, we are providing detailed information about the tools we used in our project, along with the reasons why we decided to work with them. Moreover, detailed instructions can be found on how someone could replicate all our work or parts of it in order to use it for their own projects and experiments. Finally, in the last chapter we will go over our final thoughts and opinions, suggesting other possible use cases that our thesis could be useful to but also trying to prove that, despite the negative connotations that machine learning and specifically object recognition may entail, we cannot afford to disregard its incredible potential.

2 Autonomous Vehicles

2.1 Introduction

An autonomous vehicle is one that can drive without input from a human driver from a starting point to a predetermined destination in “autopilot” mode using various in-vehicle technologies and sensors, including adaptive cruise control, active steering (steer by wire), anti-lock braking systems (brake by wire), GPS navigation technology, lasers and radar. These types of vehicles are also known as self-driving cars, driverless cars, or robotic cars. The term self-driving car is becoming a standard as these technologies continue to mature.

The continuing evolution of automotive technology aims to deliver even greater safety benefits and automated driving systems that — one day — can handle the whole task of driving when we do not want to or cannot do it ourselves. Fully automated cars and trucks that drive us, instead of us driving them, will become a reality [6].

2.2 Historical evolution

The academic roots of autonomous vehicles and the concept of self-driving automobiles may be found in the Middle Ages, centuries before the invention of the automobile. Autonomous vehicles appear as an early idea since then as one can see on a sketch by Leonardo da Vinci which could be described as a rough blueprint for a self-propelled cart (figure 2.1) [7].

Nowadays, autonomous vehicles should be able to operate without human control, they should require no human intervention. Nevertheless, there has been a long way from guide-wire vision and automatic free way to microelectronics and algorithms. By the 1920s traffic accidents have already become a main concern in the USA, and a social issue caused basically by drivers’ carelessness and misconduct emerged. In other words, the need to annihilate the factor of human error had already begun to concern car manufacturers. In an effort to address road safety, the development of the fields of aviation and radio engineering was about to set the foundations for driverless vehicles. In 1921, Radio Air Service presented their first driverless car in Ohio, USA. The car was 2.5 m long and was controlled from an army vehicle about 30 m behind. Even though the car was not technically autonomous since it was remotely driven by a driver in another vehicle nearby, the impact was huge, and the Press took interest on the prototype immediately [8].

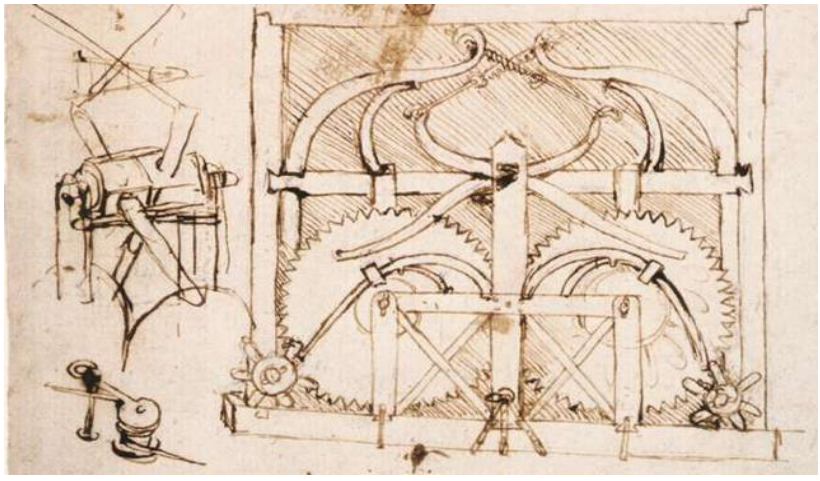


Figure 2.1: Leonardo da Vinci sketch for self-propelled cart [7]

In 1926, a Chandler car model with receiving antennas operated by another car was demonstrated by Houdina Radio Control in New York and was considered a primitive form of autonomous vehicles. In December of the same year Achen Motors launched Phantom Auto in Milwaukee. In the World Fair, in 1939, Norman Bel Geddes presented Futurama on behalf of General Motors, a company which was mainly concerned with safety and had already made a significant research on traffic accidents. Futurama was an embedded-circuit powered electric car controlled by radio.

A step away from radio remote-controlled vehicles was a miniature car, fabricated by RCA Labs in 1953. That miniature car was controlled by wires that run on a laboratory floor. Another driverless car was tested in 1958, this time created by Leland Hancock and L.N. Ress in actual highway installations near the town of Lincoln, Nebraska. A series of lights were also added along the edge of the road to assist the vehicle. General Motors immediately took to Hancock's idea and showcased Firebird in Motorama in 1959-1960. "Firebird" was a series of driverless cars with electronic guide systems. In 1966, Ohio State's University's Communication and Control Systems Laboratory realized a project with electronic devices imbedded in the roadway.

In the 1960s another significant event took place: the United Kingdom's Transport and Road Research Laboratory presented a Citroen DS which was controlled with the use of magnetic cables in the road. The car managed to reach the speed of 130km/h and it proved to be more effective than a car under human control. At the same time the United States Bureau of Public Roads suggested and offered infrastructure for the construction of experimental electronically controlled highway in four states. Only one managed to implement the project, yet the cost was prohibitive, and they quickly abandoned the project.

About twenty years later, in the 1980s, Mercedes Benz launched a robotic vehicle, a vision guided driverless van. It is in this decade that a series of projects were presented at a national or international level both in Germany and the USA. A project worth mentioning is PROMETHEUS project and EUREKA (European Traffic of Highest Efficiency and Unprecedented Safety) based on the field of vision based autonomous driving. Moreover, ALV project presented the first road following demonstration with computer vision, LIDAR and autonomous control. HRL laboratories added the first off-road map and sensor-based autonomous navigation on the ALV. Establishment of the National Automated Highway System Consortium culminated in 1997 but was later dropped because of lack of funding.

The next decade introduces newer vehicles which slowly but steadily become more and more efficient. In 1991, Bundeswihr University in Munich launched a vehicle that drove more than 1000 km on a Paris motorway in standard heavy traffic. Nonetheless, it was still a vehicle with characteristics of a semi-autonomous car with considerable human intervention.

In 1995, an S-Class Mercedes Benz, created by Dickman, travelled in different countries covering 1590km with jolting computer vision and microprocessors with integral memory for parallel processing to react in real time. In this case, it was the first-time experts had to really deal with 95% autonomous driving. In 1995 Carnegie Mellon University's Navlab succeeded 98,2% autonomous driving and in 1996 Alberto Broggi and the University of Parma demonstrate project ARGO with 94% autonomous driving guided by stereoscopic vision and algorithms to understand the environment. The vehicle drove continuously for 55km in a fully autonomous mode.

In the early 2000s in the Netherlands Park Shuttle became the first public transport system. At the same time, in the USA, several autonomous vehicles were designed and used in the Army (DEMO I / II / III) all equipped with real time control systems.

In 2010, a series of important advancements occurred. Firstly, General Motor's Electric Networked Vehicle was unveiled at the Expo 2010 in Shanghai. A major competition, VIAC or VisLab Intercontinental Autonomous Challenge was funded by the European Research Council and vehicles with negligible human intervention and high level of autonomy were presented. Furthermore, Audi's autonomous TTS research car set a benchmark as it was guided by emerging software and algorithms. Last but not least, Volkswagen's TAP (temporary AutoPilot) was launched as a milestone on the path towards accident-free driving. In 2011, the European Union starts the HAVEit (Highly Automated Vehicles for Intelligent Transport) project and the first cars licensed for autonomous driving on the streets and highways were introduced in Berlin, Germany. In 2013, Toyota presented ITS, Intelligent Transport System for safety reasons only, and in the same year Nissan used virtual steering column for its Infinity Q50 model. Finally, Parma presented its advanced autonomous car BRAiVE, a pioneer in vehicular robotics, totally autonomous. The next year Nissan's Leaf with an Advanced Driver Assistance System-not yet fully automated-was granted a license plate and was first tested in California. The same year Induct Technology France introduced Navia, a robotically driven electric shuttle, equipped with a real time three-dimensional map of the surroundings and was tested in Switzerland, England and Singapore. Finally, not long after that, Google announced its plans to demonstrate a driverless car manufactured in its secret Xlab [9].

Nowadays almost all automobile manufacturing companies in collaboration with IT companies are planning to produce fully autonomous automatic vehicles: e.g., Volkswagen, Audi, Ford, General Motors, Nissan, Hyundai, Toyota, but also by 2024, Jaguar and Land Rover, by 2025, Daimler and Benz [10].

Generally autonomous cars have evolved from automatic lanes to fully autonomous vehicles and countries all over the world regard them as an asset which will reduce crashes and car accidents, will contribute to safety in the roads and at the same time significantly reduce energy consumption, pollution and traffic congestion. On the other hand, cost remains an issue to take into account and experts are still seeking ways to afford this environmentally friendly, intelligent vehicle [11].

2.3 Technologies involved

Self-driving vehicles contain a considerable amount of technology in them. The technology has remained reasonably stable within these vehicles, but the software behind the cars is continually evolving and being upgraded. All technology has advantages and disadvantages. As a standalone device, individual sensors found in AVs will fail to perform correctly. High quality overlapping data patterns are generated by combining the strengths of each sensor such that the data processed can be as precise as possible. This technology is called sensor fusion and it works by using information from the sensors mixed together (figure 2.2) [12].

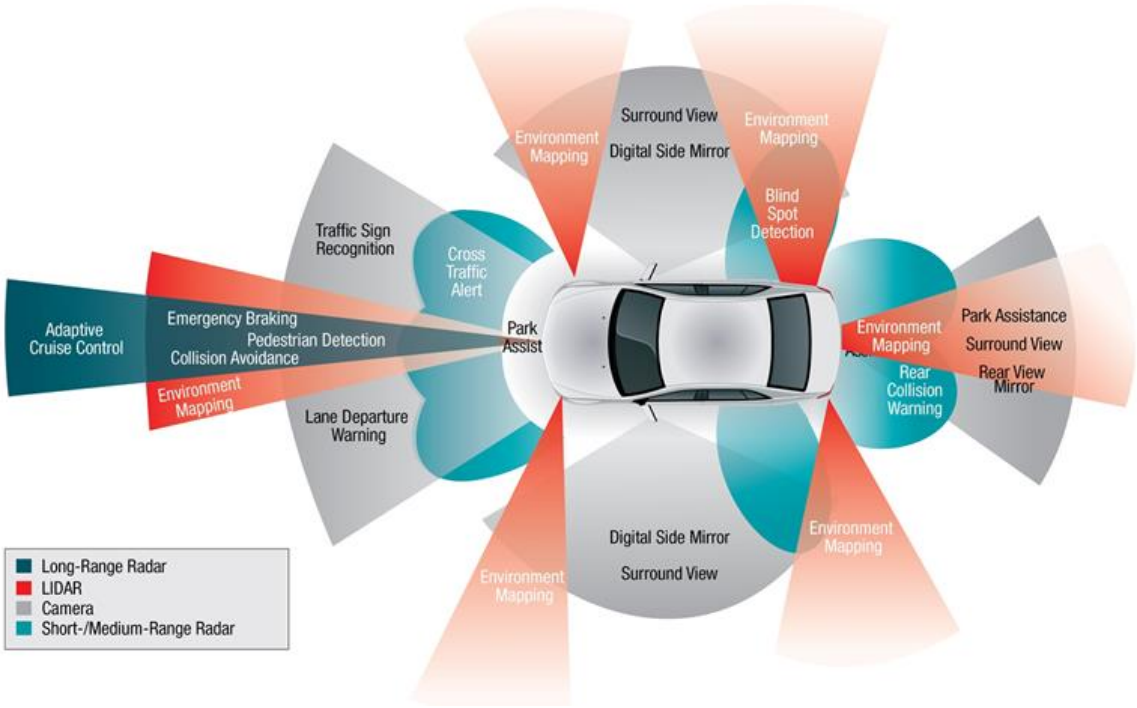


Figure 2.2: Visual representation of sensor fusion [12]

(a) Cameras

As technology billionaire Elon Musk has famously said, cameras are the only sensor technology needed for self-driving vehicles. The only thing needed to be able to truly interpret the images they received, are the appropriate algorithms. Camera images capture all a vehicle needs to drive, but we are also discovering new ways for machines to interpret visual imagery and convert it into actionable 3D data. To stitch together a 360-degree view of their surroundings, autonomous vehicles rely on cameras mounted on either side, front, back, left and right. Some have a wide field of vision and a narrower range, as much as 120 degrees. In order to include long-range graphics, some rely on a narrower vision.

In order to have a full picture of what is behind the vehicle for it to park itself, certain vehicles also incorporate fish-eye cameras, which feature super-wide lenses that provide a panoramic view [13]. For instance, Tesla cars have 8 external cameras to help them understand the world around them.

Cameras have their drawbacks, even though they have reliable visuals. Details of the surrounding area can be distinguished, but the distances of such objects need to be evaluated to determine precisely where they are. It is often more difficult for camera-based sensors to track objects in conditions of low visibility, such as mist, rain or darkness. [14].

Figure 2.3: Cameras used in a Tesla Car [15]

(b) Ultrasonic sensors

Ultrasonic sensors can play an integral part in the safety of a vehicle without a driver. By using echo periods from sound waves that bounce off nearby objects, they mimic the navigation mechanism of bats. The sensors can identify how far away the objects are by using this information and warn the vehicles onboard the closer they come [16]. Ultrasonic sensors are considered active sensors . Ultrasonic transducers, much like radar and sonar, they determine their targets by analyzing reflected signals. Using ultrasonic waves, these sensors measure the time between emission and reception thus measuring the distance between objects. This makes them especially successful at locating approaching barriers within close range, which is why they are such good sensors for parking.

A high-frequency sound wave (above 18 kHz) is generated by the transducer, translating electrical energy into sound and then back again. In different ways, these artifacts reflect ultrasonic sound waves: whereas solid materials such as aluminum, plastic and glass reflect sound well, soft materials such as clothing appear to absorb the waves. Accuracy can also vary; ultrasonic sensors specialize in the identification of solid threats, such as traffic cones and barriers. They are ideal for applications with lower speeds, short to medium distances, such as lateral movement, blind spot monitoring and parking (figure 2.4).

Every sensor has its own ideal functionality that works well for autonomous driving. For short-range uses, ultrasonic sensors are a perfect match, but for other uses, not so much . Here are the positives and limitations:

Pros [17]

- Compact and stable, they do not have moving components.
- Unaffected by the color or clarity of the material,
- Unaffected by conditions in the atmosphere such as rain, snow and dust
- Unaffected by light levels, working equally well in darkness.
- Good fit for calculating the distance to a parallel surface.

Cons [17]

- Precision is determined by angle and soft materials.
- Accuracy is impacted by sudden temperature fluctuations.
- Limited detection range-a maximum range of 8 m is available for our long-range sensors

(c) Speed and angle sensors

By measuring acceleration on both the longitudinal and vertical axes, speed sensors monitor wheel speed and relay this data to the driving safety system. On the other side, to assess where the front wheels are pointing, the steering angle sensor is used. It is possible to calculate the dynamics of the AV by combining these two sensors with other data. If geolocation is degraded or missing, such as when going through a cave, data from speed and angle sensors are merged to create a "Dead Reckoning" solution. Dead Reckoning" is essentially the method of calculating the current position of a vehicle, using a previous position and advancing that position based on known or estimated speeds over the time and course elapsed." [16]

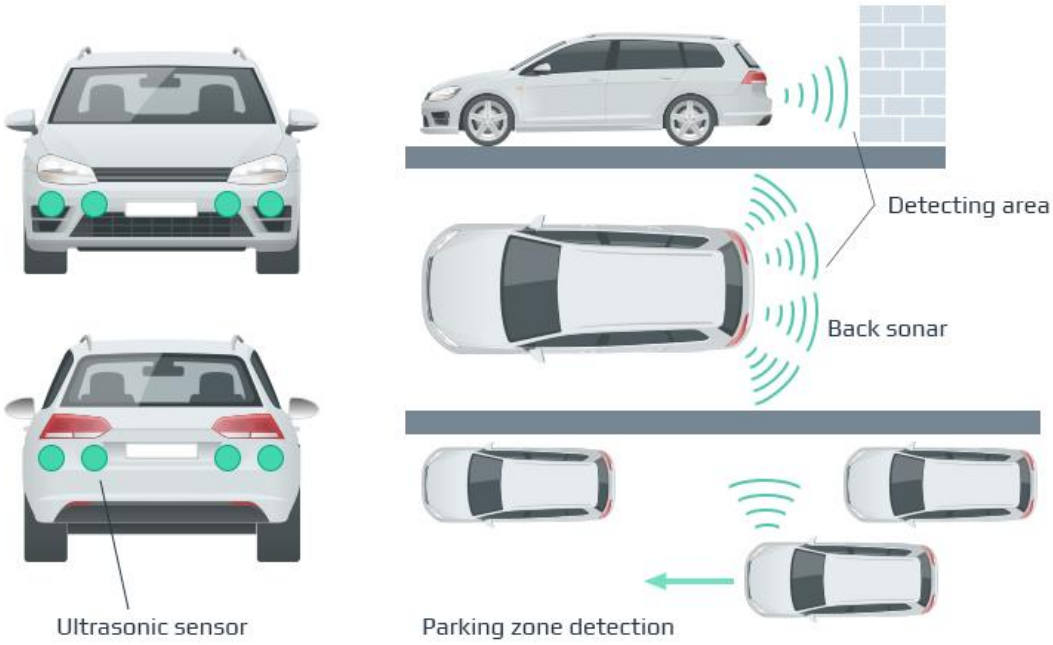


Figure 2.4: Ultrasonic sensors

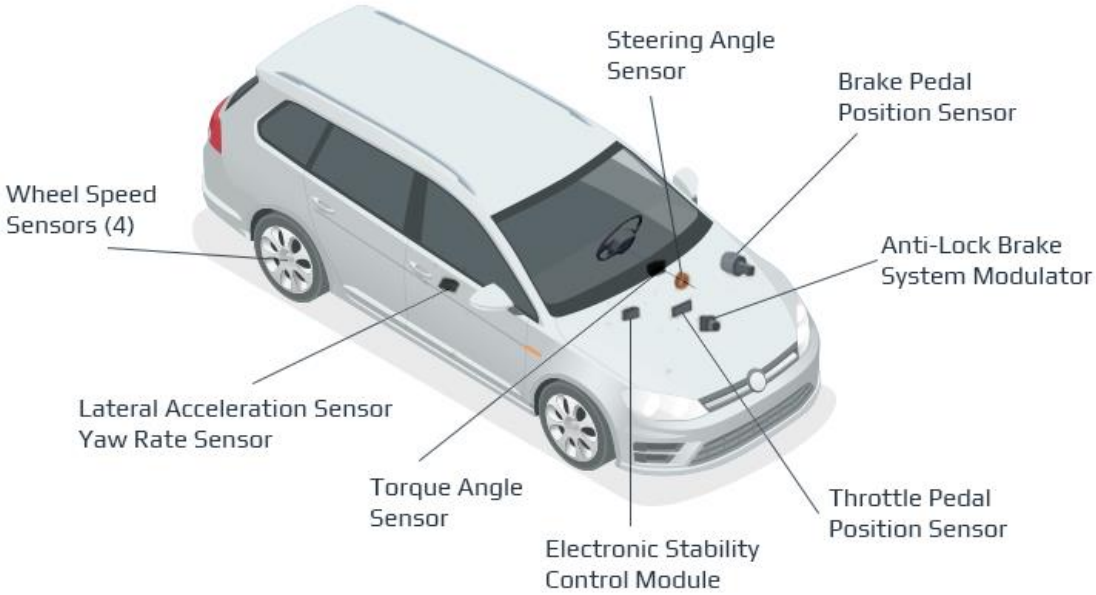


Figure 2.5: Overview of sensors that potentially could be used for "dead reckoning" process [16]

(d) Radar

Radar, along with LiDAR, visualization and cameras, is one of the key means that self-driving vehicles use to envision. The lowest resolution of the three is radar, but it can see through unfavorable weather conditions [14].

In order to search the area, detect objects at a distance and determine their speed and disposition, both Radar and LiDAR sensors use common principles. Radar, Radio Detecting and Ranging. Radio waves are used and the device consists of a radio or microwave domain transmitter emitting electromagnetic waves, a transmitting antenna, a receiving antenna (often the same transmitting and receiving antenna is used), a receiver and a processor to determine the properties of the object(s). Radio waves from the transmitter reflect off the object and return to the receiver, giving information about the object's location and speed. Its long working distance is one of the key benefits of Radar over other sensors. It may function in situations and environments of higher complexity since it is not as susceptible to mud, for instance, because it has no artificial moving parts.

On the other hand, it will also detect objects that give the target a false size due to reflection and/or optical disturbances. For instance, it was possible to classify a soda can on the road as a house. It also does not have the fidelity of some other sensors, which indicates that it is not as precise as some alternatives. Reaction speed is particularly critical for vehicles traveling quickly (50-70 km/h or more) and it is one of the major benefits of radar sensors over LiDAR. In autonomous vehicles today, radar sensors are very common. In certain situations, Radar can be a reasonable option for Applied Autonomy, where cars often drive slowly and in controlled conditions. But because the need for fast response times is important and a construction site environment, for example, provides the Radar with a lot of artifacts to respond on, another sensor may be a safer option [18].

Figure 2.5: A radar alongside a camera module. [19]

(e) LiDAR

LiDAR sensors are what you will see spinning around on top of self-driving vehicles (Figure 2.6) [20]. LiDAR or Light Detection and Ranging sensor is probably the best sensor used in AVs. Light beams from a laser beam, actually at the speed of light, emanate to distinguish surrounding objects. Its benefits include impressively precise perception of depth, which enables LiDAR to know the distance to an object within centimeters, up to 60 meters away. It is also extremely suited for 3D mapping, ensuring that returning vehicles will then predictably navigate the environment, a major advantage for most self-driving technology. The high number of areas that show potential for improvement is one of LiDAR's main strengths. These include solid-state sensors that will decrease their cost tenfold, sensor range improvements of up to 200 m, and 4-dimensional LiDAR sensors that can detect the speed and location of an object in 3-D space. In spite of these exciting advances however, LiDAR is still hindered by its considerable expense, a crucial factor.

This sensor complements and fits well with any current safety features because of its ability to measure the size, pace, and direction of the vehicle as well. [14].

Figure 2.6: LiDAR on an autonomous vehicle [20]

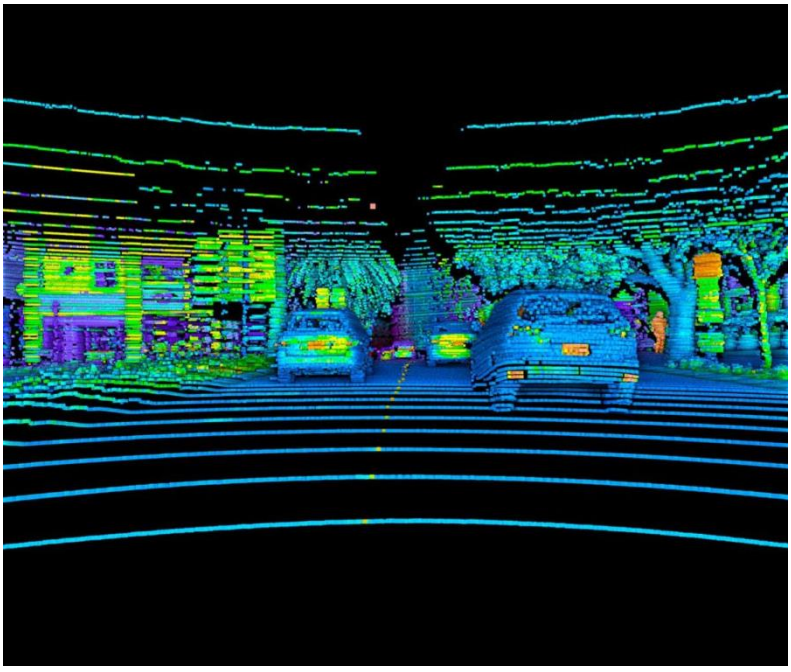


Figure 2.7: Visual 3D representation of LiDAR data [21]

Huge 3D maps a be generated with LiDAR (Figure 2.8) [22], which was its original use. Inside them, the car or robot can predictably maneuver . You will know ahead of time the limits of a lane by using a LiDAR to map and navigate an area, or that there is a stop sign or traffic light 500 m ahead. This kind of predictability is precisely what is expected by a technology used by self-driving cars and has been a major reason for the advancement of the last 5 years [21].

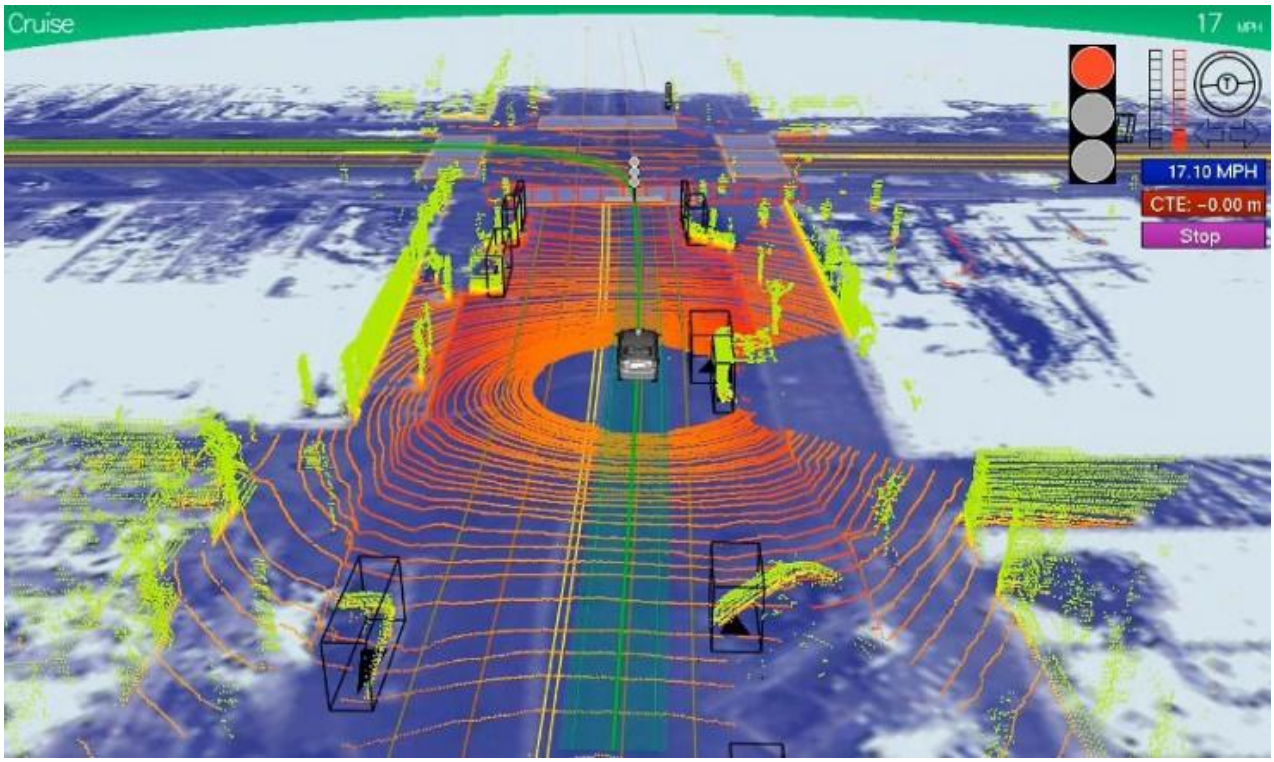


Figure 2.8: Visual 3D Map of a street created from LiDAR data [22]

Modern LiDAR helps you to distinguish between an individual moving on a bike or a person, and also at what pace and direction they are going in.

The combination of predictability, amazing navigation and high-resolution detection of objects has ensured that LiDAR is today's main sensor in self-driving vehicles, and it is difficult to see the shift in dominance [23].

(f) Other sensors

Standard GPS mapping, along with ultrasonic sensors and inertial sensors, can also be used for self-driving vehicles to get a full picture of what the vehicle is doing as well as what is happening around it. The more data generated, the easier it gets in the realm of machine learning and self-driving technology [14].

(g) Computer Power

Both self-driving vehicles, and virtually all modern cars, need an on-board computer to handle anything that happens to the vehicle in real time. AVs require intense computing speed, so they use GPUs to do their calculations, rather than conventional CPUs. Still the best GPUs, however, have begun to prove to be inadequate for the intense data processing requirements found in self-driving cars. This is why, Tesla has introduced Neural Network Accelerator chips, also called NNA.

A neural network accelerator is a processor that is optimized exclusively manage neural network workloads. As the names indicates, it is very effective in doing its job of taking data, clustering and classifying it at a very quick pace. These NNAs have extreme real time computing capacity, capable of handling image processing in real time.

Below is how many Giga operations per second they can perform, often referred to as GOPS, for a comparison of CPUs, GPUs, and NNAs:

- `CPU: 1.5 GOPS`
- `GPU: 17 GOPS`
- `NNA: 2100 GOPS`

NNAs are the clear winner, by many times [14].

2.4 Sensor Fusion

The ability to pull together signals from various radars, lidars and cameras to form a single model or picture of the world surrounding a vehicle is called sensor fusion. The resulting model is more detailed since it combines the various sensors' strengths. To support more intelligent human-like behavior, vehicle systems then use the information generated by sensor fusion.

There are underlying strengths and disadvantages within each sensor type, or "modality." Except in difficult weather situations, radars are very powerful at reliably assessing distance and altitude, but they do not read street signs or see the color of a traffic lights. Cameras read signs or identify things very well, such as pedestrians, bicyclists or other cars. However, gravel, light, fog, snow or darkness will readily blind them. Objects can be tracked correctly by lidars, but they do not have the range or affordability of cameras or radar.

Sensor fusion uses software algorithms to pull together the data from each of these sensor types to provide the most detailed, and thus precise, environmental model possible. Via a method known as interior and exterior sensor fusion, it can also compare data extracted from inside the cabin.

Figure 2.7: Completely self-driving electric Golf [24]

A vehicle may also use sensor fusion to combine information from different sensors of the same kind, such as radar, for example. By taking advantage of partly overlapping fields of view, this increases perception. More than one sensor can sense objects at the same time, as several radars observe the environment around a car. Interpreted by global 360° perception applications, it is possible to overlap or integrate detections from these different sensors, improving the likelihood and efficiency of identification of objects around the vehicle and offering a more precise and accurate representation of the environment [25].

The following are the three fundamental techniques of merging sensor data:

- Redundant sensors: All sensors give the same information for the world.
- Complementary sensors: The sensors provide independent (disjoint) types of information about the world.
- Coordinated sensors: The sensors collect information about the world sequentially.

The following are the three fundamental sensor communication schemes:

- Decentralized: No communication exists between the sensor nodes.
- Centralized: All sensors provide measurements to a central node.
- Distributed: The nodes interchange information at a given communication rate (e.g., every five scans, i.e., one-fifth communication rate).

The centralized system may be considered as a special case of a distributed system in which every scan is transmitted by the sensors to each other. In figure 2.9 [27], a pictorial description of the fusion process is given.

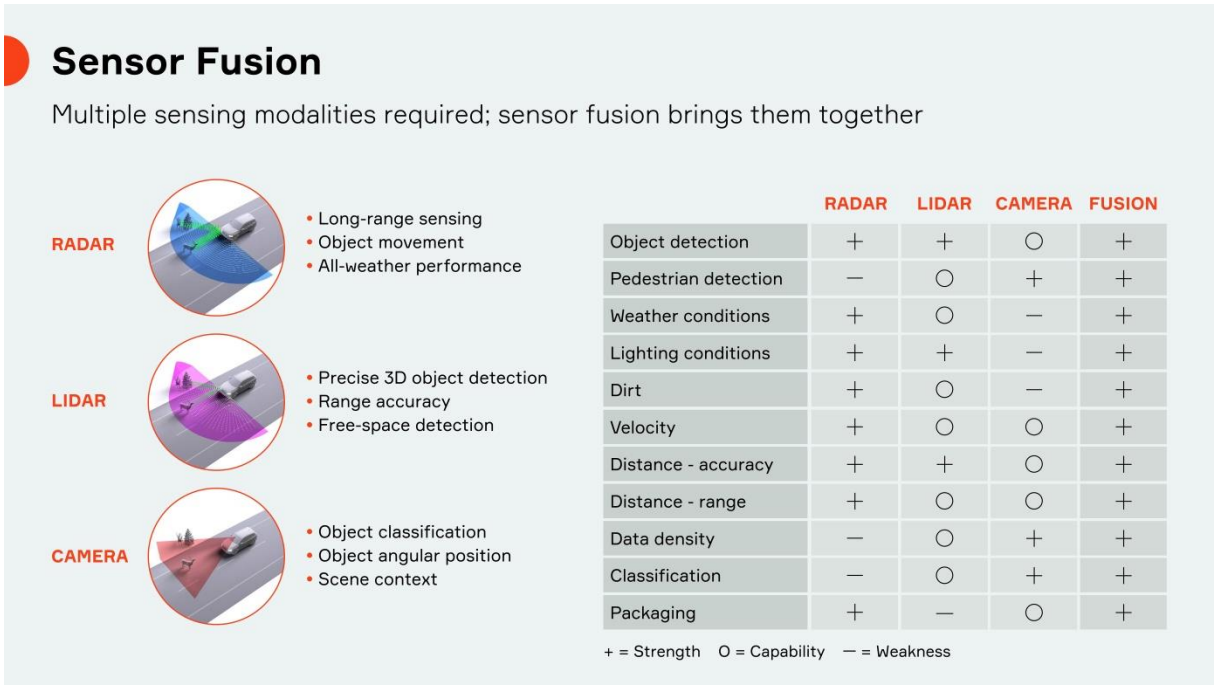


Figure 2.8: Sensor strengths and weaknesses [26]

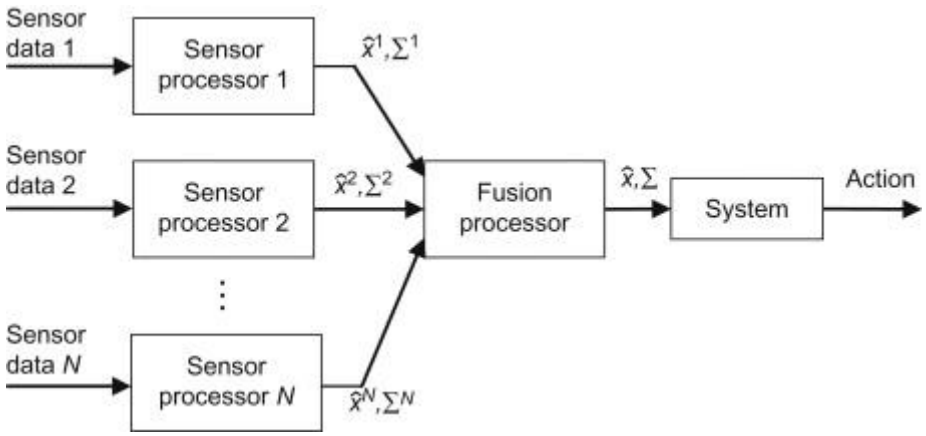


Figure 2.9: Illustration of the sensor fusion process [27]

2.5 Automation levels

The Society of Automobile Engineers sets the most internationally accepted driving automation standards, and they are classified into six driving automation levels [28].

(a) Level 0 - No automation

Level 0 comprises all vehicles in which all primary vehicle controls are still completely and exclusively controlled by the driver. Included are the braking, steering, throttle and motive power of the engine. This group contains many older motor cars. Level 0 will also be considered for vehicles which have warning systems but do not deliberately alter the vehicle's speed or direction [28].

(b) Level 1 - Driver assistance

This is the lowest automation standard. For driver aid, such as steering or braking, the car has a single integrated system (cruise control). Adaptive cruise control, also known as adaptive cruise control, counts as Level 1 because the human driver controls other facets of driving, such as steering and braking, where the vehicle should be kept at a reasonable distance behind the next car [28].

(c) Level 2 - Partial driving automation

This means Advanced Driver Assistance Systems or ADAS. Both steering and acceleration/deceleration may be controlled by the machine. The automation here falls short of self-driving because a person sits in the driver's seat and may at any point take control of the vehicle. Tesla Autopilot and Cadillac (General Motors) all rank as Category 2 Super Cruise systems [29].

(d) Level 3 - Conditional driving automation

From a technical viewpoint, the leap from Level 2 to Level 3 is substantial, but subtle if not insignificant from a human perspective. Vehicles of class 3 have capabilities for "environmental detection." An ADS can make intelligent choices for the car, such as speeding past a slow-moving vehicle, but human override is still needed. If the ADS is unable to perform the task, the driver must remain alert and ready to take control [29].

(e) Level 4 - High driving automation

The main distinction between the automation of Level 3 and Level 4 is that when things go wrong or there is a system malfunction, Level 4 vehicles may intervene. In this way, in certain conditions, these cars do not require human interaction. An individual, however, still has the possibility of overriding it manually.

Level 4 vehicles can work in self-driving mode, but they can only do so within a restricted region before legislation and technology expands, typically in an urban setting where peak speeds approach an average of 30 miles per hour. This is referred to as geofencing. As such, most current Level 4 vehicles are oriented for ride sharing. For instance:

- NAVYA, a French company, is already building and selling Level 4 shuttles and cabs in the U.S. that run fully on electric power and can reach a top speed of 55 miles per hour [30].
- Alphabet's “Waymo” recently unveiled a Level 4 self-driving taxi service in Arizona, where they had been testing driverless cars without a safety driver in the seat, for more than a year and over 10 million miles.
- Arrival Ltd, a London based technology company develops electric vehicles, primarily lightweight commercial vehicles such as vans and buses. In June 2020, Arrival announced a new passenger bus designed for coronavirus-era social distancing.
- Volvo and Baidu also announced a strategic partnership to jointly develop Level 4 electric vehicles that will serve the robotaxi market in China.

(f) Level 5 - Full driving automation

Level 5 is the ultimate experience of self-driving automation. It is not the driver's responsibility to control the car and, in some situations, they do not even have the choice to do so. The driver is not supposed to take over driving until the ADS is employed. The automatic driving functions can be used anywhere under all conditions. The Level 5 vehicle is a hundred percent driverless. Any human inside the vehicle is just a passenger and need never be involved in driving [29].

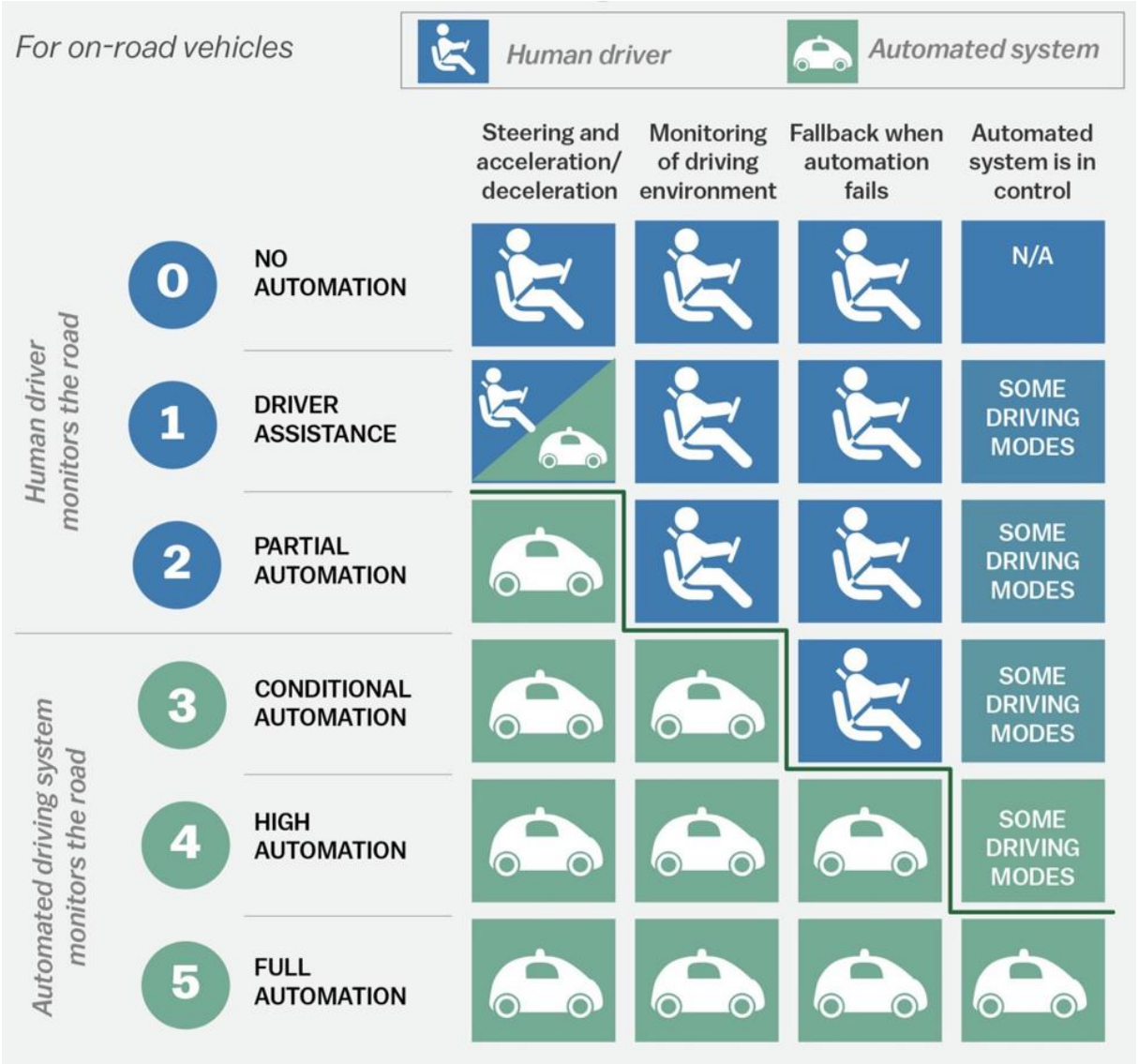


Figure 2.10: Diagram of six levels of automation [31]

2.6 Future expectations

Clearly, there is the ability for autonomous vehicle technology to reshape not only how we ride, but how we construct cities and connect with the world around us. In brief, decreases in gasoline spending, pollution, injuries, traffic and associated efficiency gains have led Morgan Stanley to project that automated vehicles would save the US economy $1.3 trillion a year. – globally, savings are projected to be over $5.6 trillion [32].

2.7 Potential concerns

Sadly, as every technology, the implementation of autonomous vehicle systems comes with potential concerns. Not without potential problems is the introduction of autonomous vehicle systems. Data privacy and the scope for autonomous vehicle hacking are chief among these. Hackers have shown in the past their ability to bypass and gain control of automotive networks via hard-wired computer links in a car. With the advent of telematics systems (internet-connected automotive services) in vehicles, hackers have the ability to find, track, and/or connect to a vehicle remotely. The future consequences

of this weakness are immense, and the market for connected cars seems to be outpacing the pace at which modern, comprehensive driver safety initiatives are being introduced.

Laws are being enacted as a means to enforce tougher regulations on automakers for cybersecurity, however the law has not advanced past the first reading. The SPY Car Act will mandate the National Highway Traffic Safety Administration to set performance standards for vehicle manufactures to include hacking protection, data security, and prevention of hacking, while also establishing a "cyber dashboard" that would reflect the safety ranking for every new car for sale.

A combination of human-operated and autonomous cars is expected to occur over several years to come, posing concerns about liability, insurance, and equity. Questions over liabilities in situations of injuries face an additional obstacle. It will be important to address how defects will be assessed in human-autonomous vehicle collisions and how such accidents will in turn impact car insurance practices. Car insurance, as we know it today, could change completely with the introduction of autonomous cars, based on ownership models. If, as is the case today, the dominant autonomous vehicle ownership model encourages the outright purchase of each car, it is possible that low-income people would not benefit from the same protection and comfort advantages that wealthier autonomous vehicle owners will have [33].

3 Technologies

3.1 Artificial Neural Networks

Artificial neural networks (ANNs) are mathematical or computational models of electronic computing and similar fields that are inspired by the brain of a human being, the central system that is able to recognize and learn patterns. A system constructed like this is also capable of solving more complicated problems. In general, ANNs are presented as highly interconnected neuron networks which can compute values from inputs [34].

The human brain consists of one hundred billion cells linked by synapses, known as neurons. If a neuron is fired by sufficient colligation inputs, the neuron will fire. This process is known as "thinking". By constructing a neural network on a computer, we are able to mimic this process. There are input and output neurons on a neural network that are connected by weighted synapses. The weights influence the proportion of forward propagation that moves through the neural network. Throughout the back propagation, the weights can be modified; this is also the portion where the neural network is in the learning process. This process of forward propagation and backward propagation is implemented iteratively on each piece of data of the training dataset. The larger the dataset size and the greater the spectrum of those data, the faster the neural network can learn and the better it can get to predict outputs. Neural networks can therefore be used to extract patterns and detect trends that are too difficult to be detected by either humans or other computer techniques, with their greater capacity to derive meaning from complicated or imprecise data.

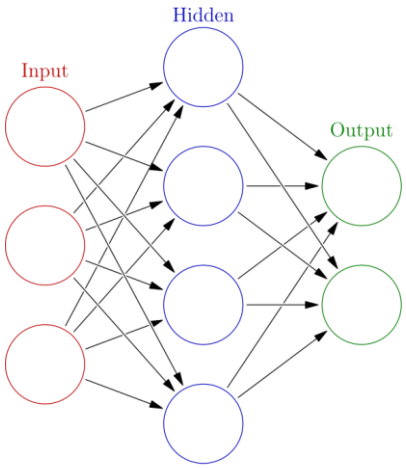


Figure 3.1: Simple neural network

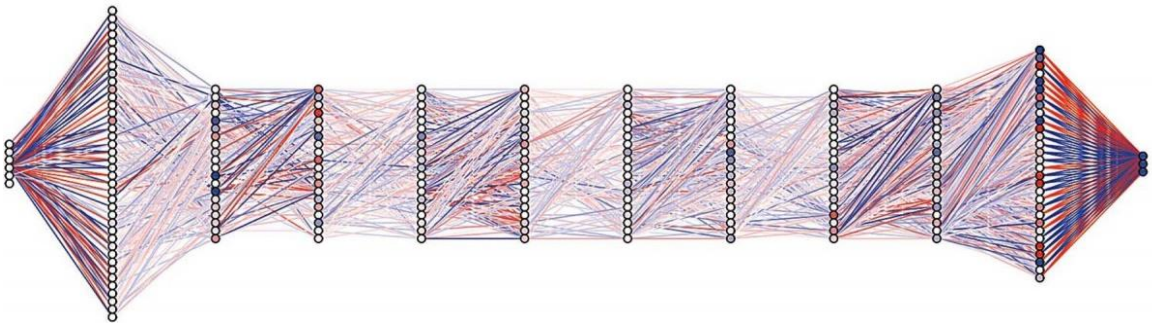


Figure 3.2: Highly complex neural network

3.2 Working with images and neural networks

Neural networks ingest, process and understand images as tensors, and tensors are matrices of numbers with additional dimensions. It can be very difficult to imagine this, so let us approach it by comparison. Only an integer, such as 7, is a scalar. A vector is a list of numbers (e.g. [7,8,9]) and a matrix is a rectangular number matrix which, like a spreadsheet, occupies many rows and columns. Geometrically, if a scalar is a zero-dimensional point, then a vector is a one-dimensional line, a matrix is a two-dimensional plane, a stack of matrices is a three-dimensional cube, and you reach the fourth dimension when each member of those matrices has a stack of feature maps attached to it [35]. Here is a 2x2 matrix for reference:

```
[ 1, 2 ]
[ 5, 8 ]
```

A tensor encompasses the dimensions beyond that 2D plane. A three-dimensional tensor can easily be pictured, with the number array arranged in a cube. Here is a flatly illustrated 2x3x2 tensor (imagine the bottom element of each 2-element array expanding along the z-axis to intuitively comprehend why it is called a 3-dimensional array) (Figure 3.3) [35]:

In code, tensor would appear like this: `[[[2,3],[3,5],[4,7]],[[3,4],[4,6],[5,8]]]`. And here is a visual representation (Figure 3.4) [35]:

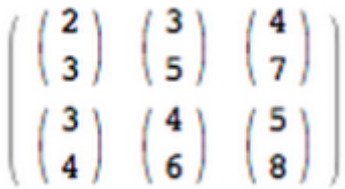


Figure 3.3: Three-dimensional tensor with the array of numbers arranged in a cube [35]

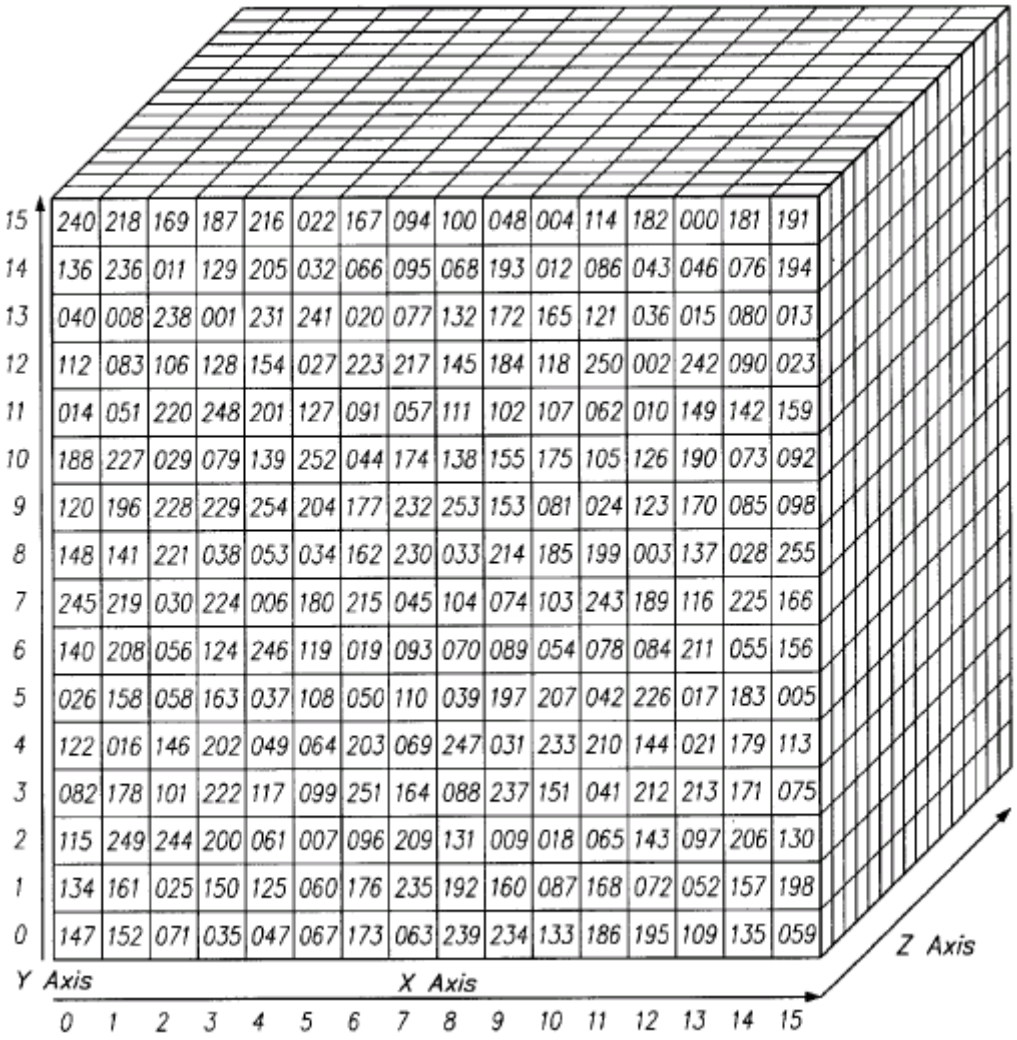


Figure 3.4: Visual representation of a tensor [35]

In other words, arrays nested inside arrays form tensors, and the nesting will go on indefinitely, allowing for an arbitrary number of dimensions much greater than what we can spatially visualize. A 4D tensor would basically replace each of these scalars with an array nested one level deeper. Fully convolutional networks, including the one below, deal with 4D tensors (Figure 3.5) [35].

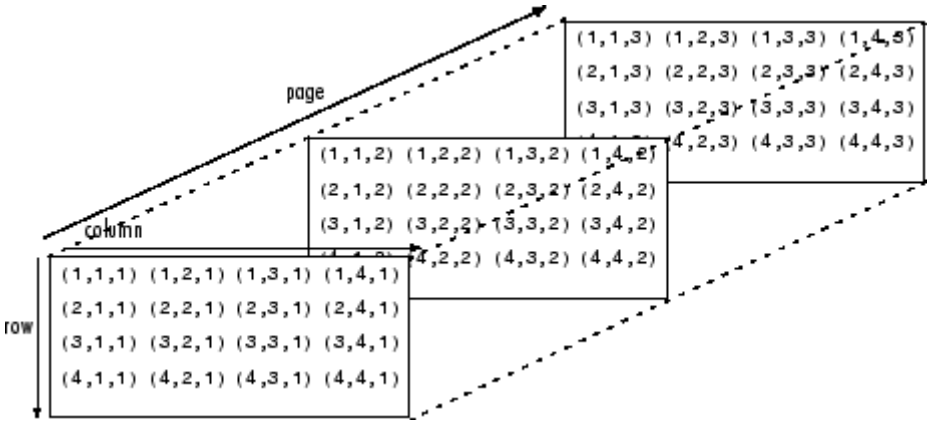


Figure 3.5: Visual representation of a 4D tensor [35]

NDArray is used synonymously with tensor or multi-dimensional arrays for most deep-learning libraries. The dimensionality of a tensor (1,2,3...n) is called its order; a fifth-order tensor will have five dimensions, for instance.

An image's width and height are easily understood. Due to how colors are encoded, the depth is necessary. For instance, RGB encoding produces an image three layers deep. Each layer is called a "channel" and through convolution a stack of feature maps occur in the fourth dimension. Features are really only details of images that convolutional networks build maps of, such as a line or a curve. So, instead of thinking of images as two-dimensional areas, they are handled as four-dimensional areas in convolutional networks [35].

3.3 Convolutional Neural Networks

3.3.1 Convolutional definition

“To convolve” means to roll together - from the Latin convolvere. For mathematical purposes, a convolution is the integral calculation of how much two functions overlap as one crosses over the other. Think of a convolution by multiplying two functions as a means of mixing them.

The visualization below (Figure 3.6) [34] displays the green curve as a function of t, the location shown by the vertical green line, indicating the convolution of the blue and red curves. The gray region shows the g(tau)f(t-tau) product as a function of t, so the convolution is precisely its area as a function of t.

Look at the tall, narrow curve of the bell standing in the center of a graph. The area below the curve is the integral. A second bell curve is near it, which is shorter and wider, slowly drifting from the left side of the graph to the right. At any point along the x-axis, the product of the overlap of those two functions is their convolution. So, the two functions are being "rolled together" in a way.

The static, underlying function (the equivalent of the immobile bell curve) is the input image being analyzed with image analysis, and the second, mobile function is referred to as the filter since it picks up a signal or feature in the image. By multiplication, the two functions are related.

The next thing to remember about convolutional networks is that they pass many filters over a single image, each one picking up a different signal. At a fairly early layer, you can imagine them as passing a horizontal, a vertical and a diagonal line filter to create a map of the image's edges [37].

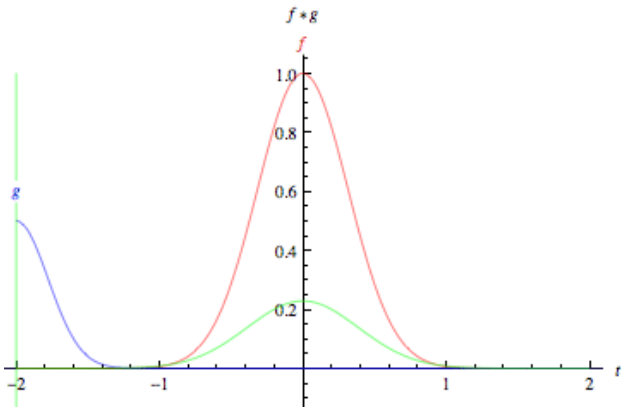


Figure 3.6: A graphical illustration of the convolution of two Gaussians (right) [36]

These filters, slices of the feature space of the image, are taken by convolutional networks and they create a map of each position where the feature occurs. Convolutional networks allow for easily scalable and robust feature engineering by studying various portions of a feature space.

So convolutional networks conduct a kind of search. Imagine a little magnifying glass moving over a big picture from left to right and re-starting from the left when it hits the end of one pass, as typewriters do. Just one thing, a short vertical line, can be recognized by the moving window. Three pixels of darkness stacked atop each other. It moves that vertical line recognizing filter over the pixels of the image, checking for matches.

Each time a match is identified, it is mapped onto a feature space particular to that visual element. The position of each vertical line match is registered in that space, a little like birdwatchers leaving pins on a map to mark where a great blue heron was last seen. Over a single image, a convolutional network runs many, many searches-horizontal ones, diagonal ones, as many as there are visual items to be looked for. After a nonlinear transform such as tanh is passed into a convolutionary layer input, it can squash input values into a range between -1 and 1 [36].

3.3.2 Convolutional Neural Network definition

Convolutional neural networks are deep artificial neural networks that are mainly used to identify photos (e.g., label what they see), cluster them through resemblance (photo search) and conduct object recognition withing scenes. There are algorithms that can recognise faces, people, street signs, tumors, platypuses, and many other visual data elements.

In order to digitize text and make natural language processing possible on analog and handwritten texts, where the images are symbols to be transcribed, convolutional networks perform optical character recognition (OCR). It is also possible to add CNNs to sound when it is physically represented as a spectrogram. More recently, convolutional networks have been extended to apply directly to text analytics as well as graph data with graph convolutional networks.

One of the key reasons why the world has awakened to the effectiveness of deep learning is the usefulness of convolutional nets (ConvNets or CNNs) in image recognition. They are pushing big advancements in computer vision (CV) with simple applications for self-driving vehicles, robots, drones, defense, medical diagnosis and treatment for the visually impaired [38].

3.3.3 How CNNs work

What we need to know about convolutional networks is that they do not interpret images like human beings do. Therefore, when it is fed to and interpreted by a convolutional network, you would have to think in a particular way about what a picture represents. Instead of flat canvases, conventional networks view images as volumes, as three-dimensional objects, to be measured only by width and height. That is because optical color pictures have an RGB encoding, combining the three colors to make the humanly perceivable the color spectrum. A convolutional network ingests such images as three separate layers of color stacked one on top of the other.

A convolutional network, thus, receives a regular color image as a rectangular box, the width and height of which are determined by the number of pixels along the dimensions and the depth of which is three layers deep, one for each RGB letter. These layers of depth are referred to as channels.

We will define them in terms of input and output volumes as images pass through a convolutionary network, representing them mathematically as matrices of multiple dimensions in this form: 30x30x3. Their dimensions change from layer to layer, for reasons that will be discussed below.

As they are the basis of the linear algebra operations used to process images, we need to pay careful attention to the exact measurements of and dimension of the volume of the image.

The intensity of R, G and B will be represented by a number for each pixel of an image, and that number will be an element in one of the three stacked two-dimensional matrices which together form the volume of the image.

These numbers are the original, raw, sensory features that are fed into the convolutional network, and its purpose is to figure out which of those numbers are important signals that actually allow it to more effectively identify images.

A convolutional network takes in square patches of pixels instead of focusing on one pixel at a time and sends them through a filter. The filter is also a square matrix that is smaller than the image and equal in size to the patch. For anyone familiar with support-vector computers, it is often called a kernel, which can ring a bell, and the filter's task is to locate patterns in the pixels.

Visualize two matrices. One of these is 30x30, and the other is 3x3. That is, the filter covers one-hundredth of the surface area of one picture channel.

We are going to take the dot product of the filter with this patch of the image channel. The performance of the dot product would be high if the two matrices have high values in the same positions. If they do not, it is going to be low. In this way, the output of the dot product can tell us if the pixel pattern in the underlying picture fits the pixel pattern represented by our filter, .

Let us assume that a horizontal line is expressed by our filter, with high values along the second row and low values along the first and third rows. Now we start with the image in the upper left corner of the underlying image and pass the filter step by step across the image until it reaches the upper right corner. The size of the step is known as stride. You can move the filter to the right one column at a time, or you can choose to make larger steps.

You take a separate dot product at each step and position the results of that dot product in a third matrix known as an activation map. The width of the activation map, or number of columns, is equal to the number of steps that the filter takes to traverse the underlying image. Because larger steps lead to fewer steps, a big step would create a smaller map of activation. This is important because the size of the matrices that are processed and generated by convolutional networks at each layer is directly proportional to how computationally costly, they are and how much time they take to train. A bigger stride equals less time and compute.

A filter that is superimposed over the first three rows slides over them and then begins from rows 4-6 of the same image again. If it has a stride of 3, it will generate a matrix of 10x10 dot products. All three channels of the underlying image, R, G and B, can be applied to that same filter representing a horizontal line. And it is possible to put together the three 10x10 activation maps, such that the combined activation map for a horizontal line on all three channels of the underlying image is also 10x10.

Now, because images have lines leading to many directions and include many different types of shapes and pixel patterns, in pursuit of those patterns, you would want to slide other filters over the underlying image. For instance, in the pixels, you might look for 96 different patterns. A stack of 96

activation maps will be created by those 96 patterns, resulting in a new volume that is 10x10x96. The input image, kernels and output activation maps that were generated can be seen in Figure 3.7 [35].

A convolution is what we just mentioned. You might think of convolution a s a fancy kind of multiplication used in signal processing. One of the key concerns with photos is that they are high-dimensional, which suggests that processing costs a lot of time and computational resources. Convolutional networks are meant to minimize the dimensionality of images in a number of ways. One approach to minimize dimensionality is by filter stride. Another way is through downsampling.

3.3.4 Max pooling/downsampling with CNNs

There are three names for the next layer of a convolutional network: max pooling, downsampling and subsampling. The activation maps are fed into a downsampling layer, and like convolutions, this process is applied one patch at a time. In this case, max pooling essentially takes the largest value from one image patch, places it next to the max values from other patches in a new matrix, and discards the remainder of the data found in the activation maps [39].

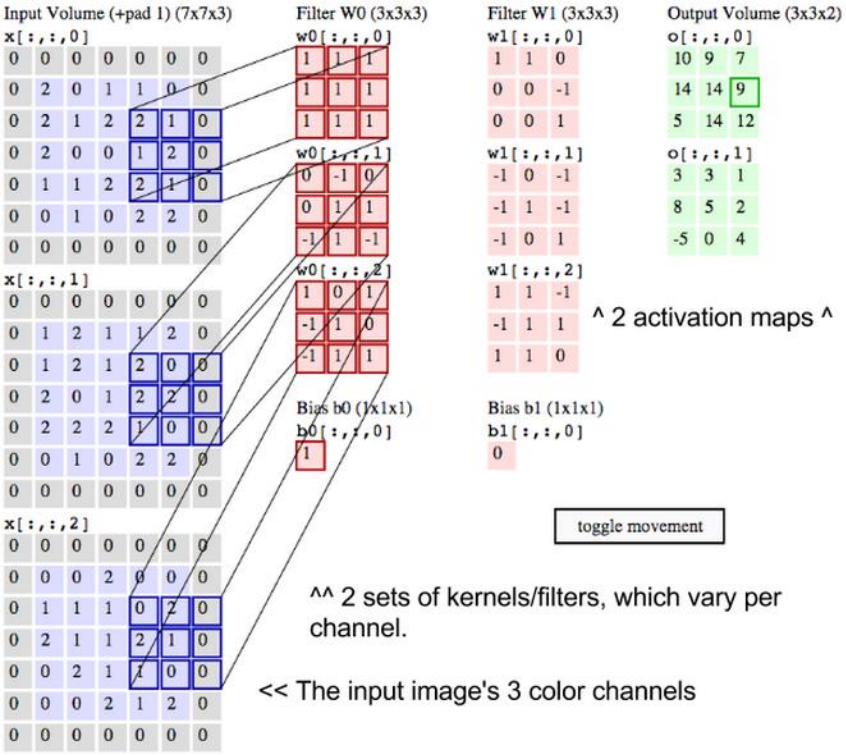


Figure 3.7: Input image, kernels and the output activation maps [35]

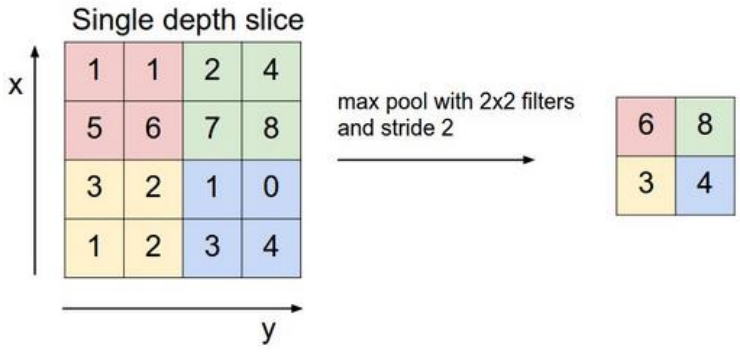


Figure 3.8: New matrix created by max pooling [35]

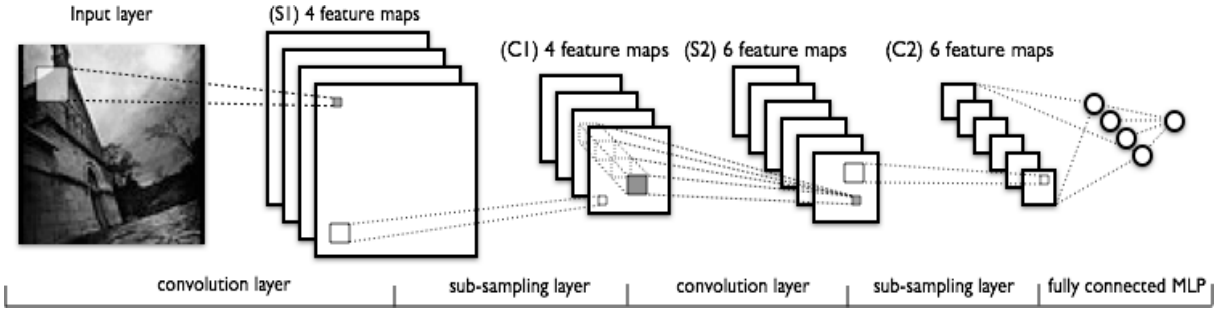


Figure 3.9: Sequence of transformations involved in a typical convolutional network [35]

In this step, a lot of information about lesser values is lost because only the positions on the image that displayed the greatest association with each feature are preserved, and these maximum values combine to form a lower dimensional space. Downsampling, however, has the effect of reducing the amount of storage and computation necessary, precisely because information is lost.

We start from the actual picture in Figure 3.9 [35] that is screened for features. The light small rectangle is the filter which passes over it. Activation maps, one for each filter we use, are stacked above each other. One patch to be down-sampled is the broader rectangle. Moving one we have the activation maps condensed through downsampling. By passing filters over the first down-sampled stack, a new set of activation maps is generated. The second downsampling condenses the second set of activation maps. Ultimately, we have a completely linked layer that categorizes output with one label per node. The patterns processed by the convolution network grow more complex as more and more detail is lost, and become more distant from visual patterns that we recognize as humans. So, forgive yourself and us if, as they grow deeper [40], convolutionary networks do not deliver basic intuitions.

3.4 R-CNN

Object detection aids in vehicle detection, pose estimation, surveillance etc. The distinction between algorithms for object detection and classification algorithms is that we attempt to draw a bounding box around the object of interest to locate it within the image. Even, in an object detection scenario, you would not actually draw only one bounding box, there might be many bounding boxes inside the picture representing multiple objects of interest, and you might not know how many beforehand.

The major reason why you cannot proceed with this problem by building a standard convolutional network followed by a fully connected layer is that the length of the output layer is variable. This is because the number of occurrences of the objects of interest is not fixed. A naive approach to solve this problem would be to take different regions of interest from the image and use a CNN to classify the presence of the object within that region. The problem with this approach is that the objects of interest might have different spatial locations within the image and different aspect ratios. Therefore, a large number of regions will have to be chosen and everything could blow up computationally. Algorithms such as R-CNN, SSD, YOLO and others have therefore been established to locate these occurrences and identify them quickly [41].

Ross Girshick et al [42] recommended an approach where we use Selective Search to extract only 2000 regions from the picture to circumvent the issue of choosing a large number of regions, naming them region proposals. Thus, now you should only deal for 2000 regions instead of having to identify a massive number of regions. These 2000 region proposals are generated using the Selective Search algorithm.

The Selective Search algorithm works as follows:

1. Generate initial sub-segmentation, by creating many candidate regions.
2. Use greedy algorithms to recursively combine similar regions into larger ones.
3. Use the generated regions to produce the final candidate region proposals.

These proposals for the 2000 candidate area are twisted into a square and fed into a convolutionary neural network that generates a 4096-dimensional output feature vector. The CNN works as a feature extractor and the dense layer of output consists of the features extracted from the image and the features extracted are fed into an SVM to classify the object's existence within that candidate region proposal. In addition to predicting the existence of an object within the proposals for the region, the algorithm also predicts four values to improve the accuracy of the bounding box, which are offset values. The algorithm may have estimated the appearance of a person, for example, provided a region proposal, but the face of that person inside that region proposal may have been cut in half. The offset values thus assist in adjusting the bounding box of the region proposal.

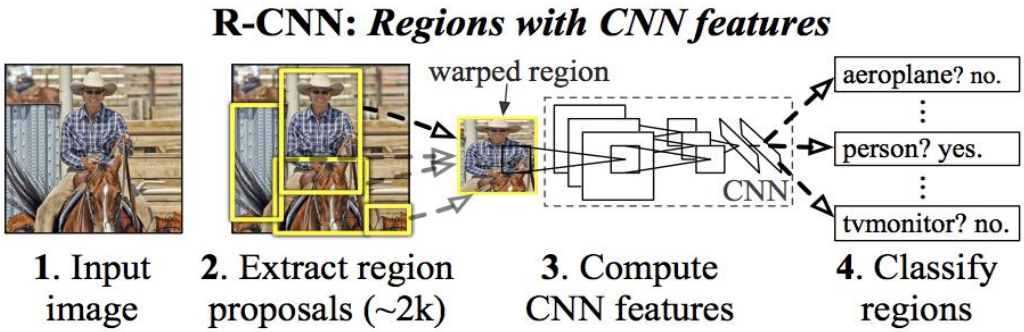


Figure 3.10: Object detection system overview [43]

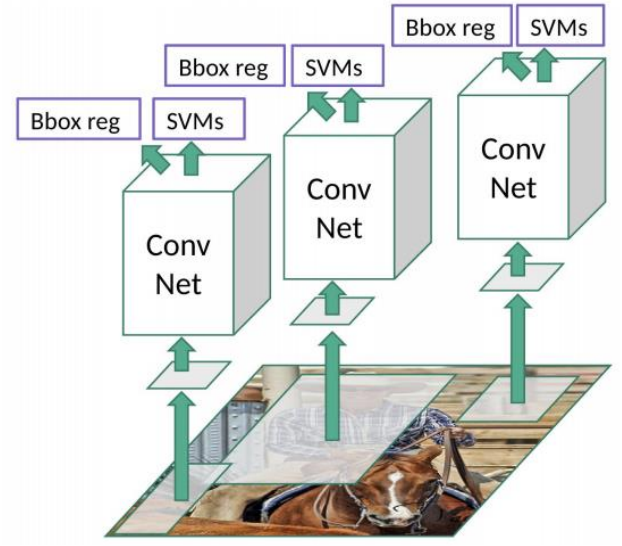


Figure 3.11: Candidate region proposal fed into different CNNs [44]

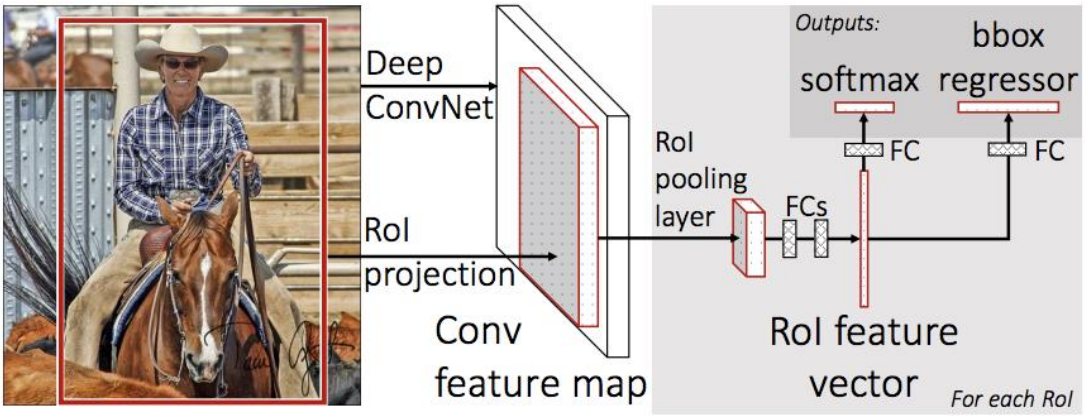


Figure 3.12: Fast R-CNN architecture [44]

3.4.1 Fast R-CNN

R-CNN comes with some disadvantages, of course. It takes a massive amount of time to train the network first and foremost, as we will have to identify 2000 area proposals per picture. Secondly, since it takes about 47 seconds for each test image, it cannot be applied in real time. The Selective Search algorithm is, last but not least, a fixed algorithm. Therefore, at that point, no learning is taking place, which could lead to the production of bad proposals for candidate regions.

Any of R-CNN's limitations were overcome by creating a faster target detection algorithm. The same author of the R-CNN paper accomplished this, and it was renamed to 'Fast R-CNN'. The method is similar to the algorithm for R-CNN. But we feed the input picture to the CNN to create a convolution feature map instead of feeding the region proposals to the CNN. From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer. We use a softmax layer from the RoI function vector to predict the class of the proposed area and also the bounding box offset values [45].

The reason why "Fast R-CNN" is better than R-CNN is that you do not have to feed 2000 area suggestions to the convolutionary neural network. Instead, only once per image is the convolution procedure completed and a feature map is created from it.

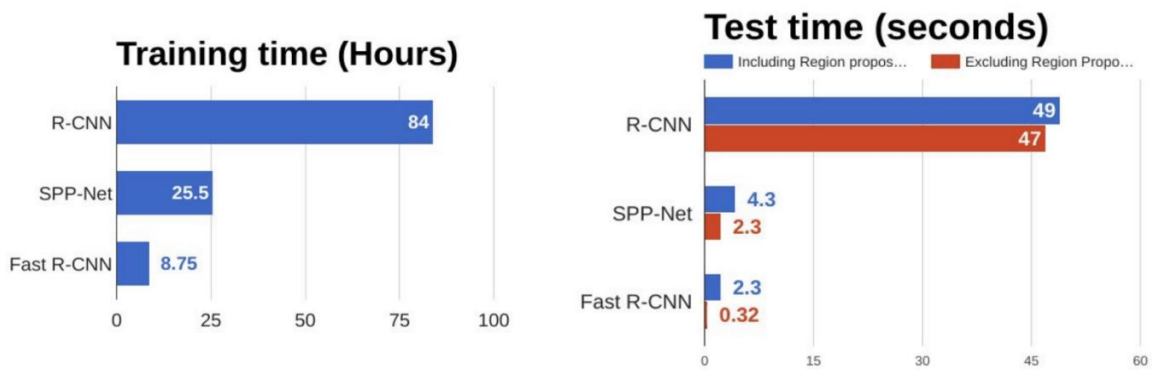


Figure 3.13: Comparison of object detection algorithms [44]

You can conclude from the above graphs that in training and trial sessions over R-CNN, Fast R-CNN is considerably faster. When you look at the efficiency of Fast R-CNN during test time, including region proposals slows down the algorithm a lot when compared to not using region proposals. This means that regional proposals are now becoming bottlenecks in the Fast R-CNN algorithm that impact its performance [43].

3.4.2 Faster R-CNN

Selective Search is used by each of the above algorithms to figure out the region proposals. Selective Search is a slow and time-consuming process that impacts the network's efficiency. Therefore, an object detection algorithm was developed by Shaoqing Ren et al. that avoids the Selective Search algorithm and enables the network to learn the region proposals.

The image is given as an input to a convolutional network that provides a convolutional feature map, similar to Fast R-CNN. A different network is used to predict the regional proposals instead of using the Selective Search algorithm on the feature map to identify the regional proposals. Using a RoI pooling layer that is then used to classify the picture within the proposed region and estimate the offset values for the bounding boxes [43], the predicted region proposals are then reshaped.

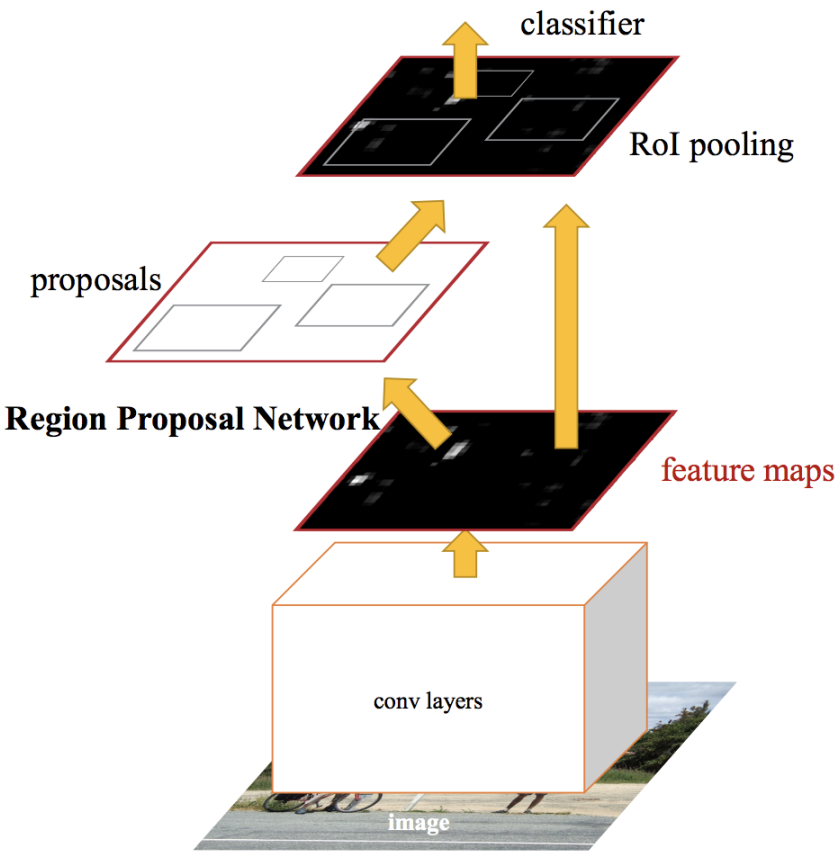


Figure 3.14: Faster R-CNN architecture [44]

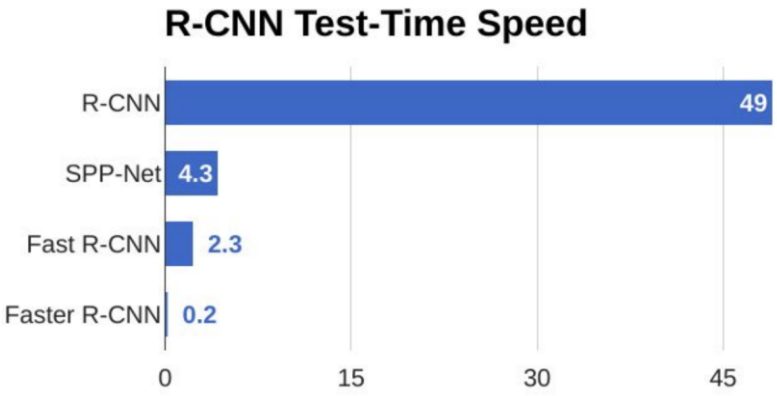


Figure 3.15: Comparison of R-CNN object detection algorithms [44]

3.5 Single Shot multibox Detector - SSD

The Single Shot Multibox Detector (SSD), which uses a single convolutionary neural network to detect an object in an image, is one of the fastest algorithms for current events in the object detection field. SSD is an algorithm for object detection that is a VGG16 architecture modification. It was released at the end of November 2016 and has hit new milestones for object detection tasks in terms of efficiency and accuracy, scoring over 74 percent mAP (mean Average Precision) on standardized datasets such as PascalVOC and COCO at 59 frames per second.

Let us begin by illustrating where the name of this architecture comes from to better understand SSD:

`Single Shot:` this means that the tasks of object localization and classification are done in a single forward pass of the network.

`MultiBox:` The name of a technique for bounding box regression developed by Szegedy et al.

`Detector:` The network is both an object detector and a classifier of those detected objects.

3.5.1 Architecture

The architecture of SSD builds on the venerable architecture of VGG16 but discards the fully connected layers. Due to its strong performance in high-quality image classification tasks and its prominence for issues where transfer learning aims to enhance results, it is the reason VGG-16 was chosen as the base network.

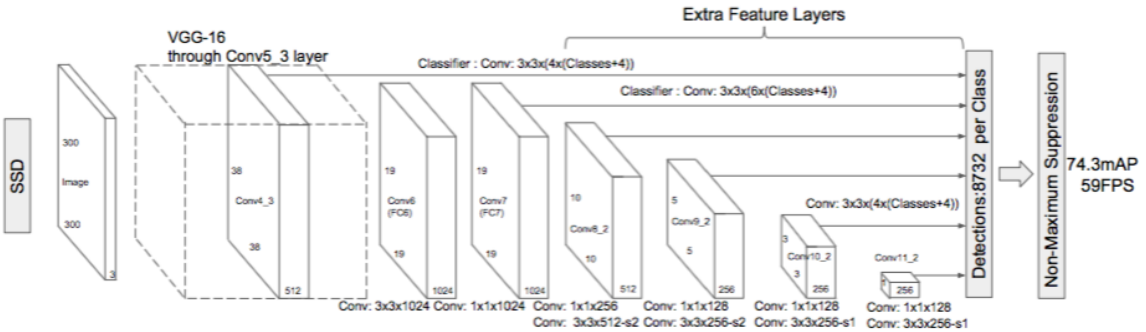


Figure 3.16: Architecture of SSD [45]

A set of auxiliary convolution layers was introduced (from conv6 and onwards) instead of the initial VGG fully connected layers, allowing the extraction of features on multiple scales and progressively reducing the input size for each subsequent layer (Figure 3.17) [46].

3.5.2 MultiBox

Szegedy's work on MultiBox, a tool for fast class-agnostic bounding box coordination proposals, inspired the bounding box regression technique of SSD. Interestingly, an Inception-style convolutional network is used in the work conducted on MultiBox. "The 1x1 convolutions seen below (Figure 3.18) [47] help in minimizing dimensionality as the number of dimensions decreases (but "width" and "height" remain the same) [47].

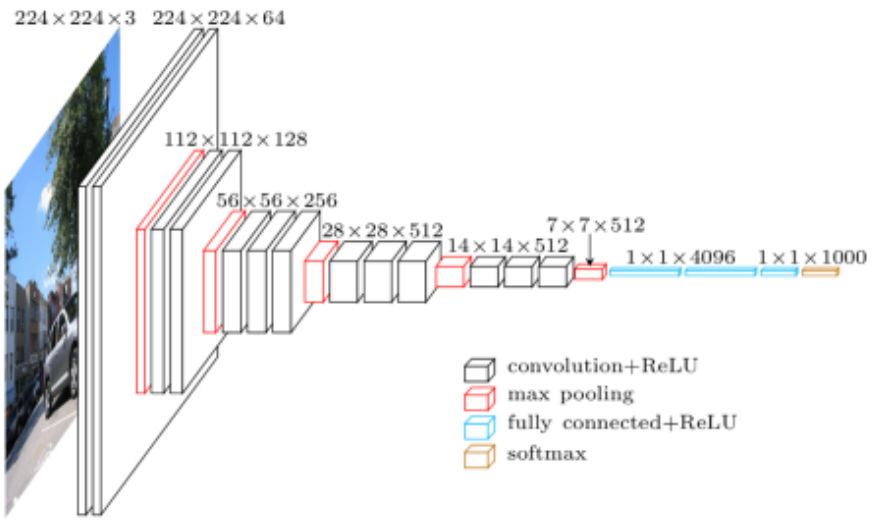


Figure 3.17: VGG architecture [46]

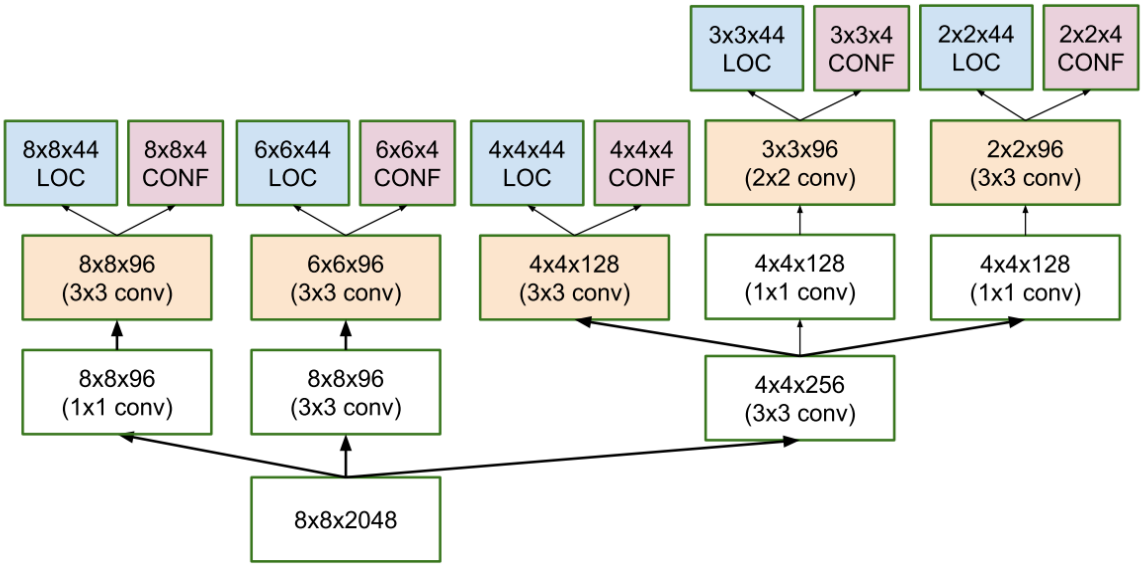


Figure 3.18: Architecture of multi-scale convolutional prediction of location & confidences of multibox [47]

The loss feature of MultiBox also combined two crucial elements that found their way into SSD:

- `Confidence Loss:` this measure how confident the network is of the objectness of the computed bounding box. Categorical cross-entropy is used to compute this loss.
- `Location Loss:` this measure how far away the network's predicted bounding boxes are from the ground truth ones from the training set. L2-Norm is used here.

The term for loss, which calculates how far from our prediction "has landed", without delving too deeply into the math, is thus:

```
multibox_loss = confidence_loss + alpha * location_loss
```

The alpha term allows one to balance the contribution of the loss of location. The aim is, as usual in deep learning, to find the parameter values that reduce the loss function most optimally, taking our forecasts closer to the ground truth [47].

3.6 You Only Look Once - YOLO

3.6.1 Introduction

YOLO is a real-time algorithm for object detection, which is also one of the most powerful algorithms for detecting objects and still incorporates many of the most groundbreaking concepts coming from the scientific field of computer vision. YOLO came to the computer vision scene with Joseph Redmon et al by the groundbreaking 2015 paper "You Only Look Once: Unified, Real-Time Object Detection," which gained a lot of support immediately from fellow computer vision researchers [48].

Detection of objects is one of the classical computer vision issues in which you work to recognize what objects are in a given image and where they are in the image. The issue of object detection is more difficult than classification, which often allows objects to be identified but does not imply where the object is in the picture. In addition, classification does not operate on images involving more than one object.

YOLO is prominent because, while still being able to run in real time, it achieves high accuracy. In the sense that it takes only one forward propagation trip through the neural network to make predictions, the algorithm 'only looks once' at the picture. After non-max suppression, which means that any object is only identified once by the object detection algorithm, it then outputs known objects along with the bounding boxes around them.

With YOLO, for those boxes, a single CNN simultaneously forecasts several bounding boxes and class probabilities. YOLO trains on full images and directly optimizes the performance of detection. This model is incredibly fast. It can process images at 45 frames per second in real time and can process up to 155 frames per second with a smaller version called 'Quick YOLO'. A distinctive aspect is that during preparation and test time, YOLO sees the whole picture so that it implicitly encodes contextual details about classes as well as their appearance [48].

The technique includes a single deep convolutionary neural network (originally a GoogLeNet version, later revised and renamed DarkNet based on VGG) that divides the input into cell grids, and each cell directly predicts a bounding box and object classification . The outcome is a huge number of bounding boxes for candidates which are consolidated by a post-processing stage into a final prediction.

Since the algorithm's first appearance in 2015, further development has been carried out, leading to further versions being published. YOLO9000, also known as YOLOv2, was released in December

2016, YOLOv3 was released in April 2018, and YOLOv4 was eventually released in April 2020 as the new version currently available. The first version proposed the general architecture, while the second version improved the concept and used predefined anchor boxes to enhance the proposal for bounding boxes. Also, model architecture and training phase was further refined in version three [49].

Although the model accuracy is similar to but not as good as Region-Based Convolutional Neural Networks (R-CNNs), because of their detection speed, they are popular for object detection, frequently demonstrated on video or with camera feed input in real time [49].

3.6.2 High Level Overview

Compared to other region proposal classification networks (fast RCNN) that perform detection on various region proposals meaning that they end up performing prediction several times for various regions in the image, Yolo's architecture is more like an FCNN (fully convolutional neural network) that passes the image (nxn) once through the FCNN and the output is a (mxm) prediction. This design divides the input image into a mxm grid and 2 bounding boxes and class probabilities for such bounding boxes for each grid generation. Notice that it is more likely that the bounding box is wider than the grid itself.

The object detection is refrained as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. As we see it the paper.

For such boxes, a single convolutional network simultaneously predicts several bounding boxes and class probabilities. YOLO trains on full images and directly optimizes the efficiency of detection. There are some advantages of this unified model over conventional object detection methods. Firstly, YOLO is incredibly fast. Since detection is framed as a problem of regression, a complex pipeline is not required. At test time, the neural network works on a new image to forecast detections. On a Titan X GPU, the base network runs at 45 frames per second without batch processing and a fast version runs at over 150 fps. This suggests that, with less than 25 milliseconds of latency, it can process streaming video in real time [50].

Second, while making forecasts, YOLO examines the picture globally. YOLO sees the entire picture at training and test time, unlike sliding window and region proposal-based methods, so it encodes contextual information about classes as well as their presence implicitly. In a picture for objects, Fast R-CNN, a top detection method, mistakes background patches because it cannot see the wider meaning. In contrast to Fast R-CNN, YOLO makes less than half the number of background mistakes.

Third, generalizable object representations are learned by YOLO. YOLO outperforms top detection methods like DPM and R-CNN by a wide margin when trained on natural images and tested on artwork. Since YOLO is strongly generalizable, when introduced to new domains or unpredictable inputs, it is less likely to break down.

In order to predict each bounding box, the network uses features from the entire image. It also forecasts all bounding boxes for a picture simultaneously for all classes. This means that the network is globally aware of the entire image and all the objects in the image. The architecture of YOLO allows for end-to-end training and real-time speeds while preserving high average accuracy.

The system separates the image of the input into a S × S grid. If an object's center falls into a grid cell, the identification of that object is the responsibility of that grid cell [50].

Bounding boxes and confidence scores for those boxes are predicted by each grid cell. These confidence scores show how sure the model is that an item is included in the box and also how precise the box believes it predicts. We formally define confidence as Pr(Object)/IOU . The confidence scores should be zero if no object appears in that cell. Otherwise, we want the confidence score to match the intersection over union (IOU) between the predicted box and the ground truth.

Each bounding box is made up of 5 predictions: x, y, w, h, and confidence. Relative to the boundaries of the grid cell, the (x, y) coordinates represent the center of the box. The width and height are predicted relative to the whole picture. Finally, between the predicted box and any ground truth box, the confidence prediction reflects the IOU. Each grid cell, Pr(Classi |Object), also predicts C conditional class probabilities. On the grid cell containing an object, these probabilities are conditioned. No matter the number of boxes B, we only expect one set of class probabilities per grid cell.

The conditional class probabilities and the individual box trust projections are multiplied at the test time,

```
Pr(Classi|Object)*Pr(Object)*IOU = Pr(Classi)*IOU
```

, that provides us with class-specific confidence scores for each box. Both the probability of the class appearing in the box and how well the predicted box matches the object are represented by these scores [50].

3.6.3 How it works

First, we need to consider what is currently being predicted in order to understand the YOLO algorithm. Ultimately, we try to predict an object class and the bounding box that determines the position of the object. Each bounding box can be described using four descriptors:

- Center of the box (*bx, by*)
- Width (*bw*)
- Height (*bh*)
- Value *c* corresponding to the class of an object.

Along, we estimate a real pc number, which is the probability that an object is present in the bounding box.

In an input image that may contain an object, YOLO does not scan for interested regions, but instead divides the image into cells, usually a 19x19 grid. Each cell is then in charge of predicting K bounding boxes [50].

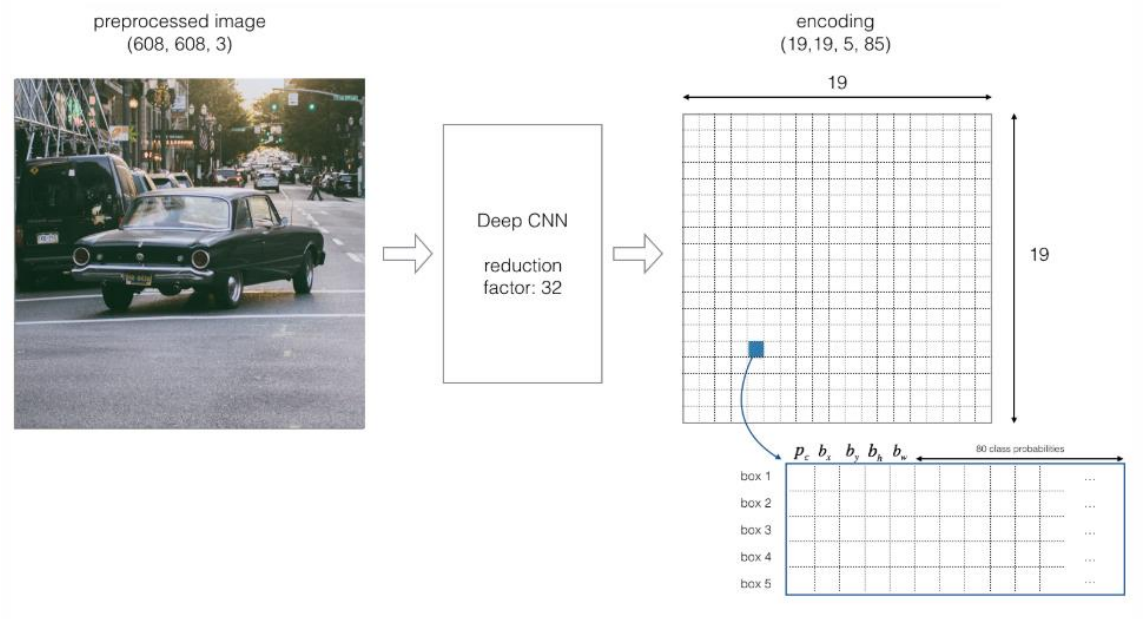


Figure 3.19: K=5 to predict possibility for 80 classes [51]

An object is only considered to lie in a certain cell if the anchor box's center co-ordinates are located in that cell. The center coordinates are still calculated relative to the cell because of this property, while height and width are calculated relative to the entire image dimension.

YOLO calculates the probability that the cell includes a certain class during the one pass of forward propagation. The equation for the same is shown in figure 3.20 [51].

The class with the maximum probability is picked and assigned to that specific cell of the grid. For all the grid cells present in the image, a similar process occurs. The class estimate is visualized in Figure 3.21 [51].

$$score_{c,i} = p_c \times c_i$$

Figure 3.20: Probability that there is an object of certain class ‘c’ [51]

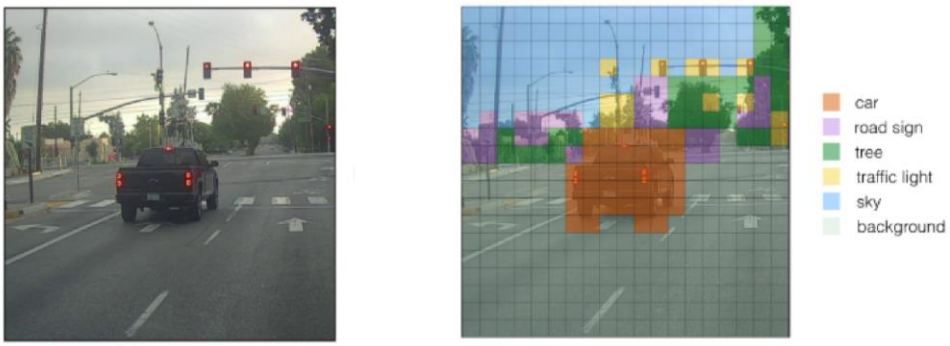


Figure 3.21: Before and after predicting the class probabilities for each grid cell [51]

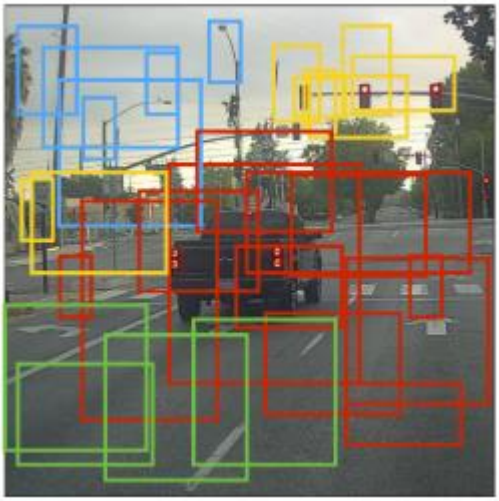


Figure 3.22: Anchor boxes [52]

The next step is Non-max suppression after predicting the class probabilities, which allows the algorithm to get rid of the unnecessary anchor boxes, as you can see that there are multiple anchor boxes determined based on the class probabilities in the figure 3.24 [51] below.

Non-max suppression excludes the bounding boxes that are very close by preforming the IoU (Intersection over Union) to solve this problem and keeps the one with the highest-class probability among them [51].

For all bounding boxes, it measures the IoU value proportional to the one with the largest class probability, then excludes the bounding boxes whose IoU value is greater than the threshold. It implies that the same object is covered by such two bounding boxes, but the other has a low probability for the same object, so it is eliminated.

Upon completed, the algorithm finds the next highest-class probability bunding box and performs the same process, until we are left with all the different bounding boxes.

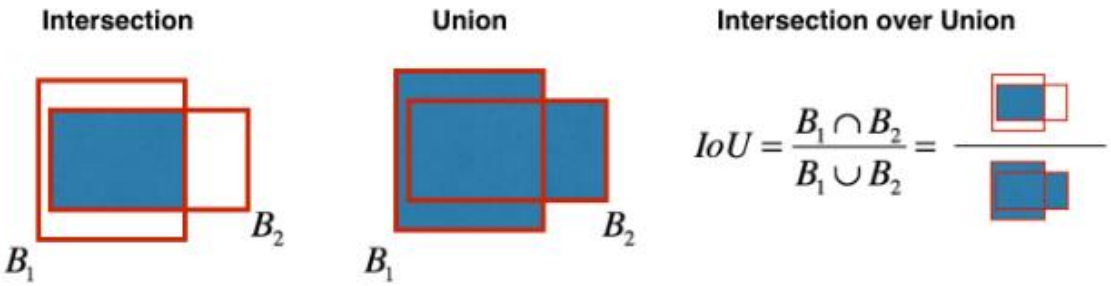


Figure 3.23: IoU operation [51]

Figure 3.24: Before and After Non-max suppression [51]

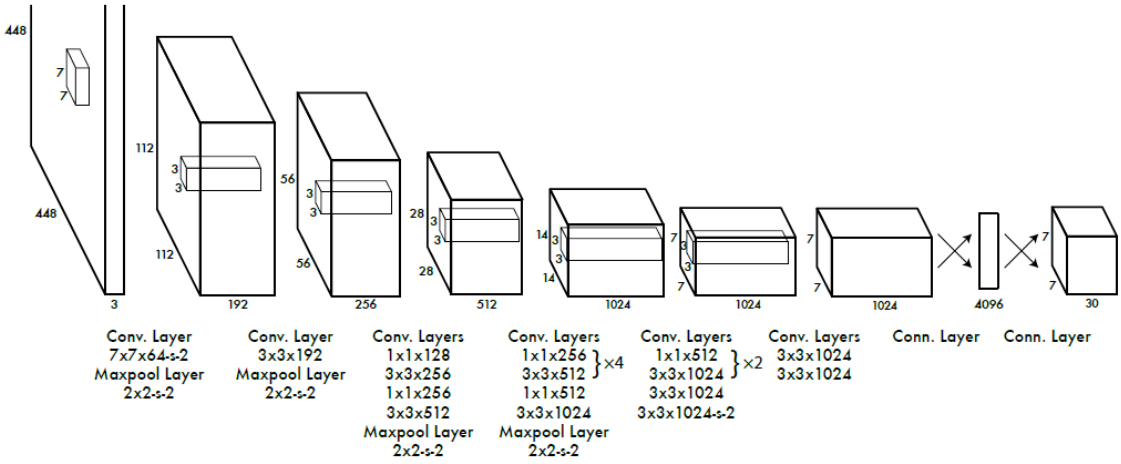


Figure 3.25: CNN architecture of YOLO, inspired by GoogleNet model image classifier [51]

After this, almost all the work is completed, the algorithm finally generates the necessary vector showing the information of the respective class bounding box. The overall algorithm architecture can be viewed in Figure 3.25 [51].

Also, the algorithm's most important parameter, its loss function, as seen below (Figure 3.26) [51]. YOLO learns about all the four parameters it predicts at the same time (discussed above).

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

Figure 3.26: Loss function for Yolo [51]

Figure 3.27: YOLO output example [52]

3.7 Summary

It is apparent that certain algorithms, such as R-CNN, have struggled to fulfill our requirements and offer us real-time performance when taking all the above into account and still taking into consideration our specific use case, which is to construct a real-time object detector. In addition, R-CNN algorithms from Fast & Faster are still not good enough. That could be attributed largely to the use of a fixed algorithm, the Selective Search algorithm. Subsequently, our only options remain SSD and YOLO. As shown in Figure 3.28 [54], which provides an overview of the comparison of object detection algorithms, both SSD and its variants are more effective than R-CNN, but not as powerful as YOLO.

The undeniable assets of YOLO, its real-time capability and precision, have therefore led us to proceed with our implementation using YOLO as our chosen algorithm. Last but not least, it should be explained that we will use the new edition of YOLO, which is version 4, as seen in Figure 3.29 [54], which gives the highest precision for the 60 frames per second that we want.

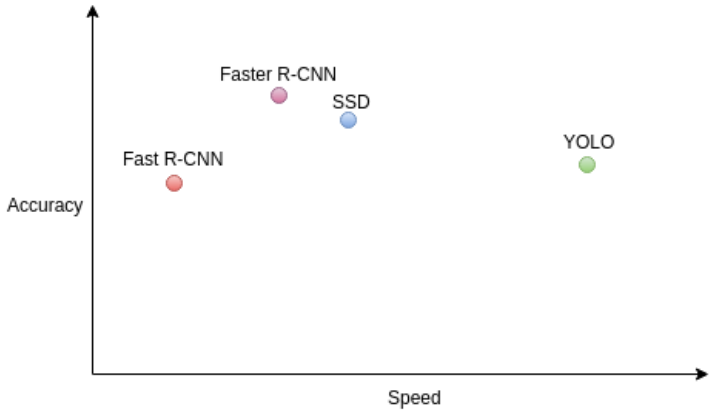


Figure 3.28: Object detection algorithms comparison [53]

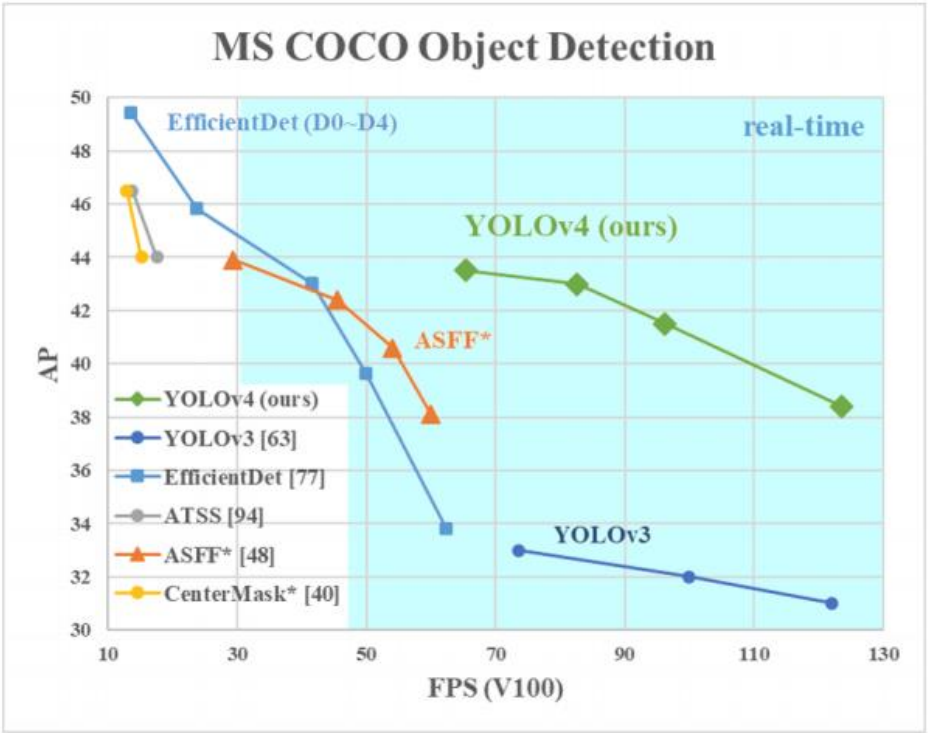


Figure 3.29: YOLOv4 achieves 43.5% AP accuracy in MS COCO dataset with 65 FPS inference speed [54]

4 Tools and implementation

4.1 Choosing between cloud or local environment

When working with AI and ML projects the choice is either to work in a cloud environment or to set up a local environment in which all the training and detection will take place.

On the one hand, working in a local environment has the advantage of being able to work offline without the need of having Internet connection available all the time. Therefore, the important benefit is to be able to work from anywhere. On the other hand, setting up a proper local environment to work on ML projects requires expensive hardware. In particular, the multi-threaded architecture of CUDA enabled graphics cards are necessary to meet the demands of underlying complex calculations needed by the majority of ML algorithms.

As for the cloud, even though in some cases a small budget may be required depending on the specific use case, it is significantly simpler to set up. Moreover, no matter whether we are a Windows, MacOS and/or Linux user as long as we have access to a good quality Internet connection and a browser, we can work on our ML projects independently of the operating system we use. Last but not least, working in a cloud environment does not rely on our hardware. In other words, there is no need for expensive graphics cards in order to be productive and efficient when training and using our algorithms.

For the purpose of this B.Sc. thesis, we decided to utilize a cloud solution mainly for two reasons. Firstly, it was ideal to get started on the practical side of our project and not waste valuable time setting everything up. Secondly, using cloud services saved us time from training our ML models and also enabled us to work on other aspects of our project while training took place in the cloud.

4.2 Tools used for our implementation

4.2.1 Google Colab

Google Colaboratory also known as Google "Colab" is a free Jupyter notebook environment that requires no setup and runs entirely (writing, running, & sharing code) on the cloud. It actually allows you to write and execute Python in your browser, with no upfront configuration required, free access to powerful GPUs and of course easy sharing [55].

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, ML, and much more [56].

Our decision for using Google Colab is based on its seamless integration with Google services. The key advantage is that we can store our training dataset (images) in Google Drive and then easily connect it to our Colab notebook to do our model training. It can also be the place where we store the training output, the weights of the trained network, which are used when running our real time detections on our videos.

4.2.2 Python

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. All these properties of Python make it the first choice for ML projects. From development to implementation and maintenance, Python is helping developers to be productive and confident about the software they are developing.

We next report some of the many benefits of using Python in ML.

(a) Simple and consistent

Python is a simple, general-purpose language offering reliable code, that helps to work quickly and efficiently. ML is all about complicated algorithms and versatile workflows. Thus, the simplicity of Python helps the developers to deal with the complex algorithms. Moreover, it saves the time of developers as they only require concentrating on solving the ML problems rather than focusing on the technicality of language.

(b) Flexibility

Python is known as the most flexible language in ML. It provides various options for users. The flexibility factor reduces the possibility of errors. It also allows the programmers to take the situation completely under control and work in a comfort way.

(c) Libraries and framework

Developers require a well-structured and well-tested environment to develop the best coding solutions. ML algorithms are very complex, but Python provides a good solution with an extensive range of libraries and frameworks as follows:

- Keras, TensorFlow, and Scikit-learn for ML
- SciPy for advanced computing
- Seaborn for data visualization
- Pandas for general-purpose data analysis
- NumPy for scientific computing and data analysis

The above solutions reduce the time and help you to develop the product faster. Moreover, the developers can use the existing libraries to implement necessary features.

(d) Readability

Python is easy to read and thus, the developers can easily understand the code. Moreover, they can make changes in it to meet their requirements. Moreover, the possibility of errors, confusion, and conflictions is almost negligible. In this way, the professionals efficiently exchange ideas, tools, and algorithms with others. These features make it more user-friendly and understandable.

(e) Platform independence

Platform independence reflects the versatility of a programming language. It refers to the framework or programming language that allows the developer to implement things on one working environment and use them on another.

Python is a platform-independent language. It is supported by many platforms, including Windows, MacOS and Linux. Moreover, these systems will not require a Python interpreter to use or execute the code.

4.3 Steps for running YOLOv4 in the cloud

Below you can find the steps for running YOLOv4 in a cloud environment.

1. We open Google colab at https://colab.research.google.com and click on the "Sign in" button on the top right corner.
2. We then click on "File" from the menu on the top and then select "New Notebook" (a new tab will open with an empty Google Colab notebook).
3. Let us first enable GPU acceleration within our newly created Colab notebook so that our YOLOv4 will be able to process detections over 100 times faster than with CPU. To do so, we click on "Edit" and then select "Notebook settings". Under the "Hardware accelerator" we select "GPU" from the dropdown menu and hit the "SAVE" button.
4. We clone the darknet framework from AlexeyAB's repository (Figure 4.1)

```
!git clone https://github.com/AlexeyAB/darknet
```

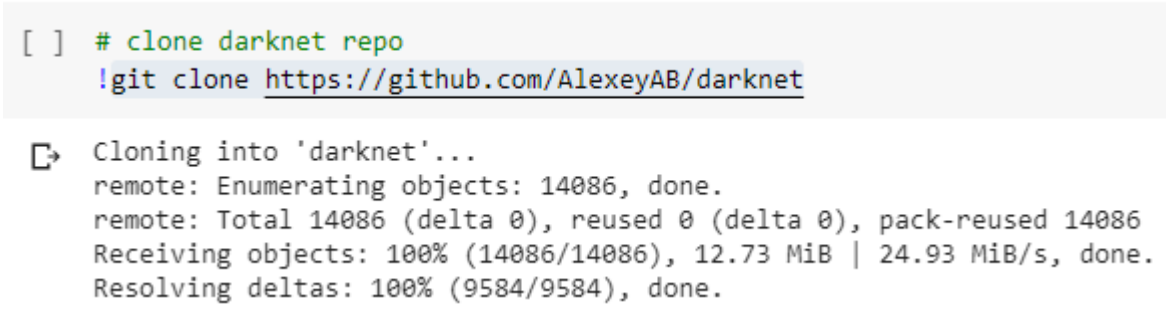


Figure 4.1: Output of the git clone command

5. We adjust the Makefile to enable OPENCV and GPU for darknet. (Figure 4.2)

```
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
```

```
[ ]  # change makefile to have GPU and OPENCV enabled
     %cd darknet
     !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
     !sed -i 's/GPU=0/GPU=1/' Makefile
     !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
     !sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile

     /content/darknet
```

Figure 4.2: Output of running above commands

6. One optional step is to verify that we have CUDA installed although Google Colab has it pre-installed by default. Running the command below will return the installed CUDA version. (Figure 4.3)

   ```
   !/usr/local/cuda/bin/nvcc --version
   ```

```
[ ]  # verify CUDA
     !/usr/local/cuda/bin/nvcc --version

     nvcc: NVIDIA (R) Cuda compiler driver
     Copyright (c) 2005-2019 NVIDIA Corporation
     Built on Sun_Jul_28_19:07:16_PDT_2019
     Cuda compilation tools, release 10.1, V10.1.243
```

Figure 4.3: Output when running the version command of CUDA

7. Now we need to build darknet framework so that we can then use the darknet executable file to run or train object detectors. We do not have to worry about any warnings when we run the `make` command. (Figure 4.4)

   ```
   !make
   ```

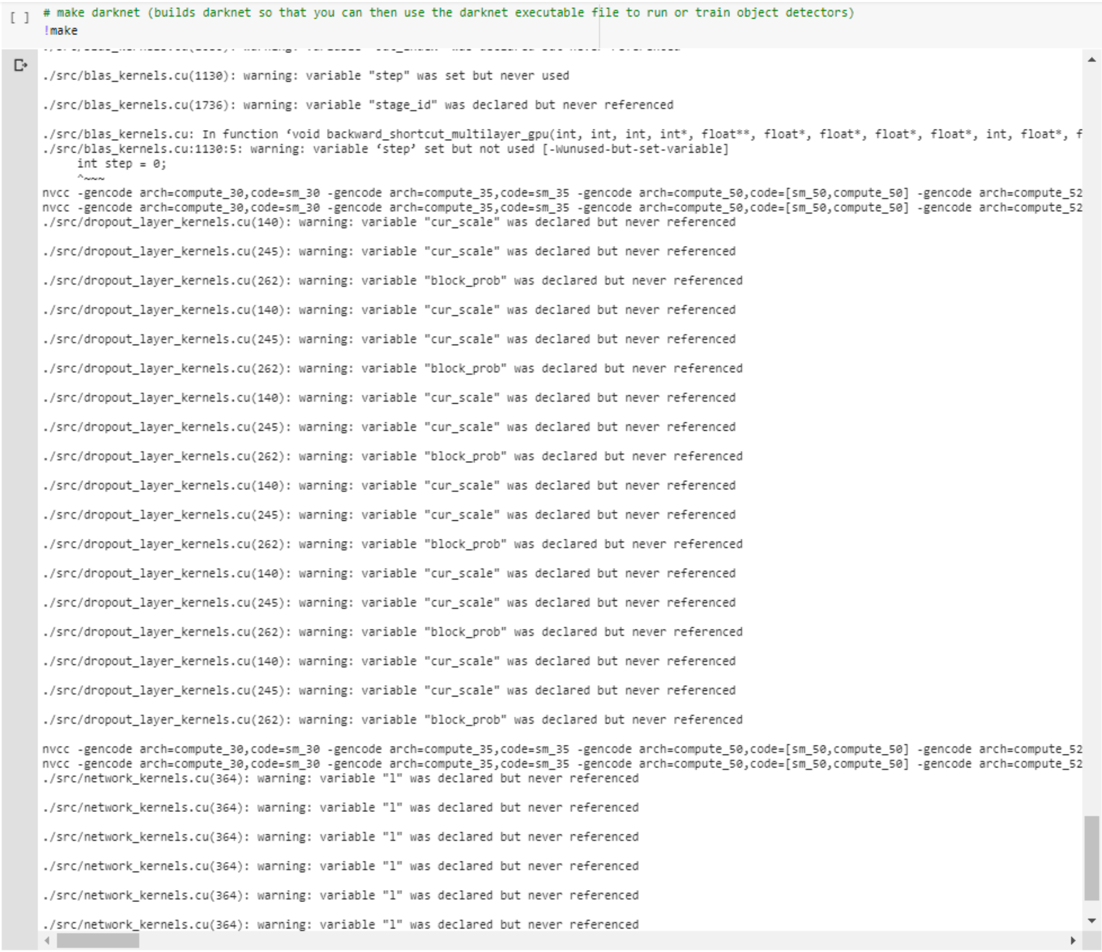


Figure 4.4: Building darknet framework with make command

8. The benefit of YOLOv4 is that it has already been trained on the coco dataset which has 80 classes that it can predict. This makes it ready to use right out of the box. We will grab these pretrained weights so that we can run YOLOv4 with these pretrained classes and get detections. (Figure 4.5)

   !wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights

Figure 4.5: Output of running `wget` command

9. We will then define three helper functions that will allow us to show the image in our Colab notebook after running our detections, as well as upload and download images to and from your cloud VM. "`imShow`" function is used for opening a given image in the cloud and is needed because the default image opening functionality provided by darknet does not work well in Colab.

```
def imShow(path):
  import cv2
  import matplotlib.pyplot as plt
  %matplotlib inline

  image = cv2.imread(path)
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height),
interpolation = cv2.INTER_CUBIC)

  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()


def download(path):
  from google.colab import files
  files.download(path)
```

10. We then need to mount our Google Drive into our cloud VM in order to be able to access its contents, meaning the images we want to run detection on, directly from there. Achieving this connection is easy in Colab. We will use the native "`google.colab`" library given in our cloud environment. Running "`drive.mount`" will give us a URL that we need to visit in order to get an Authorization code for accessing our Google Drive. We just have to copy and paste the code in the appropriate area with the prompt "`Enter your authorization code`" below the given URL and press "`Enter`". (Figure 4.6)

```
%cd ..
from google.colab import drive
```

```
drive.mount('/content/gdrive')
```

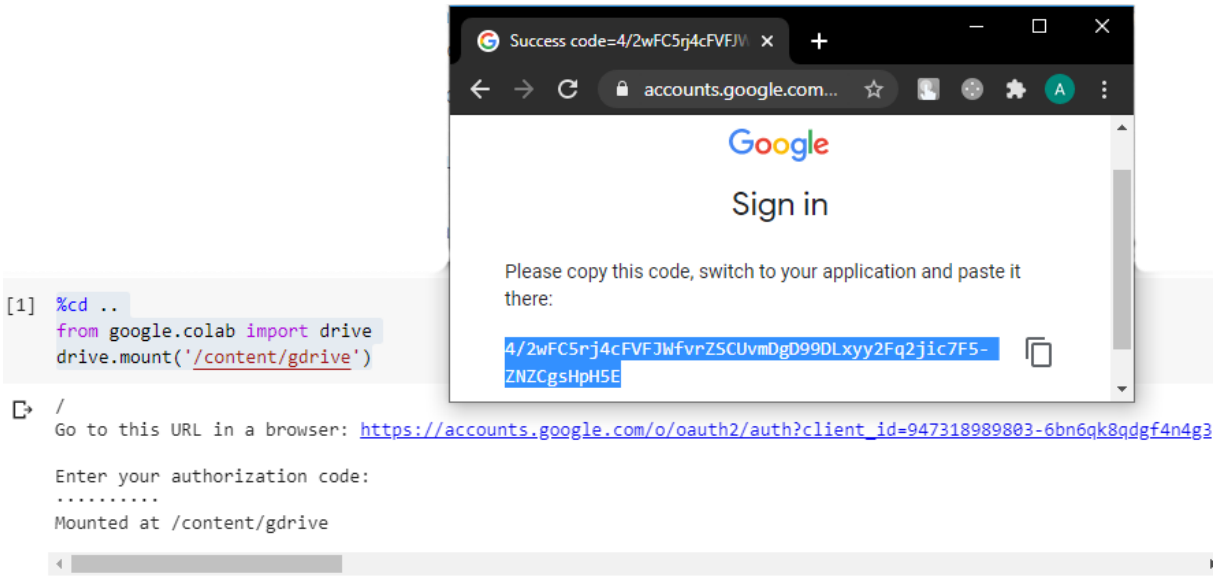


Figure 4.6: Connecting Google Drive with Colab

11. It is good practice to create a symbolic link between "`/content/gdrive/My\ Drive/`" and "`/mydrive`". This means we create a shortcut "`/mydrive`" to map to the contents within the folder "`/content/gdrive/My\ Drive/`". We should do this in order to prevent issues arising when running certain commands if we have the space in the "`My Drive`" folder path.

    ```
    !ln -s /content/gdrive/My\ Drive/ /mydrive
    ```

    ```
    !ls /mydrive
    ```

12. Now it is time for our first detection. We write "`Cd`" and go back into the darknet folder.

    ```
    %cd darknet
    ```

13. We then run detection on any image within our Google Drive. We just have to specify the image's path as the last parameter.

    ```
    !./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights /mydrive/images/street.jpg
    ```

14. Using the "`imShow`" helper function declared in the previous steps we can see the detection's results. Note that "`predictions.jpg`" is the default output name of our detection algorithm. (Figure 4.7)

    ```
    imShow('predictions.jpg')
    ```

```
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights /mydrive/images/street.jpg
imShow('predictions.jpg')
/mydrive/images/street.jpg: Predicted in 32.884000 milli-seconds.
traffic light: 56%
handbag: 69%
person: 30%
person: 55%
person: 80%
person: 99%
person: 56%
person: 96%
traffic light: 37%
person: 28%
person: 48%
handbag: 67%
handbag: 30%
bus: 87%
person: 99%
traffic light: 51%
handbag: 29%
person: 97%
traffic light: 75%
person: 90%
handbag: 29%
person: 98%
person: 93%
car: 84%
person: 99%
cell phone: 62%
handbag: 26%
handbag: 41%
car: 97%
Unable to init server: Could not connect: Connection refused

(predictions:1736): Gtk-WARNING **: 17:48:55.997: cannot open display:
```

Figure 4.7: Running detector and showing the results

15. If we want to save the detection result on our Google Drive we just run:

    ```
    !cp predictions.jpg /mydrive/
    ```

    Or if we want, we could download the predictions.jpg file using the "`download()`" helper function by running:

    ```
    download('predictions.jpg')
    ```

16. Until now everything we did was to use the YOLOv4 model on an image. Now we will run YOLOv4 on a video. We can simply and quickly achieve that by changing the detection command and instead of giving it an image, we will feed a video in it.

    ```
    !./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights -dont_show /mydrive/test.mp4 -i 0 -out_filename /mydrive/results.avi
    ```

Some notes on available command line flags given to us by the Darknet framework and YOLOv4 to allow us to customize and be more flexible.

- `-thresh` flag: we can add a threshold flag for confidences on our detections. Only detections with a confidence level above the specified threshold we set will be returned. For example if we run darknet with YOLOv4 with a threshold flag of 0.5 "`-thresh 0.5`" this will only output detections that have 50% of confidence and above. Everything else will be ignored.
- `-ext_output` flag: This external output flag will give us a few extra details about each detection within an image. It will give us the "left_x, top_y, width, height" bounding box coordinates for each detection.
- `-dont_show` flag: This flag is used to not have the image outputted after running darknet. This doesn't really affect anything when running in Colab as the image is unable to output properly straight from darknet anyway. However, by adding the "`-dont_show`" flag we get rid of the following warning from showing. (Figure 4.8)

```
Unable to init server: Could not connect: Connection refused

(predictions:1850): Gtk-WARNING **: 17:01:00.687: cannot open display:
```

Figure 4.8: Error message when trying to display images with default function in Colab

After all the above steps we have successfully run detection on both an image and a video using YOLOv4 ML model. Moving on to our specific use case we want to run the pre-trained model on an image showing traffic cones and see if the results are acceptable so we could leverage YOLOv4 model as it is, out of the box, or continue to train the model to meet the demands of our specific use case.

4.4 Steps for running YOLOv4 in the cloud on our own images

1. Steps 1 through 12 remain the same as in chapter 4.3
2. We run the detector on an image that is specific to our use case (Figure 4.9) and we get the output of the pre-trained model (Figure 4.10)

   ```
   !./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights ../cone-amz.jpg -dont_show
   ```

   ```
   imShow('predictions.jpg')
   ```

Figure 4.9: Given image to run detector on

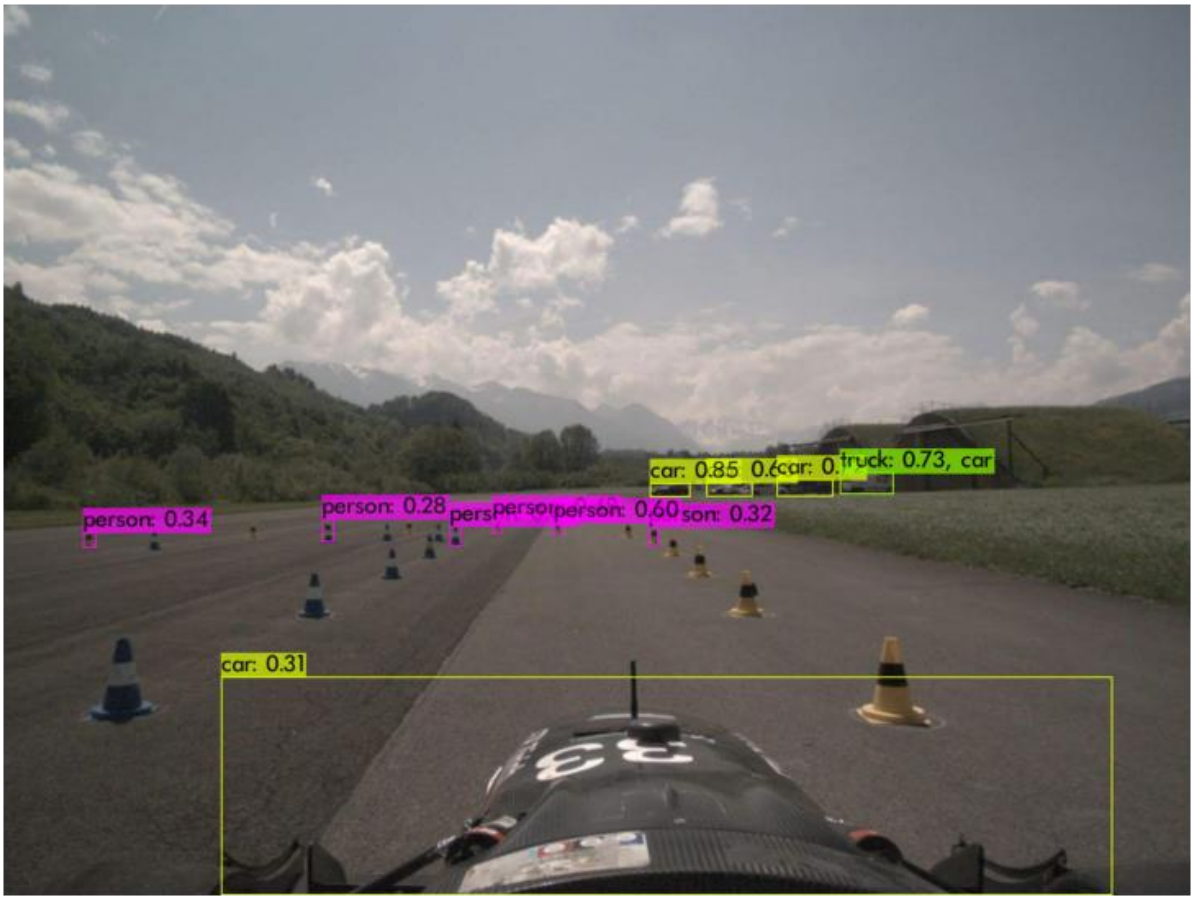


Figure 4.10: Output result of detection

As we can see it successfully recognizes cars and a truck, but it does a terrible job recognizing traffic cones. Some of them are predicted as persons and others are not recognized at all.

From the above results it is obvious that using YOLOv4 without additional training is not going to bring any useful results. We have to find our own dataset, label it and train the model on it.

4.5 Dataset preparation: gathering & labeling

4.5.1 Our dataset

After searching most of the online dataset libraries for images of traffic cones that would fit the needs of our use case, we came across the Open Source Github repository [57] of the Formula Student Objects in Context Dataset for the Formula Student Driverless competitions. The repository included data from various student formula teams that shared their content with other teams in order to create a bigger dataset so they could all benefit and improve their models. In order to become a member who could access the data of all the teams, we had to provide some work too, in the context of labeling traffic cone photos generated from videos. Videos from other more experienced teams were used to split their frames into still images. We contacted the project leader who gave us a dataset of more than 600 images to label so we could be members of this community and to be eligible to access the group's repository with over 15.000 images and their respective data files. (Figure 4.11)

Figure 4.11: Examples of photos that were provided to us

4.5.2 Labeling process

For labeling our images, we used the OpenLabeling tool [58] which is an open source program written in Python, that provides a Graphical User Interface, GUI, to select the boundaries of the objects we want to select and outputs its location in annotation formats like PASCAL VOC (.xml files) and/or in YOLO darknet (.txt files).

First of all, since our tool is written in Python, we need to install Python in our working environment, whether it is a Windows or Mac machine. After completing Python installation, we still need some extra requirements to be able to run the labelling tool as it is shown below.

1. Install OpenCV version 3.0 or above.

```
python -mpip install -U pip
python -mpip install -U opencv-python
python -mpip install -U opencv-contrib-python
```

2. Install numpy[59], tqdm[60] and lxml[61]:

```
python -mpip install -U numpy
python -mpip install -U tqdm
python -mpip install -U lxml
```

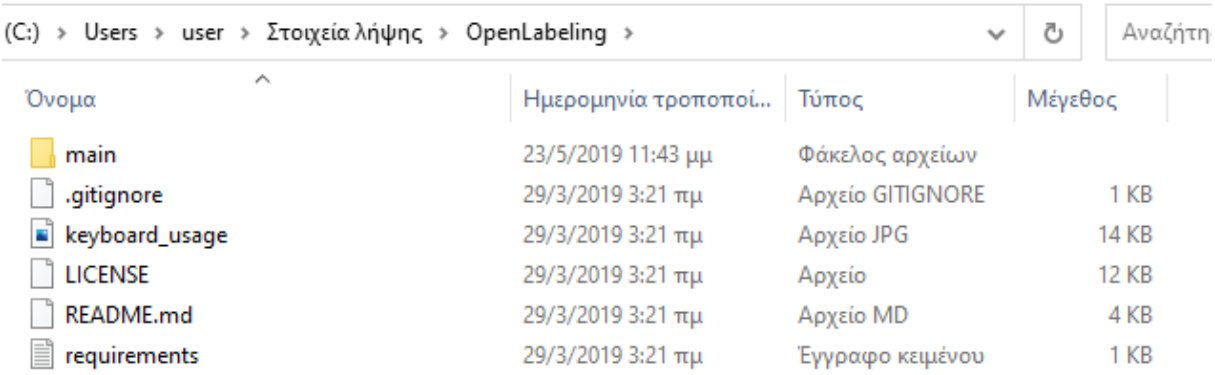


Figure 4.12: contents of OpenLabeling's github

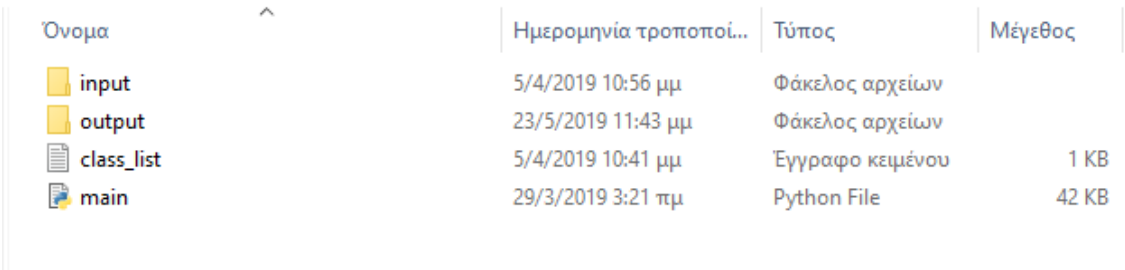


Figure 4.13: Files inside the main folder

3. In the `input` folder we need to put the images we want to label. (Figure 4.14)

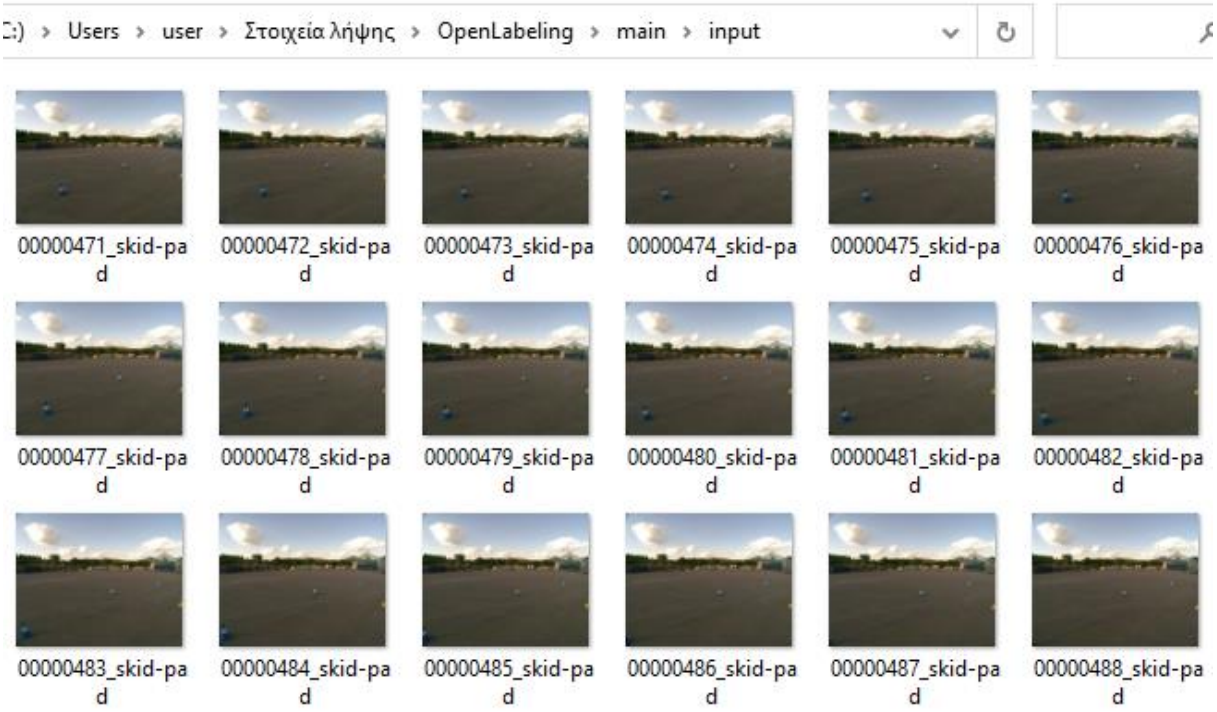


Figure 4.14: Images for labeling inside input folder

4. We next need to insert the classes we are going to be labeling for in the `class_list.txt` file, writing one class name per line.

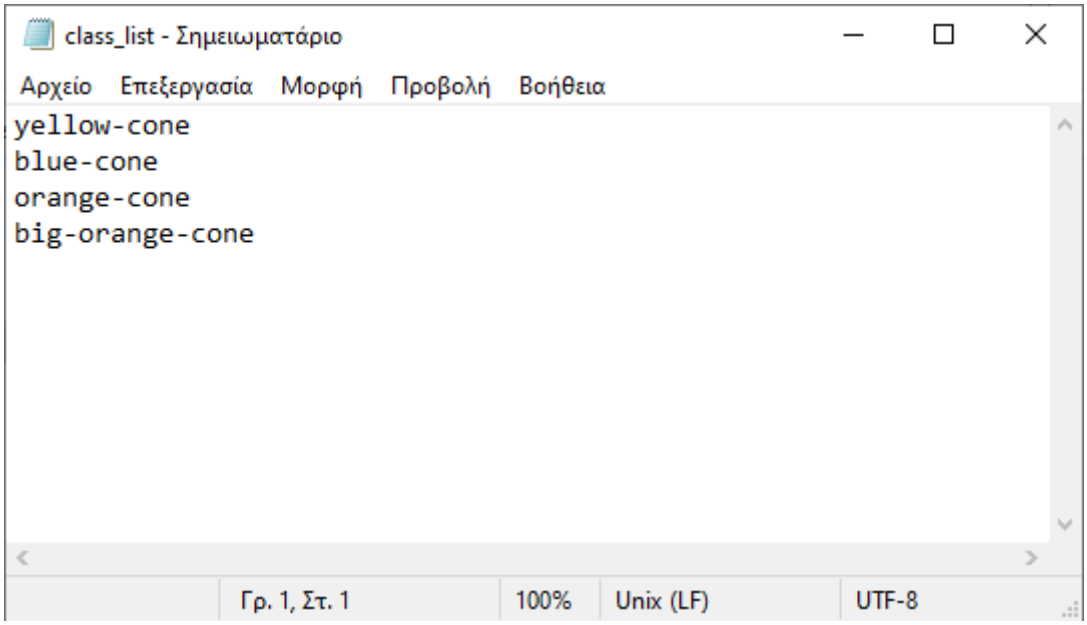


Figure 4.15: Content of class_list.txt file

5. We then need to start the application. In order to do so we "`open`" the `main.py` file with the python interpreter. We open a command line or a terminal, navigate in the main folder and type:

```
python ./main.py
```

6. The GUI of our labeling tool opens and on the top bar we can see the image number as well as the classes we are going to label. (Figure 4.16)

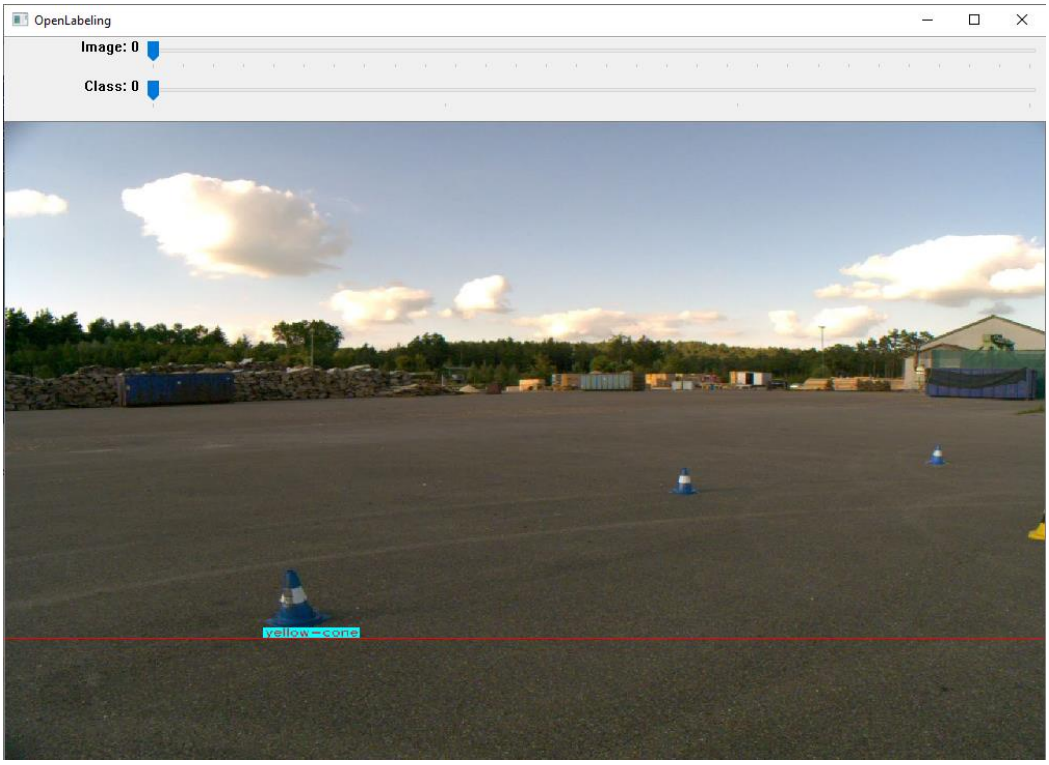


Figure 4.16: GUI of the labeling tool

7. We select the class that we want to label a specific cone and click once on the bottom left side of the cone and then click again on the opposite top right side. (Figure 4.17) In the following image we can see some useful shortcuts so we could navigate through our images and our classes quicker. (Figure 4.18)

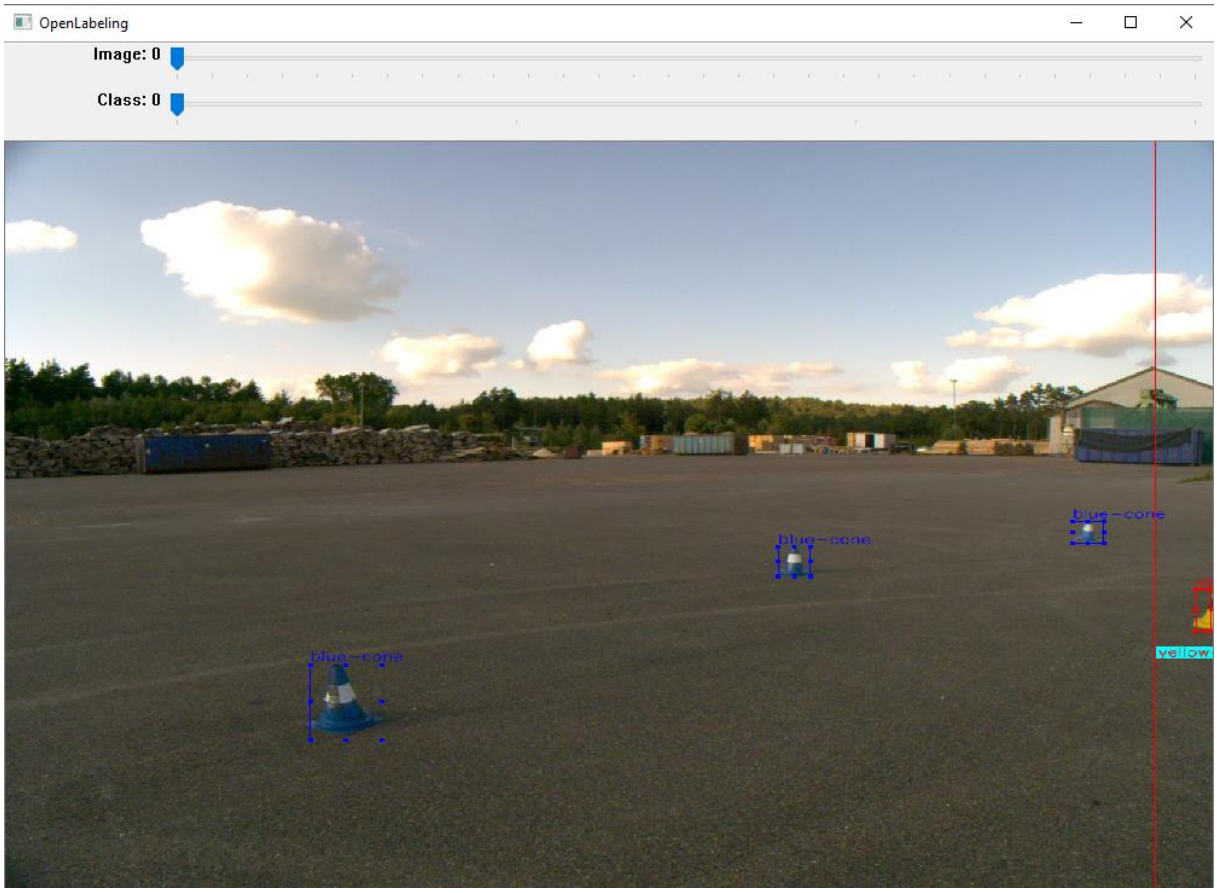


Figure 4.17: Labeling a cone

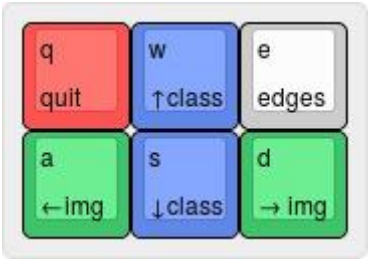


Figure 4.18: Useful keyboard shortcuts

8. Finally, we can find the annotations generated by the labelling tool in the `output` folder, in .xml format. (Figure 4.19)

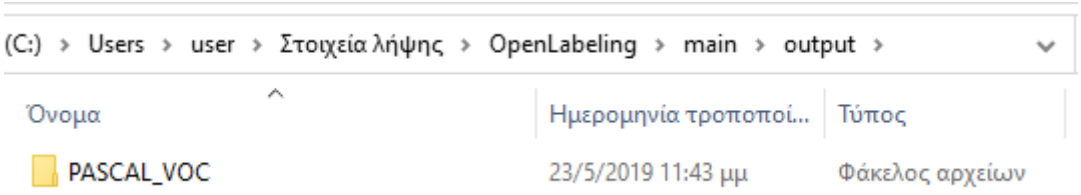


Figure 4.19: Folder containing generated labels in .xml format

*00000471_skid-pad - Σημειωματάριο

Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια

```
<annotation>
  <folder>input</folder>
  <filename>00000471_skid-pad.jpg</filename>
  <path>C:\Labeling\1st Round\OpenLabeling\main\input\00000471_skid-pad.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>1280</width>
    <height>1024</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>blue-cone</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>323</xmin>
      <ymin>717</ymin>
      <xmax>399</xmax>
      <ymax>820</ymax>
    </bndbox>
  </object>
  <object>
    <name>blue-cone</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>819</xmin>
      <ymin>556</ymin>
      <xmax>853</xmax>
      <ymax>596</ymax>
    </bndbox>
  </object>
  <object>
    <name>blue-cone</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>1131</xmin>
      <ymin>521</ymin>
      <xmax>1164</xmax>
      <ymax>550</ymax>
    </bndbox>
  </object>
  <object>
    <name>yellow-cone</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>1262</xmin>
      <ymin>613</ymin>
      <xmax>1278</xmax>
      <ymax>671</ymax>
    </bndbox>
  </object>
</annotation>
```

Figure 4.20: Inputs of a .xml format containing annotations created by labeling tool

For our labels to be accepted by the FSOCO community, after labeling we had to count the total number of labels we created. In order to realize this, we created a tool that counts all the generated labels and outputs the total number, as well as the number per color. For example, to count 10.000 labels in total (3500 blue and 6500 yellow cones). We then shared the script with the formula student community to save some time from other students that need to count their labels. (Figure 4.21)

The counter script works as follows:

1. Get the script from GitHub [62]

2. Create a new folder beside the script named `'filesToRead'` and put your output files in that directory.
3. Modify the classes number in the script, for example blue cones are 0 and orange cones are 1.
4. Run and check the output in the console.

```python
import os

currentPath = os.getcwd()
currentPath += '/filesToRead/'
#TO USE THIS ADD THE FILES IN A 'filesToRead' FOLDER IN THE LOCATION OF THE SCRIPT.
#EDIT THE CONE CODES TO THE NUMBER OF CLASS YOU ARE USING.
yellowConeCode = 0
blueConeCode = 1
orangeConeCode = 2
bigOrangeConeCode = 3
#EDIT THE CONE CODES TO THE NUMBER OF CLASS YOU ARE USING.


noOfYellowCones = 0
noOfBlueCones = 0
noOfOrangeCones = 0
noOfBigOrangeCones = 0
noOfFilesScanned = 0

for filename in os.listdir(currentPath):
    pathOfFile = currentPath + '/' + filename
    file = open(pathOfFile, "r")
    noOfFilesScanned += 1
    #print("FILE:" + filename)
    for line in file:
        if line[0] == str(yellowConeCode):
            noOfYellowCones += 1
        elif line[0] == str(blueConeCode):
            noOfBlueCones += 1
        elif line[0] == str(orangeConeCode):
            noOfOrangeCones += 1
        elif line[0] == str(bigOrangeConeCode):
            noOfBigOrangeCones += 1


print("No of Files Scanned: " + str(noOfFilesScanned))
print("No of Yellow Cones: " + str(noOfYellowCones))
print("No of Blue Cones: " + str(noOfBlueCones))
print("No of Orange Cones: " + str(noOfOrangeCones))
print("No of Big Orange Cones: " + str(noOfBigOrangeCones))
print("No of Cones " + str(noOfYellowCones + noOfBlueCones + noOfOrangeCones + bigOrangeConeCode))
```

Figure 4.21: Script that counts number of cones

4.5.3 Scripts for transforming datasets

During our work, YOLO darknet was not yet supported by the OpenLabeling tool. To overcome this issue, we had to use an annotation converter from PASCAL VOC to YOLO darknet format.

The tool we used [66] for the job was already available in the FSOCO repository. Newer versions of OpenLabeling tool now normally support YOLO format outputs.

The process to convert from PASCAL VOC to YOLO is the following.

- Download the python scripts from GitHub [63]
- Create a folder `'export'` in the main directory. (Figure 4.22)

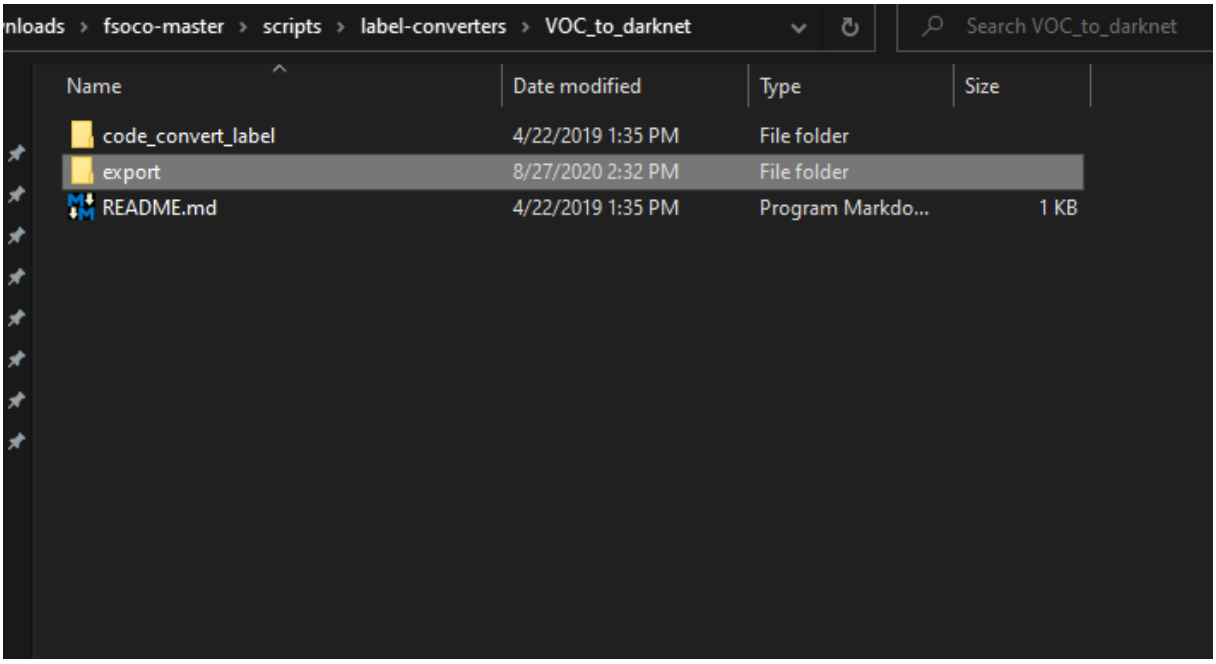


Figure 4.22: Folder containing code to convert VOC to YOLO

- Put the images and the image1.xml files (one per image) into the export folder. (Figure 4.23)

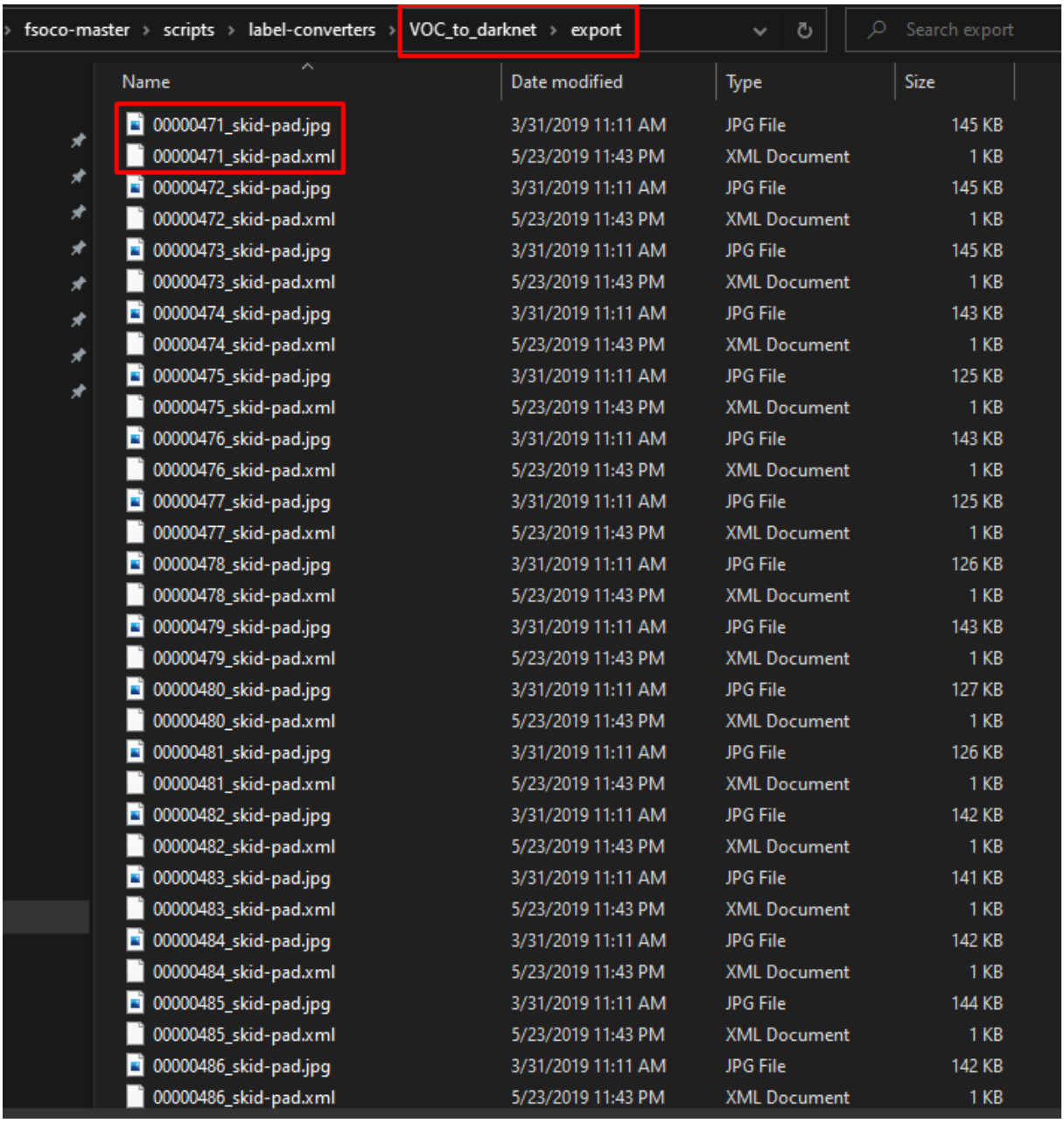


Figure 4.23: Folder containing exports before converting

- Go in the folder `'code_convert_label'`
- Run `python convert_label.py`

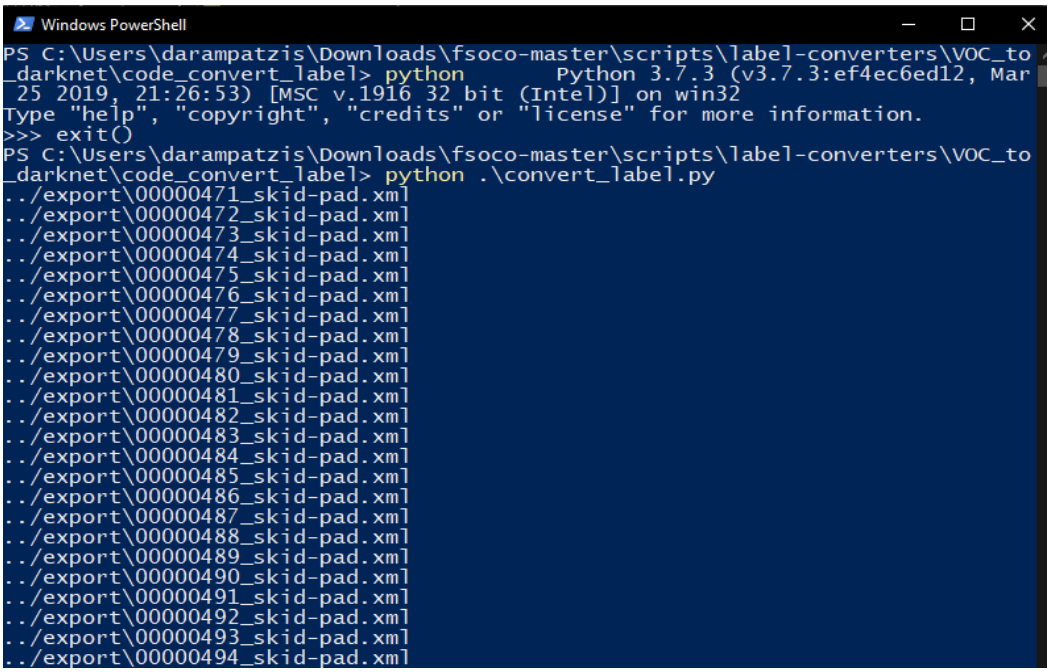


Figure 4.24: Calling the script from Windows powershell

- Now there should be a .txt file for every image in the export folder (Figure 4.25)

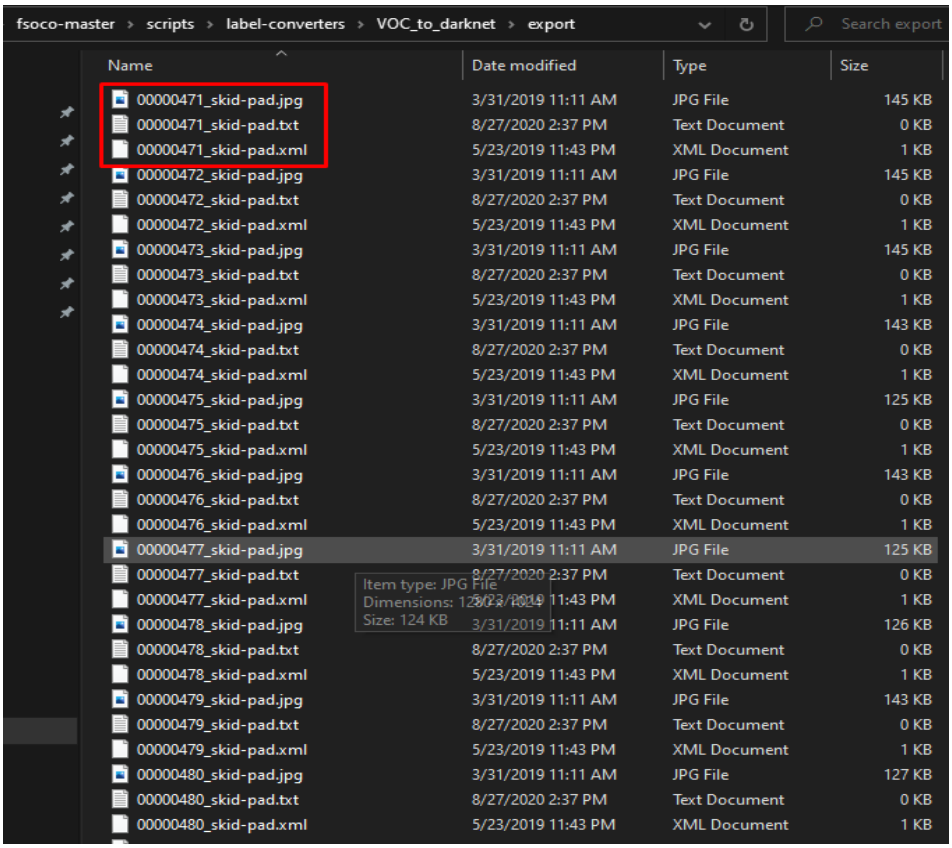


Figure 4.25: Folder containing exports before converting

- The generated .txt file has converted the PASCAL VOC .xml files to our required YOLO format.

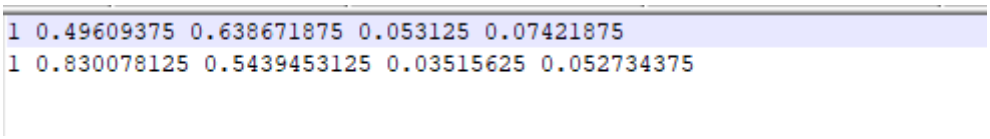


Figure 4.26: Folder containing generated labels in .xml format

4.6 Steps for training YOLOv4 in the cloud

In order to train the YOLOv4, the following steps are the same as previously described:

A. Enable GPU within our Colab

B. Clone and build darknet

C. Download pre-trained YOLOv4 weights

D. Define helper functions

E. Mount Google Drive in our Colab notebook and create symbolic link

Now comes the time to create our own custom YOLOv4 object detector to recognize any classes/objects we want. In order to create a custom YOLOv4 detector we will need the following:

- Labeled Custom Dataset from Chapter 4.5
- Custom .cfg configuration file
- The `obj.data` and `obj.names` files
- The train.txt file (test.txt is optional)

After we have finished with the labeling process as described above it is now time to do all the appropriate settings in order to start our training:

1. We start by zipping all the dataset files together, in order to upload it in our Google Drive. If we try to upload it without zipping, we will get a "`time out`" error from Google Drive and we cannot upload the dataset [64].
2. Then we copy the configuration file for YOLOv4 over to our Google drive so we can open it with our preferred text editor and tweak some settings for our training phase.

   ```
   !cp cfg/yolov4-custom.cfg /mydrive/yolov4/yolov4-obj.cfg
   ```

3. Open yolov4-obj.cfg and make the following edits [65]. Some of them are related to the number of classes we need to train for. In our example we train for 4 classes (blue_cone, yellow_cone, orange_cone, big_orange_cone):

   a. `batch=64` and `subdivisions=16` are the recommended settings for ultimate results. If we run into any issues, then we change the subdivisions to 32.

   b. `width=416` and `height=416`. These can be any multiple of 32. Although 416 is the standard, sometimes we can improve results by making values larger like 608 but sacrificing from speed as this will slow down the training.

   c. `max_batches` should be equal to the `number of classes * 2000` but no less than 6000. So, if we are training for 1, 2, or 3 classes it will be 6000, however a

detector for 5 classes would have `max_batches=10000`. For our training we need `max_batches=8000`

d. `steps` should be `(80% of max_batches), (90% of max_batches).` For our example, our max_batches are 8000 so we configure our steps to be: `steps=6400,7200`

e. For the three YOLO layers marked with `[yolo]` in our .cfg file we change the classes attribute to `classes=4` because we are training for four classes.

f. Moreover, in the three convolutional layers exactly above the three YOLO layers we changed in the previous step we set `filters` equal to `(number of classes + 5) * 3`. We are training for four classes then our filters setting is `filters=27`

g. Optional: if we run into memory issues or find the training taking very long time, we will change one line from `random=1` to `random=0` to speed up training but slightly reduce accuracy of model. This random attribute can be found in one of our YOLO layers in the .cfg file.

4. Moving on we need to create two more files.

a. One will be the `obj.names` in which we will have one class name per line in the same order as your classes.txt from the dataset generation step.(Figure 4.27)

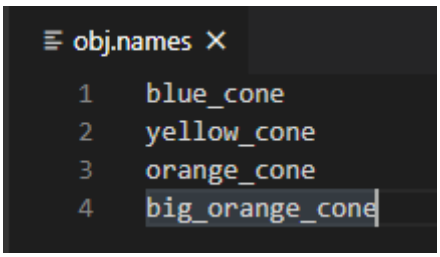


Figure 4.27: Contents of obj.names file

b. The other will be the `obj.data` in which we will have some general settings. Classes are the number of our classes we want to train for. Train and valid are two .txt files containing our train and test data. Name is the obj.names files we created above and lastly, backup is the path where we will save the weights of our model throughout training. Create a backup folder in your Google Drive and put its correct path in this file.(Figure 4.28)

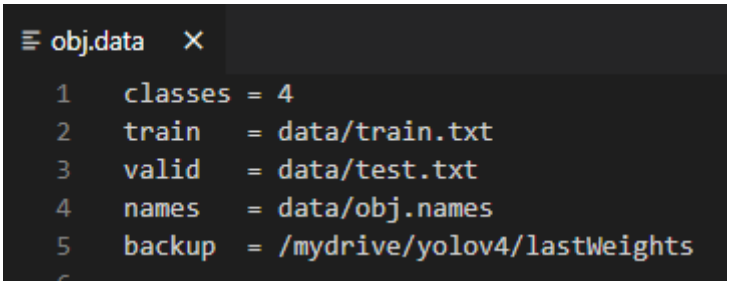


Figure 4.28: Contents of obj.data file

5. The last two configuration files needed before we can begin to train our custom YOLOv4 detector are the train.txt and test.txt files which hold the relative paths to all our training images and validation images. These two .txt files were previously created (Chapter 4.5).
6. We will download the weights for the convolutional layers of the YOLOv4 network and use them during our training. By using these weights, it helps our custom object detector to be way more accurate and not have to train as long. We do not have to use these weights, but this is the best way to help our model converge and be accurate way faster.
7. The time has finally come! You have made it to the moment of truth! You are now ready to train our custom YOLOv4 object detector on whatever crazy classes you have decided on. Thus, we run the following command. (`-dont_show` flag stops chart from popping up since Colab Notebook can't open images on the spot, -map flag overlays mean average precision on chart to see how accuracy of your model is, only add map flag if you have a validation dataset)

   ```
   !./darknet detector train <path to obj.data> <path to custom config> yolov4.conv.137 -dont_show -map
   ```

This training could take several hours depending on how many iterations we choose in the .cfg file. We will let this run as long as we possibly can. However, Colab cloud service kicks us off it's VMs if we are idle for too long (30-90 mins).

To avoid this hold (CTRL + SHIFT + i) at the same time to open up the inspector view on your browser.

Paste the following code into your console window and hit Enter

```
function ClickConnect(){
      console.log("Working");
      document
        .querySelector('#top-toolbar > colab-connect-button')
        .shadowRoot.querySelector('#connect')
        .click()
      }
setInterval(ClickConnect,60000)
```

8. Upload cfg back to cloud vm darknet/cfg
9. Copy `obj.names` and `obj.data` to darknet/data

After the first training session (approximately 8 hours of training) we run again the detector using our edited .cfg file with a small change in batch and subdivisions. We replace `batch=64` to `batch=1` and `subdivisions=16` to `subdivisions=1` as it was in the beginning. We also use our `yolov4_last.weights` from the training phase and we give it our test image. The results are shown in Figure 4.29. We can see that our model is definitely learning because it has predicted cones with above 90% accuracy. The problem that remains is that it does not yet predict the correct color for each of the cones. For this reason, we have to continue training it.

```
!./darknet detector test data/obj.data cfg/cone-yolov4.cfg
/mydrive/yolov4/lastWeights/cone-yolov4_last.weights /mydrive/amz-
400250.jpg -thresh 0.3
```

```
imShow('predictions.jpg')
```

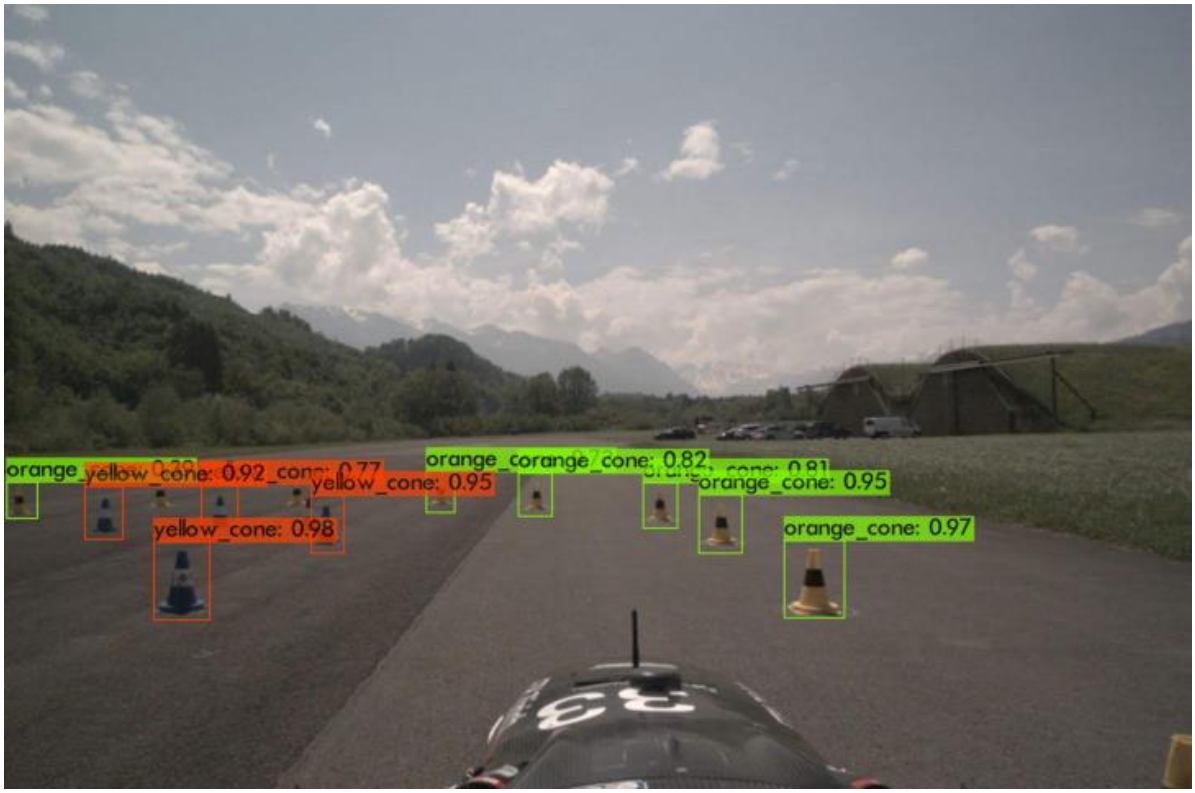


Figure 4.29: Correct cone detection but with incorrect colors

We continue our training on the same dataset but with two differentiations.

1. Firstly, we edit our .cfg file once more to give the correct `batch` and `subdivisions` values appropriate for training. (`batch=64` and `subdivisions=16`)
2. Secondly, instead of using the pre-trained weights that come with YOLOv4 we use our own weights file saved in the backup location we specified in the `obj.data` file. For our purpose the filename of the last weights file is `cone-yolov4.weights`

```
!./darknet detector train data/obj.data cfg/cone-yolov4.cfg
/mydrive/yolov4/lastWeights/cone-yolov4_last.weights -dont_show
```

We continue training our model to the point that we observe that the percentage of success at our test dataset does not decrease. Other than that, we can keep training until we have reached our desired output, meaning that our model can at that point recognize the objects we want with a high degree of accuracy. It is up to us to decide the appropriate level of classification based on our personal standards and our use case, what we want to achieve and whether the knowledge acquired meets our expectations. Finally, of course we could and should change the parameters of the training so we could affect the training speeds and efficiency of our model.

5 Conclusions and future work

As we have entered the dawn of the AI era, this B.Sc. thesis has modestly examined a sample of the scale, complexity and transformative power of machine learning on the field of real time object recognition. The thesis tried to review the related work in the field of automation systems and present the system design, software algorithm and implementation. Ever since the conception of the early CNNs algorithms that were slow and inaccurate, we have come a long way to algorithms like YOLO which are capable of handling real time recognitions with codebases of much smaller size and better results. At the beginning of our project YOLO version 3 was the latest version someone could use. However, nowadays, version 4 is already available in such a short period of time. It comes with a lot of improvements both internally in terms of optimization and algorithm speeds and externally by having better documentation that helps programmers to get started using YOLO version 4 quicker.

In general, working with YOLO is easier than it seems. The hard work of the development team of YOLO is obvious as people without deep knowledge of algorithms can use the pre-trained algorithm without many problems. Of course, programming knowledge and ease with computer usage helps in more applied or even possible training of the algorithm for custom objects. As a general note though, YOLO's ease of use is one of its many advantages given that the person implementing the algorithm is not fond of deeply understanding the inner workings and only wants a quick and mostly trouble-free use.

Our research has led us to discover very interesting alternative use cases of CNNs. One of these use cases would be an aerial drone autonomous navigation system using only visual input from an onboard camera and without relying on a Global Positioning System which is susceptible to signal interference from different sources. This could be achieved using a deep CNN combined with a regressor to output the drone steering commands. The approach is suitable for automating drone navigation in applications that exhibit regular trips or visits to the same locations such as environmental and desertification monitoring, parcel or aid delivery and drone-based wireless internet delivery [66]. In addition, another use case could be, to be used by robots to detect indoor environments better in order to accomplish more complex and flexible navigation tasks. Such a goal is even more desirable when domestic environments are taken into account [67].

Moreover, if YOLO were trained on photos of us and our friends, another intelligent application would be to recognize or confirm someone from advanced sources, for example, a computerized picture or a video stream. Because of the advancements in AI and computer vision, PCs can now beat people in many face recognition assignments, especially those in which substantial databases of human faces must be looked. A system with the ability to distinguish and recognize faces has numerous potential applications including crowd and airport surveillance, private security and enhanced human-computer interaction [68].

Finally, image recognition systems are adding great value to the educational sector by enabling students with learning disabilities to register knowledge in a way that is easier for them. For example, applications that rely on computer vision, allow text-to-speech options that greatly assist visually impaired or dyslexic students to read the content. This is not the limitation of image recognition technology's contribution to the student bodies. It is also helping in breaking free from the traditional teaching boundaries and equipping educators with high-tech learning tools [69]. We are still unable to grasp the full ramifications of YOLO's fast evolution, but we firmly believe that in the near future, we

will be able to rely on these algorithms to perform very complex and maybe life critical processes. The more tests we realize the better results we will gather, always in order to optimize the performance of the algorithm.

On the other hand, the ethical issues that arise when it comes to computer vision and real time deep learning-based object detection and therefore face recognition are numerous. On the 20th February 2020, almost a year before the submission of this thesis, Joe Redman, the creator of the YOLO algorithms, announced his refusal to continue research due to ethical concerns that military applications and privacy issues may entail. Ironically, on the previous day, 19th of February 2020, the European Commission presented its strategies for data and AI in an effort to shape Europe's digital future. Although European Union seeks to accelerate the deployment of AI, at the same time, it has the mission to establish an efficient and trustworthy legislator frame to limit substantive and institutional ramifications. In their annual report 2019, published in July 2020, the Science and Technology Options Assessment panel announced the creation of a Center of AI (C4AI) and examined opportunities and challenges of algorithmic decision-making, as well as legal and ethical reflections on AI systems becoming more and more autonomous on handling personal data as is the case of real time object and face recognition and decision making [70].

It is obvious that important, wide repercussions of AI and machine learning may cause unforeseen implications. Being able to grasp the full impact on us, humans, requires excellent understanding of relevant algorithms and continuous research on each and every possible implementation. Despite our status as computer engineers-or maybe because of it, we cannot neglect the ethical questions that arise, but risks can be countered, and opportunities enabled. The evolution of computer vision applications is inevitable. Every action that consists of a solid combination of tasks can be organized by an algorithm, and this can significantly increase the perplexity of the actions. It is in our hands to trace, control and monitor its momentum.

6 References

[1] Klaus Schwab, “The Fourth Industrial Revolution: what it means, how to respond,” World Economic Forum, 2016. [Online]. (Last accessed: December 2020) Available: www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond

[2] Pooja Agarwal, Pooja Yadav, Neelam Sharma, Ruchika Uniyal, Swati Sharma, “Research Paper on Artificial Intelligence,” Case Studies Journal, vol. 2, Issue 6, Jul 2013

[3] David L. Poole, Alan K. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*, Cambridge University Press, 2010

[4] Nils J. Nilson, *Introduction to machine learning: An early draft of a proposed textbook*, Stanford University, 1998

[5] Schwarz Chris, Thomas Geb, Nelson Kory, McCrary Michael, Sclarmann Nick, Powell Matthew, *Towards Autonomous Vehicles*, Mid-America Transportation Center, 2013

[6] James M. Anderson, Nidhi Kalra, Karlyn D. Stanley, Paul Sorensen, Constantine Samaras, A. Oluwatola, *Autonomous Vehicle Technology: A Guide for Policymakers*, RAND Corporation, 2016

[7] Marc Weber, “A History of Autonomous Vehicles,” American Energy Society, 2018. [Online]. (Last accessed: November 2020) Available: www.energytoday.net/technology/where-to-a-history-of-autonomous-vehicles

[8] Kroeger F., *Automated Driving in its Social, Historical, and Cultural Contexts*, in: Maurer, M., Gerdes, C. J., Lenz, B., Winner, H., Autonomous Driving, Springer Berlin Heidelberg, pp. 41–68. 2016

[9] K. Bimbraw, "Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology," 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Colmar, 2015, pp. 191-198.

[10] Tschangho John Kim, “Automated Autonomous Vehicles: Prospects and Impacts on Society,” Journal of Transportation Technologies, vol. 8, no. 3, pp. 137-150, July 2018

[11] Bagloee, S.A., Tavana, M., Asadi, M. et al. “Autonomous vehicles: challenges, opportunities, and future implications for transportation policies,” Journal of Modern Transport, vol. 24, pp. 284–303, August 2016

[12] Fadaie, Joshua G, *The State of Modeling, Simulation, and Data Utilization within Industry*, The Boeing Company, 2018

[13] Katie Burke, “How Does a Self-Driving Car See?,” Nvidia, 2019. [Online]. (Last accessed: June 2020) Available: blogs.nvidia.com/blog/2019/04/15/how-does-a-self-driving-car-see

[14] Trevor English, “How Do Self-Driving Cars Work?,” Interesting Engineering, 2020. [Online]. (Last accessed: November 2020) Available: interestingengineering.com/how-do-self-driving-cars-work

[15] Steven Goodridge, “Autonomous Driving and Collision Avoidance Technology,” BikeWalk NC, 2018. [Online]. (Last accessed: July 2020) Available: www.bikewalknc.org/2018/02/autonomous-driving-and-collision-avoidance-technology

[16] Ihor Starepravo, " How Sensor Fusion for Autonomous Cars Helps Avoid Deaths on the Road," Intelligent Software Engineering, 2018. [Online]. (Last accessed: December 2020) Available: www.intellias.com/sensor-fusion-autonomous-cars-helps-avoid-deaths-road

[17] Jenna Thompson, "Ultrasonic Sensors: More Than Just Parking," Level Five Supplies Ltd, 2019 [Online]. (Last accessed: November 2020) Available: levelfivesupplies.com/ultrasonic-sensors-more-than-just-parking

[18] Semcon," Lidar vs Radar for applied autonomy," Semcon, 2020. [Online]. (Last accessed: July 2020) Available: semcon.com/offerings/applied-autonomy/lidar-vs-radar-for-applied-autonomy

[19] Christoph Hammerschmidt, "Smart signal data processing slashes response time for cars," Ee News Automotive, 2019. [Online]. (Last accessed: October 2020) Available: www.eenewsautomotive.com/news/smart-signal-data-processing-slashes-response-time-cars

[20] Oliver Cameron," An Introduction to LIDAR: The Key Self-Driving Car Sensor," Voyage, 2017 [Online]. (Last accessed: November 2020) Available: news.voyage.auto/an-introduction-to-lidar-the-key-self-driving-car-sensor-a7e405590cff

[21] Steve Crowe, "Researchers back Tesla's non-LiDAR approach to self-driving cars," The Robot Report, 2019. [Online]. (Last accessed: October 2020) Available: www.therobotreport.com/researchers-back-teslas-non-lidar-approach-to-self-driving-cars

[22] Martin Alva," Will Teaching Autonomous Cars to Perceive Like A Human Solve All Safety Problems?," Amena Auto, 2019. [Online]. (Last accessed: June 2020) Available: www.amenaauto.org/2019/04/will-teaching-autonomous-cars-to-perceive-like-a-human-solve-all-safety-problems

[23] Christoph Domke, Quentin Potts, "LiDARs for self-driving vehicles: a technological arms race," Automotive World, 2020. [Online]. (Last accessed: November 2020) Available: www.automotiveworld.com/articles/lidars-for-self-driving-vehicles-a-technological-arms-race

[24] Michael Gauthier," VW On Track to Launch A Fully Autonomous Vehicle By 2025," Carscoops, 2019. [Online]. (Last accessed: November 2020) Available: www.carscoops.com/2019/11/vw-on-track-to-launch-a-fully-autonomous-vehicle-by-2025

[25] Jeremy Lee, "How Sensor Fusion Works," All about circuits, 2016. [Online]. (Last accessed: December 2020) Available: www.allaboutcircuits.com/technical-articles/how-sensor-fusion-works

[26] Aptiv, "What Is Sensor Fusion?," Aptiv, 2020. [Online]. (Last accessed: November 2020) Available: www.aptiv.com/insights/article/what-is-sensor-fusion

[27] Diego Galar, Uday Kumar, *eMaintenance: Essential Electronic Tools for Efficiency*, Academic Press, 2017.

[28] Anna Tobin, "What Are the Different Levels of Self-Driving Cars?," Forbes, 2019. [Online]. (Last accessed: November 2020) Available: www.forbes.com/sites/annatobin/2019/07/07/what-are-the-different-levels-of-self-driving-cars

[29] Matt Schmitz, "Autonomous Driving Levels and What They Mean to You," Cars.com, 2020. [Online]. (Last accessed: August 2020) Available: www.cars.com/articles/autonomous-driving-levels-and-what-they-mean-to-you-424979

[30] Viknesh Vijayenthiran, “Navya already sells fully self-driving cars, including in US,” Motor Authority, 2018. [Online]. (Last accessed: July 2020) Available: www.motorauthority.com/news/1118809_navya-already-sells-fully-self-driving-cars-including-in-us

[31] Stein Law, “The rise of autonomous vehicles,” Stein Law, 2018, [Online]. (Last accessed: July 2020) Available: www.steinlaw.com/resources/studies/the-rise-of-autonomous-vehicles

[32] “Autonomous Vehicles,” University of Minnesota, 2020 [Online] (Last accessed: December 2020) Available: minnesotago.org/application/files/3614/6222/6829/Autonomous_Vehicles.pdf

[33] Todd Litman, *Autonomous Vehicle Implementation Predictions*, Victoria Transport Policy Institute, 2020

[34] Matt Zeunert, “A super simple introduction to neural networks,” Matt Zeunert, 2016. [Online]. (Last accessed: November 2020) Available: www.mattzeunert.com/2016/12/09/neural-networks-super-simple-introduction.html

[35] Chris Nicholson, “A Beginner's Guide to Convolutional Neural Networks (CNNs),” Pathmind, 2020. [Online]. (Last accessed: October 2020) Available: wiki.pathmind.com/convolutional-network

[36] Sunil kumar Jangir, “Beginner’s Guide for Convolutional Neural Network (CNN / ConvNets),” Towards Data Science, 2018. [Online]. (Last accessed: August 2020) Available: towardsdatascience.com/beginners-guide-for-convolutional-neural-network-cnn-convnets-5a5e725ea581

[37] Ujj walkarn, “An Intuitive Explanation of Convolutional Neural Networks,” ujjwalkarn, 2016. [Online]. (Last accessed: November 2020) Available: ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets

[38] Erin Welling, *Convolutional Neural Networks in Autonomous Vehicle Control Systems*, 2017

[39] Fei-Fei Li, Ranjay Krishna, Danfei Xu, Amelie Byun, “Convolutional Neural Networks for Visual Recognition,” Stanford University, 2018. [Online]. (Last accessed: September 2020) Available: cs231n.github.io/convolutional-networks

[40] Sumit Saha, “Comprehensive Guide to Convolutional Neural Networks,” Towards Data Science, 2018. [Online]. (Last accessed: December 2020) Available: towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[41] K. L. Masita, A. N. Hasan and T. Shongwe, *Deep Learning in Object Detection: a Review*, 2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD), Durban, South Africa, pp. 1-11, 2020

[42] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, IEEE Conference on Computer Vision and Pattern Recognition, 2014.

[43] Ashish Kumar, “Artificial Intelligence in Object Detection – Report,” 2020

[44] Rohith Gandhi, “R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms,” Towards Data Science, 2018. [Online]. (Last accessed: November 2020) Available: towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[45] Jonathan Hui, “SSD object detection: Single Shot MultiBox Detector for real-time processing,” Jonathan Hui Medium, 2018. [Online]. (Last accessed: November 2020) Available: jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06

[46] Kumar Dash,“ Single Shot Detection (SSD) Algorithm”, opengenus, 2018. [Online]. (Last accessed: November 2020) Available: iq.opengenus.org/single-shot-detection-ssd-algorithm

[47] Eddie Forson,“ Understanding SSD MultiBox — Real-Time Object Detection in Deep Learning”, Towards Data Science, 2017. [Online]. (Last accessed: November 2020) Available: towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab

[48] ODSC - Open Data Science, “Overview of the YOLO Object Detection Algorithm”, Medium, 2018. [Online]. (Last accessed: November 2020) Available: medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0

[49] Jason Brownlee, “How to Perform Object Detection with YOLOv3 in Keras”, Machine Learning Mastery, 2019. [Online]. (Last accessed: November 2020) Available: machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras

[50] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, University of Washington, Allen Institute for AI, Facebook AI Research, 2016

[51] Manish Gupta, “ YOLO — You Only Look Once”, Towards Data Science, 2020. [Online]. (Last accessed: November 2020) Available: towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4

[52] Matthijs Hollemans,“ Real-time object detection with YOLO”, Machinethink, 2018. [Online]. (Last accessed: June 2020) Available: machinethink.net/blog/object-detection-with-yolo

[53] Sayon Dutta,“ Reinforcement Learning with TensorFlow”, Oreilly, 2018. [Online]. (Last accessed: November 2020) Available: www.oreilly.com/library/view/reinforcement-learning-with/9781788835725/786aac81-77a7-437e-9a75-64925d7940ca.xhtml

[54] Alexey Bochkovskiy, Chien-Yao Wang and Hong-Yuan Mark Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” Institute of Information Science Academia Sinica, Taiwan, 2020

[55] colab.research.google.com (Last accessed: September 2020)

[56] jupyter.org (Last accessed: October 2020)

[57] github.com/ddavid/fsoco (Last accessed: July 2020)

[58] github.com/Cartucho/OpenLabeling (Last accessed: July 2020)

[59] numpy.org (Last accessed: September 2020)

[60] tqdm.github.io (Last accessed: November 2020)

[61] lxml.de (Last accessed: November 2020)

[62] github.com/Dimitris-Arabatzis/fsoco/tree/master/scripts/NumerOfCones (Last accessed: December 2020)

[63] github.com/Dimitris-Arabatzis/fsoco/tree/master/scripts/label-converters/VOC_to_darknet (Last accessed: December 2020)

[64] research.google.com/colaboratory/faq.html#drive-timeout (Last accessed: November 2020)

[65] github.com/alexeyab/darknet#how-to-train-to-detect-your-custom-objects (Last accessed: September 2020)

[66] K. Amer, M. Samy, M. Shaker, M. ElHelw, *Deep Convolutional Neural Network-Based Autonomous Drone Navigation*, Center for Informatics Science Nile University, 2019

[67] C. Fernández-Caramés, V. Moreno, B. Curto, J. F. Rodríguez-Aragón, F. J. Serrano, "A Real-time Door Detection System for Domestic Robotic Navigation," Journal of Intelligent & Robotic Systems, vol. 76, pp. 119–136, 2014

[68] Jaychand Upadhyay, Parkar Rida, Sunidhi Gupta, Noman Siddique, "Smart Doorbell System based on Face Recognition," International Research Journal of Engineering and Technology (IRJET), vol. 04, issue 03, Mar 2017

[69] Zakiya Aulia Ilma, Karunia Eka Lestari, "Technology for teaching and learning geometry: beyond the visual recognition of spatial relation with brain-based learning in the junior grades," 1st International Conference on Education in Indonesia (ICE-I), 2016

[70] Scientific Foresight Unit (STOA) EPRS, "EP Panel for the Future of Science and Technology (STOA) Annual Report 2019," European Parliamentary Research Service, 2020

Appendix A: Useful scripts

Number of cones

```
#Simple tool for counting the number of cones (total and by class)
#JUST FOR DARKNET YOLO FORMAT!
#
#To use:
#1)Add your output files in the filesToRead directory.
#2)Modify the classes number in the script ex.Blue cones are 0 and orange
cones are
#3)Run and check the output.

Import os

currentPath = os.getcwd()
currentPath += '/filesToRead/'
#TO USE THIS ADD THE FILES IN A 'filesToRead' FOLDER IN THE LOCATION OF THE
SCRIPT.
#EDIT THE CONE CODES TO THE NUMBER OF CLASS YOU ARE USING.
yellowConeCode = 0
blueConeCode = 1
orangeConeCode = 2
bigOrangeConeCode = 3
#EDIT THE CONE CODES TO THE NUMBER OF CLASS YOU ARE USING.


noOfYellowCones = 0
noOfBlueCones = 0
noOfOrangeCones = 0
noOfBigOrangeCones = 0
noOfFilesScanned = 0

for filename in os.listdir(currentPath):
    pathOfFile = currentPath + '/' + filename
    file = open(pathOfFile, "r")
    noOfFilesScanned += 1
    #print("FILE:" + filename)
    for line in file:
        if line[0] == str(yellowConeCode):
            noOfYellowCones += 1
        elif line[0] == str(blueConeCode):
            noOfBlueCones += 1
        elif line[0] == str(orangeConeCode):
            noOfOrangeCones += 1
        elif line[0] == str(bigOrangeConeCode):
            noOfBigOrangeCones += 1


print("No of Files Scanned: " + str(noOfFilesScanned))
print("No of Yellow Cones: " + str(noOfYellowCones))
print("No of Blue Cones: " + str(noOfBlueCones))
print("No of Orange Cones: " + str(noOfOrangeCones))
print("No of Big Orange Cones: " + str(noOfBigOrangeCones))
print("No of Cones " + str(noOfYellowCones + noOfBlueCones +
noOfOrangeCones + bigOrangeConeCode))
```

Uniform Script

```
import os
import sys
import argparse

# function for showing a progress bar in the command line
def progressBar(count, total):
    bar_len = 50
    filled_len = int(round(bar_len * count / float(total)))

    percents = round(100.0 * count / float(total), 1)
    bar = '=' * filled_len + '-' * (bar_len - filled_len)

    sys.stdout.write('[%s] %s%s\r' % (bar, percents, '%'))
    sys.stdout.flush()


# creating the parser for the path argument that we get via the command
line
parser = argparse.ArgumentParser(description='Get a list of files from a
\"labels\" directory and output them in a new directory called
\"changedLabels\"')
parser.add_argument('-p', '--path', required=True, help='path to the
directory in which will find the \"labels\" directory')
args = parser.parse_args()

print('\nScript is running....\n')

# create the dir for the changed labels
os.mkdir('./' + args.path + '/changedLabels')

try:
    # get the list of the files in the folder labels
    dirList = os.listdir('./' + args.path + './labels')

    if len(dirList) == 0:
        raise Exception
    totalFiles = len(dirList)

    for fi in dirList:
        file = open('./' + args.path + './labels/' + fi)
        newFile = open('./' + args.path + '/changedLabels/' +
os.path.basename(file.name), 'a')

        for line in file:
            splitList = line.split(' ')
            if splitList[0] == '0':
                newFile.write('1 ' + splitList[1] + ' ' + splitList[2] + '
' + splitList[3] + ' ' + splitList[4])
            elif splitList[0] == '1':
                newFile.write('0 ' + splitList[1] + ' ' + splitList[2] + '
' + splitList[3] + ' ' + splitList[4])
            elif splitList[0] == '2':
                newFile.write('2 ' + splitList[1] + ' ' + splitList[2] + '
' + splitList[3] + ' ' + splitList[4])
            elif splitList[0] == '3':
                newFile.write('3 ' + splitList[1] + ' ' + splitList[2] + '
' + splitList[3] + ' ' + splitList[4])

        file.close()
```

```
        newFile.close()
        # progress bar
        progressBar(count=dirList.index(fi), total=totalFiles)

except FileNotFoundError:
    print('Oops: You must have a \"labels\" folder in your directory with
your labels\' files in it')

except FileExistsError:
    print('Oops: It seems like the \"changedLabels\" directory already
exists. If you want to run this script, you have to delete it first')

except Exception as error:
    print('Oops: You need to have at least one file in your directory for
this script to work')

finally:
    print('\n\n....script finished running!')
```

Splitting Script

```
import os
import sys
import random

# function for showing a progress bar in the command line
def progressBar(text, count, total):
    bar_len = 50
    filled_len = int(round(bar_len * count / float(total)))

    percents = round(100.0 * count / float(total), 1)
    bar = '=' * filled_len + '-' * (bar_len - filled_len)

    sys.stdout.write(text)
    sys.stdout.write('[%s] %s%s\r' % (bar, percents, '%'))
    sys.stdout.flush()


print('\nScript is running....\n')

try:
    # get the list of the files in the folder labels
    dirList = os.listdir('./3readyForTraining')
    imgList = []

    if len(dirList) == 0:
        raise Exception
    totalFiles = len(dirList)

    for fi in dirList:
        if (fi.endswith('.jpg') or fi.endswith('.png')):
            index = dirList.index(fi)
            imgList.insert(index, fi)
        progressBar(text='Creating list with all images: ',
count=dirList.index(fi), total=totalFiles)
    print('\n')
```

```
    testFile = open('test.txt', 'w+')
    for i in range(3300):
        randomElement = random.choice(imgList)
        imgPath = '/content/gdrive/My
Drive/Mine/TEI/_Ptyxiaki_YOLO/dataset/' + randomElement + '\n'
        testFile.write(imgPath)
        imgList.pop(imgList.index(randomElement))
        progressBar(text='Creating txt with images for testing: ', count=i,
total=3300)
    testFile.close()
    print('\n')

    trainFile = open('train.txt', 'w+')
    restImgs = len(imgList)
    for i in range(restImgs):
        imgPath = '/content/gdrive/My
Drive/Mine/TEI/_Ptyxiaki_YOLO/dataset/' + imgList[i] + '\n'
        trainFile.write(imgPath)
        progressBar(text='Creating txt with images for training: ',
count=i, total=restImgs)
    trainFile.close()


except FileNotFoundError:
    print('Oops: You must have a \"labels\" folder in your directory with
your labels\' files in it')

except FileExistsError:
    print('Oops: It seems like the \"changedLabels\" directory already
exists. If you want to run this script, you have to delete it first')

except Exception as error:
    print('Oops: Something went terribly wrong')

finally:
    print('\n\n....script finished running!')
```

Remove Class

```
import os
import sys
import argparse

# function for showing a progress bar in the command line
def progressBar(count, total):
    bar_len = 50
    filled_len = int(round(bar_len * count / float(total)))

    percents = round(100.0 * count / float(total), 1)
    bar = '=' * filled_len + '-' * (bar_len - filled_len)

    sys.stdout.write('[%s] %s%s\r' % (bar, percents, '%'))
    sys.stdout.flush()


# creating the parser for the path argument that we get via the command
line
parser = argparse.ArgumentParser(description='Get a list of files from a
```

```
\"labels\" directory and output them in a new directory called
\"changedLabels\"')
parser.add_argument('-p', '--path', required=True, help='path to the
directory in which will find the \"labels\" directory')
args = parser.parse_args()

print('\nScript is running....\n')

# create the dir for the changed labels
os.mkdir('./' + args.path + '/changedLabels')

try:
    # get the list of the files in the folder labels
    dirList = os.listdir('./' + args.path + './labels')

    if len(dirList) == 0:
        raise Exception
    totalFiles = len(dirList)

    for fi in dirList:
        file = open('./' + args.path + './labels/' + fi)
        newFile = open('./' + args.path + '/changedLabels/' +
os.path.basename(file.name), 'a')

        for line in file:
            splitList = line.split(' ')
            if splitList[0] == '0':
                newFile.write('0 ' + splitList[1] + ' ' + splitList[2] + '
' + splitList[3] + ' ' + splitList[4])
            elif splitList[0] == '1':
                newFile.write('1 ' + splitList[1] + ' ' + splitList[2] + '
' + splitList[3] + ' ' + splitList[4])
            elif splitList[0] == '2':
                newFile.write('2 ' + splitList[1] + ' ' + splitList[2] + '
' + splitList[3] + ' ' + splitList[4])
            elif splitList[0] == '3':
                newFile.write('')
            elif splitList[0] == '4':
                newFile.write('4 ' + splitList[1] + ' ' + splitList[2] + '
' + splitList[3] + ' ' + splitList[4])

        file.close()
        newFile.close()
        # progress bar
        progressBar(count=dirList.index(fi), total=totalFiles)

except FileNotFoundError:
    print('Oops: You must have a \"labels\" folder in your directory with
your labels\' files in it')

except FileExistsError:
    print('Oops: It seems like the \"changedLabels\" directory already
exists. If you want to run this script, you have to delete it first')

except Exception as error:
    print('Oops: You need to have at least one file in your directory for
this script to work')

finally:
    print('\n....script finished running!')
```

Convert MML to DarkNet

```
# A quick tool to convert the label format of munichs labeling tool #to
Darknet YOLO format.
#How to use?
#No command line parameters. Just change the variables idir, odir, #width
and height and run the python file. By default, blue cones #will be mapped
to class index 0, yellow to class 1, orange to 2 and #big-orange to 3,
though of course you can also change that.

import os

# directory of MM-Labels
idir = "raceyard/ry_2019_04/"
# directory of new Darknet Labels
odir = "raceyard_converted/"

# go through all files
for filename in os.listdir(idir):

    # create new label file
    lf = open(odir + filename, 'w+')

    # load label and skip first line
    f = open(idir + filename, "r")
    f.readline()

    # go through all remaining lines and get data
    line = f.readline()
    while line:
        data = line.split()
        x1,y1,x2,y2,label = int(data[0]), int(data[1]), int(data[2]),
int(data[3]), data[4]

   # change this to actual size
        width, height = 1280, 1024

        # calc YOLO Coordinates
        dw = 1. / width
        dh = 1. / height
        x = (x1 + x2) / 2.0
        y = (y1 + y2) / 2.0
        w = x2 - x1
        h = y2 - y1
        x = x * dw
        w = w * dw
        y = y * dh
        h = h * dh

        # get label-number
        cone_class = -1
        if(label == "blue-cone"):
            cone_class = 1
        elif (label == "yellow-cone"):
            cone_class = 0
        elif (label == "red-cone"):
            cone_class = 2
        elif (label == "big-red-cone"):
            cone_class = 3
        if(cone_class == -1 ):
            print("NO CLASS FOUND FOR: " + filename)
```

```
        # write new label into file
        lf.write(str(cone_class) + " " + str(x) + " " + str(y) + " " +
str(w) + " " + str(h) + "\n")

        # read next line
        line = f.readline()


    f.close()
    lf.close()
```
