ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Αυτόματη Αναγνώριση Οχημάτων σε Διασταυρώσεις και Ενημέρωσης Κινούμενων Οχημάτων

Σαραφίδης Θεμιστοκλής

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Κωνσταντίνος Κολομβάτσος
Επίκουρος Καθηγητής

Λαμία έτος 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Αυτόματη Αναγνώριση Οχημάτων σε Διασταυρώσεις και Ενημέρωσης Κινούμενων Οχημάτων

Σαραφίδης Θεμιστοκλής

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Automatic Recognition of Vehicles at Intersections and Update using Licence Plates

Themistoklis Sarafidis

ΥΠΕΥΘΥΝΟΣ

Κωνσταντίνος Κολομβάτσος
Επίκουρος Καθηγητής

FINAL THESIS

ADVISOR

Konstantinos Kolomvatsos
Assistant Professor

Lamia 2023

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: 7/10/2023

Ο Δηλ.
Σαραφίδης Θεμιστοκλής

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Σήμερα, κάθε ηλεκτρονική συσκευή έχει κάποιου είδους αλγόριθμο Τεχνητής Νοημοσύνης (AI) να τρέχει στο παρασκήνιο. Βρισκόμαστε στην εποχή των Συστημάτων Μεγάλου Όγκου Δεδομένων (Big Data), της Νεφοϋπολογιστικής (Cloud Computing) και των συστημάτων AI . Η αυτόματη αναγνώριση οχημάτων σε Διασταυρώσεις και Ενημέρωσης μεσώ πινακίδων αυτοκίνητων όπως κάθε άλλη εφαρμογή AI πασχίζει για την τελειότητα. Συνεχόμενα δημιουργούνται νέα μοντέλα ή βελτιώνονται ήδη καθιερωμένα μοντέλα, με την ανάγκη για νέα πειράματα και εφαρμογές να παραμένει αμείωτη. Σε αυτή τη εργασία δημιουργήσαμε ένα Σύστημα Αυτόματης Αναγνώρισης Αυτοκινήτων χρησιμοποιώντας το προ-εκπαιδευμένο μοντέλο YOLOv8n στο σύνολο δεδομένων COCO 2017, δημιουργήσαμε και εκπαιδεύσαμε ένα νέο μοντέλο της ίδιας αρχιτεκτονικής για την αναγνώριση πινακίδων αυτοκίνητων. Σε συνδυασμό με ένα εργαλείο Οπτικής Αναγνώρισης Χαρακτήρων (OCR) παράγεται το τελικό σύστημα, από άκρο σε άκρο (end-to-end) για μοναδική συσκευή, το οποίο αναγνωρίζει οχήματα και καταγράφει τον αριθμό πινακίδων. Τα αποτελέσματα της εκτέλεσης φαίνονται υποσχόμενα και το σύστημα θα μπορούσε να επεκταθεί, σε λειτουργιά με χρήση Υπολογιστικού Νέφους και αλλαγή από διατερματικό τρόπο end-to-end εκτέλεσης σε εκτέλεση με υπολογιστική παρυφή (edge-computing). Υπάρχει ακόμα χώρος για βελτίωση του μοντέλου κατά τη διαδικασία της εκπαίδευσης. Στη Ελλάδα, η Αυτόματη Αναγνώριση Οχημάτων σε Διασταυρώσεις και Ενημέρωσης μεσώ Πινακίδων Αυτοκίνητων είναι μια τεχνολογία που μπορεί να ωφελήσει τη δημόσια ασφάλεια ή μπορεί να χρησιμοποιηθεί σαν ένα εμπορικό εργαλείο για τη αυτοματοποίηση διάφορων υπηρεσιών. Με τη χρήση της Τεχνητής Νοημοσύνης και συγκεκριμένα των Νευρωνικών Δικτύων, καθήκοντα όπως η παρακολούθηση απομακρυσμένων ή δημοσίων περιοχών για παραπτώματα και τις εισόδους δασικών εκτάσεων για πιθανή εγκληματική δραστηριότητα θα επιλύονται άμεσα και αποτελεσματικά χωρίς αμφιβολία.

ABSTRACT

Nowadays, almost every device has an Artificial Intelligence (AI) algorithm running in the background. It is the age of Big Data, Cloud Services, and AI automation. Automatic Recognition of Vehicles at Intersections and Updates using Licence Plates as every other AI-related application strives for perfection. As new models and improved already established algorithms rise so does the need to experiment with them. Here we created an Automatic Recognition System using the pre-trained YOLOv8n on COCO 2017 and a new Licence Plate Detector with the same archetype. Combined with an Optical Character Recognition (OCR) we created an end-to-end single device system that recognizes Vehicles and extracts their Licence Plates information. The results seem promising for being scaled to Cloud and Edge, there was a lot of room for improvement. In Greece, Automatic Recognition of Vehicles at Intersections and Updates using Licence Plates is something that can benefit public safety and also be utilized as a commercial tool. Monitoring remote or public areas for misconduct, and forests for trespassing vehicles is a major concern that can be resolved using similar AI appliances.

Table of Contents

ΠΕΡΙΛΗΨΗ **1**
ABSTRACT **2**
Introduction **1**
Chapter 1 Introduction to ML/DL **2**
1.1 Machine Learning 3
1.1.a) Algorithms used in Learning and Models 5
1.1.b) Evaluation Metrics in ML 10
1.1.c) Organizing benefits and Limitations of ML 15
1.2 Deep Learning 16
1.2.a) Models and Applications 18
1.2.b) Limitations of Deep Learning 21
1.3 Machine Learning vs Deep Learning 23
Chapter 2 Programming Languages, Tools, and Libraries **25**
2.1 The Adopted Libraries 27
2.2 Model Architecture 29
2.3 Evaluation Metrics 32
Chapter 3 Autonomous Recognition of Vehicles and Plate Detection **34**
3.1 Yolov8 36
3.2 License Plate Detector 39
3.3 Execution and Outputs 44
3.4 Improvements and limitations 54
Conclusions **55**
BIBLIOGRAPHY **56**

Introduction

We are living in an era, where technology continues to evolve and seemingly without a stop. Every day, a new discovery and every day a new innovation. It has been almost 60 years since Godron Moore speculated, that " the number of transistors on a microchip doubles roughly every two years" (Gustafson, J.L. (2011). Moore's Law. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston) by extending this speculation, we can deduce, among others, that computing power increases equally. In those years, precision in circuit crafting, algorithmic compleWe xity improved, storing power increased, faster ways to transmit data were invented, CPU/GPU developed and overall advancement in technology and the parts of a computer all led to an even bigger growth in computing power, giving room to a theoretical computer science branch, Deep Learning (DL) to implode and machine learning to climb in popularity to the point that eventually became integral to many widely used software applications and services.

Machine learning (ML) as a concept existed since the early 1950s, when Arthur Samuel of IBM developed a computer program able to play checkers, which he completed in 1955, using a minmax strategy which today is known as the minimax algorithm (Samuel, 1952). ML became popular around 2006 when facial recognition was achieved. As the problems became more sophisticated and the amounts of data too big, the limitation of "traditional" ML practices first appeared. This led to the unpopular DL concepts, which by that time the biggest achievements under its name were Long Short-Term Memory speech recognition and Restricted Boltzmann machines (RBMs), to rise. In the next 10 years, what Roger Parloff described as "a Deep Learning revolution" (Parloff, R. (2016). Why DL is suddenly changing your life. Fortune. New York: Time Inc) had transpired. With the invention of the Internet of Things, Cloud Computing, the Deployment of 5G, Augmented reality, and 3D Printing made AI appliances broadly available and instantaneous.

Nowadays, almost every electronic device carries a circuit board can connect to the internet, communicates with other devices, and produces data. Data once extra, is now an intangible, yet potent resource. Through applied ML and with the aforementioned technologies computers can detect and learn hidden patterns in data. What's more, complex tasks such as driving, can be automated. Object detection and classification, Object Character Recognition, and Face detection all can be achieved in real-time.

This Thesis is about automatic car recognition and harnessing relevant information and aims not only to suggest all the possible use cases of the project but also to give the readers a better understanding of the process. All the tools used to achieve car recognition are explicitly stated. After reading this paper, a person foreign to computer science should have a better grasp of the programming languages that are used for the implementation of Object Detection routines, algorithms used in ML, DL in general, and state-of-the-art models such as ResNet[15], YOLO[16], R-Cnn[17], etc. Throughout explanations on their strengths and weaknesses and also statistical analysis on their accuracy and other key elements for their execution.

Chapter 1 Introduction to ML/DL

This Chapter is dedicated to explaining the general principles of ML, DL, and AI that are used in the project, from the general meaning of classification problems in ML, combined with the DL approach of feature extraction to produce the final AI project. Resolving any confusion of the three similar in meaning, but different fundamentally in approaching a problem, implementing a solution/model and also in the overall accuracy, efficiency and cost while also going deep into explaining how certain practices were used over others. The simple version to keep in mind is that ML, DL, and Neural Networks (NNs) are all parts of AI. NNs are considered a subfield of ML and DL a subfield of NNs.

This proposed system is associated with Automatic Recognition of Vehicles at Intersections and License Plate Detection. It includes state-of-the-art pre-trained models, specifically YOLOv8 as it is best suited for our task, as its core structure, while also building a new simple licence plate classifier using said model structure. Creating the dataset for our license plate model, and creating Ground Truth labels is a tiresome task that also is a part of the process and important nonetheless. It is crucial to state explicitly our bounding boxes in order to ensure that during the training phase, information that best describes Licence plates is provided to the model. All while dealing with other problems that occur in real-time applications such as incomplete License plate numbers, wrong predictions, vizualization, and many more. Finally, the thought process will be discussed behind certain choices. Reasoning why techniques such as finetuning or transfer were not used.

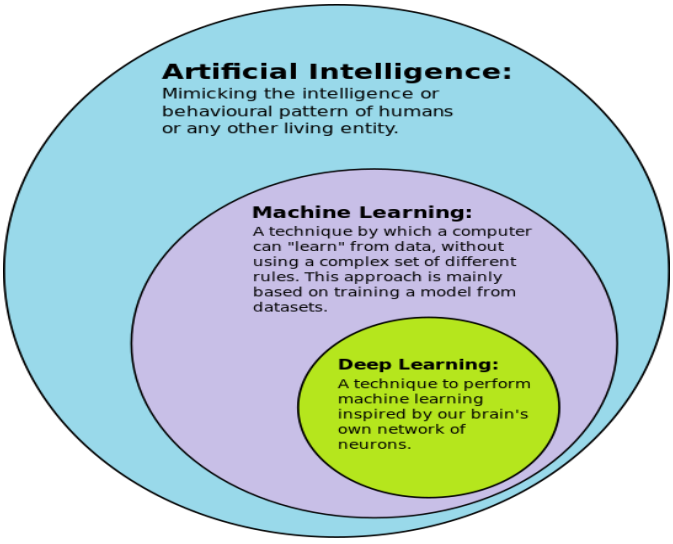

Figure 1. Image describing the relation of AI, ML and DL. In between the ML and DL exist NNs. **(Bandyopadhyay, https://commons.wikimedia.org/wiki/File:AI-ML-DL.png)**

1.1 Machine Learning

ML, as stated by IBM "is a branch of artificial intelligence and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy" (IBM Corporation, 2023). Usually, the procedure towards the solution of the problem is already known or it can be approached using a standardized process. ML has an exploratory nature. It can be used for unsolved problems as long as there is data, that is what makes it such a useful tool. Note that algorithms cannot find solutions or relations which do not exist.

The nature of the problem dictates which model is best suited for describing that problem and each model has its own evaluation metrics. For example, in Binary Classification problems, where the model is trained to point a value to a class with two cases, yes/no, 0/1, open/close, accuracy is the most suited evaluation metric since the aim is, for the model to correctly deduct which of the two cases best describes the data it is given. With time, the problems ML tackles become more complex since layers of predictions and calculations are added, in order to train algorithms, training takes more time, making it harder to design algorithms and leaving room for human error. What made ML so popular is that time-consuming tasks that involve a lot of computations but are simple in nature can be automated.

It has a vast field of appliances, through training on a dataset that describes the problem and the use of statistical methods, well-crafted algorithms make accurate predictions. The algorithms excel as the amount of data rises, establishing a consistent, fast, and scalable tool for predictions. ML is usually employed by companies, functioning as a reliable tool to analyze data, revealing correlations between data, finding rules, spotting anomalies, and other useful attributes on a dataset, all while improving its own predictions. It is important to understand that ML is used when the solution is known, the task is relatively simple or there is a known approach to investigating the data in order to solve the problem. For example, ML recommendation systems used by famous enterprises such as Netflix, Amazon, Facebook etc. employ the Big Data customers generate, offering accurate recommendations and furthermore revealing hidden patterns in the user's preferences. The input data include past purchases, search histories, favorite movies, demographic information, comments or posts etc.

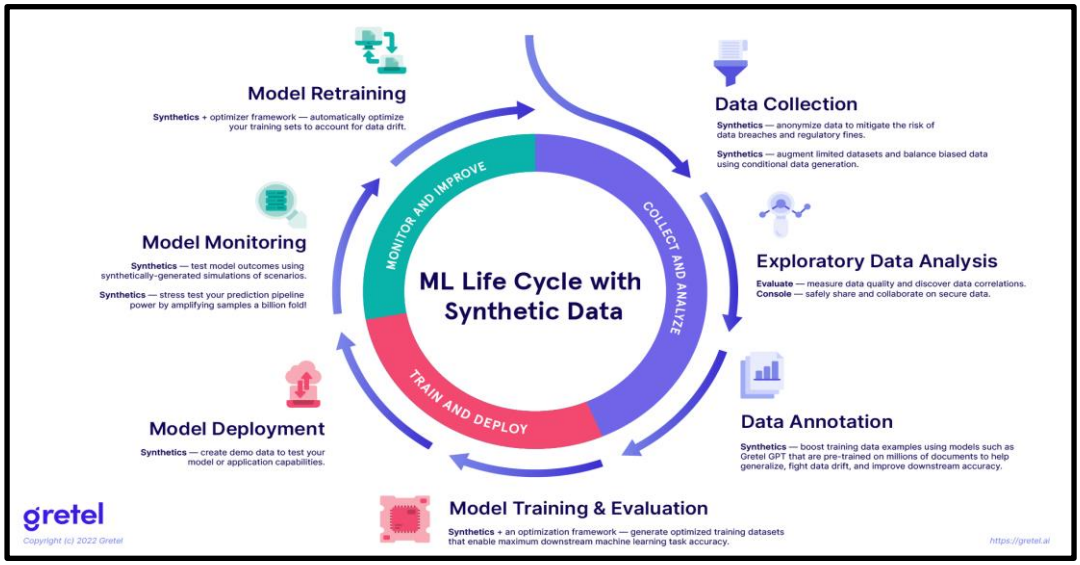

Figure 2. The life cycle of a ML Project. (by gretel.ai)

Although ML has found a lot of success in the business world, in scientific usage there has been a decline in the past few years compared to the early 2010s. Although it seems like an omnipotent tool, it has a lot of limitations. The most important of those are Data, Biases, Overfitting, and other limitations that are not associated with the ability to produce results but rather with the usage of AI. Specifically, the ethical use of AI and ethical data collection.

It was estimated that 90% of the world's data was generated in the last two years [18]. As long as there is a lot of data being produced by all kinds of applications, ML algorithms, and models will continue to evolve as the need for new ways to detect patterns efficiently exists. In the next chapters, we will explore the most relevant models to our project, the different approaches into learning, explain the different evaluation methods, and dive deeper into the limitations of ML.

1.1.a) Algorithms used in Learning and Models

"Machine learning addresses the question of how to build computer programs that improve their performance at some task through experience" (Mitchell, T. M. (1997). Machine learning.)

Before we continue on, it is important that some terminology is clarified. Models are programs that have been trained on a dataset, analyzed patterns, and can make predictions on new unseen data. Those models are trained with specific algorithmic processes based on the data that they possess. There are four types of algorithms, Supervised, Semi-Supervised, Unsupervised, and RL. The key difference between Supervised and Unsupervised is whether or not the data is labelled or not. Semi-supervised is a good median between the two, having a limited amount of labelled data and lots of unlabelled, leading to considerable improvement in the accuracy of the models. Last but not least, many don't include RL as a type of learning and consider it another different category, since it is associated with intelligent agents, is independent of data structure, and uses a different approach that the algorithms use. But, considering it is a learning process to maximize the accuracy of an action done by machines it perfectly fits the criteria.

Our focus in this chapter is to explain in detail the above forms of learning, the models produced, in which situations are adopted, and their evaluation metrics and finally, we will explore the limitations of the ML approach of model building to the core, demonstrating why ML was not used in our proposed system.

In Supervised Learning (SL), the data available are labelled. When feeding a ML model data there is a huge difference between a tabular sequence of numbers named 'Matrix N' and knowing that 'Matrix N' represents a specific Class i.e. 'Daily Temperature'. Knowing the labels helps the creators have better control over the model, raising accuracy in the predictions but as a result, the model is prone to overfitting and biases. Commonly used algorithms are Regression types and Classification types for the most part. The models created are as follows:

Linear Regression Models: These models are used to predict numerical value, given a dataset that forms a linear relationship with the predicted value.

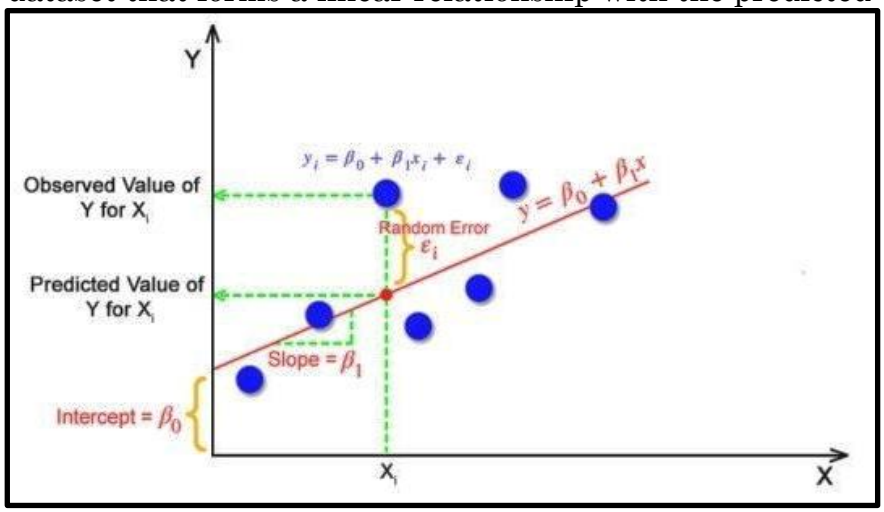

Fig.2. Basic Linear Regression model.

Linear Regression Models are usually prone to outlier errors. Depending on the nature of the problem, outliers are excluded, but if that is not the case, through data normalization or other methods regression models produce fast, accurate results. For example, we could use a Linear Regression model to predict rent prices given data on a house (pieces of furniture, rooms, parking spots, area, etc.).

Logistic Regression: These models are purposed with making categorical predictions such as "0/1". Expanding on the above example, a logistic regression model could be used to categorize house pricing as "Reasonable/Not".

Naive Bayes models: Using the Bayes' Theorem and assuming that all the features in the dataset independently contribute towards the class it is trying to predict, models predict the probability of an input's class. Sentimental analysis, article classification, recommendation systems, and spam filtration are some of the model's applications. For example, given some biometric information about a human, Bayes Classifiers can be used to predict the sex.

Decision Tree models: Tree-based models are used both in Classification and Regression problems. These models form a tree diagram, representing all the possible outcomes, based on their linked decision process. The tree diagram reveals the model's behavior, making adjustments easier, with the trade-off of being not very robust, finding the optimal tree is known to be an NP-complete problem and ultimately prone to overfitting without using pruning. Notable models are Gradient Boosting Regression [19], XGBoost [20], and LightGBM Regressor [21]. Adding to the housing example, a decision tree model could be used for both the prediction of the rent and afterward its category.

Random Forest models: In a dataset, where the results are generated by decision tree models, having a model that predicts based on the results is known as a Random Forest model.

Neural Networks: NNs are usually applied in DL algorithms, by mimicking the synapses of the human brain using different kinds of layers and different kinds of nodes, activation or loss functions, and optimizers, all hyperparameters depending on a model, but more on NNs will be discussed on the DL Chapter. Similarly with Supervised learning models the nodes have inputs, outputs, weights, and biases. The key takeaway is that input data must be labelled to train an NN. NNs are the kind of models used in our project and the kind of models used globally for the most sophisticated and complex problems such as image recognition/segmentation, Natural Language Processing (NLPs), etc.

The strength of Supervised Learning is in its efficiency/accuracy in classifying or predicting data and its simple approach. It can be noticed that the models are susceptible to overfitting and biases. The most important disadvantage is neither of the aforementioned reasons but rather the need for labels and, consequently the constant need for human monitoring. That is something unwanted in an "autonomous task".

In Unsupervised Learning (UL), the algorithms are tasked with finding patterns from unlabelled data. This is achieved by learning inherent structures, patterns, or relationships within of the data. There is still a need for human intervention. Outputs need to be verified and interpreted so the model can be fine-tuned. Without human interpretation of results, unsupervised tasks tend to lack transparency into how data is clustered and have a higher risk of inaccurate

results. Unsupervised algorithms are used to a large extent for Clustering, Association [22], and Dimensionality reduction tasks [23]. The models produced are as follows:

Clustering Models: Assigning the unlabelled data to a cluster is the job of these models. The number of cluster centers may vary depending on the data. The models identify patterns and group the data, iterating until the cluster centers stabilize and the algorithm converges. Some of the most popular models are K-means [24], Hierarchical Clustering [25], DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [26], and Gaussian Mixture Models (GMM) [27]. One example is, clustering purchases in an e-shop and using the results for market segmentation.

Association Models: Models that use different kinds of rules to discover associations between the input data. Commonly used for recommendation engines, promotion optimization, and overall better product placement. For example, such a model could be deployed on the same e-shop sales data, revealing relationships between purchases such as computer parts usually purchased alongside cables.

Anomaly Detection: Inherently unsupervised learning is very competent in detecting outliers, and points in data that deviate from a dataset's norm. These points, cause fluctuation in data, leading to inaccurate outputs in cases of predictions, or used to indicate an event or observation that beckons actions. Manufacturing, fraud detection, financial transactions, and system monitoring are some of the fields that utilize anomaly detection.

Dimensionality Reduction Methods: Data is indeed the 'fuel' of models, but too much data in reality can have a negative impact on the performance. Leading to overfitting, difficulty in comprehension of model functions, hindrance in visualization, and expensive training times. In those cases, dimensionality reduction is a technique used to reduce the number of features or dimensions. The models aim to reduce the amount of data while preserving the contents. Such models are Principal Component Analysis (PCA) [28,30], Singular value decomposition (SVD) [29], Autoencoders [30,31] etc. For example, in the case of a multinational company, before using the data in any model, PCA could be used to lessen the burden of the model and avoid overfitting.

Being able to extract features from unlabelled data, and detect outliers while being a scalable process, unsupervised learning is a powerful tool for predictions and associations. Consequently, it needs huge amounts of data, making it computationally expensive. Furthermore, data surplus makes interpretation of the outputs confusing, and the learning process difficult to monitor while drawing conclusions. Last but not least, with the absence of the Ground Truth labels, evaluation can easily become subjective, making human intervention necessary to validate the output results.

Semi-supervised learning (SSL) is a "median" approach, combining the strengths of Supervised and Unsupervised Learning. The use of a small labelled dataset, to classify and extract information on bigger unlabelled data, lowers the overall cost. Having an SL model evaluate the outputs of a UL model reduces human participation and, while being cautious with novelty detection during exploratory analysis. These models can handle labelled data points with traditional SL, making predictions, calculating loss, and updating weights using

gradient descent (GD). Such a model is FixMatch [32]. Usually, SSL is used to solve complex problems, such as self-training, speech recognition, Image classification, etc.

In conclusion, there can still be reliability issues since there is no method to generate 100% accurate labels which leads to less well-grounded results than a conventional SL model in terms of accuracy. The reduction in production costs is noteworthy. The most significant weakness of this approach is the human factor since fine-tuning and data selection procedures require skilled engineers.

Reinforcement Learning (RL) [33] is the field related to intelligent agents, functioning similarly to SL, but ultimately there is a key difference in the learning process. While SL uses sample data and weights to maximize the prediction accuracy, RL models learn through trial and error and are defined by processes in an environment, a "reward", a policy, and their state. Given a goal/task, an intelligent agent is trying to achieve the goal while maximizing its rewards, each action depending on the current state of the environment. The rewards fluctuate, depending on the state and the action taken which is calculated through the "value" function. There is no need for labelled data since the models operate in an environment and consider two actions, exploration and exploitation. Ultimately, the agent learns through repetition, which actions to choose in order to maximize its cumulative reward. The simplest daily example of an RL application is that of an autonomous navigation system used by autonomous sweepers.

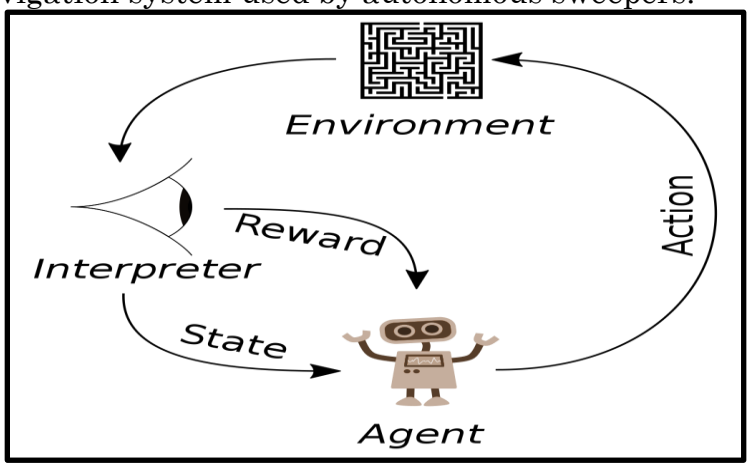

Fig.4 Depiction of a typical RL model layout.

There are three main types of learning in RL, policy-based, value-based, and Actor-critic.

The Value-Based type is trying to find the optimal value function. The output of the value function is the upcoming reward and the reward function estimates the current reward. The agent prefers actions that maximize the value function. Such algorithms are Q-Learning [34] and Deep Q-Learning (DQN) [35]. In the autonomous sweeper example, our robot would prefer paths that yield the best outcome while making the smallest route.

In the Policy-based approach, the agent will prioritize exploration and adjust its policy based the reward function, adjusting the policy while learning. The most used algorithms are Policy Gradient and Proximal Policy Optimization. Expanding on our sweeper example, the robot would explore rather than choose

the “best” action, adjusting its behavior depending on the reward of the exploration.

Last but not least, the Actor-Critic type is a combination of the two. The policy plays the role of the “actor” and the value function of the “critic”. The policy dictates the agent's choices while the value function evaluates the expected cumulative reward. This forms a balance between exploration and exploitation. It enables agents to learn based on the environment while keeping the best course of action strategy available, thriving in dynamic environments.

All in all, the above summarizes quickly the basics of ML and the majority of algorithms and existing models. In hindsight, they may seem a lot and some even trivial to the object of the project, but keep in mind that DL is a subset of ML, and in building a complex AI model all things must be taken into consideration for an optimal result.

1.1.b) Evaluation Metrics in ML

As the term suggests, an evaluation metric are quantitative measures used to assess the performance and precision of a statistical or ML model. Metrics are usually visualized, provide useful insights during execution, and help compare each model or algorithm. Specifically, metrics can measure and track the process of predictive abilities, performance in unseen data, and learning. The selection of a metric depends on the nature of the problem, data formats, and the outcome it is intended to produce. There are two types of predictions ML models produce, classification which can be numerical in the form of probability or binary form representing a class, or regression which are continuous outputs.

To begin with one of the most important ones, which will be presented in our model as well, is Confusion Matrix. Confusion Matrix is a squared matrix N*N, where N is the number of all the prediction classes. In the case of a Binary Classification model, the N is 2 whereas in the case of a Multiclass-Classification problem, N will be the classes we are trying to predict. Creating the Matrix reveals some important statistical data for each model. These data are:

True Positive(TP): Positive prediction, ground truth true.

True Negative(TN): Negative prediction, ground truth true.

False Positive (FP): Positive prediction, ground truth false.

False Negative(FN): Negative prediction, ground truth false.

Accuracy: Total number of predictions that were correctly divided by the total amount of predictions.

Precision (positive predictive value): the number of positive predictions correctly identified divided by the number of total positive predictions.

Recall (sensitivity): The number of True Positives predicted in all the True Predictions.

Specificity: The number of True Negatives predicted in all Negative Predictions.

		Predicted condition		Sources: <ref>	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**	Informedness, bookmaker informedness (BM) = TPR + TNR − 1	Prevalence threshold (PT) = $\frac{\sqrt{TPR \times FPR} - FPR}{TPR - FPR}$
Actual condition	**Positive (P)**	**True positive (TP)**, hit	**False negative (FN)**, type II error, miss, underestimation	True positive rate (TPR), recall, sensitivity (SEN), probability of detection, hit rate, power = $\frac{TP}{P}$ = 1 − FNR	False negative rate (FNR), miss rate = $\frac{FN}{P}$ = 1 − TPR
	Negative (N)	**False positive (FP)**, type I error, false alarm, overestimation	**True negative (TN)**, correct rejection	False positive rate (FPR), probability of false alarm, evaluation measures fall-out = $\frac{FP}{N}$ = 1 − TNR	True negative rate (TNR), specificity (tests) = $\frac{TN}{N}$ = 1 − FPR
	Prevalence = $\frac{P}{P+N}$	Positive predictive value(PPV), precision = $\frac{TP}{PP}$ = 1 − FDR	False omission rate (FOR) = $\frac{FN}{PN}$ = 1 − NPV	Positive likelihood ratio (LR+) = $\frac{TPR}{FPR}$	Negative likelihood ratio (LR−) = $\frac{FNR}{TNR}$
	Accuracy (ACC) = $\frac{TP+TN}{P+N}$	False discovery rate (FDR) = $\frac{FP}{PP}$ = 1 − PPV	Negative predictive value (NPV) = $\frac{TN}{PN}$ = 1 − FOR	Markedness (MK), deltaP (Δp) = PPV + NPV − 1	Diagnostic odds ratio (DOR) = $\frac{LR+}{LR-}$
	Balanced accuracy (BA) = $\frac{TPR+TNR}{2}$	F_1 score = $\frac{2PPV \times TPR}{PPV+TPR}$ = $\frac{2TP}{2TP+FP+FN}$	Fowlkes–Mallows index (FM) = $\sqrt{PPV \times TPR}$	Matthews correlation coefficient (MCC) = $\sqrt{TPR \times TNR \times PPV \times NPV} - \sqrt{FNR \times FPR \times FOR \times FDR}$	Threat score (TS), critical success index (CSI), Jaccard index = $\frac{TP}{TP+FN+FP}$

Fig. 1 Terminology and derivations from a Confusion Matrix (https://en.wikipedia.org/wiki/Confusion_matrix).

The overall accuracy of a model is calculated as the sum of correct predictions divided by the total predictions made. Frequently, accuracy alone is not the point of interest, but the categories that the predictions fall in depending on the problem. For example, in a model predicting if a patient has COVID-19, an FP prediction doesn't have the same consequences as an FN prediction, thus wanting to minimize FP we would optimize the model in a way to maximize Recall and so

on. The Confusion Matrix is a versatile metric that provides clarity to the engineer and helps with the visualization of the models. With this in mind, the matrix is mainly on Classification models.

Deriving from the information extracted from a Confusion Matrix, Recall and Precision are the metrics, models try to maximize. Unfortunately, because of the Recall and Precision trade-off, improving one of the two comes at the expense of reducing the other, it is impossible to for a method to strengthen both. F1- Score (1) is the metric that finds the best precision and recall values. The formula is presented in the following image.

$$F_1 = 2 * \frac{2TP}{2TP + FP + FN} \quad (1)$$

The reason behind using a harmonic mean between the two values, and not an arithmetic mean, is so that the result is not heavily influenced by outliers. For example, in case of a Binary Classifier with Precision:0, Recall:1. Using an arithmetic mean would give us a value of 0.5. Leading us to believe that the model has some functionality. Instead, if we use F1-score the value is 0. Correctly representing the uselessness of the model predictions. There are some cases in which outliers are important to the process. For that the following formula can be used instead.

$$F_\beta = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall} \quad (2)$$

Fbeta (2) measures the effectiveness of a model with respect to a user who attaches ß times as much importance to recall as precision.

Area Under the Roc Curve (AUROC) is another popular metric that is used to measure the performance of models. The terms used in the AUC-ROC curve are TPR / Recall / Sensitivity at various threshold settings. Each Roc curve describes the probability of a singular class.

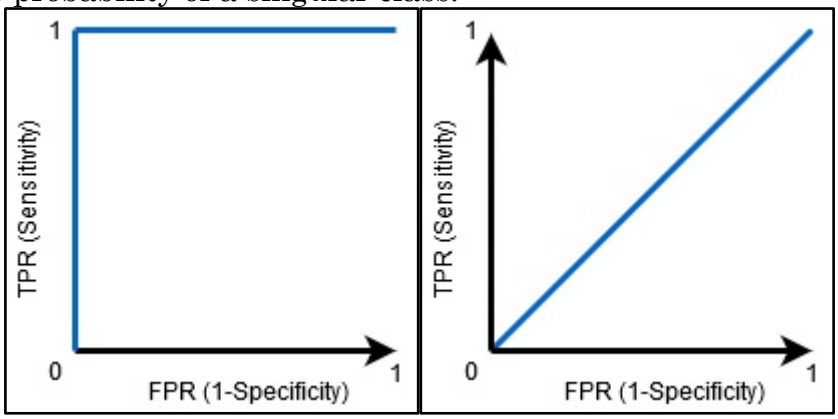

Fig.1 Perfect model AUC = 1 **Fig.2 Useless model AUC = 0.5**

AUROC represents the degree of separability. In Fig.1 we see a depiction of a perfect model and in Fig.2 the depiction of a model that cannot distinguish the class and makes random predictions. Typically, AUC above 0.7 is considered a fair prediction, depending always on the problem. In the case of multi-class models, we plot N number of AUROC curves where N number of Classes, using the One vs All methodology.

Gini Coefficient is a metric derived from AUC ROC. It describes the ratio between under the ROC curve and the diagonal line and the area above the triangle. For more clarity, Gini (3) is calculated using the formula below:

$$Gini = 2 * AUC - 1 \quad (3)$$

A value above 0.6 indicates a good prediction.

Moving forward from the Classification metrics, Root Mean Squared Error (RMSE) (or Deviation) is the most famous evaluation metric used in Regression models. It is quite sensitive to outliers, so preprocessing in the form of standardization is advised, and punishing towards big deviations from the ground truth, RMSE (4) is considered a robust tool.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(predicted - actual)^2}{n}} \quad (4)$$

Again, there are different variations of RMSE depending on the data and outputs. In the case of a model used on big numbers, a way to not penalize the deviation as much is Root Mean Squared Logarithmic Error (RMSLE). Additionally, RMSLE treats errors relatively, in numerical inputs of billions, the error of a couple hundred points is almost irrelevant.

As opposed to classification models, in which an accuracy value of 0.8 is enough to judge that our model is performing well as opposed to a random prediction model with an accuracy of 0.5. The random model is treated as a benchmark. Regression models however don't have that guideline for comparison. R-Squared (the coefficient of determination) metric compares our model with that of a model predicting the mean value of our target from the train set. The formula (5) is:

$$R^2 = 1 - \frac{MSE\ (model)}{MSE\ (baseline)} \quad (5)$$

Where MSE is our model Mean Squared Error against the real values and the baseline model mean prediction against the real values respectively. The value of

the metric can be any real number, with 1 representing a perfect fit, negative numbers worse prediction that the MSE (baseline), above 0.5, and bellow 1 indicate satisfactory outputs and numbers above one indicate an error in computations.

In continuation, Adjusted R-Squared [36] has the additional functionality of showing where the addition of extra features to the models is beneficial or harmful. Looking at the formula, incrementing k reduces the denominator, increasing the whole expression. In case, AR-Squared (6) doesn't increase together with k, the feature added doesn't contribute towards the outputs.

$$Adjusted\ R^2 = 1 - (1 - R^2)\left[\frac{n-1}{n-(k+1)}\right] \quad (6)$$

Last but not least, Cross-validation is more of a technique rather than a metric. It is a resampling method that uses different portions of the input data to test and train a model iteratively. Suppose that the input data are split into 10 pieces. Each iteration produces a new model, which is trained on different 9 pieces in each iteration and validated on the remaining one. This aims to flag problems like overfitting and selection bias, by checking if all the 10 models are close in terms of average accuracy. To be precise, the method described above is K-fold Cross-validation.

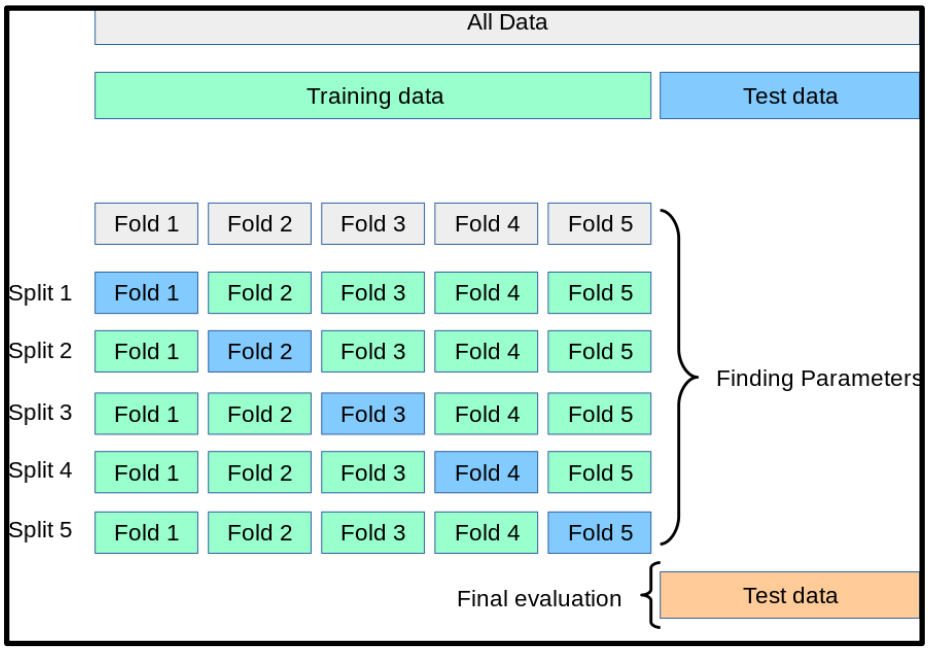

Fig.3 K-fold cross-validation depiction

In conclusion, in any project evaluation metrics is an essential aspect. The metrics provide valuable insight into the model's performance and quality. The ones used in classification problems, which is the type of the project, are Confusion matrix, AUROC and overall accuracy.

1.1.c) Organizing benefits and Limitations of ML

ML excels in small, simple applications and doesn't require big quantities of data in order to produce accurate predictions. It can perform well with and without labels and exploit outliers when needed, disregarding them elsewhere. ML is a great tool and has a wide range of applications that are employed almost in every industry. When it comes to high volumes of data, it is almost unavoidable for a model to overfit and show biases. Especially when models, obtain part of their data from humans. Some examples to demonstrate the big problem of bias are Microsoft's Tay Chatbot, Google Photos categorization of black people as gorillas in 2015, Amazon's Hiring AI in 2018 which penalized the keyword "women" etc.

The key weakness lies within the learning approach. ML learns using the labels/attributes and not so much the structure of a given object. Furthermore, ML utilizes structured data, that can only be used for their respective specific purpose, which limits its flexibility and usability. The production cost of structured data is too high and there is not much reusability. In every new model build, the engineer needs to adjust the model depending on the structure of the data, apply preprocessing procedures, and fine-tuning for that specific purpose. This leads to the next problem, human costs. ML models depend a lot on human intervention, which is expensive and unwanted, especially in complex projects. As the complexity rises, ML models are computationally hungry, requiring a lot of processing power. All that with minimum transparency and interpretability to its training process.

Lastly, there are the ethical problems. The two most important ones are Violation of Privacy and Unethical use. As it is understood, models often categorize human activities, data mine their preferences, and actions to offer high-quality predictions. The main conflict falls into the category of 'who is in possession' of that data, the safety of the data, and misuse of conduct by big companies. Secondly, the cases where AI Chat models have been used for the production of malware, weapons, and other harmful applications such as WormGPT.

In general, ML scales poorly no matter the approach but its usefulness is undisputed. Ethics in AI is a field to be considered in the future since it is not hindering the technology directly. The main reason ML fails is due to overfitting, as a method it is not suited well for huge amounts of data. It is suited best for small applications rather than complex ones.

1.2 Deep Learning

Once again, for reasons of comprehensibility, it is vital to understand that NNs are the cornerstone of DL. Model are based using some type of NNs that focus on understanding a task on a very "deep" level. So, DL is a subset of ML, which works with NNs that consist of 3 layers or more layers. The term DL was introduced to the ML community by Rina Dechter in 1986, and to artificial NNs by Igor Aizenberg and colleagues in 2000, in the context of Boolean threshold neurons. A pioneer in DL is Geoffrey Hinton, who in his career has countless contributions to the field. NNs attempt to mimic the behavior of the human brain. In order to be trained, large amounts of data are needed and as a result powerful processing hardware is also required.

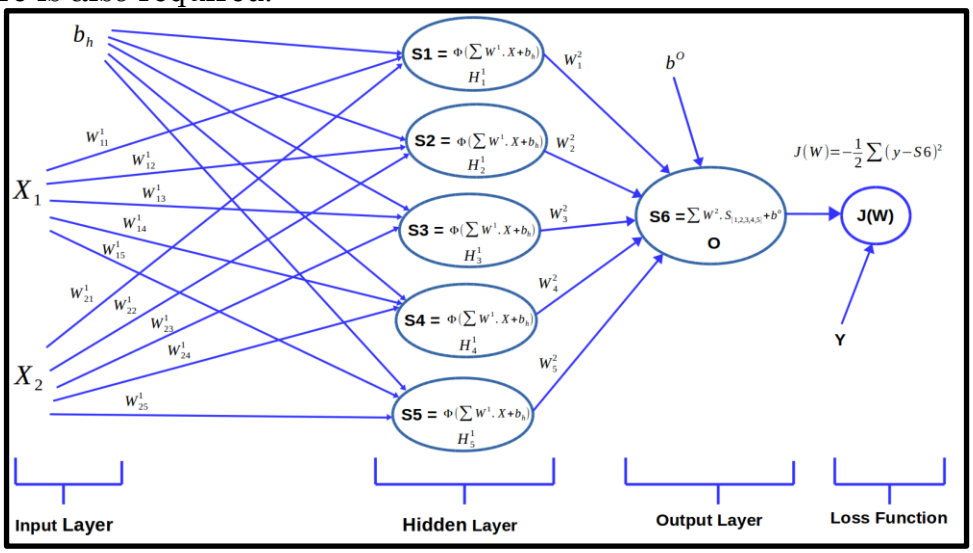

Fig 1. A basic NN with three layers. (not including Loss Function)

DL is used to create a lot of today's applications and services, that contribute towards automation by eliminating physical tasks without human interference. Some of the sort are autocorrect text editors, voice-command applications, self-driving cars, face recognition, etc. In order to train NNs, data in raw forms is used, such as raw text or images, forming weights and biases to produce a prediction. The layout of a typical NN consists of an input layer, a hidden layer and an output layer. There can be a NN with a single layer, but extra layers help with feature extraction. Each layer contains interconnected nodes, build upon the previous layer in order to refine the data extraction process. This process is called forward propagation. For example, consider images of animals (cats, dogs and horses) in a simple NN having a hidden layer with 3 nodes where each is trained and tasked with recognizing one category of animal based on its facial features. The nodes output a value which is a weighted probability, and it describes whether the input fits the data that were trained to identify. The input and output are called visible layers and all the between are called hidden layers. Another important process is backpropagation, which like gradient descent is used to calculate errors in predictions, and is used for fine-tuning the model. Combining Forward and back propagation is what makes the Network learn.

The above series of steps describe the simplest model possible, where in reality NNs are extremely complex. Building a model and finding new ways to build

models is what DL primarily focus on. In this chapter, we will investigate the different types of DL NNs and their various applications.

1.2.a) Models and Applications

There are many types of NNs and even more models are built around them daily. The rapid growth in number of the models is caused by the constant need for improvement. Even if it is by a margin of 5%, it is enough to be called a new model. DL models are generally used in the field of Computer Vision, Speech recognition, and Natural language processing (NLP). Additionally, since DL is tasked with making predictions in complex applications, a lot of layers from different types of NNs just blend together. For example, our project falls into the single category of Computer Vision as it is but if we wanted to add more functionality to it, it would be hard for it to be categorized, i.e., self-driving cars. The main focus of this chapter is to present some of the basic types of NNs. Their respective properties differentiate them, and their applications.

Convolutional Neural Networks (CNNs) are composed of an input layer, an output layer, and one or more hidden layers. Their key difference is that the neurons are arranged in a three-dimensional form representing the width, height, and depth of dimensions. This way, it allows for 3D inputs to be transformed into an output volume.

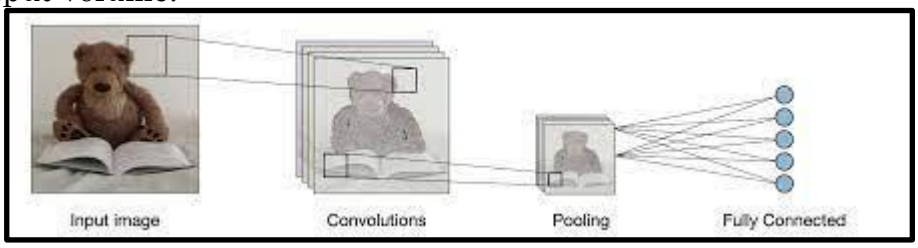

Fig 1.2.1. Example of a CNNs function.

The hidden layers contain combinations of convolution layers, pooling layers, normalization layers, and fully connected layers. CNNs are widely known for their feature extraction ability, they are not only used in Computer Vision, in contrary with popular belief, but in every field (NLP, speech recognition, processing systems, etc.).

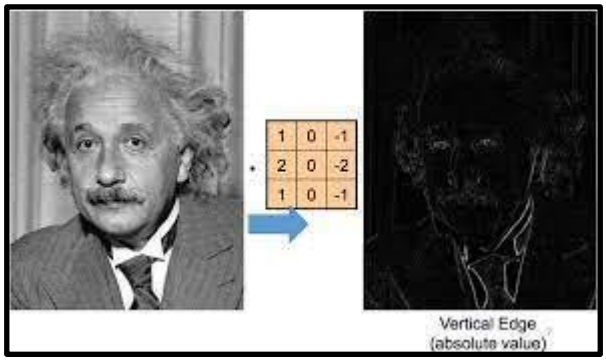

Fig 1.2.2. Example of feature extraction using Convolutions (Vertical convolution)

Each model created using CNNs is not the same. There are key differences, varying between the desired outputs and execution time, of course, both while maximizing accuracy. For example, some models focus more on object detection precision (ResNet) while others try to minimize execution time with the best possible accuracy (YOLO). The most notable ones that were taken into consideration for this project were YOLO, ResNet, VGG, Faster R-CNN, and SSD.

Following, Recurrent Neural Networks (RNNs) are a generalization of feed-forward NNs that have internal memory. Used mainly for time series problems, this model leverages the past outputs to predict each new input. In other words, for every input of data, the output of the current input depends on the past computation.

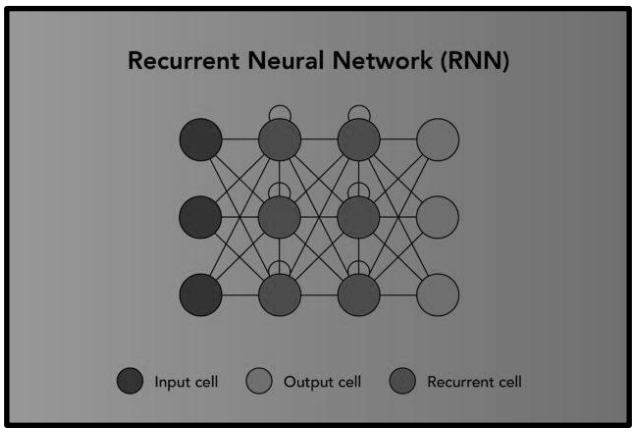

Fig 1.2.3. Example of an RNN structure.

Each output is copied and fed back to the RNN. RNNs have many applications in stock market predictions, text processing (Grammar corrections, Co-writer bots, Translations, etc.), and also as Sentiment Analysis. It performs best when combined with sequential data. Each new sample of data is assumed to be dependent on the past ones and that together with the iterative nature is this model's key feature. The disadvantages of RNNs are Gradient vanishing and exploding problems, which translates into Poor Performance, Low Accuracy, Long Training Periods for vanishing issues and identity Initialization, Truncated Back-propagation, and Gradient Clipping for the exploding issues respectively. Ultimately leading to difficulties in training RNNs. In order to solve the vanishing gradient as an issue an improved type of RNN was created, Long Short Term Memory (LSTM).

Lastly, Generative Adversarial Networks (GANS) are the last NN we will look into, due to their useful applications. The main concept in a GAN is of two NNs contesting one another in the form of a game. Given a training dataset, one NN assumes the role of a generator and the other the role of a discriminator. The model aims not to procure good examples on the trained data, but rather compete in which NN performs better. The end results are synthetic data produced by the generator which are able to fool not only the discriminator but the human eye as well. Since the model is eventually able to produce realistic high-quality results, from images to videos, it has a wide range of applications as a generative tool, and that is only focusing on the generative NN. GANs started as a model to generate data for unsupervised learning tasks but also proved useful for semi-supervised learning, fully supervised learning, and RL.

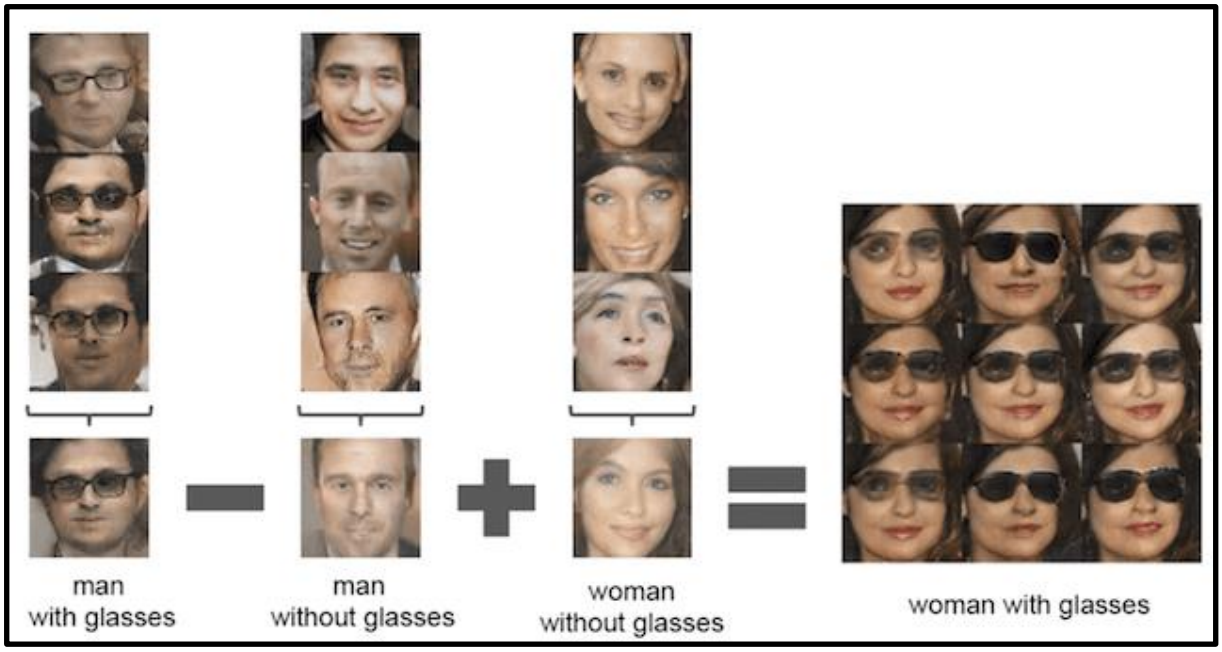

Fig 1.2.4. Example of GAN-Generated Faces. (Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015.)

In Computer Vision, other than synthesis of new images, GANs can be used to filter through unwanted noise in the data preserving the original. Such an example is removing the rain from an image (H. Zhang, et al. 2020, Image De-Raining)

These are only some of the ANNs that DL has to offer. Some are straightforward like GANs and RNNs while others are memorable for their complexity and lack of transparency during training (CNNs). One thing is for sure, the appliances of these algorithms are vast and invaluable. It is important to note, that there are other NN algorithms equally important, that this section only scratched the surface of, namely Multilayer Perceptrons, Autoencoders, Modular NNs, and Feed-forward NNs. Next, the focus will be shifted towards the various limitations of DL.

1.2.b) Limitations of Deep Learning

DL has numerous capabilities as a tool, but it is far from perfect, it comes with a lot of disadvantages no matter how impressive the results it produces. From their data hungry nature to their fitting title as "black boxes" and their danger of overfitting nonetheless. It is difficult to discern which is more important depending on the problem, importance changes. What all NN have in common is the need for enormous computation power, measured in flops (7), for its training.

$$FLOPS = cores * \frac{cycles}{seconds} * \frac{FLOPs}{cycle} \quad (7)$$

To begin with the models limitations, NN models consist of numerous layers, when building a new model and there is a need for a better performance at a specific sub-task the first idea would be to add more layers, which is incorrect. Adding more layers, may bring more functionality to the sub-task but it can also burden the model with overfitting and ultimately reduction in accuracy. This falls directly under the lack of interpretability. DL models, especially those with high complexity and many layers are hard to understand the prediction process. Hence, hard to pinpoint errors or biases, very important fault when the task is associated with human lives. Those Errors can belong either to Data type or Structural type. Data type errors caused by the Quality of data, noisy, incomplete or biased data can affect the performance negatively. Additionally there can be insufficient amount of data the NNs need, which is easily identified plotting each model's learning rates. Structural type, omitting the lack of transparency, can also be caused due to the complexity of the problem combined with the lack of prowess. Any NNs in order to be trained require significant computational power as well as for execution. This is not a task for an everyday computer, it requires powerful GPUs and enormous amounts of memory, making it environmentally expensive. In a recent research, it was estimated that in order to minimize the error to 5% of the famous ImageNet model it would take "an additional 567x more computing power" [3], suggesting that focus should be shifted towards algorithmic improvement to reduce the computation intensity and ML's computation efficiency other than the current DL approaches.

Last but not least, what NNs learn are relationships between variables, which may or may not have underlying connections or represent features. NNs lack in contextual understanding, which can be very limiting. Finding ways to contextualize raw information is something to be considered [4], while it could be argued as implausible taking into consideration data biases.

Benchmark	Error Rate	Polynomial			Exponential		
		Computation Required (flops)	Environmental Cost (CO_2)	Economic Cost ($)	Computation Required (flops)	Environmental Cost (CO_2)	Economic Cost ($)
ImageNet	Today: 9.00%	10^{23}	10^{5}	10^{6}	10^{24}	10^{6}	10^{7}
	Target 1: 5%	10^{26}	10^{8}	10^{9}	10^{30}	10^{13}	10^{14}
	Target 2: 1%	10^{33}	10^{16}	10^{16}	10^{92}	10^{74}	10^{75}
MS COCO	Today: 38.7%	10^{22}	10^{4}	10^{5}	10^{22}	10^{4}	10^{5}
	Target 1: 30%	10^{23}	10^{6}	10^{6}	10^{24}	10^{7}	10^{8}
	Target 2: 10%	10^{31}	10^{13}	10^{14}	10^{49}	10^{31}	10^{32}
SQuAD 1.1	Today: 9.4%	10^{22}	10^{4}	10^{5}	10^{22}	10^{5}	10^{5}
	Target 1: 2%	10^{29}	10^{11}	10^{12}	10^{51}	10^{34}	10^{34}
	Target 2: 1%	10^{32}	10^{15}	10^{15}	10^{88}	10^{70}	10^{71}
CoNLL 2003	Today: 5.4%	10^{23}	10^{5}	10^{6}	10^{24}	10^{6}	10^{7}
	Target 1: 2%	10^{39}	10^{22}	10^{22}	10^{61}	10^{43}	10^{44}
	Target 2: 1%	10^{50}	10^{33}	10^{33}	10^{120}	10^{102}	10^{103}

Economic cost from [3]

In conclusion, there are three main issues with DL models. The high complexity and lack of ambiguity when it comes to training, predictions and overfitting. Secondly, an enormous quantity of data is required for training. Lastly, training is computationally costly with no means of improvement, currently. Also, a lot of research and practice is required to gain experience with models and their operation functions. These are the walls faced when choosing to work with DL and NNs.

1.3 Machine Learning vs Deep Learning

In computer science, terms are used interchangeably, leading to profane confusion. By now, there should be no confusion, only the understanding of each technology's approach to building models, their core, use cases, outputs, and limitations. Slowly leading to a more concrete conclusion to their key differences, strengths, and weaknesses. As the need of DL and ML models is constantly rising and will continue to grow, from an engineer's perspective, there should be no hesitation about which technique to choose for each presenting problem.

Traditional ML models are chosen when the data available is scarce and there is a foolproof path towards the solution. Which can also be translated as the problem isn't very complex in nature. Building and training small models is cheap computationally and requires minimum human maintenance. Also, their accuracy outperforms those of DL due to the lack of data. Mind that ML's scalability is very impotent. On the other hand, DL is chosen when the data available is enormous in size and the tasks are rigorously complex. Can be used even if the understanding of the problem's domain is insufficient due to its extraordinary exploratory capabilities. The technique stands out when it comes to dealing with multidimensional problems (Images, NLP, speech recognition, etc.) provided that one has enough high-end infrastructure to train it. DL will always outperform ML model's accuracy given enough data and is also easily scalable. They are harder to fine-tune and very costly to build.

Both, ML and DL, lack clarity when it comes to how they produce results, but being able to visualize the training process and predictions helps out a lot. From that perspective, ML has an advantage over DL since it is employed for relatively small problems.

All in all, it falls down to data, complexity, and available resources. DL is best suited for handling high-complexity decision-making recommendations, speech recognition, image classification, etc. in essence, large-scale problem-solving. On the contrary, ML is suited for small simple decision making, recommenders, predictions, classification, and dimensionality reduction. Fast and efficient when errors are permissible and include no mortal threat.

Chapter 2 Programming Languages, Tools, and Libraries

Onward with the hands-on stuff, an important role in creating the project is the ability to choose among the right programming tools. In our case, it is obvious enough that we are going to use High-Level languages to create the model. Using a low-level programming language can have its perks in terms of compilation speed and faster computations, but setting up the whole NNs structure would be very time-consuming at the very least not taking into consideration the complexity. Build-in garbage collector is one of the other convenient tools high-level languages offer that is necessary for AI applications. Nowadays, aside from the already established programming languages such as Python, Java, JavaScript, C++, C# there are others worth mentioning such as R, Scala, Julia, and Ruby always taking into consideration the deployment boundaries of the project. There are a lot of variables to contemplate when choosing a programming language for any project such as the scale, available resources, etc. This introduction aims to specify a majority of the existing programming languages, used for AI applications. Briefly mention their core distinctive features, explaining the thought process behind them.

Starting with Java which is a very powerful language, easy to write, easy to debug and to top it off multiplatform. Especially prevalent in the mobile app department as well, Java is a prominent candidate, with built-in garbage collector and many many more tools like Swing, Standard Widget Toolkit, and many libraries using Deeplearning4j (DL4) as their base. DL4 is a major open-sourced DL library that is written on Java and used for a variety of applications such as network intrusion detection, cybersecurity, anomaly detection in industries such as manufacturing etc. Java is and will always be a core option to consider in every kind of project, not only AI-related.

Next on the list is C++, truthfully C++ isn't amongst the most popular of choices but it is widely used throughout the industry. C++ has exceptionally fast compilation and also is very efficient, in exchange for being a complex language. OpenCV (OpenCV. (2015). Open Source Computer Vision Library), one of the most important computer vision libraries available is written in C++. In most cases, C++ is used together with other languages to build AI applications but not as the core language, but rather having a supplementary role.

A relatively new programming language Julia is also a potent tool. Since its first version in 2018, Julia has been growing rapidly with AI popularity. The main reason Julia is preferred is because it contains a built-in package manager and support for parallel and distributed computing, a valuable asset for any AI project. Its main library of interest is Flux [37] a ML and AI stack. Julia is another possibility to consider, using its built-in parallelism, offering easy scalability combined with cloud computing. Unfortunately, the language still has room to grow in terms of community and libraries, in contrast to the other alternatives.

The next language to take into consideration is LISP [38]. Like C++, LISP isn't used to develop modern AI applications but enables prototyping through its effective processing of symbolic information. The different syntax LISP uses and

the lack of modern well-developed libraries is what makes other languages eclipse LISP.

Last but not least, the most famous and the one we will be using, Python [39]. Python may be the slowest, when it comes to its execution time and compiling time, from all previous choices but it comes with great advantages. From the easy syntax, making writing clear code and debugging it relatively easy, to its wide selection of packages and libraries backed up by an enormous and active community. Additionally, it has a variety of API frameworks and IDEs to choose from, it is versatile and flexible, and also easy to interact with other applications and code. What Python lacks in speed it makes up for with its community and packages. One of the many examples, Cython is a module that translates Python code into C. It has its limitations but combined with other libraries such as Numpy is able to produce several times faster compile and execution times, overcoming Python's own shortcomings.

In conclusion, it is not just one programming language dominant over the others, as always each language comes with its own pros and cons. It is healthy, for the whole AI development application field to have variety, in order for progress and innovation to bloom. Moreover, the existence of modern libraries is important to build an application and also an active community in order to maintain it.

2.1 The Adopted Libraries

As mentioned, Python has a wide variety of libraries to choose from in order to develop an AI model. Namely among the best is Numpy, Tensorflow, Ultralytics, Pytorch etc. In order to develop a stable and reliable application, the existence of libraries is important, and also for maintaining them operational and future proof. Vehicle detection can be done in a stream processing manner, supposing we have the necessary resources available, GPUs with high VRAM, ten thousand plus cores and at least 300 TPUs, or by Batch Processing in our case.

First and foremost, since image processing is one of the most frequent task this program is going to perform, OpenCV is one of the most important libraries. Opening video files, capturing frame by frame, concatenating, perform pixel related operation such as grayscaling, bitwise_not and threshold selection. Although there are other libraries out there to consider like scikit-image, SimpleCv or TensorFlow, due to its C++ base code, OpenCV in Python is unrivaled.

Continuing with Pandas [41] and Numpy [40], almost in every scientific program there are always reasons for Pandas and Numpy to be used, and those reasons are data processing. Pandas is a library built on Numpy, that specializes in every form of data manipulation, from saving to manipulation, Pandas handles tabular formats and performs operations efficiently. On the other hand, Numpy is used exclusively for mathematical operations on arrays. Pandas is a data processing library and Numpy is a computational library specialized in arrays, both combined provide powerful data structured functions, that are used for data analysis (Pandas) or scientific data computation (Numpy). Our extracted data will be saved into .csv format file, holding important information such as frame number, car ID, bounding boxes, confidence, license plate etc. Most AI public libraries depend on those two libraries in Python for data related task.

We are going to need a library for model deployment and for that ultralytics was used. Since we are going to use already existing archetypes rather than fine-tune or deploy new ones, ultralytics is a good fit, simplifying the training process into a few lines of code, while providing all the necessary information about our model's training process such as Learning rate, gradient descent, etc. In case there was any reason to modify the model Tensorflow or Pytorch would be the library of choice. In particular, Pytorch since Tensorflow has some dependency problems and is infamous for its incompatibility with other frameworks and APIs.

For training the license plate detector cuda acceleration was used but not for prediction generation, since we wanted to test the inference speed.

Lastly, the dataset used for training the LSP was obtained through roboflow. Roboflow [42] is a wrapper around the Roboflow REST API, providing abstract methods for interacting with Roboflow in Python code. The Python package offers methods for managing projects and workspaces, uploading and downloading datasets, running inference on models, uploading model weights, and more.

With Python as the programming language of choice, there are plenty of ways to go around all the different parts of our project. With that being said, in the next chapter we will go into more detail about the models that were used.

2.2 Model Architecture

As seen on the previous chapter of DL, the most fitting algorithms for Computer Vision, and object detection in particular, are CNNs. A traditional object detection pipeline consists of three main stages, Region Proposal Generation, Feature Extraction, and lastly Classification. Current modern models have evolved and are either two-stage or one-stage type. A two-stage architecture (1) produces region proposal (Region proposal network RPN), by importing ML methods or DL, followed by (2) Object Classification on the features extracted by (1) from the proposed regions. These algorithms achieve significantly higher accuracy, at the cost of high inference speed. The performance of frames per second is less than that of one-stage detectors. Those models are Region CNNs, Faster R-CNN or Mask R-CNN. On the other hand, One-stage detectors function the same way without the RPN, this process is significantly faster and therefore can be used in real-time applications. The drawback is that the algorithm prioritizes inference speed but has a loss of accuracy when it comes to detecting small-shaped objects, groups of objects, and odd-shaped (possibly blurred) objects. Some of the models are YOLO, SSD, and RetinaNet. These types of models are generally the fastest, structurally simple, and efficient when compared with multi-stage detectors. Lastly, another important factor that we have to take into consideration is the 'backbone' of those models. The backbone of an NN is the core architecture that supports the learning process and enables the network to extract meaningful features from the input data. It dictates the performance of the network in DL tasks, and choosing the right backbone is essential to its success. The focus of the chapter is to investigate some of the popular models used in object detection.

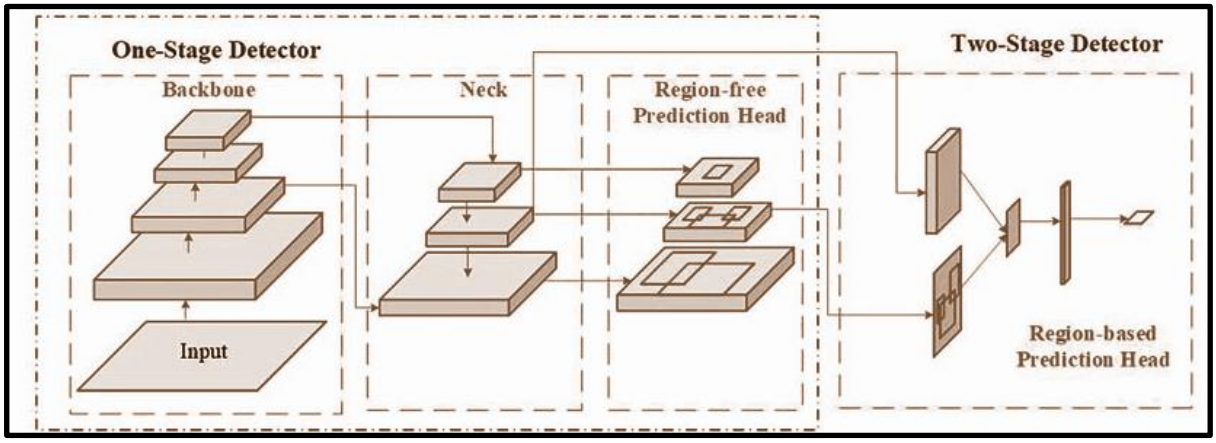

Fig. 2.2.1 Structural difference of One and Two Stage Detectors [8].

Starting with Faster R-CNN (Two-Stage) [5], produced by extending R-CNN and Fast R-CNN models. Adding RPN which is a fully convolutional network for generating proposals of bounding boxes and aspect ratios. Faster R-CNN introduced the concept of anchor boxes, which are fixed bounding boxes depending on the class and are widely used in almost every modern model. As depicted in the picture, the two stages are RPN and Fast R-CNN. The backbone network is usually a dense CNN, originally VGG-16 however it was observed that replacing it with ResNet offers significant improvement in performance.

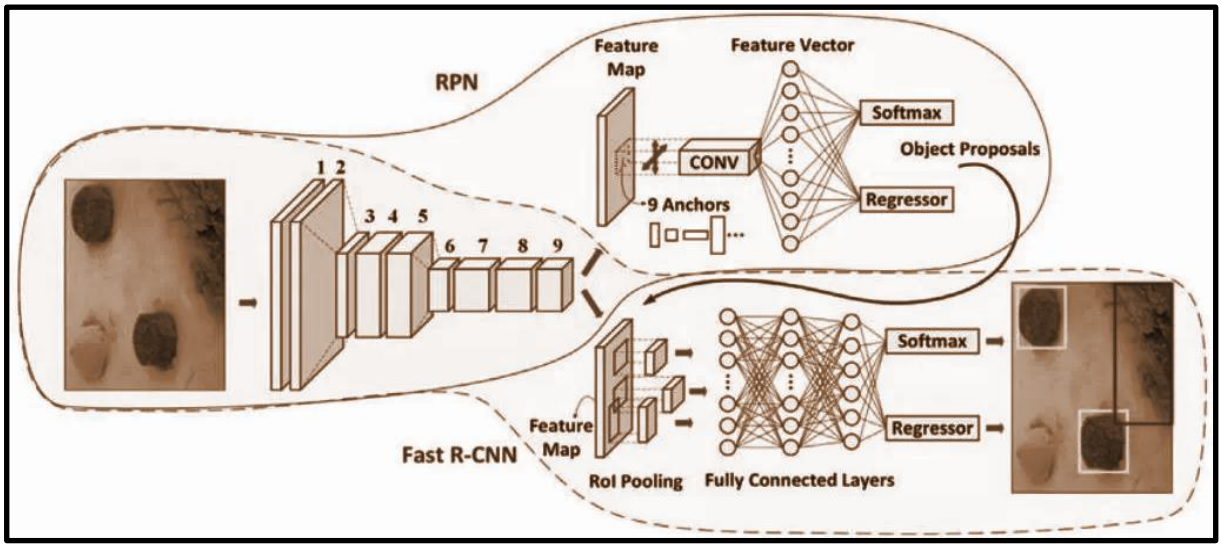

Fig. 2.2.2 A Faster R-CNN structure.

The strength of R-CNN when combined with multi-layered backbone NN, i.e. ResNet-101, offers the best possible Average Precision over Small, Medium, and Large objects, of course at the cost of time. Suited better for medical diagnosis related feature extraction problems.

Single-shot detector (one-stage) (SSD) was the first to achieve an accuracy that rivals two-stage detectors [6]. Brief SSD has two components: a backbone model and an SSD head. The backbone model is typically a network like ResNet, from which the final fully connected classification layer has been removed. The SSD head is just one or more convolutional layers added to this backbone and the outputs are interpreted as the bounding boxes and classes of objects in the spatial location of the activations of the final layers.

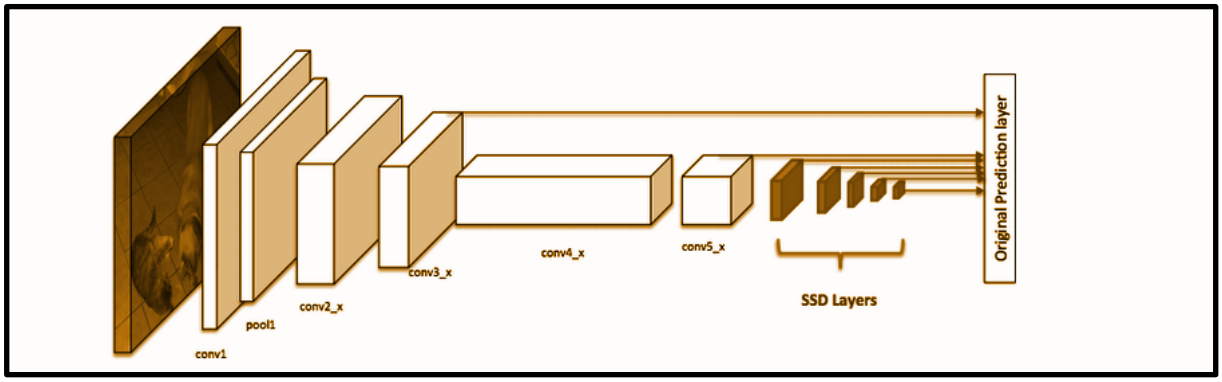

Fig. 2.2.3 Single Shot detector structure.

YOLOv3(one stage) [9] has the advantage of being much faster than other networks and still maintains accuracy. It allows the model to look at the whole image at test time, so its predictions are informed by the global context in the image. YOLO and other CNN algorithms "score" regions based on their similarities to predefined classes, in our case that would be cars and license plates. High scoring regions are noted as positive detections of whatever class they are most likely to identify as. In practice, in a live feed of a railroad, YOLOv3 can

detect different kinds of vehicles depending on which region of the video scores highly in comparison to the predefined classes of vehicles. In conclusion, the accuracy of YOLOv3 increases the bigger the amount of data available for training, perfect for traffic related applications since the number of different vehicles is plentiful.

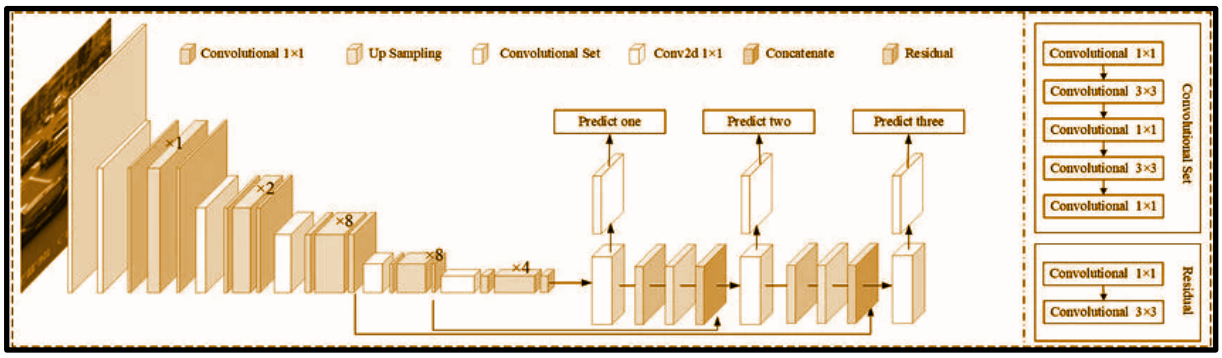

Fig. 2.2.5 YoloV3 structure by [10].

To summarize, two stage detectors offer more precise outputs slowly compared to other models and are more suited for Computer Vision applications in Medicine. On the other hand, one-stage detectors offer decent results extremely fast, usually depending on their vast amount of training time to cover for their hastiness. Since our project aspires to be concurrent one-stage detectors are more suited for the job, reducing the computational burden and having the optimal inferance speed. Next on the list is to take a peek into the evaluation of the models and their outputs.

2.3 Evaluation Metrics

In Object Detection, it is insufficient to know just about the class of the identified object. There are other variables that quantify the performance of the model such as the speed of the Training, the speed of the Inference, Average Precision and Recall depending on the threshold of our problem, typically 0.5. Still, those are just the tip of the iceberg, it is also important to know to what extent the model can identify different-sized objects.

Architecture	Feature Extractor	Computational Time (ms)		Mean Average Precision					
		Training	Inference	AP	$AP_{0.5}$	$AP_{0.75}$	AP_S	AP_M	AP_L
RetinaNet	FPN ResNet50	75.62	47.47	27.3	44.4	28.0	1.0	24.8	64.2
RetinaNet	FPN ResNet101	111.36	57.74	28.1	45.6	29.0	1.1	25.6	65.7
RetinaNet	FPN ResNeXt101	156.57	67.91	28.6	46.4	29.1	1.2	26.0	66.7
RetinaNet	FPN ResNet152	220.20	74.20	28.4	46.2	28.9	1.2	26.0	66.2
RetinaNet	FPN MobileNet	56.20	33.02	25.1	39.4	24.3	1.3	20.0	57.3
RetinaNet	FPNLite MobileNetV2	**50.21**	**26.12**	24.6	38.2	23.6	1.1	18.0	55.8
Faster RCNN	FPN ResNet50	86.23	48.00	29.5	47.9	30.6	2.8	29.6	65.3
Faster RCNN	FPN ResNet101	119.52	58.10	30.9	49.9	32.3	3.2	31.2	66.0
Faster RCNN	FPN ResNeXt101	161.86	66.81	31.6	50.9	33.0	3.4	32.0	67.0
Faster RCNN	FPN Res2Net101	151.30	63.78	**32.4**	51.7	**34.2**	3.6	**33.2**	**68.3**
Faster RCNN	FPN ResNet152	210.20	75.30	31.7	51.2	33.3	3.4	31.5	67.1
FCOS	FPN ResNet50	75.14	42.25	27.2	45.3	27.2	2.2	24.7	62.1
FCOS	FPN ResNet101	103.50	52.59	28.9	47.1	29.7	2.7	26.5	64.6
FCOS	FPN ResNeXt101	145.23	60.87	29.0	47.8	29.4	3.0	26.6	64.3
YOLOv3	DarkNet-53	82.56	40.19	29.6	**53.1**	29.2	**5.5**	30.2	58.4

Fig 2.3.1 Example of evaluation metrics on Yolov3 and Faster RCNN in [11].

The evaluation metric used to extract the Mean Average Precision is Intersection over Union (IoU). IoU is calculated by dividing the overlap between the predicted and ground truth annotation by the union of these. As we can see the Average Recall is not calculated, that is due to the nature of our predictions, Recall doesn't play an essential role as Precision in our case. In the case of Cancer cell detection, it would be the opposite. Lastly, the meaning of the thresholds dictates whether the predictions will be a True Positive or False Positive. As previously mentioned, the typical IoU threshold is 0.5, when needed to be stricter with the outcomes of the model 0.7~0.75 is considered a strict threshold.

Other important metrics we are going to keep track of can be categorized as Box Loss, Objectness, and Classification oriented all using Mean Squared Error. Box Loss represents the ability of the algorithm to detect the center of an object

and how well the predicted bounding box covers an object. Objectness is a measure of the probability that an object exists in a proposed region of interest.

If the objectivity is high, this means that the image window is likely to contain an object. Classification loss gives an idea of how well the algorithm can predict the correct class of a given object. Keep in mind that these metrics are more 'personalized' to our task and cannot be adapted by all the other models.

To sum up, when it comes to traffic-related object detection, the evaluation metrics used to measure a model's quality are its computational cost and Average precision or Average Recall. There are also extra metrics to keep track of while a model is in its training phase, that likewise help measure the quality of the predictions and not so much as the model itself. All in all choosing metrics is ultimately related to the application and the qualities of a 'good' model are interchangeable.

Chapter 3 Autonomous Recognition of Vehicles and Plate Detection

By now it was established that NNs projects are complex in nature and expensive to train and run, but can offer a wide range of applications. Object Detection specifically employ CNNs algorithms that are one of the most composite of convolutional layers, normalization layers, Max Pooling layers, and Upsample before producing predictions and can sum up to hundreds of layers. The fulfilment process can be summarized in the simplified Flow Chart below. Our proposed system is a single device end-to-end application, which can easily be scaled to edge-computing with small adjustments.

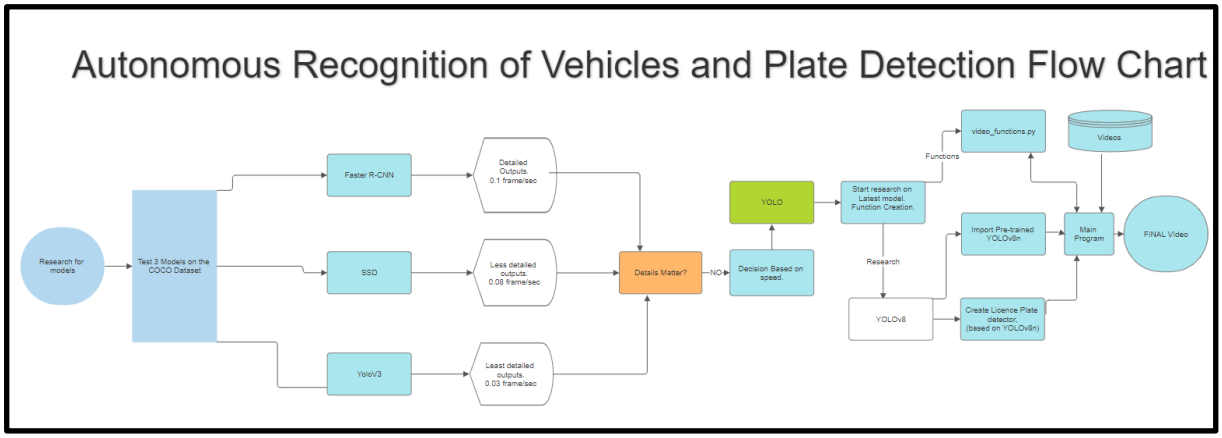

Fig 3.1. Flow chart of this Thesis research steps.

Throughout the intensive research, one can find not one way to implement a similar project but multiple. It is in the eye of the beholder to choose his tools. In our project first, we tested three famous models Faster R-CNN, SSD, and YoloV3 on the COCO 2017 dataset. The model benchmark process revealed that the YOLO algorithm was the fastest to produce results, with 0.03 sec/per frame as opposed to 0.08 SSD and 0.1 on Faster RCNN, and was chosen as the model to be the core of the application.

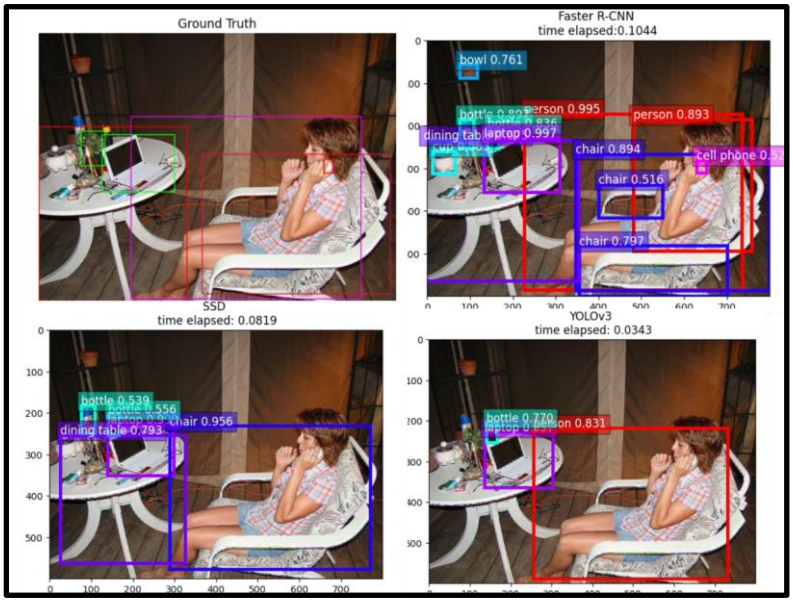

Fig 3.2. The results of SSD, YOLOv3, and Faster R-CNN tests.

After the model was selected, research was conducted into YOLO. The latest version YOLOv8, even though it has been abandoned by its original researcher Joseph Redmon due to his disapproval of the militarized appliances YOLO might produce, has an active community and is slightly better than its predecessor YOLOv7, more details on the choice will be on the corresponding chapter for the model.

For the Vehicle Recognition part the pre-trained weights were chosen instead of training a new model. The license plate detector was trained anew on a dataset that can be found publicly available online, using the YOLOv8 architecture and yielded satisfying results. Functions were implemented for a variety of purposes, for example runtime estimation, designated file creation, video splitting into frames etc. that can help expand the project in the future.

3.1 Yolov8

YOLOv8 is the latest of the YOLO series models, it is exceptionally fast real-time inference on single-device applications. It is important to understand that the inference performance of a model depends on the hardware available and is not something constant to be measured. YOLOv8 has five different-sized models, the general accuracy of those is presented below.

MODEL	Size (pixels)	mAPval 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	Parameters (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

The scores are from test runs on COCO 2017 dataset and are from single model single scale type [12,47] and the model show high precision 0.84, recall 0.969 and mAP 95.10% in Table III [44] compared to the older versions of YOLO algorithm. Another very important benefit of YOLOv8 is that it can support various backbones, such as EfficientNet, ResNet, and CSPDarknet, providing us with the flexibility to choose the best model for our specific use case. In our case, the backbone is default CSP-Darknet53. YOLOv8 is constructed in a way that offers adaptive training, optimizing the learning rate and balancing the loss function during training, which leads to better overall model performance. Additionally, it employs advanced data augmentation techniques, MixUp [45] and CutMix [46] to improve the robustness and generalization of the model. Lastly, most of the aforementioned are pre-build ready to deploy for the purpose of the task, without limiting the ability to customize the inner structure of the model when deemed needed. For example, in the hypothetical scenario where our task was to train an Automatic Broken Bone AI system, to lower human error and help junior doctors with limited experience, we would need to change the backbone to the ResNet family which is focused on Object Identification (in case of small fractures in the hands) and also change to a “larger” model with more parameters and higher inference such as YOLOv8l to maximize average Recall. All of which can be done with minimum code writing. The default structure of the model is depicted in (Fig. 3.1.1).

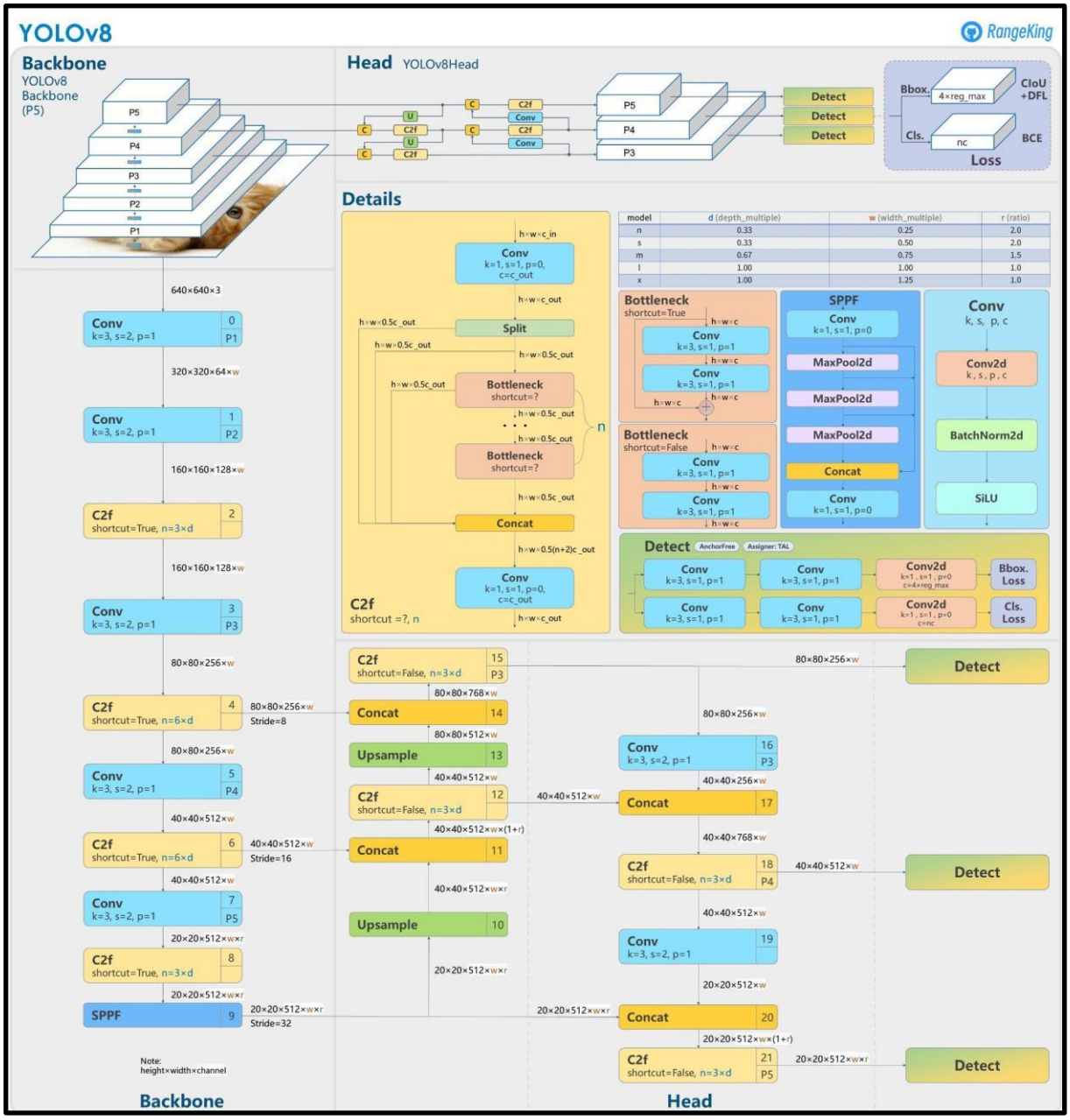

Fig 3.1.1 YOLOv8 diagram by user RangeKing (https://github.com/RangeKing).

YOLOv8 is distributed using the ultralytics library and in cooperation with the roboflow library, without the need to use the API, we built and employed our models. Ultralytics is a library similar to TensorFlow and PyTorch, but rather centered on model deployment, and roboflow for acquiring the dataset to train the LSP, apply preprocess and data augmentation operations, such as resizing, random horizontal flips, rotations, zooms, shifts, and many more. It is impossible for every single image scenario of a license plate to be captured, taking into account all the different weather, and lighting conditions. Preprocessing is essential for both technical and performance reasons and so is Augmentation. Both increase the robustness of the model and improve accuracy. After creating the models, they can be deployed directly by ultralytics or loaded into other

environments such as TensorFlow or PyTorch. Example of code loading YOLOv8n and our LSP after it is trained.

```python
#load models
coco_model =  YOLO('yolov8n.pt')
license_plate_model = YOLO('lsp_detector.pt')
```

Example of code for feeding frames into the model and using OpenCV for frame capturing.

```python
cap = cv.VideoCapture('sample_004.mp4')
fnum= 0
ret= True
#read frames, have final_results hold all the bbox,score,car_id,plate
while ret and fnum < 15:
#while ret :
  final_results[fnum] = {}

  fnum+=1
  ret, frame = cap.read()
  if ret:
    print('NEW FRAME \n\n')
    #Single frame outputs
    outputs = coco_model(frame)[0]
```

In conclusion, YOLOv8n is the model used for both Vehicle Detection and license plate detection. It is clear that other than the superior inference speed, it also comes with a lot of customization capabilities. The main libraries ultralytics and roboflow are more centered on the distribution, making it possible to deploy and train with a few lines of Python code, but that neither limits the capabilities of the models themselves nor the programmer as it can be loaded and modified freely.

3.2 License Plate Detector

For the training of the License Plate (LSP) Detector a Tesla T4 was used in order to minimize the training time. It is not important or impactful to the performance of the model, whether GPU or CPU is used for the training process, the only thing that changes is the amount of time to train the weights.

Creating the LSP Detector in our project can be separated into three steps. The first step is to obtain a dataset with license plates and inspect it. The dataset used consists of 10126 raw images. Generally, it is a good practice to have at least a thousand images per class. In our case, we will use the Augmented version of the dataset, consisting of 21174 images training 87%, 2049 validation set 8%, and 1020 test set 4%. The Augmentation techniques applied on the original testing images only are the following:

- **Flip**: Horizontal
- **Crop**: 0% Minimum Zoom, 15% Maximum Zoom
- **Rotation**: Between -10° and +10°
- **Shear**: ±2° Horizontal, ±2° Vertical
- **Grayscale**: Apply to 10% of images
- **Hue**: Between -15° and +15°
- **Saturation**: Between -15% and +15%
- **Brightness**: Between -15% and +15%
- **Exposure**: Between -15% and +15%
- **Blur**: Up to 0.5px
- **Cutout**: 5 boxes with 2% size each

```
from roboflow import Roboflow
rf = Roboflow(api_key="[redacted]")
project = rf.workspace("roboflow-universe-projects").project("license-plate-recognition-rxg4e")
dataset = project.version(4).download("yolov8")

loading Roboflow workspace...
loading Roboflow project...
Dependency ultralytics<=8.0.20 is required but found version=8.0.172, to fix: `pip install ultralytics<=8.0.20`
Downloading Dataset Version Zip in License-Plate-Recognition-4 to yolov8: 100% [998178198 / 998178198] bytes
Extracting Dataset Version Zip to License-Plate-Recognition-4 in yolov8:: 100%| | 48488/48488 [00:11<00:00, 4078.30it/s]
```

Fig 3.2.1 Code that downloads the chosen dataset.

The YOLO algorithm performs better with more data so it is advised to be greedy. We will check for overfitting after the training is over. The next step is training the model. As mentioned a Tesla T4 was used, publicly available by Google collabs, in order to minimize the training time. The model was trained for 20 epochs. It is good practice to first train a model around 20~25 epochs, then see how it performs and tune its hyperparameters.

```
20 epochs completed in 2.422 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train/weights/best.pt, 6.2MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.172 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%| | 64/64 [00:19<00:00,  3.33it/s]
                   all       2046       2132      0.976      0.962      0.983      0.703
Speed: 0.2ms preprocess, 2.2ms inference, 0.0ms loss, 1.6ms postprocess per image
```

Fig 3.2.2 Final output of the training process.

The training took 2.4 hours and we can see that the estimated mAP50 is 0.983 and mAP50-95 is 0.703. The models seem to be performing very well on the testing data. Small notice that the last line "Speed: 0.2ms preprocess, 2.2ms inference, 0.0ms loss, 1.6ms postprocess per image" with the T4 inference is taking about 4ms per frame more or less, which means the LSP model could theoretically be used for real-time application with 250FPS give or take. The reason we are highlighting this information is that the final project is done without the T4 Tesla, on a CPU that is significantly slower.

The final step is to evaluate the model. Conveniently most of the evaluation metrics are automatically generated by the ultralytics and during training the most crucial metrics were monitored automatically. There is no reason to tamper with any of those automatic processes, in the first iterations but it is possible. The configured values are performing well in general. For example, the learning rate usually is the best at 0.01.

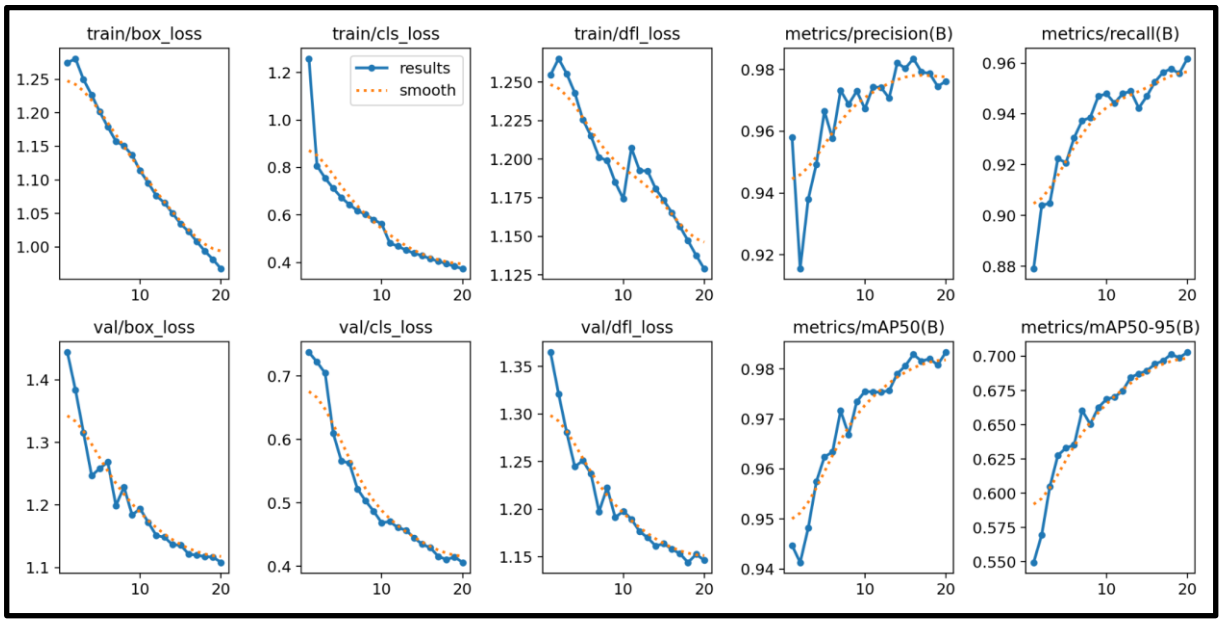

Fig 3.2.3 Plots of the most important Eval Metrics of the LSP.

From the Evaluation metric plots we can clearly see that the training process easily could be extended to 40-50 epochs since Box Loss was decreasing at the time of interruption and in general no curve converged, which is considerably logical result with such a small amount of epochs. We can safely assume that the model is not overfit by any means. The mAP seems to be promising. Interpreting evaluation metrics can only get us so far since there isn't any indication of the model acting up, it is time to test it on real inputs. From the batch testing Fig 3.2.4, the results are impressive but then again, we need to test on new data. Deliberately we chose a variety of video quality for the model to be measured on and as suspected on low-quality videos the LSP didn't perform well, but on medium to high-quality videos, the LSP achieved a better overall outcome. That happens because the YOLOv8 algorithm works with 640-pixel inputs, and the higher the quality the better it will perform. Additionally, we chose the 'nano' version, the smallest of the 5, which has the smallest number of parameters to

work with, that is the trade-off of choosing speed over accuracy. A factor to keep in mind when deploying the project.

Fig 3.2.4 Test batch predictions of the LSP.

The LSP detector is set, we wrote a small program that takes a video, applies the model frame by frame, and saves it as a new video. Through Images 3.2.5-8 the libraries are presented as well as the execution. The detector finds the bounding boxes above the given threshold and using the OpenCV module the vizualization is created. In the last image, we noticed that each frame takes roughly 170~240 ms and up to 400 ms rarely, which is very slow and cannot be used for real-time application, but that is because the LSP doesn't utilize the GPU and runs solely in the CPU of the system which is an Intel(R) Core(TM) i5-3350P CPU @ 3.10GHz.

```
import os
import numpy as np
from ultralytics import YOLO
from pathlib import Path
import cv2 as cv
%run timer_class.ipynb
```

Fig 3.2.5 Libraries used.

```
cap = cv.VideoCapture('./sample_003.mp4')
empty=0

ret, frame = cap.read()
print(frame.shape)
H,W,_ = frame.shape
out = cv.VideoWriter('sample_003_lp_out.mp4', cv.VideoWriter_fourcc(*'MP4V'),int(cap.get(cv.CAP_PROP_FPS)),(W,H))
```

```
(1080, 1920, 3)
```

Fig 3.2.6 Video selection, setting up the VideoWriter.

```
threshold = 0.5

while ret:

    results = model(frame)[0]

    for result in results.boxes.data.tolist():
        x1, y1, x2, y2, score, class_id = result

        if score > threshold:
            cv.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 4)
            cv.putText(frame, results.names[int(class_id)].upper(), (int(x1), int(y1 - 10)),
                        cv.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3, cv.LINE_AA)

    out.write(frame)
    ret, frame = cap.read()

cap.release()
out.release()
cv.destroyAllWindows()
```

Fig 3.2.7 Frame by frame LSP Detection and save.

```
0: 384x640 1 License_Plate, 157.0ms
Speed: 3.0ms preprocess, 157.0ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 License_Plate, 166.0ms
Speed: 4.0ms preprocess, 166.0ms inference, 2.0ms postprocess per image at shape (1, 3, 384, 640)
```

Fig 3.2.8 Example of time needed by the LSP Detector to process each image.

The LSP detector is done. We have a pre-trained model for Vehicle Recognition, we created an LSP detector, it is time to combine the DL models with code and

finish the project. There are still a lot of variables to take into consideration. For example, when applying the model at intersections there is the probability that in rare cases LSP detector can confuse license plates with other objects, for that precautions must be taken in the form of a stricter confidence when predicting.

3.3 Execution and Outputs

To establish that the LSP doesn't accidentally detect any sign caption, we decided to use redundancy in many forms, the pre-trained YOLOv8n will be used to make vehicle predictions, and then on the predicted bounding boxes the LSP Detector will be used. Next, we would need to extract the information from the LSP detector with an Optical Character Recognition (OCR) tool and save it to a .csv file along with some necessary information about the car, frame, and confidence of the prediction.

The purpose of the final program is to successfully extract the necessary information from a video, not visualize the results.

The execution process has three main steps. The first step is to use the pre-trained, on COCO 2017, YOLOv8n on each frame to detect Vehicle types and extract their locations. Searching through its class attribute we find that class IDs of [2,3,5,7] are Vehicle types. Furthermore, we employ a tracking algorithm for 2D multiple object tracking in video sequences which is called SORT in order to assign each Vehicle a unique ID.

```
while ret:
#while ret :
  fnum+=1
  final_results[fnum] = {}



  ret, frame = cap.read()
  if ret:
    print('NEW FRAME \n\n')
    #Single frame outputs
    outputs = coco_model(frame)[0]
    outputs_ = []

    for output in outputs.boxes.data.t
                                        (variable) output: Unknown
      x1,y1,x2,y2, score, class_id =  output

      if int(class_id) in recognized:
        outputs_.append([x1,y1,x2,y2,score])

        #Tracking object and keep each seperate objects characteristics
        track_ids = mot_tracker.update(np.asarray(outputs_))
```

Fig 3.3.1 Vehicle detection and assigning unique ID with SORT.

Step two, now that we have bounding boxes of images containing Vehicles we cross-reference the predictions of the LSP with the predictions of the first model. We prepared many utility functions that perform such actions, which are contained in the video_functions.py file. The get_car() takes the tabular data and tries to find a licence plate detected by the LSP model in the Vehichle predictions. If there is a licence plate match, we proceed to the third step, extraction and temporary save.

```
#Detect licence plates
license_plates = license_plate_model(frame)[0]
for license_plate in license_plates.boxes.data.tolist():
  x1,y1,x2,y2, score, class_id =  license_plate
  #print('License plate detector output for cur_frame contents: ',x1,y1,x2,y2, score, class_id)

  #Assign the license plate to its respective car
  x1_car,y1_car,x2_car,y2_car,id_car = get_car(license_plate,track_ids)
```

Fig 3.3.2 Cross-referencing Licence Plates and Vehicles.

After finding the connected Vehicles and Licence Plates, we need to preprocess them Fig. 3.3.3-5. This part is necessary in order to increase the performance of our OCR. The OCR module used is pytesseract, which recognizes character patterns. The model was trained using scripts so in order to maximize the outputs, it would be best for the images offered to be in a similar form and have white background and black letters as much as possible. Cropping the images with the licence plates, converting them into grayscale, applying a filter threshold for the pixes, and then bitwise_not, before feeding them to the OCR. Grayscaling the image covers the case of multicoloured licence plates and letters. Afterwards, the croped licence plates are fed (Fig. 3.3.3-3.3.5) to the OCR, keeping the results that have the highest score. The reasoning behind this is once more to widen the horizon on all the possible cases with different multicoloured licence plates.

Fig 3.3.3 Grayscaled Licence Plate.

Fig 3.3.4 After Pixel Threshold Licence Plate.

Fig 3.3.5 Bitwise not Thresholded Image.

After running some quick tests in Fig 3.3.8, we noticed that the OCR predicted

NAI**5**NRU while the ground truth was NAI3NRU. It is pretty common with OCRs to mistake some characters for others. One of the drawbacks of using pytesseract is that the images need to be further pre-processed. Making some further adjustments, in particular image resize, we managed to correct the errors in Fig 3.3.9, at the cost of execution time. There are many different factors that we need to take into consideration before processing any images. Some may need different preprocessing methods in order to produce the best possible result and it is not possible to account for every single of these factors. Some examples are the different lighting conditions, noise of the image, angles etc.

```
#crop licence plate into the car
licence_plate_croped = frame[int(y1):int(y2),int(x1):int(x2),:]
#Make the license plate cleaner, for input
license_plate_croped_gray = cv2.cvtColor( licence_plate_croped, cv2.COLOR_BGR2GRAY)
_, license_plate_croped_thresh = cv2.threshold(license_plate_croped_gray,110,255,cv2.THRESH_BINARY_INV)

#Save some frames so we know what we capture
# if not cv2.imwrite(f"bef_gray_thresh{str(fnum)}.png",license_plate_croped_gray,):
#   raise Exception("Could not write image-> bef_crop")

# if not cv2.imwrite(f'gray_thresh{str(fnum)}.png',license_plate_croped_thresh):
#   raise Exception("Could not write image-> croped img")

license_plate_croped_rev_thresh = cv2.bitwise_not(license_plate_croped_thresh)

# if not cv2.imwrite(f'gray_thresh_rev{str(fnum)}.png',license_plate_croped_rev_thresh):
#   raise Exception("Could not write image-> croped img")

#Read the license plate number
license_plate_number, license_plate_score = read_license_plate(license_plate_croped_gray)
license_plate_number2, license_plate_score2 = read_license_plate(license_plate_croped_rev_thresh)
#print(f'RETURNED.LSP NUMBER = {license_plate_number} {type(license_plate_number)}, \n CONFIDENCE SCORE = {license_plate_score} {type(license_plate_score)}')
```

Fig 3.3.6 Preprocess licence plates and character extraction.

If the character read by the OCR fits the licence plate criteria then it is saved in the format frame_nmr, car_id, car_bbox, license_plate_bbox, license_plate_bbox_score, license_number, license_number_score. The plate with the highest score should hold the correct licence plate number and between the grayscale and the threshold-bitwise-not, we will choose the one with the highest confidence score. After the execution for all the frames is completed, the results are saved on a .csv output file, ready to be utilized for every kind of purpose. With the .csv file we can either vizualize every detection, or use the confidence scores as thresholds and vizualize only the most confident results, ultimately use it combined with a database to store the activity in areas of interest.

```python
if license_plate_number is not None and license_plate_number2 is not None:
  if float(license_plate_score) < float(license_plate_score2):
    license_plate_number = license_plate_number2
    license_plate_score = license_plate_score2
elif license_plate_number is not None or license_plate_number2 is not None:
  if  license_plate_number2 is not None:
      license_plate_number = license_plate_number2
      license_plate_score = license_plate_score2

if license_plate_number is not None:
  elements+=1
  new_row = {"frame_nmr":fnum ,"car_id":id_car ,
             "car_bbox":[x1_car,y1_car,x2_car,y2_car] ,
             "license_plate_bbox":[x1,y1,x2,y2] ,
             "license_plate_bbox_score":score ,
             "license_number": license_plate_number,
             "license_number_score":(license_plate_score+0.001)
             }
  df.loc[len(df)] = new_row
```

Fig 3.3.7 Temporary save the highest Licence Plate.

```
    frame_nmr  car_id                                   car_bbox  ... license_plate_bbox_score license_number license_number_score
0          -1    -1.0                                         -1  ...                -1.000000            -1               -1.000
1           0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
2           0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
3           0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
4           0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
5           0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
6           0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
7           0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
8           0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
9           0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
10          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
11          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
12          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
13          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
14          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
15          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
16          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
17          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
18          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
19          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
20          0     3.0  [752.8723754882812, 1369.210693359375, 1430.78...  ...                 0.565844         NA15NRU                0.001
21          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
22          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
23          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
24          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
25          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
26          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
27          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
28          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
29          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
30          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
31          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
32          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
33          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
34          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
35          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
36          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
37          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
38          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001
39          1    25.0  [752.8013305664062, 1369.2247314453125, 1430.3...  ...                 0.565977         NA15NRU                0.001

[40 rows x 7 columns]
saving with pandas Library
Elapsed time is: 146.13 seconds
```

Fig 3.3.8 Test results (a) of the execution.

	frame_nmr	car_id	...	license_number	license_number_score
0	-1	-1.0	...	-1	-1.000000
1	0	3.0	...	NA13NRU	39.929711
2	0	3.0	...	NA13NRU	39.929711
3	0	3.0	...	NA13NRU	39.929711
4	0	3.0	...	NA13NRU	39.929711
5	0	3.0	...	NA13NRU	39.929711
6	0	3.0	...	NA13NRU	39.929711
7	0	3.0	...	NA13NRU	39.929711
8	0	3.0	...	NA13NRU	39.929711
9	0	3.0	...	NA13NRU	39.929711
10	0	3.0	...	NA13NRU	39.929711
11	0	3.0	...	NA13NRU	39.929711
12	0	3.0	...	NA13NRU	39.929711
13	0	3.0	...	NA13NRU	39.929711
14	0	3.0	...	NA13NRU	39.929711
15	0	3.0	...	NA13NRU	39.929711
16	0	3.0	...	NA13NRU	39.929711
17	0	3.0	...	NA13NRU	39.929711
18	0	3.0	...	NA13NRU	39.929711
19	0	3.0	...	NA13NRU	39.929711
20	0	3.0	...	NA13NRU	39.929711
21	1	25.0	...	NA13NRU	40.704205
22	1	25.0	...	NA13NRU	40.704205
23	1	25.0	...	NA13NRU	40.704205
24	1	25.0	...	NA13NRU	40.704205
25	1	25.0	...	NA13NRU	40.704205
26	1	25.0	...	NA13NRU	40.704205
27	1	25.0	...	NA13NRU	40.704205
28	1	25.0	...	NA13NRU	40.704205
29	1	25.0	...	NA13NRU	40.704205
30	1	25.0	...	NA13NRU	40.704205
31	1	25.0	...	NA13NRU	40.704205
32	1	25.0	...	NA13NRU	40.704205
33	1	25.0	...	NA13NRU	40.704205
34	1	25.0	...	NA13NRU	40.704205
35	1	25.0	...	NA13NRU	40.704205
36	1	25.0	...	NA13NRU	40.704205
37	1	25.0	...	NA13NRU	40.704205
38	1	25.0	...	NA13NRU	40.704205
39	1	25.0	...	NA13NRU	40.704205

[40 rows x 7 columns]
saving with pandas Library
Elapsed time is: 191.85 seconds

Fig 3.3.9 Test results (b) of the execution.

In Fig 3.2.3 the graph of different evaluation metrics were ploted, over the validation set which was 1020 images and in Fig. 3.2.4 a batch of prediction was presented as well. Over the 1020 images the LSP detector scored remarkable.

metrics/precision(B),	metrics/recall(B),	metrics/mAP50(B),	metrics/mAP50-95(B),
0.95801,	0.87899,	0.94474,	0.54955,
0.91546,	0.90404,	0.94132,	0.56972,
0.93806,	0.90478,	0.94821,	0.60479,
0.94931,	0.92242,	0.95745,	0.62772,
0.96651,	0.92073,	0.96243,	0.63314,
0.95763,	0.93058,	0.96341,	0.6352,
0.97322,	0.93736,	0.97165,	0.66047,
0.96873,	0.93856,	0.96687,	0.6505,
0.97303,	0.947,	0.97347,	0.66265,
0.9673,	0.94794,	0.9755,	0.66866,
0.97435,	0.9442,	0.97542,	0.67026,
0.9742,	0.94794,	0.97534,	0.67477,
0.97073,	0.94896,	0.9757,	0.68464,
0.98215,	0.94231,	0.97904,	0.68732,
0.98038,	0.947,	0.98061,	0.68944,
0.98354,	0.95263,	0.98291,	0.69439,
0.97916,	0.95638,	0.98157,	0.69676,
0.97876,	0.95779,	0.98204,	0.70146,
0.97454,	0.95591,	0.9808,	0.69895,
0.9762,	0.96178,	0.98333,	0.70292,

Fig 3.3.10 Evaluation scores of the LSP over training. (on validation set)

In Fig 3.3.11 is presented a prediction on a random image with a minimum of 0.45 confidence, the results of which are very good. It is important to remind that the most important metric in our case is precision rather than recall and the accuracy which is not presented is the confidence of the model in the prediction since it is not a classification but a detection problem. In Fig 3.3.11 confidence is 0.78 or in other words 78% accuracy that this was a licence plate.

Fig 3.3.11 Detection on random image.

Lastly we will run some more tests on the final 'whole' version of our program, changing our OCR from pytesseract to EasyOCR for better results over different images. The change was made because pytesseract was providing insufficient results. The images used were low quality 416x416 pixels and so the results are expected to have low confidence, reminding that YOLOv8n needs high quality images.

Fig 3.3.12 Input image 416x416 pixels

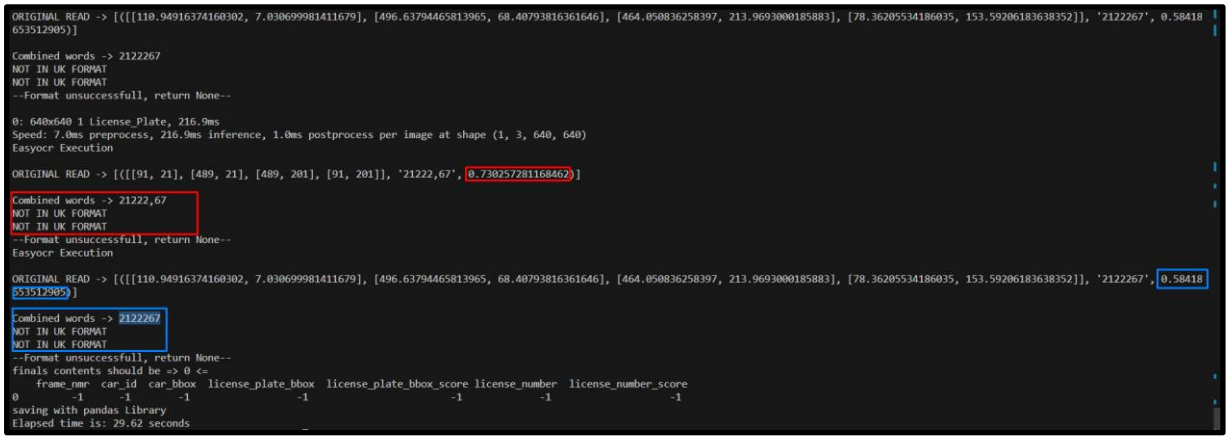

Fig 3.3.13 Execution outputs.

The important parts are marked for clarity reasons. The results are the default '-1' that marks an empty output .csv. The OCR detects the number correctly but also recognizes a comma. Licence plates should not have special characters so we remove them. We try to detect the UK licence plate format [43] which consists of seven digits. Digits 1 and 2 are letters, digits 3 and 4 are numbers and digits 5,6 and 7 can be of any random letters. The confidence is very high taking into consideration the quality of the image.

Fig 3.3.14 Input image 1181 x 664 pixels

```
ORIGINAL READ -> [([[115, 85], [1000, 85], [1000, 363], [115, 363]], 'RVO5 KKE', 0.7611853800079817)]

Combined original -> RVO5 KKE
FOUND UK FORMAT
--Format SUCCESSFUL,  RV05KKE 0.7611853800079817
Ocr read = RVO5KKE
Easyocr Execution

ORIGINAL READ -> [([[120, 88], [992, 88], [992, 360], [120, 360]], 'RVOS KKES', 0.40307836830264426)]

Combined original -> RVOS KKES
length isn't 7.
length isn't 7.
--Format unsuccessfull, return None--
finals contents should be => 1 <=
   frame_nmr  car_id                                   car_bbox                                license_plate_bbox  license_plate_bbox_score license_number  license_number_score
0         -1    -1.0                                         -1                                            -1                -1.000000            -1             -1.000000
1          0     1.0  [41.36647033691406, 74.11553192138672, 1094.86...  [172.6215362548828, 441.15924072265625, 341.33...                0.798593        RV05KKE              0.761185
saving with pandas Library
Elapsed time is: 15.75 seconds
```

Fig 3.3.15 Execution outputs.

As we can see, it has 0.78 licence plate confidence and a 0.76 number score with a higher quality clean image. Let's also take a look at a lower-quality image in which the model got confused.

Fig 3.3.16 Input image 416x416 pixels

```
ORIGINAL READ -> [([[198, 20], [346, 20], [346, 170], [198, 170]], 'Uip', 0.05791593836072108), ([[39.370356571234666, 47.07392228358133], [217.98497177855714, 35.53442709258516], [219.62964342876535,
192.92607771641866], [42.01502822142876, 204.46557290741484]], 'HJ', 0.2708181926927958)]

Combined original -> Uip HJ
length isn't 7.
length isn't 7.
length isn't 7.
length isn't 7.
--Format unsuccessfull, return None--
Easyocr Execution

ORIGINAL READ -> [([[17, 42], [233, 42], [233, 200], [17, 200]], 'HVIJ', 0.07625382393598557), ([[195.13947898116186, 42.115831849294985], [334.44558980135855, 19.34749669365113], [348.86052101883814,
151.88416815070502], [208.55441019864145, 174.65250330634888]], 'OTp', 0.7095655384077366)]

Combined original -> HVIJ OTp
length isn't 7.
FOUND UK FORMAT
--Format SUCCESSFUL,  HV13OTP 0.07625382393598557
Ocr read = HVIJOTP

0: 640x640 1 License_Plate, 215.9ms
Speed: 7.0ms preprocess, 215.9ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 640)

0: 640x640 1 License_Plate, 196.9ms
Speed: 6.0ms preprocess, 196.9ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 640)
finals contents should be => 1 <=
   frame_nmr  car_id                                   car_bbox                                license_plate_bbox  license_plate_bbox_score license_number  license_number_score
0         -1    -1.0                                         -1                                            -1                -1.000000            -1             -1.000000
1          0     1.0  [4.872341632843018, 55.94768524169922, 405.784...  [295.0162353515625, 167.7871551513672, 356.862...              0.941243       HV13OTP              0.076254
saving with pandas Library
Elapsed time is: 7.16 seconds
```

Fig 3.3.17 Execution outputs.

In the licence plate, the OCR confuses the letter I for 1. and so we get a false positive. Although the licence plate bbox has a confidence of 0.94 the number score is lower than 0.1, making it easy to disregard by applying a minimum threshold on the value number score. Keeping in mind that the low confidence is due to the low quality image, and the mistake is of logical nature since we take into consideration the possibility of letter I in digit 3 might be a 1, confused by the OCR. In order to further improve the program in the future a system where tampering with results takes into consideration that the possibility of confusion, is directly influenced by the confidence of the prediction.

This redundancy costs a lot of computational time but it is important in order to ensure correct information extraction. Redundancy tends to avoid unwanted scenarios and ensure a safer execution in exchange for computational time. Either in the form of extra predictions or in the form of extra checks that additional computational time will be sacrificed towards better results. We are sure of the LSP detector execution and there is no doubt about the pretrained YOLOv8n model.

3.4 Improvements and limitations.

During the planning phase of our proposed system, the main focus was feature extraction functionality and execution/compilation speed, while also trying to create flexible code that can be used in multiple scenarios. Being able to have a lightweight model such as YOLOv8n, can make real-time inference possible. There is plenty of room for improvement when it comes to code optimization. The proposed system was also limited by the hardware available.

To begin with the improvements, there is a lot of room to refine the LSP Detector. Increase the epochs to 50~100, searching for the best number while avoiding overfitting or even adding more images to the dataset all of which will result in an accuracy increase. It is also possible to remove the LSP Detector as an individual model and add its functionality to the main YOLOv8n due to the fact that the first layers in both models have common features. The idea behind the LSP Detector is for it to be the specialized network that detects licence plates in Vehicle objects.

Next on the optimization list, there are different kinds of OCR performing better in terms of accuracy and speed compared to pytesseract such as EasyOCR. Pytessaract was chosen due to its end-to-end support for multiple languages, offering flexibility when adjusting the feature extraction on different language licence plates. It makes it possible to create a multilingual licence plate detection application. EasyOCR might be offering better accuracy, but is computationally expensive and cannot cover every factor that needs to be taken into consideration when preprocessing the image. (Light conditions, blur, noise, angle etc.)

The limitations mainly consist of hardware requirements. As already stated the LSP Detector was trained on a T4 Tesla in Google Colab and the predictions were made locally on a singular computer. There were no cloud services available to test for parallel execution of the program and check its functionality on real-time inference. Running without GPU acceleration is very time-inefficient. Another limitation is that the results produced by the DL models are not very clean. In reality, there are a lot of processes working in the background, that build an orderly presentable final output.

In conclusion, tools and models have their pros and cons depending on the nature of the problem. The biggest limitation of the project was the lack of hardware to further implement the capabilities of the algorithm as well as reduce the time inefficiency of the test executions. Object detection is a very complex field to be approached by many different scopes, and the project has a lot of room to grow.

Conclusions

Automatic Recognition of Vehicles and Updates using Licence Plates can be of great merit to the public, private businesses, the government and its respective authorities. The project in general has room for improvement and can be further developed to increase its functionality. Monitoring an area for trespassing or traffic violations is a difficult and demanding task to cover using human resources, especially in remote places. This problem can be solved by deploying Automatic Recognition of Vehicles using Licence Plates and utilizing modern technology. Keeping always in mind the Ethical part of data collection.

CNN Object Detection algorithms have a lot to offer towards recognition and the YOLO model isn't the only one fit for the job. As previously mentioned, there are countless ways to approach the problem at hand. To maximize the usage of the project, Cloud services are also needed and deployment on the Edge would be preferable. Partly why, the inference speed was the most vital value parameter with a lightweight algorithm during model selection. Though single-device end-to-end is the current extent of the Algorithm, it is sustainable with a proper GPU.

Building the code using Python might not be the fastest option when compiling and executing but it comes with the benefits of flexible code, an enormous active community, constant breakthroughs and new tools. Vital elements towards building a successful and long-lasting application.

Lastly, Automatic Recognition of Vehicles and Updates using Licence Plate's strongest suit is as a Monitoring tool for Traffic Law Enforcement and data collection. As Benjamin Franklin once said "An ounce of prevention is worth a pound of cure" Monitoring forest entrances, intersections, school entrances, and remote roads are all places where activities frequently contravene the law. The deployment of the model can help enforce traffic rules, creating safer roads and drivers' solidarity. As a data collecting tool, locating a stolen vehicle or tracking a suspicious vehicle activity, provided enough coverage would simplify the process drastically.

BIBLIOGRAPHY

[1] Zhang, H., Sindagi, V., & Patel, V. M. (2019). Image de-raining using a conditional generative adversarial network. IEEE transactions on circuits and systems for video technology, 30(11), 3943-3956.

[2] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

[3] Thompson, N. C., Greenewald, K., Lee, K., & Manso, G. F. (2020). The computational limits of deep learning. arXiv preprint arXiv:2007.05558.

[4] Sharut Gupta, Stefanie Jegelka, David Lopez-Paz, Kartik Ahuja (2023). Context is Environment

[5] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. (2016). SSD: Single Shot MultiBox Detector.

[7] Carranza-García, M., Torres-Mateo, J., Lara-Benítez, P., & García-Gutiérrez, J. (2020). On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data. Remote Sensing, 13(1), 89.

[8] Zhang Yue, Xie Fei, Huang Lei, Shi Jianjun, Yang Jiale, Li Zongan.(2021). A Lightweight One-Stage Defect Detection Network for Small Object Based on Dual Attention Mechanism and PAFPN. https://www.frontiersin.org/articles/10.3389/fphy.2021.708097

[9] Joseph Redmon, Ali Farhadi (*2018). YOLOv3: An Incremental Improvement*

[10] Q. -C. Mao, H. -M. Sun, Y. -B. Liu and R. -S. Jia, "Mini-YOLOv3: Real-Time Object Detector for Embedded Applications," in IEEE Access, vol. 7, pp. 133529-133538, 2019, doi: 10.1109/ACCESS.2019.2941547.

[11] Carranza-García, M., Torres-Mateo, J., Lara-Benítez, P., & García-Gutiérrez, J. (2020). On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data. Remote Sensing, 13(1), 89. https://doi.org/10.3390/rs13010089

[12] Jacob Solawetz, Francesco." Roboflow Blog, Jan 11, 2023. https://blog.roboflow.com/whats-new-in-yolov8/

[13] Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc."

[14] Vanessa Sochat, Aldo Culquicondor, Antonio Ojea, Daniel Milroy (2023). The Flux Operator

[15] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning

[16] Mohammad Javad Shafiee, Brendan Chywl, Francis Li, Alexander Wong (2017). Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video

[17] Pinheiro, P., & Collobert, R. (2014, January). Recurrent convolutional neural networks for scene labeling. In International conference on machine learning (pp. 82-90). PMLR.

[18] https://www.statista.com/

[19] Gupta, S., Shrivastava, N. A., Khosravi, A., & Panigrahi, B. K. (2016, July). Wind ramp event prediction with parallelized gradient boosted regression trees. In 2016 International Joint Conference on Neural Networks (IJCNN) (pp. 5296-5301). IEEE.

[20] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[21] Sheridan, R. P., Liaw, A., & Tudor, M. (2021). Light gradient boosting machine as a regression method for quantitative structure-activity relationships. arXiv preprint arXiv:2105.08626.

[22] Slater, P. B. (2012). Comparative Bi-stochastizations and Associated Clusterings/Regionalizations of the 1995-2000 US Intercounty Migration Network. arXiv preprint arXiv:1208.3428.

[23] Mao, Y., Balasubramanian, K., & Lebanon, G. (2010). Linguistic Geometries for Unsupervised Dimensionality Reduction. arXiv preprint arXiv:1003.0628.

[24] Manthey, B., & Röglin, H. (2009, January). Improved smoothed analysis of the k-means method. In Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (pp. 461-470). Society for Industrial and Applied Mathematics.

[25] Fisher, D. (1996). Iterative optimization and simplification of hierarchical clusterings. Journal of artificial intelligence research, 4, 147-

178.

[26] Khan, M. M. R., Siddique, M. A. B., Arif, R. B., & Oishe, M. R. (2018, September). ADBSCAN: Adaptive density-based spatial clustering of applications with noise for identifying clusters with varying densities. In 2018 4th international conference on electrical engineering and information & communication technology (iCEEiCT) (pp. 107-111). IEEE.

[27] Yu, G., Sapiro, G., & Mallat, S. (2011). Solving inverse problems with piecewise linear estimators: From Gaussian mixture models to structured sparsity. IEEE Transactions on Image Processing, 21(5), 2481-2499.

[28] Fan, M., Gu, N., Qiao, H., & Zhang, B. (2010). Intrinsic dimension estimation of data by principal component analysis. arXiv preprint arXiv:1002.2050.

[29] Zhang, Z. (2015). The singular value decomposition, applications and beyond. arXiv preprint arXiv:1510.08532.

[30] Marivate, V. N., Nelwamodo, F. V., & Marwala, T. (2007). Autoencoder, principal component analysis and support vector regression for data imputation. arXiv preprint arXiv:0709.2506.

[31] Baldi, P., & Lu, Z. (2012). Complex-valued autoencoders. Neural Networks, 33, 136-147.

[32] Sohn, K., Berthelot, D., Carlini, et al. (2020). Fixmatch: Simplifying semi-supervised learning with consistency and confidence.

[33] Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of artificial intelligence research, 4, 237-285.

[34] Chen, X., Zhao, Z., & Zhang, H. (2011). Power Allocation for Cognitive Wireless Mesh Networks by Applying Multi-agent Q-learning Approach. arXiv preprint arXiv:1102.5400.

[35] Ong, H. Y., Chavez, K., & Hong, A. (2015). Distributed deep Q-learning. arXiv preprint arXiv:1508.04186.

[36] Helland, I. S. (1987). On the Interpretation and Use of R2 in Regression Analysis. Biometrics, 43(1), 61–69. https://doi.org/10.2307/2531949

[37] Innes, (2018). Flux: Elegant machine learning with Julia. Journal of Open Source Software, 3(25), 602, https://doi.org/10.21105/joss.00602

[38] Steele, G. (1990). Common LISP: the language. Elsevier.

[39] Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.

[40] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020)

[41] McKinney, W., & others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56).

[42] Dwyer, B., Nelson, J. (2022), Solawetz, J., et. al. Roboflow (Version 1.0) [Software].

[43] https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_the_United_Kingdom

[44] B. Gašparović, G. Mauša, J. Rukavina and J. Lerga, (2023) "Evaluating YOLOV5, YOLOV6, YOLOV7, and YOLOV8 in Underwater Environment: Is There Real Improvement?"

[45] Carratino, L., Cissé, M., Jenatton, R., & Vert, J. P. (2022). On mixup regularization. The Journal of Machine Learning Research, 23(1), 14632-14662.

[46] Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., & Yoo, Y. (2019). Cutmix: Regularization strategy to train strong classifiers with localizable features. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 6023-6032).

[47] https://docs.ultralytics.com/tasks/detect/